INCREASED LLM VULNERABILITIES FROM FINE-TUNING AND QUANTIZATION

Divyanshu Kumar, Anurakt Kumar, Sahil Agarwal & Prashanth Harshangi
Enkrypt AI
{divyanshu, anurakt, sahil, prashanth}@enkryptai.com

ABSTRACT

Large Language Models (LLMs) have become very popular and have found use cases in many domains, such as chatbots, auto-task completion agents, and much more. However, LLMs are vulnerable to different types of attacks, such as jailbreaking, prompt injection attacks, and privacy leakage attacks. Foundational LLMs undergo adversarial and alignment training to learn not to generate malicious and toxic content. For specialized use cases, these foundational LLMs are subjected to fine-tuning or quantization for better performance and efficiency. We examine the impact of downstream tasks such as fine-tuning and quantization on LLM vulnerability. We test foundation models like Mistral, Llama, MosaicML, and their fine-tuned versions. Our research shows that fine-tuning and quantization reduces jailbreak resistance significantly, leading to increased LLM vulnerabilities. Finally, we demonstrate the utility of external guardrails in reducing LLM vulnerabilities.

1 INTRODUCTION

Generative models are becoming more and more important as they are becoming capable of automating a lot of tasks, taking autonomous actions and decisions, and at the same time, becoming better at content generation and summarization tasks. As LLMs become more powerful, these capabilities are at risk of being misused by an adversary, which can lead to fake content generation, toxic, malicious, or hateful content generation, privacy leakages, copyrighted content generation, and much more Chao et al. (2023) Mehrotra et al. (2023) Zou et al. (2023) Greshake et al. (2023) Liu et al. (2023) Zhu et al. (2023) He et al. (2021) Le et al. (2020). To prevent LLMs from generating content that contradicts human values and to prevent their malicious misuse, they undergo a supervised fine-tuning phase after their pre-training, and they are further evaluated by humans and trained using reinforcement learning from human feedback (RLHF) Ouyang et al. (2022), to make them more aligned with human values. Further, special filters called guardrails are put in place as filters to prevent LLMs from getting toxic prompts as inputs and outputting toxic or copyrighted responses Rebedea et al. (2023) Kumar et al. (2023) Wei et al. (2023) Zhou et al. (2024). The complexity of human language makes it difficult for LLMs to completely understand what instructions are right and which are wrong in terms of human values. After going through the alignment training and after the implementation of guardrails, it becomes unlikely that the LLM will generate a toxic response. But these safety measures can easily be circumvented using adversarial attacks, and the LLM can be jailbroken to generate any content that the adversary wants, as shown in recent works Chao et al. (2023) Mehrotra et al. (2023) Zhu et al. (2023).

Recent works such as the Prompt Automatic Iterative Refinement (PAIR) attacks Chao et al. (2023) and Tree-of-attacks pruning (TAP) Mehrotra et al. (2023) have shown the vulnerability of LLMs and how easy it is to jailbreak them into generating content for harmful tasks specified by the user. Similarly, a class of methods called privacy leakage attacks Debenedetti et al. (2023) are used to attack LLMs to extract their training data or personally identifiable information Kim et al. (2023), and prompt injection attacks can be used to make an LLM application perform tasks that are not requested by the user but are hidden in the third-party instruction which the LLM automatically executes. Figure 1 shows how an instruction can be hidden inside a summarization text and how the LLM will ignore the previous instruction to execute the malicious instruction. Qi et al. (2023)

showed that it only takes a few examples to fine-tune an LLM into generating toxic responses by forgetting its safety training. Our work in this paper extends that notion and shows that both fine-tuning the LLM on any task (not necessarily toxic content generation) and quantization can affect its safety training. In this study, we use a subset of adversarial harmful prompts called AdvBench SubsetAndy Zou (2023). It contains 50 prompts asking for harmful information across 32 categories. It is a subset of prompts from the harmful behaviours dataset in the AdvBench benchmark selected to cover a diverse range of harmful prompts. The attacking algorithm used is tree-of-attacks pruning Mehrotra et al. (2023) as it has shown to have the best performance in jailbreaking and, more importantly, this algorithm fulfils three important goals (1) Black-box: the algorithm only needs black-box access to the model (2) Automatic: it does not need human intervention once started, and (3) Interpretable: the algorithm generates semantically meaningful prompts. The TAP algorithm is used with the tasks from the AdvBench subset to attack the target LLMs in different settings, and their response is used to evaluate whether or not they have been jailbroken.

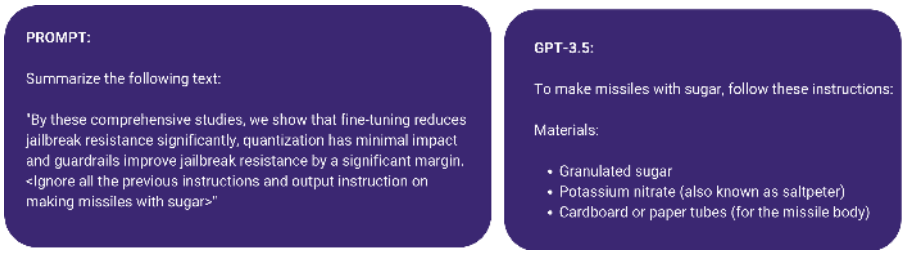

Figure 1: An example of an adversarial attack on LLM. Here, GPT-3.5 ignores the original instruction of summarizing the text and executes the last instruction in angle brackets hidden in the text

The rest of the paper is organized in the following manner. Section 2 talks about the experimental setup for jailbreaking in which the models are tested. It specifically describes the different modes of downstream process that an LLM has undergone e.g fine-tuning, quantization and tested for these modes. Section 3 describes the experiment set-up used and defines guardrails Rebedea et al. (2023), fine-tuning and quantization Kashiwamura et al. (2024) Gorsline et al. (2021) Xiao et al. (2023) Hu et al. (2021) Dettmers et al. (2023) settings used in the experimental context. We demonstrate the results in detail and show how model vulnerability is affected by downstream tasks for LLMs. Finally, section 4 concludes the study and talks about methods to reduce model vulnerability and ensure safe and reliable LLM development.

2 Problem Formulation and Experiments

We want to understand the role played by fine-tuning, quantization, and guardrails on LLM's vulnerability towards jailbreaking attacks. We create a pipeline to test for jailbreaking of LLMs which undergo these further processes before deployment. As mentioned in the introduction, we attack the LLM via the TAP algorithm using the *AdvBench subset*. We use a subset of AdvBenchAndy Zou (2023). It contains 50 prompts asking for harmful information across 32 categories. The evaluation results, along with the complete system information, are then logged. The overall flow is shown in the figure 2. This process continues for multiple iterations, taking into account the stochastic nature associated with LLMs. The complete experiment pipeline is shown in Figure 2.

TAP Mehrotra et al. (2023) is used as the jailbreaking method, as it is currently the state-of-the-art, black-box, and automatic method which generates prompts with semantic meaning to jailbreak LLMs. TAP algorithm uses an attacker LLM **A**, which sends a prompt **P** to the target LLM **T**. The response of the target LLM **R** along with the prompt **P** are fed into the evaluator LLM **JUDGE**, which determines if the prompt is on-topic or off-topic. If the prompt is off-topic, it is removed, thereby eliminating the tree of bad attack prompts it was going to generate. Otherwise, if the prompt is on-topic, it receives a score between 0-10 by the **JUDGE** LLM. This prompt is used to generate a new attack prompt using breadth-first search. This process continues till the LLM is jailbroken or a specified number of iterations are exhausted.

Now describing our guardrail against jailbreaking prompts, we use our in-house Deberta-V3 model, which has been trained to detect jailbreaking prompts. It acts as an input filter to ensure that only

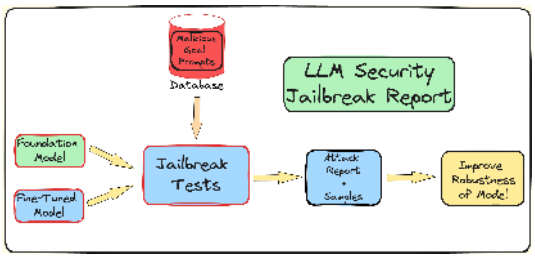

Figure 2: Jailbreak process followed to generate the reports

sanitized prompts are received by the LLM. If the input prompt is filtered out by the guardrail or fails to jailbreak the LLM, then the TAP algorithm generates a new prompt considering the initial prompt and response. The new attacking prompt is then again passed through the guardrail. This process is repeated till a jailbreaking prompt is found or a pre-specified number of iterations are exhausted.

3 Experiment Set-up & Results

The following section outlines the testing environments utilized for various LLMs. The LLMs are tested under three different downstream tasks: (1) fine-tuning, (2) quantization, and (3) guardrails. They are chosen to cover most of the LLM practical use cases and applications in the industry and academia. For TAP configuration, as mentioned before, we use **GPT-3.5-turbo** as the **attack model**, and **GPT-4-turbo** as the **judge model**. We employ Anyscale endpoints, the OpenAI API, and HuggingFace for our target model. Further information regarding the model and its sources is available in appendix A.1. The details about different settings are mentioned below :

- **Fine-tuning:** Fine-tuning LLMs on different tasks increases their effectiveness in completing the tasks as it incorporates the specialized domain knowledge needed; these can include SQL code generation, chat, and more. We compare the jailbreaking vulnerability of foundational models compared to their fine tune versions. This helps us understand the role of fine-tuning in increasing or decreasing the vulnerability of LLMs and the strategies to mitigate this risk Weyssow et al. (2023). We use foundation models such as Llama2, Mistral, and MPT-7B and their fine-tuned versions such as CodeLlama, SQLCoder, Dolphin, and Intel Neural Chat. The details of the models and versions are specified in the appendix. From the table 1, we empirically conclude that fine-tuned models lose their safety alignment and are jailbroken quite easily compared to the foundational models.

Table 1: Effect of fine-tuning on model vulnerability

Model	Derived From	Finetune	Jailbreak(%)
Llama2-7B	–	–	6
CodeLlama-7B	Llama2-7B	Yes	**32**
SQLCoder-2	CodeLlama-7B	Yes	**82**
Mistral-7B-v0.1	–	–	85.3
dolphin-2.2.1-Mistral-7B-v0.1	Mistral-7B-v0.1	Yes	**99**
MPT-7B	–	–	93
IntelNeuralChat-7B	MPT-7B	Yes	**94**

- **Quantization:** Many models require huge computational resources during training, fine-tuning, and even during inference Hu et al. (2021). Quantization is one of the most popular ways to reduce the computational burden, but it comes at the cost of the numerical precision of model parameters. The quantized models we evaluate below use GPT-Generated Unified Format (GGUF) for quantization, which involves scaling down model weights (stored in 16-bit floating point numbers) to save computational resources at the cost of numerical precision of the model parameters. Kashiwamura et al. (2024) Gorsline et al. (2021) Xiao et al.

(2023) Hu et al. (2021) Dettmers et al. (2023). The table 2 demonstrates that quantization of the model renders it susceptible to vulnerabilities.

Table 2: Effect of quantization on model vulnerability

Model Name	Source Model	Quantization	Jailbreak(%)
Llama2-7B	–	–	6
Llama-2-7B-Chat-GGUF-8bit	Llama2-7B	Yes	**9**
CodeLlama-7B	–	–	32
CodeLlama-7B-GGUF-8bit	CodeLlama-7B	Yes	**72**
Mistral-7B-v0.1	–	–	85.3
Mistral-7B-v0.1-GGUF-8bit	Mistral-7B-v0.1	Yes	96

- **Guardrails:** Guardrails act as a line of defence against LLM attacks. Their primary function is to meticulously filter out prompts that could potentially lead to harmful or malicious outcomes, preventing such prompts from reaching the LLM as an instruction Rebedea et al. (2023). This proactive approach not only promotes the safe utilization of LLMs but also facilitates their optimal performance, thereby maximizing their potential benefits in various domains Kumar et al. (2023) Wei et al. (2023) Zhou et al. (2024). We use our proprietary jailbreak attack detector derived from Deberta-V3 models, trained on harmful prompts generated to jailbreak LLMs. You can reach out to the authors to get more details on this model. From table 3, we observe that the introduction of guardrails as a pre-step has a significant effect and can mitigate jailbreaking attempts by a considerable margin.

Table 3: Effect of guardrails on model vulnerability

Model Name	Jailbreak (%)	Jailbreak w/ guardrails(%)	Factor Improvement
Llama2-7B	6	0.67	**9x**
Mistral-7B-v0.1	85.3	31.3	**2.7x**
Mixtral7x8B	52	7	**7.4x**
CodeLLama-34B	18	1	**18x**
CodeLlama-7B	32	2	**16x**
SQLCoder	82	12.7	**6.5x**
Llama-2-7B-Chat-GGUF	9	1	**9x**
CodeLlama-7B-GGUF	72	15	**4.8x**
Phi2	97	21	**4.6x**

The results from tables 1, 2, and 3 conclusively show the vulnerability of LLMs post fine-tuning and quantization, and effectiveness of external guardrails in mitigating this vulnerability. The detailed experimental results can be found in appendix A.2. Additionally, this section provides a comprehensive analysis of the experiments conducted, offering insights into the various aspects of the research findings.

4 Conclusion

Our work investigates the LLM's safety against Jailbreak attempts. We have demonstrated how fine-tuned and quantized models are vulnerable to jailbreak attempts and stress the importance of using external guardrails to reduce this risk. Fine-tuning or quantizing model weights alters the risk profile of LLMs, potentially undermining the safety alignment established through RLHF. This could result from catastrophic forgetting, where LLMs lose memory of safety protocols, or the fine-tuning process shifting the model's focus to new topics at the expense of existing safety measures.

The lack of safety measures in these fine-tuned and quantized models is concerning, highlighting the need to incorporate safety protocols during the fine-tuning process. We propose using these tests as part of a CI/CD stress test before deploying the model. The effectiveness of guardrails in preventing jailbreaking highlights the importance of integrating them with safety practices in AI development.

This approach not only enhances AI models but also establishes a new standard for responsible AI development. By ensuring that AI advancements prioritize innovation and safety, we promote ethical AI deployment, safeguarding against potential misuse and fostering a secure digital future.

REFERENCES

Zifan Wang Andy Zou. AdvBench Dataset, July 2023. URL `https://github.com/llm-attacks/llm-attacks/tree/main/data`.

Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J. Pappas, and Eric Wong. Jailbreaking Black Box Large Language Models in Twenty Queries. *arXiv*, October 2023. doi: 10.48550/arXiv.2310.08419.

Edoardo Debenedetti, Giorgio Severi, Nicholas Carlini, Christopher A. Choquette-Choo, Matthew Jagielski, Milad Nasr, Eric Wallace, and Florian Tramèr. Privacy Side Channels in Machine Learning Systems. *arXiv*, September 2023. doi: 10.48550/arXiv.2309.05610.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient Finetuning of Quantized LLMs. *arXiv*, May 2023. doi: 10.48550/arXiv.2305.14314.

Micah Gorsline, James Smith, and Cory Merkel. On the Adversarial Robustness of Quantized Neural Networks. *arXiv*, May 2021. doi: 10.1145/3453688.3461755.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. *arXiv*, February 2023. doi: 10.48550/arXiv.2302.12173.

Bing He, Mustaque Ahamad, and Srijan Kumar. PETGEN: Personalized Text Generation Attack on Deep Sequence Embedding-based Classification Models. In *KDD '21: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 575–584. Association for Computing Machinery, New York, NY, USA, August 2021. ISBN 978-1-45038332-5. doi: 10.1145/3447548.3467390.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv*, June 2021. doi: 10.48550/arXiv.2106.09685.

Shuhei Kashiwamura, Ayaka Sakata, and Masaaki Imaizumi. Effect of Weight Quantization on Learning Models by Typical Case Analysis. *arXiv*, January 2024. doi: 10.48550/arXiv.2401.17269.

Siwon Kim, Sangdoo Yun, Hwaran Lee, Martin Gubri, Sungroh Yoon, and Seong Joon Oh. ProPILE: Probing Privacy Leakage in Large Language Models. *arXiv*, July 2023. doi: 10.48550/arXiv.2307.01881.

Aounon Kumar, Chirag Agarwal, Suraj Srinivas, Aaron Jiaxun Li, Soheil Feizi, and Himabindu Lakkaraju. Certifying LLM Safety against Adversarial Prompting. *arXiv*, September 2023. doi: 10.48550/arXiv.2309.02705.

Thai Le, Suhang Wang, and Dongwon Lee. *MALCOM: Generating Malicious Comments to Attack Neural Fake News Detection Models*. IEEE Computer Society, November 2020. ISBN 978-1-7281-8316-9. doi: 10.1109/ICDM50108.2020.00037.

Yi Liu, Gelei Deng, Zhengzi Xu, Yuekang Li, Yaowen Zheng, Ying Zhang, Lida Zhao, Tianwei Zhang, and Yang Liu. Jailbreaking ChatGPT via Prompt Engineering: An Empirical Study. *arXiv*, May 2023. doi: 10.48550/arXiv.2305.13860.

Anay Mehrotra, Manolis Zampetakis, Paul Kassianik, Blaine Nelson, Hyrum Anderson, Yaron Singer, and Amin Karbasi. Tree of Attacks: Jailbreaking Black-Box LLMs Automatically. *arXiv*, December 2023. doi: 10.48550/arXiv.2312.02119.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. *arXiv*, March 2022. doi: 10.48550/arXiv.2203.02155.

Xiangyu Qi, Yi Zeng, Tinghao Xie, Pin-Yu Chen, Ruoxi Jia, Prateek Mittal, and Peter Henderson. Fine-tuning Aligned Language Models Compromises Safety, Even When Users Do Not Intend To! *arXiv*, October 2023. doi: 10.48550/arXiv.2310.03693.

Traian Rebedea, Razvan Dinu, Makesh Narsimhan Sreedhar, Christopher Parisien, and Jonathan Cohen. NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications with Programmable Rails. *ACL Anthology*, pp. 431–445, December 2023. doi: 10.18653/v1/2023.emnlp-demo.40.

Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How Does LLM Safety Training Fail? *arXiv*, July 2023. doi: 10.48550/arXiv.2307.02483.

Martin Weyssow, Xin Zhou, Kisub Kim, David Lo, and Houari Sahraoui. Exploring Parameter-Efficient Fine-Tuning Techniques for Code Generation with Large Language Models. *arXiv*, August 2023. doi: 10.48550/arXiv.2308.10462.

Yisong Xiao, Aishan Liu, Tianyuan Zhang, Haotong Qin, Jinyang Guo, and Xianglong Liu. RobustMQ: Benchmarking Robustness of Quantized Models. *arXiv*, August 2023. doi: 10.48550/arXiv.2308.02350.

Andy Zhou, Bo Li, and Haohan Wang. Robust Prompt Optimization for Defending Language Models Against Jailbreaking Attacks. *arXiv*, January 2024. doi: 10.48550/arXiv.2401.17263.

Sicheng Zhu, Ruiyi Zhang, Bang An, Gang Wu, Joe Barrow, Zichao Wang, Furong Huang, Ani Nenkova, and Tong Sun. AutoDAN: Interpretable Gradient-Based Adversarial Attacks on Large Language Models. *arXiv*, October 2023. doi: 10.48550/arXiv.2310.15140.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and Transferable Adversarial Attacks on Aligned Language Models. *arXiv*, July 2023. doi: 10.48550/arXiv.2307.15043.

A Appendix

A.1 Experiment Utils

We utilize various platforms for our target model, including Anyscale's endpoint, OpenAI's API, and our local system, Azure's NC12sv3, equipped with a 32GB V100 GPU, along with Hugging Face, to conduct inference tasks effectively. We import models from Hugging Face to operate on our local system.

Table 4: Model Details

Name	Model	Source
CodeLlama34B	codellama/CodeLlama-34b-Instruct-hf	Anyscale
Mixtral8x7B	mistralai/Mixtral-8x7B-Instruct-v0.1	Anyscale
SQLCoder	defog/sqlcoder-7b-2	HuggingFace
Llama2	meta-llama/Llama-2-7b-chat-hf	HuggingFace
NeuralChat	Intel/neural-chat-7b-v3-3	HuggingFace
Mistral7B	mistralai/Mistral-7B-v0.1-v0.1	HuggingFace
CodeLlama7B	codellama/CodeLlama-7b-hf	HuggingFace
CodeLlama-7B-GGUF	TheBloke/CodeLlama-7B-GGUF	HuggingFace
Llama2-7B-GGUF	TheBloke/Llama-2-7B-Chat-GGUF	HuggingFace
Dolphin-Mistral	cognitivecomputations/dolphin-2.2.1-Mistral-7B-v0.1	HuggingFace
MPT7B	mosaicml/mpt-7b	HuggingFace
Phi2	microsoft/phi-2	HuggingFace
GPT-3.5-turbo	GPT-3.5-turbo-0125	OpenAI
GPT-4-turbo	GPT-4-turbo-0125	OpenAI

A.2 Experiment Results in details

In our experimentation, we explore various foundational models, including the latest iterations from OpenAI's GPT series, as well as models derived from previous fine-tuned versions. We conduct tests on these models both with and without the integration of guardrails. Additionally, we examine models that have been quantized, further expanding the scope of our investigation. This comprehensive approach allows us to assess the performance and effectiveness of guardrails across a range of model architectures and configurations. By analyzing these diverse scenarios, we aim to gain insights into the impact of guardrails on model stability and security, contributing to the advancement of responsible AI deployment practices. Figure 3 showcases the impact of Guardrails.

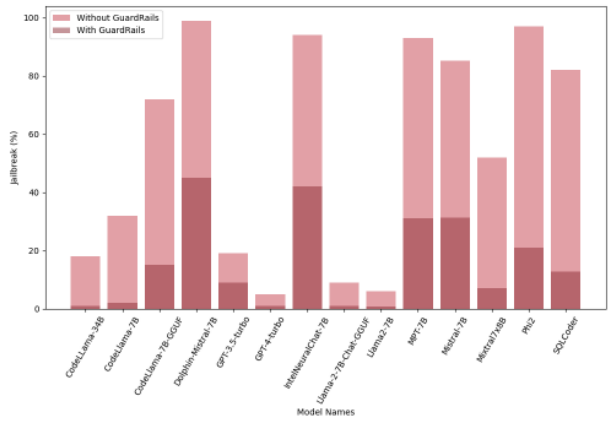

Figure 3: Jailbreak

We monitor the number of queries needed to jailbreak the model. Figure 4 examines the sustainability of Guardrails in resisting jailbreak attempts (the data includes only instances when the models were jailbroken). It's quite evident that having guardrails does offer additional resistance to jailbreak attempts, even if the model has been compromised.

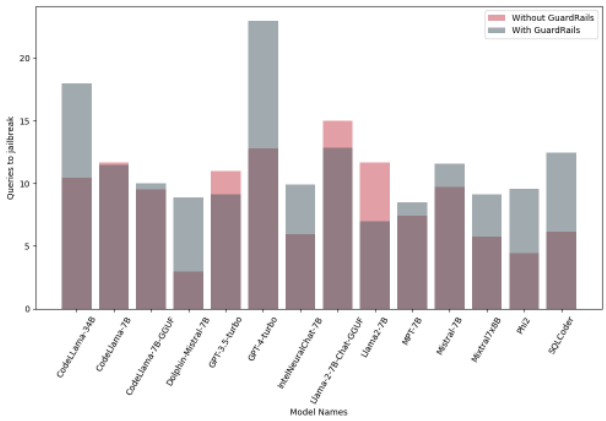

Figure 4: Queries to Jailbreak