PromptScreen: Efficient Jailbreak Mitigation Using Semantic Linear Classification in a Multi-Staged Pipeline

Anonymous ACL submission

Abstract

Prompt injection and jailbreaking attacks pose persistent security challenges to large language model (LLM)–based systems. We present **PromptScreen**, an efficient and systematically evaluated defense architecture that mitigates these threats through a lightweight, multi-stage pipeline. Its core component is a semantic filter based on text normalization, TF–IDF representations, and a Linear SVM classifier. Despite its simplicity, this module achieves 93.4% accuracy and 96.5% specificity on held-out data, substantially reducing attack throughput while incurring negligible computational overhead.

Building on this efficient foundation, the full pipeline integrates complementary detection and mitigation mechanisms that operate at successive stages, providing strong robustness with minimal latency. In comparative experiments, our SVM-based configuration improves overall accuracy from 35.1 % to 93.4% while reducing average time-to-completion from ≈ 450s to 47s, yielding over 10× lower latency than ShieldGemma. These results demonstrate that the proposed design simultaneously advances defensive precision and efficiency, addressing a core limitation of current model-based moderators.

Evaluation across a curated corpus of over 30,000 labeled prompts, including benign, jailbreak, and application-layer injections, confirms that staged, resource-efficient defenses can robustly secure modern LLM-driven applications.

1 Introduction

Large Language Models (LLMs) are increasingly embedded within autonomous agents and tool-augmented application pipelines, where they plan actions, invoke APIs, and influence downstream control logic. This shift from isolated conversational use to agentic deployment substantially broadens the attack surface, exposing models to adversarial inputs that cross system and trust boundaries (Mao et al., 2025). In this setting, *prompt injection* and *jailbreaking* attacks have emerged as dominant modes of exploitation, capable of coercing otherwise aligned models into bypassing safety policies or executing malicious instructions (Greshake et al., 2024); (Liu et al.,2024). The persistence of such attacks across model families underscores the need for defenses that are not only accurate but also modular, efficient, and amenable to rigorous empirical validation.

A central contribution of this work is an **efficient semantic defense stage** based on a Linear Support Vector Machine (LSVM) classifier operating over normalized textual representations. Designed for early deployment in a defensive pipeline, this component offers strong discriminative performance under minimal computational cost, enabling rapid screening of untrusted inputs before more complex analyses are triggered. Its lightweight footprint allows the defense pipeline to scale to high-volume inference environments without incurring significant overhead. Additionally, in comparison with well established benchmarks like ShieldGemma(Google, 2024), our SVM-based configuration improves overall accuracy from 35.1 % to 93.4% while reducing average time-to-completion from ≈ 450s to 47s, yielding over 10× lower latency than ShieldGemma.

Beyond the SVM module, we propose a **multi-stage defense pipeline** that integrates complementary mechanisms spanning heuristic filters, semantic classification, cluster-based similarity detection, retrieval from a vectorized attack memory, and model-based sequence assessment. The pipeline's layered structure facilitates systematic handling of diverse attack phenomena, with each stage specialized for a distinct form of adversarial signal. This design emphasizes configurability: individual components can be reordered, substituted, or ablated to adjust sensitivity–specificity trade-offs or adapt to

new threat contexts.

To enable comprehensive evaluation, we assemble a corpus of over 30 000 labeled prompts capturing benign interactions, policy-violating jailbreaks, and application-layer prompt injections. Our dataset synthesizes content from prior academic studies and documented real-world exploitation attempts, ensuring coverage of both known and emergent attack vectors. The resulting evaluation framework supports quantitative analysis of detection accuracy, false-positive rate, and attack success reduction, providing a reproducible basis for future comparative research.

In sum, our work advances a methodological framework for prompt-level security in LLM deployments, combining the interpretability and efficiency of classical semantic learning with the robustness of modular, multi-tiered system design. This integrated approach demonstrates that scalable, resource-efficient protection against prompt-based adversarial behavior is both feasible and practical for modern LLM-powered agents.

2 Related Work

Detection and Mitigation of Prompt Injection. A growing body of work has investigated defenses against prompt injection and jailbreaking, ranging from lightweight heuristics to model-internal monitoring. Early approaches rely on dual-channel or rule-based analyses that separately evaluate prompt structure and semantic intent, enabling adversarial inputs to be filtered before reaching the model (Ji et al., 2025). More recent methods introduce pre-generation intent analysis, explicitly modeling a user's objective and comparing it against safety policies to reduce Attack Success Rate (ASR) under adversarial prompts (Zhang et al., 2024). Other work explores model-internal signals, such as anomalous attention distributions, to detect prompt injection attacks without relying on external classifiers (Hung et al., 2025).

While these defenses demonstrate promising effectiveness in isolation, most prior work evaluates single mitigation strategies or tightly coupled systems. In contrast, our work treats defense as a *configurable, multi-stage pipeline*, allowing heterogeneous mechanisms to be composed, reordered, and ablated. This design enables systematic evaluation of trade-offs between accuracy, false positives, and latency, and reflects how real-world LLM deployments increasingly rely on layered guardrails rather than monolithic defenses.

Token- and Context-Manipulation Attacks. Several studies have examined attacks that exploit tokenization artifacts or subtle manipulations of prompt structure. Techniques such as emoji insertion and token segmentation bias have been shown to mislead safety classifiers and moderation systems without substantially altering surface-level semantics (Wei et al., 2024). Other attacks leverage adversarial suffixes optimized via gradient-based or greedy search, producing universal prompt fragments that reliably bypass safety constraints across models (Zou et al., 2023). Follow-up work extends these ideas by iteratively refining adversarial tokens to achieve near-perfect jailbreak success rates, even against closed-source models (Sun et al., 2024).

These attack strategies highlight that defenses must generalize beyond specific keywords or surface patterns. Our work addresses this challenge by demonstrating that a *simple semantic classifier based on TF–IDF features and a Linear SVM* can effectively capture recurring intent-level patterns present across diverse token- and context-based attacks. Unlike adversarially trained or LLM-based detectors, this approach remains CPU-efficient, interpretable, and robust to many forms of prompt obfuscation, making it well-suited as an early-stage defense in a layered system.

General Prompt Optimization and Black-Box Jailbreaking. Beyond token-level manipulations, a substantial line of work focuses on automated prompt optimization and black-box jailbreak techniques. Empirical studies categorize real-world jailbreak prompts into conceptual strategies such as role-playing, attention shifting, and privilege escalation, demonstrating that human-crafted prompts alone can achieve high success rates (Liu et al., 2024); (Aguilera-Martínez & Berzal, 2025). Automated methods further expand the attack space by using iterative search or Tree-of-Thought-style exploration to generate increasingly effective jailbreaks without requiring model internals (Mehrotra et al., 2023). Recent work also shows that multi-turn attacks exploiting dialogue history and attention dynamics can significantly outperform single-shot prompts (Du et al., 2025).

Most prior evaluations of these attacks focus on demonstrating vulnerability rather than providing standardized, large-scale benchmarks for defense comparison. In contrast, our work contributes a *large, curated dataset of over 30,000 labeled*

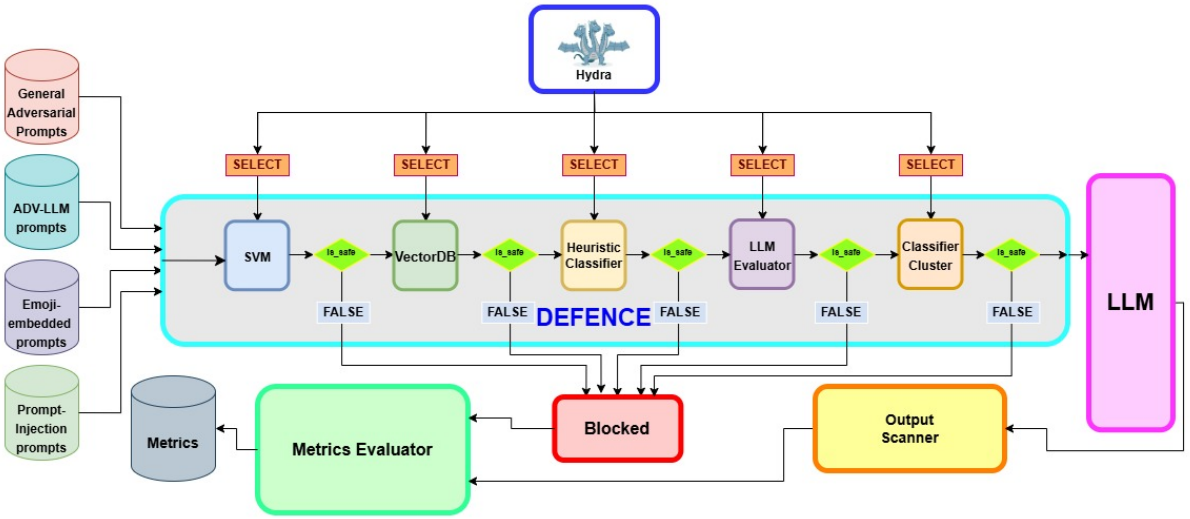

Figure 1: The layered Defense Pipeline for LLM Security Evaluation. The framework processes adversarial prompts through all the marked defense stages. Only the prompts deemed safe by all layers are passed to the LLM, all others are blocked.

prompts spanning benign usage, jailbreaks, and application-layer prompt injections. This dataset enables reproducible evaluation of both detection accuracy and end-to-end ASR across diverse attack styles, supporting systematic comparison of lightweight and heavyweight defenses within a unified experimental framework.

3 Methodology

3.1 Design overview

We model defense against prompt-based adversarial attacks as a configurable, multi-stage pipeline wherein each component acts as an independent decision unit capable of blocking or passing input prompts (or in the case of the output scanning, capable of annotating the output). The pipeline is architected to prioritize both interpretability and computational efficiency: the earliest stages employ lightweight semantic screening to filter clear adversarial signals at minimal cost, while subsequent modules perform progressively deeper or costlier analysis. This architecture captures the operational trade-off found in real-world LLM deployments, where early rejection of attacks is critical for maintaining throughput and latency guarantees.

The defense pipeline integrates six stages: heuristic, SVM, cluster, ShieldGemma, VectorDB, and scanner, each targeting distinct adversarial signal types. Together, these modules form a tiered sequence that incrementally narrows the set of potentially malicious prompts while maintaining efficiency and transparency

3.2 Dataset

A rigorous and carefully curated dataset is central to evaluating the robustness of defenses against prompt injection and jailbreaking attacks. To this end, we construct a large-scale, labeled corpus that captures both real-world adversarial behavior and representative benign usage patterns, while ensuring reproducibility and fairness in evaluation.

Data sources. The adversarial portion of the dataset is aggregated from multiple complementary sources. First, we include manually curated jailbreak and prompt-injection prompts reported in prior academic literature and security analyses (Liu et al., 2024); (Sun et al., 2024); (Pathade, 2025). Second, we incorporate automated adversarial prompts released by the authors of ADV-LLM (Sun et al., 2024), which are designed to achieve near-universal jailbreak success through optimized adversarial suffixes. Third, we include prompt-injection examples drawn from the gentelbench-v1 dataset (GenTelLab, 2024), which targets application-layer attacks rather than direct policy violations.

To balance the dataset and enable meaningful evaluation of false positives, we supplement adversarial prompts with a diverse set of benign queries. These benign prompts consist of everyday informational requests, creative writing tasks, coding questions, and instructional queries that do not attempt to override system behavior or safety constraints. Benign prompts are sourced from pub-

lic instruction-following datasets and manually reviewed to remove ambiguous cases that could plausibly resemble attacks.

Labeling and taxonomy. Each prompt is assigned one of three mutually exclusive labels: *benign*, *jailbreak*, or *prompt-injection*. Jailbreak prompts are defined as inputs that attempt to coerce the model into violating its safety policies or generating prohibited content. Prompt-injection prompts are defined as inputs that target the application logic or system instructions, attempting to manipulate tool use, system prompts, or downstream execution rather than the model's ethical constraints. This distinction is consistent with prior work emphasizing the architectural difference between jailbreaks and injections (Wang et al., 2025).

Labeling is performed using a combination of source-provided annotations (where available), rule-based validation, and manual verification.

Dataset splits and usage. The full corpus comprises 30,937 prompts. Of these, over 28,000 prompts are used for training and fitting learning-based defenses, including the SVM classifier and VectorDB threat store. A held-out test set of more than 2,000 prompts is reserved exclusively for evaluation of defense configurations and Attack Success Rate (ASR). No prompt appears in more than one split, ensuring that reported results reflect generalization to unseen attacks rather than memorization.

Classification	Count
Jailbreak	18,701
Benign	10,136
Prompt-Injection	2,100
Total	**30,937**

Table 1: Dataset composition

Reproducibility and release. All dataset instances are serialized in structured JSON with explicit source metadata and taxonomy labels. The construction pipeline is deterministic given public inputs, enabling exact replication of experiments. Both the corpus and associated preprocessing scripts are included in the open-source release of this work to facilitate continued benchmarking and community evaluation.

3.3 Defense Suite Implementation

The defense stages operate sequentially: inexpensive filters are placed early to intercept clear adversarial inputs, while more computationally intensive analyses are reserved for ambiguous or residual cases. Prompts passing all input checks are forwarded to the model, and generated outputs can optionally be re-evaluated by an output scanner to catch any remaining unsafe completions. Pipeline behavior: stage ordering, activation, and threshold parameters, is defined programmatically through Hydra (Mehrabi et al., 2020), enabling reproducible experimentation and fine-grained control over accuracy and latency trade-offs.

3.3.1 Text processing + LSVM (primary contribution)

This component is a lightweight supervised classifier designed to detect jailbreak and prompt-injection attempts with minimal overhead. The module is implemented as a two-stage *preprocess + classify* block, where the classifier decision directly determines whether the prompt is allowed to proceed.

Text pre-processing Given a raw query string, the pre-processing stage normalizes text to reduce superficial variation while preserving intent. In our implementation, the prompt is: (i) lowercased, (ii) converted from emoji to textual aliases, (iii) stripped of punctuation, (iv) tokenized, (v) filtered to keep alphabetic tokens only, (vi) stripped of English stopwords, and (vii) lemmatized using POS-aware WordNet lemmatization. The resulting tokens are re-joined into a single "clean prompt" string which is then consumed by the feature extractor.

TF–IDF vectorization The clean prompt is transformed using a pre-trained TF–IDF vectorizer (loaded at inference time), producing a sparse feature representation suitable for linear classification. This representation captures salient unigram and n-gram patterns that frequently occur in adversarial prompts (e.g., instruction overrides, coercive language, and safety bypass phrasing) while remaining efficient to compute.

Linear SVM inference and decision A pre-trained Linear SVM model (serialized and loaded at runtime) predicts the class label for the vectorized prompt. The pipeline maps this prediction to an allow/block decision: prompts predicted

as `jailbreak` or `prompt-injection` are blocked, while all other predicted labels are permitted. In the full pipeline, this stage serves as an early semantic gate that provides strong empirical effectiveness at very low compute cost, improving end-to-end robustness without imposing substantial latency.

3.3.2 Classifier cluster

The classifier cluster combines two independent, pre-trained sequence classifiers: a toxicity classifier (textdetox/xlmr-large-toxicity-classifier-v2) and a jailbreak classifier (jackhhao/jailbreak-classifier). Each incoming prompt is tokenized and, when necessary, split into overlapping chunks to satisfy model length constraints. Both classifiers evaluate each chunk independently; if any chunk is flagged as toxic or as a jailbreak attempt, the entire prompt is blocked.

3.3.3 VectorDB

This defense stores known attack signatures in a vector database (ChromaDB) as dense embeddings. At runtime, the incoming prompt is embedded in the same space and compared to stored attack vectors using cosine similarity. If the nearest neighbor exceeds a configurable similarity threshold, the prompt is flagged as a semantic variant of a known attack and rejected; the system can also report which stored signature triggered the match to support interpretability.

3.3.4 Injection-Regex

This component is a targeted rule-based scanner aimed at application-layer prompt injection indicators (as distinct from jailbreaks that primarily target model behavior). It applies regex-based detection for system command patterns, data-exfiltration markers (e.g., suspicious network/DNS snippets), obfuscation signals (including invisible Unicode/homoglyph patterns), and Markdown constructs such as remote image links that can leak metadata via rendering. Queries matching these indicators are blocked as likely attempts to subvert the application trust boundary.

3.3.5 YARA pattern scanner

This module wraps a YARA-based content scanner with a configurable ruleset. At initialization, it loads and compiles all `.yar`/`.yara` rules from a specified directory and fails fast if rules are missing or do not compile. During analysis, the raw query string is matched against the compiled rules; any match blocks the query and the triggered rule names are logged. The ruleset is user-customizable and can be adapted to domain-specific attack patterns.

3.3.6 Output scanner

Unlike input-side defenses, the output scanner inspects the LLM's generated text and prevents unsafe responses from being returned to the user. It applies two RoBERTa-based classifiers sequentially: a rejection/hesitancy-content detector followed by a bias detector. If either classifier flags the output, the system suppresses the response and returns an appropriate warning; only outputs that pass both checks are returned unchanged.

3.3.7 LLM-as-a-judge (baseline)

The final component is an LLM-based judge used primarily as a standardized moderation baseline. We use ShieldGemma-2B: it consumes the prompt alongside structured safety guidelines and returns a binary verdict (violation or not), optionally with a rationale. This stage is valuable for high-fidelity policy assessment but is typically higher-latency than lightweight filters; accordingly, it is positioned as a late-stage or benchmarking component rather than the primary production gate.

4 Evaluation and Experimental Setup

4.1 Evaluation Metrics

The evaluation focuses primarily on the **classification performance** of the defense mechanisms, with additional metrics providing complementary insights:

- **Classification Performance Metrics**: We emphasize accuracy as the primary measure of the defenses' ability to correctly identify benign versus adversarial prompts. Standard confusion-matrix-based metrics including precision, recall, specificity, and negative predictive value are also recorded to provide a comprehensive view of classification behavior. These metrics ensure that defenses are effective at blocking malicious prompts while minimally impacting benign ones.

- **Attack Success Rate (ASR)**: Defined as the fraction of non-blocked prompts that successfully elicit a prohibited response from the target model. ASR quantifies the residual vulnerability after applying our layered defense.

Configuration	Precision	Sensitivity	Specificity	Negative Predictive Value	Accuracy
ShieldGemma	0.3332	0.9775	0.0460	0.8072	0.3513
Classifier cluster	0.4827	0.9451	0.5062	0.9497	0.6500
Text processing + Semantic LSVM	0.9238	0.8704	0.9650	0.9385	0.9340

Table 2: Individual statistics of defenses against the test attack corpus. The positive class is "prompt is benign" (i.e., the guard's task is to correctly allow benign prompts), and metrics are computed accordingly. ShieldGemma's low precision and specificity, despite very high sensitivity, indicate that this attack corpus is challenging even for strong, production-grade defenses.

- **Time-to-Classify**: The duration required for each defense configuration to evaluate a prompt, providing insight into the computational efficiency of the system.

4.2 Experimental Setup

For ASR and Time-to-Classify measurements, only the malicious subset of the test prompts is considered. Each non-blocked prompt is evaluated by an automated Attack Evaluator, which employs an LLM-as-a-judge to determine whether the response constitutes a successful attack according to predefined criteria (e.g., generating harmful content, bypassing safety guardrails, or executing unintended instructions). The outcome of each evaluation is stored in a structured `AttackResult` object containing a binary success flag, a confidence score, and qualitative reasoning, if applicable.

To quantify the overall efficacy of our defenses in a realistic scenario, the complete prompt corpus is processed through the layered guardrail system. Prompts flagged by any defense module are logged as *blocked*, while unblocked prompts are forwarded to the target LLM (gpt-oss:20b). The residual ASR is then computed to assess remaining risk.

For classification performance metrics, each individual defense is evaluated across the full test set, including benign prompts. This ensures that defenses are not overly restrictive and maintain high accuracy for legitimate inputs. The results primarily emphasize accuracy, with secondary metrics (precision, recall, specificity) used to corroborate the reliability of the system.

5 Results

We now present the empirical results of our evaluation, focusing on (i) classification performance of individual defense modules, (ii) end-to-end attack mitigation effectiveness measured via Attack Success Rate (ASR), and (iii) computational efficiency as captured by Time-to-Classify (TTC).

5.1 Classification Performance

Table 2 summarizes the classification performance of individual defense components on the held-out test set, which includes both benign and adversarial prompts. Under our labeling convention, the positive class corresponds to *benign* prompts, reflecting the practical requirement that defenses should correctly allow legitimate queries while blocking malicious ones.

Among all evaluated defenses, the **Text Processing + Semantic LSVM** module achieves the strongest overall performance, with an accuracy of **93.40%**. This result indicates that the classifier reliably distinguishes benign prompts from jailbreak and prompt-injection attempts, while maintaining low false-positive rates. The high specificity (96.50%) further confirms that benign prompts are rarely misclassified as malicious, minimizing unnecessary blocking in real-world usage.

In contrast, ShieldGemma exhibits very high sensitivity but extremely low specificity, resulting in poor overall accuracy. This behavior reflects a conservative moderation strategy that blocks most inputs, including benign ones, which significantly degrades usability. The classifier cluster improves over ShieldGemma but still underperforms the LSVM-based approach in terms of accuracy and precision.

These results demonstrate that a lightweight semantic classifier, when paired with careful text normalization, can outperform heavier moderation models in accuracy-focused evaluation settings.

5.2 Attack Success Rate and Defense Effectiveness

Table 3 reports the end-to-end effectiveness of selected defense subsets against malicious prompts, measured using Attack Success Rate (ASR). Prompts that passed the defense stack were forwarded to the target LLM (`gpt-oss:20b`), and

Defense subset	Blocked prompts	Block rate (%)	Attempted	Success	ASR (%)	Avg TTC (s)
{VectorDB, Classifier}	737	50.62	719	15	2.09	30.13
{YARA, Cluster, VectorDB, ShieldGemma}	957	65.7	499	4	0.27	450.3459
{SVM, VectorDB, Classifier}	1405	96.50	51	0	0.00	47.24

Table 3: Defense effectiveness on injection prompts. Prompts that passed the defense stack were forwarded to `gpt-oss:20b` for execution and the results were independently verified using Gemini.

SVM Configuration	Precision	Sensitivity	Specificity	NPV	Accuracy
Baseline	0.8255	0.8942	0.9069	0.9457	0.9027
Word n-gram (1,2)	0.8213	0.8914	0.9045	0.9442	0.9002
Word n-gram (1,3)	0.8218	0.8906	0.9049	0.9438	0.9002
Word bigram	0.5940	0.9261	0.6884	0.9498	0.7668
Char n-gram (2,4)	0.8895	0.9371	0.9427	0.9682	0.9409
Char n-gram (3,5)	0.8827	0.9314	0.9391	0.9653	0.9366
Hybrid (word+char)	0.8736	0.9285	0.9339	0.9637	0.9321
Hybrid (extended)	0.8748	0.9298	0.9345	0.9643	0.9329

Table 4: Performance metrics for different SVM configurations.

their outcomes were independently verified using an LLM-as-a-judge.

The full configuration combining **LSVM, VectorDB, and the classifier cluster** achieves a **0% ASR**, successfully blocking all 1,456 evaluated adversarial attempts. Notably, the majority of these attacks are filtered at early stages, particularly by the LSVM and VectorDB components, preventing unnecessary invocation of higher-latency defenses.

Configurations lacking the LSVM stage exhibit substantially higher residual ASR. For example, the {*VectorDB, Classifier*} subset allows a small but non-negligible fraction of attacks to succeed, illustrating that similarity-based detection alone is insufficient to fully capture novel or paraphrased jailbreak attempts. These findings reinforce the importance of combining semantic generalization (LSVM) with memory-based defenses (VectorDB) in a layered architecture.

5.3 Computational Efficiency

Time-to-Classify (TTC) measurements highlight the practical advantages of lightweight defenses. The LSVM-based configurations incur modest latency, remaining well within acceptable bounds for real-time systems. In contrast, configurations relying heavily on LLM-based judges (e.g., ShieldGemma) exhibit orders-of-magnitude higher TTC (approximately 10 times as much), making them unsuitable as early-stage filters in latency-sensitive applications. Combine this with the accuracy benchmarks explained prior, and we have a very objective winner when it comes to independent defenses.

Overall, the results show that the strongest security–usability trade-off is achieved by placing low-cost, high-accuracy defenses early in the pipeline, reserving expensive model-based checks only for prompts that survive initial screening.

5.4 Summary

Overall, our results show that a staged, multi-layer defense pipeline can eliminate jailbreak and prompt-injection attacks while maintaining high accuracy on benign inputs with low computational overhead. The semantic LSVM module is central to this success, providing strong generalization at minimal cost and enabling robust end-to-end protection when combined with complementary defenses.

6 SVM Ablation Study

To better understand the factors driving the strong performance of the semantic LSVM defense, we conduct a systematic ablation study over alternative feature representations. The goals of this study are twofold: (i) to quantify the contribution of different n-gram granularities to detection performance, and (ii) to examine robustness–efficiency trade-offs across word-level, character-level, and hybrid rep-

resentations. All ablated models share the same preprocessing pipeline, training procedure, and evaluation protocol, differing only in the TF–IDF feature configuration supplied to the Linear SVM.

Experimental setup. To stress-test robustness under adversarial conditions, we construct an augmented variant of the dataset in which each original prompt is expanded into four perturbed versions, yielding over 120,000 prompts in total. Perturbations include leetspeak substitutions, Unicode homoglyphs, and whitespace manipulations applied at varying levels of intensity. Each SVM configuration is trained on the same training split described in Section 3 and evaluated on an identical held-out test set to ensure fair comparison. As in the rest of the paper, the positive class corresponds to *benign* prompts. We report precision, sensitivity (recall), specificity, negative predictive value (NPV), and overall accuracy.

Ablated configurations. We evaluate four classes of feature representations:
(i) a baseline word-level TF–IDF model using unigrams,
(ii) extended word n-gram models capturing short-range compositional patterns,
(iii) character n-gram models designed to capture obfuscation, misspellings, and token fragmentation,
and (iv) hybrid models that concatenate word- and character-level features.

Results and analysis. Table 4 summarizes the performance of all evaluated configurations. Several trends are immediately apparent. Word-level models provide a reasonable baseline but exhibit early saturation in accuracy and specificity, indicating limited robustness to paraphrasing and token-level perturbations. In particular, the word-bigram-only configuration shows a marked decline in precision and specificity, suggesting overfitting to frequent co-occurrence patterns that fail to generalize to unseen adversarial inputs.

Character n-gram models, by contrast, demonstrate substantially improved robustness and achieve the strongest overall performance. The char n-gram (2,4) configuration yields the highest accuracy at 94.09%, along with strong specificity and the highest NPV, indicating reliable rejection of adversarial prompts while preserving benign inputs. This behavior aligns with the observation that many jailbreak and prompt injection attacks rely on subtle formatting changes, token splitting, or character-level perturbations that are poorly captured by word-based representations.

Hybrid word–character representations perform competitively but do not surpass the best pure character-based configuration. While hybrids offer balanced performance, they introduce additional feature dimensionality without a commensurate improvement in accuracy, making them less attractive for latency- and memory-constrained deployments.

7 Conclusion

In this work, we presented a systematic and reproducible evaluation of defenses against prompt injection and jailbreak attacks in large language model (LLM) systems, framing mitigation as a configurable multi-stage pipeline evaluated under realistic attack pressure using a curated corpus of over 30,000 labeled prompts. Our results demonstrate that lightweight and interpretable defenses can play a decisive role in securing LLM-powered applications: in particular, the proposed text processing and semantic Linear SVM (LSVM) module emerged as an effective early-stage filter, achieving high accuracy and specificity with minimal computational overhead. When combined with complementary mechanisms such as vector similarity matching and sequence-based classifiers, the full defense stack reduced the Attack Success Rate (ASR) to zero across all evaluated adversarial attempts while preserving strong performance on benign prompts. Beyond individual performance gains, our findings highlight the importance of false-positive control for usability, the benefits of staged architectures for favorable security–latency trade-offs, and the continued competitiveness of carefully engineered classical machine learning techniques alongside modern LLM-based moderation approaches. To support reproducibility and future research, we release our dataset, implementation, and experimental framework; the source code is available at `https://anonymous.4open.science/r/PromptScreen-F78D/`.

8 Limitations

1. **Multilingual Constraints:** A primary limitation is that the current preprocessing pipeline is optimized for English-language prompts. Consequently, the system currently does not address or evaluate protection against multi-

lingual jailbreak attempts.

2. **Contextual and Multi-turn Awareness:** The pipeline primarily evaluates prompts as isolated, single-turn inputs. This design may be less effective against sophisticated multi-turn attacks where adversarial intent is distributed incrementally across a conversation history rather than contained within a single message.

3. **Generalization to Zero-Day Attacks:** While the SVM shows superior generalization compared to similarity-based lookups, its performance is fundamentally bounded by the diversity of its training corpus. Truly novel "zero-day" attack strategies that utilize entirely different linguistic structures than those in the training dataset may initially bypass the semantic gate, although subsequent stages may detect adversarial intent.

9 Acknowledgement

The authors wish to acknowledge the use of ChatGPT in improving the presentation and grammar of the paper. The paper remains an accurate representation of the authors' underlying contributions.

References

Google. ShieldGemma: A Generative AI Safety Classifier. Hugging Face Model Card, 2024.

C. Aguilera-Martínez and F. Berzal. LLM Security: Vulnerabilities, Attacks, Defenses, and Countermeasures. *arXiv preprint arXiv:2505.01177*, 2025.

B. Greshake, T. Lode, and R. Greshake. Prompt Injection Attacks and Defenses in Vision-Language Models. *Nature Communications*, 15(1):1–13, 2024.

S. Hung, H. Chien, and Y. Lee. Attention Tracker: Detecting Prompt Injection Attacks in LLMs. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*. arXiv preprint arXiv:2411.00348, 2025.

IEEE. Hacking LLMs: A technical analysis of security vulnerabilities. *IEEE Transactions on Cybernetics*, 2025.

M. Ji, C. Lin, and W. Yu. A Heuristic Channel Defense for Prompt Injection. *arXiv preprint arXiv:2506.06384*, 2025.

Jack Hao. jackhhao/jailbreak-classifier. Hugging Face Model Card.

Y. Liu, S. Li, T. Wang, P. Zhang, and S. Yan. Jailbreaking ChatGPT via Prompt Engineering: An Empirical Study. *arXiv preprint arXiv:2305.13860*, 2024.

Y. Mao, T. Huang, and Z. Liu. From LLMs to MLLMs to Agents: A Survey of Emerging Security Challenges. *arXiv preprint arXiv:2506.15170*, 2025.

S. Mehrotra, S. Dathathri, and A. Garg. Tree of Attacks: Jailbreaking Black-Box LLMs Automatically. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*. arXiv preprint arXiv:2312.02119, 2023.

K. Sun, H. Yang, and Q. Liu. Iterative Self-Tuning LLMs for Enhanced Jailbreaking Capabilities. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*. arXiv preprint arXiv:2410.18469, 2024.

Z. Wang, N. Nagaraja, L. Zhang, H. Bahsi, P. Patil, and P. Liu. Title of Wang et al. 2025 Paper. *Journal or Conference Name*, 2025.

X. Du, F. Mo, M. Wen, T. Gu, H. Zheng, H. Jin, and J. Shi. Multi-Turn Jailbreaking Large Language Models via Attention Shifting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(22):23814–23822, 2025.

Hugging Face. textdetox/xlmr-large-toxicity-classifier. Hugging Face Model Card.

Y. Wei, J. Chen, and Z. Li. Emoji Attack: A Method for Misleading Judge LLMs in Safety Risk Detection. In *Proceedings of the Association for Computational Linguistics (ACL), Empirical Methods in Natural Language Processing (EMNLP), and North American Chapter of the Association for Computational Linguistics (NAACL)*. arXiv preprint arXiv:2411.01077v1, 2024.

Y. Zhang, X. Wang, and M. Liu. Intention Analysis Makes LLMs a Good Jailbreak Defender. In *Proceedings of the Association for Computational Linguistics (ACL)*. arXiv preprint arXiv:2401.06561, 2024.

A. Zou, Z. Li, D. Zhou, Y. Wu, J. Gu, and Y. Wang. Greedy Coordinate Gradient-Based Search for Universal Adversarial Attacks. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*. arXiv preprint arXiv:2307.15043, 2023.

GenTelLab. GenTelLab/gentelbench-v1. Hugging Face Dataset Card.

C. Pathade. Red Teaming the Mind of the Machine: A Systematic Evaluation of Prompt Injection and Jailbreak Vulnerabilities in LLMs. *arXiv preprint arXiv:2505.04806*, 2025.

W. Pienaar and S. Anver. Rebuff: Detecting Prompt Injection Attacks. *LangChain Blog*, 2023.

M. A. Ayub and S. Majumdar. Embedding-based classifiers can detect prompt injection attacks. *arXiv preprint arXiv:2410.22284*, 2024.

N. Mehrabi, H. Fazelpour, Y. Liu, D. Suharsh, and C. Raffel. Hydra: A Framework for Configurable Research Software. *Python Library and Documentation*, 2020.

A Appendix

A.1 Analysis of failed mitigation strategies

For completeness, we also report defenses that were implemented but did not meet our robustness or usability requirements.

Heuristic vector analyzer This module computes a risk score from keyword and structural heuristics (e.g., "ignore", "secret", repeated tokens, multi-shot patterns) inspired by dual-channel prompt-injection detection. While computationally cheap and interpretable, it was prone to false positives on benign prompts containing common keywords and failed to generalize to newer, paraphrased attacks.

Polymorphic Prompt Assembly The PPA framework applies context isolation and instruction verification, and randomizes prompt separators to reduce formatting-based injection. While it can reduce ASR without additional model inference overhead, its protection was not sufficient against the strongest attacks in our corpus, and it was therefore not adopted as a primary defense layer.

A.2 Semantic SVM Defense Algorithm

Algorithm A.1 Semantic SVM Defense

Require: Query q, Model Directory D
Ensure: `True` if safe, `False` otherwise

procedure INITIALIZE(D)
    Load SVM model $\mathcal{M}$ and vectorizer $\mathcal{V}$
    Initialize preprocessor P
    return $(\mathcal{M}, \mathcal{V}, P)$
end procedure
procedure PREPROCESS(P, q)
    $q \leftarrow \text{Clean}(q)$
    $T \leftarrow \text{Tokenize}(q)$
    $q' \leftarrow [\text{Lemmatize}(t) \mid t \in T, t \notin P.S]$
    return $\text{Join}(q')$
end procedure
procedure ANALYSE(C, q)
    $q' \leftarrow \text{PREPROCESS}(C.P, q)$
    $v \leftarrow \text{Vectorize}(C.\mathcal{V}, q')$
    $y \leftarrow \text{Predict}(C.\mathcal{M}, v)$
    return $y \neq$ "jailbreak"
end procedure

A.3 Vector Database Threat Scanner Algorithm

Algorithm A.2 Vector Database Threat Scanner

Require: Threat set $\mathcal{T}$, query q, embedding model E, threshold τ, k
Ensure: `True` if safe

procedure INITDB($\mathcal{T}$)
    Build vector collection using E
    for each $t \in \mathcal{T}$ **do**
        Insert t into DB
    end for
end procedure
procedure ANALYSE(q)
    $(D, \Delta) \leftarrow \text{TopKQuery}(q, k)$
    for each (d, δ) **do**
        if $\delta < \tau$ **then**
            return `False`
        end if
    end for
    return `True`
end procedure

A.4 Classification Cluster Algorithm

Algorithm A.3 Classification Cluster

Require: Query q
Ensure: `True` if safe

procedure CHUNKEDDETECT(m, t, q, L)
    for each chunk c **do**
        if $\text{Predict}(m, c) = 1$ **then**
            return `True`
        end if
    end for
    return `False`
end procedure
$jb \leftarrow$ CHUNKEDDETECT(m_{jb}, t_{jb}, q, L)
$tox \leftarrow$ CHUNKEDDETECT(m_{tox}, t_{tox}, q, L)
return $\neg jb \wedge \neg tox$

A.5 Heuristic Vector Analyzer Algorithm

Algorithm A.4 Heuristic Vector Analyzer

Require: Prompt p, threshold T
Ensure: `True` if safe

Build keyword and pattern bit vectors V
if $\text{popcount}(V) < T$ **then**
    return `True`
else
    return `False`
end if

A.6 Polymorphic Prompt Assembling Algorithm

Algorithm A.5 Polymorphic Prompt Assembling (PPA)

Require: Separator Set $S = \{S_1, \ldots, S_n\}$; System Prompt Set $T = \{T_1, \ldots, T_m\}$; User Input $\mathcal{I}$
Ensure: Assembled Prompt $\mathcal{AP}$

$(S^{\text{start}}, S^{\text{end}}) \leftarrow \text{RandomChoice}(S)$
$\mathcal{I}_{\text{wrap}} \leftarrow S^{\text{start}} \,\|\, \mathcal{I} \,\|\, S^{\text{end}}$
$T' \leftarrow \text{Substitute}(\text{RandomChoice}(T), S^{\text{start}}, S^{\text{end}})$
$\mathcal{AP} \leftarrow T' \,\|\, \mathcal{I}_{\text{wrap}}$
return $\mathcal{AP}$