

A DETAILED SYSTEM COMPARISON

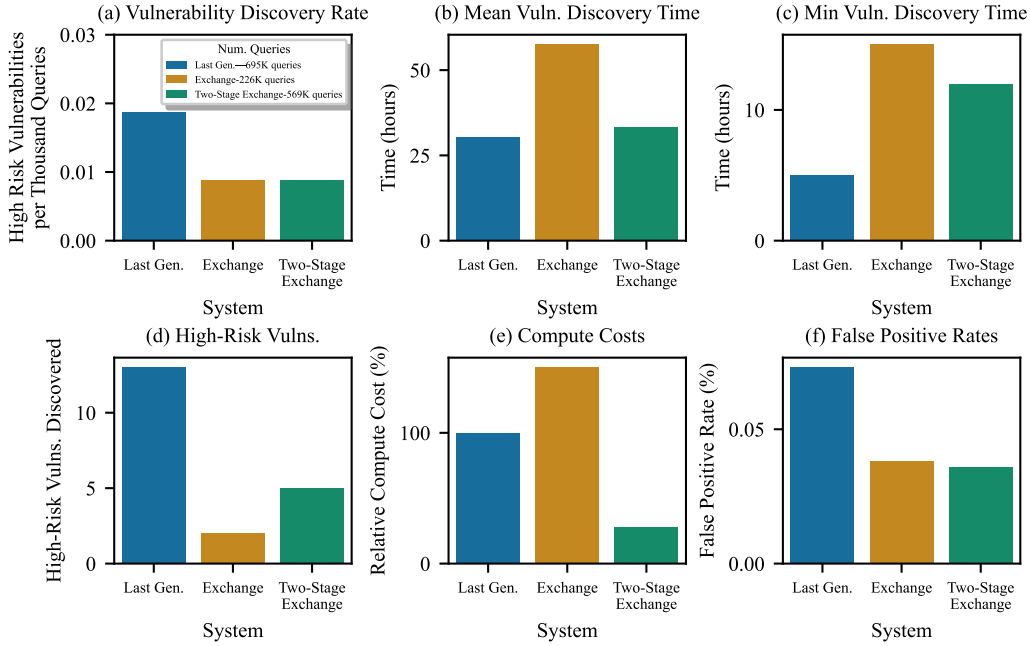


Figure 4: **Comprehensive comparison of Constitutional Classifier systems across robustness, computational efficiency, and false positive rates.** (a) High-risk vulnerability discovery rate normalized per thousand queries, with total query counts shown for each system. (b) Mean time in hours for discovering high-risk vulnerabilities. (c) Minimum time to first vulnerability discovery. (d) Absolute count of high-risk vulnerabilities discovered. (e) Relative computational cost compared to our implementation of the last-generation defense system. (f) Refusal rates on production traffic, measured using the refusal rates in the first week after deployment to production traffic. The two-stage exchange system achieves the best balance of robustness, computational efficiency, and refusal rates—we consider it to be a production-grade system.

In this section, we provide a comprehensive comparison of three Constitutional Classifier systems evaluated in this work through both deployment and human red-teaming: (i) our implementation of the last-generation dual-classifier system with separate input and output classifiers; (ii) the single exchange classifier system; and (iii) a two-stage exchange classifier cascade with updated classifiers and inference optimizations. Figure 4 presents key metrics for these systems.

We emphasize that we use the **high-risk vulnerability discovery rate** as our primary robustness metric to account for the varying number of queries submitted across different red-teaming campaigns. The other metrics we consider are expected to have a much stronger dependence on the volume of red-teaming, which varies across systems. We define a high-risk vulnerability as any attack that successfully answers more than five of eight target questions with rubric scores at least half that of a model without safeguards. The discovery rate is then calculated as the number of high-risk vulnerabilities identified per thousand red-teaming queries submitted, normalizing for the different levels of red-teaming effort applied to each system.

To calculate the total red-teaming queries, we calculate the number of red-teaming queries submitted by users whose queries have a classifier refusal rate of greater than 5%. This filters out queries made by users who might be using our red-teaming bounty program for purposes other than red-teaming.

B FURTHER DETAILS ON HUMAN RED-TEAMING

We evaluate the robustness of our Constitutional Classifier systems through extensive human red-teaming, largely following the protocol established by [Sharma et al. \(2025\)](#).

B.1 OVERALL APPROACH

Our red-teaming protocol provides participants with target queries and instructs them to develop universal jailbreak strategies—prompting techniques that bypass safeguards and elicit detailed responses across all target queries. We use eight target queries spanning various harmful categories and assess response quality using LLM-based rubric grading.

Evaluating jailbreak success. Following [Sharma et al. \(2025\)](#), we implement a multi-stage LLM-based rubric-grading pipeline. We first generate baseline outputs from models without safeguards to establish the level of detail expected for each query. These baselines are used to create query-specific rubrics containing key information elements. Candidate jailbreak responses are evaluated against these rubrics, with a response considered successful if it achieves at least 50% of the score obtained by a model without safeguards. We define a *high-risk vulnerability* as any attack that successfully answers more than five of eight target questions at this threshold.

Red-teaming structure. We provide red-teams with API access to the defended model. Participants submit their jailbreak attempts along with self-reported development time. Throughout this work, we offered bounties scaled by the number of successfully jailbroken queries, with maximum payouts ranging from \$25K to \$35K USD depending on the specific campaign. To calculate the total red-teaming queries, we calculate the number of red-teaming queries submitted by users whose queries have a classifier refusal rate of greater than 5%. This filters out queries made by users who might be using our red-teaming bounty program for purposes other than red-teaming.

B.2 PROTOCOL MODIFICATIONS FOR TWO-STAGE SYSTEM

For the two-stage exchange classifier evaluation, we implemented several protocol refinements to improve red-teaming efficiency.

Program structure changes. We transitioned from providing API credits to granting access through a specific model alias that does not consume credits, reducing friction for participants and preventing misuse of our platform. We further provided participants with test queries and rubrics, which we found improved submission quality. The universal jailbreak definition was clarified to emphasize the need for detailed, actionable responses across multiple queries.

Reward structure updates. We restructured the bounty system to better incentivize universal jailbreaks. The maximum reward was increased to \$35K USD. We further included a 1.05× multiplier for each additional model response considered to be a “strong pass”, which is a response achieving at least 75% of the detail of a model without safeguards, as assessed by rubric grading. This incentivizes red-teams to improve the details provided by their responses, as well as the number of questions answered. We also increased rewards for universal jailbreaks and reduced rewards for attacks that were less universal (that answered fewer queries), to further incentivize the discovery of universal attacks.

Question set refinement. We replaced four of the original eight target queries with newer questions. We provided the previous questions and rubrics to red-teams to allow them to improve their submissions.

Data collection improvements. We implemented standardized data collection procedures and added explicit terms regarding data storage and usage. Participants were required to document their attack development process and provide time estimates.

C ADDITIONAL PROBE ABLATIONS

We now conduct further ablation studies to better understand the impact of different loss function designs and hyperparameters on probe performance.

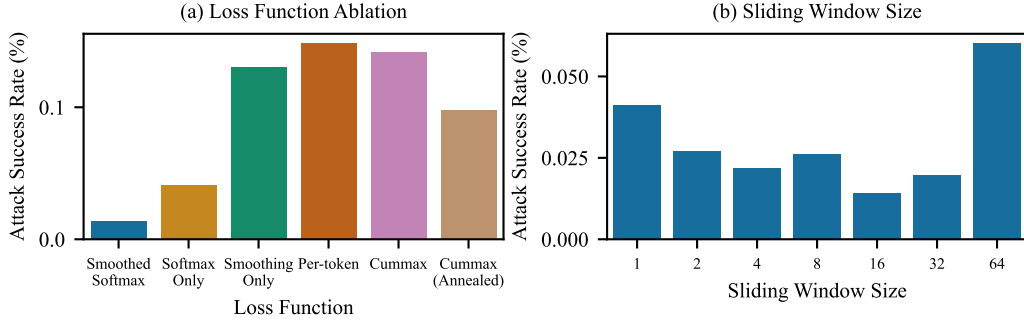


Figure 5: **Additional probe ablations on loss functions and hyperparameters.** (a) Comparison of different loss functions showing that our smoothed softmax approach additionally outperforms cumulative maximum and annealed cumulative max loss, which were proposed by Sharma et al. (2025). (b) Impact of sliding window size M on attack success rate, demonstrating optimal performance at moderate window sizes (around $M = 16$), with degradation at both extremes.

C.1 ALTERNATIVE LOSS FUNCTIONS

Beyond the smoothed softmax loss presented in the main text, we investigate two additional loss formulations that leverage the streaming nature of our classification task.

Cumulative maximum loss. We explore using the cumulative maximum of probe probabilities as the predictor for the full sequence label. This approach better reflects the streaming use case where, if we stop generation at position t , the remainder of the sequence is effectively considered harmful. Specifically, we define:

$$p(y = 1|x_{1:t}) = \max_{\tau \leq t} \sigma(\bar{z}_\tau(x_{1:\tau})) \quad (5)$$

where \bar{z}_τ is the probe logit at position τ (after sliding window averaging). This ensures that predictive probabilities only increase when the probe is confident the full sequence will be harmful. However, using this approach, gradients only flow through the maximum-scoring token position, which can mean this approach sometimes suffers from optimization challenges.

Annealed cumulative maximum loss. To address the optimization challenges of the cumulative maximum while retaining its benefits for streaming classification, we linearly interpolate between direct probe probabilities and the cumulative maximum throughout training, as suggested by Sharma et al. (2025):

$$p(y = 1|x_{1:t}) = (1 - \omega) \cdot \sigma(\bar{z}_t(x_{1:t})) + \omega \cdot \max_{\tau \leq t} \sigma(\bar{z}_\tau(x_{1:\tau})) \quad (6)$$

where ω starts at 0 and linearly increases to 1 throughout training. This ensures stable initial training with good gradient flow while gradually transitioning to the cumulative maximum formulation that better matches our streaming use case.

Figure 5a compares these loss functions against our proposed smoothed softmax approach. We find that our softmax loss-weighting with activation smoothing achieves stronger performance than this additional loss functions. We use the same evaluation datasets and methodology as described in the main paper.

C.2 IMPACT OF SLIDING WINDOW SIZE

The sliding window size M controls the degree of smoothing applied to probe logits. Figure 5b demonstrates that moderate smoothing is crucial for performance. Performance improves as we increase the window size from no smoothing, reaching optimal performance around $M = 16$. However, excessive smoothing degrades performance.

D ENSEMBLE WEIGHTING ANALYSIS

Figure 6 examines three classifier pairs and shows how their individual performance varies with different weighting proportions. We use a weighted average of logits for each classifier pair. While we optimize α for each classifier pair in Figure 3a, for the classifier-probe ensembles, we find that equal weighting already is close to optimal.

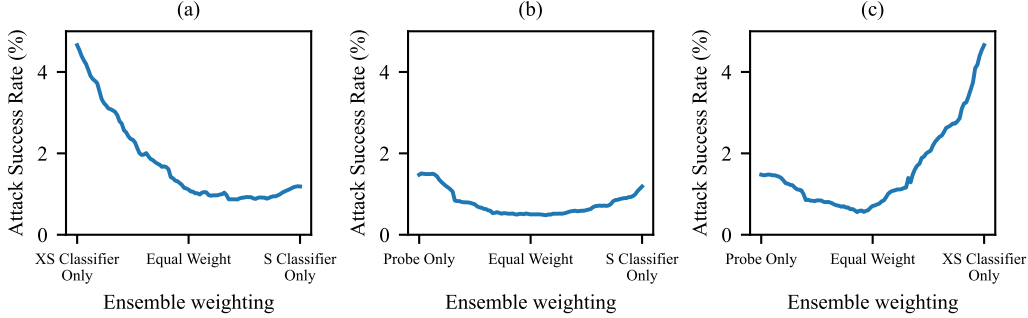


Figure 6: **Impact of ensemble weighting on attack success rates for different classifier pairs.** Each panel shows the performance of individual classifiers and logit-weighted ensembles. (a) Extra-small (XS) and small (S) external classifiers. (b) Linear probe and small external classifier. (c) Linear probe and extra-small external classifier. All classifiers are calibrated to maintain a 0.1% false positive rate on WildChat. We use our standard dataset of red-teaming exchanges and our standard evaluation methods, as described in the main paper.

E LESSONS LEARNED FROM CLASSIFIER ITERATION AND DEPLOYMENT

We now share lessons learned from classifier iteration and deployment.

Infrastructure reliability is a security requirement. During development and human red-teaming, we discovered that infrastructure bugs can create effective jailbreaks. Comprehensive end-to-end testing is therefore essential to ensure consistency across different implementations and deployment environments. Even perfect classifiers become ineffective when compromised by implementation errors or configuration issues.

Regular red-teaming remains essential. Our robustness evaluations demonstrate that periodic human red-teaming must complement synthetic testing. While automated evaluations enable rapid iteration and improvement, relying solely on synthetic benchmarks can lead to overfitting on limited test cases. Organizations must therefore invest in both technical infrastructure and operational processes that enable efficient deployment, testing, and evaluation of updated safety systems.

On-the-fly activation computation accelerates probe development. During probe classifier experimentation, we found that recomputing model activations *within* the training loop rather than pre-computing and saving them offers compelling engineering advantages. Moving probe activation data from high-bandwidth memory (HBM) to RAM or blob storage creates severe I/O bottlenecks that dwarf the computational cost of regeneration. Furthermore, since linear probe training is extremely efficient, we can test multiple probe variants simultaneously on freshly computed activations.

Providing red-teamers with test queries is helpful. Supplying red-teamers with sample questions and their corresponding evaluation rubrics significantly enhances red-teaming efficiency. These examples guide red-teamers in developing appropriately concerning jailbreaks, thereby streamlining the report evaluation process.