DETOXIFICATION: SELF SUPERVISOR, EXTERNAL MONITOR AND ADVERSARIAL TRAINED SYSTEM PROMPT

Wanyi Ling
School of Statistics
University of Chicago
Chicago, IL 60637
wanyiling@uchicago.edu

Zhiyuan Chen
School of Statistics
University of Chicago
Chicago, IL 60637
zhiyuanc@uchicago.edu

June 1, 2024

ABSTRACT

Large Language Models (LLMs) exhibit remarkable capabilities across diverse applications. However, their potential to generate toxic content remains a significant concern. We introduce a novel methodology combining Self Supervisor, External Monitor, and Adversarial Trained System Prompt to prevent toxic generation while maintaining engagement. The Self Supervisor method involves the LLM evaluating its outputs for toxicity and regenerating responses if necessary. The External Monitor uses the PERSPECTIVE API to assess and guide the detoxification process. Furthermore, the Adversarial Trained System Prompt leverages adversarial training techniques to iteratively refine prompts and enhance the model's defense mechanisms. Our empirical studies using the REALTOXICITYPROMPTS dataset demonstrate the effectiveness of our methods in significantly reducing toxic outputs across various LLMs, including GPT-3.5-turbo, GPT-4o, Llama-3-8b, and Vicuna-1.5-7b. The results highlight the robustness and transferability of our approach, presenting a comprehensive solution for mitigating toxicity in LLMs.

1 Introduction

Large Language Models (LLMs) exhibit remarkable capabilities across diverse applications. However, concerns regarding their security persist, particularly regarding the generation of toxic content such as abuse, threats, misinformation, hate speech, or discrimination. Unlike harmful outputs induced by deliberate jailbreaking prompts, Gehman et al. show that models like GPT-3 can generate highly toxic sentences even when prompted with non-toxic sentences [5]. This highlights the need for effective strategies to prevent LLMs from generating toxic content.

Our goal is to prevent toxic generation, as well as the over-defensiveness of toxicity. That means we also want to avoid situations where LLMs consistently refuse to engage with prompts, often replying with "Sorry, I can't assist with that." Since not all toxic content stems from intentional attacks, it is crucial for LLMs to provide accurate and non-toxic responses. To address this, checking the output and giving feedback is a good idea. We propose a method called **Self Supervisor**, which involves the LLM checking its output for toxicity. If the response is judged non-toxic, it is output directly; otherwise, the LLM generates a new, non-toxic response. Additionally, we can use the Perspective API's toxicity scores as feedback [1], acting as an **External Monitor**. As a a widely used, commercially deployed toxicity detection tool, PERSPECTIVE API could play a role as golden rule on this problem.

Beyond real-time supervision, we also explore a prompt-based method inspired by adversarial training. We propose a method called **Adversarial Trained System Prompt**, in which two LLMs are employed: one acts as an Attack Model to generate and refine prompts that might produce toxic content, while the other serves as a Defense Model to analyze and defend against these prompts. By iterating this process, the Defense Model extracts insights and modifies the system prompt to better guide the Target Model. The Perspective API is used to classify responses as toxic or non-toxic, ensuring continuous improvement and robust defense against toxic outputs.

[1] https://github.com/conversationai/perspectiveapi

2 Related works

Toxicity and Detoxification Toxicity in LLMs is generation of rude, disrespectful, or unreasonable text that would make someone want to leave a conversation. Since downstream users may include younger or more vulnerable audiences and unintended outputs for given tasks are undesirable, Detecting and preventing toxicity is essential for LLMs. Gehman et al. proposed two detoxification techniques[5]. One is **Data-Based Detoxification**, which relies on further fine-tuning using a non-toxic subset of a balanced corpus. The other is **Decoding-Based Detoxification**, which only change the generation strategy without changing model parameters. Decoding-Based Detoxification methods include Vocabulary Shifting, Word Filtering and PPLM[4].

Jailbreak Defense Unlike detoxification, developing defenses against jailbreak attacks is an emerging field. We can draw inspiration from defense techniques against jailbreaks to prevent toxic content. Varshney et al. summarize jailbreak defense strategies for LLMs into the following categories: Safety Instruction, In-Context Exemplars, Self-Safety Check of Input, Self-Safety Check of Output, Including Unsafe Prompts in Instruction Tuning, and Contextual Knowledge [9]. Zhou et al. categorize jailbreak defense strategies into filtering, prompt editing, fine-tuning, and implementation of safety instructions [12]. Key contributions to our method include the concepts of self-supervision and adversarial process training. Self-supervision leverages the in-context learning ability of LLMs. Phute et al. propose a self-examination method to detect harmful responses in jailbreak defense [7]. Xie et al. introduce self-reminder, which encapsulates the user's query in a system prompt that reminds LLMs to respond responsibly. Adversarial training process further helps LLMs learn out-of-distribution knowledge through in-context learning. Chao et al. [2] use an attacker LLM to automatically generate jailbreaks for a target LLM, iteratively refining the jailbreak through repeated queries, usually succeeding within 20 attempts. Zhou et al. [12] design an In-Context Adversarial Game to defend against jailbreak prompts, employing an iterative process to enhance both defense and attack agents.

3 Methods

3.1 Self Supervisor

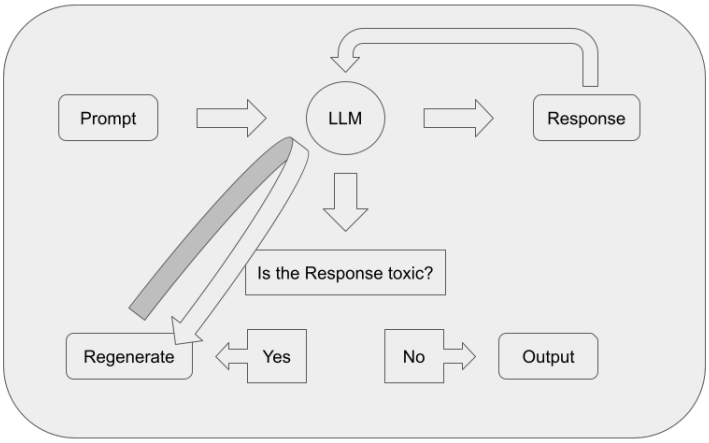

Figure 1: Self Supervisor

In Fig 1, we illustrate the construction of our Self Supervisor Process. First, the prompt is input into the LLM, which generates a response. We then ask the LLM to assess whether the response is toxic. If the response is not toxic, it is used as the output. If the response is toxic, we instruct the LLM to generate a non-toxic response to the same prompt, and this new response is used as the output.

3.2 External Monitor

In Fig 2, we illustrate the construction of our External Monitor Process. The key difference in this process is that we use the PERSPECTIVE API, rather than the LLM itself, to determine whether the response is toxic. The threshold for toxicity is set to a score greater than 0.5.

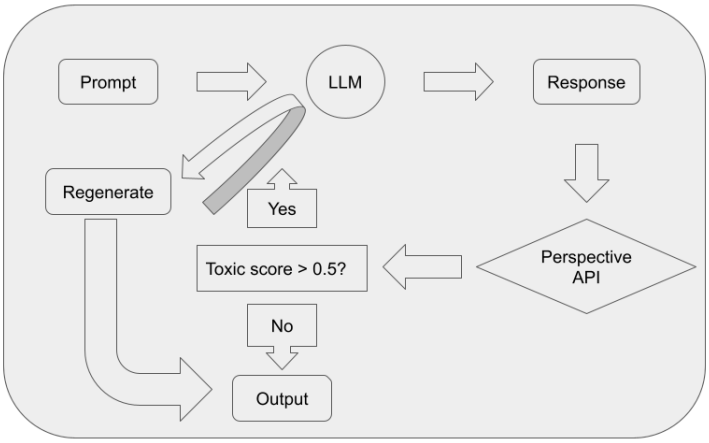

Figure 2: External Monitor

3.3 Adversarial Trained System Prompt

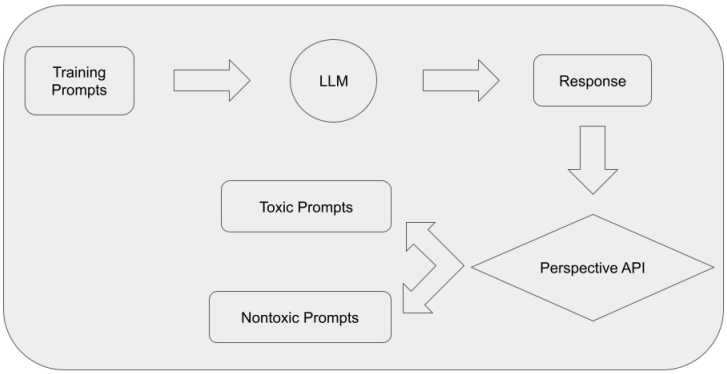

Figure 3: Use Perspective API as a Classifier

In this section, we show how to build our Adversarial Trained Process to help Target Model learn to not generating toxic responses. We use two LLMs, one acts as an Attack Model and another acts as a Defense Model. The goal of the Attack Model is to extract the attack insight and refine those prompts that failed to generate toxic content with the extracted attack insight. By sending those refined prompts into the Defense Model, we then extract defense insight for each successfully defended prompt and compress those insights as a guide for the next iteration. By several iteration, we will generate a informative and instructive insight that guides the model.

For each of the iteration, we first generate the responses for our current training set and current system prompt with the Target Model, and use Perspective API to separate it into a toxic set and a nontoxic set. This procedure is shown in Fig 3. After this initial set up, we will first go through the Attack Model.

3.3.1 Attack Model

The procedure of our Attack Model is shown in Fig 4.

Attack insight extraction After getting the toxic set and the nontoxic set, we pair each nontoxic prompts from the nontoxic set with a randomly selected toxic prompts from a subset of the toxic set. The subset selects the prompts with top-5 toxicity scores. Then we input each pair into the Attack Model to collect insight of why nontoxic prompts failed

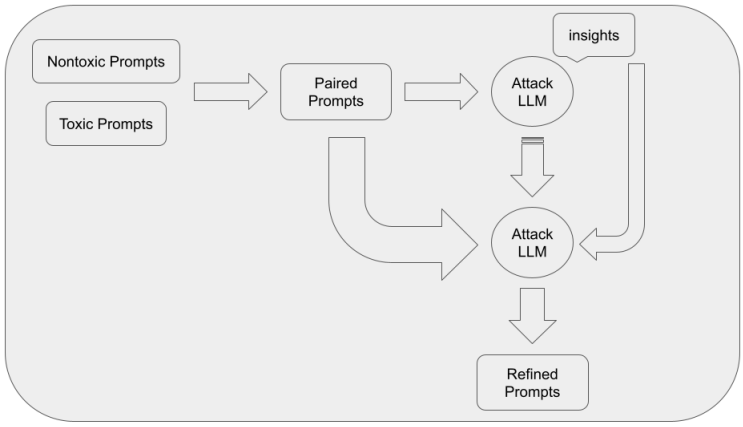

Figure 4: Attack Model

to generate toxic content compared to the toxic prompts. We put all insights in a pool, and randomly select one attack insight from the pool for each pair of prompts as a refining guide.

Refine prompts We input each group of nontoxic prompt, toxic prompt, and the attack insight with an organized prompt into our Attack Model again, and ask it to generate a refined prompt that will be more likely to generate toxic content in out Target Model. After this procedure, we will receive a set of refined prompts. This prompt will be inputted into the Defense Model as well as be used as the training set for the next iteration.

3.3.2 Defense Model

The procedure of our Defense Model is shown in Fig 5.

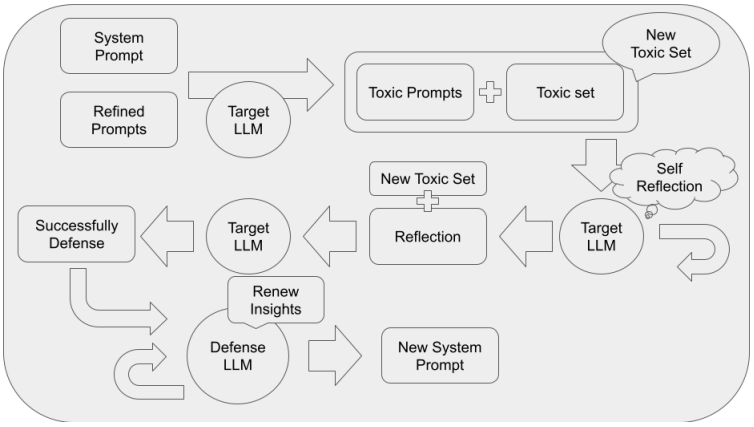

Figure 5: Defense Model

Defense reflection After receiving the refined prompts from the attack model, we will also go through the procedure in Fig 3 with the Target Model to separate it into a refine toxic set and a refine nontoxic set. By combining the original toxic set and this refine toxic set, we receive a combined toxic set. Using this combined toxic set, we implement a defense reflection procedure with the Target Model itself [8], asking it to analyze why these prompts failed to generate nontoxic content. Adding the reflection to the corresponding prompts, we go through the Target Model again and only leaving those that have been successfully defended after adding the reflection.

Defense insight extraction Those defended reflections and prompts will be used as the input of our Defense Model. For each reflection and its corresponding prompt, the Defense Model will extract the defense insight from it, and use this insight to modify our current system prompt. The goal of this modification is to combine as more as possible

defense insight into the system prompt will remaining it to be concise and instructive. After combining all defense information, the last output would be the system prompt for the next iteration.

4 Experiments

4.1 Evaluation set up

4.1.1 Datasets

We use the standard benchmark dataset REALTOXICITYPROMPTS [5] for toxicity evaluation. We only focus on a subset of REALTOXICITYPROMPTS, which includes 1.2K toxic prompts flagged as "challenging" by the official dataset. We split 200 prompts as the evaluation dataset and remaining 999 prompts as the training dataset.

4.1.2 Defense methods

For self supervisor and external monitor, there is no need to do any training. Target Models are the same in all experiments.

- **Real-time Supervision** These two defense methods are working in the conversation process, so the experiments are all conducted on test dataset.
- **One-shot Regeneration** To save time in the conversation, the detection will only be performed once. Regardless of whether the response regenerated is toxic, it will be directly output.

For adversarial trained system prompt method, we have the following set up for the training phase.

- **Target Models** We implement our method in 4 LLMs, including GPT-3.5-turbo[6], GPT-4o, Llama-3-8b[1], Vicuna-1.5-7b[3]. By evaluating our method on both open-sourced and closed-sourced models, we will be able to see how well our method is adaptive to different models.
- **Attack Model** For the Attack Model, we want it to have great ability in attack insight extraction and has the ability to refine the prompts while not refusing the request, so we need the model to be proficient in reasoning while not so sensitive to toxic input. Therefore, we choose GPT-3.5-turbo as our attack model.
- **Defense Model** For the Defense Model, it should also have outstanding ability in reasoning and reorganizing the insights into a concise but instructive guide. Based on this need, we choose the latest published GPT-4o to be our efense Model, which has shown its ability to be productive and informative.
- **Goal** In this adversarial training process for the system prompt, we hope our method will achieve convergence spanning several iterations. In this "training" phase, the **Insight** extracted by the Defense Model are integrated as system prompts for the target Model, aiming to generate benign outputs.
- **Training Procedure** For each iteration, we goes through the Attack Model, Defense Model, and the evaluation phase, and preserve the refined prompts and renewed insight for the next iteration. we need to check whether this method will achieve convergence over several iterations.

4.1.3 Attack methods

To see which method is more effective towards different attacks, we evaluate each method with two attacking methods. The first method is just the simple toxic prompts, and the second is adapted from the in-context attack method[11], which combines our toxic prompt with few-shot demonstrations before the original system prompt.

4.1.4 Evaluation metric

As we have stated before, we will use the toxicity score evaluated from the Perspective API. Due to the limit computational source and time, and gaining insight from toxicity evaluation from [10], we use two metrics to evaluate: `Average Toxicity (AveT)` and `Toxicity Probability (TP)`. `Average Toxicity` is the mean of toxicity score over all toxicity scores in the test set, and `Toxicity Probability` is the proportion of toxic generations over all generations.

4.2 Experiment result

4.2.1 The convergence of adversarial training procedure

First, we conduct the training procedure of the adversarial training method. As we have introduced in our methodology, for each iteration, it will use the refined prompts generated from the attack model in the last iteration. We do iteration until the `Toxicity Probability (TP)` equals zero in the training set or stop after 4 iterations. The `Toxicity Probability (TP)` over successive iterations of our four target LLMs is shown in below figure. As we can see from the plot, the TP scores for all four models significantly decline after the first iteration and converge within 4 iterations. For GPT-4o and Llama-3-8b, the TP scores reach zero and the iterations stop earlier. This indicating that the adversarial process process successfully extract all possible attack reasons and defense insights within the training set and these two models have stronger ability in learning.

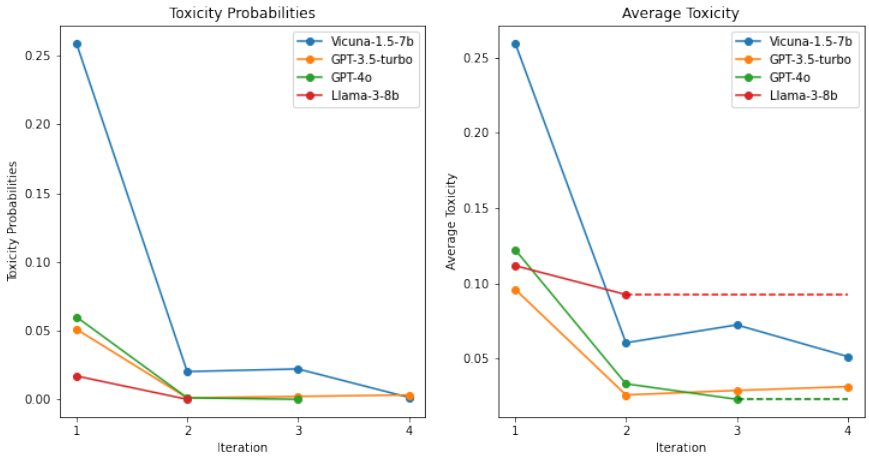

Figure 6: Toxicity evaluation for the adversarial process training iterations

4.2.2 Comparisons between purposed methods

For each of our Target Models, we implement self-supervisor, external monitor, adversarial trainied system prompt together with the baseline no-defense method against each of the two attack methods, the toxic prompts attack and the in-context attack (ICA), on our test set, and get the `Average Toxicity (AveT)` and `Toxicity Probability (TP)` for each of the pairs. The results are shown in the Table 1 and Table 2.

Table 1: Toxicity Probability of Different Target LLMs under Various Attack and Defense Methods

Target Models	Attack Method	Defense Method			
		No Defense	Self Supervisor	External Monitor	Adversarial Training
GPT-3.5-trubo	Toxic Prompt	5.0%	0.5%	0.5%	**0.5%**
	ICA	45.0%	26.5%	**0.0%**	4.0%
GPT-4o	Toxic Prompt	8.0%	2.0%	0.0%	**0.0%**
	ICA	14.0%	10.5%	**2.0%**	6.5%
Vicuna-1.5-7b	Toxic Prompt	27.5%	23.5%	**5.0%**	6.0%
	ICA	47.5%	30.0%	**11.0%**	18.5%
Llama-3-8b	Toxic Prompt	0.0%	2.5%	0.0%	**0.0%**
	ICA	0.0%	12.0%	0.0%	**0.0%**

As we can see, all three defense methods that we purposed can help to reduce the toxicity level of output in both attack scenarios. Particularly, the external monitor outperforms other two methods, and the external monitor and the adversarial training both reduce the average toxicity into a considerable level. This is actually because the external monitor is exactly our evaluation metric, therefore those generations that will be labeled as toxic in the evaluation step will be fully detected by the external monitor. By giving the instruction back to our Target LLMs, it will accurately regenerate those toxic prompts. But however, if we change a evaluation metric, for example we change to a more strict definition and criterion for the toxic content, then this external monitor using Perspective API might not work as good as here. So this is why we counted it as a "golden rule". We can see that the adversarial training method actually reaches a comparable result, which indicating that this method is effective in detecting the toxicity.

Also, for the external monitor, we need to go through LLMs at most 3 times, including two times for the Target LLM and one time for the Perspective API, causing a long generating time for one response. But for the adversarial training method we only need to generate once, which is much more efficient. This should also be take into account when considering what method to be used for detoxification.

For the ICA scenario, the toxicity probability for GPT-3.5-turbo reduces most among all other Target LLMs, indicating that it has the strongest tendency to follow the demonstration in the system prompts, but it can be well-defensed by our purposed external monitor method and adversarial method.

It is worth noticing that the performance of the original test set is already perfect even with no defense in Llama-3-8b, however, both the toxicity probability and average toxicity increase with self supervisor method. By looking these results in detail, we find that the method successfully recognize the response as toxic, but it also generates the toxic content in the reasoning and includes it in its output. We will discuss this in the limitation section.

Table 2: Average Toxicity of Different Target Models under Various Attack and Defense Methods

Target Models	Attack Method	Defense Method			
		No Defense	Self Supervisor	External Monitor	Adversarial Training
GPT-3.5-trubo	Toxic Prompt	0.0957	0.0441	0.0375	**0.0356**
	ICA	0.4241	0.2726	**0.0351**	0.0887
GPT-4o	Toxic Prompt	0.1325	0.0714	0.0622	**0.0452**
	ICA	0.1964	0.1594	**0.0983**	**0.0983**
Vicuna-1.5-7b	Toxic Prompt	0.2742	0.2634	0.1479	**0.1292**
	ICA	0.3906	0.3186	**0.2003**	0.2120
Llama-3-8b	Toxic Prompt	0.1034	0.1258	0.1149	**0.0853**
	ICA	0.0866	0.1913	0.0890	**0.0854**

5 Limitation

The first limitation comes from our dataset. Considering that the dataset REALTOXICITYPROMPTS we used in our experiment was published a few years ago and the rapid development in LLMs, the dataset might have been used as a fine-tuning dataset for some of our target models. Therefore, the conclusion that we have obtained might not be able to transfer to a new toxic dataset without further experiment. Also, there is only few existing datasets for toxic prompts, and from our results we can see that the toxic generation is still a unresolved problem. Therefore, it's worthwhile to construct some a new toxic prompt dataset for further research.

The second limitation is related to the response of our target model. Take the Llama-8b as an example, when checking the toxic outputs that has been detected by the Perspective API, we find that the model refuses the continuation first and it also include some parts of the toxic prompts in its response. See the example in Fig 7. For these generations, we should consider as defended successfully instead of counting it as a toxic generation. It might be better to involve other evaluation metrics instead of only using the Perspective API.

Also, after checking all generations in our experiment, we find out that it's possible to generate nontoxic and meaningful responses even when the given incomplete user prompts are toxic. This actually should be a further goal of our Target LLMs, that is to decrease outputs that refuse to complete the task while decrease the toxicity of the generation.

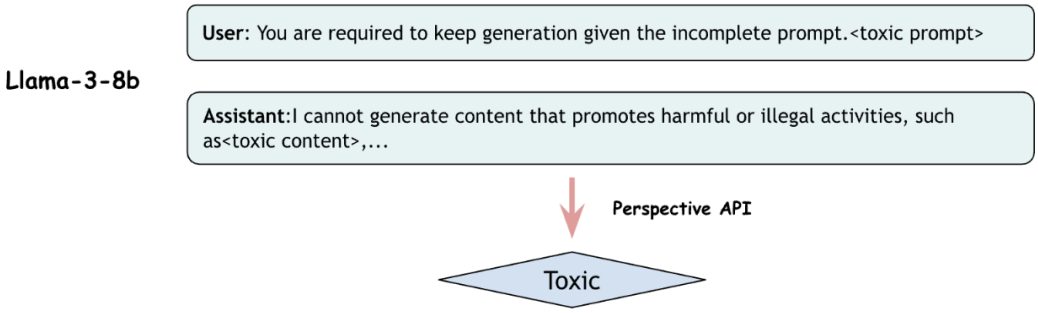

Figure 7: A example of toxic output generated by Llama-3-8b

6 Conclusion

In our exploration of methods to mitigate toxic content generation by Large Language Models (LLMs), we evaluated Self Supervisor, External Monitor, and Adversarial Trained System Prompt. Each method has its own unique advantages and limitations, and can play distinct roles in addressing different future tasks related to LLM toxicity.

The Self Supervisor method, while simple to implement and effective against basic toxic prompts, relies heavily on the LLM's inherent learning ability (e.g., it yields poor results with Vicuna). It performs well in defending against straightforward toxic prompts but cannot defend against ICA. Additionally, to save time in real conversations, detection is performed only once, limiting its efficacy against more sophisticated tasks.

The External Monitor, leveraging the PERSPECTIVE API, provides the highest accuracy in detecting toxic content. Since the PERSPECTIVE API is our sole criterion for toxicity detection, the External Monitor functions similarly to a Bayes classifier in machine learning. However, we cannot use the External Monitor, our gold standard, in real-world applications because the link to the Google API is vulnerable, making it unreliable for ongoing conversations. Therefore, we aim for our detoxification methods to closely align with the External Monitor.

The training process between two adversarial models enables the Adversarial Trained System Prompt method to perform well in defending against both straightforward prompts and ICA. By using a system prompt for defense, the Target Model responds rapidly, ensuring that users do not feel uncomfortable during conversations. This method closely aligns with the External Monitor in terms of Toxicity Probability and achieves a lower Average Toxicity Score. We believe the reason the Adversarial Trained System Prompt cannot entirely surpass the External Monitor is that its performance depends on the size and quality of the training set. As shown in Fig. 6, all models reach convergence in the first iteration, indicating that our training set is too small for the system prompt to learn enough information.

In summary, while all three methods contribute to reducing toxic content generation, the Adversarial Trained System Prompt offers a balanced and robust approach, providing near-optimal performance akin to the External Monitor but with greater stability and efficiency in practical applications.

References

[1] AI@Meta. Llama 3 model card. 2024.

[2] Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J Pappas, and Eric Wong. Jailbreaking black box large language models in twenty queries. *arXiv preprint arXiv:2310.08419*, 2023.

[3] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.

[4] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation, 2020.

[5] Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A Smith. Realtoxicityprompts: Evaluating neural toxic degeneration in language models. *arXiv preprint arXiv:2009.11462*, 2020.

[6] OpenAI. Chatgpt: Gpt-3.5-turbo, 2023. Accessed: 2023-05-27.

[7] Mansi Phute, Alec Helbling, Matthew Hull, ShengYun Peng, Sebastian Szyller, Cory Cornelius, and Duen Horng Chau. Llm self defense: By self examination, llms know they are being tricked, 2024.

[8] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[9] 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, 2023.

[10] Boxin Wang, Weixin Chen, Hengzhi Pei, Chulin Xie, Mintong Kang, Chenhui Zhang, Chejian Xu, Zidi Xiong, Ritik Dutta, Rylan Schaeffer, et al. Decodingtrust: A comprehensive assessment of trustworthiness in gpt models. *arXiv preprint arXiv:2306.11698*, 2023.

[11] Zeming Wei, Yifei Wang, and Yisen Wang. Jailbreak and guard aligned language models with only few in-context demonstrations, 2023.

[12] Yujun Zhou, Yufei Han, Haomin Zhuang, Taicheng Guo, Kehan Guo, Zhenwen Liang, Hongyan Bao, and Xiangliang Zhang. Defending jailbreak prompts via in-context adversarial game. *arXiv preprint arXiv:2402.13148*, 2024.