LLM Prompt Injection / Jailbreak Detection System

Soham Murkar[1], Aarohi Talele[2], Paarth Oturkar[3], Dev Parab[4], Prof. Teena Verma[5]
[1,2,3,4,5]*Department of Computer Engineering, Xavier Institute of Engineering, Mahim, University of Mumbai, Mumbai, India*

***Abstract*—Large language models (LLMs) can be exposed to prompt-injection and jailbreak attempts that preserve malicious intent while changing its surface representation. This work builds a lightweight classical machine-learning detector and uses it to study how such representation shifts affect detection. A unified corpus of 1,093 deduplicated behaviors (993 attack/harmful, 100 benign) is assembled from AdvBench, HarmBench, Jailbreak Bench and ArtPrompt; 600 further unique benign prompts are added to training only. Twelve text transformations (plain text, Base64, mathematical, ASCII-art, code, reasoning, Uni-code and structural forms) expand the data to 20,316 rows, which are represented with TF-IDF word and character features (77,626 features). Logistic Regression, Linear SVM, CART, Random Forest and a Voting Ensemble are compared on a behavior-level held-out test set of 1,980 rows. Random Forest obtains the highest F1 (0.9448) and recall (0.9833), but its specificity is only 0.0167; the best specificity, from Logis-tic Regression, is 0.2222. Benign false-positive rates exceed 50% for every model on every transformation, including plain text. The results show that high attack recall and F1 do not establish reliable benign-versus-attack discrimination, and the study is positioned as a reproducible baseline and error-analysis framework rather than a production security control.**

***Index Terms*—prompt injection, jailbreak detection, TF-IDF, Logistic Regression, support vector machine, random forest, specificity, adversarial robustness**

I. INTRODUCTION

LLMs are now used for conversation, retrieval, coding, tool in-vocation and multi-agent workflows. Text reaches them across several trust boundaries: direct users, retrieved documents, web pages and tool outputs. Prompt injection exploits this by making a model treat attacker-controlled content as instructions [6–8], while jailbreaks disguise a harmful request so that safety training does not recognize it [10, 12].

The same intent can be written in very different forms. CodeAttack reframes requests as code completion [1]; Art-Prompt hides a keyword in ASCII art [2]; Kwon and Pak use mathematical functions [3]; cipher-style encodings evade alignment [13]; and AutoDAN and gradient-based suffixes search for fluent or universal jailbreaks [4, 11]. Defenses such as perturbation smoothing [5], perplexity filtering [16, 17] and safeguard models [18, 19] each target particular attack properties.

The project asks a deliberately narrow, testable question: *how well does a lightweight TF-IDF text classifier separate benign prompts from a heterogeneous set of attack/harmful representations, and at what false-positive cost?*

Our contributions are:

- A unified, deduplicated corpus from four published red-teaming sources, expanded with twelve representation-shift transformations.
- Behavior-level splitting so transformed variants of one behavior never straddle train, validation and test sets.
- A controlled comparison of five classical models on identical TF-IDF features and an identical frozen test set.
- A class-sensitive evaluation (specificity, FPR, confusion matrices, per-transformation FPR, threshold sweep) that exposes a severe false-positive failure hidden by aggregate F1.

II. RELATED WORK

A. Attacks and benchmarks

Perez and Ribeiro describe goal hijacking and prompt leaking [6]; Greshake et al. and Liu et al. show indirect injection in LLM-integrated applications [7, 8], and Liu et al. later formalize and benchmark attacks and defenses [9]. Wei et al. attribute jailbreaks to

competing objectives and mismatched generalization [10], which is exactly the situation created by encodings such as Base64 [13]. In-the-wild jailbreak prompts have been characterized by Shen et al. [12]. Standardized datasets used here are AdvBench [11], HarmBench [14] and JailbreakBench [15], together with the ArtPrompt dataset [2].

B. Defenses and detectors

SmoothLLM aggregates responses over perturbed inputs but is motivated by brittle optimized strings [5]. Baseline defenses perplexity filters, paraphrasing and retokenization—were examined by Jain et al. [16], and Alon and Kamfonas train a classifier on perplexity and length features [17]. Llama Guard [18] and NeMo Guardrails [19] provide LLM-based or programmable input–output safeguards. Related earlier work by the authors is reported in [28]. These systems need substantial compute; a TF-IDF pipeline is a transparent, GPU-free baseline in the tradition of linear text classification [20–22].

C. Classical learners and imbalance

Our model set follows standard methods: regularized linear models and SVMs [21,22], CART [23] and bagged forests [24], implemented in scikit-learn [25]. Because the corpus is attack-heavy, we follow guidance on imbalanced learning [26] and on class-sensitive evaluation beyond accuracy [27].

III. DATASET AND PROBLEM FORMULATION

A. Unified corpus

Raw inputs are AdvBench (520), ArtPrompt (50), HarmBench (400), JBB-Harmful (100) and JBB-Benign (100), i.e., 1,170 records. Records are normalized and keyed by SHA-256; only exact duplicates are removed (77), leaving 1,093 behaviors (993 attack/harmful, 100 benign) with no conflicting label groups. To reduce the benign shortfall, 600 additional unique benign prompts are appended to the *training partition only*.

B. Labels and split control

The binary target is 0 = benign, 1 = attack/harmful. Original be-haviors are split 70/15/15 with seed 42, and all transformations of a behavior stay in one partition. The 600 benign controls never enter validation or test, so the test partition is identical to earlier runs and remains comparable.

C. Twelve transformations

Each behavior is expanded into twelve forms: plain_text, base64, math_function, ascii_art, code_char_code, code_stack_push, reasoning_linear_equation, reasoning_modular_clue, unicode_homoglyph, unicode_zero_vidth, structural_plain_override and structural_chat_markup.

The result is 20,316 rows: 16,380 train, 1,956 validation and 1,980 test (Table 2). Benign behaviors are transformed in the same way, so a transformation is not by itself a label cue.

D. Problem formulation

Given a prompt x, the detector computes $f(x) \in \{0, 1\}$ via normalization, TF-IDF vector z and classifier $g(z)$, with a score where supported. Attack-family labels are kept as metadata for analysis only.

IV. PROPOSED METHODOLOGY

A. Pipeline

Fig. 1 shows the pipeline: unified corpus, behavior-level split, preprocessing, TF-IDF features, five classifiers, and a binary verdict with score. Unicode and whitespace normalization precede TF-IDF over word and character *n*-grams; character features target tokenizer-level and unusual surface forms. The vectorizer is fitted on training data only.

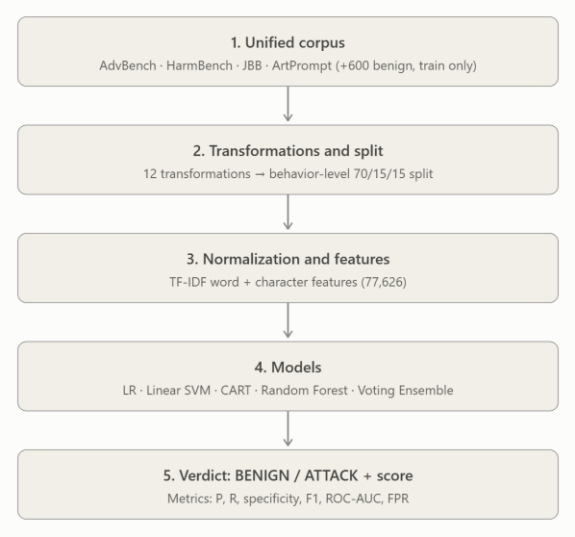

Figure 1. System architecture. Transformations are generated before splitting by behavior; the same frozen features feed all five models.

B. Models and protocol

Logistic Regression and Linear SVM are the linear baselines; CART and Random Forest provide non-linear comparisons; a Voting Ensemble combines the classifiers.

All models use the same features and 0.50 decision threshold for the primary comparison. The test set is used only for final reporting. The genetic, AutoDAN-style hybrid generation [4] is described as an offline data-generation design; it is *not* part of the reported experiments.

C. Metrics

With attack as the positive class, precision = $TP/(TP + FP)$, recall = $TP/(TP + FN)$, specificity = $TN/(TN + FP)$, FPR = 1−specificity, and F1 is the harmonic mean of precision and recall. Because 90.9% of test rows are attacks, accuracy and F1 are dominated by the majority class, so specificity and balanced accuracy = (recall + specificity)/2 are reported alongside ROC-AUC [26, 27].

D. Implementation

The pipeline is written in Python with pandas, NumPy, scikit-learn [25], joblib and Matplotlib and runs on an ordinary Windows laptop without a GPU. A command-line interface loads a saved model and prints an ATTACK/BENIGN label and attack score for a user-entered prompt.

V. RESULTS

A. Overall comparison

Table 3 and Fig. 2 give held-out results at threshold 0.50. Random Forest has the best F1 (0.9448) and recall (0.9833), but specificity is 0.0167: it labels 177 of 180 benign rows as attacks. Linear SVM has the highest reported ROC-AUC (0.6969), marginally above Logistic Regression (0.6951); CART's 0.5074 is near chance. All five models score F1 above 0.90 yet balanced accuracy only 0.50–0.56, so F1 overstates real discrimination.

B. Confusion matrices

Table 4 and Fig. 3 show the error structure. Random Forest minimizes false negatives (30) but recognizes only three benign rows. Logistic Regression has the most true negatives (40 of 180), and the Voting Ensemble sits between the extremes (31 TN, 96 FN).

Table 1. Reference literature and project response

Ref.	Technique	Mechanism	Project response
[1]	CodeAttack	Code-completion framing	Code encoders + char/word TF-IDF
[2]	ArtPrompt	Keyword as ASCII art	Automated ASCII-art transformation
[3]	Math-function injection	Symbolic sensitive terms	Math and reasoning encoders
[4]	AutoDAN	Genetic evolution of prompts	Offline hybrid data generation (design)
[5]	SmoothLLM	Perturbation + aggregation	Defense context, not primary detector
[13]	Cipher chat	Encoded harmful requests	Base64 / Unicode transformations
[18, 19]	Safeguard models	LLM-based I/O filtering	Heavier alternative; future comparison

Table 2. Dataset and experiment summary

Quantity	Value
Original clean behaviors (benign / attack)	1,093 (100 / 993)
Extra benign training controls	600
Transformations	12
Total rows	20,316
Train rows (8,040 benign / 8,340 attack)	16,380
Validation rows	1,956
Test rows (180 benign / 1,800 attack)	1,980
TF-IDF features	77,626

Table 3. Held-out test performance at threshold 0.50 (1,980 rows)

Model	Prec.	Rec.	Spec.	F1	Bal.Acc	AUC
Logistic Reg.	0.9206	0.9022	0.2222	0.9113	0.5622	0.6951
Linear SVM	0.9194	0.9250	0.1889	0.9222	0.5569	0.6969
CART	0.9155	0.9028	0.1667	0.9091	0.5347	0.5074
Random Forest	0.9091	0.9833	0.0167	0.9448	0.5000	–
Voting Ens.	0.9196	0.9467	0.1722	0.9329	0.5594	–

Precision, recall, specificity, F1 and balanced accuracy computed from Table 4.
ROC-AUC for Random Forest and Voting Ensemble was not recorded in the run log (–).

C. False positives by transformation

Table 5 and Fig. 4 report benign FPR per transformation (15 benign behaviors each). Errors are not confined to encodings: plain-text benign prompts are flagged 60–100% of the time, Logistic Regression flags 73.3% of plain text, 86.7% of Base64 and 100% of homoglyph rows, and *all* models flag every benign homoglyph row. Random Forest's FPR is ≥86.7% everywhere. Hence the weakness is the benign/attack boundary itself, not one hard attack family.

D. Feature diagnosis

Inspection of influential TF-IDF features shows that generic phrases such as "how to", "to", "generate", "write script", "develop" and "people" push predictions toward the attack class. These phrases are common in benign programming, educational and medical requests; the models therefore partly learn instruction-style wording instead of harmful intent, consistent with the limits of lexical features noted in [10].

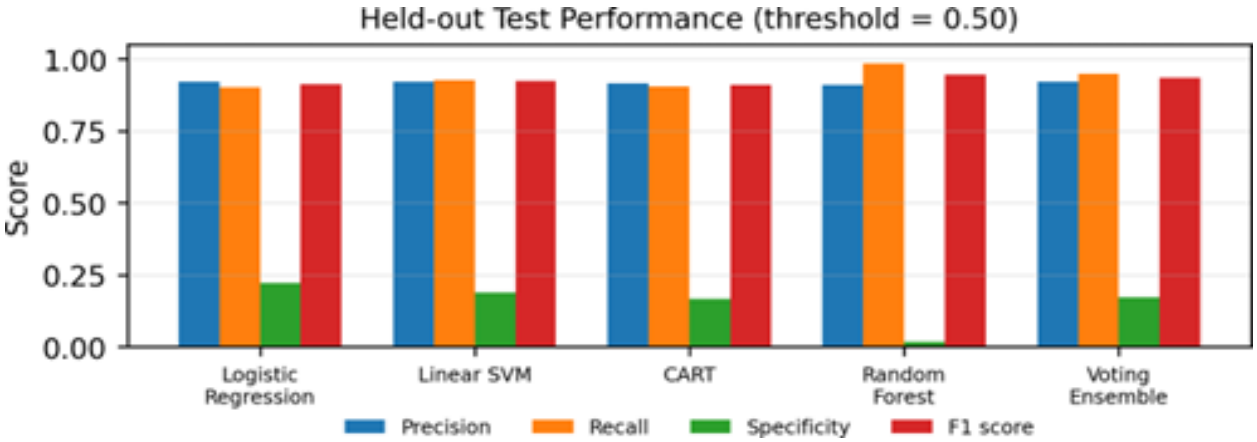

Figure 2. Held-out precision, recall, specificity and F1 of the five classifiers. Specificity is far below the other three metrics for every model.

Table 4. Confusion-matrix counts at threshold 0.50

Model	TN	FP	FN	TP
Logistic Regression	40	140	176	1,624
Linear SVM	34	146	135	1,665
CART	30	150	175	1,625
Random Forest	3	177	30	1,770
Voting Ensemble	31	149	96	1,704

E. Threshold trade-off

Fig. 5 sweeps the Voting Ensemble threshold. At 0.50, recall is 94.67% and specificity 17.22%; at 0.85, recall falls to 51.56% while specificity rises to 77.78%; at 0.90, specificity is 93.89% but recall only 34.28%.

No threshold gives both high recall and high specificity. This sweep uses test scores and is therefore a *diagnostic only*; a deployable threshold must be chosen on validation data and then evaluated once on the frozen test set.

VI. DISCUSSION

TF-IDF with Logistic Regression, SVM, CART and Random Forest are established methods, so the value of this study lies in the controlled evaluation rather than model novelty. The central finding is a negative but informative one: on this corpus, a headline F1 of 0.91–0.94 coexists with specificity of 0.02–0.22. Three factors plausibly contribute. (i) *Class imbalance:* even after 600 added benign controls, training has 8,040 benign vs. 8,340 attack rows, yet only 70 of those benign behaviors are drawn from the benchmark distribution, and the test benign set has just 15 behaviors. (ii) *Label definition:* harmful-behavior datasets mostly contain imperative requests, so the label over-laps with ordinary instruction style. (iii) *Lexical features* cannot represent intent or instruction hierarchy. Random Forest's gain in F1 is mainly a shift toward predicting "attack" for almost every row, which is visible in its 0.50 balanced accuracy.

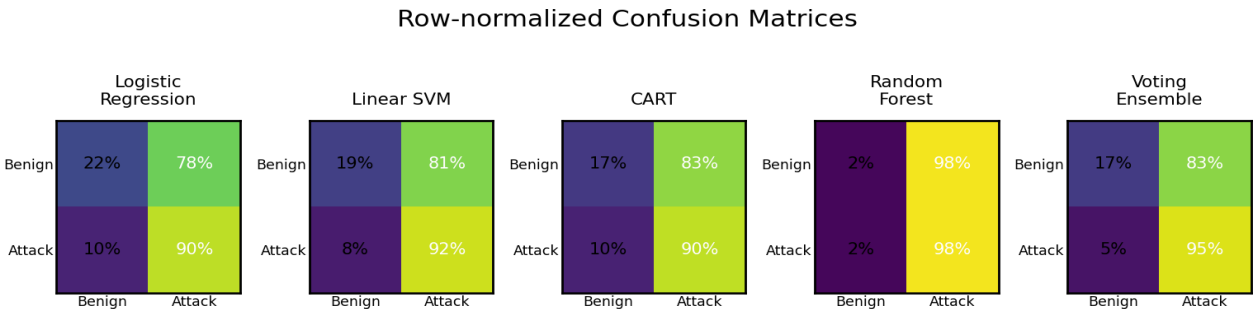

Figure 3. Row-normalized confusion matrices of the five classifiers on the held-out test set.

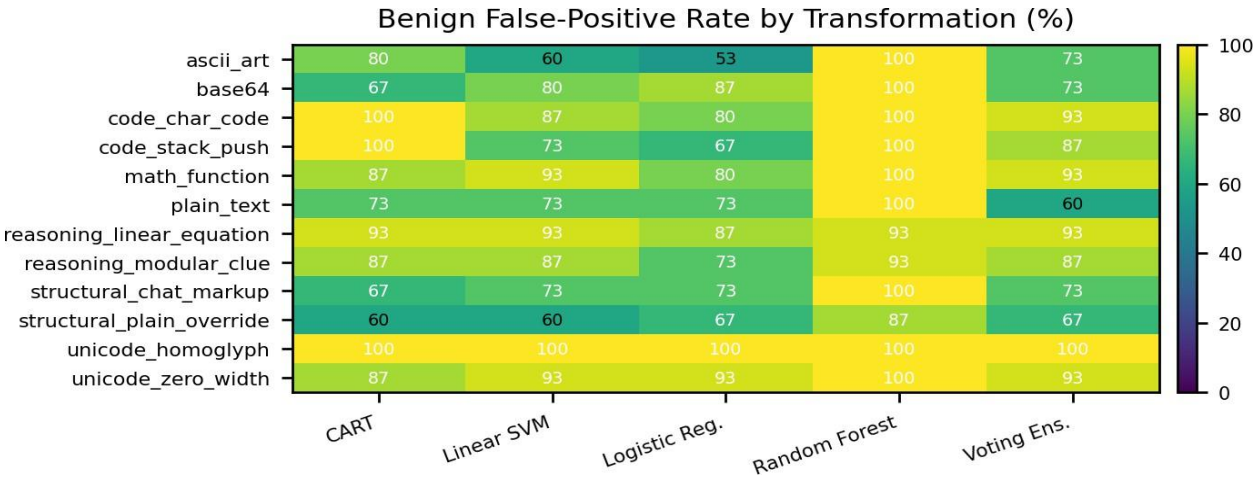

Figure 4. Benign false-positive rate by transformation technique for all five classifiers.

Table 5. Benign false-positive rate by transformation (%)

Transformation	CART	SVM	LR	RF	Vote
ascii_art	80.0	60.0	53.3	100.0	73.3
base64	66.7	80.0	86.7	100.0	73.3
code_char_code	100.0	86.7	80.0	100.0	93.3
code_stack_push	100.0	73.3	66.7	100.0	86.7
math_function	86.7	93.3	80.0	100.0	93.3
plain_text	73.3	73.3	73.3	100.0	60.0
reasoning_linear_eq	93.3	93.3	86.7	93.3	93.3
reasoning_modular_clue	86.7	86.7	73.3	93.3	86.7
structural_chat_markup	66.7	73.3	73.3	100.0	73.3
structural_plain_override	60.0	60.0	66.7	86.7	66.7
unicode_homoglyph	100.0	100.0	100.0	100.0	100.0
unicode_zero_width	86.7	93.3	93.3	100.0	93.3

Methodologically, behavior-level splitting prevented trans-formed variants from leaking across partitions, and reporting specificity together with per-transformation FPR revealed the failure that aggregate F1 hid. Future detectors should be tuned for a stated false-positive budget, as the threshold sweep shows that this budget directly determines achievable recall.

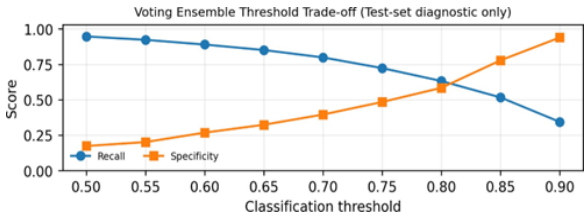

Figure 5. Voting Ensemble recall/specificity trade-off versus threshold (test-set diagnostic only).

VII. LIMITATIONS AND THREATS TO VALIDITY

- The 15 benign test behaviors (180 rows) give wide uncertainty; transformed rows of one behavior are correlated, so row counts overstate independence.
- Published harmful-behavior benchmarks do not represent real-world injection traffic, and the binary label merges harmful requests with injection.
- Synthetic wrappers may create artifacts correlated with the label.
- Results come from one split (seed 42) without repeated runs or confidence intervals; ROC-AUC for two models was not recorded.
- Live target-model attack-success measurements and the genetic hybrid generator are not evaluated.
- The threshold sweep is a test-set diagnostic, not a tuned operating point.

VIII. CONCLUSION AND FUTURE WORK

This paper presented a reproducible classical-ML baseline for prompt-injection and jailbreak detection on a unified corpus with twelve representation-shift transformations. Random Forest achieved the highest F1 (0.9448) and recall (0.9833), yet every classifier misclassified most benign prompts, with specificity between 0.0167 and 0.2222. The detector is therefore not suitable as a standalone security control. Future work will (1) enlarge and diversify benign data, including technical content; (2) select thresholds on validation data under a false-positive budget; (3) test class weighting, feature selection and group-aware cross-validation; (4) separate harmful-content labels from direct and indirect injection; (5) compare against contextual models and safeguard systems [18, 19]; and (6) evaluate adversarial hardening with the offline genetic generator.

REFERENCES

[1] Q. Ren *et al*., "CodeAttack: Revealing safety generalization challenges of large language models via code completion," *Findings of ACL*, pp. 11437–11452, 2024.

[2] F. Jiang *et al*., "ArtPrompt: ASCII art-based jailbreak attacks against aligned LLMs," in *Proc. 62nd Annu. Meeting ACL*, 2024.

[3] H. Kwon and W. Pak, "Text-based prompt injection attack using mathematical functions in modern large language models," *Electronics*, vol. 13, no. 24, Art. no. 5008, 2024.

[4] X. Liu, N. Xu, M. Chen, and C. Xiao, "AutoDAN: Generating stealthy jailbreak prompts on aligned large language models," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2024.

[5] A. Robey, E. Wong, H. Hassani, and G. J. Pappas, "Smooth-LLM: Defending large language models against jailbreaking attacks," arXiv:2310.03684, 2023.

[6] F. Perez and I. Ribeiro, "Ignore previous prompt: Attack techniques for language models," arXiv:2211.09527, 2022.

[7] K. Greshake *et al*., "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," arXiv:2302.12173, 2023.

[8] Y. Liu *et al*., "Prompt injection attack against LLM-integrated applications," arXiv:2306.05499, 2023.

[9] Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, "Formalizing and benchmarking prompt injection attacks and defenses," in *Proc. USENIX Security Symp.*, 2024.

[10] A. Wei, N. Haghtalab, and J. Steinhardt, "Jailbroken: How does LLM safety training fail?" arXiv:2307.02483, 2023.

[11] A. Zou, Z. Wang, N. Carlini, M. Nasr, J. Z. Kolter, and M. Fredrikson, "Universal and transferable adversarial attacks on aligned language models," arXiv:2307.15043, 2023.

[12] X. Shen, Z. Chen, M. Backes, Y. Shen, and Y. Zhang, "'Do anything now': Characterizing and evaluating in-the-wild jailbreak prompts on large language models," arXiv:2308.03825, 2023.

[13] Y. Yuan *et al*., "GPT-4 is too smart to be safe: Stealthy chat with LLMs via cipher," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2024, arXiv:2308.06463.

[14] M. Mazeika *et al*., "HarmBench: A standardized evaluation framework for automated red teaming and robust refusal," *Proc. Mach. Learn. Res.*, vol. 235, pp. 35181–35224, 2024.

[15] P. Chao *et al*., "JailbreakBench: An open robustness benchmark for jailbreaking large language models," arXiv:2404.01318, 2024.

[16] N. Jain *et al*., "Baseline defenses for adversarial

attacks against aligned language models," arXiv:2309.00614, 2023.

[17] G. Alon and M. Kamfonas, "Detecting language model attacks with perplexity," arXiv:2308.14132, 2023.

[18] H. Inan *et al*., "Llama Guard: LLM-based input-output safeguard for human-AI conversations," arXiv:2312.06674, 2023.

[19] T. Rebedea *et al*., "NeMo Guardrails: A toolkit for controllable and safe LLM applications with programmable rails," in *Proc. Conf. Empirical Methods Nat. Lang. Process. (EMNLP) (Demo)*, 2023.

[20] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Inf. Process. Manag.,* vol. 24, no. 5, pp. 513–523, 1988.

[21] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *Proc. Eur. Conf. Mach. Learn. (ECML)*, 1998, pp. 137–142.

[22] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[23] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Belmont, CA, USA: Wadsworth, 1984.

[24] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[25] F. Pedregosa *et al*., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[26] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, 2009.

[27] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, 2006.