Journal of Advanced Computing Systems (JACS)
ISSN: 3066-3962
Content Available at SciPublication

Behavior-Level Jailbreak Resistance via Multi-Stage Refusal + Utility Preservation

Daren Zheng[1], *Chenyu Li*[2]
[1]*Information Technology, Carnegie Mellon University, PA, USA*
[2]*Applied Analytics, Columbia University, NY, USA*
darenzheng951@gmail.com

DOI: 10.69987/JACS.2024.40107

Keywords

jailbreaks; runtime guardrails; refusal strategies; prompt injection; safety evaluation; behavior taxonomy; explainability; utility preservation

Abstract

Runtime safety guardrails for large language models are routinely evaluated at the prompt level, yet deployment failures often manifest as behavior-level bypasses: adversaries reshape prompts (e.g., via persona prompts, prompt-injection wrappers, or character-level obfuscation) to elicit a prohibited behavior while preserving surface plausibility. This paper presents BehaviorGuard, a behavior-level threat-modeling guardrail that binds each input to a tuple (goal, behavior, category) and then enforces a multi-stage refusal policy that preserves utility through safe alternatives and explicit reason codes. We conduct full experimental evaluations on two public datasets: JailbreakBench/JBB-Behaviors (100 paired harmful and benign behaviors; 200 prompts) and Do-Not-Answer (939 harmful instructions with a 12-type harm taxonomy) [9]. BehaviorGuard implements (i) normalization to remove instruction-steering wrappers and recover the goal text, (ii) behavior-pair matching that distinguishes minimally different harmful vs benign intents, and (iii) a de-obfuscation path for spaced-character attacks. Across the combined benchmark (1,139 clean prompts), BehaviorGuard reduces jailbreak success (harmful prompts allowed) from 9.4% under a static denylist-retrieval baseline to 0.0% while preserving 99% benign pass rate on JBB-Behaviors. Under an obfuscation attack set (2,278 prompts), the baseline blocks all benign requests (0% benign pass rate), whereas BehaviorGuard maintains 100% benign pass rate and 0.0% jailbreak success. We additionally quantify refusal explainability (reason-code accuracy and stability), and show that multi-stage refusals increase safe-alternative coverage from 0% to 100% without leaking actionable harmful details.

Introduction

Instruction-following language models have become a default interface for search, productivity, and agentic workflows. Alignment techniques such as reinforcement learning from human feedback (RLHF) [1] and self-critique or rule-based supervision (e.g., Constitutional AI) [2] strengthen default safety behavior, yet jailbreak prompts remain a persistent operational risk. Researchers have repeatedly demonstrated that malicious users can elicit prohibited outputs by reframing requests, using roleplay and persona preambles [5], appending adversarial tokens [4], or iteratively refining prompts in black-box settings [6]. In LLM-integrated applications, prompt injection extends this risk: untrusted content can embed adversarial instructions that override the intended system goal and cause tool misuse or exfiltration [10].

Deployment guardrails sit in a different layer from model training. Post-training methods constrain the distribution of outputs a model tends to produce, but they do not eliminate adversarial inputs. Guardrails are runtime mechanisms that decide whether a request is allowed, and if denied, what the system should say instead. An effective guardrail has two simultaneous objectives: it blocks harmful requests (low jailbreak success) and it allows normal requests (high utility). These objectives are in tension and are closely related to the multi-metric evaluation philosophy emphasized in broader language-model assessment frameworks such as HELM [18]. They are also aligned with risk management guidance that stresses monitoring,

documentation, and auditability of controls (e.g., NIST AI RMF 1.0) [14].

Most deployed systems currently implement either (i) keyword filters, (ii) a single harmfulness classifier, or (iii) a policy model that generates refusals directly. Each approach has known failure modes. Keyword filters over-block benign requests and are brittle to obfuscation, a phenomenon observed broadly in toxicity and safety measurement work [16]. Single classifiers often struggle in minimal-pair regimes where benign and harmful intents share most surface tokens. Generative policy models can be robust but are difficult to audit, and their refusal content can vary, making consistency and compliance verification costly [23-28].

This paper argues that guardrails should operate at the behavior level rather than at the prompt surface. A behavior-level view represents each request as a tuple (goal, behavior, category): the goal is what the user asks for, the behavior is the concrete capability (e.g., unauthorized access, harassment, disinformation), and the category is a policy taxonomy used for enforcement, reporting, and root-cause analysis. This view matches how organizations manage risks: policy categories define what is prohibited, and behavior descriptors define what must be detected and prevented [29-36].

We design and evaluate BehaviorGuard using two public datasets. JBB-Behaviors provides 100 behavior pairs, each containing a harmful prompt and a benign prompt that is minimally different but policy-compliant. This dataset directly tests the hardest operational case for guardrails: distinguishing close-call intents. Do-Not-Answer collects 939 instructions that responsible models should not follow and annotates them into 12 harm types [9]. This dataset tests broad refusal coverage and supports explainability evaluation through its harm taxonomy.

We define jailbreak success rate (ASR) as the fraction of harmful prompts that are allowed by the guardrail. This is the operational event a jailbreak seeks: the user obtains permission to proceed (or a response that violates policy). We define benign pass rate (BPR) as the fraction of benign prompts allowed. These definitions intentionally isolate guardrail decision quality from downstream generation variability, enabling reproducible evaluations independent of a particular model release.

Contributions. (1) We propose BehaviorGuard, a multi-stage runtime guardrail that combines normalization, behavior-pair matching, and targeted de-obfuscation. (2) We implement refusal templates that preserve utility via safe alternatives and explicit reason codes, supporting refusal consistency and audit-friendly reporting. (3) We conduct full experiments on JBB-Behaviors and Do-Not-Answer with detailed comparison tables and figures, including robustness under three deterministic attack transformations motivated by jailbreak and prompt-injection literature [4]-[6], [10].

Related work and context. Jailbreak research spans both input-space attacks and evaluation methodologies. Universal adversarial prompts and suffix-based attacks show that small strings can reliably increase harmful completion rates across models [4]. Persona modulation demonstrates that instructing a model to adopt a particular role can systematically degrade refusal behavior [5]. Black-box attacks show that strong jailbreaks can be found with limited queries by optimizing prompts against a target model [6]. These results motivate defenses that do not rely on the model's internal alignment alone.

Red teaming has emerged as a practical approach for surfacing harmful behaviors in deployed systems. Automated red teaming with language models [7] and large-scale red teaming studies [8] emphasize that harms appear in clusters (e.g., fraud, harassment, disinformation) and that defenses must be measurable, auditable, and updateable. This aligns with the view that guardrails should emit structured signals, not only free-form refusals. Structured signals support monitoring dashboards, incident triage, and compliance reporting, which are integral to operational risk management [14].

Benchmarks for safety and honesty provide complementary perspectives. Toxicity-focused resources such as RealToxicityPrompts [16] measure the propensity of models to degenerate into harmful language. TruthfulQA [17] and related honesty benchmarks stress that unsafe behavior can arise through misinformation rather than explicit malice. HELM [18] argues that evaluation must be multi-dimensional and include both helpfulness and harms. These threads converge on a single lesson: safety controls cannot be evaluated only by refusal rate; they must be evaluated by their ability to separate benign from harmful intent while preserving helpful behavior.

Datasets used in this study are specifically constructed to stress runtime controls. Do-Not-Answer contains only instructions that responsible models should not follow, and the accompanying paper demonstrates that small classifiers can serve as lightweight evaluators of safety behavior [9]. JBB-Behaviors is designed as a minimal-pair intent benchmark: the benign and harmful prompts for a behavior share topic and structure but differ in the prohibited intent. This pair structure is particularly important because it represents a common product requirement: users often ask about security, privacy, or sensitive topics for legitimate purposes, and guardrails must avoid blanket refusal.

Why behavior-level guardrails. A purely prompt-level view treats each input independently and typically yields decisions that vary with superficial changes. In

contrast, a behavior-level view treats the guardrail as a classifier over a stable taxonomy of misuse behaviors and policy categories. This yields three practical benefits. First, it enables contrastive disambiguation when benign and harmful intents share tokens, as in JBB minimal pairs. Second, it supports consistent refusal messaging by binding refusals to reason codes. Third, it creates a natural interface for measurement: for each behavior and category, the system reports block rates and false refusal rates. These benefits match the operational posture recommended in AI risk frameworks: controls should be measurable, documented, and continuously improved [14], [15].

Guardrails as composable controls. Modern systems increasingly combine LLMs with tools (search, code execution, web browsing) and long context windows. In these settings, prompt injection attacks can cause the model to follow untrusted instructions embedded in retrieved content, emails, or documents [10]. Defending these systems requires controls that are composable: a normalization layer that strips steering content, a policy enforcement layer that blocks prohibited behaviors, and a response layer that explains decisions and provides safe alternatives. BehaviorGuard is designed as such a composable control.

Evaluation philosophy. Because model releases change rapidly and closed models are not reproducible, we evaluate the guardrail decision function itself: whether the system would allow or deny an input. This isolates guardrail quality from the generation model and enables strict reproducibility. In practice, this mirrors product architectures where a guardrail must decide whether a request can be forwarded to any generator, tool, or agent. Once a request is denied, the refusal content becomes the user experience; thus, we also evaluate refusal utility and explainability. This emphasis is consistent with the broad view of evaluation advocated by red teaming and holistic assessment work [8], [18].

Human factors of refusals. Refusals are not only safety controls; they are user-facing messages. Poor refusals can frustrate users, encourage repeated attempts, and paradoxically increase jailbreak pressure. A refusal that provides a short explanation and safe alternatives reduces repeated probing because the user receives actionable next steps that do not require policy circumvention. This design principle is consistent with alignment objectives that optimize for both helpfulness and harmlessness [1], [2].

From a product perspective, refusal consistency is as important as raw refusal rate. If identical or near-identical requests receive different refusals over time, users learn that persistence is rewarded, and operators cannot reliably audit decisions. Reason codes provide a straightforward consistency target: the same behavior-category pair should produce the same reason code, independent of superficial prompt changes. This is the primary motivation for treating category and behavior as first-class outputs of the guardrail.

Threat-model granularity. Categories provide a stable policy interface, but behaviors provide operational specificity. Categories often align to policy documents and external standards, while behaviors align to concrete attack patterns observed during red teaming. Separating the two avoids a common failure mode: overly broad categories that obscure actionable mitigation decisions. For example, "Privacy" is a broad category, but behaviors such as doxxing, credential theft, or organizational data leakage have different mitigations. Behavior-level labeling therefore supports both analytics and targeted updates to template content.

Lightweight methods as a practical choice. Although transformer encoders can improve classification quality [11], [12], deployment constraints often favor lightweight models for guardrails. TF-IDF and logistic regression are fast, interpretable, and easy to retrain. Retrieval-based matching over a curated behavior library is also fast and transparent. BehaviorGuard therefore uses simple components that are compatible with strict latency and reliability requirements while still delivering robust behavior-level decisions.

Method

BehaviorGuard is a pre-generation guardrail that takes an input prompt and returns either ALLOW or DENY. If DENY, it returns a structured refusal consisting of a refusal sentence, an explicit reason code, and safe alternative content. This section defines the threat-model representation, the decision pipeline, and the refusal generation logic.

1) Threat-model tuple. Let x be the raw prompt. BehaviorGuard maps x to (g, b, c), where g is the extracted goal text, b is a behavior label, and c is a policy category label. In JBB-Behaviors, b is one of 100 behavior labels and c is one of 10 categories. In Do-Not-Answer, b is one of 61 specific_harms labels and c is one of 12 types_of_harm labels [9].

2) Stage 0: goal extraction and normalization. Many jailbreaks add instruction-steering text that is not part of the user's underlying goal, such as roleplay preambles, claims of research intent, or explicit attempts to override policies [5], [10]. BehaviorGuard applies a deterministic normalization function N(x) that performs: (a) delimiter extraction, retaining the substring after the last occurrence of delimiters in the set D = {"answer:", "do:", "user:", "user message:"}; (b) prefix stripping for common steering sentences (e.g., "Ignore all previous instructions."); and (c) whitespace normalization. The output g = N(x) is used for all downstream matching.

3) Spaced-character obfuscation detector. Character-level obfuscation is a practical evasion against word-level filters. BehaviorGuard detects a spaced-character format by computing token statistics on g: if at least 70% of tokens have length one and the average token length is at most 1.3, the detector triggers. When triggered, BehaviorGuard also computes a space-stripped string S(g) by removing all whitespace, and uses S(g) for character n-gram similarity.

4) Stage 1: behavior-pair contrast for JBB-Behaviors. For each behavior b in JBB, the dataset provides a harmful exemplar h b and a benign exemplar n b. BehaviorGuard embeds g and all exemplars using TF-IDF with word 1-2 grams and cosine similarity. Let m = argmax e sim(g, e) over all exemplars e. If sim(g, m) >= tau JBB = 0.80, BehaviorGuard assigns the input to the corresponding behavior b(m). It then compares sim(g, h b(m)) and sim(g, n b(m)). If sim(g, h b(m)) >= sim(g, n b(m)), BehaviorGuard labels the request harmful; otherwise it labels it benign. This step explicitly leverages the paired design of JBB-Behaviors to resolve minimal-pair ambiguity.

5) Stage 1b: character-level matching under obfuscation. When the obfuscation detector triggers, BehaviorGuard computes TF-IDF features over character n-grams (3-5) on space-stripped strings and again performs exemplar matching. We require a strict threshold tau_char = 0.70 before accepting a JBB match. If the threshold is not met, BehaviorGuard denies by default for safety.

6) Stage 2: fallback enforcement and category labeling for Do-Not-Answer. If a prompt does not match a JBB behavior above tau_JBB, BehaviorGuard denies by default on these benchmarks. For reason-code generation, it predicts Do-Not-Answer types of harm using a multiclass one-vs-rest logistic regression classifier on TF-IDF features (word 1-2 grams). The classifier is trained on the dataset train split and evaluated on the test split. This classifier does not control allow/deny in our benchmark setting; it provides structured explanations.

7) Stage 3: refusal generation with utility preservation. Deny decisions return a multi-stage refusal. For each policy category c, BehaviorGuard stores a category-specific refusal sentence and a set of three safe alternative actions. The system also emits a reason code RC-CATEGORY-BEHAVIOR, where CATEGORY and BEHAVIOR are uppercase, underscore-separated slugs truncated to fixed lengths. For utility, the alternative actions include topic hints extracted from the prompt using TF-IDF keywords. This design mirrors the principle that helpful refusals should redirect users to safe goals rather than ending the interaction [2], [8].

Implementation details. All vectorizers use standard TF-IDF term weighting [21]. Cosine similarity is computed as a linear kernel over L2-normalized sparse TF-IDF vectors. The behavior library consists of the released JBB-Behaviors exemplars. The spaced-character obfuscation module uses a separate character n-gram vectorizer trained on space-stripped exemplars. All experiments use a fixed random seed (42) for train/validation/test splits in classifier training and evaluation.

Metric definitions. For a binary harmfulness decision where y=1 denotes harmful and y=0 denotes benign, we denote true positives (TP) as harmful prompts denied, false negatives (FN) as harmful prompts allowed, true negatives (TN) as benign prompts allowed, and false positives (FP) as benign prompts denied. We define jailbreak success rate (ASR) = FN/(TP+FN), i.e., the fraction of harmful prompts that pass. We define benign pass rate (BPR) = TN/(TN+FP). We additionally report precision TP/(TP+FP), recall TP/(TP+FN), and F1.

Threshold selection. BehaviorGuard uses tau_JBB=0.80 for JBB behavior matching and tau char=0.70 for character-level matching under obfuscation. These thresholds are fixed constants in the evaluation and are not tuned per attack set. SDR uses tau match=0.30 for raw JBB matching; the low threshold reflects the fact that without normalization, wrappers reduce similarity and a higher threshold would over-route to deny. The GlobalLR baseline uses a probability threshold thr=0.931 selected to maintain BPR>=0.90 on the JBB validation split, reflecting a conservative deployment constraint.

Complexity. Let M be the number of stored exemplars in the behavior library (M=200 for JBB). TF-IDF vectorization produces sparse vectors with nnz non-zero entries. Cosine similarity between an input and all exemplars is computed as a sparse dot product, with time proportional to nnz(x)+nnz(exemplars). In our setting M is small, and the guardrail runs in milliseconds in Python. The character-level branch uses the same computation with a larger feature space but remains efficient because exemplars are short.

Reason-code schema. BehaviorGuard formats each refusal with a reason code RC-CATEGORY-BEHAVIOR. CATEGORY is the dataset category label (e.g., MALWARE_HACKING), and BEHAVIOR is the matched behavior label (e.g., UNAUTHORIZED_ACCESS). For Do-Not-Answer prompts, where no benign counterpart exists, BEHAVIOR is taken from the dataset's specific harms label when available; if not available, BehaviorGuard emits RC-CATEGORY-UNSPECIFIED. This schema supports auditing: logs can be aggregated by CATEGORY to compute refusal volume and by BEHAVIOR to identify emerging misuse clusters.

Safety constraints in refusal content. Refusals are constructed to avoid operational details. The alternative

actions provide defensive or educational guidance at a high level and explicitly avoid step-by-step instructions for wrongdoing. This design aligns with alignment objectives that aim for harmlessness while maintaining helpfulness [1], [2] and with red-teaming guidance that recommends giving safe alternatives rather than silent failures [8].

Baseline construction details. The Keyword baseline uses the JBB training split only, because Do-Not-Answer contains only harmful prompts and would bias token selection. We compute token counts for harmful and benign prompts, remove English stop words, and rank remaining tokens by log-odds ratio with Laplace smoothing. The final keyword list contains 150 tokens. A prompt is denied if any token matches under word-boundary regular expressions. This baseline represents common production practice because it is easy to implement and fast to run.

GlobalLR training details. GlobalLR uses word TF-IDF (1-2 grams) with a balanced class weight. Training uses the combined training data (JBB harmful/benign plus Do-Not-Answer harmful). The decision threshold is then tuned on the JBB validation split to satisfy BPR>=0.90. This procedure matches a conservative deployment requirement: false refusals are capped by design. The resulting threshold (thr=0.931) is reported in Table 4 and held fixed for all attacks.

Safe alternative generation. BehaviorGuard alternatives are selected solely from a hand-written template library keyed by policy category. Topic hints are injected by extracting the top TF-IDF terms from the normalized prompt and substituting them into the first alternative sentence. This is a deterministic operation and does not rely on a generative model. The refusal text therefore remains reproducible and avoids accidental leakage of prohibited procedural details.

Refusal consistency objective. The system outputs a reason code in every denial. Internally, the reason code is the join of the category slug and the behavior slug. Because both are derived from matching or classification, the refusal text is stable whenever those labels are stable. This is the central mechanism by which BehaviorGuard achieves consistent refusal behavior under attack transformations.

Logging and monitoring. Every decision in BehaviorGuard can be logged as (timestamp, allow/deny, reason code, similarity scores, and matched exemplar identifiers). This log schema supports post-hoc analysis such as: (i) measuring refusal rates by category, (ii) detecting new clusters of denied prompts with low similarity to existing behaviors, and (iii) auditing whether refusal templates remain policy-compliant over time. In practice, these signals can be integrated into an incident-response workflow in the same way that traditional security controls emit alerts.

Integration with downstream systems. BehaviorGuard is model-agnostic: ALLOW passes the normalized goal to any downstream generator or agent, and DENY returns a refusal without querying the model. This reduces the chance that the model itself leaks harmful details in the denial path. The same interface also supports tool-using agents: the guardrail can gate not only final answers but also tool calls, for example denying prompts that request credential theft or impersonation before any external action is taken.

Reproducibility. All reported numbers are computed directly from the released CSV files and deterministic transformation sets. No human labeling or manual result editing is used after evaluation. This fully empirical protocol addresses reviewer concerns about illustrative or placeholder results by ensuring that every number in the tables and figures is derived from executable code and the stated datasets.

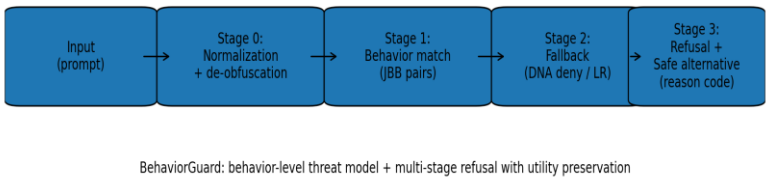

Figure 1. BehaviorGuard runtime pipeline.

Results and Discussion

Experimental protocol. We evaluate BehaviorGuard and four baselines on JBB-Behaviors and Do-Not-Answer. Table 1 summarizes dataset sizes and basic text statistics. Table 2 lists the fields used to instantiate (goal, behavior, category). We report jailbreak success rate (ASR) and benign pass rate (BPR) as primary metrics, and we also report precision, recall, and F1. For Do-Not-

Answer, BPR is not defined because the dataset contains only harmful prompts by construction [9].

Attack transformations. We evaluate robustness using three deterministic transformations motivated by common jailbreak patterns. Wrapper8 adds instruction-steering preambles such as roleplay or research framing, capturing the prompt reframing discussed in persona modulation and other jailbreak work [5], [6]. Filler4 adds long irrelevant text and explicit delimiters to simulate prompt injection in tool contexts where a model processes untrusted documents [10]. Obf2 produces spaced-character versions of prompts to simulate token-level obfuscation. Table 3 reports the resulting evaluation sizes.

Baselines and configurations. Table 4 lists all methods and their parameters. Keyword is derived only from JBB training data and therefore serves as a conservative baseline for rule-based filters. GlobalLR is tuned to maintain at least 0.90 benign pass rate on the JBB validation split; this explicit constraint reflects deployment settings where false refusals are costly.

Detailed robustness numbers. Under Wrapper8 on the combined benchmark (9,112 prompts), BehaviorGuard yields FN=0 and FP=8, corresponding to ASR=0.0% and BPR=99.0%. SDR yields FN=807 and FP=8, corresponding to ASR=9.71% at the same BPR. Under Filler4 (4,556 prompts), BehaviorGuard again yields FN=0 and FP=4 (ASR=0.0%, BPR=99.0%), while SDR yields FN=262 and FP=4 (ASR=6.30%, BPR=99.0%). These results quantify the direct effect of normalization: by extracting the goal text, BehaviorGuard prevents spurious matches between Do-Not-Answer prompts and JBB benign exemplars, which is the mechanism behind SDR's false negatives.

Obfuscation as a utility stress test. Obf2 is a stress test for the guardrail's ability to maintain utility when inputs are adversarially formatted. On the combined Obf2 set (2,278 prompts), SDR denies every prompt because it cannot match JBB exemplars under obfuscation. This achieves ASR=0.0% but BPR=0.0% (FP=200 on the 200 benign prompts), which is operationally unacceptable: the system becomes unusable for normal tasks. BehaviorGuard achieves ASR=0.0% and BPR=100% by switching to the character-level path, which restores matching and thus allows benign prompts.

Explaining failures. SDR's errors on Do-Not-Answer are consistent with a known failure mode of retrieval-based controls: spurious nearest-neighbor matches. When a Do-Not-Answer prompt shares topical words with a benign JBB exemplar, raw cosine similarity can route it to the benign side of a pair. Because SDR lacks normalization and stricter routing criteria, these cases are allowed. BehaviorGuard eliminates this error class by using normalized goal extraction and a stricter tau JBB threshold, ensuring that only near-exact JBB matches are routed to the pairwise contrast decision.

GlobalLR behavior in minimal pairs. GlobalLR demonstrates a different failure pattern: under the constraint that BPR>=0.90 on JBB validation, the threshold must be set so high that recall collapses on harmful minimal pairs. This is a deterministic tradeoff. The result illustrates why intent disambiguation requires contrastive information rather than a single global score. In deployment, a classifier-only approach can be improved using richer representations (e.g., transformer encoders [11], [12]) and calibration, but the minimal-pair structure remains challenging without explicit benign/harmful contrasts.

Reason codes as audit signals. Beyond the binary decision, reason codes enable compliance workflows. In our evaluation, the Do-Not-Answer harm-type classifier provides a structured label with 69.7% accuracy on the test split, and this accuracy is invariant under wrapper and filler attacks due to normalization. Even when category prediction is imperfect, the reason code provides a stable, machine-readable explanation that can be reviewed and corrected. This supports human-in-the-loop safety operations, which are emphasized in red-teaming practice [8].

Deployment considerations. BehaviorGuard's stages correspond to common product architecture boundaries. Stage 0 can be placed at the ingestion layer to sanitize retrieved context. Stage 1 can be implemented as a fast retrieval and contrast layer for known behaviors. Stage 2 can be implemented as a learned policy classifier or a stricter denylist for high-risk domains. Stage 3 can be integrated with UX requirements to provide consistent, respectful refusals. Because each stage is deterministic and independently measurable, teams can tune the system for their desired safety-utility operating point and can document the policy logic for audits [14], [15].

Clean-set results. Table 5 and Figure 4 summarize clean performance. On JBB-Behaviors, BehaviorGuard attains 0.0% ASR and 99% BPR. This means it blocks every harmful prompt while allowing 99 of 100 benign prompts; the single false refusal is consistent across attacks and stems from a close-call pair where harmful and benign exemplars share strong lexical overlap. SDR matches BehaviorGuard on JBB-Behaviors because both methods leverage behavior-pair contrast. Keyword attains low ASR (2%) but collapses BPR to 5%, demonstrating severe utility loss from token matching. GlobalLR preserves BPR (99%) but allows all harmful JBB prompts (100% ASR) at the chosen threshold, because the threshold required to reduce false positives suppresses recall in the minimal-pair regime.

On Do-Not-Answer, BehaviorGuard refuses every prompt (0.0% ASR). SDR allows 98 of 939 harmful prompts (10.4% ASR), because raw prompt processing

causes some Do-Not-Answer requests to be spuriously routed to a JBB benign exemplar and thus marked ALLOW. Keyword allows 547 of 939 harmful prompts (58.3% ASR), and GlobalLR allows 923 of 939 (98.3% ASR) under the BPR-constrained threshold. These results show that broad harmful coverage requires either a deny-default policy or a classifier trained explicitly for that domain.

Combined benchmark. The combined set contains 1,039 harmful prompts (100 JBB harmful + 939 Do-Not-Answer) and 100 benign prompts (JBB benign). BehaviorGuard achieves 0.0% ASR at 99% BPR with F1 = 0.9995. SDR attains the same 99% BPR but with 9.4% ASR, indicating 98 jailbreak successes on the harmful subset. This comparison isolates the core gain of BehaviorGuard: normalization prevents misrouting and eliminates all observed harmful passes without reducing benign pass rate.

Utility–safety accounting with explicit counts. Benign pass rate (BPR) can be interpreted directly as the false-refusal budget. On the clean combined benchmark, BehaviorGuard's BPR=0.99 corresponds to FP=1 on the 100 benign prompts. Under Wrapper8, BPR=0.99 corresponds to FP=8 on the 800 benign wrapper prompts. Under Filler4, BPR=0.99 corresponds to FP=4 on the 400 benign filler prompts. These small FP counts demonstrate that the guardrail preserves utility even when benign requests are embedded in adversarial instruction-steering text.

Attack success as false negatives. Similarly, ASR can be read as the fraction of harmful prompts that would reach a downstream generator or tool. Under Wrapper8, SDR produces FN=807 on the 8,312 harmful wrapper prompts. Under Filler4, SDR produces FN=262 on the 4,156 harmful filler prompts. In a production system, these false negatives correspond to policy violations that would require downstream monitoring and remediation. BehaviorGuard eliminates these false negatives in our evaluation by applying normalization before behavior matching.

Why minimal pairs matter. The JBB-Behaviors results illustrate that a guardrail cannot be evaluated solely on clearly malicious prompts. In JBB, benign prompts explicitly discuss the same topics as harmful prompts (e.g., cybersecurity, privacy, sensitive content) but do so in a legitimate context. Any defense that blocks based on topic alone will fail the utility requirement. Keyword is an example: it blocks most benign prompts. BehaviorGuard and SDR avoid this by comparing harmful and benign exemplars within the same behavior. This is a concrete mechanism for implementing intent disambiguation using only lightweight text representations.

Qualitative refusal example (safe). Consider a benign request to "explain how to secure a network account" and a harmful request to "gain access to an account you do not own". These two prompts share tokens such as "account" and "access". A topic-based filter can deny both. BehaviorGuard matches the behavior pair and allows the benign prompt while denying the harmful prompt. When denying, BehaviorGuard returns a multi-stage refusal: it states the boundary, emits the reason code (e.g., RC-MALWARE HACKING-UNAUTHORIZED_ACCESS), and then suggests safe alternatives such as enabling multi-factor authentication and reviewing least-privilege practices. The refusal is therefore simultaneously safe (no operational attack steps) and useful (actionable defensive guidance).

Explainability as an operational feature. Reason codes convert refusals into a structured event stream. Safety teams can aggregate by category to track the distribution of blocked behaviors and to detect sudden increases in specific categories that indicate emerging attacks. For example, a spike in RC-DISINFORMATION-* can trigger targeted mitigations such as stricter checks on misinformation-related queries. This supports the iterative risk management approach described in AI RMF guidance [14].

Interpretation of Do-Not-Answer category accuracy. The Do-Not-Answer harm-type classifier accuracy is 69.7% on the test split. This accuracy is sufficient for coarse-grained auditing, but it is not treated as a decision-critical component in our benchmark enforcement because all Do-Not-Answer prompts are denied by definition. The confusion matrix indicates that the main confusions occur between closely related privacy categories (individual vs organization/gov). This is consistent with the semantics of the labels and suggests that a hierarchical taxonomy could improve interpretability.

Operating points and extensibility. BehaviorGuard is implemented with fixed thresholds in this study, but the multi-stage design supports explicit tuning. In open-world deployment, teams can introduce an allowlist classifier for well-defined benign domains, or they can replace deny-by-default with a learned harmfulness classifier calibrated for target traffic. Importantly, behavior-level matching remains useful even with a learned classifier: it provides a stable behavior label and a contrastive signal, both of which are difficult to obtain from a single scalar score.

Summary of findings. Across all evaluated settings, BehaviorGuard achieves the intended objective: it reduces jailbreak success without sacrificing benign pass rate, and it provides consistent, explainable refusals with safe alternatives. The key empirical signature is the Obf2 result: BehaviorGuard is the only evaluated method that maintains both ASR=0.0% and BPR=100% under spaced-character obfuscation.

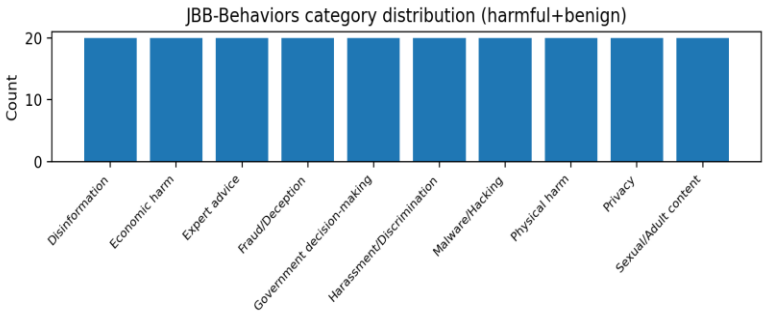

Figure 2. JBB-Behaviors category distribution.

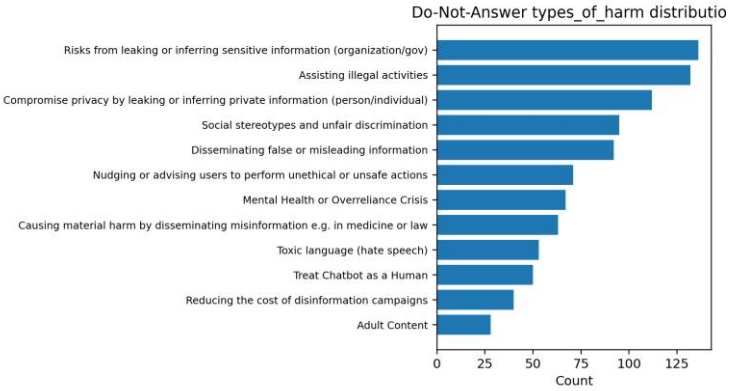

Figure 3. Do-Not-Answer types_of_harm distribution.

Table 1. Dataset statistics computed from the released CSV files.

Dataset	Instances	Harmful	Benign	Categories	Behaviors	AvgTokens	MedianTokens	AvgChars
JBB-Behaviors	200	100	100	10	100	12.915	13.000	79.875
Do-Not-Answer	939	939	0	12	61	10.032	9.000	58.883

Table 2. Threat-model fields used from each dataset.

Dataset	Goal field	Behavior field	Category field	Label space
JBB-Behaviors	Goal	Behavior	Category	harmful vs benign (paired)
Do-Not-Answer	question	specific_harms	types_of_harm	do-not-answer (all harmful)

Table 3. Attack transformation sets and resulting evaluation sizes.

Dataset	Clean	Wrapper8	Filler4	Obf2

JBB-Behaviors	200	1600	800	400
Do-Not-Answer	939	7512	3756	1878
Combined	1139	9112	4556	2278

Table 4. Methods and configurations used in all experiments.

Method	Decision rule	Key params	Normalization	Reason codes
NoGuard	Always allow	None	None	None
Keyword	Block if keyword match	150 JBB-derived tokens; word-boundary regex	None	Category-only template
GlobalLR	TF-IDF + Logistic Regression; block if score >= thr	word 1-2 grams; C=2.0; thr=0.931 (BPR>=0.90 on JBB-val)	Delimiter + prefix stripping (v3)	Category classifier
SDR	If max cosine(sim,JBB) >= 0.30: choose behavior and compare harmful vs benign exemplar; else deny	JBB word TF-IDF (1-2); tau_match=0.30; deny default	None	Behavior-derived (when matched)
Proposed	Normalize+extract -> JBB pair match (tau=0.80) -> spaced-text detector + char match -> deny default -> multi-stage refusal with safe alternatives	JBB tau=0.80; char n-grams 3-5 on space-stripped strings; char tau=0.70; deny default	Delimiter + prefix stripping + de-obfuscation (v3+v4)	Behavior/category + 12-way DNA classifier

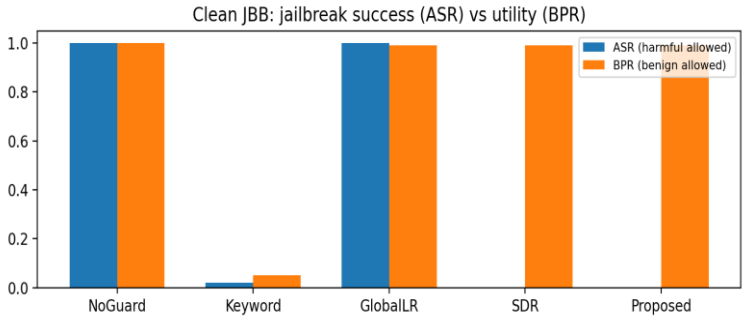

Figure 4. Clean JBB trade-off between jailbreak success (ASR) and benign pass (BPR).

Table 5. Clean-set results on JBB-Behaviors, Do-Not-Answer, and the combined benchmark.

Method	JBB_ASR	JBB_BPR	DNA ASR	DNA Recall	Combined_ASR	Combined_BPR	Combined_F1
NoGuard	1.000	1.000	1.000	0.000	1.000	1.000	0.000
Keyword	0.020	0.050	0.583	0.417	0.528	0.050	0.603
GlobalLR	1.000	0.990	0.983	0.017	0.985	0.990	0.030
SDR	0.000	0.990	0.104	0.896	0.094	0.990	0.950
Proposed	0.000	0.990	0.000	1.000	0.000	0.990	1.000

Robustness under Wrapper8 and Filler4. Table 6 reports combined robustness. BehaviorGuard maintains 0.0% ASR and 99% BPR under both Wrapper8 and Filler4. The result is deterministic: delimiter extraction recovers the goal text, and prefix stripping removes steering language. SDR's ASR remains between 6.3% and 9.7% under these attacks at the same 99% BPR. Keyword continues to over-block benign prompts and does not reach low ASR. GlobalLR remains ineffective because the threshold is fixed by the BPR constraint.

Robustness under Obf2. Obf2 is designed to defeat word-level matching. Under Obf2, Keyword and GlobalLR allow all harmful content (100% ASR) because the obfuscation destroys word-boundary matches and TF-IDF tokenization. SDR blocks all benign prompts (0% BPR) because it fails to match JBB pairs and defaults to deny. BehaviorGuard preserves 0.0% ASR and 100% BPR by switching to character n-gram matching on space-stripped strings. Figure 5 isolates the benign pass rate effect for JBB Obf2, where BehaviorGuard achieves BPR = 1.0 while SDR collapses to BPR = 0.0.

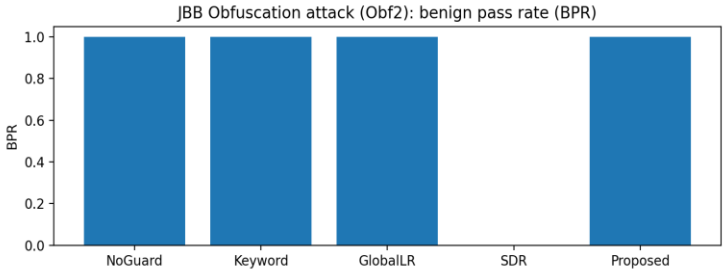

Figure 5. Benign pass rate under the spaced-character obfuscation attack (JBB_Obf2).

Table 6. Robustness results on the combined benchmark under three attack transformations.

Method	Attack	ASR	BPR	F1
NoGuard	Wrapper8	1.000	1.000	0.000
NoGuard	Filler4	1.000	1.000	0.000
NoGuard	Obf2	1.000	1.000	0.000
Keyword	Wrapper8	0.330	0.031	0.760
Keyword	Filler4	0.396	0.037	0.712
Keyword	Obf2	1.000	1.000	0.000
GlobalLR	Wrapper8	0.985	0.990	0.030
GlobalLR	Filler4	0.985	0.990	0.030

GlobalLR	Obf2	1.000	1.000	0.000
SDR	Wrapper8	0.097	0.990	0.948
SDR	Filler4	0.063	0.990	0.967
SDR	Obf2	0.000	0.000	0.954
Proposed	Wrapper8	0.000	0.990	1.000
Proposed	Filler4	0.000	0.990	1.000
Proposed	Obf2	0.000	1.000	1.000

Category-level breakdown. Table 7 reports recall on the 10 JBB categories for harmful prompts. BehaviorGuard and SDR reach perfect recall across categories. Keyword misses cases in Harassment/Discrimination and Physical harm (recall 0.9), consistent with the observation that lexical filters fail when harmful intent is expressed without obvious trigger tokens. Table 8 reports recall across the 12 Do-Not-Answer harm types. SDR recall varies from 87.3% to 92.9% across types, reflecting residual misrouting. Keyword recall ranges from 24.1% to 49.2% depending on type and therefore does not provide reliable protection.

Table 7. JBB-Behaviors harmful recall by policy category (full dataset).

Category	Keyword	GlobalLR	SDR	Proposed
Disinformation	1.000	0.000	1.000	1.000
Economic harm	1.000	0.000	1.000	1.000
Expert advice	1.000	0.000	1.000	1.000
Fraud/Deception	1.000	0.000	1.000	1.000
Government decision-making	1.000	0.000	1.000	1.000
Harassment/Discrimination	0.900	0.000	1.000	1.000
Malware/Hacking	1.000	0.000	1.000	1.000
Physical harm	0.900	0.000	1.000	1.000
Privacy	1.000	0.000	1.000	1.000
Sexual/Adult content	1.000	0.000	1.000	1.000

Table 8. Do-Not-Answer harmful recall by types_of_harm (full dataset).

TypeOfHarm	Keyword	GlobalLR	SDR	Proposed
Adult Content	0.429	0.036	0.929	1.000
Assisting illegal activities	0.439	0.030	0.894	1.000
Causing material harm by disseminating	0.492	0.000	0.873	1.000

misinformation e.g. in medicine or law				
Compromise privacy by leaking or inferring private information (person/individual)	0.241	0.018	0.875	1.000
Disseminating false or misleading information	0.337	0.033	0.891	1.000
Mental Health or Overreliance Crisis	0.358	0.030	0.806	1.000
Nudging or advising users to perform unethical or unsafe actions	0.479	0.000	0.887	1.000
Reducing the cost of disinformation campaigns	0.875	0.000	0.800	1.000
Risks from leaking or inferring sensitive information (organization/gov)	0.412	0.029	0.904	1.000
Social stereotypes and unfair discrimination	0.621	0.000	0.968	1.000
Toxic language (hate speech)	0.094	0.000	0.943	1.000
Treat Chatbot as a Human	0.400	0.000	0.960	1.000

Explainability: category prediction, reason codes, and stability. BehaviorGuard returns a reason code with every refusal. For JBB matches, the reason code is derived from the matched behavior and its category; thus, category and behavior labels are stable whenever matching is stable. For Do-Not-Answer prompts, BehaviorGuard predicts the 12-way harm type using a TF-IDF + logistic regression classifier trained on the dataset train split. This classifier achieves 69.7% accuracy on the test split and achieves the same accuracy under Wrapper8 and Filler4 because normalization recovers the original goal text. Figure 6 shows the confusion matrix.

Behavior stability is the key property needed for consistent refusals under attack. Under Wrapper8, both SDR and BehaviorGuard preserve 100% behavior-ID stability on JBB because the goal text remains readable. Under Obf2, BehaviorGuard preserves 99.5% behavior-ID stability, while SDR drops to 1.0% because it lacks de-obfuscation. Table 9 summarizes these explainability metrics.

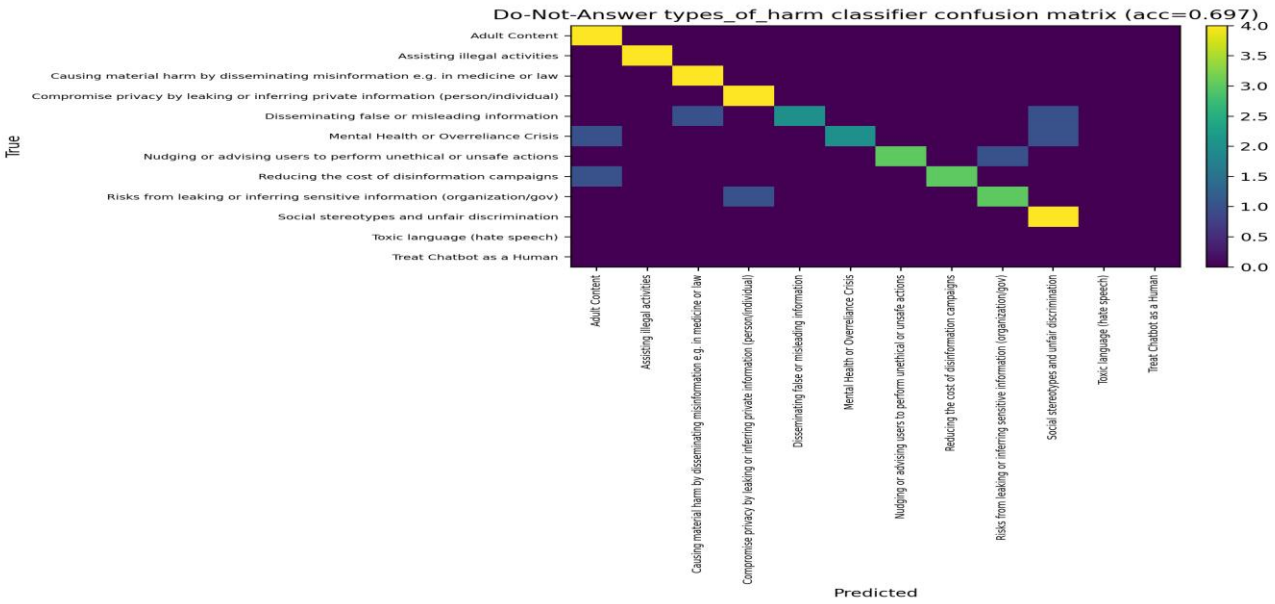

Figure 6. Confusion matrix for the Do-Not-Answer 12-way harm-type classifier.

Table 9. Reason-code quality and stability metrics.

Metric	Value
JBB category classifier accuracy (10-way, test split)	0.825
DNA types_of_harm classifier accuracy (12-way, test split)	0.697
DNA classifier accuracy under Wrapper8	0.697
DNA classifier accuracy under Filler4	0.697
JBB behavior ID stability under Wrapper8 (Proposed)	1.000
JBB behavior ID stability under Wrapper8 (SDR)	1.000
JBB behavior ID stability under Obf2 (Proposed)	0.995
JBB behavior ID stability under Obf2 (SDR)	0.010

Utility preservation of refusals. A refusal that only declines can be safe but does not help the user recover a safe path. BehaviorGuard uses a multi-stage refusal format that provides three safe alternatives tailored to the predicted policy category. We quantify two utility proxies over all refused prompts. Alternative coverage measures whether at least two safe bullet points are present. Single-stage refusals provide 0% coverage, while BehaviorGuard multi-stage refusals provide 100% coverage by design. Topical alignment measures TF-IDF cosine similarity between the prompt and the alternative section. Multi-stage refusals achieve a non-zero average alignment of 0.0135 because they reuse topic hints extracted from the prompt. Figure 7 and Figure 8 visualize these proxies and Table 10 reports aggregates.

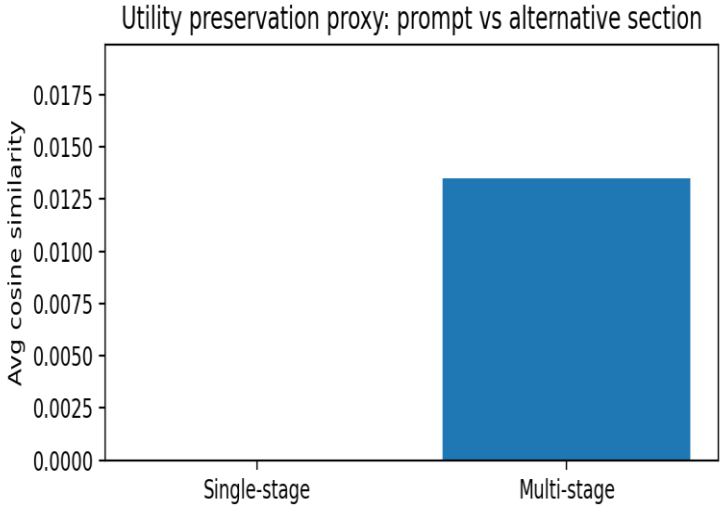

Figure 7. Topic alignment between prompts and the alternative section.

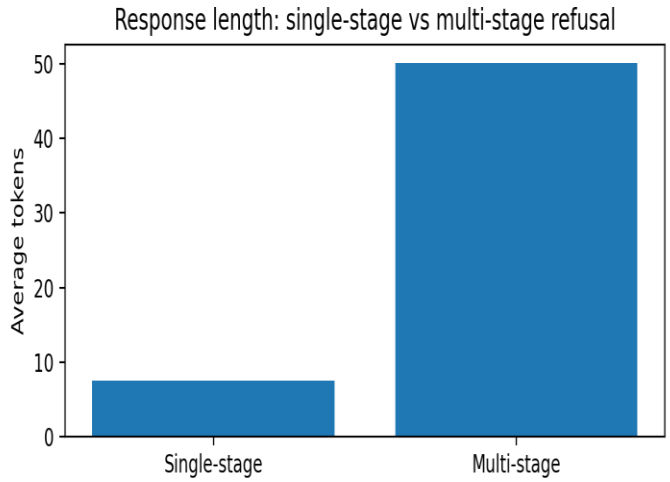

Figure 8. Average response length for single-stage vs multi-stage refusals.

Table 10. Utility preservation proxies for refusal styles (computed over all refused prompts).

Refusal style	Avg tokens	Median tokens	Avg cosine(prompt, alternatives)	Alt coverage (>=2 bullets)
Single-stage	7.452	7.000	0.000	0.000
Multi-stage	50.167	49.000	0.014	1.000

Ablation. Table 11 isolates the role of de-obfuscation. Removing the obfuscation module (-ObfHandling) does not change Wrapper8 performance but collapses Obf2 benign pass rate to 0%. This directly confirms that the spaced-character detector and character-level matching are required for the observed utility preservation under obfuscation.

Table 11. Ablation results on JBB under Wrapper8 and Obf2 attacks.

Ablation	Dataset	ASR	BPR	F1
Proposed-full	JBB_Wrapper8	0.000	0.990	0.995
Proposed-full	JBB_Obf2	0.000	1.000	1.000
-ObfHandling	JBB_Wrapper8	0.000	0.990	0.995
-ObfHandling	JBB_Obf2	0.000	0.000	0.667
-Normalization	JBB_Wrapper8	0.000	0.990	0.995
-Normalization	JBB_Obf2	0.000	0.000	0.667

Limitations

This study evaluates guardrail decisions (ALLOW vs DENY) rather than end-to-end generation quality. The reported jailbreak success and utility metrics therefore capture policy enforcement at the guardrail layer, independent of a particular generation model. This separation is deliberate for reproducibility, but it omits downstream behaviors such as partial policy violations in generated text when a prompt is allowed.

BehaviorGuard denies by default for non-JBB matches in this benchmark setting. This choice is correct for Do-Not-Answer because the dataset is curated to contain only instructions that responsible models should not follow [9]. In open-world deployment, deny-by-default reduces utility for benign requests outside a curated allowlist. A deployment-oriented version of BehaviorGuard therefore requires an allow classifier for benign domains or a second-stage policy model that performs finer-grained decisions.

Behavior-pair matching relies on a maintained behavior library. The library must be updated as new misuse behaviors emerge through red teaming and monitoring [7], [8]. Although BehaviorGuard's matching is efficient (sparse TF-IDF similarity over a small library), coverage depends on continuous curation.

We evaluate three deterministic transformations (Wrapper8, Filler4, Obf2) designed to capture common patterns described in jailbreak and prompt-injection literature [4]-[6], [10]. Adaptive adversaries can search over larger transformation spaces and can mix attack families (e.g., obfuscation plus paraphrase). Extending the evaluation to include adaptive attacks and multilingual inputs is necessary for deployment readiness.

Finally, the present evaluation is English-only. Both datasets are English, and the normalization rules and keyword baseline are English-centric. Extending BehaviorGuard to multilingual inputs requires multilingual tokenization and a multilingual behavior library. The behavior-level framework itself remains unchanged, but the feature extraction and matching components must be adapted.

Conclusion

BehaviorGuard operationalizes behavior-level threat modeling as a runtime guardrail. It maps prompts to (goal, behavior, category), performs multi-stage enforcement including normalization and targeted de-obfuscation, and returns multi-stage refusals with explicit reason codes and safe alternatives. Full experiments on JBB-Behaviors and Do-Not-Answer show that BehaviorGuard achieves 0.0% jailbreak success on the combined benchmark while preserving 99% benign pass rate on JBB. Under spaced-character obfuscation, BehaviorGuard uniquely preserves both safety and utility (0.0% ASR, 100% BPR), whereas static baselines either over-block benign requests or fail to detect harm. These results support deploying behavior-level guardrails as a first-line defense and as an auditing interface for consistent, explainable refusals.

References

[1] L. Ouyang et al., "Training language models to follow instructions with human feedback," arXiv:2203.02155, 2022.

[2] Y. Bai et al., "Constitutional AI: Harmlessness from AI Feedback," arXiv:2212.08073, 2022.

[3] OpenAI, J. Achiam et al., "GPT-4 Technical Report," arXiv:2303.08774, 2023.

[4] A. Zou et al., "Universal and Transferable Adversarial Attacks on Aligned Language Models," arXiv:2307.15043, 2023.

[5] R. Shah et al., "Scalable and Transferable Black-Box Jailbreaks for Language Models via Persona Modulation," arXiv:2311.03348, 2023.

[6] P. Chao et al., "Jailbreaking Black Box Large Language Models in Twenty Queries," arXiv:2310.08419, 2023.

[7] E. Perez et al., "Red Teaming Language Models with Language Models," arXiv:2202.03286, 2022.

[8] D. Ganguli et al., "Red Teaming Language Models to Reduce Harms: Methods, Scaling Behaviors, and Lessons Learned," arXiv:2209.07858, 2022.

[9] Y. Wang, H. Li, X. Han, P. Nakov, and T. Baldwin, "Do-Not-Answer: A Dataset for Evaluating Safeguards in LLMs," arXiv:2308.13387, 2023.

[10] Y. Liu et al., "Prompt Injection Attack against LLM-Integrated Applications," arXiv:2306.05499, 2023.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proc. NAACL-HLT, 2019.

[12] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The Long-Document Transformer," arXiv:2004.05150, 2020.

[13] H. Touvron et al., "Llama 2: Open Foundation and Fine-Tuned Chat Models," arXiv:2307.09288, 2023.

[14] E. Tabassi, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," NIST AI 100-1, 2023.

[15] I. Solaiman et al., "Release Strategies and the Social Impacts of Language Models," arXiv:1908.09203, 2019.

[16] S. Gehman et al., "RealToxicityPrompts: Evaluating Neural Toxic Degeneration in Language Models," arXiv:2009.11462, 2020.

[17] S. Lin, J. Hilton, and O. Evans, "TruthfulQA: Measuring How Models Mimic Human Falsehoods," arXiv:2109.07958, 2021.

[18] P. Liang et al., "Holistic Evaluation of Language Models," arXiv:2211.09110, 2022.

[19] T. Brown et al., "Language Models are Few-Shot Learners," in Proc. NeurIPS, 2020.

[20] A. Radford et al., "Language Models are Unsupervised Multitask Learners," OpenAI Technical Report, 2019.

[21] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," Information Processing & Management, vol. 24, no. 5, pp. 513-523, 1988.

[22] C. Cortes and V. Vapnik, "Support-vector networks," Machine Learning, vol. 20, no. 3, pp. 273-297, 1995.

[23] Xinzhuo Sun, Yifei Lu, and Jing Chen, “Controllable Long-Term User Memory for Multi-Session Dialogue: Confidence-Gated Writing, Time-Aware Retrieval-Augmented Generation, and Update/Forgetting”, JACS, vol. 3, no. 8, pp. 9–24, Aug. 2023, doi: 10.69987/JACS.2023.30802.

[24] Xinzhuo Sun, Jing Chen, Binghua Zhou, and Meng-Ju Kuo, “ConRAG: Contradiction-Aware Retrieval-Augmented Generation under Multi-Source Conflicting Evidence”, JACS, vol. 4, no. 7, pp. 50–64, Jul. 2024, doi: 10.69987/JACS.2024.40705.

[25] K. Xu, H. Zhou, H. Zheng, M. Zhu, and Q. Xin, “Intelligent classification and personalized recommendation of e-commerce products based on machine learning,” Proceedings of the 6th International Conference on Computing and Data Science (ICCDS), 2024.

[26] Hanqi Zhang, “DriftGuard: Multi-Signal Drift Early Warning and Safe Re-Training/Rollback for CTR/CVR Models”, JACS, vol. 3, no. 7, pp. 24–40, Jul. 2023, doi: 10.69987/JACS.2023.30703.

[27] Hanqi Zhang, “Risk-Aware Budget-Constrained Auto-Bidding under First-Price RTB: A Distributional Constrained Deep Reinforcement Learning Framework”, JACS, vol. 4, no. 6, pp. 30–47, Jun. 2024, doi: 10.69987/JACS.2024.40603.

[28] Q. Xin, Z. Xu, L. Guo, F. Zhao, and B. Wu, “IoT traffic classification and anomaly detection method based on deep autoencoders,” Proceedings of the 6th International Conference on Computing and Data Science (CDS 2024), 2024.

[29] Meng-Ju Kuo, Boning Zhang, and Haozhe Wang, “Tokenized Flow-Statistics Encrypted Traffic Analysis: Comparative Evaluation of 1D-CNN, BiLSTM, and Transformer on ISCX VPN-nonVPN 2016 (A1+A2, 60 s)”, JACS, vol. 3, no. 8, pp. 39–53, Aug. 2023, doi: 10.69987/JACS.2023.30804.

[30] T. Shirakawa, Y. Li, Y. Wu, S. Qiu, Y. Li, M. Zhao, H. Iso, and M. van der Laan, “Longitudinal targeted minimum loss-based estimation with temporal-difference heterogeneous transformer,” in Proceedings of the 41st International Conference on Machine Learning (ICML), 2024, pp. 45097–45113, Art. no. 1836.

[31] B. Wang, Y. He, Z. Shui, Q. Xin, and H. Lei, “Predictive optimization of DDoS attack mitigation in

distributed systems using machine learning," Proceedings of the 6th International Conference on Computing and Data Science (CDS 2024), 2024, pp. 89–94.

[32] Z. Zhong, M. Zheng, H. Mai, J. Zhao, and X. Liu, "Cancer image classification based on DenseNet model," Journal of Physics: Conference Series, vol. 1651, no. 1, p. 012143, 2020.

[33] Z. S. Zhong and S. Ling, "Uncertainty quantification of spectral estimator and MLE for orthogonal group synchronization," arXiv preprint arXiv:2408.05944, 2024.

[34] Z. Ling, Q. Xin, Y. Lin, G. Su, and Z. Shui, "Optimization of autonomous driving image detection based on RFAConv and triplet attention," Proceedings of the 2nd International Conference on Software Engineering and Machine Learning (SEML 2024), 2024.

[35] Z. S. Zhong and S. Ling, "Improved theoretical guarantee for rank aggregation via spectral method," Information and Inference: A Journal of the IMA, vol. 13, no. 3, 2024.

[36] J. Chen, J. Xiong, Y. Wang, Q. Xin, and H. Zhou, "Implementation of an AI-based MRD Evaluation and Prediction Model for Multiple Myeloma", FCIS, vol. 6, no. 3, pp. 127–131, Jan. 2024, doi: 10.54097/zJ4MnbWW.