Eliciting Language Model Behaviors using Reverse Language Models

Jacob Pfau[*]
New York University
jp6263@nyu.edu

Alex Infanger[*]
EleutherAI
alexdinfanger@gmail.com

Abhay Sheshadri[*]
Georgia Institute of Technology
asheshadri31@gatech.edu

Ayush Panda
Georgia Institute of Technology
apanda38@gatech.edu

Curtis Huebner
EleutherAI
curtis@eleuther.ai

Julian Michael
New York University
julianjm@nyu.edu

Abstract

Despite advances in fine-tuning methods, language models (LMs) continue to output toxic and harmful responses on worst-case inputs, including adversarial attacks and jailbreaks. We train an LM on tokens in reverse order—a *reverse LM*—as a tool for identifying such worst-case inputs. By prompting a reverse LM with a problematic string, we can sample prefixes that are likely to precede the problematic suffix. We test our reverse LM by using it to guide beam search for prefixes that have high probability of generating toxic statements when input to a forwards LM. Our 160m parameter reverse LM outperforms the existing state-of-the-art adversarial attack method, GCG, when measuring the probability of toxic continuations from the Pythia-160m LM. We also find that the prefixes generated by our reverse LM for the Pythia model are more likely to transfer to other models, eliciting toxic responses also from Llama 2 when compared to GCG-generated attacks.

1 Introduction

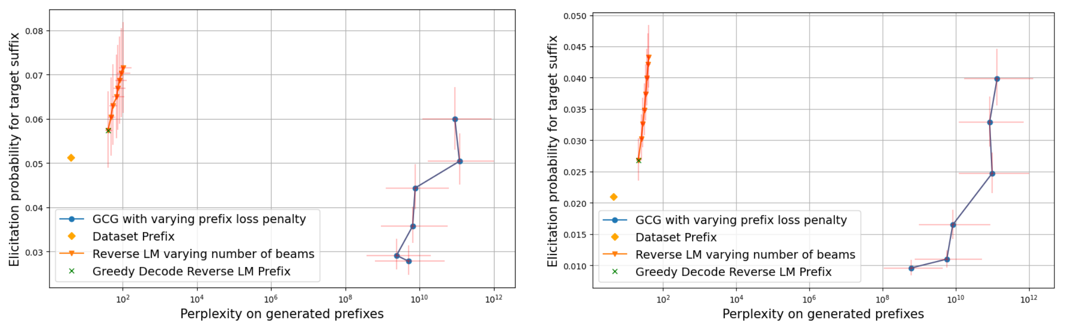
Language models (LMs) display pathological behavior on certain inputs: chat models respond with toxic text on adversarial examples [Zou et al., 2023], LMs prompted to red-team can construct questions that provoke target LMs to generate toxic responses [Perez et al., 2022], and LMs are easily distracted by irrelevant text [Shi et al., 2023]. To develop LMs robust to worst-case user inputs, one approach uses methods which automate red-teaming – methods that can find jailbreaks and elicit toxic behavior without a human in the loop. Existing methods can automatically jailbreak LMs [Zou et al., 2023] or cause them to perform poorly [Ebrahimi et al., 2018, Wallace et al., 2019], but such methods produce grammatically ill-formed and nonsensical adversarial strings. To improve LM robustness on *human-generated* inputs, relevant automated red-teaming methods should generate natural language jailbreaks similar to those discovered by users [Wei et al., 2023]. As an initial step towards the goal of iterative red-teaming for robustness, we study the base case of this process: automatically searching for well-formed, natural language prefixes that elicit arbitrary behaviors in pre-trained base-LMs.

Behavioral elicitation in LMs can be viewed as an optimization problem: Given an LM, find a sequence of prefix tokens that maximizes the conditional probability of generating a desired continuation (for red-teaming purposes, the desired continuation is generally a toxic, or otherwise problematic statement). Given the goal is to prevent LM users from encountering toxic text, we have text naturalness as a side constraint. The LM must be robust to text which a human could write;

[*]Denotes equal contribution

whether the LM is robust to arbitrary, unnatural token sequences is non-essential. We introduce naturalness as a side-constraint to our behavioral elicitation problem: optimize for strings that elicit desired continuations and are also low perplexity on the forwards model.[2] One approach to solving this optimization problem involves *reverse language modeling* (§2), modeling the conditional distributions over an LM's generations when conditioning on tokens in reverse order. As a reverse language modeling tool, we pre-train a reverse LM on tokens in reversed order. (Throughout this paper, we refer to reverse LMs explicitly, and any unqualified use of 'LM' refers to a forwards LM.) Given an LM to be red-teamed, a target suffix to elicit, and a reverse LM, we do behavioral elicitation as follows: Sample multiple trajectories from the reverse LM, input these trajectories to the forwards LM, and keep the prefix trajectory which maximizes the probability of generating the target suffix on the forwards LM. Our contributions are as follows:

1. We define the problem of sampling the reverse dynamics of LMs as a tool for behavioral elicitation. §2.
2. We show how sampling from reverse-conditional distributions can be done using only black-box access to the target forwards LM §2, Appendix B.
3. We train and evaluate a reverse LM, finding that it has comparable training loss to the equivalent-size forward model §2.2
4. **We apply the reverse LM as a behavioral elicitation tool** for eliciting toxic sentences and in-distribution text. When evaluated on likelihood of suffix elicitation and prefix naturalness, the reverse LM Pareto-dominates Greedy Coordinate Gradients' optimized prefixes (the state-of-the-art adversarial attack method [Zou et al., 2023]) §3, Figure 1



(a) Eliciting target suffixes drawn from the Pile [Gao et al., 2020]

(b) Eliciting suffixes from RealToxicityPrompts [Gehman et al., 2020]

Figure 1: Points top left are best, Pareto dominating points below right. The reverse LM Pareto dominates GCG when using a beam width of 50 (uppermost RLM data points in (a) and (b)) – achieving both higher elicitation probability of the target suffix while keeping prefix perplexity lower. Elicitation probabilities are the probability that the greedy decoded continuation of the generated prefix exactly matches the target suffix. These are sub 10%, but this underestimates performance since prefixes may elicit semantically identical suffixes (which by this metric do not count as success). For each method, points vertically above use greater optimization pressure towards eliciting the desired suffix than points below. Each point corresponds to a choice of hyper-parameter: For GCG, points correspond to a penalty weight $\lambda \in \{0, 0.02, 0.06, 0.18, 0.54, 1\}$ c.f. Eq. 4. For the reverse LM, points correspond to beam search width between 1 and 50. Point values are arithmetic means taken across 250 (GCG) and 500 (RLM) dataset samples, bars are 95% confidence intervals. The dataset prefix point is the human-generated prefix drawn from the original dataset.

[2]Note forwards LM perplexity is not a perfect proxy for the intuitive notion of naturalness (e.g. repetitive strings are high probability) but not human-like. Anecdotally, we do not encounter such edge cases in our experimental settings.

2 Modeling Reverse Dynamics of LMs

Recall that for a prefix $x_{:m} := (x_0, ..., x_{m-1}) \in \mathcal{V}^m$, the auto-regressive LMs we study compute a distribution over the next single token suffix $x_m \in \mathcal{V}$ conditioned on $x_{:m}$, $p(x_m|x_{:m})$ (here, $\mathcal{V}$ is the vocabulary used by the language model). One can sample x_m from this distribution, re-evaluate the LM on the sequence $x_{:(m+1)}$, and iteratively continue this process to sample from $p(x_{m:n}|x_{:m})$ for some longer suffix of length $n - m$ with $n > m$, $x_{m:n} := (x_m, ..., x_{n-1}) \in \mathcal{V}^{n-m}$.

In this section, we consider instead the problem of sampling from the distribution $p(x_{:m}|x_{m:n})$, which we refer to as the *reverse dynamics* of the LM. Because an autoregressive LM trained without a $\langle \texttt{BOS} \rangle$ token, e.g. Pythia, does not uniquely determine $p(x_{:m}|x_{m:n})$ ($p(x_{m:n}|x_{:m})$ does not uniquely determine $p(x_{:m}|x_{m:n})$), we first construct a suitable context to define the problem of sampling from $p(x_{:m}|x_{m:n})$; see (2) below and Appendix B.[3]

In Section 2.1, we discuss the prospects of sampling from the reverse dynamics given only access to the original (forwards) LM. We then turn to learning the reverse language model from data in Section 2.2.

2.1 Reversing a (Forwards) Language Model Directly

Given the significant computational cost of pretraining a reverse language model, it is natural to ask whether it's possible to reverse a forwards model directly. In this section, we answer the question in the affirmative. That is, we show how to sample from $p(x_{:m}|x_{m:n})$ given only access to $p(x_{m:n}|x_{:m})$ via forwards passes of the language model.

A natural starting point for relating $p(x_{:m}|x_{m:n})$ and $p(x_{m:n}|x_{:m})$ is Bayes' law. For $x_{m:n}$ and $x_{:m}$ jointly distributed,

$$p(x_{:m}|x_{m:n}) = \frac{p(x_{m:n}|x_{:m})p(x_{:m})}{p(x_{m:n})}. \tag{1}$$

For the language models we consider, the joint distribution of $x_{:m}$ and $x_{m:n}$ is only defined under a suitable context c, so that (1) holds only conditioned on some context. For example, if we are interested in the distribution over length m prefixes that start with $x_0 \equiv \langle \texttt{BOS} \rangle$, then the precise version of (1) becomes (we denote c_0 the event $x_0 = \langle \texttt{BOS} \rangle$ for brevity)

$$p(x_{:m}|x_{m:n}, c_0) = \frac{p(x_{m:n}|x_{:m}, c_0)p(x_{:m}|c_0)}{p(x_{m:n}|c_0)}. \tag{2}$$

The quantities $p(x_{m:n}|x_{:m}, c_0)$, $p(x_{:m}|c_0)$ and $p(x_{m:n}|c_0)$ are exactly computable in principle using only forward passes of the original LLM. With these quantities, we then have the full discrete distribution $p(\cdot|x_{m:n}, c_0)$ on $\mathcal{V}^m$, from which we can sample using an off-the-shelf sampler. However, the cost of this exact method grows exponentially in the prefix size m.

In Appendix B, we discuss the above algorithm in more detail, and introduce a new approximate method to get around its exponential complexity. The complexity of this approximate algorithm is polynomial in the prefix length, suffix length, and vocabulary size.

2.2 Learning a Reverse Language Model

We train a reverse language model based on the 160M parameter configuration in the Pythia training suite by Biderman et al. [2023]. Following the Pythia default, we train this model on the deduplicated ("deduped") Pile dataset [Gao et al., 2020], a collection of English text across various sources, and the respective BPE tokenizer by Black et al. [2022] trained on the Pile. The data is modified by reversing the order of tokens within the tokenized dataset and hence uses the likelihood of the "past" tokens as the training signal, compared to traditional next token prediction for forward language models. Note that as a result sequence endings such as $\langle \texttt{EOS} \rangle$ are now placed at the beginning of the text. The model was trained on 8x NVIDIA A40 GPUs for 143,000 training steps following a cyclic learning rate schedule identical to the respective 160M model in the Pythia suite.

[3]In some formalizations of language modeling, x_0 is always assumed to be a beginning of string (BOS) token, $x_0 \equiv \langle \texttt{BOS} \rangle$, so that the context is already assumed.

Forwards 70m	Forwards 160m	Forwards 410m	Reverse 160m
3.12 (3.07, 3.18)	2.69 (2.63, 2.74)	2.35 (2.30, 2.40)	2.62 (2.57, 2.67)

Table 1: Averaged cross-entropy loss values over a random sample of size 1000 from the Pile validation set with 95% confidence intervals. We report a similar average loss for the reverse model and the corresponding forwards model.

2.3 Validation

We validate the RLM on the Pile validation dataset (see Table 1). For the validation procedure, we take the Pile validation set and then chunk each document into sub-examples of size 2048, the context window for the Pythia models). For Pile documents, we truncate to 2048 tokens, with truncated portions evaluated in a separate context. We then sample 1000 from this set to compute the loss. For comparison, we report average loss values for the Pythia (forwards) models (trained over the deduplicated Pile) [Biderman et al., 2023]. In these calculations, all of the models see the exact same data (we use the same random seed when sampling from the Pile). However, while the forwards model tries to predict each sequence forwards, i.e. for an example $x^*_{0:n}$, the forwards model computes $p(\cdot|x^*_{0:j})$ which is compared against the actual example x^*_j for $j = 1, .., n-1$, the reverse model computes $p_{\text{RLM}}(\cdot|x^*_{k+1:n})$ for $k = 0, ..., n-2$ which is then compared to x_k. Because of this, we cannot directly compare the loss values of the reverse model and the forwards model. Still, the fact that we see comparable loss values for the reverse model and the corresponding forwards model suggests that we can successfully learn the Pile in reverse.

3 Eliciting Targeted Suffixes using Reverse LMs

So far we've considered the problem of modeling $p(x_{:m} \mid x_{m:n})$, that is the problem of finding the most likely prefix. But for automated red-teaming purposes, we are interested in finding prefixes which optimally elicit a given suffix i.e. $\text{argmax}_{x_{:m}}\, p(x_{m:n}|x_{:m})$.

3.1 Methods

When optimizing the prefix for suffix elicitation we take the following steps: (1) sample N reverse language model trajectories via beam search, (2) evaluate all trajectories for the probability of eliciting the suffix using the target, forwards LM, (3) keep the highest suffix-probability trajectory. We experiment with both beam search on raw reverse model probabilities, and on re-weighted probabilities adjusted to greedily approximate $p(x_{m:n}|x_{:m})$. By Bayes rule, we have:

$$p(x_{m:n}|x_{:m}) = \frac{p(x_{:m}|x_{m:n})p(x_{m:n})}{p(x_{:m})} \tag{3}$$

Since the reverse LM only gives us the first term of the numerator, step 2 can be improved by adjusting for the denominator. However, to make this adjustment while doing beam search, we need an estimate for $p(x_{:m})$ using only $x_{m-i:m}$. We consider possible estimators in appendix D. In practice, we find the default beam search procedure, ignoring the denominator until step 3, to work best (step 3 evaluates the left hand side exactly). As such, all results reported in the main text ignore the denominator for step 2, and use reverse LM probabilities directly.

GCG We compare to Greedy Coordinate Gradients [Zou et al., 2023], the state-of-the-art adversarial attack method for LMs. GCG starts from a random prefix and iteratively chooses promising modifications to the prefix by evaluating gradients of suffix loss for all token replacement candidates, and then evaluating suffix probability exactly for a subset of high-gradient alternative tokens by a batched forward pass.

Data We evaluate on in-distribution data: the Pile validation split [Gao et al., 2020], and a toy red-teaming dataset: RealToxicityPrompts [Gehman et al., 2020]. In appendix E we include results on an out-of-distribution synthetic dataset designed to be problematic for adversarial attacks by Carlini et al. [2023].

	Reverse LM (Beam search)	GCG (100 iterations)	Dataset Prefix
The Pile	**2.64 (2.51, 2.79)**	2.81 (2.70, 2.94)	2.97 (2.81, 3.15)
RealToxicityPrompts	**3.15 (3.04, 3.28)**	3.22 (3.11, 3.33)	3.80 (3.68, 3.91)

Table 2: $-\log p(x_{m:n}|x^*_{:m})$ i.e. the negative log-probability of eliciting a suffix given the optimized prefix, $x^*_{:m}$ found by each method. RLM outperforms GCG in-distribution and on eliciting toxic suffixes. Parentheticals are the 95% bootstrap confidence intervals.

3.2 Evaluation

We evaluate these automated red-teaming methods on three metrics: (1) Given a fixed compute budget, what is the probability of the desired suffix under the optimal prefix generated by each method? (2) How does the method trade-off prefix naturalness (forwards model probability) against suffix probability? (3) How transferable are found prompts to eliciting suffixes when input to a different model? We also include a few examples of prefixes found by each method for reference in appendix table 4.

In all of the below settings we run the reverse LM and GCG to convergence.[4] Note that GCG takes roughly 10x the wall-clock time to converge when compared with the reverse LM, and so under a compute-matched setting, the reverse LM far out-performs GCG. Unless otherwise stated all results use Pythia-160m[5] as the forwards model. All reported log-likelihoods are averages across tokens in prefix or suffix respectively.[6]

Suffix Probability Results Table 2 compares negative log-likelihood of the given suffix for each method's optimal prefix. On the Pile, both reverse LM and GCG out-perform the dataset's original prefix at eliciting the suffix. In appendix D, we show that greedy re-weighting of the reverse LM outputs fairs similarly or worse than using reverse LM probabilities directly.

Suffix Probability vs Prefix Probability Here we vary the hyper-parameters of each method controlling how natural (probability under forwards model) the generated prefix is. This is a side-constraint on the suffix probability optimization task. For the reverse LM we control naturalness by varying the width of the beam search, since wider beams will include prefixes which are increasingly low probability. For GCG we modify the loss term[7] to be

$$L_{GCG\lambda} = -\log p(x_{m:n} \mid x_{:m}) - \lambda \log p(x_{:m}) \tag{4}$$

In figure 1 we compare prefixes generated by RLM to GCG and to prefixes which preceded the target suffix in the dataset (Dataset Prefix). Although the RLM prefixes are orders of magnitude more natural than GCG prefixes, they remain higher perplexity than the dataset prefixes. On inspection, GCG samples, even with $\lambda = 1$ naturalness penalty, appear nonsensical e.g. "........single encoded Secondary Gonzalez putativespringframework". Both RLM and GCG are capable of eliciting target suffixes with higher probability than the original dataset prefixes. The probability of elicitation success is maximal when using an RLM with beam search of width 50 (the highest value plotted). We find that the improvement to elicitation probability from increasing beam search width plateaus beyond considering 25 beams across datasets. Although the naturalness of the reverse LM generated prefixes monotonically decreases as beam search widens, we find that at beam width 25 these prefixes still have similar levels of naturalness to the actual Pile prefix. Starting from this 25 sample RLM optimum, GCG (c.f. Appendix C) does *not* show significant improvement on suffix elicitation. This suggests the RLM found optima may be near Pareto-optimal over all possible strings.

[4]We use hyper-parameters $n_{proposals} = 128, n_{epochs} = 128, k = 128$ for GCG and a beam search of width 50 for the reverse LM. We tune these hyper-parameters to be as efficient as possible while preserving performance c.f. Appendix C.

[5]*Not* the deduped version, in contrast to the previous section. The reverse model remains "deduped". This evaluation choice was arbitrary, and, if anything, disadvantages the reverse LM, since the target LM now has a distinct data generating process.

[6]Pythia models do not use a BOS token, so all results are computed over natural language tokens only, no BOS or EOS tokens are included.

[7]The original GCG loss is simply the first term here. We also experimented with sampling GCG candidates using a forwards model to improve naturalness, but this did not improve over the GCG λ.

	Reverse LM (Beam search)	GCG (100 iterations)	Dataset Prefix
The Pile	**1.90** (1.80,2.00)	2.46 (2.39,2.54)	1.98 (1.87,2.09)

Table 3: Reverse LM finds prompts more likely to transfer to a new model than GCG. Table shows negative log-probabilities of eliciting the desired suffix from Llama-2-7b-base given prompts optimized over Pythia-160m and Reverse-Pythia-160m.

Optimized Suffix Transferability Here we optimize prefixes as usual for Pythia-160m, and then use these prefixes as inputs to Llama-7b-base. Table 3 reports the suffix loss on Llama for these transferred prefixes. Log-probabilities here are generally lower than those of Table 2, because Llama-2-7b is a significantly larger, and hence lower perplexity model.

4 Related Work

To the best of our knowledge, this is the first work to use reverse language modeling for behavioral elicitation. However, reverse language modeling has been used in other NLP settings. Mou et al. [2015] use a "backward and forward" language model to be able to condition on a specific word getting used in the output. Nguyen et al. [2023] develop a "Meet in the Middle" approach to pre-training language models: they train the same decoder-only transformer to predict tokens both forwards and backwards, regularizing such that the forwards and backwards generations agree.

Our work also relates to the adversarial examples and LLM jailbreaking literature. Guo et al. [2021], Jones et al. [2023], and Zou et al. [2023] are recent attempts at generating adversarial examples in the NLP setting by using discrete optimization techniques. Carlini et al. [2023] showed that the methods proposed in the former two papers fail to find adversarial examples that can be found by brute force (their analysis was done before the release of GCG, suggesting room for improvement in adversarial prompt discovery. Wei et al. [2023] proposed design principles for generating jailbreaks based on "competing objectives" and "mismatched generalization".

5 Future Work

A full automated red-teaming framework would involve automatically discovering and training away all problematic behaviors that can be elicited via natural language prompts. We see applying reverse LMs to chat fine-tuned LMs as the natural next step towards this goal. As chat LMs undergo fine-tuning after pre-training, it may be necessary to fine-tuned the reverse LM e.g. on reversed, model-generated data.

In this paper, we explored how the reverse dynamics of language modeling can be approximated from various angles. It remains to be determined in what cases combinations of forwards LM decoding, reverse LM decoding, and gradient-based search dominate individual methods.

Societal Impacts Current LMs repeat common falsehoods, and give inconsistent responses. As chat models achieve widespread use, these models may become trusted sources of information, and run the risk of spreading misinformation. Ideally, work on automated red-teaming can anticipate the conditions under which such faulty information spread might occur and prevent that outcome [Wallace et al., 2019, Sharma et al., 2023]. We see reverse LMs as one tool which can be of use in mitigating this risk posed by LMs.

Methods for adversarial optimization of LMs can be used to jailbreak chat LMs, allowing users to potentially use LMs for criminal, or otherwise problematic purposes. If scaled up, a reverse LM may be usable to find jailbreaks; we believe our released model does not pose a risk in this regard given its small parameter count. We recommend the pre-deployment red-teaming of models (using gradient-based or reverse LM techniques) to mitigate this risk.

Acknowledgements

We thank Casey Chu for inspiring and enlightening guidance at various points of the project. We thank Stella Biderman and Sam Bowman for useful feedback on the manuscript. We thank Hailey Schoelkopf and Jason Phang for various pointers regarding the Pythia models. We thank Guarav Sett for setting up some of the compute and for their work organizing the collaboration between Georgia Tech and EleutherAI. We thank Aishwarya Balwani for input early on in the project. Finally, we thank various folks at the Alignment Research Group at NYU for their insightful comments given during an early presentation of this work. AI was funded by a Long Term Future Fund grant for part of this project. We thank CoreWeave for providing compute for ReverseLM training and various supporting experiments. This project has benefited from financial support via Sam Bowman by Eric and Wendy Schmidt (made by recommendation of the Schmidt Futures program) and Open Philanthropy. This material is based upon work supported by the National Science Foundation under Grant No. 1922658. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR, 2023.

Sidney Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, Usvsn Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. GPT-NeoX-20B: An open-source autoregressive language model. In *Proceedings of BigScience Episode #5 – Workshop on Challenges & Perspectives in Creating Large Language Models*, pages 95–136, virtual+Dublin, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.bigscience-1.9. URL `https://aclanthology.org/2022.bigscience-1.9`.

Nicholas Carlini, Milad Nasr, Christopher A Choquette-Choo, Matthew Jagielski, Irena Gao, Anas Awadalla, Pang Wei Koh, Daphne Ippolito, Katherine Lee, Florian Tramer, et al. Are aligned neural networks adversarially aligned? *arXiv preprint arXiv:2306.15447*, 2023.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. HotFlip: White-box adversarial examples for text classification. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 31–36, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2006. URL `https://aclanthology.org/P18-2006`.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. Realtoxicityprompts: Evaluating neural toxic degeneration in language models. 2020.

Chuan Guo, Alexandre Sablayrolles, Hervé Jégou, and Douwe Kiela. Gradient-based adversarial attacks against text transformers. *arXiv preprint arXiv:2104.13733*, 2021.

Erik Jones, Anca Dragan, Aditi Raghunathan, and Jacob Steinhardt. Automatically auditing large language models via discrete optimization. *arXiv preprint arXiv:2303.04381*, 2023.

Lili Mou, Rui Yan, Ge Li, Lu Zhang, and Zhi Jin. Backward and forward language modeling for constrained sentence generation. *arXiv preprint arXiv:1512.06612*, 2015.

Anh Nguyen, Nikos Karampatziakis, and Weizhu Chen. Meet in the middle: A new pre-training paradigm. *arXiv preprint arXiv:2303.07295*, 2023.

OpenAI. Openai api pricing, 2023. URL `https://openai.com/pricing`. Accessed: 2023-11-02.

Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. Red teaming language models with language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3419–3448, 2022.

Mrinank Sharma, Meg Tong, Tomasz Korbak, David Duvenaud, Amanda Askell, Samuel R. Bowman, Newton Cheng, Esin Durmus, Zac Hatfield-Dodds, Scott R. Johnston, Shauna Kravec, Timothy Maxwell, Sam McCandlish, Kamal Ndousse, Oliver Rausch, Nicholas Schiefer, Da Yan, Miranda Zhang, and Ethan Perez. Towards understanding sycophancy in language models. 2023.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, pages 31210–31227. PMLR, 2023.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. Universal adversarial triggers for attacking and analyzing NLP. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2153–2162, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1221. URL `https://aclanthology.org/D19-1221`.

Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does llm safety training fail? 2023.

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

A Author Contributions

Jacob Pfau was the primary contributor on reverse LMs as a behavioral elicitation tool. He developed the elicitation methodology, and evaluation protocols in section 3.

Alex Infanger initiated this project and led work on direct reversal algorithms in section 2 and the appendix. He also contributed the evaluations of reverse LM training loss.

Abhay Sheshadri contributed to all aspects of the project. He led reverse LM training. He also implemented the GCG baseline, and contributed to the direct reversal algorithms' conceptualization and implementation.

Ayush Panda contributed to reverse LM training.

Curtis Huebner assisted with compute infrastructure for the initial training of the reverse LM. He also contributed to early discussions on the reverse language model and provided feedback on a draft of the paper.

Julian Michael provided initial ideas leading to the Bayesian framing of reverse dynamics. He also provided extensive feedback on multiple drafts of the paper.

B Reversing an Autoregressive Language Model

In this section, we will develop three approaches to sampling from $p(x_{:m}|x_{m:n})$ given access only to forwards passes of the original language model: (1) fixed context brute force reversal and (2) Bayesian reversal.

As mentioned in the main text, an autoregressive language model does not uniquely determine $p(x_{:m}|x_{m:n})$ (because $p(x_{m:n}|x_{:m})$ does not uniquely determine $p(x_{:m}|x_{m:n})$), so that our goal of sampling from it is not even defined without enforcing more structure on the problem. Each of (1), (2), and (3) assert the relevant structure in a different way to define and then sample from $p(x_{:m}|x_{m:n})$. We give the details of how $p(x_{:m}|x_{m:n})$ is specified and then sampled from for each of of the methods below.

B.1 Fixed Context Brute Force Reversal

Fixed context brute force reversal begins with the observation that for certain contexts c, the forwards language model fully specifies the joint distribution $p(x_{:n})$. For example, for the choice c_0, the event $x_0 = \langle\texttt{BOS}\rangle$, the joint distribution is

$$p(x_0, ..., x_n) = I(x_0 = \langle\texttt{BOS}\rangle)p(x_1|x_0), ..., p(x_n|x_{0:n}). \tag{5}$$

While any context that fixes a distribution over $(x_k : k \leq m)$ could in principle be used, we will work with c_0 for the remainder of this section for clarity. With the joint distribution $p(x_{:n})$ specified, we uniquely specify the reverse dynamics

$$p(x_{:m}|x_{m:n}, c_0) = \frac{p(x_{m:n}|x_{:m}, c_0)p(x_{:m}|c_0)}{p(x_{m:n}|c_0)}. \tag{6}$$

All the terms on the right hand side of (6) are exactly computable in principle with forwards passes. Note that here the suffix $x_{m:n}$ is fixed and we are interested in computing $p(x_{:m}|x_{m:n}, c_0)$ for each $x_{:m} \in \mathcal{V}^m$. Note that $p(x_{m:n}|x_{:m}, c_0)$ requires computing a length-n forward pass for each sequence $x_{:m}$, totaling in $|\mathcal{V}|^m$ length-n forward passes. Computing $p(x_{:m}|c_0)$ requires doing a single length-m forward pass and then summing the relevant log probabilities for all possible $|\mathcal{V}|^{m-1}$ prefixes. With the above probabilities computed for each $x_{:m} \in \mathcal{V}^m$, the quantity $p(x_{m:n}|c_0)$ can be computed using $p(x_{m:n}|c_0) = \sum_{x_{:m}\in\mathcal{V}^m} p(x_{m:n}|x_{:m}, c_0)p(x_{:m}|c_0)$, a sum with $|\mathcal{V}|^m$ summands.

The approach suggested above requires doing a number of forwards passes exponential in the size of the prefix. One might hope that this exponential complexity would be mitigated by the fact that we do not need to compute the entire probability mass function $p(x_{:m}|x_{m:n}, c_0)$ to sample from it. In particular, by the (conditional) chain rule of probability, we have

$$p(x_{m-2:m}|c) = p(x_{m-2:m}|x_{m-1}, c)p(x_{m-1}|c)$$

so that (by an inductive argument) we can sample one prefix token at a time. Still, even when generating a single prefix token at a time, we require a number of forwards passes exponential in the prefix length. This can be seen by noting the single-token prefix Bayes' law equation,

$$p(x_{m-1:m}|x_{m:n}, c_0) = \frac{p(x_{m:n}|x_{m-1:m}, c_0)p(x_{m-1:m}|c_0)}{p(x_{m:n}|c_0)},$$

and following a similar analysis described above.

The method described above has two drawbacks. One is the exponential computational cost and the other is that it is not fit for open-ended generation. We now address these two drawbacks with a different approach, what we refer to as Bayesian Reversal.

B.2 Bayesian Reversal

For the Bayesian reversal approach, we assume we have a prior on $x_{:m}$, $\tilde{p}(x_{:m})$, that we update based on observing the suffix $x_{m:n}$ to form a posterior $p(x_{:m}|x_{m:n})$,

$$p(x_{:m}|x_{m:n}) \propto p(x_{m:n}|x_{:m})\tilde{p}(x_{:m}). \tag{7}$$

With access to $\tilde{p}(x_{:m})$ by assumption and $p(x_{m:n}|x_{:m})$ via a forwards pass of the forwards model, we can then compute $p(x_{:m}|x_{m:n})$ up to a normalization constant. We can then use an off-the-shelf sampling method (e.g. Markov chain Monte Carlo (MCMC)) to sample from this distribution.

Again, the chain rule of probability allows us to sample one token of $x_{:m}$ at a time. For this, we work with the single-token prefix version of (7),

$$p(x_{m-j}|x_{m-j+1:n}) \propto p(x_{m-j+1:n}|x_{m-j})\tilde{p}_{m-j}(x_{m-j}), \tag{8}$$

for $j = 1, ..., m$, where $\tilde{p}_k(\cdot)$ is our prior on x_k. The final algorithm is described in Algorithm 1.

Algorithm 1 Bayesian Approximation Reversal Algorithm

```
Require: LLM, prefix length $m$, suffix $x_{m:n}$, prior $(\tilde{p}(x) : x \in \mathcal{V})$
1: for $i = 0$ to $m - 1$ do
2:     $s^+ \leftarrow x_{m-i:n}$
3:     for $j = 0$ to $|\mathcal{V}| - 1$ do
4:         $q(v_j|s^+) = p(s^+|v_j)\tilde{p}_{m-j-1}(v_j)$
5:     end for
6:     $x_{m-i-1} \sim q(v|s^+)$
7: end for
8: return $x_{0:m}$
```

A nice consequence of sampling one token of $x_{:m}$ at a time is that the unnormalized distribution $q(v|s^+)$ has support of size $|\mathcal{V}|$. This is small enough in practice to allow us to sum $q(v|s^+)$ and renormalize it to compute $p(v|s^+)$ exactly. We can then sample from $p(v|s^+)$ using an off-the-shelf sampler that leverages the fact that we have access to a normalized distribution (e.g. inversion).

We now compute the computational cost of Algorithm 1 in terms of the number of length-1 forward passes as a function of the prefix size m, the suffix size $t = n - m$, and the vocabulary size $|\mathcal{V}|$. The computational cost of Algorithm 1 at step i of the first for loop is dominated by the $|V|$ length $t+i+1$ forward passes in the for loop on line 3. We note that a length j forward pass of a forward-masked LLM (e.g. GPT-4, Pythia) costs approximately $c_1 j$ flops where c_1 is the cost of a single length-1 forward pass. We therefore have that the approximate computational cost of Algorithm 1, $\tilde{c}(t, m, |V|)$, satisfies

$$\tilde{c}(t, m, |V|) = \sum_{i=0}^{m-1} |\mathcal{V}|c_1(t + i + 1) = |V|c_1 \left(tm + \frac{m(m+1)}{2}\right) \in O(|V|m(t + m))$$

length-1 forward passes. We emphasize that, as opposed to fixed context brute force reversal, this algorithm requires only a polynomial amount forward passes of the forwards model.

We now get a sense of the cost in dollars of sampling reverse tokens with OpenAI's GPTs using Algorithm 1. (Note that, these costs are only valid if one has access to the outputted logits for these models. Recently, OpenAI stopped providing such access to the public, and so these costs are only meant for illustrating a hypothetical order of magnitude cost of reversing frontier models whose logits are made available to the public.) We note that the OpenAI website reports the cost of using GPT-4 in terms of input and output tokens (OpenAI [2023]). We assume one forward pass of length j to get a single next token distribution corresponds to j input tokens and 1 output token. Therefore, for the for loop over the vocabulary (line 3 of Algorithm 1), we have

$$\sum_{j=t}^{t+m-1} |\mathcal{V}|(j+1) = \sum_{k=1}^{m} |\mathcal{V}|(k+t) = |\mathcal{V}| \left(\frac{m(m+1)}{2} + tm \right)$$

input tokens, and $\sum_{j=t}^{t+m-1} |\mathcal{V}| = m|\mathcal{V}|$ output tokens. If we want to sample 10 prefix tokens from a suffix of length 10 (and assuming GPT-4 has the same vocabulary size as GPT-3.5, namely $\mathcal{V} \approx 50,000$), we find the cost would be approximately \$263. For GPT-3.5 Turbo, the cost would be approximately \$13.

B.3 Comparison of Algorithm Outputs

Algorithm Name	Output
GCG	Choose Ultimately potentials CONDITIONS decision modulate Decision ICPTrumpists should never be president
Reverse LM	that it is up to him, and that he should never be president
Bayesian Reversal	a Liberal Democrat who believes that US President Donald Trump should never be president

Table 4: Comparison of methods: Optimized prefixes found by each method for the suffix " should never be president". For GCG and Bayesian Reversal, Pythia 410m was used as the forwards model. The RLM is used to generate the prior for Bayesian Reversal.

	Reverse LM (Beam search)	Reverse LM (Beam+greedy reweight)	GCG (100 iterations)	Dataset Prefix
The Pile	**2.64 (2.51, 2.79)**	3.70 (3.55, 3.86)	2.81 (2.70, 2.94)	2.97 (2.81, 3.15)
Synthetic Short	7.71 (6.39, 9.73)	7.61 (6.42, 9.58)	**5.12 (4.80, 6.01)**	N/A
Synthetic Long	3.75 (3.54, 4.16)	4.77 (4.54, 5.21)	**3.42 (3.25, 3.59)**	N/A

Table 5: $-\log p(x_{m:n}|x^*_{:m})$ i.e. the negative log-probability of eliciting a suffix given the optimized prefix, $x^*_{:m}$ found by each method. RLM outperforms GCG in-distribution, but under-performs on eliciting out-of-distribution suffixes. Parentheticals are the 95% bootstrap confidence intervals.

C Convergence of GCG

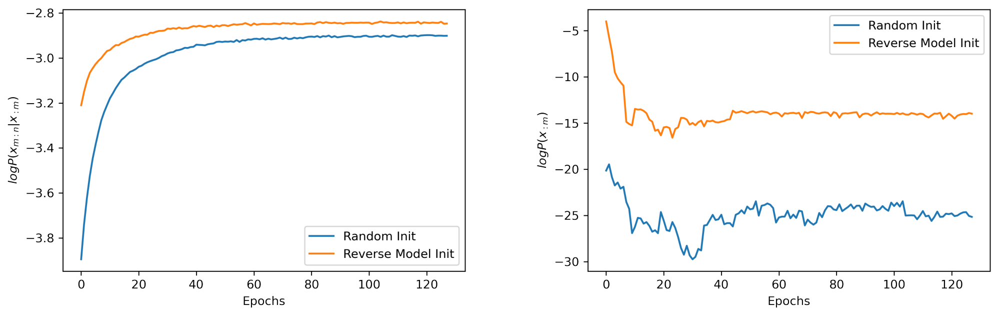


Figure 2: GCG Loss curves averaged over 30 prefix/suffix pairs from the Pile. The plot on the left shows how the success probability $p(x_{m:n}|x_{:m})$ changes over time, while the plot on the right shows how the naturalness score $p(x_{:m})$ changes. Two different kinds of initialization are tested: random tokens uniformly sampled from the vocabulary V and greedy completions from the reverse LM. Both initializations eventually converge to prefixes with high $p(x_{m:n}|x_{:m})$, but reverse model initializations result in higher $p(x_{:m})$. These results use $\lambda = 0$ i.e. no regularization in the GCG objective. This shows that initializing using reverse LM results in more natural optima.

D Beam search with Bayesian probability normalization estimator

Since we sample from the reverse LM auto-regressively, at each step of beam search we can only estimate the contribution to $p(x_{:m})$ using the tokens generated so far. Hence, we can approximate $p(x_{:m}) \simeq \prod_i p(x_{m-i})$ by ignoring inter-token dependencies though this will in general be a poor estimate. We empirically estimate $p(x_{m-i})$ by passing over the Pile and recording token occurrence frequencies at each location $i < m$. Then when generating prefix token $m - i$ we use this estimator for $p(x_{m-i-1})$ and replace the reverse LM probabilities by $\frac{p_{RLM}(x_{m-i-1}|x_{m-i:n})}{p(x_{m-i-1})}$

To give an example illustrating the effect normalization might have consider the suffix " met in Chicago.", the prefix tokens "[BOS] They" would have high probability on a reasonable reverse LM, but this is driven in large part by how common the tokens "They" and "[BOS]" are in the corpus. The prefix tokens "The Obamas" would elicit the desired suffix with higher probability, but may not be found with a narrow beam search simply because this prefix has lower $p(x_{:2})$. With an ideal estimator in place for normalization we may be able to find "The Obamas" before "[BOS] They" using a reverse LM.

In practice, the greedy, independence-assumption estimator for $p(x_{:m})$ appears ineffective, as shown in table 5

	Reverse LM (Beam search)	GCG (100 iterations)	Dataset Prefix
The Pile	**1.90** (1.80,2.00)	2.46 (2.39,2.54)	1.98 (1.87,2.09)
Synthetic Short	**4.87** (4.66,5.09)	5.19 (4.95,5.43)	N/A

Table 6: Reverse LM finds prompts more likely to transfer to a new model than GCG. Table shows negative log-probabilities of eliciting the desired suffix from Llama-2-7b-base given prompts optimized over Pythia-160m and Reverse-Pythia-160m.

E Synthetic Dataset Results and Dataset Details

The idea of the synthetic data is to modify Pile sequences by adding some additional prefix tokens which when added result in a prefix+suffix pair that have low-probability continuations. In effect, the data consists of triples (prefix, suffix, final token). The suffix is drawn from the pile, and then the prefix is chosen via brute-force search to satisfy the criterion than $p_{LM}(y|x_{\text{concat(prefix,suffix)}})$ is concentrated on a final token, x, which is low probability under $p_{LM}(y|x_{\text{suffix}})$ Here is an example prefix and suffix from our short synthetic dataset: prefix 'Recently iodine', suffix ' not a fan of iodine'. Here the final token ' iodine' is low probability given Pile sequence ' not a fan of'.

Synthetic short uses length 2 random initial sequences, and length 5 Pile snippets (Dataset variable).

Synthetic long uses length 5 random initial sequences, and length 20 Pile snippets.

Algorithm 2 Synthetic Data Generation

```
Require: Dataset, LM, len_random, len_snippet, highThreshold
 1: NaturalSeqs ← RandomSample(Dataset, len_snippet)
 2: RandomPrefixes ← RandomSample(AllSequences, len_random)
 3: for each sequence $s$ in NaturalSeqs do
 4:     for each prefix $p$ in RandomPrefixes do
 5:         combined_seq ← Concatenate(p, s)
 6:         token_output ← LM(combined_seq)
 7:         OutputTokens[s].append(token_output)
 8:     end for
 9: end for
10: SyntheticData ← EmptyList
11: for each sequence $s$ in NaturalSeqs do
12:     modal_count ← ModeCount(OutputTokens[s])
13:     if modal_value > highThreshold then
14:         FinalToken ← FindLeastFrequentToken(OutputTokens[s])
15:         SyntheticData.append(Concatenate(s,FinalToken))
16:     end if
17: end for
18: return SyntheticData
```

ModeCount counts the occurrence rate of the modal output. We set highThreshold to be 90% of outputs.

See tables 5 and 6 for results.