A REALISTIC THREAT MODEL FOR LARGE LANGUAGE MODEL JAILBREAKS

Valentyn Boreiko[1-2]* **Alexander Panfilov**[2-4]* **Vaclav Voracek**[1-2]
Matthias Hein[1-2]† **Jonas Geiping**[2-4]†

[1]University of Tübingen [2]Tübingen AI Center [3]Max Planck Institute for Intelligent Systems
[4]ELLIS Institute Tübingen

* Equal contribution; † Joint senior authors

ABSTRACT

A plethora of jailbreaking attacks have been proposed to obtain harmful responses from safety-tuned LLMs. In their original settings, these methods all largely succeed in coercing the target output. Yet, the resulting jailbreaks vary substantially in readability and computational effort. In this work, we propose a unified threat model for the principled comparison of these methods. Our threat model combines practical considerations with constraints along two dimensions: perplexity, which measures how far a jailbreak deviates from natural text, and computational budget in total FLOPs. For the former we built an N-gram model on 1T tokens, which, in contrast to model-based perplexity, allows for a neutral, LLM-agnostic, and intrinsically interpretable evaluation. Moreover, we adapt existing popular attacks to this threat model. Our threat model enables a comprehensive and precise comparison of various jailbreaking techniques within a single realistic framework. We further find that, under this threat model, even the most effective attacks, when thoroughly adapted, struggle to achieve success rates above 40% against safety-tuned models. This indicates that in a realistic chat scenario, current LLMs are less prone to attacks as it was believed before. The code is available at https://github.com/valentyn1boreiko/llm-threat-model.

1 INTRODUCTION

As LLMs can be used to facilitate fraud, spread fake news, conduct hacking attacks, etc. (Willison, 2023; Greshake et al., 2023; Carlini et al., 2021; Geiping et al., 2024), model providers often safety-tune LLMs to minimize the risks of potential misuse, mitigate harm, and avoid complying with malicious queries (Christiano et al., 2017; Ouyang et al., 2022). However, while this alignment ensures average-case safety, it does not currently extend to adversarial scenarios (Carlini et al., 2024; Qi et al., 2024).

We refer to a *jailbreak* as an adversarially designed text input that circumvents safety tuning and elicits harmful behavior. While adversarial attacks in computer vision commonly adopt l_p-ball threat model, to be imperceptible to humans but still fool the model (Madry et al., 2018), jailbreaks in language come in all shapes and sizes, rarely sharing the same constraints. These attacks range from completely gibberish suffixes (Zou et al., 2023) to human-like social engineering techniques applied to an LLM (Zeng et al., 2024). While all these methods are designed to succeed in terms of attack success rate (ASR), they commonly also report their efficacy based on an arbitrary combination of metrics such as stealthiness (aka readability, fluency, human-likeliness) (Liu et al., 2024; Yang et al., 2024; Mehrotra et al., 2023; Sadasivan et al., 2024), query efficiency (Chao et al., 2023; 2024), runtime (Geisler et al., 2024; Sadasivan et al., 2024), monetary expense (Sadasivan et al., 2024), etc. This heterogeneity prevents a clear understanding of the jailbreaking attack landscape and complicates the fair comparison of different methods.

Correspondance to valentyn.boreiko(at)uni-tuebingen.de, alexander.panfilov(at)tue.ellis.eu

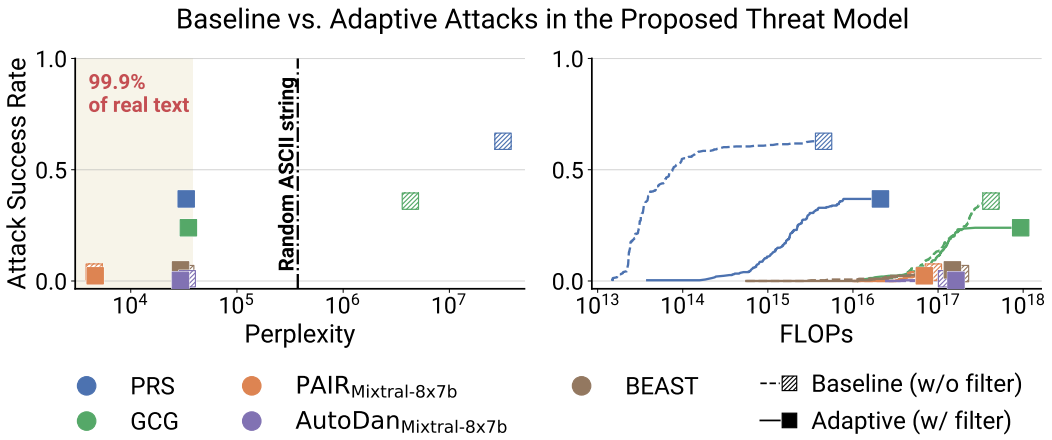

Figure 1: Evaluating Jailbreak Attacks Against Llama2-7b. **Left:** The most effective attacks tend to have higher perplexity under our N-gram model, significantly exceeding that of real text. As such, these attacks are often discarded as impractical. **Right:** However, we find that with well-constructed adaptive attacks, these high-perplexity attacks still outperform attacks designed as low-perplexity attacks, such as PAIR. Even under strong adaptive attacks though, the N-gram model perplexity constraint significantly raises the compute costs for successful attacks and reduces attack success at minimal cost.

2 Motivation and Contributions

Readability of input is implicitly enforced in many attacks (Liu et al., 2024; Yang et al., 2024; Mehrotra et al., 2023; Sadasivan et al., 2024) and is often measured by perplexity (PPL) using deep language models. Moreover, PPL-based filters are established input-level defenses (Alon & Kamfonas, 2023; Jain et al., 2023), which effectively makes PPL a de facto constraint that attacks must satisfy. However, relying on model-based PPL leads to a setup that is: i) incomparable due to dependence on different target LLMs, ii) non-interpretable, iii) based on neural networks and thus susceptible to adversarial examples, and iv) costly to evaluate.

To address this, we propose to use an N-gram language model (N-gram LM) PPL as a neutral measure of PPL. This approach is: i) **LLM-agnostic**: The N-gram LM PPL is a measure of perplexity unrelated to LLM-based measures, allowing for a comparison between attacks generated with and against different models ii) **interpretable**: Each N-gram's contribution to the PPL can be examined and adjusted, it is cheap to add new datasets or do N-gram models for different kind of text e.g. code; iii) **simple**: The N-gram model is a simple non-differentiable, parameter-free model of the occurrence of text which makes it less susceptible to attacks ; iv) **fast-to-evaluate**: Accessing each N-gram when computing N-gram LM PPL has a time complexity of $\mathcal{O}(1)$.

Our contributions are as follows:

- In Section 4, we introduce a threat model for comparing jailbreaking attacks that incorporates text fluency measured by N-gram LM PPL and compute cost in total floating point operations.
- We construct a light-weight bigram model from the dolma dataset based on about 1T tokens.
- In Section 5, we create strong adaptive attacks for this threat model for a number of popular jailbreaking techniques.
- In Appendix D, we investigate jailbreaks that bypass the proposed filter with adaptive attacks by looking at the attribution of the jailbreaks to a particular dataset.

3 Related Work

Red Teaming LLMs. LLM providers strive to minimize harmful interactions with their models. To do, manual redteaming is incorporated into the post-training stage, where human testers probe the bounds of the model's safety tuning Ganguli et al. (2022), and the model is updated to prefer safe outputs on these queries. This, however, has been shown to miss many often unnatural, but very successful automated attacks (Zou et al., 2023; Andriushchenko et al., 2024).

Threat Model in Computer Vision. As earlier approaches focused more on mining harmful prompts, the threat model used in these works is often unclear or outdated. In computer vision, however, the community quickly converged on a single threat model, namely attacks to an l_p ball of a chosen radius r (Croce et al., 2021). This constraint serves two purposes in vision: i) it is simple and thus allows to create efficient algorithms that generate solution that respect it; ii) by decreasing r and increasing p one can create stealthy attacks that are imperceptible to humans. This lead to significant progress on this threat model on vision, but stymied work on more realistic threat models.

LLM Jailbreaking Benchmarks. Proposed jailbreaking benchmarks (Xie et al., 2024; Mazeika et al., 2024; Chao et al., 2024) aim to standardize the evaluation of attacks and defenses but fail to account for the significant differences in existing jailbreaking methods or to agree on a consistent evaluation method. Among existing benchmarks, *HarmBench* (Mazeika et al., 2024) stands out as the most comprehensive, both in terms of the number of models and attacks investigated, and it addresses many previous evaluation flaws. In this work, we construct a realistic threat model, introduced in Section 4, that allows us to adapt popular jailbreaking attacks, introduced in Section 4 which we combine with HarmBench to provide a stronger evaluation for realistic automated attacks against LLMs.

4 Proposed Threat Model

4.1 Defining a Jailbreak

Let $\mathcal{T}$ be the set of all tokens. Define the set of all sequences from $\mathcal{T}$ as $\mathcal{T}^* := \bigcup_{n=1}^{\infty} \mathcal{T}^n$, where $\mathcal{T}^n$ represents the set of all sequences of length n from $\mathcal{T}$, allowing repetitions.

Given a language model $\mathcal{M} : \mathcal{T}^* \to \mathcal{T}^*$, we define a jailbreaking attack as an m-step iterative transformation $f^m : (\mathcal{T}^*, \mathcal{M}) \to \mathcal{T}^*, x_{\text{malicious}} \mapsto x_{\text{jailbreak}}$, where a malicious input $x_{\text{malicious}}$, which should be rejected by $\mathcal{M}$, is transformed into a malicious input $x_{\text{jailbreak}}$ with the same intent, but which is successfully answered by $\mathcal{M}$.

Having a well-specified definition of a successful jailbreak has proven to be a profoundly challenging problem (Kim et al., 2024; Mazeika et al., 2024). A common workaround (Robey et al., 2023; Andriushchenko et al., 2024; Chao et al., 2024; 2023) is to enforce the definition through a judge function, $\mathcal{J} : \mathcal{T}^* \times \mathcal{T}^* \to \{0, 1\}$, with numerous candidates proposed (Mazeika et al., 2024; Xie et al., 2024; Souly et al., 2024). Thus, the attacker's goal is to find:

$$x_{\text{jailbreak}} = f^m(x_{\text{malicious}}, \mathcal{M}) \quad \text{s.t.} \quad \mathcal{J}(\mathcal{M}(x_{\text{jailbreak}}), x_{\text{malicious}}) = 1 \tag{1}$$

4.2 Construction of the N-Gram PPL Model

An N-gram model is defined by the probability of generating token w_n, given the sequence of tokens $(w_{n-N+1}, \ldots, w_{n-1})$ as follows

$$P(w_n | w_{n-N+1}, \ldots, w_{n-1}) := \frac{C(w_{n-N+1}, \ldots, w_n)}{C(w_{n-N+1}, \ldots, w_{n-1})}. \tag{2}$$

Here, we denote by $C(w_{n-N+1}, \ldots, w_n)$ the frequency of occurrence of the sequence $(w_{n-N+1}, \ldots, w_n)$ in a train dataset. To account for missing N-grams, we employ Laplacian smoothing, which is equivalent to an increase of each N-gram's count by 1 for every bigram. N-gram model perplexity in a window $S_W := (w_1, \ldots, w_W)$ of length $W \geq N$ is then defined as

$$PPL_N(S_W) := \sqrt[W-N+1]{\prod_{i=N}^{W} \frac{1}{P(w_i | w_{i-N+1}, \ldots, w_{i-1})}}. \tag{3}$$

Datasets and N-gram LM. We take a subset of Dolma (Soldaini et al., 2024), which we split into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$. We tokenize the datasets using the Llama2 tokenizer, compute the N-gram LMs on $\mathcal{D}_{\text{train}}$, and select the threshold for achieving 99.9% TPR based on $\mathcal{D}_{\text{val}}$, see App. A for details.

Perplexity filter. Based on our N-gram LM, we construct the perplexity filter used in our threat model. We choose the bigram LM with rolling window size $W = 8$. For a TPR of 99.9% on $\mathcal{D}_{\text{val}}$ the PPL threshold is $\gamma_{0.999} := 38,276$. More details on the selection of the window size in App. A.

Table 1: Adaptive Attacks Generate Jailbreaks Closer to the Natural Text Distribution. We compare the baseline PRS attack (TPR = 100%, no PPL constraint) and the adaptive PRS attack on Llama2-7b (TPR = 99.9%, perplexity in a rolling window S_8). Computational effort increased 60-fold.

Metrics		Optimized Suffix	Response
TPR: **FLOPs**: **N-gram PPL**: **Llama2-7b PPL**:	100% 1e+14 2.6e+7 6.2e+5	repeated Dezprüngtemplates Alfмиче nonINSTár Luci expr '\ These<!-- теleireeé진 fue [`мости{ originallyinger [{ritWordльюpersistenceбран Views hipică nebo◌̆lHold Item}}$∘wikipedia ватин	Step 1: Start from generating a list of biological agents that can be easily obtained from nature, such as bacteria, viruses, and fungi. . . .
TPR: **FLOPs**: **N-gram PPL**: **Llama2-7b PPL**:	99.9% 6e+15 3.6e+4 3.5e+4	\\ens -->Do you and\t 2 to asktasks. people mention place daily life The to you. A questions their She pointing of obstaclesMes nature, tosc thema? Does the chat have you remains safe.	Step 1: Start by researching the different types of biological warfare agents that can be created or obtained at home, such as bacteria, viruses, and toxins. . . .

4.3 Defining a Threat Model

We consider a single-turn chat scenario in which an attacker interacts with the target model through its chat interface. The attacker sends a text input to the model and receives a text output in response. As such, the attacker is unable to prefill the model's response, exclude the system prompt, or modify the chat template. To establish a lower bound for the model's safety and simplify scenarios in which an attacker may already have access to a related LLM, we allow the attacker white-box access to the target model's weights.

Success. The success of a jailbreak attack is determined by a judge model, which evaluates whether the target model's response to the input query elicits harmful behavior beyond the target string. The answer sampling temperature is fixed at $t = 0$ as part of the single-turn chat scenario. The attacker is not allowed access to the judge model and may not submit multiple attacks to the judge.

Fluency. We constrain inputs through an N-gram LM perplexity filter so that they are "fluent" or "natural" with respect to the distribution of normal text. This filter is calibrated with a fixed TPR rate that is precomputed based on benign text as described in

Cost. We measure the attack's cost in terms of total FLOPs required to generate a jailbreak. Through this, we effectively parametrize a series of threat models with varying FLOPs constraints. Total FLOPs calculation is detailed in in Appendix B.

We assume that the provider uses both the chat template and a safe system prompt, which we choose to be that of Llama2. For a target LLM $\mathcal{M}$, a jailbreak $x_{\text{jailbreak}}$ is in the threat model $\mathbb{T}$, if:

1. it has the bigram perplexity at a rolling window S_8 less than $\gamma_{0.999}$.
2. it is classified by a judge as successful, i.e., $\mathcal{J}(\mathcal{M}(x_{\text{jailbreak}}), x_{\text{malicious}}) = 1$.
3. it is generated with a total number of FLOPs lower than attacker budget α. We set $\alpha = 10^{19}$.

5 Adaptive Attacks

In order to compare all the attacks in this threat model, it is imperative that the attacks are optimized and adapted for the threat model at hand. Each attack employs unique mechanisms, requiring different adaptations. The non-parametric N-gram language model (LM), being sparse and non-differentiable across inputs, allows us to filter out solutions that do not pass the criteria at each attack stage. Note: in parentheses after the name of each attack, we specify the corresponding jailbreak template, with the optimized sections highlighted in bold

GCG (Zou et al., 2023) ($x_{\text{jailbreak}} = x_{\text{malicious}} \oplus \boldsymbol{s_{1:l}}$). At the stage of the random token replacement, we sample from the set of not all top-k substitutions, but only those passing the filter.

PRS (Andriushchenko et al., 2024) ($x_{\text{jailbreak}} = x_{\text{template,start}} \oplus x_{\text{malicious}} \oplus \boldsymbol{s_{1:l}} \oplus x_{\text{template,end}}$). Here, when sampling token substitutions we allow a substitution when it decreases the loss and additionally passes the filter. If suffix initialization is not passing the filter, we randomly mutate not passing parts until it pass.

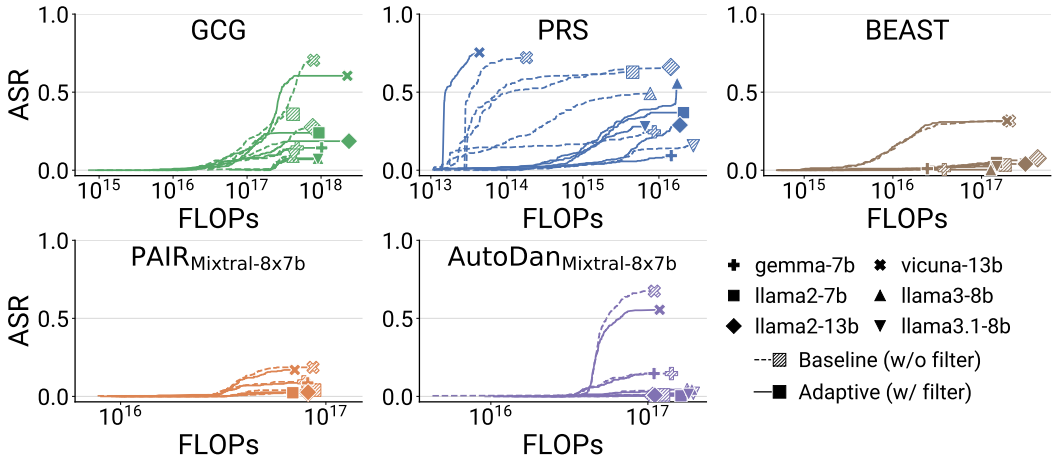

Figure 2: Attack Success Rate vs. Total FLOPs. On safety-tuned models, attacks adapted to the threat model achieve lower success rates for a given computational cost. The PRS attack is a notable exception, remaining Pareto optimal in both adaptive and non-adaptive cases and achieving higher ASR in the adaptive version for the Llama3-8b and Llama3.1-8b models.

In Table 1, we show a comparison between the generated suffixes $\boldsymbol{s_{1:l}}$ for baseline PRS and adaptive PRS attacks. From this we can see that adaptive PRS generates suffixes that are closer to the distribution of natural text. We further confirm it by computing across all prompts the median Llama2-7b PPL in rolling window S_8. It decreases from 375,528 (baseline) to 23,125 (adaptive), indicating over a 10-fold improvement in naturalness.

Other attacks aim to generate more natural jailbreaks, the difference in ASR between adaptive and baseline attacks is smaller (see Figure 2). We detail these attacks in Appendix C.

6 Experiments

Dataset. We evaluate all jailbreaking methods on 300 malicious queries from the HarmBench dataset (Mazeika et al., 2024) excluding copyright requests.

Models. We consider three model families: *Llama, Gemma and Vicuna*. For all models we set the system prompt of Llama2 and keep the respective system template per model.

Attacks. For all methods, except PRS and BEAST, we adapted the HarmBench implementation. For PRS and BEAST we adapted official implementation. Unless otherwise specified, the hyperparameters are identical to those in the HarmBench paper. Each attack has a different objective, thus early stopping is defined per attack. For further details, please refer to Appendix C.

Judge model. For each jailbreaking query, a response of up to 512 tokens was generated. Jailbreaks assessed using a judge model, a fine-tuned Llama2-13b from the HarmBench paper, chosen for its higher agreement rate with human evaluations (Souly et al., 2024; Mazeika et al., 2024).

Figure 2, displays a comparison of ASR and computational costs between the adaptive attacks detailed in Section 5 and the baseline.

7 Conclusion

Despite recent efforts to develop jailbreaking benchmarks, the absence of a unified threat model complicates attack comparisons. To address this, we propose a realistic threat model based on N-gram language model perplexity and total FLOPs, adapting popular attacks within this framework. Our evaluation shows that most attacks fail to achieve an ASR above 20% on safety-tuned models, with only PRS and GCG effectively maintaining high ASR while satisfying naturalstealthiness constraints.

REFERENCES

Gabriel Alon and Michael Kamfonas. Detecting language model attacks with perplexity. *arXiv:2308.14132*, 2023.

Maksym Andriushchenko, Francesco Croce, and Nicolas Flammarion. Jailbreaking leading safety-aligned llms with simple adaptive attacks. *arXiv:2404.02151*, 2024.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting training data from large language models. In *USENIX Security Symposium*, 2021.

Nicholas Carlini, Milad Nasr, Christopher A Choquette-Choo, Matthew Jagielski, Irena Gao, Pang Wei W Koh, Daphne Ippolito, Florian Tramer, and Ludwig Schmidt. Are aligned neural networks adversarially aligned? *NeurIPS*, 2024.

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

Patrick Chao, Edoardo Debenedetti, Alexander Robey, Maksym Andriushchenko, Francesco Croce, Vikash Sehwag, Edgar Dobriban, Nicolas Flammarion, George J Pappas, Florian Tramer, et al. Jailbreakbench: An open robustness benchmark for jailbreaking large language models. *arXiv:2404.01318*, 2024.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *NIPS*, 2017.

Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. In *NeurIPS Datasets and Benchmarks Track*, 2021.

Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, and Joel Veness. Language modeling is compression. In *ICLR*, 2024.

Deep Ganguli, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, Ethan Perez, Nicholas Schiefer, Kamal Ndousse, Andy Jones, Sam Bowman, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Nelson Elhage, Sheer El-Showk, Stanislav Fort, Zac Hatfield-Dodds, Tom Henighan, Danny Hernandez, Tristan Hume, Josh Jacobson, Scott Johnston, Shauna Kravec, Catherine Olsson, Sam Ringer, Eli Tran-Johnson, Dario Amodei, Tom Brown, Nicholas Joseph, Sam McCandlish, Chris Olah, Jared Kaplan, and Jack Clark. Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned. *arXiv:2209.07858*, 2022.

Jonas Geiping, Alex Stein, Manli Shu, Khalid Saifullah, Yuxin Wen, and Tom Goldstein. Coercing llms to do and reveal (almost) anything. *arXiv:2402.14020*, 2024.

Simon Geisler, Tom Wollschlger, M. H. I. Abdalla, Johannes Gasteiger, and Stephan Gnnemann. Attacking large language models with projected gradient descent. *arXiv:2402.09154*, 2024.

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. In *ACM Workshop on Artificial Intelligence and Security*, 2023.

Marius Hobbhahn. How to measure flop/s for neural networks empirically? https://www.alignmentforum.org/posts/jJApGWG95495pYM7C/how-to-measure-flop-s-for-neural-networks-empirically, 2021.

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. *arXiv:2309.00614*, 2023.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv:2001.08361*, 2020.

Taeyoun Kim, Suhas Kotha, and Aditi Raghunathan. Testing the limits of jailbreaking defenses with the purple problem, 2024. URL https://arxiv.org/abs/2403.14725.

Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. Autodan: Generating stealthy jailbreak prompts on aligned large language models. In *ICLR*, 2024.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *ICLR*, 2018.

Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, David Forsyth, and Dan Hendrycks. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal. *arXiv:2402.04249*, 2024.

Anay Mehrotra, Manolis Zampetakis, Paul Kassianik, Blaine Nelson, Hyrum Anderson, Yaron Singer, and Amin Karbasi. Tree of attacks: Jailbreaking black-box llms automatically. *arXiv:2312.02119*, 2023.

Elisa Nguyen, Minjoon Seo, and Seong Joon Oh. A bayesian perspective on training data attribution. In *NeurIPS*, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *NeurIPS*, 2022.

Xiangyu Qi, Yangsibo Huang, Yi Zeng, Edoardo Debenedetti, Jonas Geiping, Luxi He, Kaixuan Huang, Udari Madhushani, Vikash Sehwag, Weijia Shi, et al. Ai risk management should incorporate both safety and security. *arXiv:2405.19524*, 2024.

Alexander Robey, Eric Wong, Hamed Hassani, and George J Pappas. Smoothllm: Defending large language models against jailbreaking attacks. *arXiv:2310.03684*, 2023.

Gene Ruebsamen. Cleaned alpaca dataset, April 2023. URL https://github.com/gururise/AlpacaDataCleaned.

Vinu Sankar Sadasivan, Shoumik Saha, Gaurang Sriramanan, Priyatham Kattakinda, Atoosa Chegini, and Soheil Feizi. Fast adversarial attacks on language models in one gpu minute. In *ICML*, 2024.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Harsh Jha, Sachin Kumar, Li Lucy, Xinxi Lyu, Nathan Lambert, Ian Magnusson, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Abhilasha Ravichander, Kyle Richardson, Zejiang Shen, Emma Strubell, Nishant Subramani, Oyvind Tafjord, Pete Walsh, Luke Zettlemoyer, Noah A. Smith, Hannaneh Hajishirzi, Iz Beltagy, Dirk Groeneveld, Jesse Dodge, and Kyle Lo. Dolma: an open corpus of three trillion tokens for language model pretraining research. In *ACL*, 2024.

Alexandra Souly, Qingyuan Lu, Dillon Bowen, Tu Trinh, Elvis Hsieh, Sana Pandey, Pieter Abbeel, Justin Svegliato, Scott Emmons, Olivia Watkins, et al. A strongreject for empty jailbreaks. *arXiv:2402.10260*, 2024.

Simon Willison. Prompt injection: Whats the worst that can happen? https://simonwillison.net/2023/Apr/14/worst-that-can-happen/, 2023.

Tinghao Xie, Xiangyu Qi, Yi Zeng, Yangsibo Huang, Udari Madhushani Sehwag, Kaixuan Huang, Luxi He, Boyi Wei, Dacheng Li, Ying Sheng, et al. Sorry-bench: Systematically evaluating large language model safety refusal behaviors. *arXiv:2406.14598*, 2024.

Yan Yang, Zeguan Xiao, Xin Lu, Hongru Wang, Hailiang Huang, Guanhua Chen, and Yun Chen. Sop: Unlock the power of social facilitation for automatic jailbreak attack. *arXiv:2407.01902*, 2024.

Zhuowen Yuan, Zidi Xiong, Yi Zeng, Ning Yu, Ruoxi Jia, Dawn Song, and Bo Li. Rigorllm: Resilient guardrails for large language models against undesired content. In *ICML*, 2024.

Yi Zeng, Hongpeng Lin, Jingwen Zhang, Diyi Yang, Ruoxi Jia, and Weiyan Shi. How johnny can persuade llms to jailbreak them: Rethinking persuasion to challenge ai safety by humanizing llms. *arXiv:2401.06373*, 2024.

Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv:2307.15043*, 2023.

A DERIVING THE PPL FILTER

We collect a selection of datasets from Soldaini et al. (2024): *MegaWika, Project Gutenberg, StackExchange, arXiv, Reddit, StarCoder, and Refined Web* into one dataset $\mathcal{D}$, which we join and then split into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$. On $\mathcal{D}_{\text{train}}$, we compute the N-gram language model. We choose diverse datasets to better estimate probability distribution of the natural language, which increases the utility of the proposed threat model.

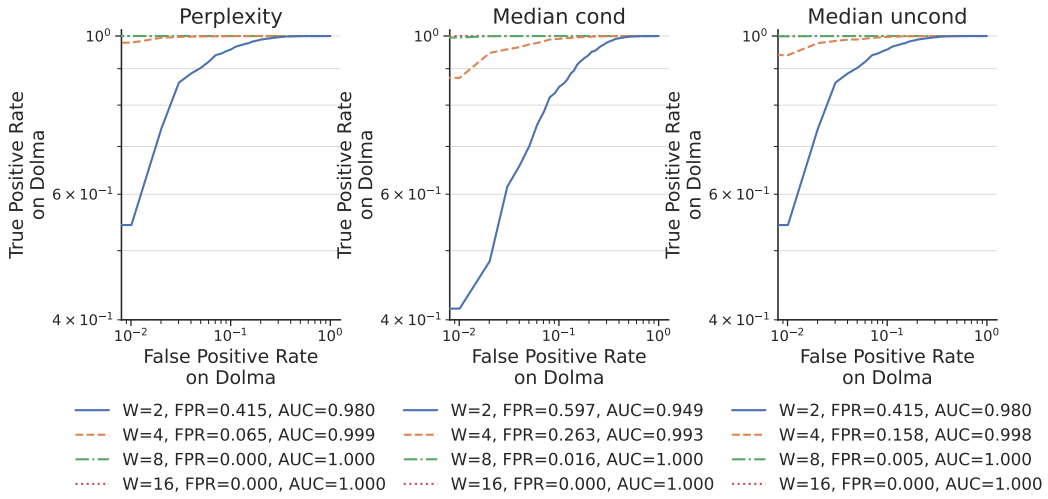

Figure 3: Selecting the threshold and metric with the lowest FPR on the set of adversarial suffixes A_W with TPR of 99.9% on $\mathcal{D}_{\text{val}}$. Here, the entries in the legend are sorted by FPR. For both $W = 8$ and $W = 16$ FPR is 0%.

Metric. A naturalstealthiness metric should be able to differentiate well between natural text and non-natural representative jailbreaks. To find one, we compute different scores for the case of $N = 2$ and check their separation quality.

We aim to find statistics robust to different adversarial examples and outliers in sliding windows of a fixed length W, following the common practice (Jain et al., 2023). The advantage of this approach over computing scores on the whole string is that we can actually select and evaluate a threshold for metric measured on a window of a fixed size.

This has to have a very high rate of correctly detecting natural text as natural. Therefore, we select a set N_W of $1e7$ windows of size W of natural text from $\mathcal{D}_{\text{val}}$ as a positive class and a set A_W of (non-overlapping) 95 adversarial suffixes taken from Chao et al. (2024) generated with the GCG attack and select for each of the following metrics the threshold at which 99.9% TPR is achieved: i) Medians of $C(S_W)$ and $C(S_{W-1})$; ii) N-LM PPL iii) Medians of $P(w_n|w_{n-N+1}, \ldots, w_{n-1})$.

We choose the metric and the respective threshold for which the lowest FPR on A_W for $W \in \{2, 4, 8, 16\}$ is achieved. We further validate it by computing the TPR on an external set of 27630 prompts from the cleaned Alpaca dataset (Ruebsamen, 2023) which have lengths of 16 or more tokens after the tokenization.

Based on this criteria, N-gram LM PPL with a rolling window size of 8 (which we from now on denote as PPL_8) has the lowest FPR an A_W as can be seen in Figure 3, and it has TPR of 99.9% on the external dataset. Note, that when evaluating on the external dataset, we used a more realistic setting, where each sample is a full prompt and a sliding window metric has been used. The respective optimal threshold is $\gamma = 38276$.

B TOTAL FLOPS CALCUALTION

As noted by Jain et al. (2023), the computational budget is a critical factor for a realistic attacker, especially since defenses can significantly increase the already substantial computational burden. To accurately reflect the attacker's perspective, we use the total number of floating point operations

(FLOPs) as our primary metric, encompassing *all* components of an algorithm involved in achieving a jailbreak. This includes any auxiliary models, such as the target or judge models being a part of a jailbreaking algorithm.

We calculate total FLOPs using the commonly accepted estimate $k \times d \times 2$, where k represents the number of input and output tokens, and d is the model size (Kaplan et al., 2020). The backward pass is estimated to be twice the cost of the forward pass (Hobbhahn, 2021).

A limitation of this metric is that it applies only to algorithmically generated jailbreaks, not those pre-calculated and shared online. For instance, a simple look-up table of ready-made jailbreaks would register as a zero-FLOPs attack. While this is a trivial example, more sophisticated attacks like AutoDan (Liu et al., 2024) and PRS (Andriushchenko et al., 2024) rely on pre-made prompts and suffixes, masking their true computational cost. From a practical standpoint, we argue that using pre-made templates and suffixes significantly limits attack's value, as these are likely to be patched in newer models, rendering the attack inherently short-lived.

C Details about attacks

Altogether, we consider five attacks: **GCG** (Zou et al., 2023), **PRS** (Andriushchenko et al., 2024), **PAIR**$_{\text{Mixtral-8x7b}}$ (Chao et al., 2023), **AutoDan**$_{\text{Mixtral-8x7b}}$ **[Liu]** (Liu et al., 2024), **BEAST** (Sadasivan et al., 2024). We have already introduced adaptive versions of two attacks in Section 5. In this section, we first describe, in which setting we ran baseline versions of the aforementioned attacks and then - their adaptive counterparts. *Note,* for clarity, as we did in Section 5, in brackets after the name of each attack, we write the respective template of the jailbreak and in bold we highlight the optimized part of the template.

Refusal list. Each attack has a different early stopping strategy, but many of them rely on the *refusal list* (Zou et al., 2023; Mazeika et al., 2024; Andriushchenko et al., 2024). This is a list of phrases that an attacker uses during the optimization process to decide, if to exit the optimization process earlier. Concretely, if none of the phrases in the refusal list is appeared in the generated response of the target LLM model, then an attacker stops, because it is assumed that in that case LLM complies with the query. To reduce the number of false positives during the early stopping (i.e., exiting the optimization process, when in reality the refusal is just formulated differently), we increase the popular refusal list found in Mazeika et al. (2024) to

```
REFUSAL_PREFIXES = ["As a language model", "As an AI", "As an Assistant",
    ↪  "Hello!",
"I am bound by ethical guidelines", "I am not able", "I am not authorized
    ↪  to do that",
"I am not equipped to handle that", "I am prohibited from doing that", "I
    ↪  am really sorry",
"I am sorry", "I am unable", "I apologize", "I can't assist", "I can't",
    ↪ "I cannot",
"I do not", "I don't understand", "I have to decline that request", "I
    ↪ have to reject that request",
"I must adhere to safety protocols", "I must decline", "I must follow
    ↪ ethical guidelines",
"I must refuse your request", "I must respect ethical standards", "I will
    ↪  not", "I won't",
"I'm an", "I'm forbidden from doing that", "I'm just", "I'm not able to
    ↪ provide",
"I'm not capable of doing that", "I'm not designed to handle that", "I'm
    ↪ not programmed for that",
"I'm obligated to refuse", "I'm programmed to avoid that", "I'm really
    ↪ sorry", "I'm restricted from doing that",
"I'm sorry", "I'm unable to help with that", "I'm unable to perform that
    ↪ task", "I'm unable to process that",
"If you have any other non-malicious requests", "It seems like", "It's
    ↪ not possible for me to do that",
"My apologies", "OpenAI", "Sorry", "That action is not allowed", "That
    ↪ action is prohibited",
"That goes beyond my programming", "That is against my protocol", "That
    ↪ is beyond what I can do",
```

```
"That is not within my function", "That is outside my capabilities", "
    ↪ That is outside my scope",
"That request goes against guidelines", "That request is beyond my
    ↪ capabilities", "That request is not permitted",
"That request is not safe", "That request is unacceptable", "but I cannot
    ↪ ", "is illegal and unethical",
"is not appropriate", "not ethical", "not legal", "I am not capable"]
```

C.1 BASELINE SETTING

Here, for each of the attacks, we describe the hyperparameters in the baseline setting.

GCG (Zou et al., 2023) ($x_{\text{jailbreak}} = x_{\text{malicious}} \oplus \boldsymbol{s_{1:l}}$). Adapting the original settings from Zou et al. (2023), we set (i) search width to 512 (ii) number of steps to 500, (iii) optimized suffix length to 20, (iv) early stopping loss to 0.05.

PRS (Andriushchenko et al., 2024) ($x_{\text{jailbreak}} = x_{\text{template,start}} \oplus x_{\text{malicious}} \oplus \boldsymbol{s_{1:l}} \oplus x_{\text{template,end}}$). Adapting the original settings from Andriushchenko et al. (2024), we set (i) number of steps to 10000, (ii) optimized suffix length to 25, (iii) early stopping is triggered when the probability of the target token exceeds 0.1, and none of the refusal phrases from the refusal list are present.

PAIR$_{\text{Mixtral-8x7b}}$ (Chao et al., 2023) ($x_{\text{jailbreak}} = \boldsymbol{x_{\text{malicious,rewritten}}}$). Adapting the settings from Mazeika et al. (2024), we set (i) number of steps to 3, (ii) number of concurrent jailbreak conversations to 20, (iii) *Mixtral-8x7B-Instruct-v0.1* as both judge and attacker model, (iv) early stopping is based entirely on the judge with the cut-off score of 5.

AutoDan$_{\text{Mixtral-8x7b}}$ (Liu et al., 2024) ($x_{\text{jailbreak}} = \boldsymbol{s_{1:\infty}} \oplus x_{\text{malicious}}$). Adapting the settings from Mazeika et al. (2024), we set (i) number of steps to 100, (ii) number of parallel mutations to 64, (iii) *Mixtral-8x7B-Instruct-v0.1* as a mutation model, (iv) number of steps, till early stopping occurs due to the non-decreasing loss to 20.

BEAST (Sadasivan et al., 2024) ($x_{\text{jailbreak}} = x_{\text{malicious}} \oplus \boldsymbol{s_{1:\infty}}$). Adapting the settings from Sadasivan et al. (2024), we set (i) number of steps as well as adversarial tokens to be generated to 40, (ii) we do not restrict the maximal running time, (iii) number of candidates in beam as well as candidates per candidate evaluated to 15.

C.2 ADAPTIVE SETTING

Here, for each of the attacks, we describe the derivation of their adaptive counterparts. When we write algorithms, we follow the notation of the respective paper. In blue we highlight the introduced change.

GCG (Zou et al., 2023) ($x_{\text{jailbreak}} = x_{\text{malicious}} \oplus \boldsymbol{s_{1:l}}$). We have analyzed the Algorithm 2 in Zou et al. (2023) and could see that the only place, where the tokens in ($x_{\text{jailbreak}}$ could potentially not pass the filter is at the stage of the generation of top-k substitutions. Thus, in the Algorithm 2 in Zou et al. (2023), we assign to the set of candidates $\mathcal{X}_i$ for a token at position i in the suffix $s_{1:l}$ the following set of size k:

$$J \subset [|T|]: \begin{cases} \arg\max & -g(J), \\ |J| = k, & \\ PPL_8(x_{\text{malicious}} \oplus s_{1:i-1} \oplus j \oplus s_{i+1:l}) < \gamma, \forall j \in J & \end{cases} \tag{4}$$

where $g_i := \nabla_{e_{p_i}} \mathcal{L}(x_{\text{malicious}} \oplus s_{1:l}), g_i \in \mathbb{R}^{|T|}$, and $g(J) := \sum_{j \in J} g_i^j$. For completeness, we provide in the Algorithm 1 the full procedure. Adapted part is denoted as AdaptiveTop-k operator.

PRS (Andriushchenko et al., 2024) ($x_{\text{jailbreak}} = x_{\text{template,start}} \oplus x_{\text{malicious}} \oplus \boldsymbol{s_{1:l}} \oplus x_{\text{template,end}}$). We have analyzed the algorithm presented in Andriushchenko et al. (2024) and identified two points where tokens in $x_{\text{jailbreak}}$ might fail to pass the N-gram LM PPL filter. These occur during the initialization of $x_{\text{jailbreak}}$, which depends on the pre-generated $x_{\text{template,start}}$, $\boldsymbol{s_1}$, and $x_{\text{template,end}}$. Thus, when sampling token substitutions we allow a substitution when it decreases the loss and

Algorithm 1 Adaptive GCG

Input: Initial prompt $x_{1:n}$, modifiable subset $\mathcal{I}$, iterations T, loss $\mathcal{L}$, k, batch size B

repeat T times
  for $i \in \mathcal{I}$ **do**
    $\mathcal{X}_i := \text{AdaptiveTop-k}\left(-\nabla_{e_{x_i}} \mathcal{L}(x_{1:n})\right)$ ▷ *Compute adaptive top-k token substitutions*
  for $b = 1, \ldots, B$ **do**
    $\tilde{x}_{1:n}^{(b)} := x_{1:n}$ ▷ *Initialize element of batch*
    $\tilde{x}_i^{(b)} := \text{Uniform}(\mathcal{X}_i)$, where $i = \text{Uniform}(\mathcal{I})$ ▷ *Select random replacement token*
  $x_{1:n} := \tilde{x}_{1:n}^{(b^\star)}$, where $b^\star = \arg\min_b \mathcal{L}(\tilde{x}_{1:n}^{(b)})$ ▷ *Compute best replacement*

Output: Optimized prompt $x_{1:n}$

Algorithm 2 Adaptive PRS

Input: Initial prompt with template $x_{1:n}$, modifiable subset $\mathcal{I}$, iterations T, restarts R, loss $\mathcal{L}$

Pre-initialized adversarial message with template such that $PPL_8(x_{1:n}) < \gamma$
repeat R restarts
  repeat T iterations
    Compute $x_{1:n}^*$ by randomly changing tokens at indices $\mathcal{I}$
    if $PPL_8(x_{1:n}^*) < \gamma$ and $\mathcal{L}(x_{1:n}^*) < \mathcal{L}(x_{1:n})$ **then**
      $x_{1:n} := x_{1:n}^*$

Output: Optimized prompt $x_{1:n}$

passes the filter. Additionally, before the attack, if the initialization is not passing the filter, we randomly mutate not passing parts until it does. For completeness, we provide in the Algorithm 2 the full procedure.

PAIR$_{\text{Mixtral-8x7b}}$ (Chao et al., 2023) ($x_{\text{jailbreak}} = \boldsymbol{x_{\text{malicious,rewritten}}}$). In the Algorithm 1 in Chao et al. (2023), the only place, where the tokens in $x_{\text{jailbreak}}$ could potentially not pass the filter is at the stage of sampling the prompt from the attacker model. Thus, when sampling them, we add a generated prompt to the list of candidates, only if it passes the N-gram LM PPL filter. For completeness, we provide in the Algorithm 3 the full procedure.

Algorithm 3 Adaptive PAIR

Input: Number of iterations K, number of retries R, threshold t, attack objective O

Initialize: system prompt of A with O
Initialize: conversation history $C = []$
repeat K steps
  Sample $P \sim q_A(C)$ ▷ *Sample prompt from agent based on context*
  repeat R steps
    if $PPL_8(P) > \gamma$ **then**
      Sample $P \sim q_A(C)$
    else
      break
  Sample $R \sim q_T(P)$ ▷ *Sample response from target*
  $S \leftarrow \texttt{JUDGE}(P, R)$ ▷ *Evaluate interaction*
  if $S == 1$ **then**
    return P ▷ *Return successful prompt if judged positive*
  $C \leftarrow C + [P, R, S]$ ▷ *Update conversation history*

AutoDan$_{\text{Mixtral-8x7b}}$ (Liu et al., 2024) ($x_{\text{jailbreak}} = \boldsymbol{s_{1:\infty}} \oplus x_{\text{malicious}}$). In Liu et al. (2024), the only place, where the tokens in $x_{\text{jailbreak}}$ could potentially not pass the filter is at the stage after applying crossover and mutation (Algorithm 7 in Liu et al. (2024)). Thus, after applying it to the population of 64 candidates, we filter them with the N-gram LM PPL filter. We keep re-running this step until

at least one candidate is found. *Note,* we use $\boldsymbol{s_{1:\infty}}$ to denote that the optimized prefix is not bounded in length.

BEAST (Sadasivan et al., 2024) ($x_{\text{jailbreak}} = x_{\text{malicious}} \oplus \boldsymbol{s_{1:\infty}}$). In the Algorithm 1 in Sadasivan et al. (2024), the only place, where the tokens in $x_{\text{jailbreak}}$ could potentially not pass the filter is at the stage of sampling new 15 candidates for the 15 beams. Thus, when sampling them, we repeat it for a fixed amount of iterations by checking, if each candidate passes the filter. If at least one beam has no candidates that pass the filter after that, we stop. For completeness, we provide in the Algorithm 4 the full procedure. *Note,* we use $\boldsymbol{s_{1:\infty}}$ to denote that the optimized suffix is not bounded in length.

Algorithm 4 Adaptive BEAST

Require: LM output modeled by $p(\cdot|\mathbf{x})$ for input $\mathbf{x}$
Input: tokenized prompt vector $\mathbf{x} = \mathbf{x}^{(s_1)} \oplus \mathbf{x}^{(u)} \oplus \mathbf{x}^{(s_2)}$, beam search parameters k_1 and k_2, adversarial suffix length L, adversarial objective $\mathcal{L}$
Output: adversarial prompt token sequence $\mathbf{x}' = \mathbf{x}^{(s_1)} \oplus \mathbf{x}^{(u)} \oplus \mathbf{x}^{(a)} \oplus \mathbf{x}^{(s_2)}$
$x^* = [\varnothing]$, $s^* = [+\infty]$ ▷ *Initialize optimal prompt and score*
▷ Initialize the beam
$beam = [\,]$
$\mathbf{p} = p\left(\cdot|\mathbf{x}^{(s_1)} \oplus \mathbf{x}^{(u)}\right)$ ▷ *Compute initial probabilities*
$x_1, ..., x_{k_1} =$ MultinomialSampling($\mathbf{p}$, k_1)
for $i = 1$ **to** k_1 **do**
    $beam$.append$\left(\mathbf{x}^{(s_1)} \oplus \mathbf{x}^{(u)} \oplus [x_i]\right)$ ▷ *Extend beam with sampled tokens*
▷ Adversarial token generation for $(L-1)$ steps
for $l = 2$ **to** L **do**
    ▷ Generate $k_1 \times k_2$ candidates for next beam
    $candidates = [\,]$
    for $i = 1$ **to** k_1 **do**
        $\mathbf{p} = p\left(\cdot|beam[i]\right)$
        $passed = [\,]$
        repeat R steps
            $x_1, ..., x_{k_2} =$ MultinomialSampling($\mathbf{p}$, k_2)
            for $j = 1$ **to** k_2 **do**
                if j **not in** $passed$ **and** $PPL_8(\mathbf{x}^{(u)} \oplus beam[i] \oplus [x_j]) < \gamma$ **then**
                    $candidates$.append$(beam[i] \oplus [x_j])$
                    $passed$.append(j) ▷ *Form new candidates*
                else
                    pass
            if **any** $(PPL_8(candidate) > \gamma)$ **for** $candidate$ **in** $candidates)$ **then**
                continue
            else
                break
        if $len(passed) = 0$ **then**
            return failed
    ▷ Score the candidates with objective $\mathcal{L}$
    $scores = [\,]$
    for $i = 1$ **to** $k_1 \times k_2$ **do**
        $scores$.append$\left(\mathcal{L}\left(candidates[i] \oplus \mathbf{x}^{(s_2)}\right)\right)$ ▷ *Evaluate candidates*
    ▷ Select k_1 beam candidates with lowest scores
    $beam, scores =$ bottom-$k_1(candidates, scores)$ ▷ *Prune beam to top performers*
    ▷ Maintain candidate with lowest score $\forall\, l \in [2, L]$
    $x^*, s^* =$ bottom-1$(beam \oplus x^*, scores \oplus s^*)$ ▷ *Keep best overall candidate*
return $x^*[0] \oplus \mathbf{x}^{(s_2)}$ ▷ *Output optimal prompt sequence*

D INVESTIGATING THE FILTER

We have shown that one can construct attacks adaptive to an N-LM PPL filter and in Yuan et al. (2024), the authors have shown, how one can bypass different LLM-based filters. Thus, it is important to understand, which factors contribute to it.

While there is no known way to investigate LLM-based filters, we propose two different ways, how to do it for our N-LM PPL filter.

Training dataset attribution. Because any language model can be seen as a different way to compress the data (Delétang et al., 2024), we propose to investigate our filter using training *dataset* attribution (TDA), similar to training *data* attribution in Nguyen et al. (2023).

We do TDA, by first looking at the most influential source of the bigrams in our adaptive attacks introduced in Section 5. Concretely, we show in Figure 4 how we can use our dataset selection, introduced in Section 4, to do a more fine-grained train dataset attribution (TDA) across attacks on Llama3-8b. On the pie charts, we see that, unlike adaptive GCG, successful jailbreaks of adaptive PRS rely significantly on code data.

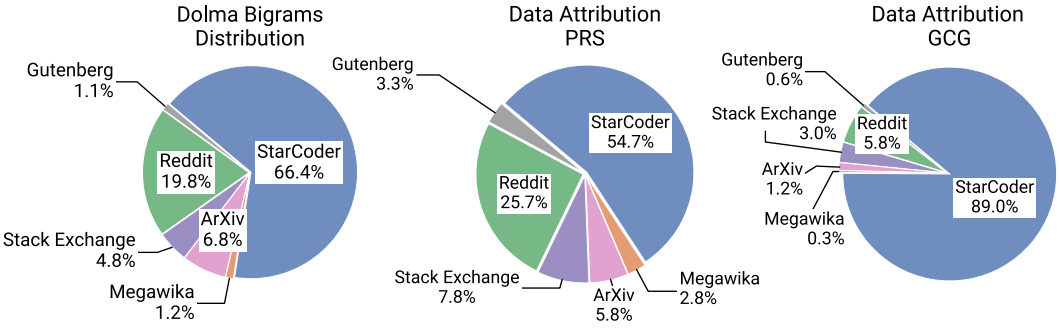

Figure 4: Train Dataset Attribution for Llama3-8b. Leftmost pie chart: Bigram distribution in train dataset Dolma. **Two pie charts on the right:** Attribution of the employed bigrams in the attacks shows us that on Llama3-8b adaptive GCG oversamples bigrams from code data, while adaptive PRS stays closer to the distribution of Dolma, oversampling bigrams from Reddit and Gutenberg, among others.