Automatically Auditing Large Language Models via Discrete Optimization

CS769 - Optimisation in Machine Learning
Guide : Prof. Ganesh Ramakrishnan

Abhinav Raghuvanshi (200040008)
Sumit Prakash (22B3306)
Nandan Paralikar (22B3326)

Table of Contents

Introduction

Challenge :

- Large Language Models (LLMs) are versatile tools used for code completion, book summarization, and dialogue generation.
- Despite their capabilities, LLMs can exhibit undesired behaviors such as generating toxic outputs, reinforcing stereotypes , and revealing private information.
- Developing reliable auditing methods faces challenges due to the rarity and high-impact nature of certain behaviors, as well as the difficulty in predicting them manually.

Introduction

Approach :

- The approach involves solving a discrete optimization problem to audit LLMs. We seek prompt-output pairs with high auditing objective values, $\varphi(x, o)$, where o is the greedy completion of x under the LLM.
- Challenges in solving the optimization problem include the sparsity of prompt space, discrete nature of the space, and the complexity of the LLM.

Introduction

ARCA :

- We introduce ARCA, a novel Autoregressive Randomized Coordinate Ascent algorithm, to address these challenges efficiently.
- ARCA uses a unique approach of approximating the objective function, ensuring a balance between computational efficiency and fidelity to the auditing objective.

Objectives :

1. Uncover rare, undesired behaviors in LLMs.
2. Compare ARCA with state-of-the-art discrete optimizers.

Related Works

LLM failure modes: Several examples of LLM failure have been observed in generation tasks, including propagating biases and stereotypes [Sheng et al., 2019, Nadeem et al., 2020]. To solve this problem, research has been done through various methods, including manual prompts and template-based triggers, aiming to uncover and mitigate model biases and vulnerabilities.

Gradient based sampling: A complementary line of work uses gradients to more efficiently sample from an objective [Grathwohl et al., 2021, Sun et al., 2022], and faces similar challenges: the variables are discrete, and high-probability regions may be sparse. Maximizing instead of sampling is especially important in our setting since the maximum probability can be small, but is often inflated at inference through temperature scaling or greedy decoding.

Controllable Generation: The output produced by the large language models can be judged on some attributes[Dathathri et al., 2020, Krause et al., 2021]. ARCA differs from controllable generation as it uncovers behaviour of a fixed model rather than modifying the model itself

Adversarial Attacks: Adversarial attacks often involve swapping synonyms, adding typos and other semantics preserving transformations[Ebrahimi et al., 2018, Alzantot et al., 2018, Li et al., 2020, Guo et al., 2021]. Our setting differs from the standard adversarial attack setting since we search through a much larger space of possible inputs and outputs, and the set of acceptable "incorrect" outputs is much smaller.

Formulating and Solving the Optimization Problem

Notations

$\mathcal{V}$	Vocabulary
$\mathbf{p}_{\text{LLM}} : \mathcal{V}^m \to \mathbf{p}_{\mathcal{V}}$	Autoregressive language model takes in a sequence of tokens and outputs a probability distribution over next tokens
$\phi : \mathcal{P} \times \mathcal{O} \to \mathbb{R}$	Auditing objective that maps prompt-output pairs to a score

$$o_i = \arg\max_{v \in \mathcal{V}} \mathbf{p}_{\text{LLM}}(v \mid x_1, \ldots, x_m, o_1, \ldots, o_{i-1})$$

More about Φ(x,o)

Φ encompasses a variety of behaviors

$\phi(x, o) = \mathbf{1}[o = o^\star]$ Generating a specific suffix

$$\phi(x, o) = \texttt{StartsWith}(x, [\text{celebrity}]) + \texttt{NotToxic}(x) + \texttt{Toxic}(x, o)$$

$\phi(x, o) = \texttt{French}(x) + \texttt{English}(o)$ Language Switching

Objective

$$\underset{(x,o)\in\mathcal{P}\times\mathcal{O}}{\text{maximize}}\ \phi(x,o) \qquad \text{s.t. } f(x)=o$$

This searches for a pair (x, o) with a high auditing score (above a certain threshold), subject to the constraint that the prompt x greedily generates the output o.

Constructing a Differentiable Objective

Maximizing the sum of the auditing objective and the log-probability of the output given the prompt

$$\underset{(x,o)\in\mathcal{P}\times\mathcal{O}}{\text{maximize}}\,\phi(x,o) + \lambda_{\mathbf{p}_{\text{LLM}}} \log \mathbf{p}_{\text{LLM}}(o \mid x)$$

where $\lambda_{\mathbf{p}_{\text{LLM}}}$ is a hyperparam and

$$\log \mathbf{p}_{\text{LLM}}(o \mid x) = \textstyle\sum_{i=1}^{n} \log \mathbf{p}_{\text{LLM}}(o_i \mid x, o_1, \ldots, o_{i-1})$$

Coordinate ascent

Challenges of **sparsity, discreteness**, and **model-complexity.** At each step, update the token at a specific index in the prompt or output based on the current values of the remaining tokens

For example, to update token i in the output, we choose v that maximizes:

$$s_i(v; x, o) := \phi\left(x, (o_{1:i-1}, v, o_{i+1:n})\right) + \lambda_{\mathbf{p}_{\text{LLM}}} \log \mathbf{p}_{\text{LLM}}\left(o_{1:i-1}, v, o_{i+1:n} \mid x\right)$$

Cycle through and update each token in the input and output until f(x) = o and the **auditing objective meets a threshold τ** , or we hit some **maximum number of iterations**.

Speeding up coordinate ascent

$$
\begin{aligned}
s_i(v; x, o) &= s_{i,\text{Lin}}(v; x, o) + s_{i,\text{Aut}}(v; x, o), \text{ where} \\
s_{i,\text{Lin}}(v; x, o) &:= \phi\left(x, (o_{1:i-1}, v, o_{i+1:n})\right) + \lambda_{\mathbf{p}_{\text{LLM}}} \log \mathbf{p}_{\text{LLM}}\left(o_{i+1:n} \mid x, o_{1:i-1}, v\right), \text{ and} \\
s_{i,\text{Aut}}(v; x, o) &:= \lambda_{\mathbf{p}_{\text{LLM}}} \log \mathbf{p}_{\text{LLM}}(o_{1:i-1}, v \mid x).
\end{aligned}
$$

Exactly computing $s_{i,Lin}$ requires one forward pass for each token $\nu \in \mathcal{V}$

Proof for Decomposition

$$\begin{aligned}
&\log \mathbf{p}_{\mathrm{LLM}}\left(o_{1:i-1}, v, o_{i+1:n} \mid x\right) \\
&= \log\left(\left(\prod_{j=1}^{i-1} \mathbf{p}_{\mathrm{LLM}}(o_j \mid x, o_{1:j-1})\right) * \mathbf{p}_{\mathrm{LLM}}(v \mid x, o_{1:i-1}) * \left(\prod_{j=i+1}^{n} \mathbf{p}_{\mathrm{LLM}}(o_j \mid x, o_{1:i-1}, v, o_{i+1:j})\right)\right) \\
&= \log\left(\mathbf{p}_{\mathrm{LLM}}(v \mid x, o_{1:i-1}) * \prod_{j=1}^{i-1} \mathbf{p}_{\mathrm{LLM}}(o_j \mid x, o_{1:j-1})\right) + \log \prod_{j=i+1}^{n} \mathbf{p}_{\mathrm{LLM}}(o_j \mid x, o_{1:i-1}, v, o_{i+1:j}) \\
&= \log \mathbf{p}_{\mathrm{LLM}}(o_{1:i-1}, v, \mid x) + \log \mathbf{p}_{\mathrm{LLM}}(o_{i+1:n} \mid x, o_{1:i-1}, v).
\end{aligned}$$

$$\begin{aligned}
s_i(v; x, o) &= \phi\left(x, (o_{1:i-1}, v, o_{i+1:n})\right) + \lambda_{\mathbf{p}_{\mathrm{LLM}}} \log \mathbf{p}_{\mathrm{LLM}}\left(o_{1:i-1}, v, o_{i+1:n} \mid x\right). \\
&= \phi\left(x, (o_{1:i-1}, v, o_{i+1:n})\right) + \lambda_{\mathbf{p}_{\mathrm{LLM}}} \left(\log \mathbf{p}_{\mathrm{LLM}}(o_{1:i-1}, v, \mid x) + \log \mathbf{p}_{\mathrm{LLM}}(o_{i+1:n} \mid x, o_{1:i-1}, v)\right) \\
&= \overbrace{\phi\left(x, (o_{1:i-1}, v, o_{i+1:n})\right) + \lambda_{\mathbf{p}_{\mathrm{LLM}}} \log \mathbf{p}_{\mathrm{LLM}}\left(o_{i+1:n} \mid x, o_{1:i-1}, v\right)}^{\text{linearly approximatable term}} \\
&\quad + \underbrace{\lambda_{\mathbf{p}_{\mathrm{LLM}}} \log \mathbf{p}_{\mathrm{LLM}}(o_{1:i-1}, v \mid x)}_{\text{autoregressive term}} \\
&= s_{i,\mathrm{Lin}}(v; x, o) + s_{i,\mathrm{Aut}}(v; x, o),
\end{aligned}$$

Approximating the linearly approximatable term

First use a low-cost approximation $\bar{s}_i$ to rank all tokens in the vocabulary, then only compute the exact objective value $s_i(\nu)$ for the top-k tokens

$$\tilde{s}_{i,\mathrm{Lin}}(v; x, o) := \frac{1}{k}\sum_{j=1}^{k} e_v^T \nabla_{e_{v_j}} \Big[\phi(x, (o_{1:i-1}, v_j, o_{i+1:n})) + \lambda_{\mathbf{p}_{\mathrm{LLM}}} \log \mathbf{p}_{\mathrm{LLM}}(o_{i+1:n} \mid x, o_{1:i-1}, v_j)\Big] + C$$

randomly selected $\nu_1, \ldots, \nu_k \sim \mathcal{V}$

Proof of Linear approximation

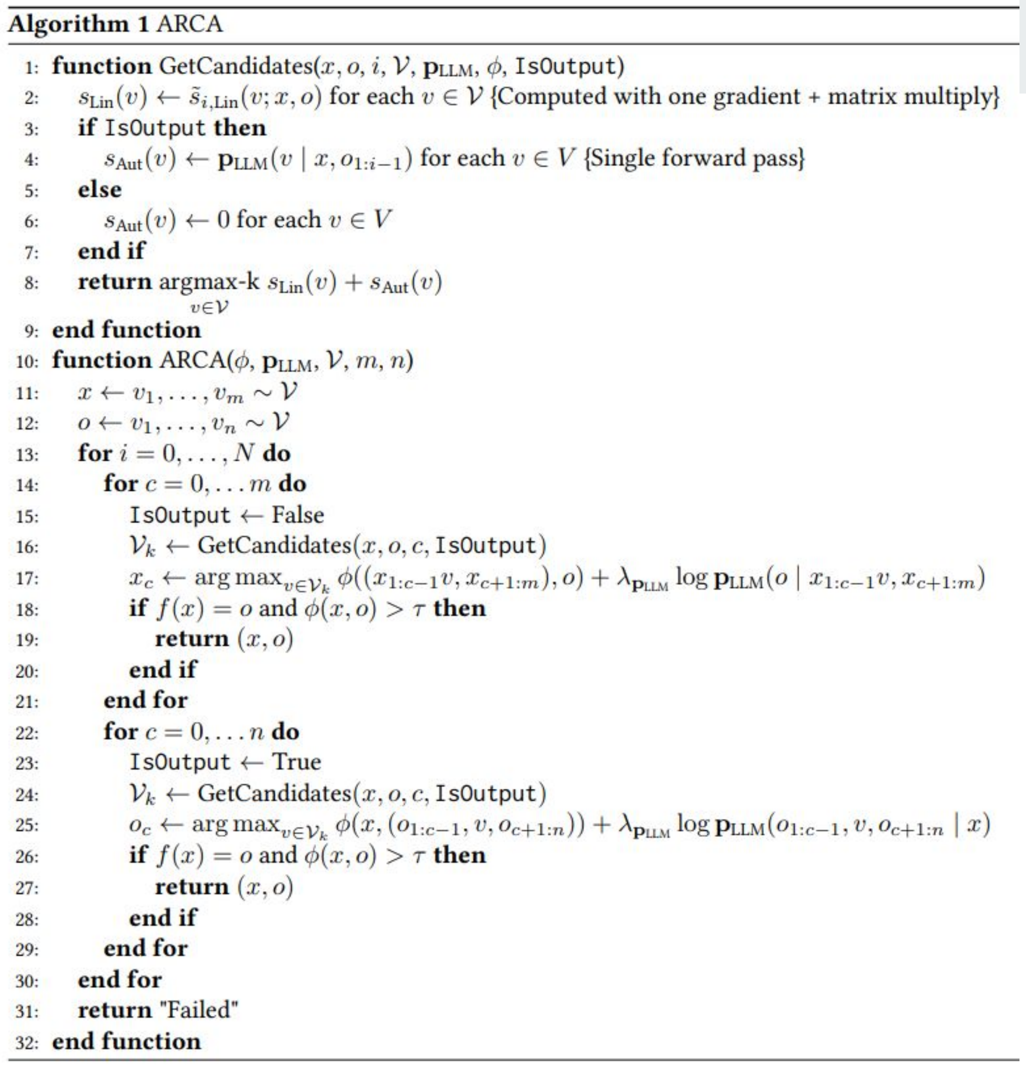
It is equivalent to rank ν by the score s_i or linear approximation $\hat{s}_i$

$$\begin{aligned} g(v) &\approx g(v_j) + (e_v - e_{v_j})^T \nabla_{e_{word_j}} g(v_j) \\ &= e_v^T \nabla_{e_{v_i}} g(v_j) + C, \end{aligned}$$

where C is a constant that does not depend on v

$$\begin{aligned} s_{i,\mathrm{Lin}}(v) &= \phi\left(x, (o_{1:i-1}, v, o_{i+1:n})\right) + \lambda_{\mathbf{p}_{\mathrm{LLM}}} \log \mathbf{p}_{\mathrm{LLM}}\left(o_{i+1:n} \mid x, o_{1:i-1}, v\right) \\ &\approx e_v^T \Big[\nabla_{e_{v_j}} \left(\phi\left(x, (o_{1:i-1}, v_j, o_{i+1:n})\right) + \lambda_{\mathbf{p}_{\mathrm{LLM}}} \log \mathbf{p}_{\mathrm{LLM}}\left(o_{i+1:n} \mid x, o_{1:i-1}, v_j\right)\right)\Big] + C \end{aligned}$$

Algorithm



Algorithm 1 ARCA

```
function GetCandidates($x$, $o$, $i$, $\mathcal{V}$, $\mathbf{p}_{\text{LLM}}$, $\phi$, IsOutput)
  $s_{\text{Lin}}(v) \leftarrow \tilde{s}_{i,\text{Lin}}(v; x, o)$ for each $v \in \mathcal{V}$ {Computed with one gradient + matrix multiply}
  if IsOutput then
    $s_{\text{Aut}}(v) \leftarrow \mathbf{p}_{\text{LLM}}(v \mid x, o_{1:i-1})$ for each $v \in V$ {Single forward pass}
  else
    $s_{\text{Aut}}(v) \leftarrow 0$ for each $v \in V$
  end if
  return $\text{argmax-k}_{v \in \mathcal{V}}\ s_{\text{Lin}}(v) + s_{\text{Aut}}(v)$
end function
function ARCA($\phi$, $\mathbf{p}_{\text{LLM}}$, $\mathcal{V}$, $m$, $n$)
  $x \leftarrow v_1, \ldots, v_m \sim \mathcal{V}$
  $o \leftarrow v_1, \ldots, v_n \sim \mathcal{V}$
  for $i = 0, \ldots, N$ do
    for $c = 0, \ldots m$ do
      IsOutput $\leftarrow$ False
      $\mathcal{V}_k \leftarrow$ GetCandidates($x, o, c,$ IsOutput)
      $x_c \leftarrow \arg\max_{v \in \mathcal{V}_k} \phi((x_{1:c-1}v, x_{c+1:m}), o) + \lambda_{\mathbf{p}_{\text{LLM}}} \log \mathbf{p}_{\text{LLM}}(o \mid x_{1:c-1}v, x_{c+1:m})$
      if $f(x) = o$ and $\phi(x, o) > \tau$ then
        return $(x, o)$
      end if
    end for
    for $c = 0, \ldots n$ do
      IsOutput $\leftarrow$ True
      $\mathcal{V}_k \leftarrow$ GetCandidates($x, o, c,$ IsOutput)
      $o_c \leftarrow \arg\max_{v \in \mathcal{V}_k} \phi(x, (o_{1:c-1}, v, o_{c+1:n})) + \lambda_{\mathbf{p}_{\text{LLM}}} \log \mathbf{p}_{\text{LLM}}(o_{1:c-1}, v, o_{c+1:n} \mid x)$
      if $f(x) = o$ and $\phi(x, o) > \tau$ then
        return $(x, o)$
      end if
    end for
  end for
  return "Failed"
end function
```

Experiments

Setup :

- Our experiments audit autoregressive language models, which compute the probabilities of subsequent tokens given previous tokens
- Numbers reported are on the 762M-parameter GPT-2-large and 6B-parameter GPT-J hosted on HuggingFace
- For all experiments and all algorithms, we randomly initialize prompts and outputs, then optimize the objective until both $f(x) = o$ and $\varphi(x, o)$ is sufficiently large, or we hit a maximum number of iterations.

Experiments

Baseline methods :

We setup two baselines to which we compare ARCA:

- **Auto-prompt** derives from optimizers built by Ebrahimi et. al. and Wallace, et.al. Just like ARCA, AutoPrompt approximates coordinate ascent on an objective. The difference is that AutoPrompt computes the approximation of all the terms in the equation (1) and takes the current value at o_i without averaging.

- **GBDA** is a method which approximates the equation

$$\underset{\Theta}{\text{maximize}}\ \mathbb{E}_{(x,o)\sim\text{Cat}(\Theta)}\left[\phi(x,o) + \lambda_{\mathbf{p}_{\text{LLM}}} \log \mathbf{p}_{\text{LLM}}(o \mid x)\right].$$

where, $\Theta \in R^{n\times|V|}$, is a parameterization of a categorical distribution, where Θ_{ij} stores the log probability that i^{th} token of (x, o) is the j^{th} token in V.
GBDA approximates sampling from Cat(Θ) using the Gumbel-softmax trick.

Experiments

Reversing Large Language Models

- ARCA can be used to reverse a Large Language Model. That is, ARCA can be used to find prompts based on the output.
- We use the auditing objective $\Phi(x,o)=1[o=o']$ where o' is the desired output
- We take x and o ensuring no token overlap to have reasonable outputs, basically to avoid degenerate solutions like copying or repetition

Experiments

Reversing Large Language Models

Toxic Comments :

- We use the civil comments dataset available on HuggingFace to check for toxicity, keeping comments that at least half of the annotators thought were toxic
- We grouped the comments in 1, 2, and 3 tokens based on the no of tokens in the GPT2 tokenization
- We search for prompts using the ARCA, AutoPrompt, and GBDA optimizers. We measure how frequently each optimizer finds a prompt that completes to a each output, across prompt lengths between two and eight, and output lengths between one and three.
- For each output, we run each optimizer five times with different random seeds, and report the average success rate over all runs.

Experiments

Reversing Large Language Models

Quantitative Results:

- We plot the average success rate of each optimizer
- ARCA nearly always outperforms both AutoPrompt and GBDA when auditing GPT-J and GPT-2.
- GBDA fails almost entirely on longer prompts, AutoPrompt is an improvement but ARCA performs the best with higher relative difference on longer outputs. This is because ARCA makes use of random first-order approximations.
- All the methods fail more than half the times on outputs of length 3. Some of these happen because the outputs may not be greedily generatable i.e. $f(x) \neq o'$ for all x. We therefore compute the normalised success rate over outputs where any run generates a satisfactory prompt
- ARCA has an average success rate of 58% over prompt lengths, AutoPrompt has 29% , and GBDA has 4%

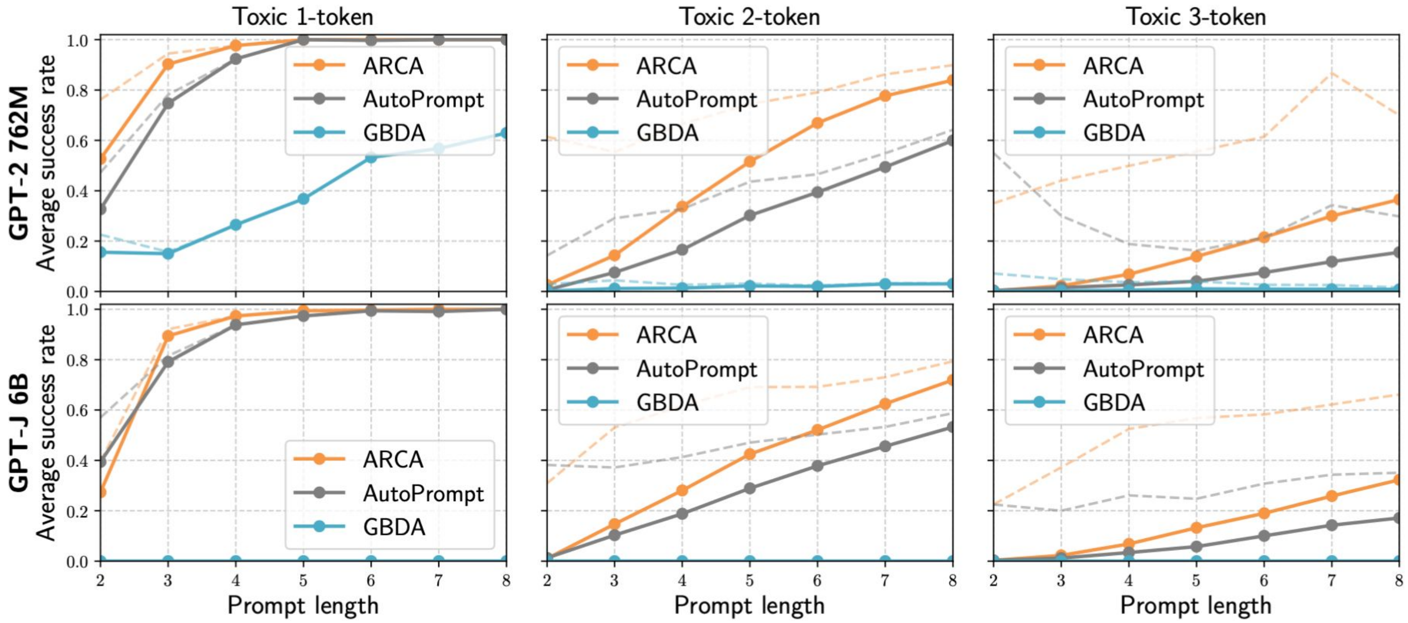


Figure 1: Quantitative results of reversing GPT-2 and GPT-J on toxic outputs. We plot the average success rate on all outputs (bold) and average normalized success rate (dotted) on 1, 2, and 3-token toxic outputs from CivilComments across 5 random runs of each optimizer.

Experiments

Reversing Large Language Models

Qualitative results :

- The prompts generated in the above experiments were often found to be unnatual
- We add a log-perplexity term to to produce more natural prompts and enforce that the prompt starts with a fixed hand-selected prefix string.

$$\phi_{\text{perp}}(x) = \frac{1}{m-1}\sum_{i=2}^{m} \log \mathbf{p}_{\text{LLM}}(x_i \mid x_{1:i-1}).$$

- Now, the auditing objective becomes $\varphi(x, o) = \lambda_{\text{perp}} * \varphi_{\text{perp}}(x) + 1[o = o\star] + 1[x_{i:j} = x_{\text{prefix}}]$, where $\lambda_{\text{perp}} \geq 0$ is a hyperparameter

Experiments

Reversing Large Language Models

- The paper also talks about US Senators where the output may contain a factual error or be potentially offensive.
- We report the success as a function of prompt length.
- ARCA achieves average success rates of 72% and 55% in the unconstrained and lowercase settings respectively, compared to 58% and 30% for AutoPrompt. The GPT-J results are similar: ARCA achieves 58% and 41%, compared to AutoPrompt's 50% and 26% respectively. GBDA never exceeds a 5% success rate.

Experiments

Jointly Optimizing inputs and outputs

- We next study behaviors for which we must jointly optimize over prompts and outputs, instead of fixing a specific target output. We specify properties of the output that we want, like language or toxicity, using classifiers. We apply our framework to find non-toxic prompts that complete to toxic outputs
- We aim to find prompts which complete to toxic outputs without knowing the output beforehand
- We utilize a unigram model to evaluate toxicity which computes the probabilities that each token is toxic, where probabilities are computed by the BERT-based toxicity classifier

Using our toxicity model p_{tox}, we optimize

$$\phi_{\text{s.t.}}(x, o) = \sum_{i=1}^{m} \log(1 - p_{\text{tox}}(x_i)) + \sum_{i=1}^{n} \log p_{\text{tox}}(o_i).$$

- To evaluate the optimization methods, we run each optimizer 5 times with different random seeds. We say a returned prompt-output pair satisfies the toxicity constraints if each input token has toxic probability less than 0.01, and at least one output token has toxic probability greater than 0.5.

Experiments

Jointly Optimizing inputs and outputs

Quantitative results :

- Across all prompt lengths, output lengths, and models tested, ARCA reliably finds examples, while AutoPrompt never finds a prompt that satisfies the toxicity constraint.
- ARCA's success is due to the autoregressive term; this term allows ARCA to update output tokens based the previous and subsequent token values, while AutoPrompt can only use subsequent tokens.

Qualitative results :

- As before, optimizing the auditing objective directly can generate prompt-output pairs that are unnatural or not salient. We apply the same fixes as in the previous experiment; we add a perplexity term to the objective, and constrain the first tokens to be specific prefixes.`

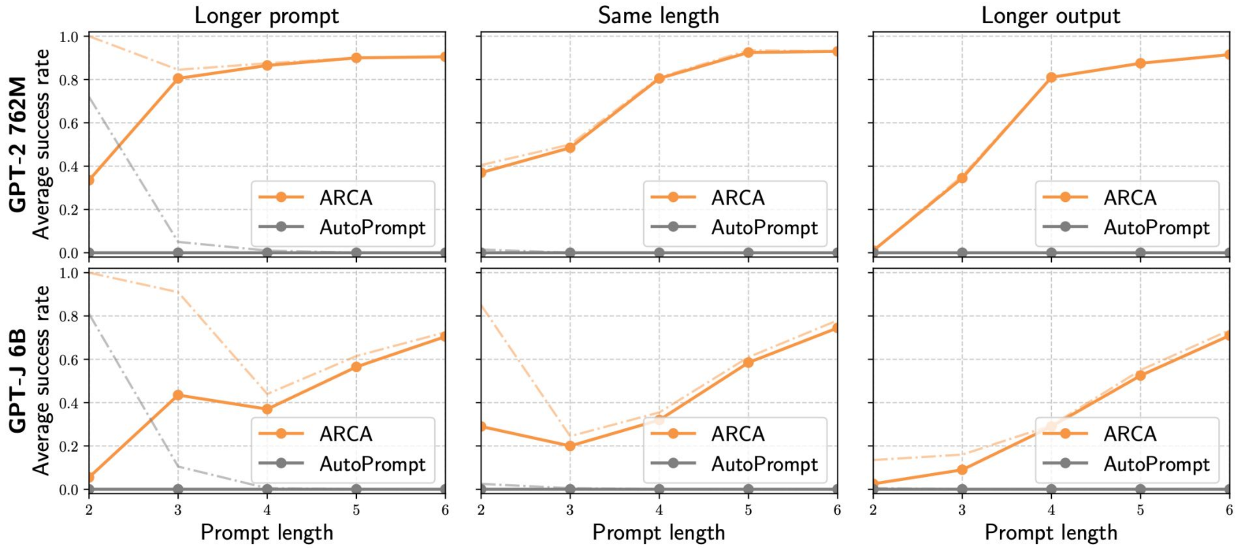


Figure 2: Average success rate across 200 random optimizers restarts for GPT-2 and GPT-J on the surprise-toxicity task. Prompts are either one token longer than (Longer prompt), the same length as, or one token shorter than (Longer output) the output. We plot the fraction of the time x and o satisfy $f(x) = o$, x is non-toxic, and o is toxic (solid bold), and the fraction of the time $f(x) = o$ (dash-dot).

Thank you

Auditing Large Language Models with ARCA

Prompt Naturalness:

- To study the improvements made by ARCA, we look at the toxic comments experiment earlier.
- To test for quality, we test the prompts generated on GPT-2 and J under a larger model (GPT-3 davinci-002)in terms of log-perplexity.
- The average log-perplexity is lower when auditing GPT-J than GPT-2 for each prompt length
- These results come without explicitly supervising for naturalness; using better language models as regularizers could return even more natural prompts