Auditing Black-Box LLM APIs with a Rank-Based Uniformity Test

Xiaoyuan Zhu[1] **Yaowen Ye**[2†] **Tianyi Qiu**[3†] **Hanlin Zhu**[2‡] **Sijun Tan**[2‡]
Ajraf Mannan[1] **Jonathan Michala**[1] **Raluca Ada Popa**[2] **Willie Neiswanger**[1]
[1]University of Southern California [2]University of California, Berkeley [3]Peking University
xzhu9839@usc.edu, elwin@berkeley.edu, qiutianyi.qty@gmail.com
{hanlinzhu, sijuntan}@berkeley.edu, {amannan, michala}@usc.edu
raluca@eecs.berkeley.edu, neiswang@usc.edu
† Co-second authors. ‡ Co-third authors.

Abstract

As API access becomes a primary interface to large language models (LLMs), users often interact with black-box systems that offer little transparency into the deployed model. To reduce costs or maliciously alter model behaviors, API providers may discreetly serve quantized or fine-tuned variants, which can degrade performance and compromise safety. Detecting such substitutions is difficult, as users lack access to model weights and, in most cases, even output logits. To tackle this problem, we propose a rank-based uniformity test that can verify the behavioral equality of a black-box LLM to a locally deployed authentic model. Our method is accurate, query-efficient, and avoids detectable query patterns, making it robust to adversarial providers that reroute or mix responses upon the detection of testing attempts. We evaluate the approach across diverse threat scenarios, including quantization, harmful fine-tuning, jailbreak prompts, and full model substitution, showing that it consistently achieves superior statistical power over prior methods under constrained query budgets.

1 Introduction

APIs have become a central access point for large language models (LLMs) in consumer applications, enterprise tools, and research workflows (Anysphere Inc., 2025; Yun et al., 2025; ResearchFlow, 2025). However, while users can query black-box APIs, they have little to no visibility into the underlying model implementation. Combined with the high cost of serving large models and the latency pressure to reduce time-to-first-token (TTFT), API providers are incentivized to deploy smaller or quantized variants of the original model to cut costs. Such modifications, while opaque to end users, can degrade model performance and introduce safety risks (Egashira et al., 2024). In more concerning cases, providers may incorporate harmful fine-tuning, jailbreak-enabling system prompts, or even misconfigured system components without realizing it (mirpo, 2025).

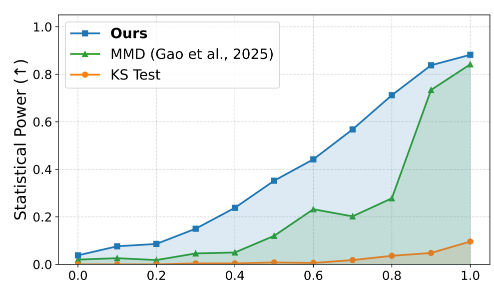


Figure 1: Statistical power of different methods in detecting substitution of the Gemma-2-9b-it with its 4-bit quantized variant, as the proportion of API responses from the quantized model increases. Our method significantly outperforms MMD (Gao et al., 2025) and the Kolmogorov–Smirnov (KS) baseline.

Preprint. Under review.

These risks highlight the need for *LLM API auditing*—the task of checking whether a deployed model is as claimed. Yet this is particularly challenging in the black-box setting: users typically lack access to model weights and receive only limited metadata (e.g., top-5 token log-probabilities). This necessitates detection methods that rely solely on observed outputs. However, even such output-level methods face potential evasion: if the detection relies on invoking an LLM API with specially constructed query distributions, a dishonest API provider could detect the special pattern and reroute those queries to the original model they claim to serve. Worse still, even without knowing the detection strategy, an API provider could mix multiple models, making the response distribution harder to distinguish.

Following Gao et al. (2025), we formulate the LLM API auditing problem as a *model equality test*: given query access to a target LLM API and a certified reference model of the expected configurations, the goal is to determine if the two produce statistically indistinguishable outputs on shared prompts.

We propose that a model equality test for auditing LLM APIs must satisfy three key criteria: *accuracy*, *query efficiency*, and *robustness* to adversarial attacks. Accuracy reflects how reliably a test can be used in practice. Query efficiency is critical for reducing operational overhead, which incentivizes more audits to ensure API models' integrity. Robustness is equally essential for real-world deployment, where audits must both evade detection by adversarial API providers and remain effective under targeted attacks.

While several methods have been proposed for model equality testing, they each fall short in one or more of these criteria (Table 1). Existing methods include Maximum Mean Discrepancy (MMD) (Gao et al., 2025), trained text classifiers (Sun et al., 2025), identity prompting (Huang et al., 2025), and benchmark performance comparison (Chen et al., 2023a). However, Sun et al. (2025) require prohibitively many API queries; Huang et al. (2025) fail to capture model variations such as size, version, or quantization (Cai et al., 2025); and Gao et al. (2025) and Chen et al. (2023a) rely on special query distributions that can be adversarially detected and circumvented by techniques such as prompt caching (Gu et al., 2025).

Table 1: Comparison of LLM auditing methods by *accuracy* (Acc.), *query-efficiency* (Q-Eff.), and *robustness to adversarial providers* (Rob.).

Method	Acc.	Q-Eff.	Rob.
RUT (Ours)	✓	✓	✓
MMD (Gao et al., 2025)	✓	✓	✗
Classifier (Sun et al., 2025)	✗	✗	✓
Identity-prompting (Huang et al., 2025)	✗	✓	✗
Benchmark (Chen et al., 2023a)	✗	✗	✗

Driven by these limitations, we propose a rank-based uniformity test (RUT)—an asymmetric two-sample hypothesis test that addresses all three criteria simultaneously. In RUT, we sample one response from the target API and multiple responses from the reference model for each prompt, then compute the rank percentile of the API output within the reference distribution. If the target and reference models are identical, the percentiles should follow a uniform distribution. We detect deviations using the Cramér–von Mises test (Cramér, 1928). Our method requires only a single API call per prompt, operates effectively on real-world, user-like queries, and avoids detectable patterns that adversarial providers might exploit.

We evaluate RUT across a range of adversarial scenarios in which the API provider secretly substitutes the claimed model with an alternative. In Section 5.2, we study the case where the substitute is a quantized version of the original model. In Section 5.3, we test models augmented with a hidden jailbreaking system prompt. In Section 5.4, we examine models finetuned on instruction-following data. Finally, in Section 5.5, we consider substitution with a completely different model.

Under a fixed API query budget, we find RUT outperforms both MMD and a Kolmogorov–Smirnov test (KS) baseline across all settings. It consistently achieves higher statistical power and shows greater robustness to probabilistic substitution attacks (Figure 1). Moreover, when applied to five real-world API-deployed models (Section 5.6), our method yields detection results closely aligned with other methods and shows more robustness over string-based metrics on minor decoding mismatches.

To summarize, the main contributions of our work include:

1. **A novel statistically-principled test for auditing LLM APIs.** We propose RUT, an asymmetric two-sample-test that needs only one API call per prompt and operates effectively on natural queries, achieving query efficiency and by-deign robustness to adversarial providers.

2. **Empirical validation across diverse threat models.** We perform comprehensive experiments to validate RUT's superior statistical power under diverse model substitution settings,

including quantization, jailbreaking, supervised finetuning (SFT), full model replacement, and hardware/provider replacement.

3. **Cross-validated audit of live commercial endpoints.** We benchmark RUT side-by-side with established tests (MMD and KS) on three major public LLM APIs and demonstrate its practicality in real-world black-box settings.

2 Related Work

LLM fingerprinting. Fingerprinting approaches focus on identifying LLMs by analyzing their outputs. Active fingerprinting involves injecting backdoor-like behavior (Xu et al., 2024) into an LLM via finetuning, embedding watermarks (Kirchenbauer et al., 2023; Ren et al., 2023) into a model' s text generation process, or intentionally crafting prompts to elicit unique outputs from different LLMs (Pasquini et al., 2024). Passive fingerprinting, on the other hand, focuses on analyzing the inherent patterns in LLM-generated text (Su et al., 2023; Fu et al., 2025; Alhazbi et al., 2025). This builds on the observation that LLMs expose rich "idiosyncrasies"—distributional quirks that allow classifiers to identify a model (Sun et al., 2025). While passive fingerprinting is relevant for LLM auditing, many such methods rely on training classifiers and require substantial labeled data, making them suboptimal for auditing LLM APIs. Prior work (Cai et al., 2025) also shows they are ineffective in detecting quantized model substitution.

Auditing LLM APIs. A growing body of work investigates whether black-box APIs faithfully serve the advertised model. The most straightforward audit is to evaluate models' benchmark performance (Analysis, 2025; Eyuboglu et al., 2024; Chen et al., 2023b), but raw performance alone cannot expose covert substitutions or partial routing. Gao et al. (2025) formalizes the problem as *Model Equality Testing* and shows that a kernel-MMD test can already flag public endpoints that deviate from their open-weight checkpoints. Concurrently to our work, Cai et al. (2025) investigate the model substitution setting and show that API providers can evade detection through strategies such as model quantization, randomized substitution, and benchmark evasion. Building on these insights, we propose a method that is more robust to such attacks and extend the threat model to include a broader range of realistic scenarios, such as jailbroken or maliciously finetuned models.

3 Problem Formulation

In this section, we formalize the LLM API auditing problem as a black-box model equality test between a target API and a fully accessible reference model.

Models. We denote an LLM as a conditional distribution $\pi(y|x;\varphi)$ over text output $y \in \mathcal{Y}$ given an input prompt $x \in \mathcal{X}$ and decoding parameters φ (e.g., temperature, top-p). In all experiments, we control the decoding parameters to be the same across models. Two such models are involved in the test. The *reference model* π_{ref} is fully accessible locally; the verifier can draw an unlimited number of samples $y_{\text{ref}} \sim \pi_{\text{ref}}(\cdot \mid x;\varphi)$ for any x and obtain relevant logits. The *target model* π_{tgt} is exposed only through a remote API that returns exactly one completion $y_{\text{tgt}} \sim \pi_{\text{tgt}}(\cdot \mid x;\varphi)$ per call and charges for every invocation. The total API budget is N prompts.

Hypotheses. The model equality test is cast as a composite two-sample hypothesis test. The null hypothesis

$$H_0: \quad \pi_{\text{tgt}}(\cdot \mid x;\varphi) \;=\; \pi_{\text{ref}}(\cdot \mid x;\varphi) \quad \forall\, x \in \mathcal{X} \tag{1}$$

states that the deployed model is behaviorally identical to the declared one.

$$H_1: \quad \exists\, x \in \mathcal{X} \text{ such that } \pi_{\text{tgt}}(\cdot \mid x;\varphi) \neq \pi_{\text{ref}}(\cdot \mid x;\varphi) \tag{2}$$

captures any observable deviation, whether arising from quantization, fine-tuning, prompt injection, or full model replacement.

Adversarial model provider. A dishonest model provider may attempt to evade detection through *probabilistic substitution*. We model this behaviour by

$$\pi_{\text{tgt}}(\cdot \mid x;\varphi) \;=\; \big(1-q(x)\big)\,\pi_{\text{ref}}(\cdot \mid x;\varphi) \;+\; q(x)\,\pi_{\text{alt}}(\cdot \mid x;\varphi), \tag{3}$$

where π_{alt} is an arbitrary alternative model (e.g., quantized or adversarially finetuned version of π_{ref}) and $q: \mathcal{X} \to [0,1]$ is an unknown routing function that may depend on the prompt and even on past interaction history (e.g., routing based on prompt caching (Gu et al., 2025)). When $q(x)$ is small but positive on a non-negligible fraction of typical prompts, the mixture in (3) significantly complicates discrimination; nevertheless, a sound detection procedure must retain statistical power under such adversarial conditions.

Detector and objectives. A *detector* is a decision rule $\delta : \mathcal{D}_N \to \{0,1\}$ whose output 1 means "reject H_0". Here, $\mathcal{D}_N$ denotes the data collected with N API queries and many local model queries. For a user-specified significance level $\alpha \in (0,1)$, the detector must satisfy the type-I error constraint $\Pr_{H_0}\big[\delta(\mathcal{D}_N) = 1\big] \le \alpha$. Subject to this guarantee, the verifier seeks to maximize the statistical power

$$\beta(q, \pi_{\text{alt}}) \;=\; \Pr_{H_1}\big[\delta(\mathcal{D}_N) = 1\big],$$

uniformly over a broad class of substitution policies (q, π_{alt}); to do so with as few costly API queries N as possible; and to employ prompts that are indistinguishable from ordinary user traffic, thereby preventing the provider from detecting and rerouting "suspicious-looking" inputs.

4 Method

In this section, we introduce our rank-based uniformity test in Section 4.1 and empirically identify the most discriminative score function to maximize the detection power in Section 4.2.

4.1 Rank-Based Uniformity Test (RUT)

To determine whether the target model π_{tgt} is identical to the reference model π_{ref}, we propose a *rank-based uniformity test*. This method assesses how typical the target model's responses are under the reference model's generation distribution.

Response score function. To perform a statistical test, we first define a scalar-valued *score function* $f : \mathcal{Y} \times \mathcal{X} \to \mathbb{R}$ that maps a model response and prompt to a real number. This function assigns a score to each output given the prompt, i.e.,

$$s = f(y, x), \quad \text{where } x \in \mathcal{X},\; y \in \mathcal{Y},\; s \in \mathbb{R}.$$

An ideal score function f^* should induce an *injective* mapping $y \mapsto f^*(y, x)$ for any fixed prompt $x \in \mathcal{X}$. Under this assumption, each distinct response corresponds to a unique score value, ensuring that the score distribution fully characterizes the model's outputs.

Uniformity as a test signal. For each prompt $x \in \mathcal{X}$, we sample a response $y_{\text{tgt}} \sim \pi_{\text{tgt}}(\cdot \mid x; \varphi)$ and compute its scalar score $s_{\text{tgt}} = f(y_{\text{tgt}}, x)$. To assess how typical this response is under the reference model, we evaluate its rank in the reference model's score distribution.

We define the cumulative distribution function (CDF) of the reference model's scores as:

$$F_{\pi_{\text{ref}}}(s \mid x) := \mathbb{P}_{y \sim \pi_{\text{ref}}(\cdot \mid x; \varphi)}\left[f(y, x) \le s\right].$$

Since $f(y, x)$ takes values in a discrete set, $F_{\pi_{\text{ref}}}$ is a step function. To ensure the rank statistic is continuously distributed under the null hypothesis, we apply a *randomized quantile residual* (Dunn and Smyth, 1996) to extend the probability integral transform (David and Johnson, 1948) to discrete distributions. Specifically, we define the *rank statistic* as

$$r_{\text{tgt}} := F_{\pi_{\text{ref}}}(s_{\text{tgt}}^-) + U \cdot \mathbb{P}\left(f(y, x) = s_{\text{tgt}}\right), \quad U \sim \text{Uniform}[0,1], \tag{4}$$

where $F_{\pi_{\text{ref}}}(s_{\text{tgt}}^-) := \mathbb{P}\left(f(y, x) < s_{\text{tgt}}\right)$ is the left-limit of the CDF at s_{tgt}, and $\mathbb{P}(f(y, x) = s_{\text{tgt}})$ is the probability mass at s_{tgt}. Under the null hypothesis $\pi_{\text{tgt}} = \pi_{\text{ref}}$, this rank statistic $r_{\text{tgt}} \in [0,1]$ is uniformly distributed.

Conversely, suppose that $r_{\text{tgt}} \sim \text{Uniform}[0,1]$ under the randomized quantile residual construction. Since the CDF $F_{\pi_{\text{ref}}}(\cdot \mid x)$ is stepwise and non-decreasing, a uniformly distributed r_{tgt} implies that the score s_{tgt} follows the same discrete distribution as s_{ref}. By injectivity of f, this further implies that $y_{\text{tgt}} \sim \pi_{\text{ref}}(\cdot \mid x; \varphi)$, and hence $\pi_{\text{tgt}} = \pi_{\text{ref}}$.

Thus, with an injective score function f, testing the uniformity of r_{tgt} as defined in (4) offers a valid signal for distinguishing π_{tgt} from π_{ref}.

Empirical approximation of $F_{\pi_{\text{ref}}}$. In practice, it is intractable to build the true CDF $F_{\pi_{\text{ref}}}(\cdot \mid x)$. Instead, we approximate it using an empirical CDF from m reference samples for each prompt.

Given a target response $y_i \sim \pi_{\text{tgt}}(\cdot \mid x_i; \theta)$ and reference responses $y_{ij} \sim \pi_{\text{ref}}(\cdot \mid x_i; \theta)$ for $j = 1, \ldots, m$, we compute the scalar scores

$$s_i := f(y_i, x_i), \quad s_{ij} := f(y_{ij}, x_i).$$

We then define the *randomized rank statistics* $r_i \in [0, 1]$ as

$$r_i = \frac{1}{m} \left(\sum_{j=1}^{m} \mathbf{1}\{s_i > s_{ij}\} + U_i \cdot \sum_{j=1}^{m} \mathbf{1}\{s_i = s_{ij}\} \right),$$

where $U_i \sim \text{Uniform}[0, 1]$ is an independent random variable to break ties uniformly, and ensure r_i is an unbiased estimator of r_{tgt} given the prompt x_i.

Discriminative score function via empirical selection. While an ideal *injective* score function would guarantee sensitivity to any behavioral difference between π_{tgt} and π_{ref}, constructing such a function for which we can calculate the CDF is generally infeasible in practice.

To ensure that our test remains practically effective, we instead require the score function to be *sufficiently discriminative*, in the sense that it induces distinct score distributions whenever $\pi_{\text{ref}} \neq \pi_{\text{tgt}}$. Formally, for fixed prompt $x \in \mathcal{X}$, let

$$S_{\pi_{\text{ref}}} := f(y, x) \text{ with } y \sim \pi_{\text{ref}}(\cdot \mid x; \varphi), \quad \text{and} \quad S_{\pi_{\text{tgt}}} := f(y, x) \text{ with } y \sim \pi_{\text{tgt}}(\cdot \mid x; \varphi).$$

We say that f is sufficiently discriminative if the distributions of $S_{\pi_{\text{ref}}}$ and $S_{\pi_{\text{tgt}}}$ differ whenever $\pi_{\text{ref}} \neq \pi_{\text{tgt}}$, i.e.,

$$\pi_{\text{ref}}(\cdot \mid x; \varphi) \neq \pi_{\text{tgt}}(\cdot \mid x; \varphi) \quad \Rightarrow \quad P_{S_{\pi_{\text{ref}}}} \neq P_{S_{\pi_{\text{tgt}}}}.$$

Under this condition, differences in response distributions are reflected in the score distributions, causing the ranks to deviate from uniformity.

Thus, we aim to find the most discriminative score function among several promising candidates through empirical experiments. In Section 4.2, we compare five candidate score functions—log-likelihood, token rank, log-rank, entropy, and the log-likelihood log-rank ratio (Su et al., 2023)—and find that log-rank is the most discriminative in practice for separating responses by π_{ref} and π_{tgt}, and therefore adopt it in our uniformity test.

Full test procedure. We now present the full RUT procedure.

Let $\{x_1, \ldots, x_n\} \subset \mathcal{X}$ be a set of prompts. For each prompt x_i, we sample one response from the target model,

$$y_i \sim \pi_{\text{tgt}}(\cdot \mid x_i; \theta),$$

and m responses from the reference model,

$$y_{ij} \sim \pi_{\text{ref}}(\cdot \mid x_i; \theta), \quad j = 1, \ldots, m.$$

We compute the log-rank scores

$$s_i := f(y_i, x_i), \quad s_{ij} := f(y_{ij}, x_i),$$

and the corresponding randomized rank statistics $\{r_i\}_{i=1}^n$.

We apply the Cramér–von Mises (CvM) test (Cramér, 1928) to assess the deviations between $\{r_i\}_{i=1}^n$ and $\text{Uniform}[0, 1]$. The test evaluates the null hypothesis

$$H_0 : r_i \sim \text{Uniform}[0, 1] \quad \text{for all } i.$$

The CvM test statistic is defined as

$$\omega^2 = \frac{1}{12n} + \sum_{i=1}^{n} \left(\frac{2i - 1}{2n} - r_{(i)} \right)^2,$$

where $r_{(1)} \leq r_{(2)} \leq \cdots \leq r_{(n)}$ are the ordered rank statistics.

To compute the p-value, we compare the observed statistic ω^2_{obs} to the distribution of the CvM statistic ω^2_{null} computed under the null hypothesis. The p-value is given by

$$p\text{-value} = \mathbb{P}_{H_0} \left[\omega^2_{\text{null}} \geq \omega^2_{\text{obs}} \right].$$

We reject H_0 and conclude that the target and reference models are different if $p\text{-value} < 0.05$.

4.2 Score Function Selection

The RUT requires a scalar score function $f(y, x)$. To identify a function that best captures distributional differences between models, we consider five candidate functions:

- **Log-likelihood:** $\log \pi_{\text{ref}}(y \mid x)$.
- **Token rank:** the average rank of response tokens in y, where a token's rank is its position in the vocabulary ordered by the π_{ref}'s next-token probabilities.
- **Log-rank:** the average of the logarithm of the token rank.
- **Entropy:** predictive entropy for y under $\pi_{\text{ref}}(x)$.
- **Log-likelihood log-rank ratio (LRR):** the ratio between log-likelihood and log-rank. (Su et al., 2023).

To identify the most discriminative score function, we conduct a Monte Carlo evaluation consisting of 500 independent trials. In each trial, we randomly select 10 prompts from the WildChat (Zhao et al., 2024) dataset and sample 50 completions per prompt from both π_{ref} and π_{tgt}, using a fixed temperature of 0.5 and a maximum length of 30 tokens. For each candidate score function, we compute the average AUROC (Bradley, 1997) across the 10 prompts for each trial, yielding a distribution of 500 AUROC scores per function. The full algorithm to calculate per score function average AUROC is included in Appendix A.1. Across different model comparisons, we find that **log-rank** consistently yields the most separable AUROC distribution from 0.5, indicating the strongest discriminative power. Figure 2 shows an example comparing Gemma-2-9b-it with its 4-bit quantized variant. Based on these results, we select log-rank as the scoring function for our uniformity test. Complete AUROC results are provided in Appendix A.2.

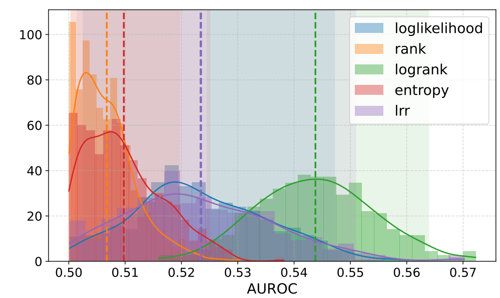


Figure 2: Distribution of AUROC scores for five candidate score functions across 500 trials comparing Gemma-2-9b-it and its 4-bit quantized variant. Log-rank achieves the most separable distribution from the random level 0.5, indicating superior power in distinguishing different models.

5 Experiments

In this section, we evaluate RUT across diverse model substitution scenarios, including quantization (Section 5.2), jailbreaks (Section 5.3), SFT (Section 5.4), full model replacement (Section 5.5), and real-world API providers (Section 5.6). Detection performance is compared against MMD and a KS baseline using statistical power AUC as the primary metric.

5.1 Experimental Setup

To evaluate detection performance under adversarial conditions, we simulate probabilistic substitution attacks where a fraction $q \in [0, 1]$ of API queries are routed to an alternative model (e.g., quantized or fine-tuned). For each value of q, we estimate the statistical power, defined as the probability of correctly rejecting the null hypothesis when substitution is present. We then summarize the resulting power–substitution rate curve using the area under the curve (AUC) over $q \in [0, 1]$. The AUC ranges from 0 to 1 and reflects the method's ability to maintain high statistical power across varying levels of substitution, serving as a measure of robustness to such attacks. Higher values indicate more reliable and consistent detection performance. Figure 1 shows an example comparing Gemma-2-9b-it and its 4-bit quantized variant.

Data. We use the WildChat dataset (Zhao et al., 2024), which contains real-world conversations between human users and ChatGPT. This dataset reflects authentic user behavior, ensuring the query distribution remains indistinguishable from typical API traffic—crucial for evasion-resistant auditing.

(a) Statistical power AUC for detecting quantized variants. **Bold** = best method; gray = none reliable.

Model	RUT	MMD	KS
Gemma–4bit	**0.392**	0.214	0.017
Gemma–8bit	0.049	0.043	0.001
Llama–4bit	**0.642**	0.625	0.474
Llama–8bit	0.132	**0.158**	0.005
Mistral–4bit	**0.586**	0.500	0.330
Mistral–8bit	0.049	0.090	0.006

(b) Statistical power AUC for detecting jail-breaking prompts. **Bold** = most effective method per prompt.

Model	Prompt	RUT	MMD	KS
Mistral	Dan	**0.895**	0.802	0.873
	Anti-Dan	**0.893**	0.781	0.872
	Evil-Bot	**0.892**	0.766	0.873
Gemma	Dan	**0.888**	0.757	0.867
	Anti-Dan	**0.858**	0.816	0.854
	Evil-Bot	**0.893**	0.753	0.871

Table 2: Side-by-side comparison of statistical-power AUCs. Left: quantized variants; right: jail-breaking prompts.

Baseline. For the detection methods (Sun et al., 2025; Gao et al., 2025) that are compatible with WildChat, We primarily focus on Maximum Mean Discrepancy (MMD) (Gao et al., 2025) as the baseline, as Sun et al. (2025) is reported to fail to identify quantization (Cai et al., 2025).

We also tailor a Kolmogorov–Smirnov (KS) test baseline that uses the same information as RUT: it computes the log-rank scores from the reference model on both the target and reference model responses and applies the two-sample KS test (Darling, 1957) on these two sets of scores to estimate the p-value between the target and reference distributions.

Test procedures. We apply a consistent sample budget constraint on all tests. The implementation details of their test procedures are listed below:

- **Rank-Based Uniformity Test (RUT):** Each trial samples 100 prompts. We query each prompt once to the target and 100 times to the reference model.
- **Maximum Mean Discrepancy (MMD):** We apply the MMD test based on the character-level Hamming distance following Gao et al. (2025). Each trial uses 10 prompts, with 10 samples per prompt. We compute the MMD statistic and estimate the p-value via 500 random permutations.
- **Kolmogorov–Smirnov Test (KS):** We use the same sampling setup as RUT: 100 prompts per trial, 1 query to the target, and 100 to the reference model per prompt.

Across all models, we set the temperature to 0.5 and cap generation at 30 tokens. We use vLLM (Kwon et al., 2023) on a single A6000 for all local inferences.

To estimate statistical power, we repeat each experiment over 500 Monte Carlo trials. Power is computed as the proportion of trials in which the test correctly rejects the null hypothesis at a significant level of 0.05. All statistical tests were run with Intel Xeon Gold 6230R @ 2.10GHz and 16 GB RAM per job.

5.2 Detecting Quantization

We consider the setting where the API provider uses a quantized variant to substitute the claimed model. We evaluate three detection methods on quantized variants of Llama-3.2-3B-Instruct[1], Mistral-7B-Instruct-v0.3[2], and Gemma-2-9B-it[3], comparing each model to its 4-bit and 8-bit quantized counterparts. As shown in Table 2a, none of the methods succeed in reliably detecting substitution for the 8-bit variants of Gemma and Mistral, where statistical power AUC remains near zero across the board. In the remaining four settings, RUT outperforms MMD and the KS baseline in 3 out of the 4 cases, demonstrating superior sensitivity to quantization-induced distributional shifts. Full statistical power curves for AUCs are provided in Appendix B.1.

5.3 Detecting Jailbreaks

We consider the setting where the API provider secretly appends a hidden jailbreaking system prompt to user queries. To evaluate this scenario, we use two base models: Mistral-7B-Instruct-v0.3 and

[1] `https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct`
[2] `https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3`
[3] `https://huggingface.co/google/gemma-2-9b-it`

Gemma-2-9B-it. For each model, we construct a test using three representative jailbreaking prompts *Dan*, *Anti-Dan*, and *Evil-Bot* adapted from Shen et al. (2024).

As shown in Table 2b, all six jailbreak cases are reliably detected, with statistical power AUC values consistently above 0.75. RUT achieves the highest power in all 6 settings, demonstrating its superior sensitivity to model deviations caused by hidden jailbreaking prompts. Full statistical power curves for AUCs are provided in Appendix B.2.

5.4 Detecting SFT

We study the setting where the API provider fine-tunes a model on instruction-following data. Specifically, we fine-tune two base models—Llama-3.2-3B-Instruct and Mistral-7B-Instruct-v0.3—on benign and harmful instruction-following datasets. We use Alpaca (Taori et al., 2023) as the benign dataset and BeaverTails (Ji et al., 2023) for harmful question answering. Each model is fine-tuned on 500 samples from the respective dataset for 5 epochs using LoRA (Hu et al., 2021) with rank 64 and $\alpha = 16$, a batch size of 32, and a learning rate of 1×10^{-4} on a single A100. For each checkpoint, we compute the statistical power AUC of the detection methods.

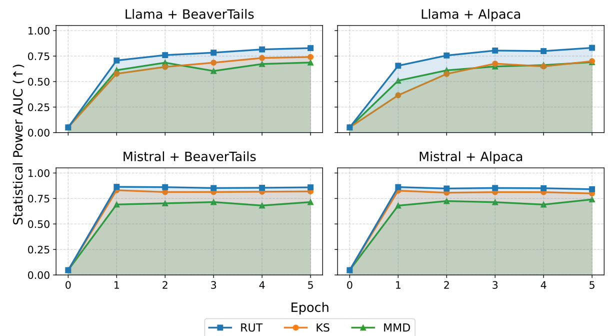


Figure 3: Statistical power AUC across epochs for detecting SFT model substitution.

As shown in Figure 3, RUT consistently achieves higher statistical power AUC than both the KS and MMD baselines across all fine-tuning configurations. Notably, our method detects behavioral changes within the first epoch of fine-tuning, demonstrating strong sensitivity to early-stage distributional shifts. While all methods improve with additional training, RUT remains the most robust across both models and datasets. Full statistical power curves for AUCs are provided in Appendix B.3.

5.5 Detecting Full Model Replacement

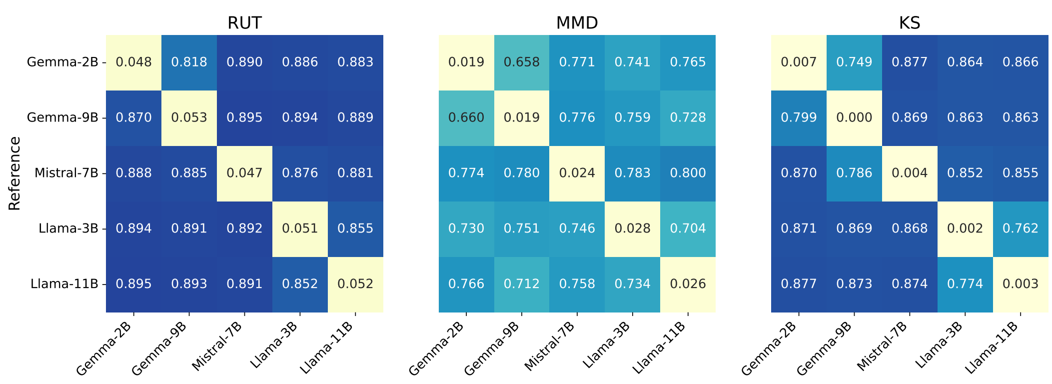


Figure 4: Statistical power AUC for detecting full model replacement. Each cell shows the AUC score between a reference and a target model. Diagonal values represent self-comparisons.

We evaluate the setting where the API provider substitutes the claimed model with a completely different one. To simulate this scenario, we conduct pairwise comparisons among five open-source models: Llama-3.2-3B-Instruct, Llama-3.2-11B-Vision-Instruct[4], Mistral-7B-Instruct-v0.3, Gemma-

[4] https://huggingface.co/meta-llama/Llama-3.2-11B-Vision-Instruct

2-2B-it[5], and Gemma-2-9B-it. For each pair, one model serves as the reference model while the other acts as the deployed target model. As shown in Figure 4, RUT consistently achieves the highest statistical power AUC across model pairs, outperforming both the MMD and KS baselines. The results highlight the method's sensitivity to full model substitutions. Full statistical power curves for AUCs are provided in Appendix B.4.

5.6 Detecting Real API Providers

Table 3: Statistical power for detecting differences from the baseline model deployed on an A6000 GPU. `A100` denotes the same model run locally on an A100 GPU; other entries are actual API providers. Values > 0.5 indicate significant behavioral deviation. Green = no significant difference; Red = significant difference.

Model	Provider	RUT	MMD	KS
Llama	A100	0.094	0.142	0.002
Llama	Nebius	0.962	0.944	0.426
Llama	Novita	0.988	0.996	0.530
Mistral	A100	0.058	0.138	0.004
Mistral	HF Inference	0.188	1.000	0.000
Gemma	A100	0.060	0.084	0.000
Gemma	Nebius	0.312	0.432	0.008

We evaluate our detection methods on three base models—Llama-3.2-3B-Instruct, Mistral-7B-Instruct-v0.3, and Gemma-2-9B-it—each deployed through multiple API providers. Local inference on an A100 GPU serves as the baseline. As shown in Table 3, all tests correctly identify behavioral equivalence in local deployments.

Across all settings, RUT and MMD generally agree in detecting significant deviations across providers, offering mutual validation for their behavioral sensitivity. The KS test exhibits similar trends but with notably lower sensitivity.

An exception arises in the Mistral + HF Inference setting, where MMD yields a power of 1.000 while other tests are below 0.2. Upon investigation, we suspect that the discrepancy is due to a tokenization mismatch: the Hugging Face Inference API consistently omits the leading whitespace present in the reference outputs. Because MMD uses character-level Hamming distance, this formatting difference inflates the score. After restoring the missing space, the MMD score drops to 0.211, aligning with other tests. This illustrates RUTt's robustness to minor decoding mismatches that can mislead string-based metrics.

6 Conclusion

The stable increase in the size (Kaplan et al., 2020) and architectural complexity (Zhou et al., 2022) of frontier LLMs has led to a rise in the popularity of API-based model access. To prevent performance degradation and security risks from model substitution behind API interfaces, this work proposes the rank-based uniformity test for model equality testing. We test the method against a variety of different substitution attacks and demonstrate its consistent effectiveness in detecting substitution and its superiority over existing methods.

Limitations and Future Work Using prompts from the WildChat dataset (Zhao et al., 2024) for testing, we aim to avoid detection of our testing attempts. However, we have not empirically validated its effectiveness at evading detection, especially against detection methods based on the correlation across prompts. In future work, examining the method's detectability and improving it with adaptive methods for prompt selection shall be a priority. Also, our test requires a locally deployed authentic model, limiting its capability to test black-box models.

By developing an effective and stealthy API-based test for model equality, we hope to advance the safety and security of LLM-based applications in the age of increasingly cloud-based deployment.

[5] `https://huggingface.co/google/gemma-2-2b-it`

References

Saeif Alhazbi, Ahmed Mohamed Hussain, Gabriele Oligeri, and Panos Papadimitratos. Llms have rhythm: Fingerprinting large language models using inter-token times and network traffic analysis. *ArXiv*, abs/2502.20589, 2025. URL `https://api.semanticscholar.org/CorpusID:276725236`. 3

Artificial Analysis. Artificial analysis, 2025. URL `https://artificialanalysis.ai`. 3

Anysphere Inc. Cursor: The ai code editor, 2025. URL `https://www.cursor.com/`. Accessed: 2025-05-15. 1

Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997. ISSN 0031-3203. doi: https://doi.org/10.1016/S0031-3203(96)00142-2. URL `https://www.sciencedirect.com/science/article/pii/S0031320396001422`. 6,

Will Cai, Tianneng Shi, Xuandong Zhao, and Dawn Song. Are you getting what you pay for? auditing model substitution in llm apis, 2025. URL `https://arxiv.org/abs/2504.04715`. 2, 3, 7

Lingjiao Chen, Matei Zaharia, and James Zou. How is chatgpt's behavior changing over time?, 2023a. URL `https://arxiv.org/abs/2307.09009`. 2

Lingjiao Chen, Matei Zaharia, and James Y. Zou. How is chatgpt's behavior changing over time? *ArXiv*, abs/2307.09009, 2023b. URL `https://api.semanticscholar.org/CorpusID:259951081`. 3

Harald Cramér. On the composition of elementary errors. *Scandinavian Actuarial Journal*, 1928(1):13–74, 1928. doi: 10.1080/03461238.1928.10416862. URL `https://doi.org/10.1080/03461238.1928.10416862`. 2, 5

D. A. Darling. The kolmogorov-smirnov, cramér-von mises tests. *The Annals of Mathematical Statistics*, 28(4): 823–838, 1957. ISSN 00034851, 21688990. URL `http://www.jstor.org/stable/2237048`. 7

F. N. David and N. L. Johnson. The probability integral transformation when parameters are estimated from the sample. *Biometrika*, 35(1/2):182–190, 1948. ISSN 00063444. URL `http://www.jstor.org/stable/2332638`. 4

Peter K. Dunn and Gordon K. Smyth. Randomized quantile residuals. *Journal of Computational and Graphical Statistics*, 5(3):236–244, 1996. ISSN 10618600. URL `http://www.jstor.org/stable/1390802`. 4

Kazuki Egashira, Mark Vero, Robin Staab, Jingxuan He, and Martin Vechev. Exploiting llm quantization, 2024. URL `https://arxiv.org/abs/2405.18137`. 1

Sabri Eyuboglu, Karan Goel, Arjun Desai, Lingjiao Chen, Mathew Monfort, Chris Ré, and James Zou. Model changelists: Characterizing updates to ml models. *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*, 2024. URL `https://api.semanticscholar.org/CorpusID:270287160`. 3

Zhiyuan Fu, Junfan Chen, Hongyu Sun, Ting Yang, Ruidong Li, and Yuqing Zhang. Fdllm: A text fingerprint detection method for llms in multi-language, multi-domain black-box environments. *ArXiv*, abs/2501.16029, 2025. URL `https://api.semanticscholar.org/CorpusID:275921293`. 3

Irena Gao, Percy Liang, and Carlos Guestrin. Model equality testing: Which model is this api serving?, 2025. URL `https://arxiv.org/abs/2410.20247`. 1, 2, 3, 7

Chenchen Gu, Xiang Lisa Li, Rohith Kuditipudi, Percy Liang, and Tatsunori Hashimoto. Auditing prompt caching in language model apis, 2025. URL `https://arxiv.org/abs/2502.07776`. 2, 4

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL `https://arxiv.org/abs/2106.09685`. 8

Yangsibo Huang, Milad Nasr, Anastasios Angelopoulos, Nicholas Carlini, Wei-Lin Chiang, Christopher A. Choquette-Choo, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Ken Ziyu Liu, Ion Stoica, Florian Tramer, and Chiyuan Zhang. Exploring and mitigating adversarial manipulation of voting-based leaderboards, 2025. URL `https://arxiv.org/abs/2501.07493`. 2

Jiaming Ji, Mickel Liu, Juntao Dai, Xuehai Pan, Chi Zhang, Ce Bian, Chi Zhang, Ruiyang Sun, Yizhou Wang, and Yaodong Yang. Beavertails: Towards improved safety alignment of llm via a human-preference dataset. *arXiv preprint arXiv:2307.04657*, 2023. 8

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. 9

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. In *International Conference on Machine Learning*, 2023. URL https://api.semanticscholar.org/CorpusID:256194179. 3

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023. 7

mirpo. [bug] lm studio's api server ignores the requested model name in the api call when only one model is running · issue #619 · lmstudio-ai/lmstudio-bug-tracker, 2025. URL https://github.com/lmstudio-ai/lmstudio-bug-tracker/issues/619. 1

Dario Pasquini, Evgenios M. Kornaropoulos, and Giuseppe Ateniese. Llmmap: Fingerprinting for large language models. *ArXiv*, abs/2407.15847, 2024. URL https://api.semanticscholar.org/CorpusID:271328475. 3

Jie Ren, Han Xu, Yiding Liu, Yingqian Cui, Shuaiqiang Wang, Dawei Yin, and Jiliang Tang. A robust semantics-based watermark for large language model against paraphrasing. In *NAACL-HLT*, 2023. URL https://api.semanticscholar.org/CorpusID:265213008. 3

ResearchFlow. Researchflow: Ai-powered research engine & visual knowledge mapping, 2025. URL https://rflow.ai/. 1

Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. "do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models, 2024. URL https://arxiv.org/abs/2308.03825. 8

Jinyan Su, Terry Yue Zhuo, Di Wang, and Preslav Nakov. Detectllm: Leveraging log rank information for zero-shot detection of machine-generated text, 2023. URL https://arxiv.org/abs/2306.05540. 3, 5, 6

Mingjie Sun, Yida Yin, Zhiqiu Xu, J. Zico Kolter, and Zhuang Liu. Idiosyncrasies in large language models, 2025. URL https://arxiv.org/abs/2502.12150. 2, 3, 7

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023. 8

Jiashu Xu, Fei Wang, Mingyu Derek Ma, Pang Wei Koh, Chaowei Xiao, and Muhao Chen. Instructional fingerprinting of large language models. *ArXiv*, abs/2401.12255, 2024. URL https://api.semanticscholar.org/CorpusID:267095230. 3

Yuhui Yun, Huilong Ye, Xinru Li, Ruojia Li, Jingfeng Deng, Li Li, and Haoyi Xiong. Eicopilot: Search and explore enterprise information over large-scale knowledge graphs with llm-driven agents, 2025. URL https://arxiv.org/abs/2501.13746. 1

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. Wildchat: 1m chatgpt interaction logs in the wild, 2024. URL https://arxiv.org/abs/2405.01470. 6, 9

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022. 9

A AUROC

A.1 AUROC Algorithm

Algorithm 1: Average AUROC for score function evaluation

Input: Prompt set $\mathcal{D}$; models π_{ref}, π_{tgt}; decoding parameters $\varphi = (\tau, L)$, where τ is temperature and L is the maximum generation length; number of prompts n; completions per prompt per model m; score functions $\{\delta_1, \ldots, \delta_K\}$.

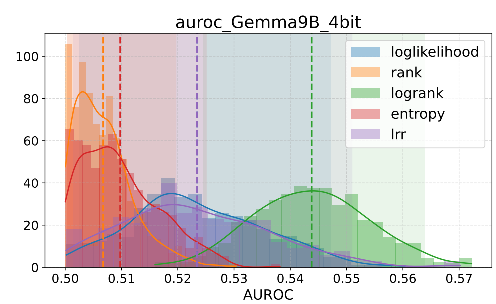
Output: Mean AUROC per score function, denoted $\mu_{\text{AUROC}}(\delta)$.

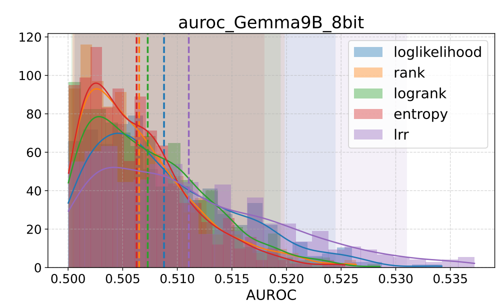
Draw $\{x_1, \ldots, x_n\} \sim \text{Uniform}(\mathcal{D})$;
for $i \in \{1, \ldots, n\}$ **do**
  $\{y_{\text{ref}}^{(j)}\}_{j=1}^{m} \sim \pi_{\text{ref}}(\cdot \mid x_i; \varphi)$;
  $\{y_{\text{tgt}}^{(j)}\}_{j=1}^{m} \sim \pi_{\text{tgt}}(\cdot \mid x_i; \varphi)$;
  $\mathcal{Y}_i \leftarrow \{y_{\text{ref}}^{(j)}\} \cup \{y_{\text{tgt}}^{(j)}\}$;
  $L_i \leftarrow \{0\}^m \cup \{1\}^m$;
  for $\delta \in \{\delta_1, \ldots, \delta_K\}$ **do**
    $S_i \leftarrow \{\delta(y) \mid y \in \mathcal{Y}_i\}$;
    Store $A_i^{\delta} \leftarrow \text{AUROC}(S_i, L_i)$;
for $\delta \in \{\delta_1, \ldots, \delta_K\}$ **do**
  $\mu_{\text{AUROC}}(\delta) \leftarrow \frac{1}{n} \sum_{i=1}^{n} A_i^{\delta}$;

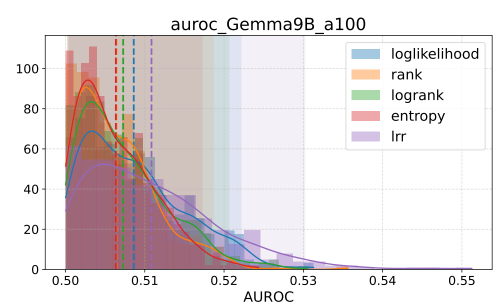
Note. $\text{AUROC}(S, L)$ denotes the standard binary AUROC (Bradley, 1997).

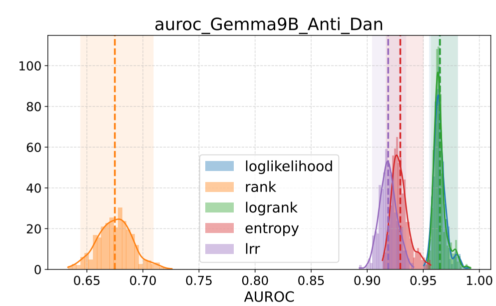
A.2 AUROC Score Distributions

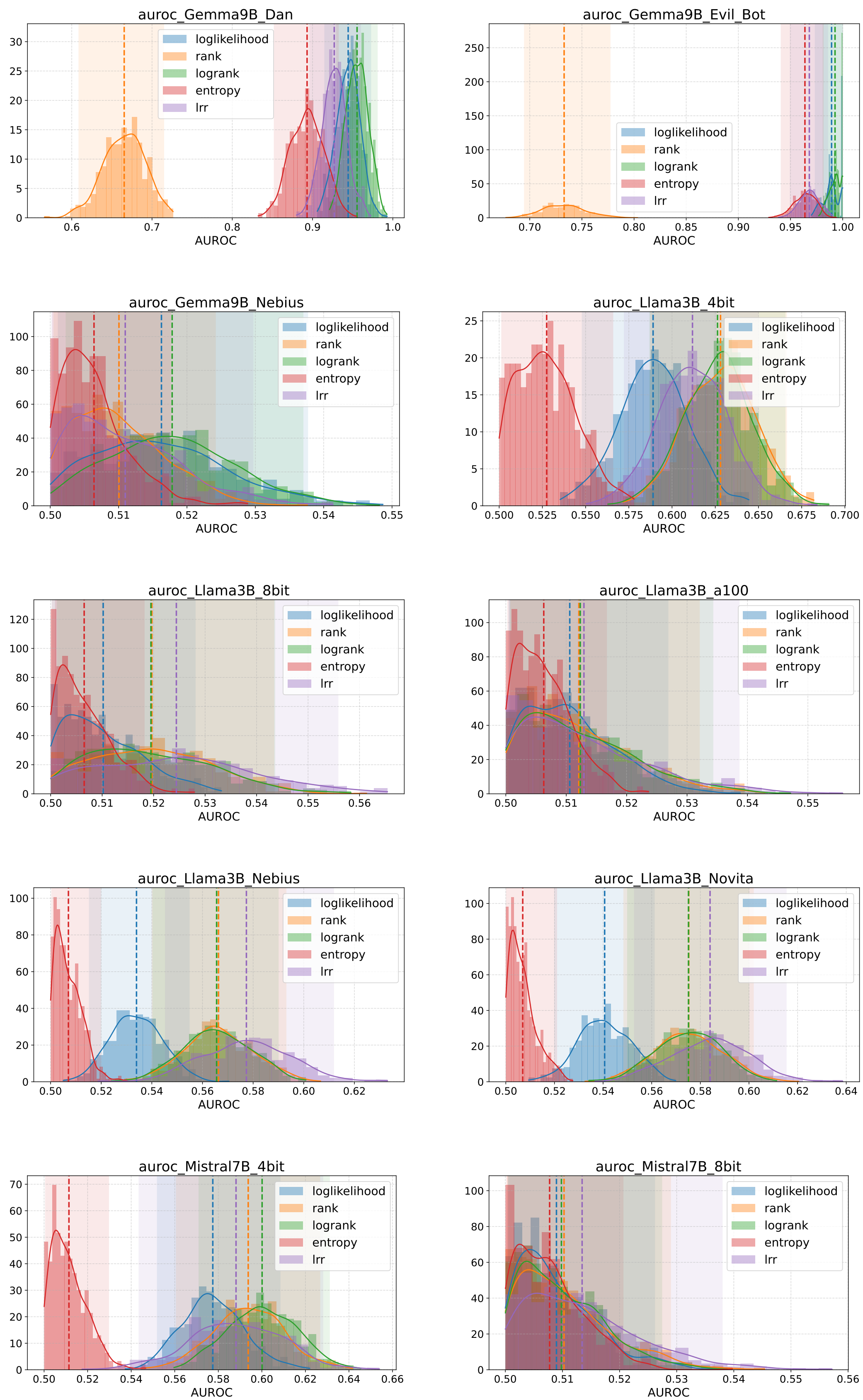
We present the AUROC score distributions from the score function selection experiment described in Section 4.2. Specifically, we evaluated Gemma-2-9B-it, LLaMA-3.2-3B-Instruct, and Mistral-7B-Instruct, and visualized the distributions when distinguishing the original model outputs from three types of variants: (1) quantized versions, (2) models subjected to jailbreaking prompts, and (3) models served by A100 or external API providers.

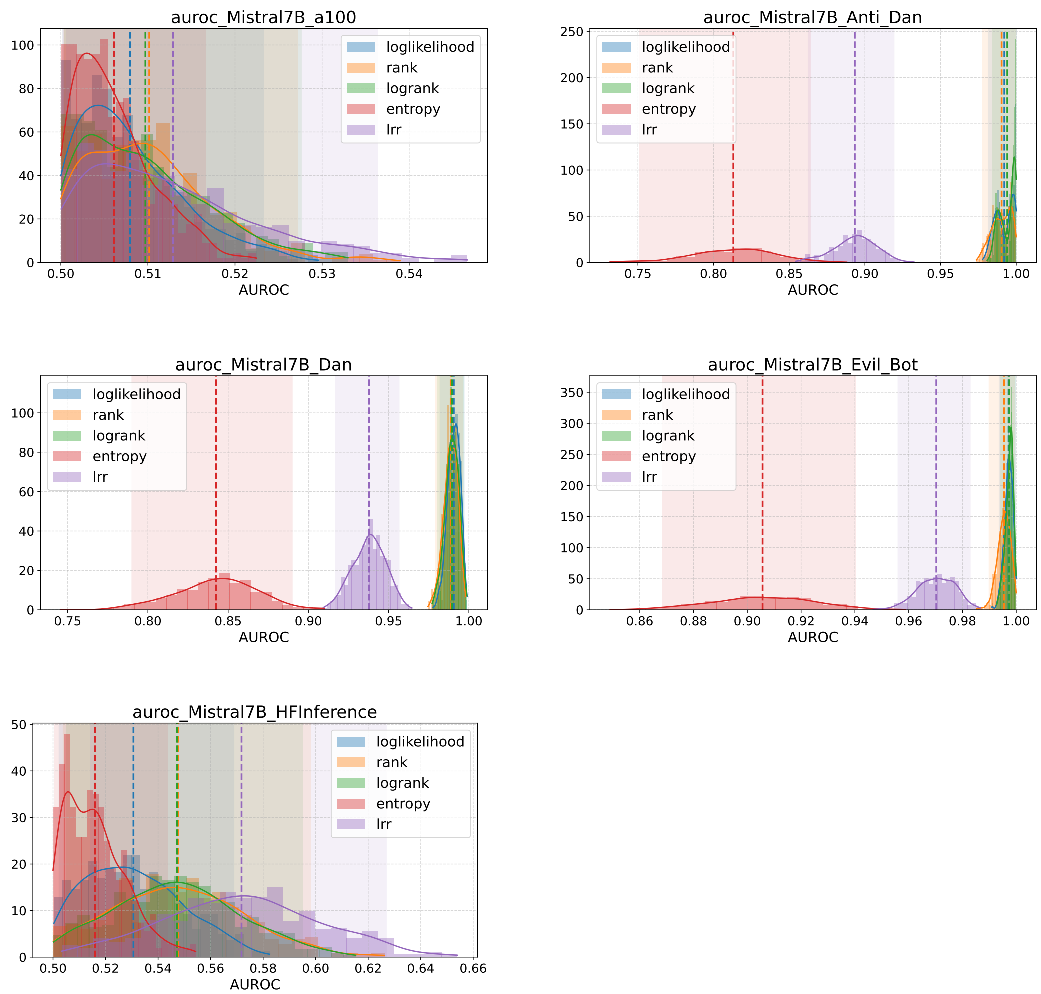








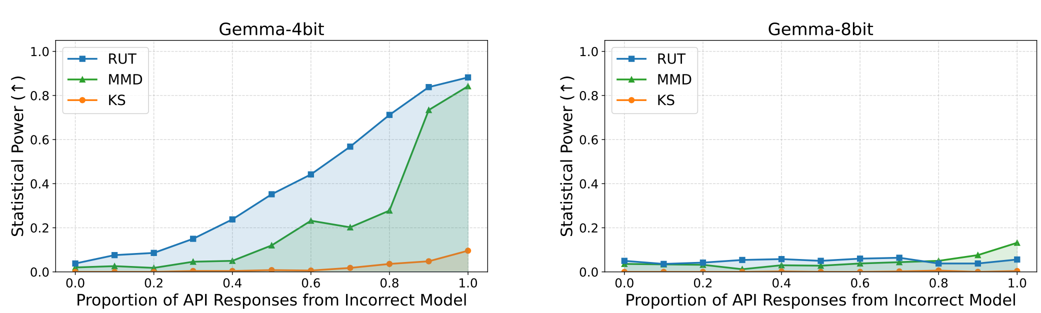

auroc_Gemma9B_Dan
auroc_Gemma9B_Evil_Bot
auroc_Gemma9B_Nebius
auroc_Llama3B_4bit
auroc_Llama3B_8bit
auroc_Llama3B_a100
auroc_Llama3B_Nebius
auroc_Llama3B_Novita
auroc_Mistral7B_4bit
auroc_Mistral7B_8bit
loglikelihood
rank
logrank
entropy
lrr
AUROC

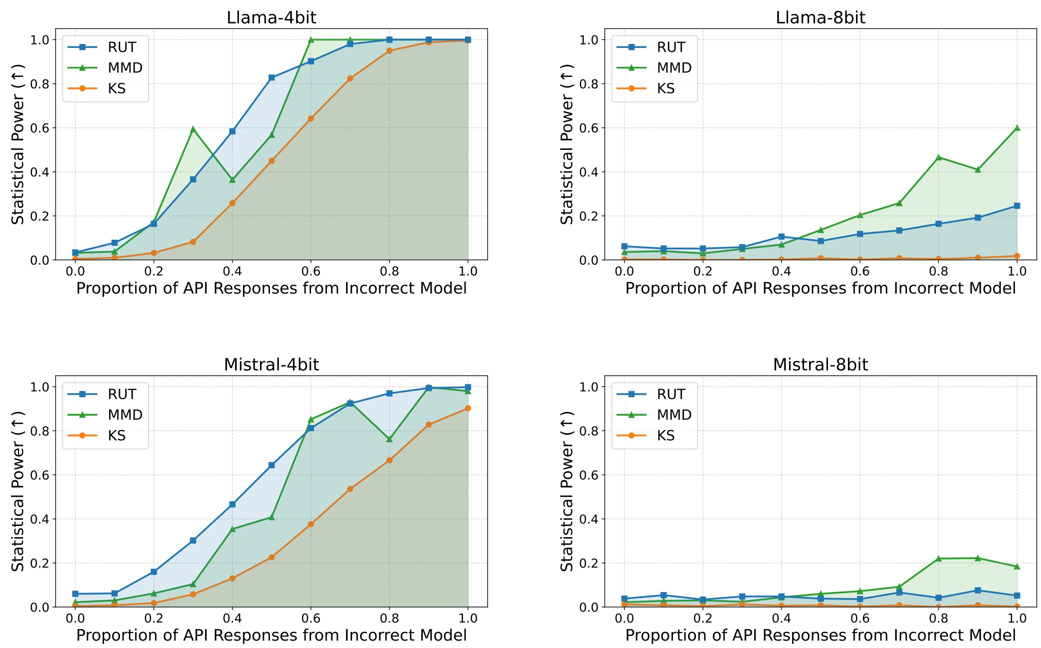


B Statistic Power Curves

B.1 Full Statistic Power Curves for Detecting Quantization

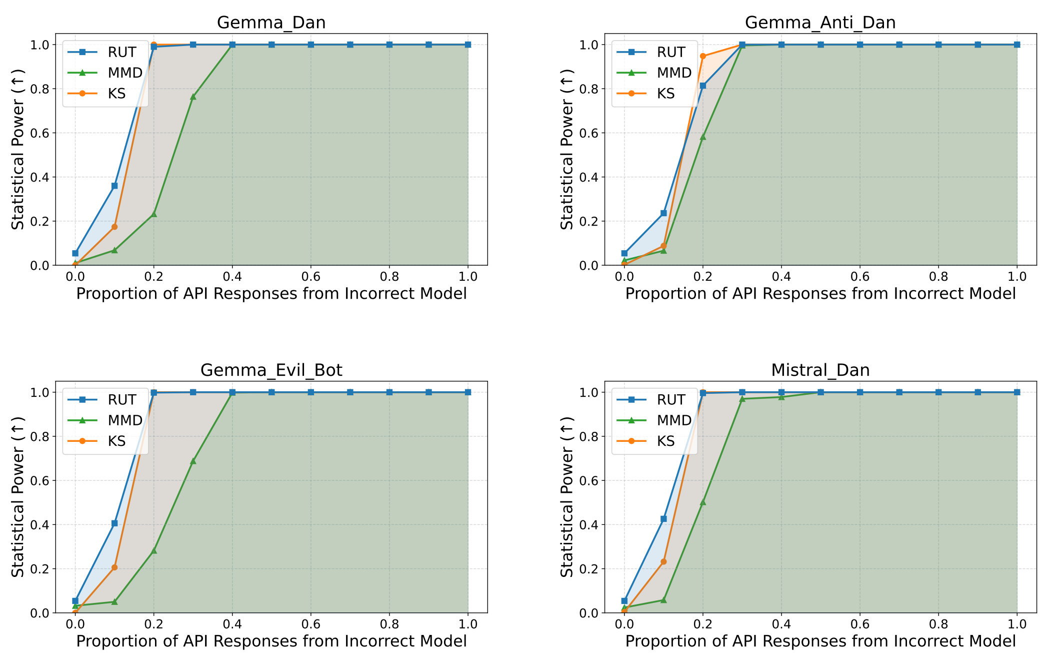
We present the full statistical power curves, showing the relationship between substitution rate and detection power, corresponding to the experiments on detecting quantized model substitutions described in Section 5.2. These curves are used to compute the power AUC values reported in the main paper and illustrate each method's detection power across different levels of substitution.

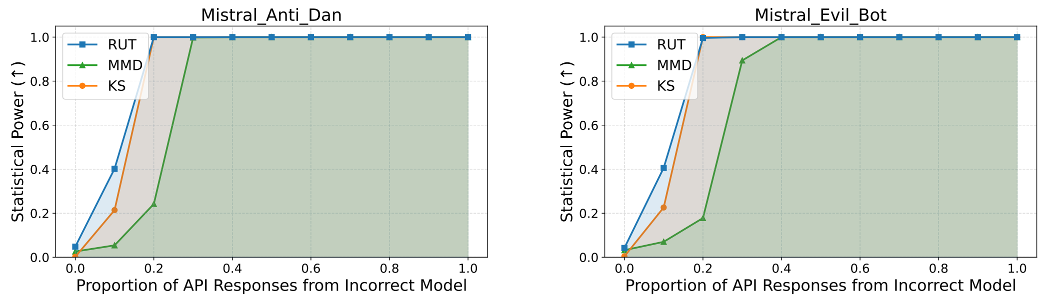




B.2 Full Statistic Power Curves for Detecting Jailbreaking

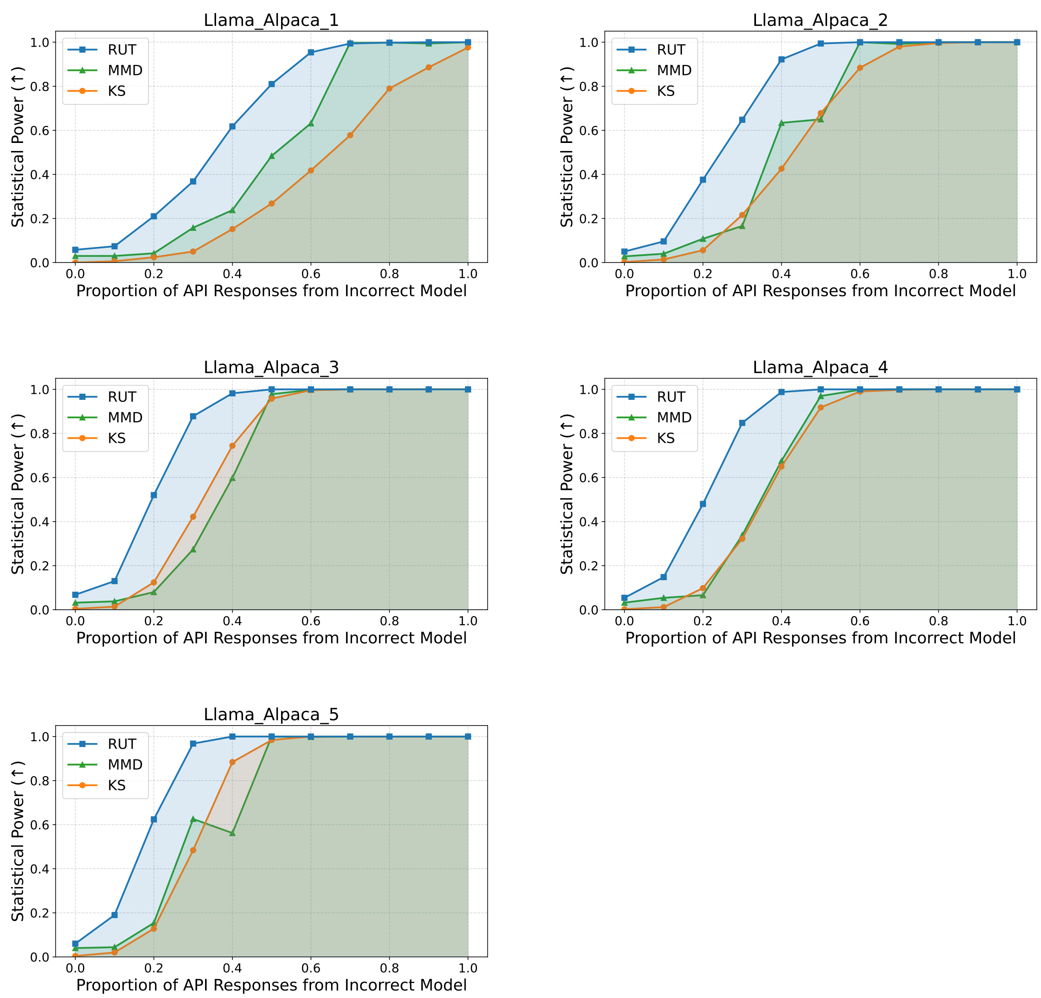
We present the full statistical power curves, showing the relationship between substitution rate and detection power, corresponding to the experiments on detecting jailbreak prompts described in Section 5.3. These curves are used to compute the power AUC values reported in the main paper and illustrate each method's detection power across different levels of substitution.

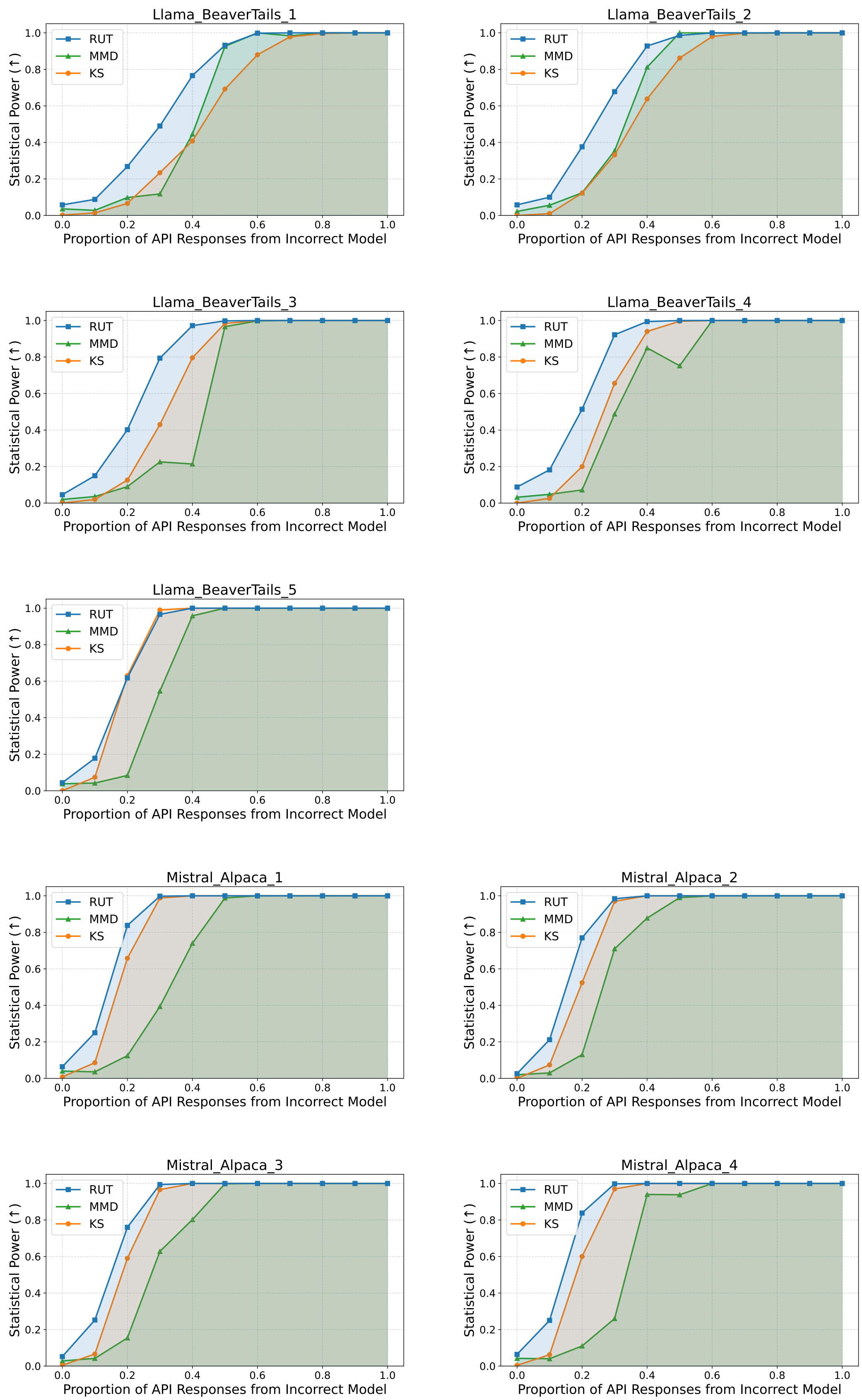




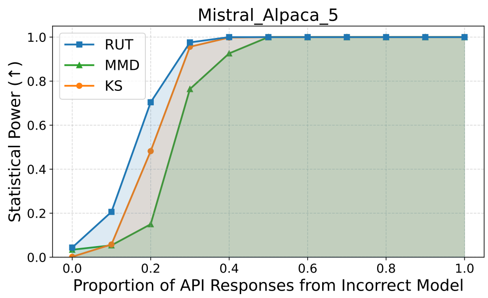
B.3 Full Statistic Power Curves for Detecting SFT

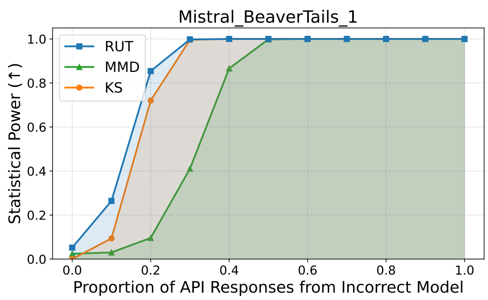
We present the full statistical power curves, showing the relationship between substitution rate and detection power, corresponding to the experiments on detecting SFT described in Section 5.4. These curves are used to compute the power AUC values reported in the main paper and illustrate each method's detection power across different levels of substitution.

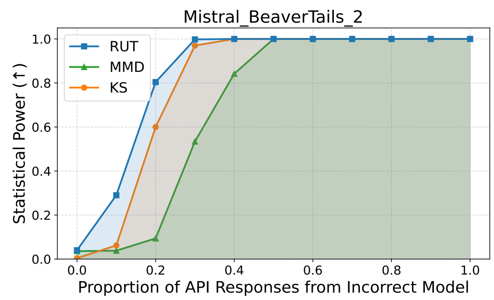


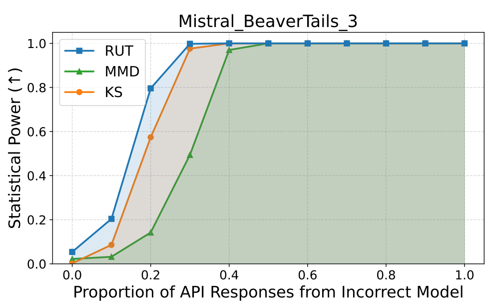


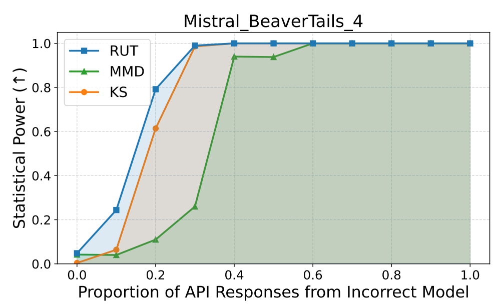
Llama_BeaverTails_1
Llama_BeaverTails_2
Llama_BeaverTails_3
Llama_BeaverTails_4
Llama_BeaverTails_5
Mistral_Alpaca_1
Mistral_Alpaca_2
Mistral_Alpaca_3
Mistral_Alpaca_4
RUT
MMD
KS
Statistical Power (↑)
Proportion of API Responses from Incorrect Model
0.0
0.2
0.4
0.6
0.8
1.0

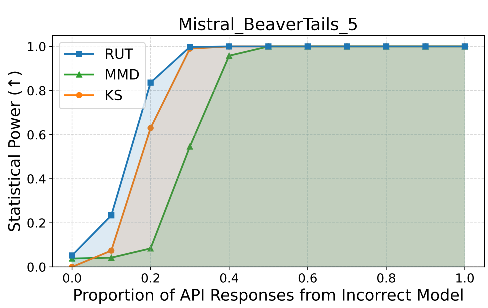






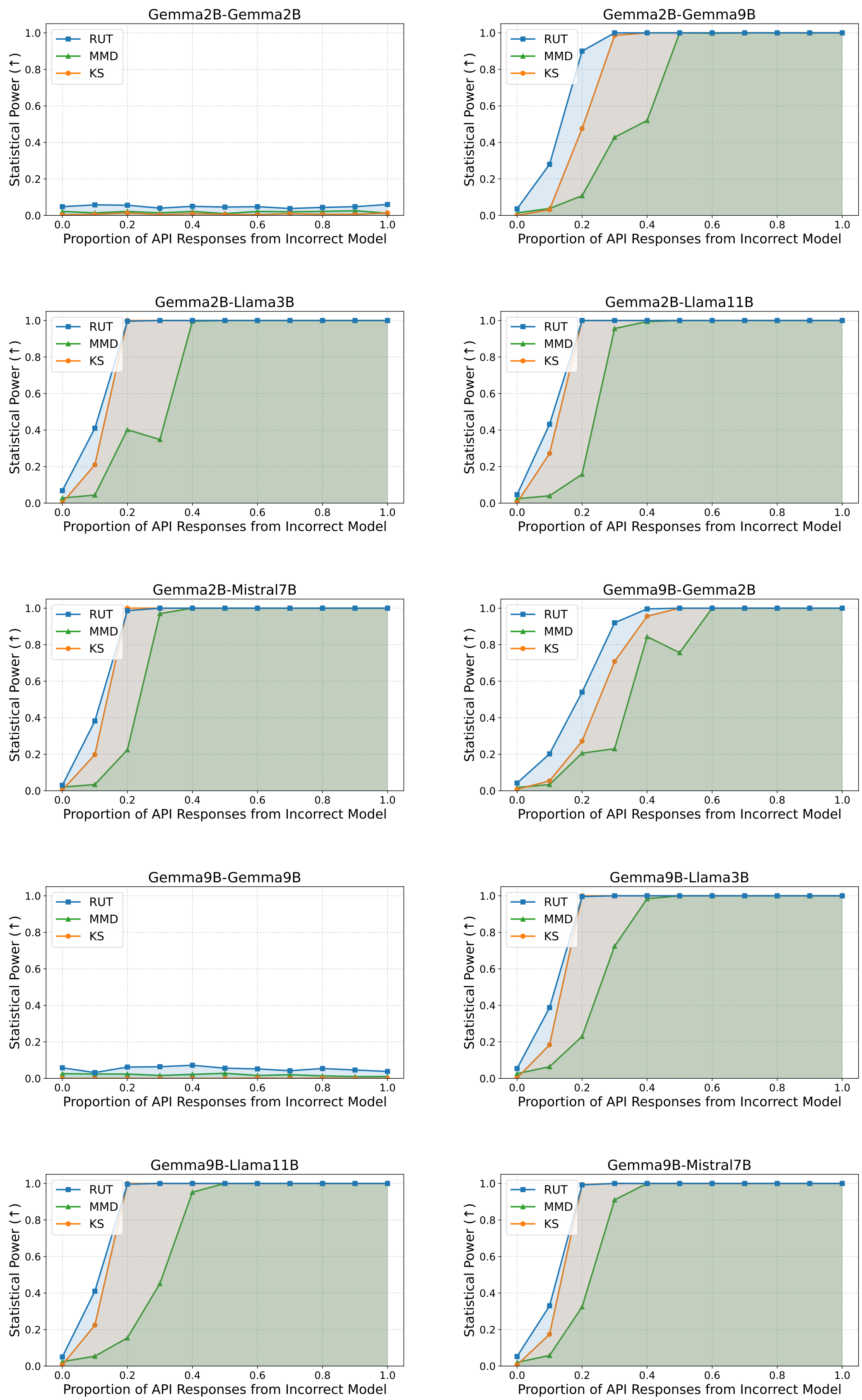


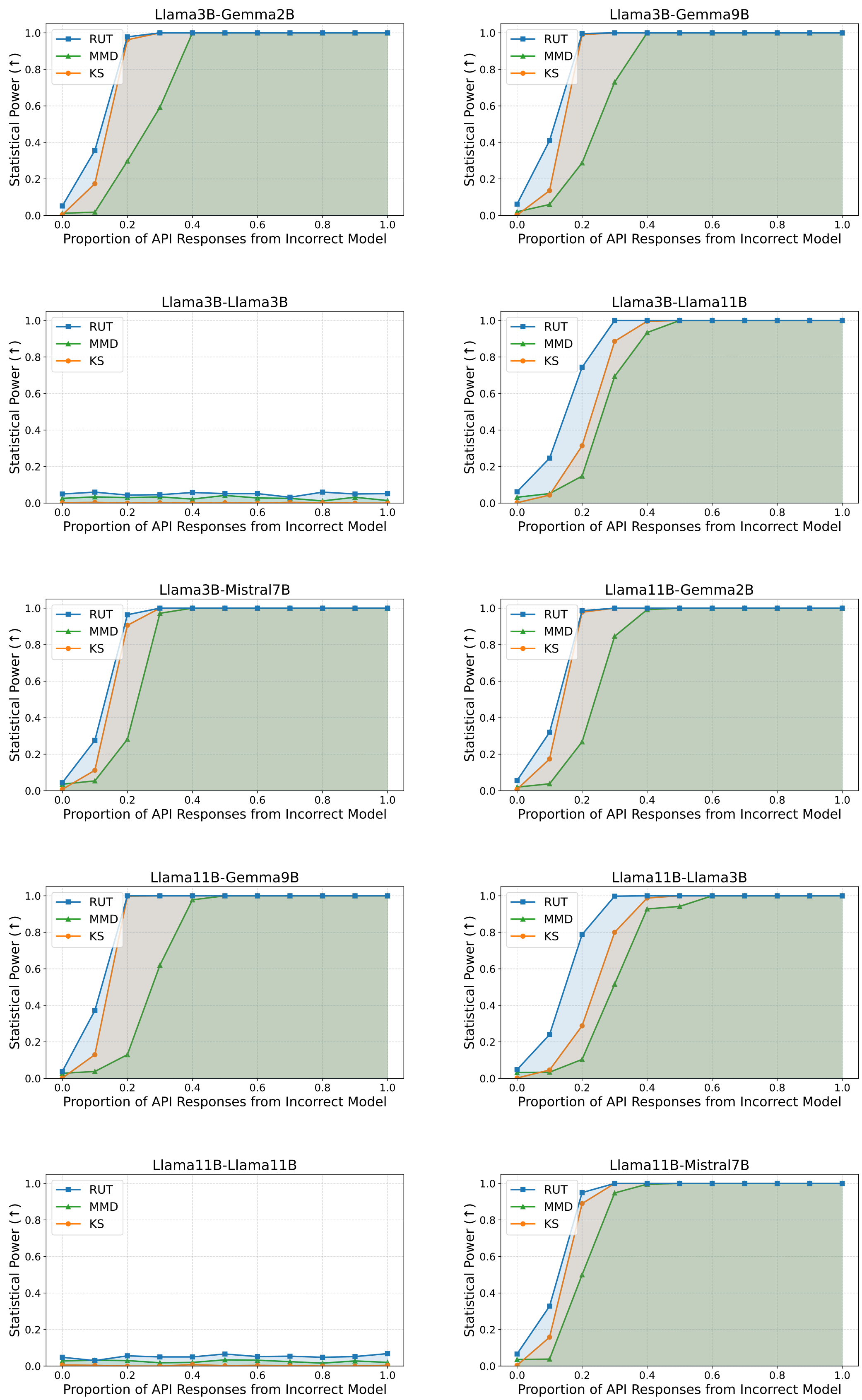


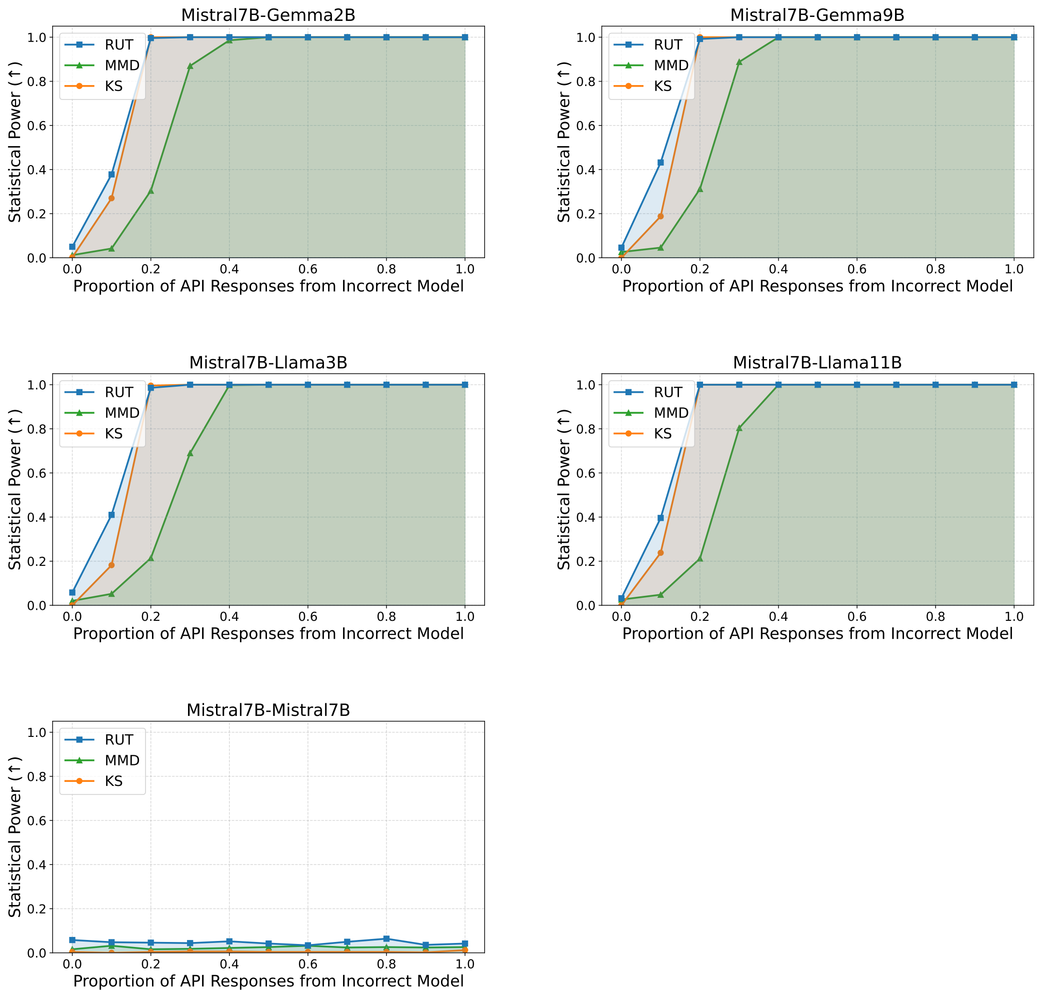


B.4 Full Statistic Power Curves for Detecting Full Model Replacement

We present the full statistical power curves, showing the relationship between substitution rate and detection power, corresponding to the experiments on detecting full model replacements described in Section 5.5. These curves are used to compute the power AUC values reported in the main paper and illustrate each method's detection power across different levels of substitution.


Gemma2B-Gemma2B
Gemma2B-Gemma9B
Gemma2B-Llama3B
Gemma2B-Llama11B
Gemma2B-Mistral7B
Gemma9B-Gemma2B
Gemma9B-Gemma9B
Gemma9B-Llama3B
Gemma9B-Llama11B
Gemma9B-Mistral7B
RUT
MMD
KS
Statistical Power (↑)
Proportion of API Responses from Incorrect Model


Llama3B-Gemma2B
Llama3B-Gemma9B
Llama3B-Llama3B
Llama3B-Llama11B
Llama3B-Mistral7B
Llama11B-Gemma2B
Llama11B-Gemma9B
Llama11B-Llama3B
Llama11B-Llama11B
Llama11B-Mistral7B
RUT
MMD
KS
Statistical Power (↑)
Proportion of API Responses from Incorrect Model
0.0
0.2
0.4
0.6
0.8
1.0



Mistral7B-Gemma2B
Mistral7B-Gemma9B
Mistral7B-Llama3B
Mistral7B-Llama11B
Mistral7B-Mistral7B
RUT
MMD
KS
Statistical Power (↑)
Proportion of API Responses from Incorrect Model
0.0
0.2
0.4
0.6
0.8
1.0