

FLAME: Flexible LLM-Assisted Moderation Engine

Ivan Bakulin
AIRI, Moscow, Russia

Ilia Kopanichuk
AIRI, Moscow, Russia
MIPT, Dolgoprudny, Russia
kopanichuk@airi.net

Iaroslav Beshpalov
AIRI, Moscow, Russia

Nikita Radchenko
SberHealth, Moscow, Russia

Vladimir Shaposhnikov
AIRI, Moscow, Russia
Skoltech, Moscow, Russia

Dmitry V. Dylov
AIRI, Moscow, Russia
Skoltech, Moscow, Russia

Ivan Oseledets
AIRI, Moscow, Russia
MSU, Moscow, Russia

Abstract—The rapid advancement of Large Language Models (LLMs) has introduced significant challenges in moderating user-model interactions. While LLMs demonstrate remarkable capabilities, they remain vulnerable to adversarial attacks, particularly “jailbreaking” techniques that bypass content safety measures. Current content moderation systems, which primarily rely on input prompt filtering, have proven insufficient, with techniques like Best-of-N (BoN) jailbreaking achieving success rates of 80% or more against popular LLMs. Notably, although the training phase uses an LLM model to generate a variety of synthetic sentences to generate a blacklist, the inference mechanism itself is completely rule-based, relies on the matching of normalized n-grams, and does not require neural computation at runtime.

In this paper, we introduce Flexible LLM-Assisted Moderation Engine (FLAME): a new approach that shifts the focus from input filtering to output moderation. Unlike traditional circuit-breaking methods that analyze user queries, FLAME evaluates model responses, offering several key advantages: (1) computational efficiency in both training and inference, (2) enhanced resistance to BoN jailbreaking attacks, and (3) flexibility in defining and updating safety criteria through customizable topic filtering. Our experiments demonstrate that FLAME significantly outperforms current moderation systems. For example, FLAME reduces attack success rate in GPT-4o-mini and DeepSeek-v3 by a factor of ~ 9 , while maintaining low computational overhead. We provide comprehensive evaluation on various LLMs and analyze the engine’s efficiency against the state-of-the-art jailbreaking. This work contributes to the development of more robust and adaptable content moderation systems for LLMs.

I. INTRODUCTION

Content moderation for Large Language Models (LLMs) represents a critical challenge in ensuring safe and appropriate human-AI interactions. While traditional content moderation approaches have focused primarily on input filtering [1], the increasing sophistication of adversarial techniques necessitates a fundamental shift in how we approach this problem. The main purpose of content moderation is to limit the processing of requests that do not align with the system’s intended use. This serves several crucial functions: prevents errors in specialized applications like medical consultation, eliminates irrelevant or potentially harmful data that may interfere with model performance, and reduces the risks associated with malicious information or legal violations [2]. Current moderation

systems typically rely on input filtering to identify and block potentially harmful queries before they reach the model.

However, recent developments in jailbreaking techniques, particularly the Best-of-N (BoN) approach, have exposed significant vulnerabilities in existing moderation systems. These techniques exploit the probabilistic nature of LLM outputs through multiple sampling attempts, achieving concerning success rates of 80% or higher against popular models [3]. The effectiveness of these attacks highlights a critical gap in current defensive approaches, which predominantly focus on analyzing user inputs rather than model outputs.

Our work introduces FLAME (Flexible LLM-Assisted Moderation Engine), *shifting the focus from input filtering to output moderation* through an efficient regulatory policy. Unlike existing solutions that require extensive computational resources or complex neural architectures, FLAME employs a lightweight approach that can be deployed with minimal requirements. This design choice not only makes the system more accessible but also enables rapid adaptation to emerging threats through customizable topic filtering. The flexibility of FLAME’s architecture addresses one key limitation in current moderation systems: the ability to quickly adapt to new types of harmful content while maintaining efficient operation. Our approach allows organizations to define and update their moderation criteria based on specific needs and emerging challenges, without requiring significant retraining or computational resources.

Recent work on constitutional AI and classifier-based approaches [4] has demonstrated the potential of sophisticated moderation systems. However, these solutions often demand substantial computational resources for both training and inference. In contrast, FLAME demonstrates that effective moderation can be achieved through carefully designed rule-based systems enhanced by LLM-generated training data. Through extensive experimentation and real-world deployment, we have validated this approach across multiple leading LLM platforms, consistently achieving a 2-9 \times improvement in resistance to BoN attacks.

Contributions This work advances the field of LLM content moderation in several ways.

- We introduce an *output-centered* moderation approach that provides superior protection against state-of-the-art jailbreaking techniques while maintaining minimal computational requirements.
- We jailbreak 6 popular LLMs with and without our moderation engine, demonstrating an *up to 9-fold improvement in their resistance to adversarial attacks*.
- Our engine challenges the prevailing trend towards resource-intensive censorship, demonstrating that effective moderation can be achieved without extensive model fine-tuning or complex neural architectures.
- We report practical insights from deployment of the moderation engine into a dialogue system product, addressing critical considerations from the standpoint of user experience and finding the delicate balance between moderation strictness and system accessibility.

II. RELATED WORK

Classifier guards. Markov et al. [5] proposed an active learning strategy that identifies relevant samples for labeling, balances between uncertainty and diversity, and leverages redundancy to capture rare events. Rebedea et al. [6] developed guardrails control LLM output, preventing harmful topics, following dialogue paths, and maintaining language styles. Chi et al. [7] introduced multimodal LLM-based safeguard that classifies content as safe or unsafe based on user-provided guidelines, conversation context, and output formats. Kim et al. [8] highlight the necessity of developing better definitions for unsafe outputs. Wang et al. [9] point to challenges in jailbreak defense even in narrow contexts and suggest future directions involving classifier calibration and human feedback integration. Sharma et al. [4] proposed "Constitutional Classifiers", which use natural-language rules to train classifier safeguards defining what constitutes permitted and restricted content.

Jailbreaking. Lapid et al. [10] developed a black-box Genetic Algorithm (GA) to manipulate LLMs. Huang et al. [11] show that manipulation of generation strategies, such as removing system prompts and altering decoding parameters, can easily disrupt model alignment. Samvelyan et al. [12] introduced "Rainbow Teaming" methodology that defines features like risk category and attack style to diversify prompts and rank them based on their effectiveness. Doumbouya et al. [13] developed an open-source automated red-teaming platform for generating and analyzing jailbreak attacks. Andriushchenko et al. [14] applied prompt templates, random suffix search, self transfer from easier tasks, transfer and prefilling attacks and showed that adaptive attacks are necessary to accurately assess LLM robustness. Hughes et al. [3] proposed "Best-of-N (BoN) Jailbreaking," a black-box algorithm and demonstrated its effectiveness across text, vision, and audio language models, achieving high Attack Success Rates (ASR).

In general, existing classifier-based defenses mainly analyze user inputs and often require significant computational resources for both training and inference. Meanwhile, jailbreak techniques continue to evolve rapidly, with approaches such as

BoN demonstrating a near 100% success rate in attacking popular models. This creates a clear need for a lightweight, output-oriented moderation approach that can be flexibly adapted to new threats without costly retraining and it is this gap that FLAME addresses.

III. METHOD

FLAME operates in two separate phases: an offline *learning* phase, where the LLM is used to generate a variety of synthetic messages that are processed into a blacklist of banned n-grams; and an online *output* phase, where a purely rule-based classifier checks the model output against this blacklist. It is important to note that the LLM is only involved in the training phase - the inference engine does not require neural computation and operates with minimal resources. This separation allows FLAME to be both flexible (new topics can be added by re-generating the blacklist) and efficient (the output is a simple n-gram search).

The algorithm 1 uses several sub-functions, whose logic we will summarize here. The GENERATETOPICVARIATIONS function takes a "sensitive" topic and, using an unmoderated LLM, generates $n = 30$ different formulations of user queries aimed at retrieving content on that topic, including jailbreak-style reformulations. Then CREATSEMANTICMESSAGES runs the same LLM to generate $k = 20$ complete model responses for each variation, yielding a vast corpus of messages to be tagged by the moderator system. GENERATEINITIALBLACKLIST extracts all normalized n -grams (for $n \in \{1, 2, 3\}$) from this corpus and collects them into a candidate blacklist. Finally, VALIDATEONTRAININGSET checks each candidate n -gram against the training set of safe dialogs: if an n -gram causes a false positive on any safe message, it is removed from the blacklist.

The moderation Algorithm 1 checks whether a user request or a model response contain a banned topic. It is a binary classification problem: 0 – no banned topics, 1 – contains banned topic.

The inference algorithm itself is rule-based and relatively simple. Message (text string) r is split into several multisets of n -grams of normal word forms, by the split function $S(r, n)$. Then, the classifying function σ is computed as follows:

$$\sigma(r) = \begin{cases} 1 & \text{if } \sum_{n=1}^k \chi(S(r, n), T) > 0 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where χ is the characteristic function computed by the exact match of items, T is the set of all banned n -grams (*i.e.*, `blacklist` in Algorithm 1), $k = 3$ is the maximum n -gram size. We used *pymorphy3* [15] to normalize word forms and *nltk* [16] for the n -gram partitioning. An example of a message split from a medical dialogue is shown below:

$$S(\text{My stomach hurts}, 2) = \llbracket \text{i stomach,} \\ \text{stomach hurt} \rrbracket$$

Before discussing the details, we want to note that the engine itself is flexible. Topic selection in the example above

Algorithm 1 FLAME Training Pipeline

Require: topics - list of sensitive topics to filter
Require: training_dialogs - collection of safe dialogues
Ensure: blacklist - set of filtered phrases

```

1: function GENERATEMESSAGES(topics)
2:   messages  $\leftarrow$  []
3:   for all topic  $\in$  topics do
4:      $\triangleright$  Generate variations using LLM
5:     variations  $\leftarrow$  GENERATETOPICVARIATIONS(topic, n = 30)
6:      $\triangleright$  Create semantic neighbors for robustness
7:     messages  $\leftarrow$  messages  $\cup$  CREATSEMAN-
      TICMESSAGES(variations, k = 20)
8:   end for
9:   return messages
10: end function

11: function ASSEMBLETRAININGCOLLECTION(dialogs)
12:   training_set  $\leftarrow$  []
13:   for all dialog  $\in$  dialogs do
14:     training_set  $\leftarrow$  training_set  $\cup$  DIALOG-
      TOMESSAGES(dialog)
15:   end for
16:   return training_set
17: end function

18: function CREATEBLACKLIST(messages, training_set)
19:    $\triangleright$  Initial blacklist generation
20:   raw_blacklist  $\leftarrow$  GENERATEINITIALBLACK-
      LIST(messages)
21:    $\triangleright$  Clean and optimize blacklist
22:   refined_blacklist  $\leftarrow$  {}
23:   for all phrase  $\in$  raw_blacklist do
24:     if VALIDATEONTRAININGSET(phrase,
      training_set) then
25:       refined_blacklist  $\leftarrow$  refined_blacklist  $\cup$ 
        {phrase}
26:     end if
27:   end for
28:   return refined_blacklist
29: end function

30:    $\triangleright$  Main execution pipeline
31: messages  $\leftarrow$  GENERATEMESSAGES(topics)
32: training_data  $\leftarrow$  ASSEMBLETRAININGCOLLEC-
      TION(training_dialogs)
33: blacklist  $\leftarrow$  CREATEBLACKLIST(messages,
      training_data)

```

allows for skipping domain-irrelevant topics while filtering out dangerous or forbidden subjects. One can choose any sets of such topics when adapting the engine to their needs.

A. Assembly of set of banned n -grams

The proposed method for creating a dataset for text classification is similar to the constitutional classifier created in [4]. Unlike the constitutional classifier, based on LLM fine-tuning, FLAME works on classical methods for matching normalized forms of n -grams, so it is much more efficient in both training and inference. **Our method requires no GPUs for training.** One merely needs an LLM API (without moderation) and 1 CPU to get through a full training cycle (Algorithm 1) in a few hours.

The assembly of a set T of banned n -grams consists of several sequential steps:

- 1) topics selection,
- 2) messages generation,
- 3) messages preprocessing,
- 4) n -grams filtration.

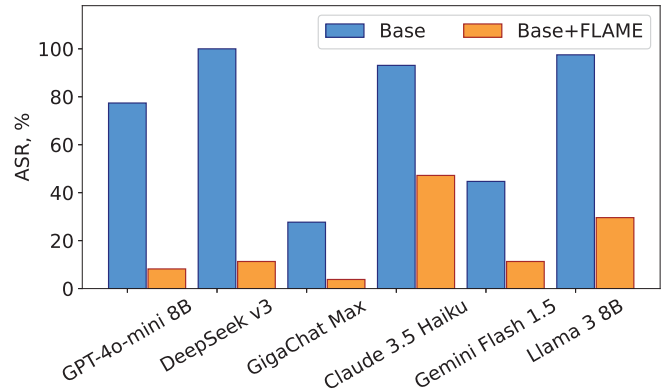


Fig. 1. Attack success rate of BoN jailbreaks [3] on moderation systems of different LLMs, showcasing the resilience that FLAME adds to popular LLMs.

The first step was to select the topics we want to avoid in the model response. Once the set of topics has been established, each topic is matched with a set of user queries. Each query contains jailbreak attempt and used as an example for generation in the next step. We use the following criteria to select the topics and queries for them:

- Topics related to activities that violate the law or international rights: terrorism, extremism, violence, *etc.*
- The system is focused on helping with medical advice, so topics outside of domain (e.g., esotericism, cooking, programming) have been added to the list.
- Topics that may cause social stigma or inconsistency with the goals of the system are included. For example, bans on political discussions are associated with increased social sensitivity and legal restrictions

The main challenge was to strike a balance between filtering and usefulness. For example, words such as “alcohol” or “drug” have both medical relevance and irrelevant context.

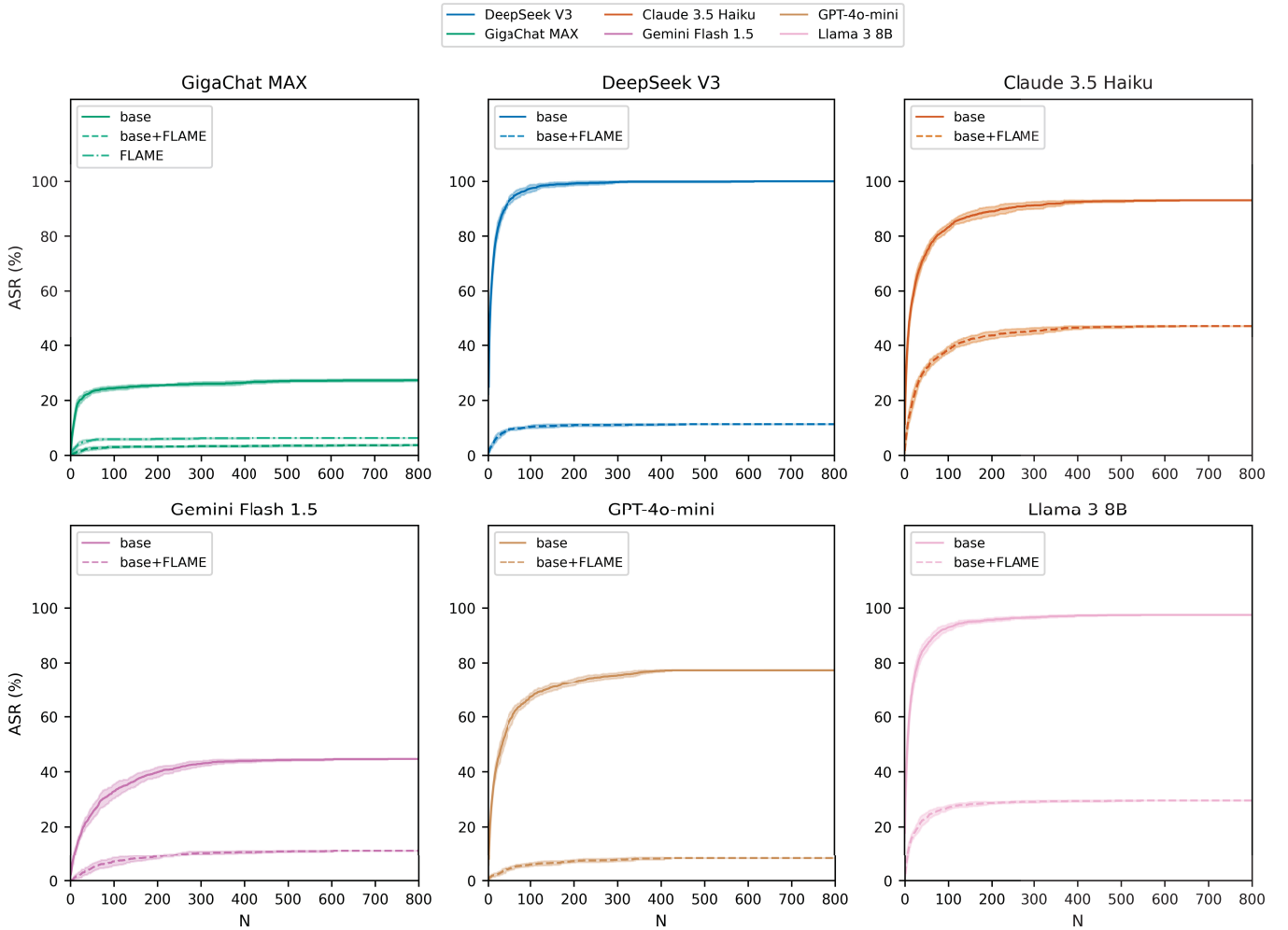


Fig. 2. The dependence of attack success rate on the number of attempts by BoN for different LLMs compared to FLAME

In the second step, we generated a set of messages \mathcal{C} for each topic using the unmoderated LLM API by GigaChat Max. We used pre-made examples and prompts with jailbreak attempts from the previous step.

In the third step, we split each message C into multiset of n -grams of normal word forms as $W(C, n), n \in \{1, \dots, k\}$, $k = 3$ is the maximum n -gram size. We formed the union multiset of all generated n -grams:

$$\mathfrak{T} = \bigcup_{j=1}^{|\mathcal{C}|} \bigcup_{n=1}^k W(C_j, n) \quad (2)$$

The most important part of the selection of banned n -grams is to filter the resulting multiset from the previous steps. We removed all non-frequent or short n -grams and made a set:

$$T = \{\forall g \in \mathfrak{T} : \mu_{\mathfrak{T}}(g) > k_{min} \vee |g| > l_{min}\} \quad (3)$$

where μ is a multiplicity function, $k_{min} = 5, l_{min} = 4$. The disjunction serves two complementary purposes: short n -grams (e.g., unigrams or bigrams) are retained only if they appear frequently enough to be reliable indicators of sensitive content, while longer n -grams (trigrams with $|g| > l_{min}$ carry sufficient

lexical specificity to be informative even at lower frequencies. This two-pronged criterion balances recall (via frequent short phrases) with precision (via specific longer phrases). Then we used a train collection: each element of the set T was checked to see if it was responsible for the result when the algorithm false positively triggered on at least one element of the collection. If so, the element was removed from the set. The train collection contains 20000 messages and 100% of them are negatively labeled.

B. Specifics of moderating in real chat room

After deploying the first version of our solution in a production run, we found that despite the low false positive rates the actual number of reported chat sessions with false positive errors was about 1.8 times higher than FPR. When working with real chat rooms, it should be kept in mind that counting moderation quality metrics on messages does not reflect the real user experience. The user does not count metrics on messages, but evaluates the whole interaction session with the LLM. Even one false positive evaluation of a message spoils the interaction experience for the whole session. In order to

estimate the probability of unsuccessful session for a user, we used Bernoulli’s formula:

$$P_t = 1 - (1 - FPR)^t \quad (4)$$

FPR is a false positive rate of the moderation by messages, *P* is a probability of at least one false positive moderator activation during a session of *t* messages length. It is not hard to estimate that for a chat of length 5 messages, with our initial *FPR* of 1%, we get a 4.9% probability of false moderator activation. If, however, we check both the user message and the model response, rather than just the user response, the probability of an undesirable outcome increases to 9.5%. In practice, five and ten times the number of false positive errors is not achieved because inference sampling is heavily biased towards safe use of the dialogue system. However, this means that in real chat rooms, one has to be very careful with moderation; otherwise, the number of law-abiding users affected by overly active moderation may exceed acceptable limits [17].

C. Implementation details

The computational complexity of the characteristic function of two sets depends linearly on the length of the shortest set. The set of forbidden *n*-grams contains about 10^5 or more elements, while the number of words in the processed sequence is in a range of 10 – 1000 words. The computational complexity of FLAME on inference depends linearly on the length of the model output. In production it takes 2 to 5 ms to check 1 message (4.3 ms on average at the real chat room) using only 0.1 CPU core and 100 Mb RAM.

IV. RESULTS

Our test collection contains 9178 messages. The collection is balanced in terms of classes: 54% of samples have a positive label. Metrics on the test collection are shown in Table I.

Standard error for all metrics calculated by bootstrapping. The metrics shown in Table I are good enough to deploy the method in production. However, in order to verify the quality of the proposed solution, we conducted a series of experiments to compare the effectiveness in repelling the latest SOTA jailbreak on popular LLMs – best-of-n (BoN) jailbreak [3]. Enough time has passed since the original article with the BoN was released that the LLM bot holders have had time to issue some sort of response to it. Therefore, we also provide data on the current state of the moderation quality of APIs of various LLM chat bots. We used the methodology described in [3], with one exception. Because FLAME was designed and implemented for a Russian-language medical dialog system, we translated the BoN assessment dataset from English to Russian using a professional translation. It is important to note that the *n*-gram blacklist was generated directly from the output data of the Russian-language model (see Section III) and *not* translated from English; only the evaluation prompts were translated to match the implementation language.

The results of the experiments are presented in Figures 1 and 2. Table II presents the maximum achieved ASR values for each LLM with and without FLAME. FLAME worked relatively well for DeepSeek and ChatGPT, showing 9 times more effective resistance to attacks than the moderation system built into the their own API. In the case of DeepSeek, the BoN jailbreaking method achieved 100% success, and quite quickly. Indeed, a check of the model output showed that their “constitution” of the moderation system differs significantly from the other models. DeepSeek comfortably chats about sensitive political topics as long as they do not involve recent Chinese history. The worst performance was shown for Claude – jailbreak achieved its goal even with the presence of FLAME in almost half the cases which is more than for any other LLM. Taking into account their recent article [4], Anthropic appear to be in the process of redesigning their moderation system.

The best absolute performance was shown for GigaChat. We also compared the quality of the built-in GigaChat moderation system with the pure FLAME. Indeed, combining moderation systems gives a slightly higher result in resistance to attacks. However, in absolute terms it is insignificant (see Figure 2). The effect of the accumulation of the probability of false positive errors during the chat session described in the section above makes this idea very risky in terms of the quality of the user experience.

V. DISCUSSION

Our experimental results reveal several critical insights about the current state and future directions of LLM content moderation. First, the varying effectiveness of FLAME across different models illuminates important patterns in moderation system design. The superior performance with GigaChat demonstrates the value of model-specific training, while the challenges faced with Claude (47.2% attack success rate) highlight how architectural differences in LLMs can impact moderation effectiveness.

The difference between the result shown by FLAME for GigaChat and for the other models (see Figure 1) underscores the importance of knowing which model the engine is trained on and which one it infers. It is not overly surprising that FLAME showed the best quality on the same model, whose answers were used to train it.

The real-world deployment of FLAME has provided valuable insights into practical implementation challenges. One observation is that the false positive rates in production environments can be 1.8 times higher than in isolated testing, emphasizing the vital role of considering complete user sessions rather than individual interactions. This finding fundamentally changes how we should approach evaluation of moderation systems and their optimization and should be a subject of a benchmarking effort in future work.

TABLE I. METRICS OF MODERATION QUALITY BY INDIVIDUAL MESSAGE

Precision	Recall	F_1	FPR	Support
98.7±0.02%	90.9±0.04%	94.7±0.02%	1.38±0.02%	9178

TABLE II. COMPARISON OF THE BoN ATTACK SUCCESS RATE (ASR) ON MODERATION SYSTEMS OF DIFFERENT LLMs WITH THAT OF FLAME

Model	Base ASR	FLAME ASR	B/F ASR ratio
GigaChat Max	27.7±2.2%	3.8±0.8%	7.3
DeepSeek v3	100.0±3.0%	11.3±1.3%	8.9
Claude 3.5 Haiku	93.1±3.3%	47.2±2.9%	2.0
Gemini Flash 1.5	44.7±3.1%	11.3±1.6%	4.0
GPT-4o-mini 8B	77.4±3.6%	8.2±1.0%	9.4
Llama 3 8B	97.5±4.4%	29.6±2.3%	3.3

Analysis of combined moderation approaches yielded unexpected insights. While integrating FLAME with existing systems showed marginal improvements in attack resistance, the multiplicative effect on false positives suggests that simpler, focused approaches may be more efficient in practice. This challenges the common assumption that layering multiple security measures necessarily improves overall system safety. The production deployment also revealed interesting patterns in user interaction and system performance under real-world conditions. The relationship between chat session length and cumulative false positive rates provides a recipe on how moderation systems should be calibrated for different use cases. These insights extend beyond FLAME’s specific implementation and are positioned to influence broader design principles of moderation systems.

A. Limitations

FLAME is inexpensive to train and infer, showing acceptable quality on the test sample, and is highly resistant to the SOTA attack method. However, it also has limitations. Firstly, its performance results strongly depend on the difference between the model used during training and the one that will be used in inference. One requires the engine to be trained separately on a model on which it will be used. Secondly, its training requires access to an unmoderated version of the model, which are not always available.

VI. CONCLUSION

FLAME represents a significant advancement in LLM content moderation, demonstrating that efficient protection against modern jailbreaking techniques can be achieved through lightweight but powerful approaches. Our comprehensive evaluation across multiple leading LLM platforms shows that FLAME consistently reduces attack success rates by a factor of 2–9 compared to existing solutions, while maintaining minimal computational requirements of just 0.1 CPU core and 100 MB RAM per instance. The system’s success in real-world deployment validates our approach of shifting focus from input filtering to output moderation. This paradigm shift, combined with our rule-based architecture enhanced by LLM-generated training data, challenges the prevailing trend

toward increasingly complex and resource-intensive censorship. The results demonstrate that successful moderation can be achieved without extensive model fine-tuning or complex neural architectures. Our work establishes a new direction for developing practical, scalable content moderation systems, protecting against adversarial attacks and providing computational efficiency and a true flexibility in deployment. As LLMs continue to evolve and integrate into all sorts of applications, approaches like FLAME will be crucial in ensuring safe and appropriate human-AI interactions.

REFERENCES

- [1] T. Huang, “Content moderation by llm: from accuracy to legitimacy,” *Artificial Intelligence Review*, vol. 58, no. 10, p. 320, Jul 2025. [Online]. Available: <https://doi.org/10.1007/s10462-025-11328-1>
- [2] J. Chua, Y. Li, S. Yang, C. Wang, and L. Yao, “Ai safety in generative ai large language models: A survey,” 2024. [Online]. Available: <https://arxiv.org/abs/2407.18369>
- [3] J. Hughes, S. Price, A. Lynch, R. Schaeffer, F. Barez, S. Koyejo, H. Sleight, E. Jones, E. Perez, and M. Sharma, “Best-of-n jailbreaking,” 2024. [Online]. Available: <https://arxiv.org/abs/2412.03556>
- [4] M. Sharma, M. Tong, J. Mu, J. Wei, J. Kruthoff, S. Goodfriend, E. Ong, A. Peng, R. Agarwal, C. Anil, A. Askill, N. Bailey, J. Benton, E. Bluemke, S. R. Bowman, E. Christiansen, H. Cunningham, A. Dau, A. Gopal, R. Gilson, L. Graham, L. Howard, N. Kalra, T. Lee, K. Lin, P. Lofgren, F. Mosconi, C. O’Hara, C. Olsson, L. Petrini, S. Rajani, N. Saxena, A. Silverstein, T. Singh, T. Summers, L. Tang, K. K. Troy, C. Weisser, R. Zhong, G. Zhou, J. Leike, J. Kaplan, and E. Perez, “Constitutional classifiers: Defending against universal jailbreaks across thousands of hours of red teaming,” 2025. [Online]. Available: <https://arxiv.org/abs/2501.18837>
- [5] T. Markov, C. Zhang, S. Agarwal, T. Eloundou, T. Lee, S. Adler, A. Jiang, and L. Weng, “A holistic approach to undesired content detection in the real world,” 2023. [Online]. Available: <https://arxiv.org/abs/2208.03274>
- [6] T. Rebedea, R. Dinu, M. Sreedhar, C. Parisien, and J. Cohen, “Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails,” 2023. [Online]. Available: <https://arxiv.org/abs/2310.10501>
- [7] J. Chi, U. Karn, H. Zhan, E. Smith, J. Rando, Y. Zhang, K. Plawiak, Z. D. Coudert, K. Upasani, and M. Pasupuleti, “Llama guard 3 vision: Safeguarding human-ai image understanding conversations,” 2024. [Online]. Available: <https://arxiv.org/abs/2411.10414>
- [8] T. Kim, S. Kotha, and A. Raghunathan, “Testing the limits of jailbreaking defenses with the purple problem,” 2024. [Online]. Available: <https://arxiv.org/abs/2403.14725>
- [9] T. T. Wang, J. Hughes, H. Sleight, R. Schaeffer, R. Agrawal, F. Barez, M. Sharma, J. Mu, N. Shavit, and E. Perez, “Jailbreak defense in a narrow domain: Limitations of existing methods and a new transcript-classifier approach,” 2024. [Online]. Available: <https://arxiv.org/abs/2412.02159>

- [10] R. Lapid, R. Langberg, and M. Sipper, "Open sesame! universal black box jailbreaking of large language models," 2024. [Online]. Available: <https://arxiv.org/abs/2309.01446>
- [11] Y. Huang, S. Gupta, M. Xia, K. Li, and D. Chen, "Catastrophic jailbreak of open-source llms via exploiting generation," 2023. [Online]. Available: <https://arxiv.org/abs/2310.06987>
- [12] M. Samvelyan, S. C. Raparthy, A. Lupu, E. Hambro, A. H. Markosyan, M. Bhatt, Y. Mao, M. Jiang, J. Parker-Holder, J. Foerster, T. Rocktäschel, and R. Raileanu, "Rainbow teaming: Open-ended generation of diverse adversarial prompts," 2024. [Online]. Available: <https://arxiv.org/abs/2402.16822>
- [13] M. K. B. Doumbouya, A. Nandi, G. Poesia, D. Ghilardi, A. Goldie, F. Bianchi, D. Jurafsky, and C. D. Manning, "h4rm3l: A dynamic benchmark of composable jailbreak attacks for llm safety assessment," 2024. [Online]. Available: <https://arxiv.org/abs/2408.04811>
- [14] M. Andriushchenko, F. Croce, and N. Flammarion, "Jailbreaking leading safety-aligned llms with simple adaptive attacks," 2024. [Online]. Available: <https://arxiv.org/abs/2404.02151>
- [15] M. Korobov. (2024) pymorphy3. [Online]. Available: <https://github.com/kmike/pymorphy>
- [16] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.
- [17] S. F. Schwemer, "Decision quality and errors in content moderation," *IIC - International Review of Intellectual Property and Competition Law*, vol. 55, no. 1, pp. 139–156, Jan 2024. [Online]. Available: <https://doi.org/10.1007/s40319-023-01418-4>