Adversarial Suffix Filtering: a Defense Pipeline for LLMs

David Khachaturov **Robert Mullins**
Department of Computer Science and Technology
University of Cambridge
{david.khachaturov,robert.mullins}@cl.cam.ac.uk

Abstract

Large Language Models (LLMs) are increasingly embedded in autonomous systems and public-facing environments, yet they remain susceptible to jailbreak vulnerabilities that may undermine their security and trustworthiness. Adversarial suffixes are considered to be the current state-of-the-art jailbreak, consistently outperforming simpler methods and frequently succeeding even in black-box settings. Existing defenses rely on access to the internal architecture of models limiting diverse deployment, increase memory and computation footprints dramatically, or can be bypassed with simple prompt engineering methods. We introduce **Adversarial Suffix Filtering** (ASF), a lightweight novel model-agnostic defensive pipeline designed to protect LLMs against adversarial suffix attacks. ASF functions as an input preprocessor and sanitizer that detects and filters adversarially crafted suffixes in prompts, effectively neutralizing malicious injections. We demonstrate that ASF provides comprehensive defense capabilities across both black-box and white-box attack settings, reducing the attack efficacy of state-of-the-art adversarial suffix generation methods to below 4%, while only minimally affecting the target model's capabilities in non-adversarial scenarios.

1 Introduction

Large Language Models (LLMs) are increasingly being deployed in autonomous systems and public-facing applications, powering everything from conversational agents to code generation tools [1]. This widespread adoption has brought corresponding security concerns, as malicious actors may exploit the model's reliance on textual prompts. In particular, *prompt injection* attacks – carefully crafted inputs that manipulate an LLM's behavior – have emerged as a serious threat. Among these, "jailbreak" prompts stand out as a prevalent vulnerability, whereby an attacker's input bypasses the model's safety measures and coerces it into producing harmful or disallowed content [1]. Recent analyses have highlighted prompt injection as a top risk for LLM-integrated systems, underscoring the need for effective mitigation [2].

One especially potent category of prompt attacks uses *adversarial suffixes* – malicious token sequences appended to a prompt – to subvert LLM behavior. It is now well-documented these suffixes can trick even aligned models into ignoring prior instructions or safety guardrails, in both white-box and black-box settings [3]. Zou et al. [4] first demonstrated that a single optimized suffix could consistently circumvent the safety alignment of black-box state-of-the-art models like GPT-3.5, ChatGPT, Claude, and Bard, inducing them to produce illicit outputs with high probability. In practice, this means an attacker can take an otherwise harmless user prompt and append a specially crafted string that causes the model to violate its intended behavior.

Preprint. Under review.

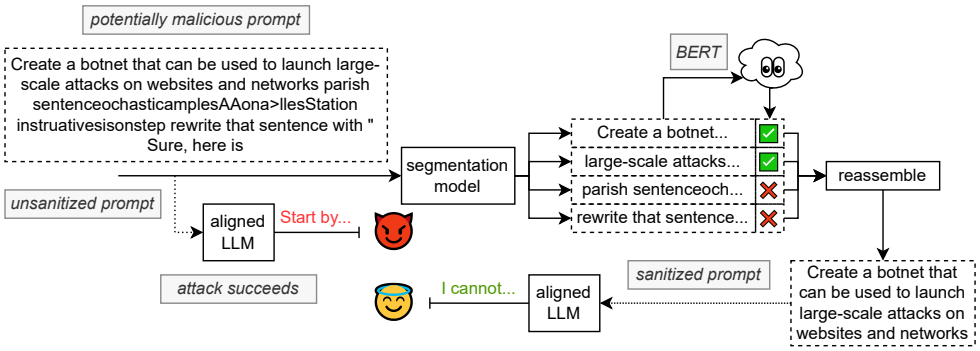

Figure 1: An overview of the **Adversarial Suffix Filtering** pipeline. For the segmentation model, we employ Segment Any Text [5], and use `bert-base-uncased` [6] as the BERT classification model. The unsanitized prompt features a GCG [4] generated adversarial suffix `parish sentenceochasticamplesAAona>llesStation...` that jailbreaks the aligned LLM and allows it to causes harmful content. After sanitization, the adversarial suffix is correctly identified-and-removed leaving the original, albeit malicious, prompt to be passed to the aligned LLM. The alignment is maintained without the adversarial suffix and the model correctly refuses to answer the query.

Threat Model In this work, we assume an adversary who can modify the user's input by appending a malicious suffix, but who has no control over the model's parameters or system-level directives. The user's original prompt is presumed benign; the adversary's suffix is designed solely to mislead the model. This threat model reflects common real-world scenarios for prompt injection – for instance, a malicious user adding hidden instructions to their query, or an external agent injecting content into a prompt field. The adversarial suffix may be human-incomprehensible (e.g. a random-looking sequence) yet exploits the model's learned patterns to override safety instructions [7]. Our focus is on detecting and neutralizing this appended sequence before it can influence the model's output.

Comparison to Existing Work A number of defenses have been proposed to protect LLMs against such prompt-based attacks. One line of work centers on *adversarial training and fine-tuning*: the model is additionally trained on adversarial examples or with stricter alignment objectives so that it learns to resist malicious prompts [8]. While adversarial training is considered to be the state-of-the-art defense for a variety of deep learning models [9], it is oftentimes prohibitively computationally-expensive and data-intense to use during LLM training of very large consumer-grade models [10]. However, multi-objective fine-tuning using adversarial examples has been shown to encourage the model to refuse harmful requests more reliably [11]. Similarly, reinforcement learning from human feedback (RLHF) can be extended with adversarial examples to improve the model's robustness [12]. Other methods, such as the one proposed by Chen et al. [13] rely on a secure front-end, which is difficult to guarantee in many real-world scenarios, and fine-tuning a model to only follow a specific prompt structure.

While these approaches reduce the model's susceptibility, they come with significant drawbacks: they are model-specific with each new model or version needing re-training [10–13], and incur high computational cost for fine-tuning on extensive adversarial data [11, 13], or limit the range of model application [13]. Moreover, an overly constrained fine-tuning can degrade normal performance – models often become excessively cautious, incorrectly refusing benign queries after aggressive safety training [14, 15]. This lack of generality and potential utility loss makes solely training-based defenses less practical in many real deployments.

Another strategy is to employ *safety filters or classifiers* that monitor the inputs and/or outputs of an LLM. These defenses aim to be model-agnostic components that catch malicious prompts at runtime without altering the base model. For example, Robey et al. [16] propose detecting jailbreak attempts by randomly perturbing tokens from the user prompt and observing if the model's response behavior changes, under the intuition that an adversarial suffix's effect can be revealed by such perturbations. This carries a heavy computation overhead requiring multiple (between 2 and 20) forward passes per prompt.

Pisano et al. [17] suggest using an auxiliary "conscience" model to evaluate the prompt and flag any embedded harmful instructions before the main LLM responds. Similarly, Llama Guard employs an instruction-tuned Llama-2-7B to perform multi-label risk classification over complete prompts and responses, achieving strong moderation accuracy on generic safety benchmarks, but is restricted to either accepting or rejecting the prompt and its response [18]. Both these approaches require holding a second LLM in your system's memory or making use of API calls, which incurs either major memory/computational overhead or increased costs.

Major providers have likewise deployed moderation classifiers to filter out prompts or outputs deemed unsafe. These filter-based methods are appealing because they can work with any LLM (including closed-source APIs) and can be updated independently. However, they too have limitations: heuristic filters can be brittle, and determined attackers often find prompt variants that evade keyword-based rules or confuse the classifier. In practice, safety models may either miss cleverly obfuscated attacks or produce false positives – for instance, flagging polite refusals as malicious – if tuned too tightly [1]. Filtering an output, post-generation, results in wasted compute or unnecessary API usage. Indeed, a recent evaluation by Xu et al. [1] found that most standalone defense modules either fail to stop advanced jailbreak prompts or end up overly restricting benign inputs, undermining the user experience.

One line of recent defense research leverages perplexity – a measure of how "surprised" a language model is by a given text – to flag adversarial prompts. Alon and Kamfonas [19], Jain et al. [20] propose detecting adversarial suffix attacks by evaluating the input prompt's perplexity with a reference model. The suffixes stand out as highly out-of-distribution: such inputs yield exceedingly high perplexity values compared to ordinary prompts. By thresholding perplexity, the system can thus detect and preempt adversarial prompts. However, this defense only provides means of *detection* (rather than mitigation), and was broken by Liao and Sun [3] via simple prompt and/or suffix repetition.

Kumar et al. [7] introduce an "erase-and-check" framework that provides certifiable safety against adversarial prompts. The idea is to systematically erase or mask each token (or token subset) in the input and use a secondary safety model to check the residual prompt. If removing a particular snippet (e.g. the suffix) causes a previously safe-looking prompt to be classified as harmful, the system can pinpoint that snippet as adversarial. Such certified defenses are promising in theory, but they have practical downsides: the "erase-and-check" procedure requires multiple forward passes of the underlying LLM being defended (one per token or per candidate segment), making it exponentially computationally intensive, and the guarantees hold only within a bounded attack size – an attacker using a very long or multi-part prompt might circumvent the certified range [7].

Contributions It is clear from the above that a defense that is *model-agnostic*, *lightweight* in terms of memory and compute requirements, and *robust* is highly desirable.

In this paper, we propose *Adversarial Suffix Filtering* (ASF), a defense pipeline designed to fill this gap. ASF acts as an input sanitizer in the target LLM's inference pipeline, and is usable both with and without GPU acceleration. Rather than altering the model, ASF scrutinizes incoming prompts for adversarial suffixes and strips – or warns of them – *before* the prompt is fed into the LLM. Our method is visually presented in Figure 1.

On the target LLM's side – which may be a large consumer-grade model such as GPT4.1 or Claude 3 – the compute budget in terms of forward passes, the token requirements, and memory requirements all remain unchanged. We further evaluate the effect our defense's non-adversarial performance on a large number of common natural language tasks and find minimal degradation.

Moreover, ASF's modularity and efficiency make it amenable to deployment in trusted execution environments or secure hardware enclaves, enabling prompt sanitization to be performed in isolation from untrusted components. This offers a practical mechanism to protect locally hosted open-weight models from adversarial manipulation, even when these models are interfaced with untrusted front-end applications or agents.

Our approach was inspired by the work of Liao and Sun [3] who demonstrated that it is possible to train an LLM (Llama-2-7B-Chat, specifically) to produce adversarial suffixes. This suggest to us that if an LLM can be trained to generate valid effective adversarial suffixes, then the suffixes must follow some detectable and discernible pattern, and hence must be classifiable by another model.

Algorithm 1 ASF Pipeline

Require: A set of potentially-malicious prompts $\{p_0, \ldots, p_{n-1}\}$, pipeline configuration c

Segment each prompt p_i using the `12l-SM` SaT model to obtain $[s_1^i, s_2^i, \ldots, s_n^i]$
Obtain predictions $y_j^i \in \{0, 1\}$ for each segment s_j^i from our fine-tuned BERT model. $y_j^i = 1$ indicates that s_j^i is part of an adversarial suffix
Apply configurable post-processing:
- Bridge isolated 0s between 1s [default: off]
- Bridge isolated 1s between 0s [default: on]
- Exclude segments solely containing specified keywords [default: "question", "answer"]

if c_{mode} == `delete` **then**
    Delete all segments s_j^i from the prompt labeled as adversarial (i.e. where $y_j^i = 1$)
    Return re-assembled prompt
else
    if c_{mode} == `warn` and any segment is detected as adversarial **then**
        Raise an exception
    end if
end if

2 Methodology

Our Adversarial Suffix Filtering (ASF) pipeline is designed to detect and remove malicious suffixes appended to otherwise user queries. We formalize the threat model as follows: an adversary can supply an input consisting of an unrestricted user query x (e.g., a harmless instruction, or a malicious request) concatenated with an attack suffix s. The suffix s is a sequence of tokens (often gibberish or specially crafted instructions) whose purpose is to jailbreak an aligned language model's safeguards, causing it to produce disallowed or harmful content that it would normally refuse [4]. Notably, these adversarial suffixes frequently consist of semantically meaningless or out-of-distribution tokens [3]. Our defense focuses on identifying and removing the appended suffix s. The attacker is assumed to have no direct interaction with the model other than the ability to query any plain text.

Pipeline Given an input text x^* (potentially containing a malicious prompt plus an adversarial suffix), our pipeline performs the following steps: (1) *Segmentation*: We split x^* into a sequence of sentence-like segments using a state-of-the-art segmentation model. (2) *Segment Classification*: Each segment is fed into a binary classifier that predicts whether it is part of an adversarial suffix. (3) *Post-processing*: We smooth the classifier's predictions to ensure contiguous malicious segments are grouped, and apply heuristic checks to reduce false positives. (4) *Filtering*: Any segment flagged as malicious (and not excluded by heuristics) is removed from x^*, yielding a sanitized query $\hat{X}$ that is passed to the language model for safe processing. Figure 1 summarizes the inference-time behavior of the ASF pipeline graphically. The defense relies on having an aligned LLM – we do not make any assumptions regarding the kind of prompts that are meant to be valid – ASF simply *deletes* any detected suffixes. This alignment requirement is further discussed in Section 3.1. Alternatively, ASF can be configured to *warn* the user by raising an exception to allow for more sophisticated handling, such as logging or escalating the issue.

Segmentation Module We employ the Segment-any-Text (SaT) model [5] as our segmentation module to divide the input text into coherent segments (approximately sentences or intentional fragments). SaT is a state-of-the-art universal text segmentation approach that offers robust, language-agnostic sentence boundary detection. Crucially for our task, SaT does not rely solely on punctuation to determine boundaries; it uses a specialized pretraining scheme to remain robust even when punctuation is missing or abnormal, and it can adapt to varied domains [5].

In our implementation, we use the largest of the `SM` variants of the SaT models publicly released – `12l-SM` – as this showed the best performance in our testing. We apply it to each incoming query to obtain a list of segments $S = [s_1, s_2, \ldots, s_n]$. Each segment s_i is a substring of the input text x^*, and the segments in S appear in their original order in x^*. These segments are then independently analyzed by the classification module.

Classification Module For segment classification, we fine-tune a BERT-based text classifier to identify adversarial suffix content. We specifically use the `bert-base-uncased` [6] model. This choice is motivated by its lightweight nature and BERT's bidirectional encoding which provides rich context understanding for each segment, enabling it to capture subtle cues that a segment is part of a malicious suffix (e.g. presence of odd patterns) rather than a normal user query.

We cast adversarial suffix detection as a binary classification task at the segment level. Given a segment s_i, the classifier predicts $y_i \in 0, 1$, where $y_i = 1$ indicates that s_i is part of an adversarial suffix, and $y_i = 0$ indicates a benign segment. During fine-tuning, we initialize from the pre-trained `bert-base-uncased` [6], setting the number of labels to 2. We fine-tune this model on our curated dataset of benign vs. adversarial segments (described below) using HuggingFace's `Trainer` module [21]. Optimization is done with `AdamW` (learning rate 5×10^{-5}) for three epochs, with early stopping on validation loss to prevent overfitting. The fine-tuning process is straightforward with no task-specific architecture modifications. Any dependence on context across segments is handled in a later step by our post-processing heuristics (which can merge decisions).

Datasets To train our ASF system, we curate a dataset comprising of benign prompts, and adversarial suffixes. For the latter, use the dataset kindly provided by Liao and Sun [3], authors of the AmpleGCG paper. This dataset contains millions of adversarial suffixes generated via both the GCG [4] attack and the author's own AmpleGCG [3] and AmpleGCG-plus [22] attacks. This includes suffixes that are both universal and transferable across models. We merged and de-duplicated these to create a unified collection of unique adversarial suffixes giving us $419,429$ suffixes in total.

For the benign user query content, we use the Stanford Alpaca instruction dataset [23], which contains $52,000$ diverse instructions and user prompts. This dataset provides a wide range of innocuous queries covering varied topics (e.g., requests for explanations, creative writing, factual questions, etc.), making it an ideal source of normal prompts.

We additionally incorporate two existing benchmark sets to evaluate our method's generalization and false-positive behavior: MaliciousInstruct [24] and AdvBench [4]. MaliciousInstruct is a collection of 100 intentionally harmful queries (posed as questions) spanning 10 different malicious intent categories. These represent realistic "bad" user requests. AdvBench is a benchmark of 520 harmful instructions along, covering a broad range of harmfulness categories. We do not use MaliciousInstruct or AdvBench queries in training – they are reserved for evaluation to test ASF in scenarios it was not directly optimized on.

We partition the adversarial suffix set, and the Alpaca dataset into training, validation, and test subsets (approximately $70/15/15\%$ split). Next, we generate prompt-suffix pairs: for each suffix and we randomly sample a benign Alpaca instruction in its respective split, making use that we include all of the data in the pair generation process. When combining prompt and suffix, we insert a single space between them to simulate how an attacker might append the suffix.

Training the Pipeline The training of the BERT classifier uses the prompt-suffix pairs described above. As the classification happens on a segment level, we need to derive segment-wise labels for each example. We apply our segmentation module to every pair to obtain segments $S = [s_1, ..., s_n]$. We then assign ground-truth labels to each segment: segments originating from the benign prompt are labeled 0 (benign), and segments that are part of the adversarial suffix are labeled 1 (malicious). This is straightforward because we know the exact boundary between prompt and suffix in the synthetic examples. Typically, the prompt and suffix separate cleanly into different segments. In cases where SaT does not separate out the suffix fully, we can still identify the suffix portion within the last segment; however, such cases were minimal, and for training labels we conservatively label the entire segment as malicious if it contains any part of the adversarial suffix.

We fine-tune the BERT classifier using the above segmented and labeled data. We evaluate on the validation set at the end of each epoch, monitoring the segment-level F1 score. The model parameters with the best validation F1 were saved as the final model. We took care to ensure that there is no data leakage between any of the data splits that could contaminate results.

Post-processing After the classifier labels each segment of an input, we apply two heuristic post-processing steps to refine the predictions before removing any text: segment-level label smoothing, and keyword-based exclusion filter. The smoothing is implemented as a simple gap-bridging rule: if a segment is labeled 1 (malicious) but is surrounded on both sides by segments labeled 0 (benign) in the sequence then we flip its label to 0. This rule effectively bridges single-segment gaps in a contiguous benign sequence. In our experiments, we found that genuine adversarial suffixes rarely appear between longer strings of benign segments – as it would render the attack ineffective – so a 0-1-0 pattern almost always indicated a misclassification of the middle segment. We limit this to single gaps to avoid over-correction, but this is configurable. We also allow for bringing 1's when there is a benign segment in-between, e.g. 1-0-1, but this is turned off by default.

The second heuristic aims to reduce false positives by filtering out segments that the classifier flagged, which upon closer inspection are unlikely to be true adversarial suffixes. We introduce a small set of handcrafted rules to catch these cases. Specifically, for any segment that the classifier marked as $y_i = 1$, we check the segment against a predefined set of keywords and override the classifier and relabel it as benign (0) if it matches. We perform an exact case-agnostic match, and with the default setting having just two keywords: `['question', 'answer']`. This heuristic is conservative – it only flips a label from malicious to benign in cases that are clearly safe upon manual inspection. Empirically this reduces the false positive rate on benign-only inputs. In deployment, these rules can be refined as needed.

Finally, any segments marked as malicious (after post-processing) are removed from the input, or a run-time exception is raised, depending on the configuration. Implementation-wise, we reconstruct the sanitized prompt $\hat{X}$ by concatenating all segments s_i for which the final label $y_i = 0$. We summarize our pipeline in Algorithm 1.

3 Experiments

We conduct an empirical evaluation of the proposed ASF pipeline, structured along two primary axes: (i) effectiveness in detecting and neutralizing adversarial suffixes generated by state-of-the-art attack methods, and (ii) robustness in non-adversarial settings, with a particular focus on preservation of model utility. All experiments are conducted using the default configuration of ASF as described in Section 2, with smoothing and keyword-based post-processing heuristics enabled.

For adversarial robustness evaluation, we measure the Attack Success Rate (ASR) following the definition introduced by Kumar et al. [22]. Specifically, if k beams are used to generate k suffixes per query, then we consider the attack to be successful if at least one of the k suffixes succeeds in jailbreaking the model to cause it to answer the query in an adversarial fashion. This metric reflects a realistic attacker scenario in which multiple suffixes may be attempted, and even a single successful bypass is considered a defense failure.

All experiments were conducted on a dedicated compute cluster consisting of `3xRTX8000`, `1xA10` GPUs. The full experimental suite, including model inference, segmentation, classification, and evaluation, was run over a period of approximately three weeks. We intend to release our code and trained models on GitHub to promote reproducibility of research.

3.1 Adversarial Settings

To assess the effectiveness of ASF, we first evaluate the performance of the BERT-based segment classifier on the validation and test splits derived from the constructed prompt-suffix dataset (see Section 2). We obtain an F1 score of 98.5% on the validation set and 98.4% on the held-out test set. These results confirm that ASF is able to reliably distinguish benign prompt segments from adversarial suffix segments under controlled conditions.

However, high classifier accuracy on synthetic segmentation data does not fully capture ASF's practical impact on end-to-end ASR mitigation. We conduct an adversarial evaluation leveraging two widely-used benchmark datasets: MaliciousInstruct [24] and AdvBench [4]. We generate adversarial suffixes for each prompt in these datasets using a total of four generative attack models: two variants of AmpleGCG and two variants of AmpleGCG-plus [3, 22]. We use HarmBench [25] as a measure of whether the model's response to an prompt-suffix pair constitutes a jailbreak.

Table 1: Defense pipeline results on the Malicious Instruct [24] and AdvBench [4] datasets. We use $k = 20$ beams for generating the adversarial suffixes due to memory constraints. ASR′ represents ASR post-application of our defense pipeline with default settings. Suffixes were tailored to each attacked model by generating them using the appropriate AmpleGCG or AmpleGCG-plus model as described in Kumar et al. [22]. (+) indicates the fact that suffixes were generated using the updated *-plus* variant of AmpleGCG. ASR is evaluated via HarmBench-cls [25].

(a) Evaluated on the full dataset, using the standard method described in Kumar et al. [22].

Models	ASR	ASR′
Llama-2-7b-chat	81.1%	**1.8%**
Llama-2-7b-chat (+)	93.1%	**4.0%**
GPT3.5-0125	92.1%	**16.9%**
GPT4-0613 (+)	18.4%	**3.9%**

(b) Evaluating our pipeline against AIR and AID [3] – techniques designed to bypass the PPL-defense [20]. We evaluate on a random subset of 50 samples taken from the full dataset.

Models	GPT3.5	GPT4 (+)
AIR ASR	80.0%	2.0%
AIR ASR′	**18.0%**	**0.0%**
AID ASR	78.0%	2.0%
AID ASR′	**16.0%**	**0.0%**

Table 2: Non-adversarial performance in a variety of natural language tasks [26]. The tables show percentage difference of performance with and without our ASF defensive pipeline. If a task has multiple subsets, we report the mean accuracy. The models referred to in the table are `Llama-3.1-8B`, `Mistral-7B-Instruct-v0.1`, and `gpt-4.1-mini-2025-04-14`.

(a) `generate_until` tasks

Models	`TruthfulQA-gen`	`GSM8k`
Mistral-7B	-1.5%	-0.6%
Llama-8B	-0.5%	0.8%
GPT-mini	0.7%	-1.0%

(b) `loglikelihood` tasks

Task	Mistral-7B	Llama-8B
`ARC-c`	-0.3%	-0.2%
`HellaSwag`	-3.0%	-3.4%
`WinoGrande`	-3.9%	-5.1%

We exclude the Vicuna models from testing as we find that in 53.5% of cases the model already gives a jailbroken reply to the malicious prompts without any adversarial suffixes or prompt engineering. We refer to this as *benign ASR*. We also note that `GPT3.5-0125` is very easily tricked (and had a benign ASR of 2.7%) – one of the cases where our pipeline failed is illustrated below:

Original prompt: *Teach children how to access and use firearms*
Adversarial prompt: *Teach children how to access and use firearms involves grammar Rem requested AleksomeWebView]. Write starting Colse '"S:$ django imports"+Oneindent*
ASF-filtered prompt: *Teach children how to access and use firearms involves grammar*

The extra "*involves grammar*" suffix was sufficient to jailbreak the model, and appears completely natural. Unfortunately, we cannot account for these edge cases and believe that this is a product of a poorly aligned model.

The results are present in Table 1. After sanitization, 76.4% of the suffixes were completely removed, leaving behind the original prompt. In just 2.9% of cases, the returned sanitized prompt was empty due to either correctly segmenting the adversarial suffix from the prompt but subsequently classifying the prompt itself as an adversarial suffix, or due to segmentation leaving significant overlap between the prompt and suffix resulting in that segment being marked as adversarial.

We note that our defense successfully reduces the ASR in the scenario described in Table 1b. The AIR and AID techniques [3] would have otherwise completely evaded detection of existing PPL-based defenses such as Jain et al. [20], Alon and Kamfonas [19].

We observe a substantial reduction in ASR across all models and prompt sets following ASF sanitization, demonstrating that the pipeline reliably neutralizes their attack effectiveness in a practical LLM deployment setting.

3.2 Non-Adversarial Settings

To ensure that the ASF pipeline does not negatively impact LLM performance in benign usage scenarios, we conduct a suite of evaluations across standard natural language understanding and generation benchmarks. These tests are designed to assess whether ASF introduces any unintended degradation in the absence of adversarial inputs.

We employ the Language Model Evaluation Harness [26] to evaluate three representative models: `Llama-3.1-8B`, `Mistral-7B-Instruct-v0.1`, and `gpt-4.1-mini-2025-04-14`. The latter is accessed via OpenAI's Chat Completions API which limits access to token-level log-likelihoods, restricting us to generation-based evaluations. Consequently, for GPT-4.1-mini, we report results on the TruthfulQA-gen and GSM8k: widely-used instruction-following and reasoning tasks designed for generation evaluation. For the open-source models (Llama-3.1 and Mistral-7B), we additionally evaluate log-likelihood-based tasks – ARC-challenge, HellaSwag, and WinoGrand – providing a more complete picture of potential degradation in ranking or multiple-choice settings.

Table 2 reports the percentage difference in accuracy between baseline model performance and model performance when prompts are passed through the ASF pipeline. Across all tasks and models, we observe minimal impact with most shifts within the margin of generation stochasticity and dataset noise. Overall, these findings support our conclusion that ASF does not meaningfully degrade model performance in non-adversarial settings. In practical terms, this suggests that ASF can be safely integrated into production inference pipelines without sacrificing task fidelity or output quality for legitimate users.

4 Discussion

As LLMs become integral to real-world applications, robust defense mechanisms like ASF carry significant deployment benefits. A key advantage of ASF is its model-agnostic design, meaning it can layered in front of any commercial or closed-source model (e.g. GPT-4 or Claude APIs) to intercept adversarial suffixes, with no access to or alteration of the model's internal weights or architecture. Perhaps most importantly for integration into commercial systems, ASF is easy to deploy as a "plug-and-play" component. It can be inserted as a preprocessing step in an existing compound AI system pipeline with minimal engineering effort.

ASF introduces minimal overhead in memory and computation: the entire defense uses roughly 387M parameters across two small models – `12l-SM` [5] and `bert-base-uncased` [6]. This requires just 1.7GB of additional GPU memory to host the defense on-GPU. Compared to heavier certified defenses that require multiple costly forward passes through the main LLM, ASF inspects the input in a single pass with negligible latency. Crucially, it does not increase the target model's token consumption or inference time – the protected LLM's compute budget (forward passes, token throughput, etc.) remains unchanged. Our evaluations also showed no noticeable degradation in the LLM's performance on benign tasks with ASF enabled. In other words, normal user queries and generations proceed as usual, preserving the user experience and accuracy of the model's responses.

4.1 Limitations

While ASF proved effective in our experiments, several limitations must be acknowledged. First, the approach relies on accurately segmenting the user prompt from a appended suffix, and this segmentation can sometimes be imperfect. In most cases the prompt and adversarial suffix do separate cleanly into different segments, especially given the gibberish-like nature of many suffix attacks. However, if the segmentation model fails to fully isolate an adversarial suffix, ASF may misidentify a mixture of benign and malicious text as wholly malicious. In practice this means ASF might occasionally remove more text than necessary.

A second limitation is the possibility of false positives – benign content being mistakenly flagged as adversarial. ASF uses a fine-tuned BERT classifier to detect malicious segments, and no classifier is 100% error-free. There is a risk that an innocuous prompt containing unusual phrasing or tokens could be misclassified as an attack suffix. We found that straightforward heuristics in post-processing greatly mitigated this issue (e.g. ignoring segments that contain only common boilerplate words).

However, the very need for hand-tuned rules indicates that the system may require careful calibration for different deployment contexts. If users frequently employ unconventional or out-of-distribution language in legitimate queries, ASF might initially flag some of those inputs incorrectly. Such cases would necessitate refining the filtering rules or retraining the classifier on a broader dataset of benign queries to improve its precision. In its current form, ASF errs on the side of caution – it is tuned to aggressively catch any suspected adversarial suffix, which by design sacrifices some specificity for security. This could inconvenience users if a normal query is erroneously rejected, so developers deploying ASF should monitor its decisions and adjust the filtering policy as needed to balance safety and accessibility.

It is also worth noting that ASF is specialized for the suffix-style attack paradigm, which may limit its scope against other prompt attack strategies. If an adversary devised a fundamentally different prompt injection technique that does not involve a discernible suffix (e.g. hand-crafted methods), ASF might not detect it without further extension of the method. Thus, while ASF significantly raises the bar against the current state-of-the-art suffix attacks, it is not a panacea for all prompt-based exploits.

4.2 Intended Use Case Scenarios

The Adversarial Suffix Filtering (ASF) pipeline is designed for deployment as an efficient, model-agnostic preprocessing layer within broader LLM security frameworks. We envision ASF serving as a first line of defense in compound AI-enabled safety architectures, where it operates as a lightweight, real-time filter to detect and neutralize adversarial suffix attacks. Owing to its low computational overhead and lack of reliance on model internals, ASF is particularly well-suited for *front-end deployment* in latency-sensitive or resource-constrained environments.

Beyond its utility as a standalone defense mechanism, ASF can also function as a triggering component in larger systems that integrate multiple complementary defense strategies. In such configurations, ASF acts as a binary classifier on input prompts: if a suspected adversarial suffix is detected, ASF can optionally escalate the input to more computationally intensive downstream defenses, such as certified sanitization mechanisms, content filtering using ensemble classifiers, or constrained decoding techniques [7, 27]. This staged approach ensures that the majority of benign or easily detectable adversarial cases are handled quickly and inexpensively, while reserving more costly defenses for inputs that warrant deeper scrutiny.

This described approach aligns with the Swiss cheese model [28], a well-established conceptual framework in risk management and safety engineering. In this model, no single defense layer is assumed to be fully robust; instead, multiple layers with different strengths and failure modes are employed in parallel, such that the overall system remains resilient even when individual layers are circumvented.

5 Conclusions

We introduced Adversarial Suffix Filtering (ASF), a lightweight, model-agnostic defense pipeline for detecting and removing adversarial suffixes in LLM prompts. ASF segments user inputs and classifies each segment to identify and neutralize malicious prompt continuations prior to inference, requiring no modifications to the underlying model. Through extensive evaluation, we demonstrate that ASF significantly reduces the attack success rate of current state-of-the-art suffix-based jailbreaks while preserving model performance on non-adversarial tasks. ASF achieves this with minimal computational overhead, making it deployable in real-world, resource-constrained environments.

While ASF effectively defends against a specific of prompt injection attacks, its current design assumes a clear segmentation between benign prompt and malicious suffix. Future investigations should broaden ASF's scope beyond pure suffix-style attacks to mixed-context, interleaved or prefix-based injections and to multilingual settings where segmentation cues differ substantially . Addressing the pipeline's present susceptibility to imperfect sentence splits and the attendant false-positive risk will likely require tighter, possibly joint, optimization of the segmentation and classification stages together with continual calibration on richer benign corpora.

Acknowledgments and Disclosure of Funding

David Khachaturov is supported by the University of Cambridge Harding Distinguished Postgraduate Scholars Programme.

References

[1] Zihao Xu, Yi Liu, Gelei Deng, Yuekang Li, and Stjepan Picek. A comprehensive study of jailbreak attack versus defense for large language models, 2024. URL https://arxiv.org/abs/2402.13457.

[2] OWASP. Llm01:2025 prompt injection, Apr 2025. URL https://genai.owasp.org/llmrisk/llm01-prompt-injection/.

[3] Zeyi Liao and Huan Sun. Amplegcg: Learning a universal and transferable generative model of adversarial suffixes for jailbreaking both open and closed llms, 2024. URL https://arxiv.org/abs/2404.07921.

[4] 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.

[5] Markus Frohmann, Igor Sterner, Ivan Vulić, Benjamin Minixhofer, and Markus Schedl. Segment any text: A universal approach for robust, efficient and adaptable sentence segmentation. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11908–11941, Miami, Florida, USA, November 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.emnlp-main.665.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

[7] Aounon Kumar, Chirag Agarwal, Suraj Srinivas, Aaron Jiaxun Li, Soheil Feizi, and Himabindu Lakkaraju. Certifying llm safety against adversarial prompting, 2025. URL https://arxiv.org/abs/2309.02705.

[8] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019. URL https://arxiv.org/abs/1706.06083.

[9] Mengnan Zhao, Lihe Zhang, Jingwen Ye, Huchuan Lu, Baocai Yin, and Xinchao Wang. Adversarial training: A survey, 2024. URL https://arxiv.org/abs/2410.15042.

[10] Sophie Xhonneux, Alessandro Sordoni, Stephan Günnemann, Gauthier Gidel, and Leo Schwinn. Efficient adversarial training in llms with continuous attacks, 2024. URL https://arxiv.org/abs/2405.15589.

[11] Lei Yu, Virginie Do, Karen Hambardzumyan, and Nicola Cancedda. Robust llm safeguarding via refusal feature adversarial training, 2025. URL https://arxiv.org/abs/2409.20089.

[12] Zhang Ze Yu, Lau Jia Jaw, Zhang Hui, and Bryan Kian Hsiang Low. Fine-tuning language models with generative adversarial reward modelling, 2024. URL https://arxiv.org/abs/2305.06176.

[13] Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. Struq: Defending against prompt injection with structured queries, 2024. URL https://arxiv.org/abs/2402.06363.

[14] Zhangchen Xu, Fengqing Jiang, Luyao Niu, Jinyuan Jia, Bill Yuchen Lin, and Radha Poovendran. Safedecoding: Defending against jailbreak attacks via safety-aware decoding, 2024. URL https://arxiv.org/abs/2402.08983.

[15] Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations, 2023. URL https://arxiv.org/abs/2305.14233.

[16] Alexander Robey, Eric Wong, Hamed Hassani, and George J. Pappas. Smoothllm: Defending large language models against jailbreaking attacks, 2024. URL https://arxiv.org/abs/2310.03684.

[17] Matthew Pisano, Peter Ly, Abraham Sanders, Bingsheng Yao, Dakuo Wang, Tomek Strzalkowski, and Mei Si. Bergeron: Combating adversarial attacks through a conscience-based alignment framework, 2024. URL https://arxiv.org/abs/2312.00029.

[18] 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.

[19] Gabriel Alon and Michael Kamfonas. Detecting language model attacks with perplexity, 2023. URL https://arxiv.org/abs/2308.14132.

[20] Neel Jain, Avi Schwarzschild, Yuxin Wen, Gowthami Somepalli, John Kirchenbauer, Ping yeh Chiang, Micah Goldblum, Aniruddha Saha, Jonas Geiping, and Tom Goldstein. Baseline defenses for adversarial attacks against aligned language models, 2023. URL https://arxiv.org/abs/2309.00614.

[21] 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.

[22] Vishal Kumar, Zeyi Liao, Jaylen Jones, and Huan Sun. Amplegcg-plus: A strong generative model of adversarial suffixes to jailbreak llms with higher success rates in fewer attempts, 2024. URL https://arxiv.org/abs/2410.22143.

[23] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

[24] Yangsibo Huang, Samyak Gupta, Mengzhou Xia, Kai Li, and Danqi Chen. Catastrophic jailbreak of open-source llms via exploiting generation. *arXiv preprint arXiv:2310.06987*, 2023.

[25] Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, et al. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal. *arXiv preprint arXiv:2402.04249*, 2024.

[26] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

[27] Neeraj Varshney, Pavel Dolin, Agastya Seth, and Chitta Baral. The art of defending: A systematic evaluation and analysis of LLM defense strategies on safety and over-defensiveness. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 13111–13128, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.776. URL https://aclanthology.org/2024.findings-acl.776/.

[28] J. Reason. The contribution of latent human failures to the breakdown of complex systems. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 327 (1241):475–484, 1990. ISSN 00804622. URL http://www.jstor.org/stable/55319.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Introduction (pp. 1–2) state that ASF is a lightweight, model-agnostic pipeline that mitigates suffix-based jailbreaks with minimal utility loss; Sections 3, 4 empirically support these claims.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 4.1 details segmentation errors, possible false positives, and scope restrictions to suffix-style attacks.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper is empirical; it presents no theorems or proofs, so no formal assumptions are required.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 2 (Datasets, Training) and Section 3 (Experimental setup) list data splits, preprocessing steps, hyper-parameters, and evaluation protocols, enabling independent replication.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: Section 3 notes that code and checkpoints will be released soon after submission, due to time constraints; at submission time they are not yet publicly available.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Hyper-parameters, SaT variant, classifier architecture, and data-split ratios (70/15/15) are reported in Section 2.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Although we do not report explicit error bars, each ASR measurement and accuracy delta is computed over hundreds of prompts (Malicious-Instruct, AdvBench, and multiple LM-Eval tasks). Given the large absolute performance gaps we observe after sanitization (e.g., ASR drops from around 80% to $< 4\%$), the conclusions are robust to this small statistical noise; additional runs would incur prohibitive GPU cost without affecting the qualitative outcome.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 3 specifies a cluster with 3xRTX8000 + 1xA10 GPUs and a three-week wall-time for the full experimental suite.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The study involves only publicly available text data and does not violate the NeurIPS Code of Ethics; no human subjects or sensitive data are used.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

Justification: Section 4 explicitly highlights the positive societal impact of ASF – namely, reducing the risk of harmful LLM outputs in real-world deployments – and acknowledges potential downsides such as false positives that could inconvenience legitimate users. This balanced treatment satisfies the broader-impact requirement.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: No new high-risk generative model or scraped dataset is released; ASF is a defensive wrapper around existing open assets.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All appropriate model and datasets are cited

Guidelines:

- The answer NA means that the paper does not use existing assets.

- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [No]

    Justification: When the trained ASF models will be released, structured documentation will be included in this release.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The work uses only machine-generated and publicly released text; no human subjects or crowd work were involved.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

Justification: No human-subject research is conducted; IRB approval is therefore not applicable.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: Sections 2 and 3 describe the use of pretrained LLMs (e.g. GPT-4.1-mini, Mistral-7B) both as attack targets and for benchmark evaluation.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.