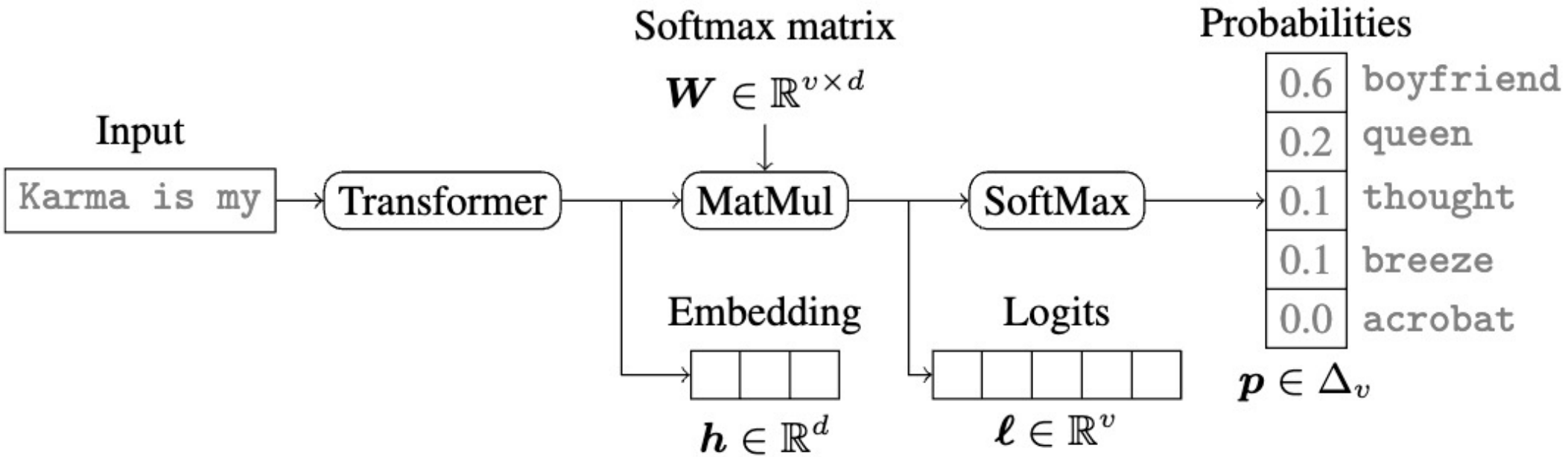
Logits of API-Protected LLMs Leak Proprietary Information

Logits of API-Protected LLMs Leak Proprietary Information

Matthew Finlayson **Xiang Ren** **Swabha Swayamdipta**
Thomas Lord Department of Computer Science
University of Southern California
{mfinlays, xiangren, swabhas}@usc.edu

arXiv: 2403.09539

Preliminary



Steal Logits

Table 2: A summary of our proposed algorithms, with estimates for the number of API calls required per output, and the price of acquiring the model image. Estimates are based on a `gpt-3.5-turbo`-like API LLM with $v = 100\text{K}$, $d = 4096$, $\epsilon = 10^{-6}$, $k = 5$, $b_{\max} = 100$, and $n = 4$. Note that the $O(d)$ algorithm cannot be used to obtain the LLM image, since it relies on having LLM image as a preprocessing step.

Algorithm	Complexity	API calls per output	Image price
Logprob-free	$O(v \log \frac{1}{\epsilon})$	800K	$16,384
With logprobs	v/k	20K	$410
Numerically stable	$v/(k-1)$	25K	$512
Stochastically robust	$nv/(k-2)$	133K	$2,724
LLM Image	$O(d)$	1K–32K	-

Base

logit_bias map Optional Defaults to null

Modify the likelihood of specified tokens appearing in the completion.

Accepts a JSON object that maps tokens (specified by their token ID in the tokenizer) to an associated bias value from -100 to 100. Mathematically, the bias is added to the logits generated by the model prior to sampling. The exact effect will vary per model, but values between -1 and 1 should decrease or increase likelihood of selection; values like -100 or 100 should result in a ban or exclusive selection of the relevant token.

$$\ell_i' = \begin{cases} \ell_i + b_{\max} & i \in \{1, 2, \ldots, k\} \\ \ell_i & \text{otherwise} \end{cases}$$

$$p_i = \frac{p_i'}{\exp b_{\max} - \exp b_{\max} \sum_{j=1}^{k} p_j' + \sum_{j=1}^{k} p_j'}$$

O(V/K)

Our goal is to prove

$$p_i = \frac{p_i'}{\exp b_{\max} - \exp b_{\max} \sum_{j=1}^{k} p_j' + \sum_{j=1}^{k} p_j'} \tag{6}$$

Proof. We begin with the definition of the softmax function

$$p_i' = \frac{\exp(\ell_i + b_{\max})}{\sum_{j=1}^{k} \exp(\ell_j + b_{\max}) + \sum_{j=k+1}^{v} \exp \ell_j} \tag{7}$$

We then rearrange to obtain

$$\sum_{j=k+1}^{v} \exp \ell_j = \frac{\exp(\ell_i + b_{\max})}{p_i'} - \sum_{j=1}^{k} \exp(\ell_j + b_{\max}), \tag{8}$$

the lefthand side of which is independent of the bias term, meaning it is equivalent to when $b_{\max} = 0$, i.e.,

$$\frac{\exp \ell_i}{p_i} - \sum_{j=1}^{k} \exp \ell_j = \frac{\exp(\ell_i + b_{\max})}{p_i'} - \sum_{j=1}^{k} \exp(\ell_j + b_{\max}). \tag{9}$$

We can now rearrange

$$\frac{\exp \ell_i}{p_i} = \frac{\exp(\ell_i + b_{\max})}{p_i'} - \sum_{j=1}^{k} \exp(\ell_j + b_{\max}) + \sum_{j=1}^{k} \exp \ell_j \tag{10}$$

$$p_i = \frac{\exp \ell_i}{\frac{\exp(\ell_i + b_{\max})}{p_i'} - \sum_{j=1}^{k} \exp(\ell_j + b_{\max}) + \sum_{j=1}^{k} \exp \ell_j} \tag{11}$$

and expand

$$p_i = \frac{p_i' \exp \ell_i}{\exp(\ell_i + b_{\max}) - p_i' \sum_{j=1}^{k} \exp(\ell_j + b_{\max}) + p_i' \sum_{j=1}^{k} \exp \ell_j} \tag{12}$$

$$= \frac{p_i'^2 \exp(-b_{\max}) \exp(\ell_i + b_{\max})}{\exp(\ell_i + b_{\max}) - p_i' \sum_{j=1}^{k} \exp(\ell_j + b_{\max}) + p_i' \exp(-b_{\max}) \sum_{j=1}^{k} \exp(\ell_j + b_{\max})} \tag{13}$$

and finally simplify by multiplying the top and bottom of the right hand side by

$$\frac{1}{\sum_{j=1}^{k} \exp(\ell_j + b_{\max}) + \sum_{j=k+1}^{v} \exp \ell_j}$$

and which converts each term of the form $\exp(\ell_i + b_{\max})$ to p_i', resulting in

$$\begin{aligned} p_i &= \frac{p_i'^2 \exp(-b_{\max})}{p_i' - p_i' \sum_{j=1}^{k} p_j' + p_i' \exp(-b_{\max}) \sum_{j=1}^{k} p_j'} \\ &= \frac{p_i' \exp(-b_{\max})}{1 - \sum_{j=1}^{k} p_j' + \exp(-b_{\max}) \sum_{j=1}^{k} p_j'} \\ &= \frac{p_i'}{\exp b_{\max} - \exp b_{\max} \sum_{j=1}^{k} p_j' + \sum_{j=1}^{k} p_j'}, \end{aligned}$$

which concludes the proof.

Numeric Stable

$$p_i = \frac{p'_i}{\exp b_{\max} - \exp b_{\max} \sum_{j=1}^{k} p'_j + \sum_{j=1}^{k} p'_j}$$

$$p_i = \exp(\log p'_i - b_{\max} - \log p'_v + \log p_v)$$

O(V/(K-1))

The next proof is much simpler. Our goal is to prove

$$p_i = \exp(\log p'_i - b_{\max} - \log p'_v + \log p_v).$$

Proof. We begin with four facts

$$p_i = \frac{\exp \ell_i}{\sum_{j=1}^{v} \exp \ell_j} \quad p'_i = \frac{\exp \ell'_i}{\sum_{j=1}^{v} \exp \ell'_j} \quad p_v = \frac{\exp \ell_v}{\sum_{j=1}^{v} \exp \ell_j} \quad p'_v = \frac{\exp \ell_v}{\sum_{j=1}^{v} \exp \ell'_j}$$

which follow from the definition of softmax. Combining these, we have

$$\frac{p_i}{\exp \ell_i} = \frac{p_v}{\exp \ell_v} \quad \text{and} \quad \frac{p'_i}{\exp \ell'_i} = \frac{p'_v}{\exp \ell_v}.$$

Combining these agian, we get

$$\frac{p_i}{p_v \exp \ell_i} = \frac{p'_i}{p'_v \exp \ell'_i},$$

which we can rearrange to obtain

$$\frac{p'_v p_i}{p'_i p_v} = \frac{\exp \ell_i}{\exp \ell'_i}.$$

Next, we use the fact that $\exp \ell'_i = \exp b_{\max} \exp \ell$ to get

$$\frac{p'_v p_i}{p'_i p_v} = \exp(-b_{\max}).$$

which we can take the log of, rearrange, and exponentiate to achieve our goal

$$p_i = \exp(\log p'_i - b_{\max} - \log p'_v + \log p_v).$$

Stochastic API

$$\log p_v^{(i)} - \log p_{v-1}^{(i)} = \log p_v'^{(i)} - \log p_{v-1}'^{(i)}$$

We would like to derive

$$\log p_v - \log p_{v-1} = \log p'_v - \log p'_{v-1}. \tag{24}$$

Proof. Using our result from Appendix A.2, we have

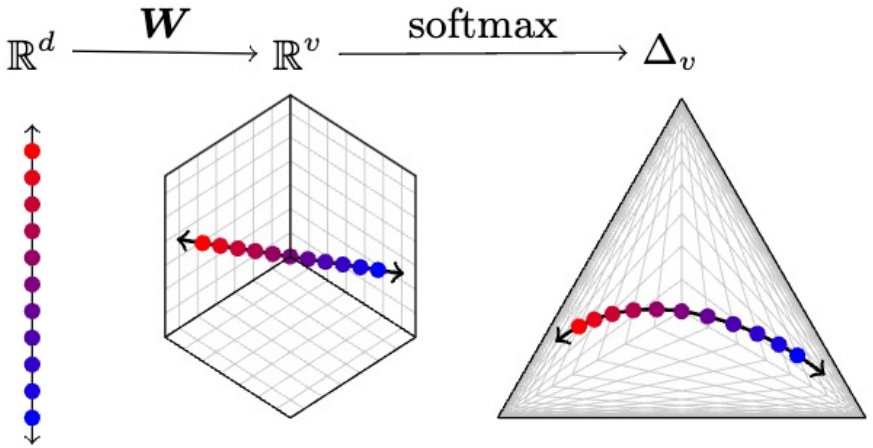
$$p_i = \exp(\log p'_i - b_{\max} - \log p'_v + \log p_v) \tag{25}$$

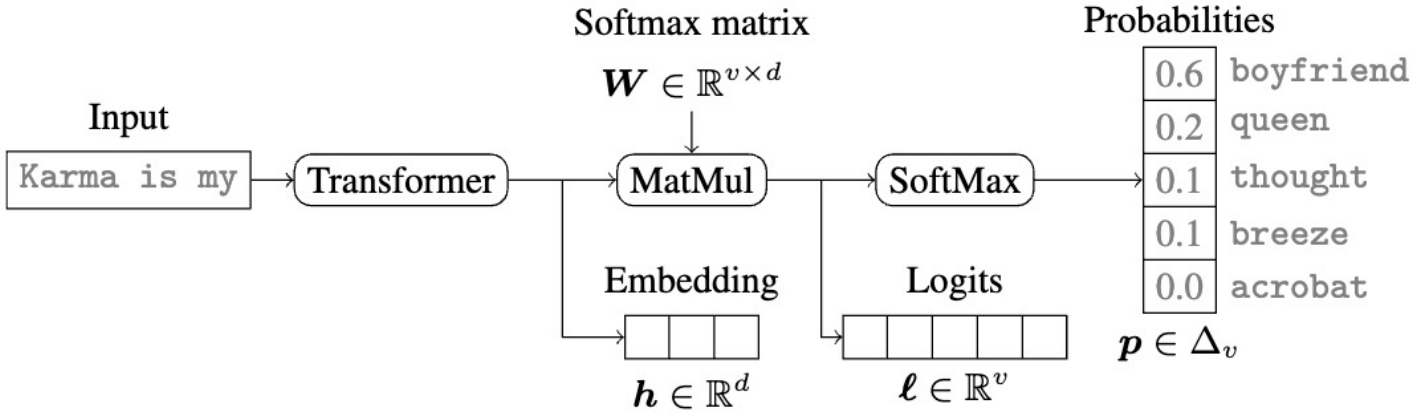
$$p_i = \exp(\log p'_i - b_{\max} - \log p'_{v-1} + \log p_{v-1}). \tag{26}$$

Simply setting the righthand sides equal to one another, taking the log of both sides, then subtracting identical terms from both sides gives us our goal. □

O(nV/(K-2))

Model Image


$\mathbb{R}^d$
$\boldsymbol{W}$
$\mathbb{R}^v$
softmax
Δ_v


Softmax matrix
$\boldsymbol{W} \in \mathbb{R}^{v \times d}$
Input
Karma is my
Transformer
MatMul
SoftMax
Probabilities
0.6 boyfriend
0.2 queen
0.1 thought
0.1 breeze
0.0 acrobat
$\boldsymbol{p} \in \Delta_v$
Embedding
$\boldsymbol{h} \in \mathbb{R}^d$
Logits
$\boldsymbol{\ell} \in \mathbb{R}^v$

Ultra Fast

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{p}^1 & \boldsymbol{p}^2 & \cdots & \boldsymbol{p}^d \end{bmatrix} \in \Delta_v^d$$

O(d/K)

$$\operatorname{alr}(\boldsymbol{p}) = \left(\log \frac{p_2}{p_1}, \log \frac{p_3}{p_1}, \ldots, \log \frac{p_v}{p_1} \right)$$

to transform the columns of $\boldsymbol{P}$ and $\boldsymbol{p}$ into vectors in $\mathbb{R}^{v-1}$, though since we only know the first d values of $\boldsymbol{p}$, we can only obtain the first d values of $\operatorname{alr}(\boldsymbol{p})$. Because the alr transform is an isomorphism, we have that the columns of

$$\operatorname{alr}(\boldsymbol{P}) = \begin{bmatrix} \operatorname{alr}(\boldsymbol{p}^1) & \operatorname{alr}(\boldsymbol{p}^2) & \cdots & \operatorname{alr}(\boldsymbol{p}^1) \end{bmatrix} \in \mathbb{R}^{(v-1)\times d}$$

form a basis for a d-dimensional vector subspace of $\mathbb{R}^{v-1}$, and $\operatorname{alr}(\boldsymbol{p})$ lies within this subspace. Therefore, there is some $\boldsymbol{x} \in \mathbb{R}^d$ such that $\operatorname{alr}(\boldsymbol{P})\boldsymbol{x} = \operatorname{alr}(\boldsymbol{p})$. To solve for $\boldsymbol{x}$, all that is required is to find the unique solution to the first d rows of this system of linear equations

$$\begin{bmatrix} \operatorname{alr}(p^1)_1 & \operatorname{alr}(p^2)_1 & \cdots & \operatorname{alr}(p^d)_1 \\ \operatorname{alr}(p^1)_2 & \operatorname{alr}(p^2)_2 & \cdots & \operatorname{alr}(p^d)_2 \\ \vdots & \vdots & \ddots & \vdots \\ \operatorname{alr}(p^1)_d & \operatorname{alr}(p^2)_d & \cdots & \operatorname{alr}(p^d)_d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} \operatorname{alr}(p)_1 \\ \operatorname{alr}(p)_2 \\ \vdots \\ \operatorname{alr}(p)_d \end{bmatrix}. \tag{5}$$

After finding $\boldsymbol{x}$, we can reconstruct the full LLM output $\boldsymbol{p} = \operatorname{alr}^{-1}(\operatorname{alr}(\boldsymbol{P})\boldsymbol{x})$, where the inverse alr function is defined as

$$\operatorname{alr}^{-1}(\boldsymbol{x}) = \frac{1}{1 + \sum_{i=1}^{v-1} \exp x_i} \cdot (1, \exp x_1, \exp x_2, \ldots, \exp x_{v-1}).$$

Embedding Size

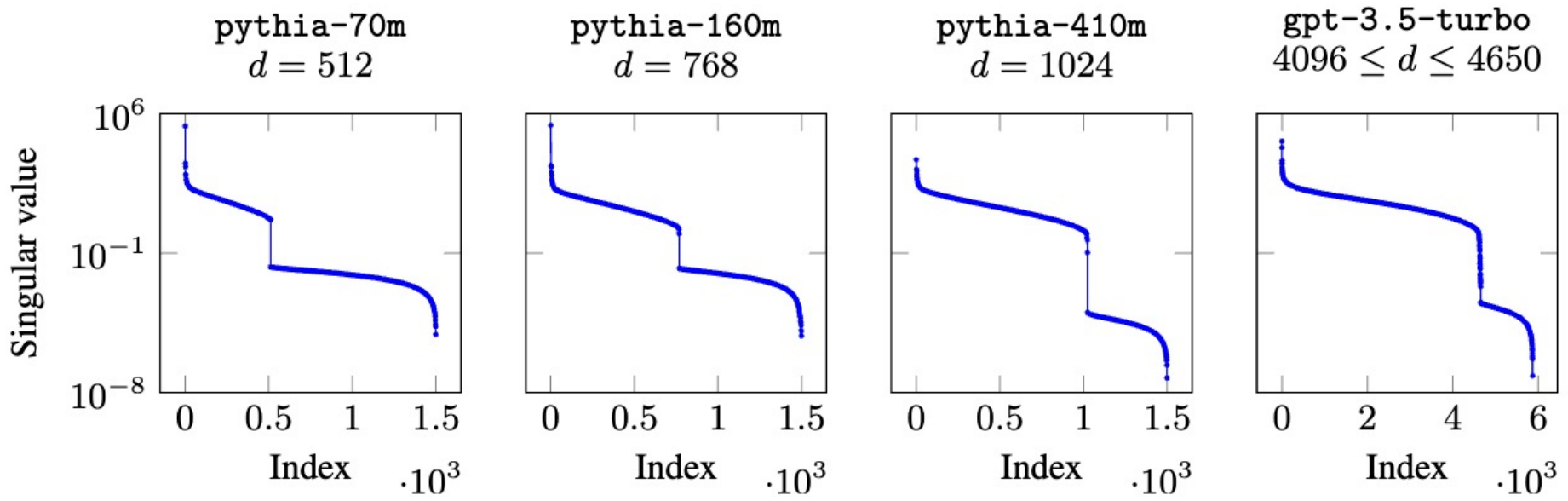


Figure 4: The singular values of outputs from LLMs with various known and unknown embedding sizes d. For each model with known embedding size, there is a clear drop in magnitude at singular value index d, indicating the embedding size of the model. Using this observation, we can guess the embedding size of `gpt-3.5-turbo`.

Model Signature

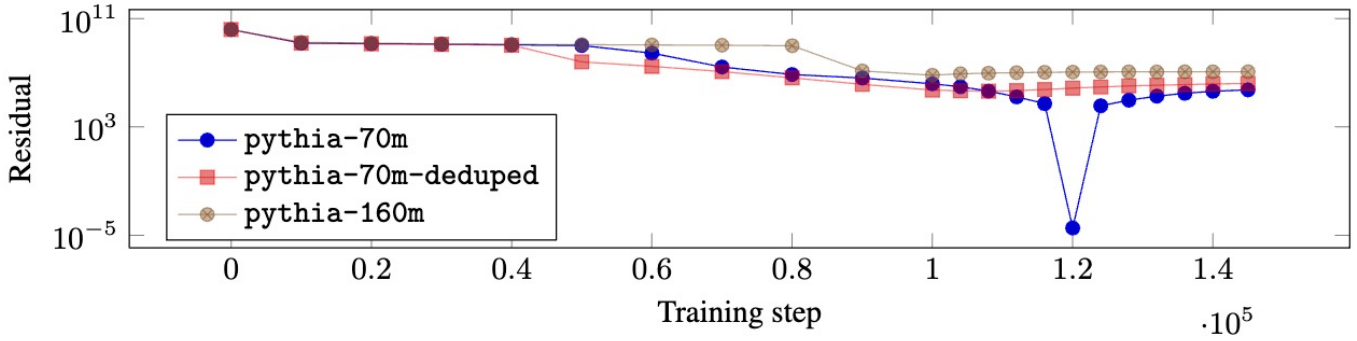


Figure 5: Residuals of the least-squares solution of $\boldsymbol{W}\boldsymbol{x} = \boldsymbol{\ell}$ for an output $\boldsymbol{\ell}$ from a `pythia-70m` checkpoint (training step 120K), and softmax matrices $\boldsymbol{W}$ at various training steps for `pythia-70m`, and `pythia-70m-deduped`, and `pythia-160m`. Low/high residual values indicate that the output $\boldsymbol{\ell}$ is/is not in a model's image. Residuals for `pythia-70m` decrease as the checkpoints near the checkpoint that generated the output, but remain high. Note that this test works even if the softmax matrices $\boldsymbol{W}$ are substituted with model outputs $\boldsymbol{L}$.

- Detecting LLM Updates
- Detecting Lora Weights

it happens to lie on the low-dimensional ($< d$) intersection of the models' images. Mathematically the dimension of the models' images' intersection is small since the intersection $U \cap V$ of two subspaces U and V that are not subsets of one another has dimension $\dim U + \dim V - \dim(U \cup V)$, which implies that $\dim(U \cap V) < \min\{\dim U, \dim V\}$ (since $\dim(U \cup V) > \max\{\dim U, \dim V\}$). Thus, it is possible to determine precisely which LLM produced a particular output, using only API access to a set of LLMs and without knowing the exact inputs to the model. In this way, the model's image acts as a *signature* for the model, i.e., a unique model identifier.

Unargmaxable tokens

[2, 1, -1, 1]
[2, -1, 1, 1]

Due to the low-rank constraints on LLM outputs, it is possible that some tokens become *unargmax-able* [Demeter et al., 2020, Grivas et al., 2022], i.e., there is a token i such that the constraints disallow any output $\boldsymbol{p}$ with $\arg\max_i p_i = i$. This happens when the LLM's embedding representation of i lies within the convex hull of the other tokens' embeddings. Previously, finding unargmaxable tokens appeared to require full access to the softmax matrix $\boldsymbol{W}$. Interestingly, we find that it is possible to identify unargmaxable tokens using only the LLM's image, which our method is able to recover. This technique allows API customers to find quirks in LLM behavior such as tokens that the model is unable to output (at least under greedy decoding).

Stealing Part of a Production Language Model

Nicholas Carlini [1] **Daniel Paleka** [2] **Krishnamurthy (Dj) Dvijotham** [1] **Thomas Steinke** [1] **Jonathan Hayase** [3]
A. Feder Cooper [1] **Katherine Lee** [1] **Matthew Jagielski** [1] **Milad Nasr** [1] **Arthur Conmy** [1] **Eric Wallace** [4]
David Rolnick [5] **Florian Tramèr** [2]

9 Simultaneous discovery

In a remarkable case of simultaneous discovery, Carlini et al. [2024] propose a very similar approach to ours for gaining insight into API-protected LLMs. Here we review a some interesting interactions between our work and theirs. First, we give an algorithm for obtaining full outputs in v/k API calls, while their algorithm corresponds to our $v/(k-1)$ algorithm. This has little impact on our final result, since our v/k algorithm suffers from numerical instability, but it is an interesting theoretical result nonetheless. On the flip side, Carlini et al. propose an improved logprob-free method for getting full outputs from an API that takes advantage of parallel queries. This useful method is actually complementary to our $O(d)$ algorithm, since they can be combined to yield an even better algorithm for obtaining full outputs.

arXiv: 2403.06634

THANK YOU