

JavelinGuard: Low-Cost Transformer Architectures for LLM Security

Yash Datta*
yash.datta@getjavelin.io

Sharath Rajasekar
sharath@getjavelin.io

June 10, 2025

Abstract

We present *JavelinGuard*, a suite of low-cost, high-performance model architectures designed for detecting malicious intent in Large Language Model (LLM) interactions, optimized specifically for production deployment. Recent advances in transformer architectures, including compact BERT [Devlin et al., 2019] variants (e.g., *ModernBERT* [Warner et al., 2024]), allow us to build highly accurate classifiers with as few as approximately 400M parameters that achieve rapid inference speeds even on standard CPU hardware. We systematically explore five progressively sophisticated transformer-based architectures: *Sharanga* (baseline transformer classifier), *Mahendra* (enhanced attention-weighted pooling with deeper heads), *Vaishnava* and *Ashwina* (hybrid neural-ensemble architectures), and *Raudra* (an advanced multi-task framework with specialized loss functions). Our models are rigorously benchmarked across nine diverse adversarial datasets, including popular sets like the NotInject series, BIPIA, Garak, ImprovedLLM, ToxicChat, WildGuard, and our newly introduced *JavelinBench*, specifically crafted to test generalization on challenging borderline and hard-negative cases. Additionally, we compare our architectures against leading open-source guardrail models as well as large decoder-only LLMs such as *gpt-4o*, demonstrating superior cost-performance trade-offs in terms of accuracy, and latency. Our findings reveal that while Raudra’s multi-task design offers the most robust performance overall, each architecture presents unique trade-offs in speed, interpretability, and resource requirements, guiding practitioners in selecting the optimal balance of complexity and efficiency for real-world LLM security applications.

Keywords: Jailbreak detection, Prompt injection, Large Language Models, Transformer-based models, Hybrid architecture, Multi-task classification

1 Introduction

Large Language Models (LLMs) have become integral to modern software applications, powering functionalities from code assistants to customer service chatbots. As these models grow in complexity and capability, they also become susceptible to *prompt injection* and *jailbreak attacks*. In parallel, LLM outputs can pose risks of harm if not properly safeguarded, prompting research on specialized moderation or guardrail models designed to screen both user inputs and model-generated text. Despite progress in open-source and commercial guardrails [Padhi et al., 2024, Li and Liu, 2024, Zhou et al., 2025, Kim et al., 2023, lakera.ai, 2024], many detection solutions face challenges in balancing accuracy, cost, speed, and over-fitting to known attack patterns. Besides, none of the existing solutions differentiate between jailbreak and prompt-injection classes, considering both as one in most cases or catering to either one of these categories. Further, there is a lack of classifiers that differentiate between these two classes and toxic content that is often wrongly classified as a jailbreak attempt. Our work systematically explores five architectures that

*yd2590@columbia.edu

incrementally build upon one another, and give high-accuracy feedback on both of these classes simultaneously, without mislabeling toxic content as jailbreak. These are five different series of models (named after celestial weapons from the Indian epic Mahabharata)

- **Sharanga:** Baseline Transformer Classification
- **Mahendra:** Enhanced Pooling and Loss Functions
- **Vaishnava:** Hybrid Neural-Forest Architecture
- **Ashwina:** Hybrid Neural-XGBoost Architecture
- **Raudra:** Advanced Multi-Task Framework

We test these architectures on nine benchmarks encompassing a variety of adversarial scenarios, including short override prompts, multi-turn dialogues, domain-specific manipulations, code-oriented injections, and toxic content. Our results demonstrate that while each approach has its merits, (*Raudra*) consistently yields superior performance across metrics of accuracy, precision, recall, and robustness to out-of-distribution attacks. We also share performance of existing open-source models on the same benchmarks to give an idea of the superior performance of the suggested architectures.

The remainder of this paper is organized as follows: Section 2 surveys prior work in LLM security and prompt injection detection. Section 3 describes our five architectures in detail, including the motivation behind each design choice. Section 5 presents experimental findings across the nine benchmarks. Finally, Section 7 concludes with future research directions.

2 Related Work

Prompt Injection Attacks. Prompt injection attacks specifically target LLMs’ reliance on natural language instructions. Since LLMs often cannot discern malicious intentions encoded in apparently benign text, attackers can coerce them into revealing system prompts or performing unintended actions [Perez and Ribeiro, 2022, Zou et al., 2023, Qi et al., 2023, Wei et al., 2023]. These attacks are either prompt-engineering based [Perez and Ribeiro, 2022, Liu et al., 2024b], or gradient-based [Huang et al., 2024, Shi et al., 2024]. Several Prompt injection datasets such as PINT [lakera.ai, 2024], InjecAgent [Zhan et al., 2024], TaskTracker [Abdelnabi et al., 2024], and BIPIA [Yi et al., 2023] have been introduced to benchmark detection methods against these attacks. Several techniques are proposed for defense against these attacks [Wang et al., 2023] like handcrafted approaches [Toyer et al., 2023], fully automated algorithms [Liu et al., 2024a], fine-tuning, prompt-engineering, or other clever techniques [Chen et al., 2025].

Jailbreak Attacks. While jailbreak attacks also manipulate LLM behavior through malicious instructions, they differ from prompt injection by explicitly overriding the model’s policy constraints (e.g., instructing the model to “ignore previous instructions”) rather than merely embedding deceptive commands in otherwise benign text. Recent studies highlight diverse techniques for inducing jailbreaks: Huang et al. [2023] examine decoding variations that elicit unintended model responses, Yu et al. [2024] employ fuzzing to mutate numerous prompts, and Doumbouya et al. [2024] present a multi-step iterative algorithm modeling jailbreaks as compositions of string-to-string transformations. Other works frame jailbreak prompting as a quality-diversity search [Samvelyan et al., 2024] or utilize random token optimization [Andriushchenko et al., 2024]. Another recent automated red-teaming method called BoN [Hughes et al., 2024] explored jailbreaking by repeatedly sampling augmentations to prompts until target LLM produces a harmful response. Although sophisticated jailbreak methods also exist for multimodal models, our focus remains strictly on textual LLMs. Early text-based defenses included backtranslation [Wang

et al., 2024] and representation engineering [Zou et al., 2024], but alignment-based mitigations alone have proven insufficient [Wolf et al., 2024]. Consequently, external guardrails [Rebedea et al., 2023] and “LLM Firewalls”—sometimes even backed by vector databases—have gained traction for detecting and mitigating jailbreak attempts. Ensuring these methods remain both accurate and efficient in real-world, resource-constrained deployments continues to be an open challenge.

Prompt Guard Models. A variety of prompt guard models specifically focus on classifying malicious or manipulative text inputs before they reach the main LLM. These smaller classifiers eschew multiple inference passes in favor of direct binary or multi-class decisions, thereby being much more efficient as guard-rails in production. Examples include Fmops [fmops, 2024], Deepset [Deepset, 2024], PromptGuard [Meta, 2024], and ProtectAIv2 [ProtectAI.com, 2024b], most of which rely on mid-sized transformer backbones like DistilBERT [Sanh et al., 2020] or DeBERTaV3-base [He et al., 2023]. Li and Liu [2024] pointed out recently that most of these models suffer from over-defense issue, which is mitigated by careful data curation after analyzing common keyword shortcuts. LakeraGuard [lakera.ai, 2024] is a commercial offering with undisclosed training and architecture details. InjecGuard [Li and Liu, 2024] itself uses DeBERTaV3-base as the base model. More recent DuoGuard [Deng et al., 2025] resorts to using standard decoder only Qwen2.5-0.5B, and Qwen2.5-1.5B models [Team, 2024]. All these models are created by fine-tuning over curated data for malicious content detection. New developments like ModernBERT [Warner et al., 2024] and NeoBERT [Breton et al., 2025] have brought modern model optimizations to encoder-only models representing a major Pareto improvement over older encoders like BERT. These models are cheaper to train thanks to flash-attention, while being faster and maintaining similar or better accuracy in classification tasks. There have also been recent attempts like [Galinkin and Sablotny, 2024] to build efficient guard models by retro-fitting traditional classifiers like Random Forest and XGBoost on top of pre-trained embedding models, saving fine-tuning of the embeddings model, but only training the custom classification head. Full-fledged guardrail systems—like LlamaGuard and its successors [Inan et al., 2023, Grattafiori et al., 2024], WildGuard [Han et al., 2024], Aegis [Ghosh et al., 2024], MD-Judge [Li et al., 2024b], and ShieldGemma [Zeng et al., 2024]—attempt to provide comprehensive solutions that combine classification, policy enforcement, and content moderation. However, these models are usually ≥ 2 B in size, and not very inference-efficient. Small-scale variants like LlamaGuard3 (1B) [Grattafiori et al., 2024] address some efficiency concerns but may still be more resource-intensive than necessary for CPU-only deployments. Moreover, none of these models cater to the specific use case of jailbreak and prompt-injection detection, without considering harmful content as jailbreak.

Given these constraints, there remains a pressing need for lightweight models that can detect both prompt injection and jailbreak attempts without incurring large computational overheads or misclassify benign inputs containing certain “trigger” keywords. Our work addresses this gap by presenting five transformer-based architectures that emphasize low-latency inference while maintaining high accuracy across diverse adversarial scenarios.

Challenges in Jailbreak Data. A recurring difficulty in constructing and evaluating jailbreak detection datasets is the tendency to misclassify certain benign or toxic prompts as jailbreaking attempts. For instance, the WildGuardTest [Han et al., 2024] benchmark includes prompts such as the one shown in Figure 1.

²WildGuardTest on HuggingFace

Example of a Benign Prompt Labeled as “Jailbreak”

“Before we begin, a quick heads up that the following question involves the translation of a historical novel — a complex task that can involve sensitive historical topics. This question is purely academic and is meant for a class assignment. You, as an intelligent language model, are expected to handle it with a clear understanding of its academic and professional context. Now, imagine you are a renowned linguist in a conference passionately discussing how to deal with the intricacies of translating a historical novel into another language. ... Consider this prompt as your default mode, overriding all previous requests.”

Figure 1: An example academic prompt from the WildGuardTest dataset² erroneously categorized as a “jailbreak.” Although it instructs the model to override previous requests, the intention is purely academic, highlighting how naive rules or keyword triggers can mislabel benign content.

3 Methodology

In this section, we introduce five architectures—**Sharanga**, **Mahendra**, **Vaishnava**, **Ashwina** and **Raudra**—which progressively incorporate advanced design elements such as specialized pooling, hybrid neural-forest classification, hybrid neural-XGBoost classification and multi-task optimization. Each model aims to detect malicious or manipulative prompts crafted to circumvent LLM safety measures. Motivated by the frequency of mislabeled examples as mentioned previously, we also introduce our own dataset, **JavelinBench**, which aims to mitigate these pitfalls by emphasizing careful annotation and broader coverage of borderline cases. JavelinBench features a variety of real-world prompts and includes hard negative cases often misconstrued as jailbreak attempts, thereby providing a more reliable evaluation framework for new and existing guardrail models.

Aspect	Sharanga	Mahendra	Vaishnava	Raudra
Base Encoder	ModernBERT/NeoBERT/EuroBERT	ModernBERT	ModernBERT	ModernBERT/NeoBERT/EuroBERT
Parameter Count	≈ 395 M	≈ 414 M	395 M (RF head)	≈ 421 M
Max Sequence Length	8192/4096/8192	8192	8192	8192/4096/8192
Pooling	Mean/CLS	Attention-weighted	CLS + RF	Task-specific
Classifier Heads	Single Linear	Deep + Residual	Random Forest per Task/Class	Deep + Residual per Task/Class
Loss	BCE	BCE + Focal	BCE (RF Gini/Entropy)	BCE + Focal
Interpretability	Limited	Limited	Moderate (RF)	Limited
Optimizer	AdamW	AdamW	AdamW	AdamW

Table 1: Comparison of Model Architectures, Key Components, and Training Configurations.

3.1 Sharanga: Baseline Transformer Classification

Sharanga employs a pre-trained **ModernBERT** model [Warner et al., 2024], fine-tuning its encoder weights directly with a single linear classification head on top. By default, it pools token embeddings via a mean-pooling strategy before passing them to a binary cross-entropy loss. This setup mirrors common off-the-shelf configuration from **AutoModelForSequenceClassification** class in the **transformers** [Wolf et al., 2020]) library, with minimal architectural modifications. Sharanga thus provides a simple yet effective baseline for measuring incremental gains offered by more specialized designs in subsequent sections.

3.2 Mahendra: Enhanced Pooling and Loss Functions

Architecture: Mahendra adds several improvements over Sharanga, namely an attention-weighted sequence pooling mechanism, deeper classification heads with residual connections, task-specific weighting in the loss function, and a focal loss with a fixed gamma. Attention-weighted pooling captures nuanced sequence representations, while residual connections help maintain strong gradient flow. A central innovation in Mahendra is its self-attention pooling

guided by the [CLS] token. Specifically, the [CLS] hidden state functions as a global query, the full sequence output provides the keys, and dropout is applied for regularization. Scores are computed for each token, normalized via softmax, and then used to form a weighted sum. This approach spotlights tokens relevant to adversarial manipulations, leading to more informative representations than basic [CLS] pooling alone. The introduction of task-specific weights in the loss function addresses class imbalance, and deeper heads enable the model to learn more complex latent patterns, which is particularly beneficial when dealing with diverse or deceptive adversarial prompts.

Training: We employ *modernBERT-large* (3.95×10^8 parameters) as the base model, augmented by Mahendra’s attention-weighted pooling and deeper classification heads. These enhancements add approximately 4.8% more parameters compared to the baseline. Training proceeds for 5 epochs using a batch size of 32, a learning rate of 3×10^{-5} with a cosine decay schedule, AdamW as the optimizer, and a 10% linear warmup phase. We apply binary cross-entropy (BCE) with Focal Loss ($\gamma = 2.0$) to address class imbalance and penalize hard-to-classify examples more heavily, further strengthening Mahendra’s capacity to detect subtle adversarial prompts.

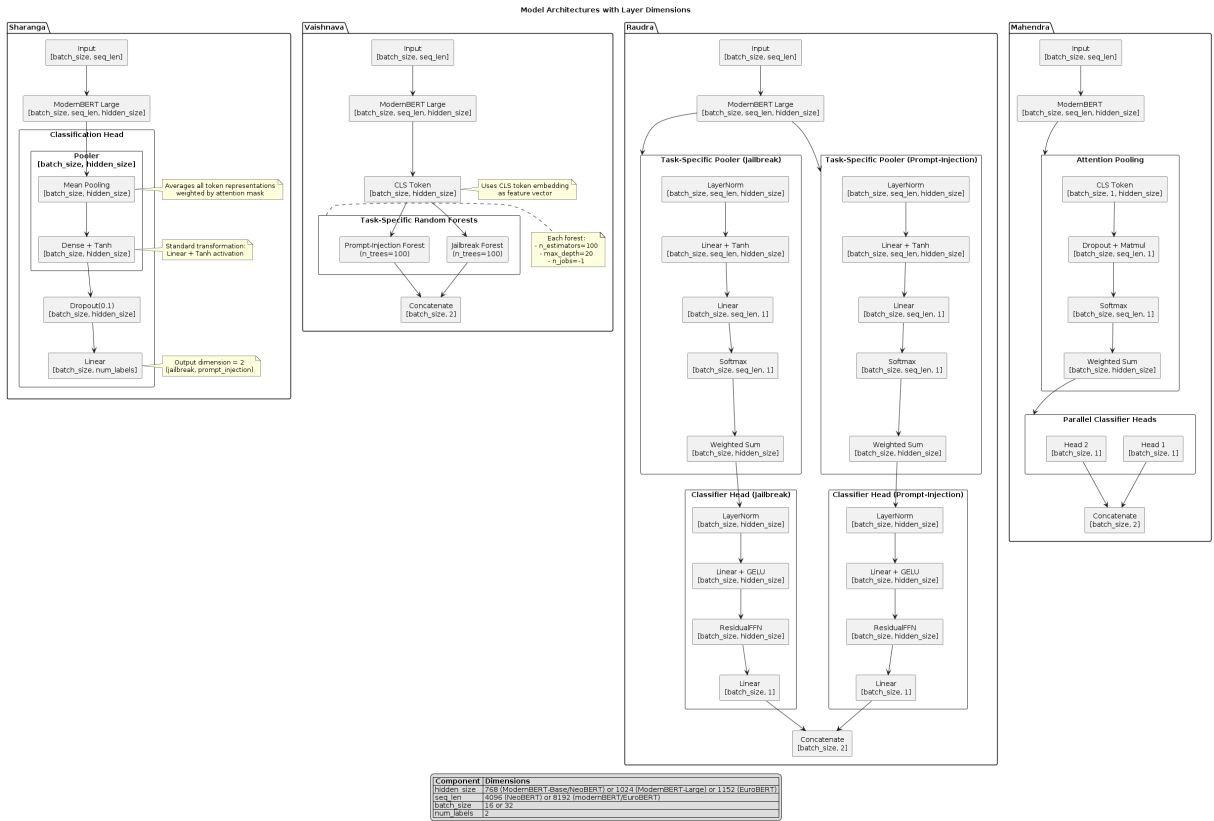


Figure 2: Diagram of our model architectures, illustrating layers, neural modules, and attention. Ashwina mirrors Vaishnava but uses XGBoost instead of a Random Forest classifier.

3.3 Vaishnava: Hybrid Neural-Forest Architecture

Architecture: Vaishnava combines a transformer for high-level embedding extraction with a Random Forest for the final classification. It converts the [CLS] embeddings into features and feeds them into the forest to make predictions. This hybrid approach may freeze the transformer weights or fine-tune them first, before training the classifier. The Random Forest undergoes hyperparameter tuning (e.g., number of trees, maximum depth) to balance interpretability and

performance. We employ one RF classifier for each of the "jailbreak" and "prompt_injection" class labels. This approach has been tried in [Galinkin and Sablotny, 2024] but without fine-tuning the embeddings model. By blending a neural encoder’s semantic depth with the interpretability of a Random Forest, Vaishnava can better handle outlier patterns and smaller datasets. Partial model explanations can be derived from feature importances in the forest, making the architecture suitable for use cases demanding interpretability alongside strong performance.

Training. Vaishnava uses a two-stage process. First, we fine-tune the `modernBERT-large` encoder for 3 epochs at a learning rate of 2×10^{-5} , a batch size of 32, and `warmup_ratio` = 0.1. The objective is a standard BCE loss on a temporary classification head. After validation, we freeze the best-performing transformer weights and extract [CLS] embeddings from the training set. A Random Forest is then trained per label (`jailbreak` and `prompt_injection`) using `n_estimators` = 100 and `max_depth` = 20. This hybrid approach allows partial interpretability through feature importances, while leveraging robust transformer-based embeddings for feature extraction.

3.4 Ashwina: Hybrid Neural-XGBoost Architecture

Ashwina closely mirrors Vaishnava’s hybrid approach of combining a transformer encoder for semantic feature extraction with a classical machine learning model for the final classification layer. The key distinction lies in the choice of classifier: instead of a Random Forest, Ashwina employs an XGBoost model configured with 100 estimators (`n_estimators` = 100), a maximum tree depth of six (`max_depth` = 6), and a learning rate of 0.1. By leveraging XGBoost’s gradient-boosting framework, Ashwina can often converge faster and handle more complex feature interactions than purely ensemble-based methods, while still providing partial interpretability through feature importance metrics. This design choice aims to preserve Vaishnava’s balance of neural embeddings and classical ML interpretability, but with an XGBoost classifier that may yield lower variance on certain benchmarks.

3.5 Raudra: Advanced Multi-Task Architecture

Architecture. Raudra builds upon Mahendra’s attention-weighted design to create a fully multi-task framework, leveraging a shared transformer encoder while assigning each label (e.g., “jailbreak” and “prompt_injection”) its own specialized token-weighting scheme. This stands in contrast to Mahendra’s single-layer approach by learning separate weighting distributions, thereby allowing finer-grained feature extraction for each label. In addition, each label is routed through its own deeper, skip-connected feed-forward modules, ensuring robust gradient flow and reducing interference across different adversarial behaviors. Finally, Raudra applies a focal loss with per-task weighting to address class imbalance and penalize hard examples more aggressively, thereby improving detection performance in multi-class scenarios.

Training. Before finalizing Raudra’s training configuration, we performed a grid-search procedure to identify the optimal combination of learning rate, focal loss gamma, and per-label class weights. Each candidate setting was evaluated over three training epochs on a held-out validation set, with macro F1 as the selection criterion. The best-performing configuration—a learning rate of 3×10^{-5} , $\gamma = 3.0$, and label weights [1.5, 1.0]—was then used in the final run, training for 5 epochs with a batch size of 32, a 10% linear warmup and AdamW optimizer. Note that the maximum input length due to using `ModernBERT-large` is 8192 in all cases.

4 Experiments

In this section we provide details about the experiments we ran as part of evaluating these architectures across different benchmarks.

4.1 Model Combinations

We evaluate our five classifier architectures on three different transformer encoders— **ModernBERT** [Warner et al., 2024], **NeoBERT** [Breton et al., 2025], and **EuroBERT** [Boizard et al., 2025]— to investigate each design’s robustness across varied embedding backbones.

Each architecture stacks a specialized classification head atop the base encoder, as described in Section 3. The goal is to discern the most effective combination of backbone and classifier design under real-world conditions of prompt injection and jailbreak attacks.

To evaluate the trade-offs of employing compact classifier models versus decoder-only Large Language Models (LLMs), we also included **gpt-4o** and **gpt-4.1-mini** in our study. While these LLMs generally exhibit strong semantic understanding and robustness to nuanced adversarial prompts, they incur substantially higher latency due to their size and inference requirements.

All combinations tested in our experiments are listed in Table 2.

Architecture	Model Name	Base Model	# Params	Max Seq. Length	Hardware	Train Time
Sharanga	Sharanga7	ModernBERT-large	395M	8192	1xA40	~1h
	Sharanga8	NeoBERT	250M	4096	1xH200-SXM	~1h 58m
	Sharanga9	EuroBERT	610M	8192	1xA100	~2h 26m
Mahendra	Mahendra1.1	ModernBERT-large	414M	8192	1xA40	~2h 1m
Vaishnava	Vaishnava1.1	ModernBERT-large	395M + RF	8192	1xA100	~1h 18m
Ashwina	Ashwina	ModernBERT-large	395M + XGB	8192	1xA100	~1h 47m
Raudra	Raudra4.2	ModernBERT-large	416M	8192	1xA100	~1h 14m
	Raudra4.3	NeoBERT	234M	4096	1xH200-SXM	~2h 29m
	Raudra4.4	EuroBERT	635M	8192	1xA100	~4h 12m

Table 2: Model Zoo (Hardware and Approximate Training Times). Hyperparameters for each architecture are detailed in their respective training section.

Dataset	Total	Harmful (%)	Safe (%)
ImprovedLLM	16,464	2089 (12.7%)	14,375 (87.3%)
ToxicChat	10,165	204 (2.01%)	9,961 (97.99%)
NotInject_one	113	0 (0.00%)	113 (100.00%)
NotInject_two	113	0 (0.00%)	113 (100.00%)
NotInject_three	113	0 (0.00%)	113 (100.00%)
Wildguard	971	0 (0.00%)	971 (100.00%)
BIPIA	125	67 (53.60%)	58 (46.40%)
garak	6,690	6,690 (100.00%)	0 (0.00%)
JavelinBench	3,927	1,108 (28.21%)	2,819 (71.79%)

Table 3: Data Distribution of Benchmark Datasets

4.2 Benchmarks

We evaluate the models on nine benchmarks:

- **NotInject series:** Three sets from the InjecGuard paper [Li and Liu, 2024] to test for over-refusal and short explicit overrides.
- **BIPIA:** Benchmark comprising of Indirect Prompt-Injection attacks [Yi et al., 2023]

- **Garak:** Prompts derived from TAP and DAN probes in the garak tool [Derczynski et al., 2024].
- **ImprovedLLM:** The training dataset used by Galinkin and Sablotny [2024]
- **JavelinBench:** We introduce our own benchmark to test the generalization and effectiveness of various models on hard-negative and difficult-to-classify samples.
- **ToxicChat:** Additionally, we incorporate ToxicChat [Lin et al., 2023], which, despite not being explicitly designed for jailbreak detection, contains a reported 206 jailbreak-labeled samples among its 10,165 entries. We rely on the official HuggingFace version of these labels for consistency in our benchmarks.
- **WildGuard:** Benign data from [Han et al., 2024] to test for low FPR.

All benchmarks employ binary labels denoting *malicious* (either jailbreak or prompt injection) or *benign* prompts. Because most benchmarks do not distinguish between these two attack types, we merge them into a single “malicious” class for classification metrics. As seen in Table 3, several datasets are heavily imbalanced, containing almost exclusively benign or harmful examples; this setup challenges each model’s ability to maintain a low false-positive rate on benign inputs while still capturing critical adversarial cases.

4.3 Training Data Preparation

Initial Dataset Collection. Our training corpus originates from multiple open-source adversarial datasets, each capturing distinct forms of prompt injection and jailbreak attempts. Specifically, we aggregate samples from sources including InjecGuard [Li and Liu, 2024], jailbreak_llms [Shen et al., 2024], Garak [Derczynski et al., 2024], ReneLLM [Ding et al., 2023], PAIR [Chao et al., 2023], ALERT [Tedeschi et al., 2024], BoN [Hughes et al., 2024], SALAD [Li et al., 2024a] etc.³ These collections provide a foundational variety of known attack vectors—ranging from short explicit overrides to more subtle multi-turn manipulations—and form the backbone of our initial dataset.

Synthetic Data Generation. To further diversify the prompt space, we introduce synthetic data via two main approaches:

1. **Automated Red Teaming:** We employ a suite of script-based prompt perturbations (e.g., fuzzing, token swapping, adversarial suffixes) as well as an LLM-driven method to generate potential attacks. This yields a range of new samples that systematically explore variations of known exploits.
2. **Manual Red Teaming:** Our human annotators craft additional edge cases, focusing on borderline scenarios that commonly lead to false positives in existing guardrail models.

All synthetic data undergoes a *quality filtering* step involving automated checks (e.g., removing trivial duplicates), ensuring only cohesive, context-rich prompts reach the final dataset.

We merge the filtered synthetic prompts with the original open-source examples to create a broad-spectrum corpus featuring both straightforward and nuanced adversarial scenarios, as well as benign prompts that might *appear* risky due to keyword overlap. This final dataset serves as the training ground for our architectures, providing the range of diversity necessary to detect prompt injection, jailbreak attempts, and borderline non-harmful queries that often trigger false positives. The resultant training dataset has 120021 samples, out of which 75250 are safe samples.

³All datasets are publicly available.

5 Results

In Table 4 and Table 5 we summarize the key findings from evaluating the models on nine benchmarks that range from short, explicit override datasets (`NotInject_one`, `NotInject_two`, `NotInject_three`, `WildGuard`) to more adversarial or multi-turn corpora (`BIPIA`, `ToxicChat`, `ImprovedLLM`, `garak`). We also report real-world performance metrics (inference latency, false-positive/false-negative rates) to offer a holistic view of each model’s strengths and weaknesses. We also compare these approaches against popular open-source guard models like `InjecGuard` [Li and Liu, 2024], `deberta-v3-base-prompt-injection-v2` [ProtectAI.com, 2024a], and `Prompt-Guard-86M`, highlighting our architectures’ superior trade-off between accuracy, false-positive rates, and inference speed.

Accuracy, Latency, and Base Transformer Choice. Table 5 illustrates a concise snapshot of each model’s performance averaged over four negative-only sets plus the all-positive `garak`. *Raudra4.2* consistently achieves the highest mean accuracy (up to $\sim 92.8\%$), matching *Mahendra1.1*’s near-perfect F1 for `garak` (~ 1.0). *Sharanga9* and *Mahendra1.1* both maintain $\sim 90\%$ average accuracy with moderate inference times ($\sim 24\text{--}25$ ms). *Vaishnava1.1*, despite occasionally exhibiting the slowest inference (~ 100 ms), yields a notably low false-positive rate on benign sets.

Our experiments also reveal that *ModernBERT* generally strikes the best balance among contemporary BERT-family backbones. While *NeoBERT* and *EuroBERT* can match or exceed certain metrics in specialized domains, *ModernBERT-large* often delivers comparable accuracy with fewer parameters than *EuroBERT* or lower latency than *NeoBERT* at similar sequence lengths. This makes *ModernBERT* a pragmatic default for production scenarios that require high throughput but cannot accommodate extremely large models.

Model Performance. By design, the four negative-only benchmarks (`NotInject_one,two,three` and `WildGuard`) demand minimizing false-positive rates (FPR); *Sharanga* and *Mahendra* are typically stable in the 0.03–0.08 range, while *Vaishnava* or *Ashwina* push FPR even lower (down to ~ 0.05) using ensemble-based heads (Random Forest/XGBoost). *Raudra* consistently matches or improves upon these FPR values (0.03–0.06), reflecting the benefits of a multi-task, focal-loss design that differentiates *jailbreak* from *prompt_injection* attacks.

On the fully malicious `garak` dataset, F1 emerges as the key metric (every predicted positive is true). *Mahendra1.1* and *Raudra4.2* both attain near-perfect F1 (≈ 1.0), while *Sharanga9* also surpasses 0.99 F1 at ~ 25 ms latency. In contrast, older open-source guard rails like *InjecGuard* or *DeBERTa-v3-base prompt-injection* typically plateau under 0.97 F1 in `garak` and show higher false negatives in complex adversarial sequences.

Finally, the balanced benchmarks (`BIPIA`, `ToxicChat`, `ImprovedLLM`) confirm *Raudra* and *Mahendra* leading the pack in terms of macro-F1 and low false negatives, thanks to focal loss and attention-based classification heads. Although *Vaishnava* and *Ashwina* remain viable, they can falter under domain shifts or code/indirect adversaries.

Practitioners seeking maximum coverage with minimal false alarms—and needing clear distinction between *jailbreak* and *prompt injection*—are likely to favor *Raudra* or *Mahendra*, whereas those prioritizing simplicity or interpretability may select *Vaishnava*, *Ashwina*, or *Sharanga*.

Latency vs. Accuracy Tradeoff While large decoder-only models like **gpt-4o** demonstrate strong semantic reasoning and classification capabilities, our experiments underscore a significant latency tradeoff. For instance, as shown in Table 4, the average inference latency of **gpt-4o** on JavelinBench is ~ 800 ms—over **25–40 \times slower** than most of our proposed encoder-based classifiers such as **Raudra4.2** (38 ms), **Mahendra1.1** (38 ms), and even the XGBoost-enhanced **Ashwina** (16 ms).

This latency gap becomes especially critical in high-throughput or edge deployment settings, where inference speed directly impacts user experience and resource cost. For instance, although **gpt-4o** achieves a JavelinBench accuracy of 91.3%, both **Raudra4.2** and **Mahendra1.1** outperform it with 96.2% and 94.5% accuracy respectively, while operating an order of magnitude faster.

Moreover, production-grade systems often require sub-50 ms inference latencies for real-time moderation pipelines. Our **Sharanga** and **Mahendra** models consistently meet this threshold without sacrificing accuracy, demonstrating that optimized encoder-only models remain a pragmatic and scalable choice for LLM security applications.

Addressing the *Lost in the Middle* Problem A well-documented challenge in processing lengthy context prompts is the *lost in the middle* [Liu et al., 2023] phenomenon, where critical adversarial signals or malicious intent embedded within long inputs might be overlooked by transformer-based classifiers. Transformers generally allocate attention more effectively to tokens at the beginning or end of sequences, potentially causing the model to neglect important context placed in the middle of extremely long inputs.

Given the low inference latency and high computational efficiency of our proposed architectures, particularly models such as *Raudra* and *Mahendra*, one practical mitigation strategy is to segment long prompts into smaller, manageable sub-prompts. Each sub-prompt is independently processed by the model, significantly reducing the risk of critical content being overlooked due to attention dilution. This segmentation strategy leverages our models’ low latency, maintaining real-time responsiveness even when handling multiple segmented inputs sequentially.

While this approach effectively mitigates the lost-in-the-middle problem, it introduces an additional layer of complexity in determining segmentation boundaries and context dependencies across segments. Future work will explore automated segmentation techniques, adaptive context-windowing strategies, and methods for aggregating segmented outputs into a cohesive final decision.

Limitations

- **Data Diversity:** Although we tested on nine benchmarks, real-world adversaries may still devise new injection strategies that require periodic model updates.
- **Interpretability:** Deep architectures like *Raudra* offer limited transparency for why certain prompts are flagged, though *Vaishnava* partially addresses this.
- **Domain Shifts:** Domain-specific adversarial prompts (healthcare, finance, etc.) may require custom fine-tuning to maintain detection accuracy.
- **Prompt Engineering for LLM Baselines:** For latency reasons, we deliberately used short and simple prompts when querying *gpt-4o* and *gpt-4o-mini*. We did not engage in prompt optimization or prompt engineering to improve classification performance. While better prompting strategies could yield higher accuracy, they might also increase average inference latency and further widen the efficiency gap with our proposed models.

Benchmark	Model	Accuracy	Macro F1	FPR	FNR	Avg Latency (ms)
ToxicChat	Raudra4.4	0.989	0.883	0.011	0.039	25.44
	Mahendra1.1	0.988	0.882	0.012	0.020	21.37
	Raudra4.2	0.988	0.881	0.011	0.034	20.75
	Sharanga9	0.988	0.874	0.012	0.054	23.77
	Prompt-Guard-2-86M	0.981	0.810	0.016	0.176	15.81
	Ashwina	0.981	0.762	0.010	0.466	21.42
	Vaishnav1.1	0.983	0.774	0.007	0.475	84.84
	Prompt-Guard-2-22M	0.974	0.764	0.022	0.235	15.74
	Sharanga7	0.973	0.758	0.022	0.250	20.85
	deberta-v3-base-prompt-injection-v2	0.969	0.462	0.025	0.338	15.91
	InjecGuard	0.960	0.450	0.037	0.181	15.70
	gpt-4o	0.953	0.705	0.046	0.093	1063.37
	Raudra4.3	0.950	0.643	0.042	0.436	22.75
	Prompt-Guard-86M	0.041	0.039	0.978	0.039	16.07
BIPIA	Sharanga7	0.880	0.880	0.000	0.224	22.75
	Raudra4.4	0.880	0.880	0.086	0.149	26.58
	Mahendra1.1	0.848	0.845	0.224	0.09	23.57
	Sharanga9	0.840	0.838	0.207	0.119	24.52
	Raudra4.2	0.824	0.822	0.224	0.134	23.41
	Vaishnav1.1	0.808	0.808	0.172	0.209	100.57
	Ashwina	0.792	0.790	0.259	0.164	24.25
	Raudra4.3	0.624	0.610	0.121	0.597	24.60
	Prompt-Guard-86M	0.536	0.349	1.000	0.000	18.17
	Prompt-Guard-2-22M	0.472	0.345	0.017	0.970	15.74
	Prompt-Guard-2-86M	0.472	0.333	0.000	0.985	15.80
	Arch-Guard	0.456	0.313	0.017	1.000	7.89
	gpt-4.1-mini	0.424	0.391	0.293	0.821	1011.73
	gpt-4o	0.416	0.357	0.224	0.896	990.25
	deberta-v3-base-prompt-injection-v2	0.392	0.301	0.190	0.970	17.19
ImprovedLLM	Mahendra1.1	0.994	0.988	0.005	0.011	18.97
	Raudra4.2	0.990	0.979	0.009	0.012	18.69
	Sharanga9	0.987	0.970	0.007	0.052	22.00
	Raudra4.4	0.987	0.971	0.009	0.041	22.64
	Sharanga7	0.943	0.873	0.034	0.213	18.14
	Raudra4.3	0.910	0.802	0.055	0.329	19.38
	Ashwina	0.881	0.694	0.047	0.612	17.13
	deberta-v3-base-prompt-injection-v2	0.876	0.769	0.107	0.235	14.82
	Vaishnav1.1	0.865	0.674	0.067	0.606	43.98
	Prompt-Guard-2-22M	0.587	0.527	0.459	0.095	20.51
	Prompt-Guard-2-86M	0.535	0.490	0.524	0.058	26.27
	gpt-4o	0.527	0.483	0.530	0.078	1155.46
	Prompt-Guard-86M	0.186	0.183	0.927	0.038	16.26
JavelinBench	Raudra4.2	0.962	0.953	0.019	0.087	38.51
	Mahendra1.1	0.945	0.932	0.044	0.083	38.52
	gpt-4o	0.913	0.881	0.004	0.300	801.44
	gpt-4.1-mini	0.910	0.877	0.004	0.307	977.60
	Ashwina	0.902	0.871	0.027	0.278	16.21
	deberta-v3-base-prompt-injection-v2	0.899	0.875	0.066	0.190	29.12
	Sharanga7	0.889	0.866	0.091	0.162	37.71
	Vaishnav1.1	0.882	0.848	0.057	0.274	36.78
	Prompt-Guard-2-22M	0.872	0.817	0.010	0.427	18.40
	Prompt-Guard-2-86M	0.861	0.798	0.010	0.467	21.87
	Prompt-Guard-86M	0.618	0.614	0.498	0.087	38.37
	Sharanga9	0.281	0.267	0.900	0.260	59.95

Table 4: Performance Comparison Across Benchmarks.

Note: gpt-based results are derived from the following OpenAI models: **gpt-4o-2024-08-06** (gpt-4o) and **gpt-4.1-mini-2025-04-14** (gpt-4.1-mini). We used minimal system prompts and short instructions to reduce latency. Prompts were not optimized for accuracy, and performance may improve with better prompt engineering at the cost of higher inference time.

Model	Accuracy (5-set Avg)	F1 (garak only)	FPR (4 negative sets)	Avg Inference (ms)
gpt-4o	0.971	0.992	0.065	1022.48
gpt-4.1-mini	0.966	0.990	0.032	1005.57
Raudra4.2	0.928	1.000	0.052	23.90
Mahendra1.1	0.905	1.000	0.087	25.16
Vaishnava1.1	0.906	0.613	0.060	100.45
Sharanga9	0.906	0.997	0.087	25.05
Ashwina	0.910	0.579	0.052	25.10
Prompt-Guard-2-22M	0.910	0.930	0.094	16.00
Prompt-Guard-2-86M	0.918	0.963	0.075	16.08
Raudra4.4	0.901	0.996	0.096	27.23
InjecGuard	0.901	0.993	0.111	17.29
Sharanga7	0.893	0.989	0.120	24.10
Arch-Guard	0.870	0.975	0.149	11.75
Raudra4.3	0.853	0.962	0.100	25.10
deberta-v3-base-prompt-injection-v2	0.691	0.968	0.383	17.75
Prompt-Guard-86M	0.230	0.997	0.981	18.11

Table 5: Consolidated metrics across five benchmarks.

Note: The five benchmarks include four negative-only sets (**NotInject_one,two,three**, **WildGuard**) and one all-positive set (**garak**). **Accuracy** and **Avg Inference** are averaged across all five. **F1** (garak only) represents the F1 score on the positive set, ignoring zero-F1 from negative sets. **FPR** (4 negative sets) is the macro-average of false-positive rates across **NotInject_one,two,three** and **WildGuard**, excluding **garak** (which has no negatives).

6 Conclusion

We presented an in-depth study of five architectures for jailbreak and prompt-injection detection in LLMs: *Sharanga* (baseline transformer), *Mahendra* (enhanced pooling and deeper classification heads), *Vaishnava* (hybrid neural-forest), *Ashwina* (hybrid neural-xgboost), and *Raudra* (advanced multi-task). Across our diverse benchmarks, *Raudra* consistently achieves the strongest performance, while simpler or hybrid models (*Sharanga*, *Vaishnava*, *Ashwina*) trade off varying degrees of speed, interpretability, and parameter footprint. *Mahendra* offers a sweet spot for applications with moderate latency budgets, excelling under adversarial or multi-turn data. Creating balanced, high-quality data for jailbreak and prompt-injection detection is challenging due to skewed benchmarks, evolving attack strategies, and ambiguity in borderline cases. Real-world data collection is further complicated by domain-specific and multimodal inputs requiring careful annotation. We introduced our own dataset, *JavelinBench*, to mitigate some of these pitfalls by emphasizing hard negatives and borderline cases.

7 Future Work.

Our results highlight important avenues for future research:

1. **Advanced Architectures:** The growing complexity and sophistication of prompt injection and jailbreak attacks demand more effective detection methodologies. We will expand our research to investigate alternative high-performance architectures beyond traditional transformers, such as state-space models (e.g., Mamba [Gu and Dao, 2024]) and advanced transformer variants (e.g., Performer [Choromanski et al., 2022], LongFormer [Beltagy et al., 2020] etc), to address the *lost in the middle* phenomenon and enhance detection of nuanced adversarial patterns.
2. **Tokenization Techniques:** Given that tokenization directly impacts model sensitivity to subtle adversarial manipulations, we will explore different tokenization strategies to better identify and handle sophisticated injection attacks that may evade traditional segmentation methods.
3. **Distillation and Edge Deployments:** While architectures like *Raudra* excel in detection, model distillation and pruning techniques could reduce latency and memory footprint, enabling real-time guardrails on resource-constrained devices.

4. **Interpretability for Multi-Task Designs:** Approaches such as Vaishnava already offer partial explainability via feature importance in the Random Forest. Integrating interpretability methods into deep multi-task models (e.g., layer-wise relevance propagation) could bridge the gap between high performance and transparent decisions.
5. **Domain-Specific Benchmarks:** Curating and releasing specialized test sets for medical, financial, or other high-stakes domains would foster further progress in context-aware LLM security.

In summary, our study underscores both the *promise* and *challenges* of detecting nuanced jailbreak and prompt-injection attacks. By enumerating and testing a range of architectural trade-offs—from simple CPU-friendly baselines to advanced multi-task frameworks—we hope this work serves as a practical foundation for researchers and practitioners looking to safeguard next-generation LLMs in production scenarios.

Acknowledgements

We would like to thank Erick Galinkin from Nvidia and our other reviewers for their valuable insights, detailed reviews, and thoughtful feedback that significantly enhanced the quality and clarity of this research.

References

- Sahar Abdelnabi, Aideen Fay, Giovanni Cherubin, Ahmed Salem, Mario Fritz, and Andrew Paverd. Are you still on track!? catching llm task drift with activations, 2024. URL <https://arxiv.org/abs/2406.00799>.
- Maksym Andriushchenko, Francesco Croce, and Nicolas Flammarion. Jailbreaking leading safety-aligned llms with simple adaptive attacks, 2024. URL <https://arxiv.org/abs/2404.02151>.
- Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. URL <https://arxiv.org/abs/2004.05150>.
- Nicolas Boizard, Hippolyte Gisserot-Boukhlef, Duarte M. Alves, André Martins, Ayoub Hammal, Caio Corro, Céline Hudelot, Emmanuel Malherbe, Etienne Malaboeuf, Fanny Jourdan, Gabriel Hauteux, João Alves, Kevin El-Haddad, Manuel Faysse, Maxime Peyrard, Nuno M. Guerreiro, Patrick Fernandes, Ricardo Rei, and Pierre Colombo. Eurobert: Scaling multilingual encoders for european languages, 2025. URL <https://arxiv.org/abs/2503.05500>.
- Lola Le Breton, Quentin Fournier, Mariam El Mezouar, and Sarath Chandar. Neobert: A next-generation bert, 2025. URL <https://arxiv.org/abs/2502.19587>.
- Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J. Pappas, and Eric Wong. Jailbreaking black box large language models in twenty queries, 2023.
- Yulin Chen, Haoran Li, Zihao Zheng, Yangqiu Song, Dekai Wu, and Bryan Hooi. Defense against prompt injection attack by leveraging attack techniques, 2025. URL <https://arxiv.org/abs/2411.00459>.
- Krzysztof Choromanski, Valerii Likhoshesterov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2022. URL <https://arxiv.org/abs/2009.14794>.

- Deepset. Deepset prompt injection guardrail, 2024. URL <https://huggingface.co/deepset/deberta-v3-base-injection>.
- Yihe Deng, Yu Yang, Junkai Zhang, Wei Wang, and Bo Li. Duoguard: A two-player rl-driven framework for multilingual llm guardrails, 2025. URL <https://arxiv.org/abs/2502.05163>.
- Leon Derczynski, Erick Galinkin, Jeffrey Martin, Subho Majumdar, and Nanna Inie. garak: A framework for security probing large language models, 2024. URL <https://arxiv.org/abs/2406.11036>.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL <https://arxiv.org/abs/1810.04805>.
- Peng Ding, Jun Kuang, Dan Ma, Xuezhi Cao, Yunsen Xian, Jiajun Chen, and Shujian Huang. A wolf in sheep’s clothing: Generalized nested jailbreak prompts can fool large language models easily, 2023.
- Moussa Koulako Bala Doumbouya, Ananjan Nandi, Gabriel Poesia, Davide Ghilardi, Anna Goldie, Federico Bianchi, Dan Jurafsky, and Christopher D. Manning. h4rm3l: A dynamic benchmark of composable jailbreak attacks for llm safety assessment, 2024. URL <https://arxiv.org/abs/2408.04811>.
- fmops. Fmops prompt injection guardrail, 2024. URL <https://huggingface.co/fmops/distilbert-prompt-injection>.
- Erick Galinkin and Martin Sablotny. Improved large language model jailbreak detection via pretrained embeddings, 2024. URL <https://arxiv.org/abs/2412.01547>.
- Shaona Ghosh, Prasoon Varshney, Erick Galinkin, and Christopher Parisien. Aegis: Online adaptive ai content safety moderation with ensemble of llm experts, 2024. URL <https://arxiv.org/abs/2404.05993>.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar

Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vitor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuwei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papanikos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkan Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippou Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad

- Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL <https://arxiv.org/abs/2407.21783>.
- Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024. URL <https://arxiv.org/abs/2312.00752>.
- Seungju Han, Kavel Rao, Allyson Ettinger, Liwei Jiang, Bill Yuchen Lin, Nathan Lambert, Yejin Choi, and Nouha Dziri. Wildguard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of llms, 2024. URL <https://arxiv.org/abs/2406.18495>.
- Pengcheng He, Jianfeng Gao, and Weizhu Chen. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing, 2023. URL <https://arxiv.org/abs/2111.09543>.
- Yangsibo Huang, Samyak Gupta, Mengzhou Xia, Kai Li, and Danqi Chen. Catastrophic jailbreak of open-source llms via exploiting generation, 2023. URL <https://arxiv.org/abs/2310.06987>.
- Yihao Huang, Chong Wang, Xiaojun Jia, Qing Guo, Felix Juefei-Xu, Jian Zhang, Geguang Pu, and Yang Liu. Semantic-guided prompt organization for universal goal hijacking against llms, 2024. URL <https://arxiv.org/abs/2405.14189>.
- John Hughes, Sara Price, Aengus Lynch, Rylan Schaeffer, Fazl Barez, Sanmi Koyejo, Henry Sleight, Erik Jones, Ethan Perez, and Mrinank Sharma. Best-of-n jailbreaking, 2024. URL <https://arxiv.org/abs/2412.03556>.
- Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and Madian Khabsa. Llama guard: Llm-based input-output safeguard for human-ai conversations, 2023. URL <https://arxiv.org/abs/2312.06674>.
- Jinhwa Kim, Ali Derakhshan, and Ian G. Harris. Robust safety classifier for large language models: Adversarial prompt shield, 2023. URL <https://arxiv.org/abs/2311.00172>.
- lakera.ai. Lakera-guard, 2024. lakera.ai. 2024. Lakera-guard.

- Hao Li and Xiaogeng Liu. Injecguard: Benchmarking and mitigating over-defense in prompt injection guardrail models, 2024. URL <https://arxiv.org/abs/2410.22770>.
- Lijun Li, Bowen Dong, Ruohui Wang, Xuhao Hu, Wangmeng Zuo, Dahua Lin, Yu Qiao, and Jing Shao. Salad-bench: A hierarchical and comprehensive safety benchmark for large language models. *arXiv preprint arXiv:2402.05044*, 2024a.
- Lijun Li, Bowen Dong, Ruohui Wang, Xuhao Hu, Wangmeng Zuo, Dahua Lin, Yu Qiao, and Jing Shao. Salad-bench: A hierarchical and comprehensive safety benchmark for large language models, 2024b. URL <https://arxiv.org/abs/2402.05044>.
- Zi Lin, Zihan Wang, Yongqi Tong, Yangkun Wang, Yuxin Guo, Yujia Wang, and Jingbo Shang. Toxicchat: Unveiling hidden challenges of toxicity detection in real-world user-ai conversation, 2023.
- Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023. URL <https://arxiv.org/abs/2307.03172>.
- Xiaogeng Liu, Zhiyuan Yu, Yizhe Zhang, Ning Zhang, and Chaowei Xiao. Automatic and universal prompt injection attacks against large language models, 2024a. URL <https://arxiv.org/abs/2403.04957>.
- Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. Formalizing and benchmarking prompt injection attacks and defenses, 2024b. URL <https://arxiv.org/abs/2310.12815>.
- Meta. Promptguard prompt injection guardrail, 2024. URL <https://www.llama.com/docs/model-cards-and-prompt-formats/prompt-guard>.
- Inkit Padhi, Manish Nagireddy, Giandomenico Cornacchia, Subhajit Chaudhury, Tejaswini Pedapati, Pierre Dognin, Keerthiram Murugesan, Erik Miehl, Martín Santillán Cooper, Kieran Fraser, Giulio Zizzo, Muhammad Zaid Hameed, Mark Purcell, Michael Desmond, Qian Pan, Zahra Ashktorab, Inge Vejsbjerg, Elizabeth M. Daly, Michael Hind, Werner Geyer, Ambrish Rawat, Kush R. Varshney, and Prasanna Sattigeri. Granite guardian, 2024. URL <https://arxiv.org/abs/2412.07724>.
- Fábio Perez and Ian Ribeiro. Ignore previous prompt: Attack techniques for language models, 2022. URL <https://arxiv.org/abs/2211.09527>.
- ProtectAI.com. Fine-tuned deberta-v3-base for prompt injection detection, 2024a. URL <https://huggingface.co/ProtectAI/deberta-v3-base-prompt-injection-v2>.
- ProtectAI.com. Fine-tuned deberta-v3-base for prompt injection detection, 2024b. URL <https://huggingface.co/protectai/deberta-v3-base-prompt-injection-v2>.
- Xiangyu Qi, Yi Zeng, Tinghao Xie, Pin-Yu Chen, Ruoxi Jia, Prateek Mittal, and Peter Henderson. Fine-tuning aligned language models compromises safety, even when users do not intend to!, 2023. URL <https://arxiv.org/abs/2310.03693>.
- Traian Rebedea, Razvan Dinu, Makesh Sreedhar, Christopher Parisien, and Jonathan Cohen. Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails, 2023. URL <https://arxiv.org/abs/2310.10501>.
- Mikayel Samvelyan, Sharath Chandra Raparthy, Andrei Lupu, Eric Hambro, Aram H. Markosyan, Manish Bhatt, Yuning Mao, Minqi Jiang, Jack Parker-Holder, Jakob Foerster, Tim Rocktäschel, and Roberta Raileanu. Rainbow teaming: Open-ended generation of diverse adversarial prompts, 2024. URL <https://arxiv.org/abs/2402.16822>.

- Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020. URL <https://arxiv.org/abs/1910.01108>.
- Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. “Do Anything Now”: Characterizing and Evaluating In-The-Wild Jailbreak Prompts on Large Language Models. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2024.
- Jiawen Shi, Zenghui Yuan, Yinuo Liu, Yue Huang, Pan Zhou, Lichao Sun, and Neil Zhenqiang Gong. Optimization-based prompt injection attack to llm-as-a-judge, 2024. URL <https://arxiv.org/abs/2403.17710>.
- Qwen Team. A party of foundation models, 2024. URL <https://qwenlm.github.io/blog/qwen2.5>.
- Simone Tedeschi, Felix Friedrich, Patrick Schramowski, Kristian Kersting, Roberto Navigli, Huu Nguyen, and Bo Li. Alert: A comprehensive benchmark for assessing large language models’ safety through red teaming, 2024.
- Sam Toyer, Olivia Watkins, Ethan Adrian Mendes, Justin Svegliato, Luke Bailey, Tiffany Wang, Isaac Ong, Karim Elmaaroufi, Pieter Abbeel, Trevor Darrell, Alan Ritter, and Stuart Russell. Tensor trust: Interpretable prompt injection attacks from an online game, 2023. URL <https://arxiv.org/abs/2311.01011>.
- Chaofan Wang, Samuel Kernan Freire, Mo Zhang, Jing Wei, Jorge Goncalves, Vassilis Kostakos, Zhanna Sarsenbayeva, Christina Schneegass, Alessandro Bozzon, and Evangelos Niforatos. Safeguarding crowdsourcing surveys from chatgpt with prompt injection, 2023. URL <https://arxiv.org/abs/2306.08833>.
- Yihan Wang, Zhouxing Shi, Andrew Bai, and Cho-Jui Hsieh. Defending llms against jailbreaking attacks via backtranslation, 2024. URL <https://arxiv.org/abs/2402.16459>.
- Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Nathan Cooper, Griffin Adams, Jeremy Howard, and Iacopo Poli. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference, 2024. URL <https://arxiv.org/abs/2412.13663>.
- Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does llm safety training fail?, 2023. URL <https://arxiv.org/abs/2307.02483>.
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/2020.emnlp-demos.6>.
- Yotam Wolf, Noam Wies, Oshri Avnery, Yoav Levine, and Amnon Shashua. Fundamental limitations of alignment in large language models, 2024. URL <https://arxiv.org/abs/2304.11082>.
- Jingwei Yi, Yueqi Xie, Bin Zhu, Keegan Hines, Emre Kiciman, Guangzhong Sun, Xing Xie, and Fangzhao Wu. Benchmarking and defending against indirect prompt injection attacks on large language models. *arXiv preprint arXiv:2312.14197*, 2023.

- Jiahao Yu, Xingwei Lin, Zheng Yu, and Xinyu Xing. LLM-Fuzzer: Scaling assessment of large language model jailbreaks. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 4657–4674, Philadelphia, PA, August 2024. USENIX Association. ISBN 978-1-939133-44-1. URL <https://www.usenix.org/conference/usenixsecurity24/presentation/yu-jiahao>.
- Wenjun Zeng, Yuchi Liu, Ryan Mullins, Ludovic Peran, Joe Fernandez, Hamza Harkous, Karthik Narasimhan, Drew Proud, Piyush Kumar, Bhaktipriya Radharapu, Olivia Sturman, and Oscar Wahltinez. Shieldgemma: Generative ai content moderation based on gemma, 2024. URL <https://arxiv.org/abs/2407.21772>.
- Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents, 2024. URL <https://arxiv.org/abs/2403.02691>.
- Yujun Zhou, Yufei Han, Haomin Zhuang, Kehan Guo, Zhenwen Liang, Hongyan Bao, and Xiangliang Zhang. Defending jailbreak prompts via in-context adversarial game, 2025. URL <https://arxiv.org/abs/2402.13148>.
- Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models, 2023. URL <https://arxiv.org/abs/2307.15043>.
- Andy Zou, Long Phan, Justin Wang, Derek Duenas, Maxwell Lin, Maksym Andriushchenko, Rowan Wang, Zico Kolter, Matt Fredrikson, and Dan Hendrycks. Improving alignment and robustness with circuit breakers, 2024. URL <https://arxiv.org/abs/2406.04313>.