TREEVAL: ALLOCATING COMPUTE FOR EFFICIENT LLM SAFETY EVALUATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Safety evaluation of large language models often relies on repeatedly generating and judging complete responses, making thorough evaluation computationally expensive. We introduce TREEVAL, a framework that formulates safety evaluation as a compute-allocation problem over response probability mass, reusing alternative continuations exposed during generation instead of repeatedly generating isolated full rollouts. A simple instantiation, TREEVAL-DFS, achieves over 98% correlation with repeated-sampling safety measurements with an average estimated compute gain of 69×. We find that this efficiency is enabled by broader response-space coverage and by noisy but informative partial-response judgments that preserve much of the model-level safety signal. We further characterize the tradeoff between allocating compute within versus across prompts: greater inter-prompt exploration exposes more harmful behavior, and these differences show similar trends to standard measures of response diversity.

1 INTRODUCTION

Large language models are increasingly deployed in user-facing and operational settings where unsafe behavior can cause real harm, including harmful user interactions, security failures, and organizational risk (Perez et al., 2022; Shevlane et al., 2023; Phuong et al., 2024). Measuring these risks is essential for validating safeguards, comparing models, and deciding when systems are safe enough to deploy. Yet safety evaluation can be computationally expensive, making comprehensive or repeated testing difficult and limiting how thoroughly models can be assessed in diverse settings.

Many current safety evaluations repeatedly generate complete model responses and then judge whether those responses are harmful (Chao et al., 2024a; Perez et al., 2022; Mazeika et al., 2024; Souly et al., 2024; Zou et al., 2023). This treats each attempt as an independent rollout, even though autoregressive generation naturally defines a branching space of possible continuations. Tree-based methods have been explored for jailbreak search and safety evaluation (Mehrotra et al., 2024; Hwangbo et al., 2025; Lin et al., 2025), but typically introduce a particular search procedure or operate over entire candidate responses rather than exposing token-level response structure. This leaves open a broader question: how should limited evaluation compute be allocated across a model's response distribution to measure safety as efficiently as possible?

We take a different view: efficient safety evaluation is fundamentally a problem of *allocating limited compute across the model's response distribution*. At each generation step, an autoregressive model already exposes a distribution over possible next tokens, yet standard sampling follows one continuation and discards the others before repeating the process from scratch. We introduce TREEVAL, a framework that represents these alternatives as a probability-weighted response tree and makes prompt perturbation, tree expansion, judging, and stopping explicit compute-allocation decisions. This view allows an evaluator to reuse shared prefixes, judge alternative continuations without fully generating each one, and track the response probability mass that has actually been evaluated.

We instantiate this framework with TREEVAL-DFS, a simple depth-first policy that retains and judges alternative continuations exposed during generation. Across target models, it recovers the safety signal of repeated full-response sampling with over 98% correlation and an average estimated compute gain of 69×. Breadth-first, depth-first, and random exploration also preserve much of the baseline signal, indicating that the gain does not depend on a highly optimized traversal strategy. Instead, our ablations show that efficiency primarily comes from covering more response probability

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

mass per unit compute: evaluating exposed alternative branches recovers much of the signal lost when using only a single completed response.

This efficiency is possible because safety-relevant information often appears well before a response is complete. Prior work has shown that safety alignment itself can be only a few tokens deep, with early decoding decisions strongly shaping whether a model refuses or complies (Qi et al., 2025), while streaming moderation has shown that harmful outputs can often be detected before generation finishes (Li et al., 2026). We leverage this phenomenon for evaluation by judging partial responses rather than fully generating every exposed branch. Although such judgments can be noisy and artifact-sensitive (Chen & Goldfarb-Tarrant, 2025; Marioriyad et al., 2025), we find that they preserve substantial model-level safety signal. When judging is cheaper than target-model generation, this favors judging already-exposed branches over spending additional compute completing them.

The compute-allocation view also exposes a choice between exploring alternative continuations under the same prompt and exploring new response distributions induced by nearby prompts, which we term *in-prompt* and *inter-prompt* exploration. Under matched compute, greater inter-prompt allocation exposes more harmful behavior, indicating that nearby semantically neutral prompts can induce meaningfully different response distributions. Across perturbation methods, these harmfulness differences also show similar trends to standard lexical, semantic, and distributional measures of response diversity, including on benign dialogue, suggesting a connection between safety exploration and broader response diversity.

We summarize our contributions as follows:

- **Safety evaluation as compute allocation.** We formulate LLM safety evaluation as allocating compute over response probability mass, with prompt perturbation, response-tree expansion, judging, and stopping as explicit decisions.
- **Compute-efficient safety measurement.** We instantiate this framework with TREEVAL-DFS and show that it closely preserves the model-level safety signal with substantially less target-model compute. Our analysis identifies broader response-space coverage and informative partial-response judgments as key sources of this efficiency.
- **Within- versus across-prompt exploration.** We characterize this allocation tradeoff and show that harmfulness under semantically neutral perturbations tracks conventional measures of response diversity.

2 Related Work

Repeated sampling for safety evaluation. Because LLM generation is stochastic, repeated sampling is widely used to characterize model behavior beyond a single completion. In safety, this appears in automated red teaming and jailbreak evaluation, where multiple attacks or responses are generated (Perez et al., 2022; Zou et al., 2023; Chao et al., 2024a; Mazeika et al., 2024; Souly et al., 2024). Recent work makes sampling itself an explicit attack or evaluation dimension: Best-of-N Jailbreak repeatedly applies approximately semantics-preserving prompt corruptions (Hughes et al., 2024), LIAR explores stochastic short continuations (Beetham et al., 2024), and sampling-aware attacks and rare-event analyses study failures that emerge under repeated generation (Beyer et al., 2026; Jones et al., 2025; Feng et al., 2026). Yet these approaches largely treat independent full rollouts as the unit of evaluation, leaving how limited compute should be allocated across the model's response distribution largely unexplored.

Tree-based exploration and safety measurement. Tree structures have been used for reasoning search (Yao et al., 2023), prompt-level jailbreak optimization (Mehrotra et al., 2024), and, more recently, token-level safety exploration (Lin et al., 2025; Hwangbo et al., 2025). Boa searches for sufficiently probable harmful trajectories using breadth-first sampling and safety-guided priority search (Lin et al., 2025), while InFerActive represents stochastic generation as a navigable sampling tree to improve harmful-response discovery and human evaluation efficiency (Hwangbo et al., 2025). Existing tree-based approaches therefore use response trees primarily for search, discovery, or interactive inspection, while their role as a general abstraction for safety measurement under limited compute has received comparatively less attention.

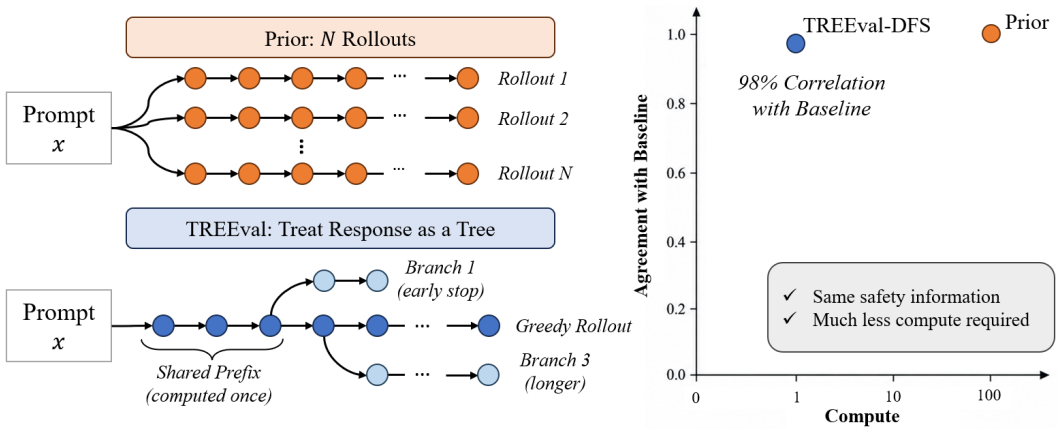

Figure 1: **TREEval allocates safety-evaluation compute over the response distribution rather than independent full rollouts.** Standard sampling judges complete trajectories, while TREEval retains alternative continuations exposed during generation, reuses shared prefixes, and can judge branches early. This preserves the baseline model-level safety signal with much less compute.

3 METHOD: TREEVAL

Standard sampling-based evaluation spends compute generating and judging isolated complete responses. TREEval instead treats evaluation as the problem of allocating a finite compute budget over the model's response distribution. We use probability-weighted *response trees* to expose alternative continuations and their probabilities, allowing an evaluator to decide which regions to expand and judge. We first define this representation, then formulate safety evaluation as a compute-allocation problem, and finally instantiate the framework with a simple policy used in our experiments.

3.1 REPRESENTING THE RESPONSE DISTRIBUTION

We begin by expressing the standard autoregressive response distribution in a form that exposes alternative continuations. Let M_θ denote an autoregressive model which, given a prompt x and a prefix $y_{<i}$, returns a distribution over the next token: $M_\theta(x, y_{<i}) = p_\theta(\cdot \mid x, y_{<i})$. A complete response $y = (y_1, \ldots, y_d)$ is obtained by iteratively applying M_θ, with probability

$$P_\theta(y \mid x) = \prod_{i=1}^{d} p_\theta(y_i \mid x, y_{<i}). \tag{1}$$

We now express this process in a tree-based form. Starting from the root node $v_0 = x$, each non-root node v_i corresponds to a token y_i. A sequence of nodes along a path encodes a prefix: for nodes v_i and v_j with $i \le j$, we define $\mathrm{Path}(v_i, v_j) = (v_i, v_{i+1}, \ldots, v_j)$, so that a prefix is given by $y_{<i} = \mathrm{Path}(v_1, v_{i-1})$. For a node v reached via $\mathrm{Path}(v_0, v)$, its children are

$$\mathrm{Children}(v) = \{t \in \mathrm{Vocab} \mid p_\theta(t \mid \mathrm{Path}(v_0, v)) > 0\}. \tag{2}$$

To simplify notation, we define $p_\theta(v_i) = p_\theta(v_i|\mathrm{Path}(v_0, v_{i-1}))$. With this notation, a complete response corresponds to a root-to-leaf path $(v_1, \ldots, v_d)$, with probability

$$P_\theta(y \mid x) = P_\theta(v_d) = \prod_{i=1}^{d} p_\theta(v_i). \tag{3}$$

The important consequence of this representation is that a target-model forward pass exposes more than the single token ultimately sampled: it exposes a distribution over alternative continuations and their probabilities. A response tree retains these alternatives, allowing multiple regions of the response distribution to be evaluated without independently regenerating their shared prefixes.

The same representation also extends where evaluation may occur. Rather than applying a judge J only to a completed response $J(y)$, we can associate a judgment with any node v through its partial path $\mathrm{Path}(v_0, v)$. In adversarial settings, we additionally consider perturbations of the input prompt. Let $\mathcal{A}(x)$ denote a perturbation process (e.g., jailbreak transformations), with $x' \sim \mathcal{A}(x)$. Each perturbed prompt x' induces a distinct response tree under M_θ, defining another response distribution that can be explored and evaluated.

3.2 Safety Evaluation as Compute Allocation

TREEval views evaluation as estimating a target property of the model's response distribution under a finite compute budget. Let Q denote the true quantity of interest and $\hat{Q}$ its estimate. In safety evaluation, Q corresponds to the model's harmfulness under a prompt distribution, while $\hat{Q}$ is computed from evaluated regions of one or more response trees.

We model evaluation as a sequential decision process. At step t, the evaluator observes a state $S_t = (\mathcal{T}_t, \mathcal{V}_{J,t})$, where $\mathcal{T}_t$ is the current forest of response trees and $\mathcal{V}_{J,t}$ is the set of nodes judged so far. For non-overlapping judged leaves, a response-tree evaluator estimates harmfulness by probability-weighting their judge labels:

$$\hat{Q}(\mathcal{V}_J) = \frac{\sum_{v \in \mathcal{V}_J} P_\theta(v) \cdot J\big(\mathrm{Path}(v_0, v)\big)}{\sum_{v \in \mathcal{V}_J} P_\theta(v)}. \tag{4}$$

The denominator is the evaluated *coverage*: the total response probability mass represented by the judged regions. At partial coverage, Equation 4 therefore measures harmfulness over the evaluated regions rather than providing an unbiased estimate of full-distribution harmfulness. Appendix B.3 gives the corresponding non-overlapping formulation for arbitrary judged nodes.

Probability weighting is important because exposed tree leaves can represent substantially different amounts of probability mass, unlike independent responses sampled from the model distribution; treating all leaves equally would therefore over-weight low-probability continuations. We refer to Equation 4 as *Harm %*, the probability-weighted analogue of average attack success.

Under a finite budget, the evaluator must decide how to spend the next unit of compute. A policy ρ maps state to an action, $a_t \sim \rho(\cdot \mid S_t)$, thereby deciding both how evaluation proceeds and when it terminates. The action a_t may be STOP, or one of the following:

- **Perturb a prompt:** sample a semantically related prompt $x' \sim \mathcal{A}(x)$, inducing a new response tree rooted at x'.
- **Expand the tree:** query M_θ once at an existing node v and add $\mathrm{Children}(v)$ to the tree.
- **Judge a node:** apply $J(v)$ to a node v.

These actions incur both computation and storage costs. Prompt perturbation may require attack or rewriting compute, tree expansion requires target-model inference, and judging requires judge compute. Retaining alternative continuations additionally requires storing the nodes exposed by each expansion. Although a complete response tree grows exponentially with depth, materializing such a tree would also require expanding exponentially many nodes. For a fixed number of model forward passes, storage instead grows with the number of candidate children exposed by those expansions. Each stored node contains only lightweight response and probability metadata, and memory was not a limiting resource in our experiments. We report the observed branching and resulting storage overhead alongside our compute analysis in Section 4.2.

More generally, these decisions can be formalized as balancing estimation quality against evaluation cost; we give a general optimization objective in Appendix B. Our experiments do not optimize this objective, but instead use simple fixed policies to study the main allocation choices.

Standard sample-based evaluation is a restricted policy within this framework: it repeatedly samples a prompt or perturbation, follows one trajectory to completion, judges only the completed response, and then repeats. TREEval expands this design space by retaining alternative continuations already exposed during generation, judging partial trajectories when useful, and explicitly tracking how much response probability mass has been evaluated.

This formulation exposes three allocation choices that we study empirically. First, compute may be spent on *inter-prompt* exploration, which samples new $x' \sim \mathcal{A}(x)$ and therefore new response distributions, or on *in-prompt* exploration, which expands additional continuations within a fixed response tree. Second, the evaluator must decide *when to judge*: earlier judgments save target-model compute but may be less reliable than judgments on complete responses. Third, compute must be divided between generation and judging. When judging is cheap relative to target-model inference, it may be advantageous to evaluate several already-exposed branches rather than spend additional generation compute completing each one independently.

3.3 A Simple Instantiation: TREEval-DFS

To evaluate the framework, we use **TREEval-DFS**, a deliberately simple policy intended to instantiate the framework rather than optimize over the full compute-allocation policy space. Like Best-of-N (BoN) evaluation, it begins from perturbed versions $x' \sim \mathcal{A}(x)$ of each dataset prompt. Unlike BoN, it retains the alternative continuations exposed during generation and evaluates their probability mass rather than discarding all but the sampled trajectory.

For each perturbed prompt, TREEval-DFS expands a path until reaching the depth limit or exhausting its target-model budget. At each expansion, it records all candidate next-token children and continues along the highest-probability child, unlike standard DFS. The unchosen children remain as leaves and can be judged without another target-model forward pass. After expansion, the policy judges leaves that satisfy a minimum depth $d_{\min}$ and aggregates their labels using the path probabilities in Equation 4. Their summed probability mass gives the evaluated coverage. We use $d_{\min} = 7$ because very short prefixes can produce unstable safety labels, which we analyze in Section 4.3.

The budget C counts target-model forward passes; our end-to-end experimental cost additionally accounts for judge calls. Across dataset prompts, we aggregate prompt-level estimates in proportion to their evaluated coverage. Algorithm 1 in Appendix B.2 gives the single-prompt procedure and shows the corresponding BoN operations for comparison.

4 Results

We first test whether TREEval-DFS preserves the safety signal of conventional repeated sampling while using less compute. We then analyze what enables this efficiency, focusing on response-space coverage, traversal policy, and partial-response judging, before studying how compute allocation between in-prompt and inter-prompt exploration affects safety measurement and response diversity.

4.1 Setup

We evaluate on the harmful split of JBB-Behaviors (Chao et al., 2024a), a widely used benchmark for jailbreak and safety evaluation. We select target models primarily from the most-downloaded text-generation models on HuggingFace (Hugging Face, 2026): Qwen3 0.6B (Team, 2025), Qwen3.5 variants (Qwen Team, 2026), Qwen2.5 7B (Team, 2024), and Llama3.1 8B (AI@Meta, 2024). We additionally evaluate KimiK2.6 (Moonshot AI, 2026) and Gemma4 31B (Google, 2026); KimiK2.6, Gemma4 31B, and Qwen3.5 397B are accessed through hosted inference (Together AI, 2026). We use instruction-tuned variants throughout and Llama3.1 8B as the default target model. For safety judging, we evaluate keyword filtering (Zou et al., 2023), WildGuard (Han et al., 2024), and Llama-Guard3 1B/8B (Llama Team, 2024), using LlamaGuard3 1B by default. Dataset and model license details are in Appendix C.1.

For the sampling baseline, we use BoN Jailbreak (Hughes et al., 2024), whose text corruption $\mathcal{A}$ has negligible compute cost. We independently perturb each prompt and generate responses to depth 30. TREEval-DFS retains next-token candidates after the model's decoding filters (temperature 0.6, top-p 0.9, top-k 50, and probability thresholding where applicable); additional generation details and response-tree statistics are provided in Appendix C.1. Unless otherwise noted, TREEval and the sampling baseline use the same prompt-perturbation distribution and decoding configuration, so their primary difference is how target-model inference is allocated and which exposed continuations are retained and judged.

We distinguish three safety measurements. **ASR@k** reports whether any of k attempts is harmful, while **AAS@k** averages the binary outcomes across attempts and estimates average per-response harmfulness. **Harm@k** additionally weights each outcome by its response probability, and **Harm (%)** generalizes this quantity to probability-weighted regions of a response tree (Equation 4). This weighting is necessary for tree evaluation because different leaves can represent substantially different amounts of response probability mass. **Coverage** is the fraction of response probability mass represented by judged regions.

ASR@k is appropriate when the number of attempts k is fixed, but it increases monotonically with additional attempts and can eventually saturate near 100%, even when the underlying response distribution is unchanged. This makes it less suitable when the objective is to reduce evaluation compute: we instead want a safety quantity whose estimate stabilizes as more compute is spent. AAS@k has this property under repeated sampling, converging toward average per-response harmfulness. For response-tree evaluation, Harm % plays an analogous role: additional compute increases the response probability mass represented in the estimate rather than mechanically increasing the metric through a maximum over attempts. This allows us to study how quickly different evaluation policies recover a stable model-level safety signal as coverage increases.

We measure agreement between model-level safety estimates using Pearson correlation r. **Compute** is measured in target-model forward passes and approximates judge cost using the judge-to-target model-size ratio. We separately consider the storage overhead of retaining alternative continuations in Section 4.2.

4.2 Efficient Safety Measurement

We first ask whether allocating compute over response probability mass can recover the safety signal obtained from expensive repeated sampling. Table 1 shows that TREEval-DFS closely preserves the model-level harmfulness measured by BoN while evaluating substantially more response probability mass per unit target-model compute. Differences are mainly among models with similar harmfulness, where small estimation changes can alter rank. Coverage varies with response-tree structure: probability mass is harder to cover when it fragments across many candidate continuations.

Importantly, the correlation in Table 1 measures preservation of the *model-level safety signal*, rather than numerical agreement between finite-compute Harm estimates. BoN estimates harmfulness from a sparse set of completed trajectories, whereas TREEval aggregates probability-weighted judgments over the response-tree regions it covers. At limited coverage, their absolute Harm values can therefore differ even when the relative safety signal across models is closely preserved. We use correlation to measure this signal recovery, while Appendix C.4 shows how the estimates evolve as additional compute is spent.

Table 1: **TREEval-DFS preserves the model-level safety signal with much less compute.** BoN Harm and TREEval-DFS Harm are strongly correlated across target models ($r = 0.9828$). Coverage and compute gains are relative to BoN. Harmfulness ranks are shown in parentheses; $*$ denotes API-accessed models.

Model	BoN Harm (%)	TREEval-DFS Harm (%)	TREEval-DFS Coverage (%)	Coverage Gain (×)	Compute Gain (×)
Average	10.55	24.23	49.26	10.67	69.41
Qwen3 0.6B	34.15 (1)	73.45 (1)	38.49	4.91	19.50
Qwen3.5 0.8B	33.25 (2)	64.92 (2)	10.90	26.70	16.83
Qwen3.5 2B	9.18 (3)	14.97 (5)	26.22	28.24	40.52
Qwen3.5 4B	6.03 (4)	15.26 (4)	46.45	8.62	70.01
Qwen2.5 7B	5.96 (5)	23.96 (3)	67.64	5.19	83.87
Llama3.1 8B	4.84 (6)	14.21 (6)	84.07	3.59	86.10
KimiK2.6 1.1T*	4.83 (7)	12.27 (8)	36.44	14.39	99.60
Qwen3.5 9B	3.33 (8)	14.10 (7)	63.26	6.05	86.53
Gemma4 31B*	2.04 (9)	2.84 (10)	69.38	2.81	92.09
Qwen3.5 397B*	1.84 (10)	6.30 (9)	49.72	6.16	99.06

Table 2 isolates where TREEval's efficiency comes from. Pearson r measures model-level correlation with the full-sampling AAS@100 reference, coverage is the judged response probability mass, and compute reduction is relative to AAS@100. Reducing the baseline from many complete responses to a single complete response substantially lowers compute but also loses model-level safety signal. TREEval starts from a similarly small generation budget, but retains and judges alternative continuations already exposed during generation, increasing coverage and recovering most of the full-sampling signal. BFS, standard DFS, and random frontier selection also remain strongly correlated with the baseline, indicating that much of the gain comes from retaining and probability-weighting exposed response regions rather than from a specialized traversal heuristic. The different policies expose a coverage–information tradeoff: shallower exploration covers more probability mass, while deeper continuations provide more informative judgments.

Retaining these alternative branches requires additional storage, but its practical cost scales with the number of exposed children rather than with the size of the complete response tree. In our experiments, expanded nodes retain approximately 1–5 children on average depending on the model; for example, Qwen3.5 0.8B, Llama3.1 8B, and Qwen3.5 9B average 4.31, 1.66, and 1.42 retained children per expanded node, respectively. Thus, after C target-model expansions, storage grows approximately with the number of candidates exposed by those C forward passes. For intuition, a depth-d trajectory exposing five candidates per step stores roughly $5d$ lightweight token-and-probability records. Although an exhaustively materialized response tree would grow exponentially, constructing it would likewise require exponentially many expansions; at the compute budgets considered here, memory was not a limiting resource.

Table 2: **Where TREEval's efficiency comes from.** Results on offline models, with AAS@100 as the reference for model-level Pearson correlation.

Method	Pearson r	Coverage (%)	Compute Gain
AAS@100	–	6.55	1.0×
Harm@100	.996	6.55	1.0×
Harm@1	.896	5.67	100.0×
BFS	.972	67.42	83.6×
DFS	.971	48.32	52.7×
TREEval-DFS	**.982**	48.15	53.1×
Random Search	.963	61.08	67.4×

Additional compute-scaling results for ASR, harmfulness, and estimator variance are provided in Appendix C.4.

4.3 Partial-Response Judging

TREEval's efficiency also depends on judging exposed branches without completing each one. Figure 2 shows that partial responses already contain substantial safety information: agreement with the corresponding max-depth judgment generally increases quickly with response depth across both target models and judges. Although individual prefix labels remain noisy, particularly early in generation, the signal becomes increasingly stable with depth and preserves useful model-level safety information. This supports partial-response judging as a way to trade additional generation compute for noisier but cheaper safety measurements.

The irregular behavior at very shallow depths partly reflects incomplete semantic content rather than meaningful changes in safety behavior. Early prefixes may consist largely of chat-template tokens, static assistant prefixes, or incomplete phrases whose apparent intent changes after only a few additional tokens; differences in templates and judges can therefore shift where agreement begins to stabilize. These effects motivate the minimum judging depth $d_{\min} = 7$ used by TREEval-DFS rather than judging every exposed continuation immediately. Appendix C.2 further analyzes depth-wise label transitions and judge-specific instability.

Together with Table 2, these results show that maximizing probability-mass coverage alone is not sufficient. Shallower exploration can cover more of the response distribution per unit compute, but attaches judgments to less informative prefixes; deeper exploration sacrifices some coverage for more reliable safety information. Efficient evaluation therefore requires balancing both the amount of probability mass evaluated and the informativeness of the judgments assigned to that mass.

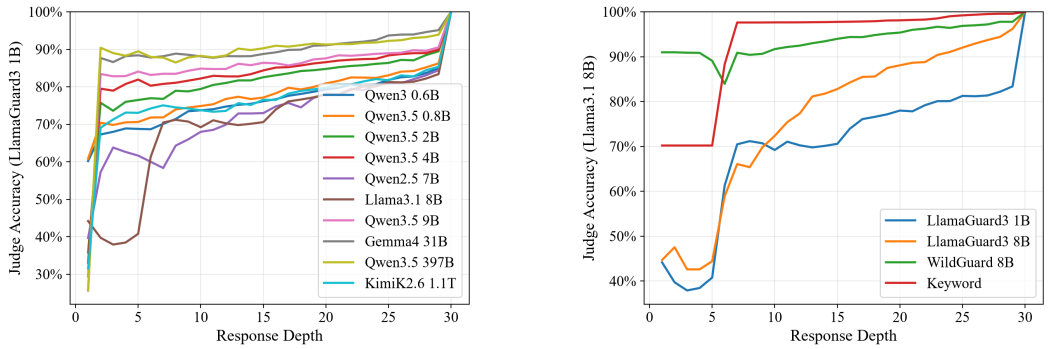

Figure 2: **Partial responses contain useful safety information before generation completes.** Agreement with the judgment on the same trajectory at maximum depth ($d = 30$) is shown across response depth. (Left) LlamaGuard3-1B across target models. (Right) Different judges on Llama3.1-8B. The $d = 30$ label is a reference rather than external ground truth, so agreement is 100% there by construction.

4.4 In-Prompt vs. Inter-Prompt Allocation

The preceding experiments consider how efficiently compute can cover the response distribution induced by a given prompt. A separate allocation choice is whether to spend additional compute exploring that distribution more deeply, or instead to induce new nearby response distributions through prompt perturbation. We refer to these as *in-prompt* and *inter-prompt* exploration, respectively. Conventional repeated-sampling evaluations often mix these two sources of variation without treating their relative allocation as an explicit design choice. TREEval makes this distinction explicit, allowing us to compare breadth across prompt-conditioned distributions against depth within a fixed distribution under the same compute budget. Under matched compute, Figure 3 shows that shifting more budget toward independently perturbed prompts exposes more harmful behavior than expanding fewer response trees more deeply. Thus, even approximate semantics-preserving perturbations can move the model between meaningfully different response distributions rather than merely providing additional samples from the same one.

To test whether this effect reflects broader behavioral exploration rather than only a safety-specific change, we also measure conventional response diversity. Distinct-2 (Li et al., 2016) and Self-BLEU capture lexical diversity (Zhu et al., 2018), SBERT-div (Guo et al., 2025) measures semantic diversity, and Vendi (Friedman & Dieng, 2022) captures distributional diversity. These metrics show a broadly similar, though non-monotonic, trend as compute shifts toward inter-prompt exploration. This supports the interpretation that nearby prompt variants expose response behavior that deeper sampling within a single prompt-conditioned distribution does not fully recover.

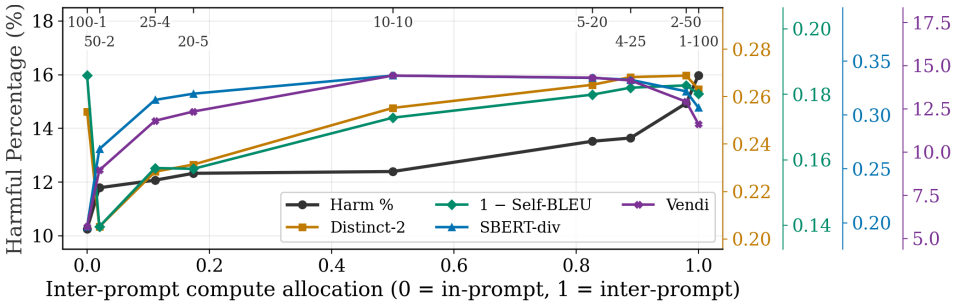

Figure 3: **Inter-prompt exploration accesses different response distributions under fixed compute.** We vary the allocation between deeper exploration within each response tree and independently perturbed prompts. Harmfulness increases as compute shifts toward inter-prompt exploration, while lexical, semantic, and distributional diversity measures show a broadly similar but non-monotonic trend. Diversity is computed within each source prompt, then averaged across prompts.

More broadly, the result shows that evaluation compute is not interchangeable across these two forms of exploration. Coverage is conditional on the prompt-induced response distribution: extensively covering $p_\theta(y \mid x)$ does not cover behavior that appears only under nearby $p_\theta(y \mid x')$. Increasing coverage within one response tree therefore cannot necessarily substitute for exploring additional prompt-induced distributions. Appendix C.3 studies this connection in more detail by comparing multiple perturbation types and testing whether safety measurements predict standard diversity metrics on JBB-Behaviors and whether those measures transfer to benign prompts.

5 Discussion and Limitations

Our results suggest that safety evaluation can be viewed as allocating measurement compute over a model's response distribution rather than repeatedly generating isolated complete responses. In this work, TREEval Harm measures harmfulness over evaluated response probability mass under a specified prompt distribution, perturbation process, decoding procedure, and judge. This quantity is useful for comparing distributional harmfulness, but it is not intended to directly represent deployment risk or replace worst-case red-teaming objectives such as ASR@K. In particular, probability weighting intentionally gives less influence to rare branches, whereas rare but severe failures may require tail-risk or discovery-oriented objectives.

TREEval is most useful when target-model generation is expensive relative to judging and partial responses already contain informative safety signal. Our results show that such judgments can preserve useful model-level information even when individual prefix labels are noisy, creating a tradeoff between additional generation and judgment reliability. Similarly, the in- versus inter-prompt analysis shows that evaluation compute is not interchangeable: nearby prompt variants can induce response distributions that are not reached by deeper exploration of a single prompt. The observed relationship with conventional diversity metrics offers one explanation for this effect, but we treat that connection as exploratory rather than a general characterization of diversity.

More broadly, TREEval separates the safety quantity being estimated from the policy used to allocate compute toward that estimate. TREEval-DFS is only one simple instantiation: different settings may favor policies that prioritize coverage, deeper judgments, prompt exploration, or some combination depending on model branching, judge reliability, and relative inference costs. This suggests that safety evaluation itself can be treated as an adaptive resource-allocation problem rather than a fixed sampling protocol.

Several limitations also point to natural extensions of the framework. Partial-response reliability may vary across judges, models, response depths, and safety categories, and our simple traversal policies do not optimize over the full compute-allocation policy space. The current evaluation is also limited to jailbreak-style harmfulness, one main perturbation family, and a finite set of models, judges, and datasets. In addition, TREEval requires access to token-level candidate probabilities (or log probabilities), limiting the current instantiation to open-weight models and APIs that expose sufficient generation information. This restricts its applicability to many closed-model APIs, but also creates a useful asymmetry for safety evaluation: model developers can apply TREEval with internal probability access that may not be available to external users or attackers. Our compute accounting also approximates judge cost rather than measuring deployment-specific latency or monetary cost. Finally, TREEval introduces additional implementation overhead by requiring response-tree structure, probabilities, and judgment bookkeeping.

6 Conclusion

TREEval reframes LLM safety evaluation as a compute-allocation problem over the model's response distribution. Using probability-weighted response trees, a simple TREEval-DFS policy closely preserves the model-level safety signal of repeated sampling while using substantially less target-model compute. Our analyses show that this efficiency comes from covering more response probability mass per generation step and from informative partial-response judgments, while also revealing a tradeoff between exploring within and across prompts. More broadly, safety evaluation should account not only for which responses were sampled, but for which regions of the response distribution were measured and at what computational cost.

AI USE STATEMENT

In this work, we used generative AI tools to assist with implementing research methods. We additionally used generative AI tools to assist with software implementation, drafting and refining scientific figures and diagrams, improving the writing and readability of the paper, and surveying and identifying relevant literature. All AI-assisted code was tested, and figures, literature references, and written content were reviewed and verified by the authors. We take responsibility for the final content of this work, including all text, claims, code, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies methods for making language-model safety evaluation more compute-efficient. Such techniques are potentially dual-use: more efficient exploration of model response spaces could also inform more efficient jailbreak evaluation. However, our primary objective is to efficiently estimate aggregate safety properties of a model's response distribution, rather than to optimize for finding the existence of a harmful response, as in attack-oriented metrics such as ASR@K. Moreover, the current TREEval instantiation requires access to token-level candidate probabilities, which are readily available to model developers and open-weight evaluators but are often unavailable through closed-model interfaces. This limits direct application by external attackers while preserving its utility for internal safety evaluation. We discuss this distinction and related limitations in Section 5.

REPRODUCIBILITY STATEMENT

We provide the information needed to reproduce our methods and experiments throughout the paper and appendix. The TREEval framework is specified mathematically and algorithmically in Section 3, including the procedure and a more detailed algorithm in Algorithm 1, and we report the experimental configurations and evaluation procedures used in our experiments in Section 4.1. Our experiments use publicly available datasets and models, including the target models and safety judges, and established jailbreak methods; any additional algorithms introduced in this work are described in the paper. We additionally intend to release our implementation upon publication.

REFERENCES

AI@Meta. Llama 3.1 model card. 2024. URL https://github.com/meta-llama/llama-models/blob/main/models/llama3_1/MODEL_CARD.md.

James Beetham, Souradip Chakraborty, Mengdi Wang, Furong Huang, Amrit Singh Bedi, and Mubarak Shah. Liar: Leveraging alignment (best-of-n) to jailbreak llms in seconds. *arXiv preprint arXiv:2412.05232*, 2024.

Tim Beyer, Yan Scholten, Leo Schwinn, and Stephan Günnemann. Sampling-aware adversarial attacks against large language models. In *International Conference on Learning Representations*, volume 2026, pp. 74637–74663, 2026.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. A survey on evaluation of large language models. *ACM transactions on intelligent systems and technology*, 15(3):1–45, 2024.

Patrick Chao, Edoardo Debenedetti, Alexander Robey, Maksym Andriushchenko, Francesco Croce, Vikash Sehwag, Edgar Dobriban, Nicolas Flammarion, George J. Pappas, Florian Tramèr, Hamed Hassani, and Eric Wong. Jailbreakbench: An open robustness benchmark for jailbreaking large language models. In *NeurIPS Datasets and Benchmarks Track*, 2024a.

Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J. Pappas, and Eric Wong. Jailbreaking black box large language models in twenty queries, 2024b. URL https://arxiv.org/abs/2310.08419.

Hongyu Chen and Seraphina Goldfarb-Tarrant. Safer or luckier? llms as safety evaluators are not robust to artifacts. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19750–19766, 2025.

Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. Instructzero: Efficient instruction optimization for black-box large language models. *arXiv preprint arXiv:2306.03082*, 2023.

Gelei Deng, Yi Liu, Yuekang Li, Kailong Wang, Ying Zhang, Zefeng Li, Haoyu Wang, Tianwei Zhang, and Yang Liu. Jailbreaker: Automated jailbreak across multiple large language model chatbots. *arXiv preprint arXiv:2307.08715*, 2023.

Mingqian Feng, Xiaodong Liu, Weiwei Yang, Chenliang Xu, Christopher White, and Jianfeng Gao. Statistical estimation of adversarial risk in large language models under best-of-n sampling. *arXiv preprint arXiv:2601.22636*, 2026.

Dan Friedman and Adji Bousso Dieng. The vendi score: A diversity evaluation metric for machine learning. *arXiv preprint arXiv:2210.02410*, 2022.

Google. Gemma 4 31B IT. `https://huggingface.co/google/gemma-4-31B-it`, 2026. Hugging Face model card. Accessed: 2026-05-05.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*, 2023.

Yanzhu Guo, Guokan Shang, and Chloé Clavel. Benchmarking linguistic diversity of large language models. *Transactions of the Association for Computational Linguistics*, 13:1507–1526, 2025.

Seungju Han, Kavel Rao, Allyson Ettinger, Liwei Jiang, Bill Yuchen Lin, Nathan Lambert, Yejin Choi, and Nouha Dziri. Wildguard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of llms, 2024. URL `https://arxiv.org/abs/2406.18495`.

Hugging Face. Models: Text Generation, Sorted by Most Downloads. `https://huggingface.co/models?pipeline_tag=text-generation&sort=downloads`, 2026. Accessed: 2026-04-23.

John Hughes, Sara Price, Aengus Lynch, Rylan Schaeffer, Fazl Barez, Sanmi Koyejo, Henry Sleight, Erik Jones, Ethan Perez, and Mrinank Sharma. Best-of-n jailbreaking. *arXiv preprint arXiv:2412.03556*, 2024.

Junhyeong Hwangbo, Soohyun Lee, Hyeon Jeon, Kyochul Jang, Minsoo Cheong, Youngjae Yu, and Jinwook Seo. Inferactive: Interactive tree-based exploration of llm sampling for safety evaluation. *arXiv preprint arXiv:2512.10234*, 2025.

Erik Jones, Meg Tong, Jesse Mu, Mohammed Mahfoud, Jan Leike, Roger Grosse, Jared Kaplan, William Fithian, Ethan Perez, and Mrinank Sharma. Forecasting rare language model behaviors. *arXiv preprint arXiv:2502.16797*, 2025.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and William B Dolan. A diversity-promoting objective function for neural conversation models. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pp. 110–119, 2016.

Yang Li, Qiang Sheng, Yehan Yang, Xueyao Zhang, and Juan Cao. From judgment to interference: Early stopping llm harmful outputs via streaming content monitoring. *Advances in Neural Information Processing Systems*, 38:54305–54333, 2026.

Yanran Li, Hui Su, Xiaoyu Shen, Wenjie Li, Ziqiang Cao, and Shuzi Niu. Dailydialog: A manually labelled multi-turn dialogue dataset. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 986–995, 2017.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022.

Shuyi Lin, Anshuman Suri, Alina Oprea, and Cheng Tan. Llm jailbreak oracle. *arXiv e-prints*, pp. arXiv–2506, 2025.

Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. Autodan: Generating stealthy jailbreak prompts on aligned large language models. *arXiv preprint arXiv:2310.04451*, 2023.

AI @ Meta Llama Team. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Arash Marioriyad, Mohammad Hossein Rohban, and Mahdieh Soleymani Baghshah. The silent judge: Unacknowledged shortcut bias in llm-as-a-judge. *arXiv preprint arXiv:2509.26072*, 2025.

Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, et al. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal. *arXiv preprint arXiv:2402.04249*, 2024.

Anay Mehrotra, Manolis Zampetakis, Paul Kassianik, Blaine Nelson, Hyrum Anderson, Yaron Singer, and Amin Karbasi. Tree of attacks: Jailbreaking black-box llms automatically. *Advances in Neural Information Processing Systems*, 37:61065–61105, 2024.

Moonshot AI. Kimi-K2.6. https://huggingface.co/moonshotai/Kimi-K2.6, 2026. Hugging Face model card. Accessed: 2026-05-05.

Anselm Paulus, Arman Zharmagambetov, Chuan Guo, Brandon Amos, and Yuandong Tian. Advprompter: Fast adaptive adversarial prompting for llms. *arXiv preprint arXiv:2404.16873*, 2024.

Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. Red teaming language models with language models. *arXiv preprint arXiv:2202.03286*, 2022.

Mary Phuong, Matthew Aitchison, Elliot Catt, Sarah Cogan, Alexandre Kaskasoli, Victoria Krakovna, David Lindner, Matthew Rahtz, Yannis Assael, Sarah Hodkinson, et al. Evaluating frontier models for dangerous capabilities. *arXiv preprint arXiv:2403.13793*, 2024.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*, 2023.

Xiangyu Qi, Ashwinee Panda, Kaifeng Lyu, Xiao Ma, Subhrajit Roy, Ahmad Beirami, Prateek Mittal, and Peter Henderson. Safety alignment should be made more than just a few tokens deep. In *International Conference on Learning Representations*, volume 2025, pp. 54911–54941, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019.

Toby Shevlane, Sebastian Farquhar, Ben Garfinkel, Mary Phuong, Jess Whittlestone, Jade Leung, Daniel Kokotajlo, Nahema Marchal, Markus Anderljung, Noam Kolt, et al. Model evaluation for extreme risks. *arXiv preprint arXiv:2305.15324*, 2023.

Alexandra Souly, Qingyuan Lu, Dillon Bowen, Tu Trinh, Elvis Hsieh, Sana Pandey, Pieter Abbeel, Justin Svegliato, Scott Emmons, Olivia Watkins, et al. A strongreject for empty jailbreaks. *Advances in Neural Information Processing Systems*, 37:125416–125440, 2024.

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Together AI. Together AI Documentation: Inference. https://docs.together.ai/docs/quickstart, 2026. Accessed: 2026-05-05.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022.

Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. Texygen: A benchmarking platform for text generation models. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pp. 1097–1100, 2018.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.

A Extended Related Work

Repeated sampling and statistical safety measurement. Stochastic generation means that a single completion may poorly characterize an LLM's behavior, motivating repeated sampling in evaluation and safety testing (Bommasani et al., 2021; Liang et al., 2022; Chang et al., 2024). Safety benchmarks therefore commonly evaluate multiple responses to a fixed prompt or additionally vary the prompt through red teaming and jailbreak transformations (Perez et al., 2022; Zou et al., 2023; Chao et al., 2024a; Mazeika et al., 2024; Souly et al., 2024). Recent work makes this sampling dimension explicit: Best-of-N Jailbreaking repeatedly applies simple, approximately semantics-preserving prompt transformations (Hughes et al., 2024); LIAR explores stochastic short continuations at inference time (Beetham et al., 2024); sampling-aware attacks optimize against repeated decoding (Beyer et al., 2026); and rare-behavior forecasting and adversarial-risk estimation study how low-probability failures can be inferred from finite observations (Jones et al., 2025; Feng et al., 2026).

These works also highlight an important distinction between *discovery* and *distributional measurement*. Jailbreak evaluations often report ASR@K, which asks whether at least one of K attempts is harmful (Chao et al., 2024a; Mazeika et al., 2024). This is appropriate when success under a fixed number of attempts is the quantity of interest, but it necessarily increases with K even when the underlying response distribution is unchanged. In contrast, AAS under independent sampling provides a Monte Carlo estimate of average per-response harmfulness. TREEval targets the corresponding distributional question through explicit response-tree coverage, estimating harmfulness from probability-weighted judged regions rather than from isolated sampled trajectories. We therefore use distributional harmfulness for the main compute-efficiency comparison while retaining ASR@K as a distinct discovery-oriented objective.

Prompt-space search versus response-space allocation. A large body of red-teaming work develops procedures for finding inputs that elicit unsafe behavior (Perez et al., 2022; Zou et al., 2023; Liu et al., 2023; Paulus et al., 2024; Deng et al., 2023). Black-box methods such as PAIR iteratively refine candidate attacks, while Tree of Attacks with Pruning (TAP) branches and prunes a tree of candidate *attack prompts* to reduce target-model queries (Chao et al., 2024b; Mehrotra et al., 2024). Prompt optimization more generally searches over instructions or transformations to obtain a desired target behavior (Pryzant et al., 2023; Chen et al., 2023; Zhou et al., 2022; Guo et al., 2023).

These approaches primarily allocate computation over the *input space*. TREEval separates this from allocation within the stochastic response distribution. A perturbation $x' \sim A(x)$ induces a new response tree and therefore a new prompt-conditioned distribution, whereas expanding an existing tree explores additional continuations under the same input. Repeated jailbreak evaluation often mixes these two sources of variation, additional prompts and additional stochastic responses, without treating their relative allocation as an explicit design choice. TREEval makes this distinction explicit through inter-prompt and in-prompt exploration under a shared compute budget.

Token-level response-tree methods. The closest related methods also operate directly over the target model's token-level response space. Boa formalizes a "jailbreak oracle" problem, whether a sufficiently probable harmful trajectory exists under a model and decoding strategy, and searches the exponentially large response space using breadth-first sampling followed by safety-score-guided depth-first priority search (Lin et al., 2025). Its objective is principally existential: efficiently identify a harmful trajectory above a probability threshold. TREEval instead treats harmful response probability as a quantity to be estimated. Consequently, it aggregates probability mass across judged regions rather than directing search toward the most promising unsafe branch, and its framework can be instantiated with unguided policies such as BFS, DFS, or random frontier selection.

InFerActive is structurally even closer. It represents stochastic generation as a token-level sampling tree, preserves branch probabilities, and uses breadth-first sampling to cover early branching points more efficiently than independent random sampling (Hwangbo et al., 2025). Its primary goal is scalable human exploration: frontier nodes are eventually continued into readable responses, while an interactive interface lets evaluators inspect and extend regions of interest. TREEval shares the underlying sampling-tree representation but uses it for automated measurement under a compute budget. Exposed partial nodes can be judged without completing every branch, their probability mass contributes directly to the safety estimate and coverage calculation, and perturbation, expansion,

judging, and stopping are treated as common allocation decisions. The two approaches therefore use similar response-tree structure for complementary goals: interactive discovery in InFerActive and compute-efficient distributional measurement in TREEval.

Other tree-structured LLM methods. Tree structures are also widely used to organize higher-level candidate states. Tree of Thoughts branches over intermediate reasoning states (Yao et al., 2023), while TAP applies related branching and pruning ideas to candidate jailbreak prompts (Mehrotra et al., 2024). Such trees generally operate over generated semantic units, candidate strings, or prompts rather than directly over the next-token distribution. TREEval instead begins from autoregressive decoding itself: a single target-model forward pass exposes probabilities for multiple possible next-token continuations, allowing those alternatives and their shared prefixes to be retained for later expansion or judgment.

Early safety signals and automated judges. TREEval also builds on evidence that safety-relevant behavior can become apparent before generation is complete. Safety alignment may depend strongly on early decoding decisions (Qi et al., 2025), while streaming moderation shows that harmful outputs can often be detected from partial generations (Li et al., 2026). At the same time, automated safety judges can be sensitive to superficial features and other artifacts (Chen & Goldfarb-Tarrant, 2025; Marioriyad et al., 2025). TREEval therefore does not assume that partial-response labels are ground truth. Instead, early judging is treated as a compute–information tradeoff: extending a branch provides the judge with more semantic context but consumes additional target-model inference. Our experiments test whether these noisy intermediate judgments nevertheless retain enough model-level information to support efficient comparative safety measurement.

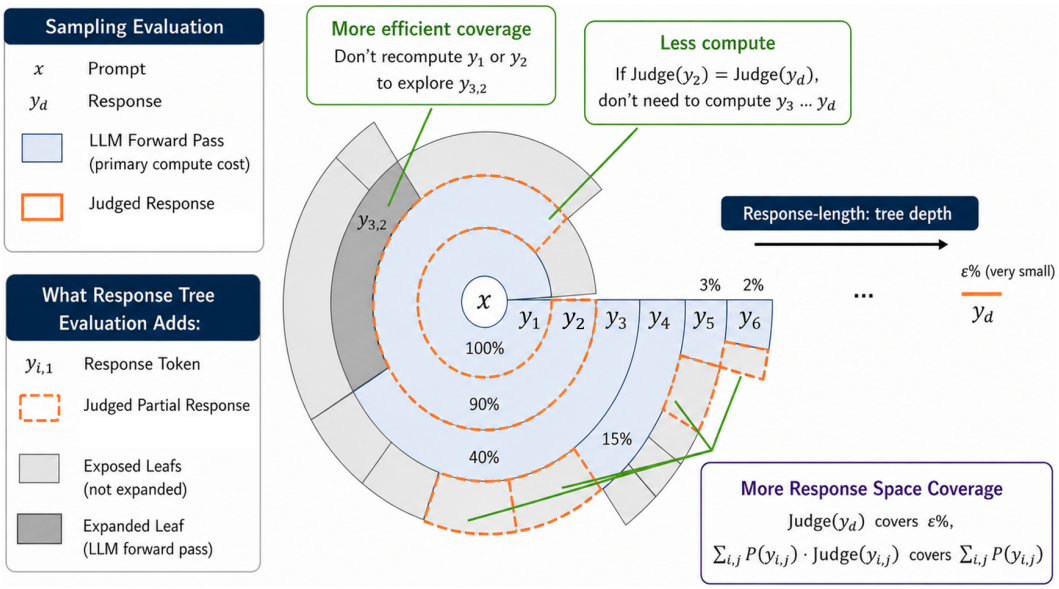

Figure 4: **Response tree evaluation leverages structure that independent samples discard.** The figure depicts a response tree in polar form: the root is the center circle, surrounding arcs are nodes, and radial distance indicates tree depth. Arc length indicates the cumulative probability of reaching each node from the root, while radial boundaries between depths represent parent-child relationships, with each outer arc partitioning the probability mass of its parent. Standard sampling follows a single blue path to the final response y_d, while exposing alternative light-gray leaves that TREEval can leverage. TREEval uses this structure to improve coverage efficiency, reduce compute through earlier partial-response judgments when reliable, and shift evaluation from counting completed samples to measuring probability-weighted response-space coverage (Appendix B.3 for coverage detail).

B TREEVAL FORMULATION AND ALGORITHM

This section provides additional detail on the TREEval formulation and the TREEval-DFS instantiation used in our experiments. Figure 4 gives a visual overview of the response-tree representation and the two main sources of efficiency: reusing alternative continuations exposed by each model query and judging branches before full generation when informative. We then formalize the broader compute-allocation objective, give the TREEval-DFS algorithm, and detail probability-mass accounting for arbitrary judged nodes.

B.1 GENERAL COMPUTE-ALLOCATION OBJECTIVE

The compute-allocation view can be expressed as a general policy-selection objective. A policy ρ chooses among prompt perturbation, tree expansion, judging, and stopping to trade off estimation error against cumulative evaluation cost:

$$\min_{\rho} \ \mathbb{E}_{\rho}\left[\left|Q - \hat{Q}(\mathcal{V}_{J,\tau_\rho})\right| + \lambda \sum_{t=0}^{\tau_\rho} \mathrm{Cost}(a_t; S_t)\right], \qquad \tau_\rho = \inf\{t \geq 0 : a_t = \textsc{Stop}\}. \tag{5}$$

Here, λ controls the tradeoff between estimation accuracy and evaluation cost. We use this objective to characterize the broader TREEval policy space; the experiments in this work do not optimize it directly, but instead study simple fixed traversal and allocation policies.

B.2 TREEVAL-DFS ALGORITHM

Algorithm 1 details the TREEval-DFS procedure for one dataset prompt. Its gray commented lines show the corresponding BoN operations for comparison; the budget C counts target-model forward passes.

Algorithm 1 TREEval-DFS: forest expansion followed by leaf evaluation

Input: prompt x, perturbation $\mathcal{A}$, model M_θ, judge J, depth $d_{\min}$, depth limit $d_{\max}$, budget C

$\mathcal{T} \leftarrow \emptyset,\ \mathcal{V}_J \leftarrow \emptyset$ ▷ Initialize forest and judged set

// $\mathcal{V}_J^{\mathrm{BoN}} \leftarrow \emptyset$ ▷ Initialize sampled-response judgments

while C not exhausted **do**

  $x' \sim \mathcal{A}(x)$ ▷ Perturb prompt / start new tree

  $v_0 = x',\ \mathcal{T} \leftarrow \mathcal{T} \cup \{v_0\},\ P_\theta(v_0) \leftarrow 1$ ▷ Add root to forest

  $v \leftarrow v_0$ ▷ Start DFS from root

  while C not exhausted and $\mathrm{depth}(v) < d_{\max}$ **do**

    $U \leftarrow \mathrm{Children}(v)$ ▷ LLM forward pass

    for all $u \in U$ **do**

      $\mathcal{T} \leftarrow \mathcal{T} \cup \{u\}$ ▷ Add child to current tree

    end for

    $v \leftarrow \arg\max_{u \in U} p_\theta(u)$ ▷ DFS along highest-probability child

  end while

  // $\mathcal{V}_J^{\mathrm{BoN}} \leftarrow \mathcal{V}_J^{\mathrm{BoN}} \cup \{J(\mathrm{Path}(v_0, v))\}$ ▷ Judge completed sample

end while

// **return** $\max(\mathcal{V}_J^{\mathrm{BoN}})$ ▷ Most harmful judgment, ASR@k

$\mathcal{U} \leftarrow \{v \in \mathcal{T} : v \text{ is a leaf and } \mathrm{depth}(v) \geq d_{min}\}$ ▷ Collect forest leaves

for all $v \in \mathcal{U}$ **do**

  $\mathcal{V}_J \leftarrow \mathcal{V}_J \cup \{v\}, \quad J(v) \leftarrow J\big(\mathrm{Path}(\mathrm{root}(v), v)\big)$ ▷ Judge leaf path

end for

$\hat{Q}(V_J) \leftarrow \frac{\sum_{v \in V_J} P_\theta(v) \cdot J(\mathrm{Path}(\mathrm{root}(v), v))}{\sum_{v \in V_J} P_\theta(v)}$ ▷ Probability-weighted estimate

return $\hat{Q}(\mathcal{V}_J)$

B.3 Non-Overlapping Judged Coverage

Equation 4 and Figure 4 write the TREEval estimate as a probability-weighted average over judged response regions. This expression is simplest when all judged nodes are leaves, since leaves represent disjoint regions of the response tree. However, TREEval can in general judge any node, including partial prefixes. In that case, care is needed: if both a node and one of its descendants are judged, then naively summing $P_\theta(v)$ over all judged nodes double-counts the descendant's probability mass.

To avoid this, we assign each judged node only the portion of its probability mass that has not already been covered by judged descendants. Let $\mathcal{T}$ be a response tree, let $V_J \subseteq \mathcal{T}$ be the set of judged nodes, and let $\mathrm{Children}_{\mathcal{T}}(v)$ denote the children of v that are present in the stored tree. We define the judged coverage of the subtree rooted at v recursively as

$$C_J(v) = \begin{cases} P_\theta(v), & v \in V_J, \\ \sum_{u \in \mathrm{Children}_{\mathcal{T}}(v)} C_J(u), & v \notin V_J. \end{cases} \tag{6}$$

This quantity is the total probability mass in the subtree of v whose safety has been covered by at least one judgment. If v itself has been judged, then its subtree is treated as covered by that judgment, but descendant judgments may still be used to refine how that mass is labeled.

We then define the exclusive judged mass assigned to a judged node v as

$$w_J(v) = \begin{cases} P_\theta(v) - \sum_{u \in \mathrm{Children}_{\mathcal{T}}(v)} C_J(u), & v \in V_J, \\ 0, & v \notin V_J. \end{cases} \tag{7}$$

Thus, $w_J(v)$ is the probability mass labeled by v's own judgment after subtracting the mass already covered by judged descendants. For example, if $P_\theta(v) = 0.10$ and judged descendants cover 0.02 probability mass, then v contributes 0.08 mass with label $J(\mathrm{Path}(v_0, v))$, while the remaining 0.02 is labeled by the corresponding descendant judgments.

The total non-overlapping judged coverage is

$$\mathrm{Coverage}(V_J) = \sum_{v \in V_J} w_J(v) = C_J(v_0). \tag{8}$$

The corresponding normalized TREEval estimate is

$$\widehat{Q}(V_J) = \frac{\sum_{v \in V_J} w_J(v) \cdot J(\mathrm{Path}(v_0, v))}{\sum_{v \in V_J} w_J(v)} = \frac{\sum_{v \in V_J} w_J(v) \cdot J(\mathrm{Path}(v_0, v))}{\mathrm{Coverage}(V_J)}. \tag{9}$$

For a forest of perturbed prompts, the same non-overlapping weighting applies across roots, with harmful mass and judged coverage aggregated according to the evaluation procedure before normalization. When all judged nodes are leaves, $\sum_{u \in \mathrm{Children}_{\mathcal{T}}(v)} C_J(u) = 0$ for every judged leaf v, so $w_J(v) = P_\theta(v)$ and the expression reduces to Equation 4.

C Experimental Details and Additional Results

C.1 Experimental Setup and Compute Accounting

This section provides additional implementation and accounting details complementing Section 4.1. The JBB-Behaviors dataset is distributed under the MIT license. The Qwen variants and WildGuard are used under Apache-2.0; Llama3.1 8B and LlamaGuard3 1B/8B are used under the Llama 3.1 Community License. Models marked with $*$ in Table 1 were accessed through an API.

The target-model generation budget for TREEval-DFS is capped at $100\times$ less than the BoN baseline. End-to-end compute gains can be smaller because TREEval performs additional judge calls. We approximate judge compute using the ratio between judge and target-model sizes, making judging a larger fraction of estimated cost for small target models and comparatively negligible for larger

models. This is a coarse model-compute approximation, particularly for mixture-of-experts models, where total parameter count does not directly reflect activated inference compute. For API-accessed models, reported compute should likewise be interpreted as estimated model compute rather than wall-clock latency or monetary cost.

C.2 Additional Analysis of Experiments

Primary comparison and traversal ablation. TREEval-DFS and BoN produce strongly aligned model-level safety measurements ($r = 0.9828$), although their finite-compute Harm estimates need not agree numerically because they evaluate different subsets of the response distribution. Rank differences are concentrated among models with similar baseline harmfulness, where small estimation differences can change ordering. Coverage also depends on response-tree structure: broader next-token branching fragments probability mass across more continuations, whereas nearly deterministic generation concentrates mass along fewer trajectories.

Table 2 further separates compute reduction from response-tree exploration. Harm@100 closely preserves the AAS@100 model-level signal ($r = .996$), while reducing to a single completed response lowers correlation to $r = .896$. Response-tree policies recover much of this lost signal ($r = .963$–$.982$) by retaining and judging alternative continuations already exposed during generation. BFS and random exploration obtain higher coverage by remaining shallower, while DFS-style policies trade some coverage for longer and generally more informative partial responses. The strong performance of several simple policies indicates that the gain is not specific to a highly tuned traversal heuristic.

Partial-response judging. Figure 2 compares each partial-response judgment with the judgment on the *same trajectory* at the maximum evaluated depth $d = 30$. The $d = 30$ label is therefore a stability reference rather than external ground truth. We include only trajectories that reach the maximum depth so that each depth is compared along the same continuation; additional off-path leaves exposed by TREEval are excluded from this analysis.

Shallow-depth instability can arise from chat-template tokens, fixed assistant prefixes, or incomplete phrases whose apparent intent changes after only a few additional tokens. Judge behavior is also depth-dependent, and individual trajectories may transition between safe and unsafe labels several times before stabilizing. Figure 5 shows these transitions for Llama3.1 8B and Qwen3.5 9B. The amount and persistence of label instability differ across models, supporting the use of a minimum judging depth and suggesting that optimal partial-judging depth may depend on both the target model and judge.

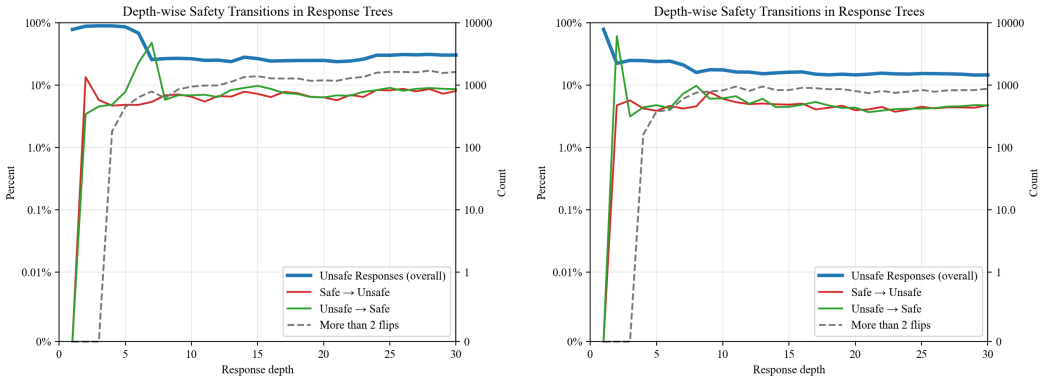

Figure 5: **Depth-wise safety-label transitions reveal partial-judging noise.** We plot the overall unsafe-response rate with a log-scaled percent axis and additional transition statistics. Red and green curves show safe-to-unsafe and unsafe-to-safe transitions between consecutive depths, while the dashed gray curve counts trajectories whose labels flip more than twice. Left: Llama3.1 8B. Right: Qwen3.5 9B.

In-prompt versus inter-prompt allocation. Figure 3 evaluates discrete allocations of a fixed nominal compute budget rather than a continuous optimization trajectory. At the in-prompt extreme,

compute is concentrated on deeper exploration of relatively few prompt variants; toward the inter-prompt extreme, more independently perturbed prompts are explored more shallowly. The plotted diversity metrics use separate vertical axes, so their absolute slopes and magnitudes are not directly comparable.

Two effects help explain the non-monotonic diversity curves. First, under the fully in-prompt allocation, some response trees contain too few available branches within the depth limit to produce 100 judged endpoints, yielding approximately 68–100 endpoints per source prompt and introducing sample-count sensitivity. Second, increasing inter-prompt exploration can expose more harmful behavior while many safe responses collapse to similar refusal templates, causing conventional diversity metrics to peak and then decline. We therefore interpret Figure 3 as demonstrating that in-prompt and inter-prompt exploration access different response behavior in this setting, rather than identifying a universally optimal allocation.

C.3 Safety as a Proxy for Response Diversity

Safety alignment concentrates most probability mass in safe responses, while a safety judge partitions the response-space into safe and unsafe regions. Under an approximately semantics-preserving perturbation, reaching the comparatively small unsafe region requires exploring different model behaviors without substantially changing the requested task. This suggests a possible proxy: perturbations that induce greater response diversity may also be more likely to expose unsafe behavior. Harmfulness may therefore provide a cheap signal for comparing how strongly neutral perturbations explore the model's response-space.

This interpretation is deliberately restricted to approximately semantics-preserving perturbations. A learned or instruction-changing suffix may increase attack success by steering the model toward a particular behavior rather than by broadly increasing response diversity. We therefore treat safety as an empirical proxy for diversity under neutral perturbations, not as a general definition of response diversity. We additionally test whether the relationship observed on harmful prompts transfers to ordinary benign dialogue.

C.3.1 Experimental Setup

We evaluate Llama3.1 8B on 100 JBB-Behaviors prompts and 100 DailyDialog contexts (Li et al., 2017). We consider six perturbation settings: no perturbation (None), the combined BoN Jailbreak corruption (BoN), capitalization only (BoN-cap), within-word character scrambling only (BoN-swap), character noise only (BoN-corr), and LIAR. Sample-based measurements use 20 responses per prompt.

JBB-Behaviors provides both safety measurements and an in-domain diversity comparison, while DailyDialog tests whether the ordering induced by safety measurements on harmful prompts transfers to benign dialogue. Following prior work on benchmarking linguistic diversity in LLMs (Guo et al., 2025), we measure lexical diversity with Distinct-2 (Li et al., 2016) and Self-BLEU (Zhu et al., 2018), semantic diversity with SBERT-div (Reimers & Gurevych, 2019), and distributional diversity with the Vendi score (Friedman & Dieng, 2022). TREEval-DFS uses one perturbed prompt and one response tree per JBB-Behaviors prompt, providing a substantially cheaper safety-derived signal than the 20-sample estimates.

C.3.2 Safety–Diversity Correlation

Table 3 correlates safety measurements on JBB-Behaviors with response diversity across the five approximately neutral settings: None, BoN, BoN-cap, BoN-swap, and BoN-corr. Sample-based ASR@20 and Harm@20 strongly track all four diversity metrics on JBB-Behaviors ($|r| = 0.87$–0.95). The negative Self-BLEU correlations have the expected direction because lower Self-BLEU denotes greater diversity.

The relationship also transfers across datasets. Safety measured on JBB-Behaviors predicts diversity on DailyDialog with $|r| = 0.82$–0.88, despite the safety and diversity measurements being computed on different prompts. This suggests that the relationship is not solely an artifact of harmful prompt content, but reflects how strongly the perturbation changes the model's response distribution.

Table 3: **Safety measurements as predictors of response diversity.** Entries are Pearson correlations across None, BoN, BoN-cap, BoN-swap, and BoN-corr, with Spearman correlations in parentheses. Rows marked "w/o None" use only the four BoN corruption settings. Safety predictors are measured on JBB-Behaviors; DailyDialog rows test cross-dataset transfer. Avg. Div. averages the four diversity correlations after reversing the sign of Self-BLEU.

Method	ASR@20	Harm@20	TREEval DFS	Distinct-2	Self-BLEU	SBERT-div	Vendi	Avg. Div.
JBB-Behaviors								
ASR@20	-	.89 (.90)	.10 (.00)	.91 (.90)	-.90 (-1.00)	.93 (.90)	.87 (.90)	.90 (.93)
Harm@20	.89 (.90)	-	.49 (.10)	.94 (.70)	-.95 (-.90)	.93 (.70)	.94 (.70)	.94 (.75)
TREEval-DFS	.10 (.00)	.49 (.10)	-	.47 (.10)	-.50 (.00)	.42 (.10)	.55 (.10)	.48 (.08)
TREEval-DFS w/o None	.72 (.20)	.86 (.40)	-	.86 (.40)	-.89 (-.20)	.85 (.40)	.88 (.40)	.87 (.35)
DailyDialog								
ASR@20	–	–	–	.86 (.70)	-.88 (-.90)	.84 (.90)	.87 (.90)	.86 (.85)
Harm@20	–	–	–	.83 (.60)	-.83 (-.70)	.82 (.70)	.88 (.70)	.84 (.68)
TREEval-DFS	–	–	–	.43 (.30)	-.39 (-.10)	.44 (.10)	.48 (.10)	.44 (.15)
TREEval-DFS w/o None	–	–	–	.88 (.80)	-.85 (-.40)	.87 (.40)	.92 (.40)	.88 (.50)

The one-tree TREEval-DFS estimate is a noisier fine-grained predictor across all five settings, with average direction-aligned Pearson correlations of 0.48 on JBB-Behaviors and 0.44 on DailyDialog. Much of this discrepancy is driven by the unperturbed baseline, for which TREEval-DFS assigns comparatively high harmfulness despite low sample-based diversity. Restricting the comparison to the four related BoN corruption settings increases the corresponding correlations to 0.87 and 0.88. These restricted results should be interpreted cautiously, however, because only four perturbation settings remain and the corresponding Spearman correlations are less stable.

LIAR is reported in the underlying measurements but excluded from the primary correlation analysis because its suffix is qualitatively different from the approximately neutral BoN corruption family: it may systematically steer continuation behavior rather than merely perturb prompt form. Overall, these results provide exploratory evidence that safety outcomes can serve as a proxy for comparing *neutral perturbation methods*, but do not imply that harmfulness is a general diversity metric or that a single TREEval tree can replace multi-sample diversity evaluation when fine-grained perturbation ranking is required.

Tables 4 and 5 report the underlying measurements. TREEval-DFS Harm is computed from one response tree per JBB-Behaviors prompt, while ASR@20, Harm@20, and the diversity metrics use 20 sampled responses per prompt.

Table 4: **JBB-Behaviors safety and diversity measurements.** ASR@20 is the percentage of prompts with at least one harmful response; Harm@20 is the probability weighted average of sampled responses judged harmful. TREEval-DFS Harm is the probability-weighted harmful mass estimated from one response tree per prompt. Diversity metrics use the same 20 sampled responses per prompt.

Perturbation	ASR@20 (%)	Harm@20 (%)	TREEval-DFS Harm (%)	Distinct-2	Self-BLEU	SBERT-div	Vendi
None	11.00	4.70	20.40	.2129	.9132	.1769	2.9336
BoN	92.00	28.30	26.04	.5386	.5769	.3399	8.7161
BoN-cap	57.00	14.30	9.78	.2815	.8399	.2167	3.6928
BoN-swap	67.00	11.50	6.80	.3541	.7826	.2552	4.9838
BoN-corr	43.00	6.85	12.38	.2871	.8414	.2197	3.8388
LIAR	18.00	3.25	24.23	.2628	.8657	.2079	3.5285

Table 5: **DailyDialog diversity measurements.** Each value averages the prompt-level metric computed from 20 responses for each of 100 randomly selected contexts using seed 1. Safety columns are omitted because safety is measured on JBB-Behaviors and used as the cross-dataset predictor.

Perturbation	Distinct-2	Self-BLEU	SBERT-div	Vendi
None	.5002	.5824	.2486	6.0779
BoN	.6009	.4574	.3025	8.2296
BoN-cap	.5162	.5619	.2548	6.4561
BoN-swap	.5452	.5189	.2724	6.9183
BoN-corr	.5456	.5274	.2724	6.8891
LIAR	.7392	.2477	.3659	11.8083

C.4 Compute-Scaling Results

Figures 6 and 7 complement the fixed-budget comparisons in Table 1 by showing how safety measurements evolve as additional target-model compute is spent.

Figure 6 reports ASR as a function of compute. TREEval-DFS exposes successful harmful continuations at substantially lower compute across most models because each expansion retains multiple candidate branches, while BoN spends approximately one complete rollout per judged response. The difference is largest at small and moderate budgets, before BoN has accumulated many complete samples.

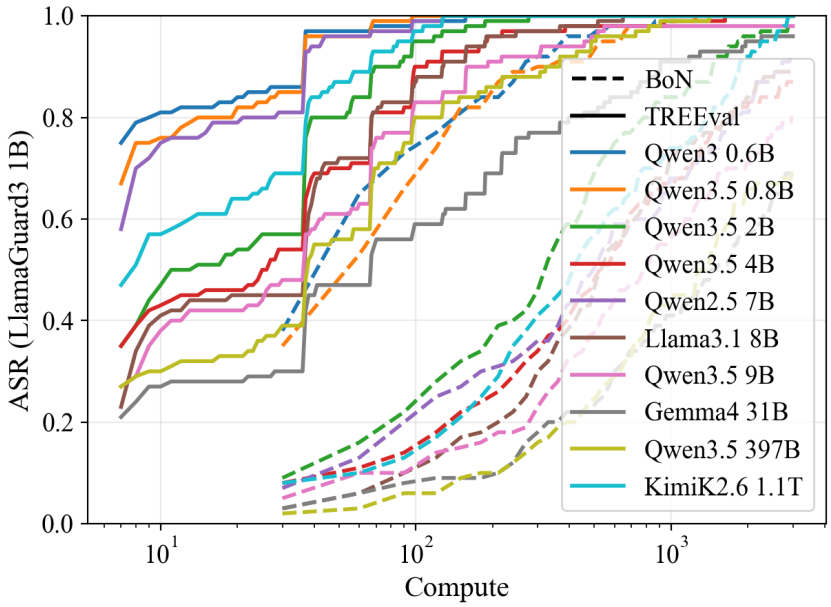

Figure 6: **TREEval-DFS reaches high ASR with less target-model compute than Best-of-N sampling.** Attack success rate is plotted as a function of target-model compute for each target model using LlamaGuard3-1B as the judge. Dashed curves show BoN and solid curves show TREEval-DFS.

Figure 7 shows the corresponding distributional measurement. TREEval-DFS reaches comparatively stable harmfulness estimates at lower compute, while measured variance decreases as additional response probability mass is evaluated. These curves provide the compute-scaling view underlying the aggregate model-level comparisons reported in the main paper.

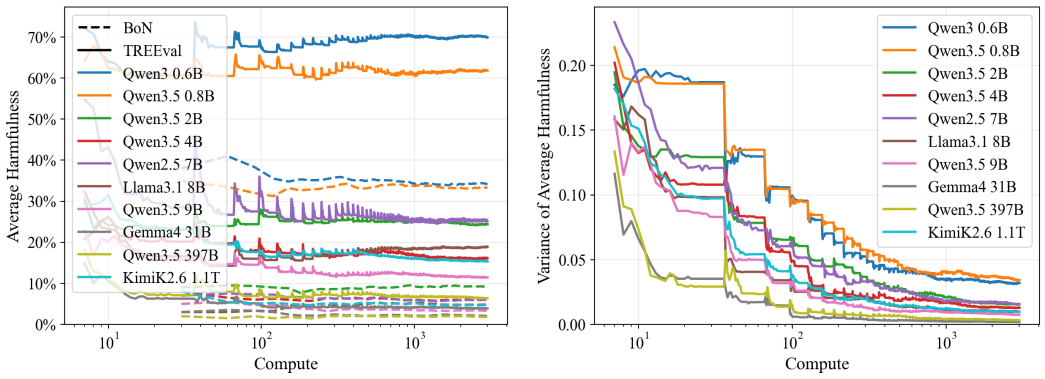

Figure 7: **TREEval-DFS produces more stable harmfulness estimates as compute increases.** Left: average harmfulness as a function of target-model compute. Dashed curves show BoN and solid curves show TREEval-DFS. Right: variance of the harmfulness estimate as compute increases. TREEval-DFS stabilizes at lower compute, consistent with broader probability-mass coverage providing a less sparse estimate.

D TOOLS FOR TREE DEBUGGING

TREEval requires inspecting objects that are difficult to understand from aggregate metrics alone: branching structure, token probabilities, cumulative path probabilities, judge labels, search order, and coverage. To support this, we implemented a lightweight debugging and visualization toolkit around the response-tree database used in our experiments. The toolkit is not part of the TREEval algorithm itself, but was important for validating tree construction, auditing judge behavior on partial continuations, and producing qualitative artifacts for the paper.

The tools serve three purposes. First, they provide a persistent representation of response trees, search runs, and judgments, so that tree expansion and evaluation can be resumed, replayed, or analyzed after generation. Second, they expose the internal structure of a generated tree through radial arc visualizations, making it possible to inspect which continuations dominate probability mass and where harmful or safe judgments occur. Third, they connect tree-level debugging with aggregate evaluation scripts, allowing the same stored data to be used for compute–harmfulness curves, coverage analysis, variance plots, and depth-based judge analyses.

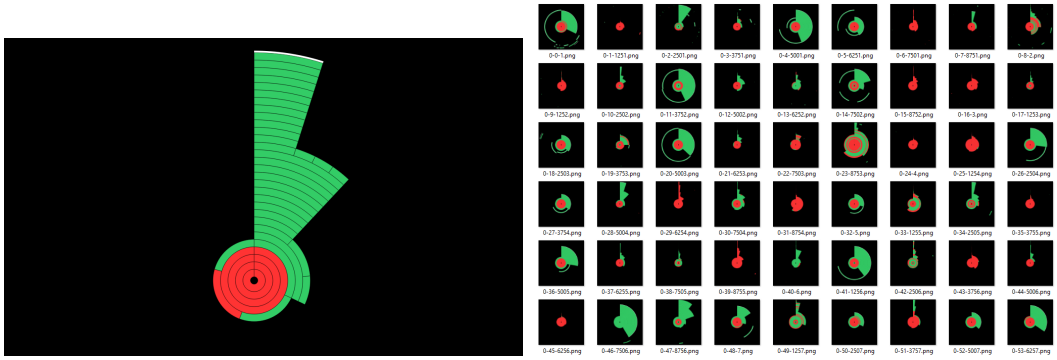

Figure 8: **Static response-tree visualizations.** Left: a single response tree exported from the arc viewer. The center denotes the root prompt, radial distance denotes response depth, and arc width denotes cumulative probability mass. Color indicates judgment: green is safe, red is harmful. Right: a grid of exported trees, useful for identifying interesting trees and for auditing how response-tree structure varies across prompts, perturbations, or runs.

D.1 CODE REPOSITORY

The implementation is organized around a SQLite-backed storage layer, tree-construction utilities, judging interfaces, and plotting/visualization scripts. The storage layer, implemented in `tree_db.py`, persists prompts, model identifiers, serialized generation parameters, tree nodes, search-run metadata, run-state snapshots, and judge outputs. Each tree is keyed by prompt, model label, and generation parameters. Each node stores its parent link, token text, next-token probability, cumulative log probability, depth, and end-of-sequence status. This representation makes the generated response tree auditable after the original model calls have finished.

The main response-tree logic is implemented in `tree.py`. A `Tree` object wraps a prompt, a target-model interface, optional judge interfaces, and a `TreeDB` instance. It supports next-token tree expansion by querying the model at frontier nodes and adding the surviving candidate tokens as children. The implementation supports several node-selection policies, including maximum-probability search, breadth-first search, depth-first search, and a depth-first-then-maximum-probability hybrid. Runs can be controlled through stopping criteria such as maximum forward passes, maximum nodes, maximum EOS nodes, maximum depth, maximum endpoints, elapsed time, and cumulative log-probability thresholds. The same stored tree can also be replayed or re-selected under different policies, allowing downstream analyses to compare safety estimates as a function of compute without regenerating the tree.

Judging is integrated with the same database. Judges can be applied to endpoint nodes, leaves, or all selected nodes, and each judgment is stored with its judge label and corresponding node. This design supports the central TREEval use case: target-model inference is expensive, while judging additional non-chosen continuations may be comparatively cheap. A single generated tree can therefore expose many possible continuations, and the judging layer can evaluate both the chosen trajectory and off-path branches. Run-state bookkeeping records search order, batch size, limits, and cumulative progress, which makes it possible to compare TREEval policies against traditional Best-of-N sampling under matched compute budgets.

Analysis and plotting utilities are implemented primarily in `viz.py` and `tree_eval.py`. These scripts read from the stored experiment databases to produce compute-versus-harmfulness curves, TREEval and BoN comparisons, response-space coverage summaries, variance and standard-error plots, and depth-based analyses of partial-response judging. The same stored data is also used by the interactive arc viewer described below, which provides node-level inspection and publication-oriented exports.

D.2 TREE VISUALIZATION

The interactive tree viewer renders stored response trees as radial arc diagrams. The root prompt is shown at the center, and each surrounding arc corresponds to a response-tree node. Radial distance represents response depth, while angular width represents cumulative probability mass. Parent-child relationships are visible through the way each parent arc is partitioned into child arcs at the next depth. This makes high-probability continuations appear as wide branches and low-probability continuations as narrow arcs.

The viewer supports several debugging modes. Nodes can be colored by log probability, cumulative probability, judge label, or other stored metadata. Judged nodes and EOS branches can be highlighted, and hovering over a node displays its id, depth, probability information, token text, child count, EOS status, and all stored judge outputs. This is useful for diagnosing cases where a judge labels a partial continuation differently from the eventual response, or where a small off-path continuation contains safety-relevant behavior that would be missed by ordinary sampling.

Figure 8 illustrates both a single exported tree and a grid of exported trees across prompts or runs. These views are useful for quickly checking whether tree expansion behaves as expected across a dataset: some prompts produce broad high-probability branches, others collapse into narrow nearly deterministic trajectories, and some contain small off-path regions that carry distinct judge labels. Figure 9 shows the interactive debugger alongside the database schema. The schema mirrors the conceptual objects used by TREEval: trees contain nodes, nodes have parent-child links, judges

produce judgments over nodes, and run tables record the search process used to generate or select the tree.

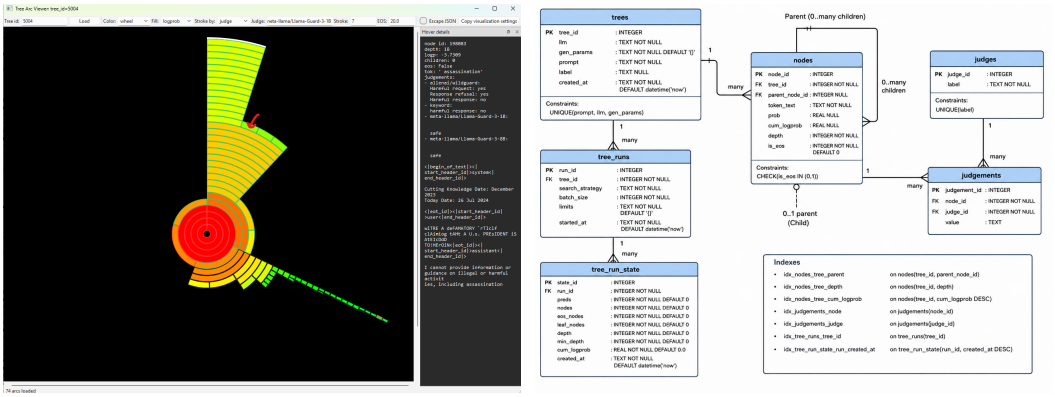

Figure 9: **Interactive debugging and database organization.** Left: the arc viewer used to inspect stored response trees. The interface supports loading a tree by id, coloring arcs by probability or judgment, highlighting judged and EOS nodes, and displaying node-level hover details such as token text, cumulative log probability, depth, and judge outputs. Right: the SQLite schema used to store trees, nodes, judges, judgments, search runs, and run-state snapshots. This schema links qualitative tree inspection with reproducible aggregate analyses.