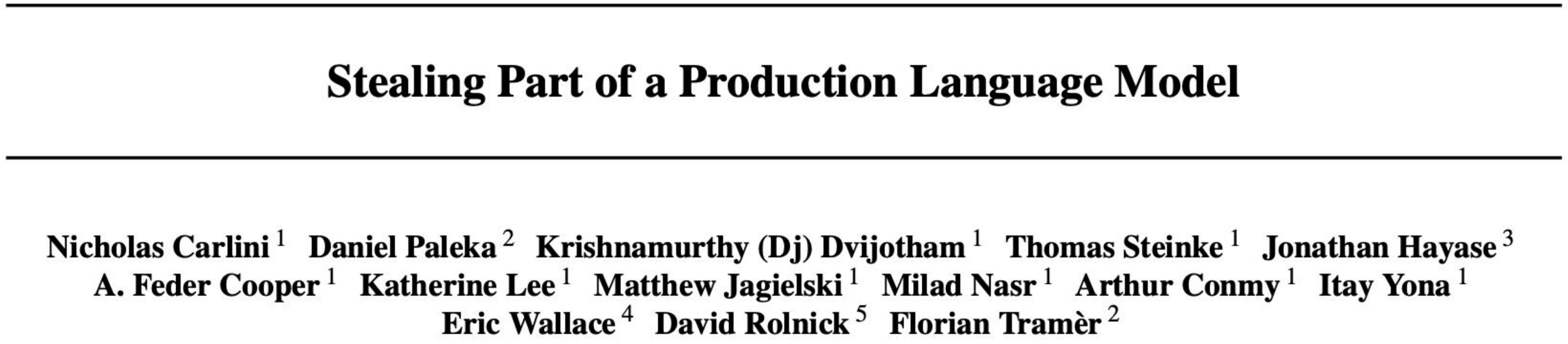
Stealing Part of a Production Language Model

Mahler Revsine and Tianwei Zhao



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] Itay Yona[1]
Eric Wallace[4] David Rolnick[5] Florian Tramèr[2]

This isn't a paper……

…… it's a heist!

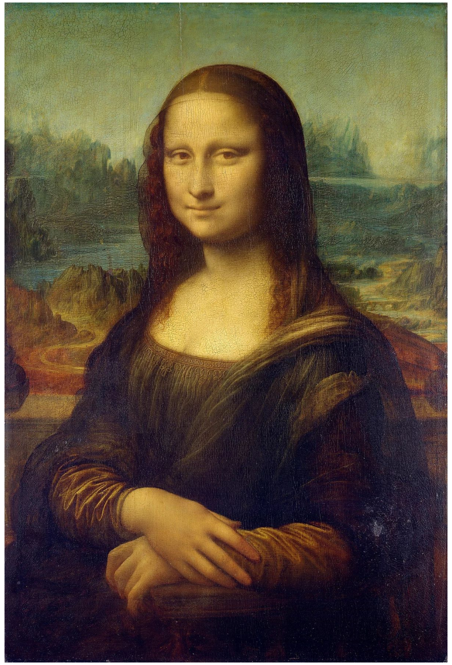
$100,000,000

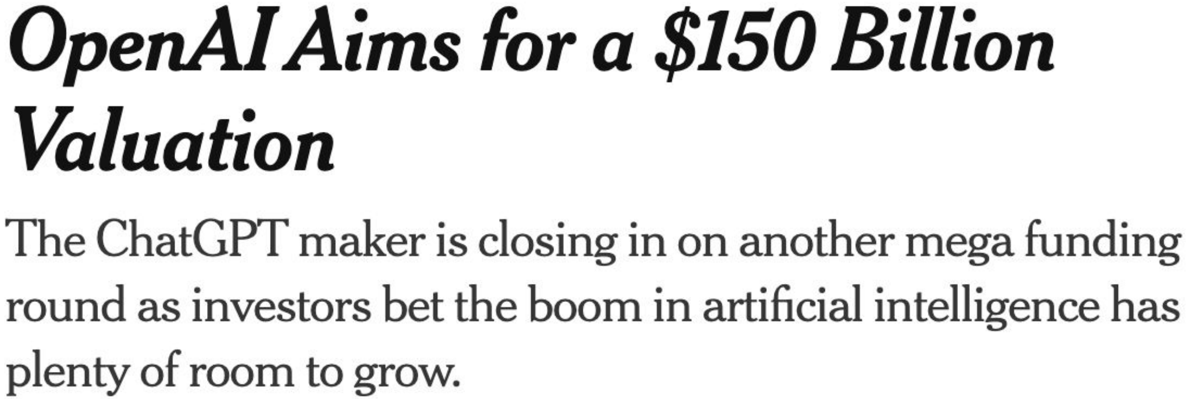
$100,000

We aren't robbing a bank…

…or stealing a painting…

… this is a heist of an **LLM**!

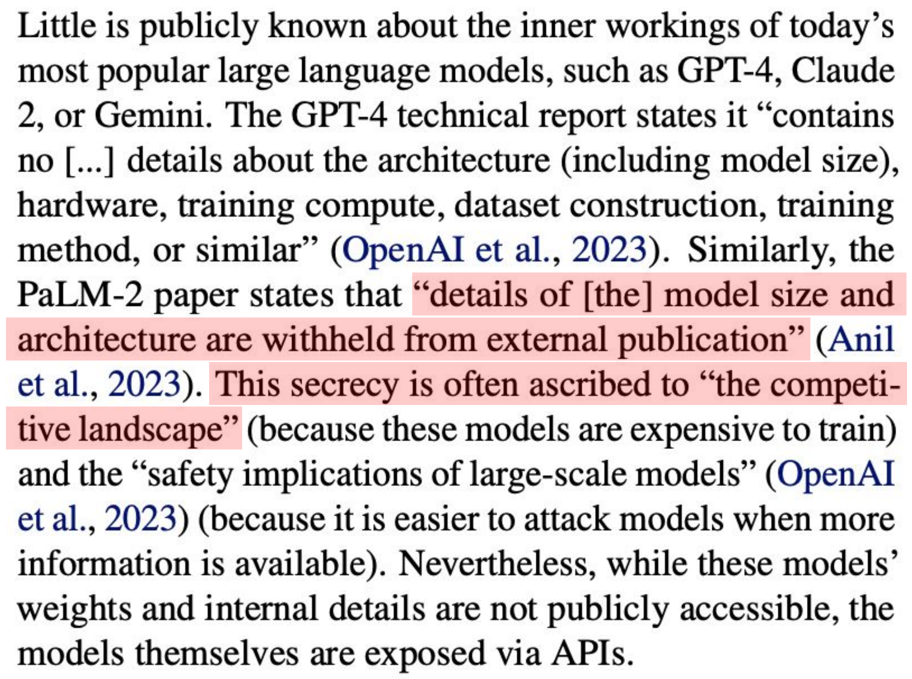


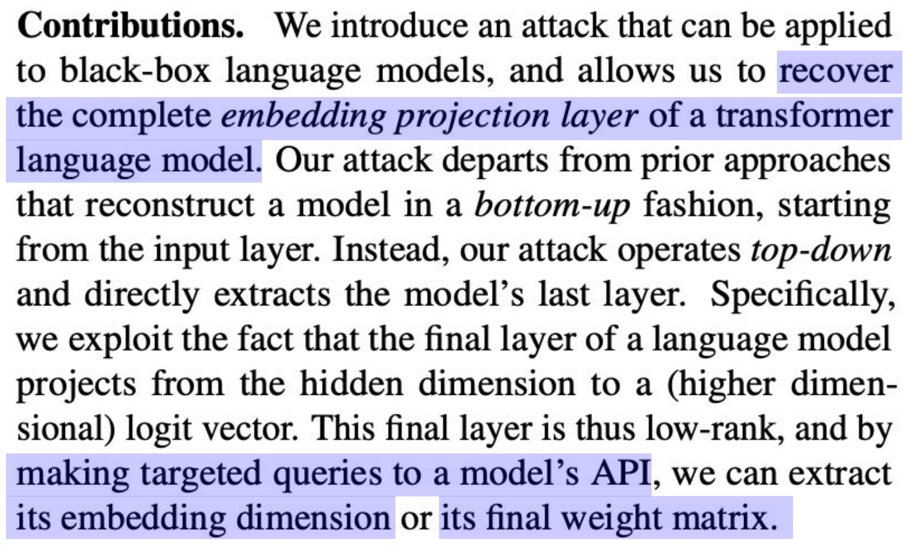


OpenAI Aims for a $150 Billion Valuation

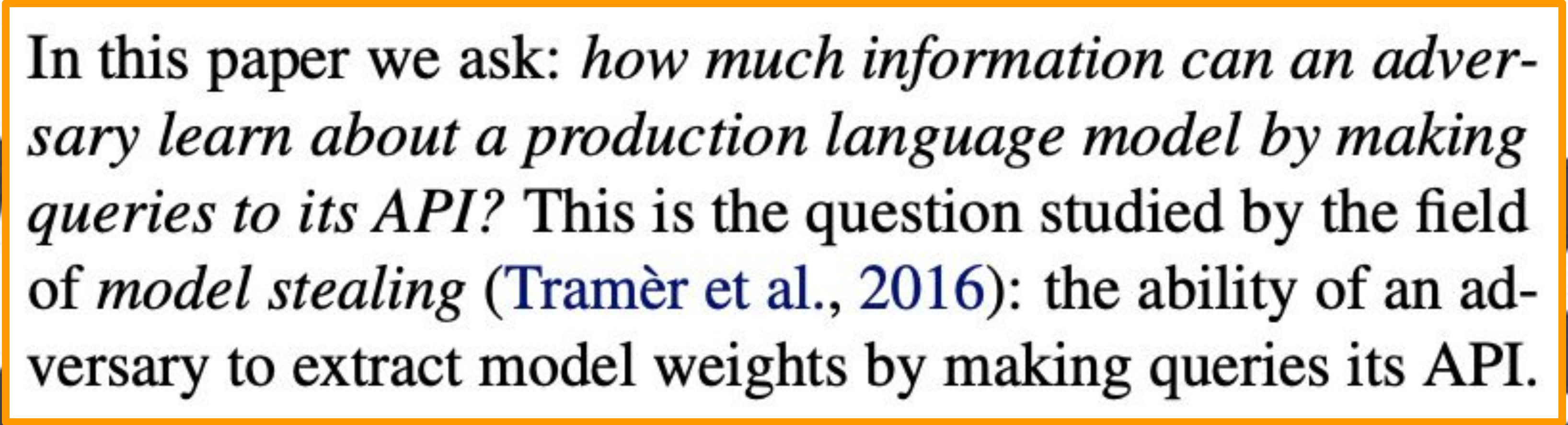
The ChatGPT maker is closing in on another mega funding round as investors bet the boom in artificial intelligence has plenty of room to grow.

Security is tight


Little is publicly known about the inner workings of today's most popular large language models, such as GPT-4, Claude 2, or Gemini. The GPT-4 technical report states it "contains no [...] details about the architecture (including model size), hardware, training compute, dataset construction, training method, or similar" (OpenAI et al., 2023). Similarly, the PaLM-2 paper states that "details of [the] model size and architecture are withheld from external publication" (Anil et al., 2023). This secrecy is often ascribed to "the competitive landscape" (because these models are expensive to train) and the "safety implications of large-scale models" (OpenAI et al., 2023) (because it is easier to attack models when more information is available). Nevertheless, while these models' weights and internal details are not publicly accessible, the models themselves are exposed via APIs.


Contributions. We introduce an attack that can be applied to black-box language models, and allows us to recover the complete *embedding projection layer* of a transformer language model. Our attack departs from prior approaches that reconstruct a model in a *bottom-up* fashion, starting from the input layer. Instead, our attack operates *top-down* and directly extracts the model's last layer. Specifically, we exploit the fact that the final layer of a language model projects from the hidden dimension to a (higher dimensional) logit vector. This final layer is thus low-rank, and by making targeted queries to a model's API, we can extract its embedding dimension or its final weight matrix.

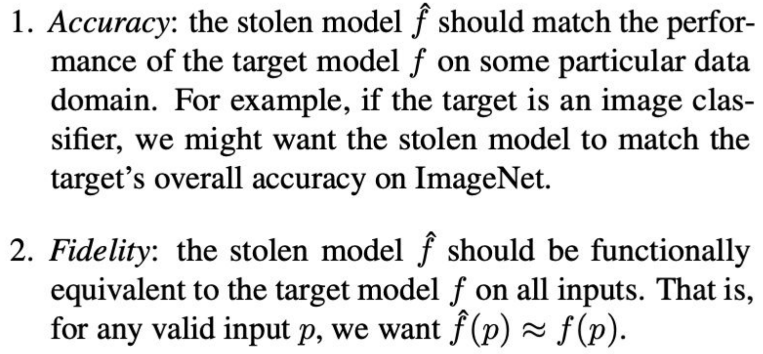
But we have a plan


In this paper we ask: *how much information can an adversary learn about a production language model by making queries to its API?* This is the question studied by the field of *model stealing* (Tramèr et al., 2016): the ability of an adversary to extract model weights by making queries its API.

Carlini, Nicholas, et al. "Stealing part of a production language model." *arXiv preprint arXiv:2403.06634* (2024).

Our approach: model stealing attacks

- Aim to **recover functionality** of a **black-box model**
- Optimize for **accuracy** or **fidelity**
- Early work computed **gradients** of **ReLU-based** neural networks to recreate model weights
- Production quality LLMs are too **big**, **complex**, and **secure**
- Recent work demonstrates stealing **final layer** of models with public pretrained encoder layer



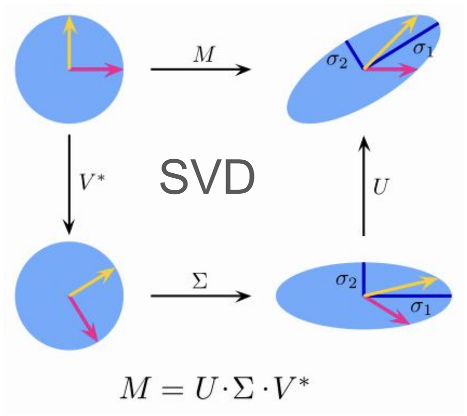
1. *Accuracy*: the stolen model $\hat{f}$ should match the performance of the target model f on some particular data domain. For example, if the target is an image classifier, we might want the stolen model to match the target's overall accuracy on ImageNet.

2. *Fidelity*: the stolen model $\hat{f}$ should be functionally equivalent to the target model f on all inputs. That is, for any valid input p, we want $\hat{f}(p) \approx f(p)$.

Meet the team

Nicholas Carlini
Research Scientist, Google DeepMind

The mastermind

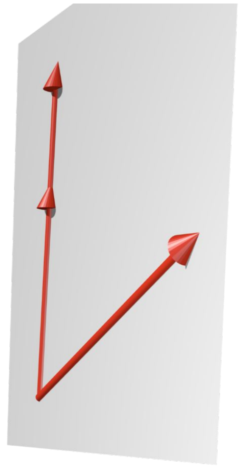


The hacker

$$rank\left(\begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 0 \\ 3 & 0 & 4 \end{bmatrix}\right)$$

The secret agent

Matrix rank



Number of l**inearly independent rows** or **columns**

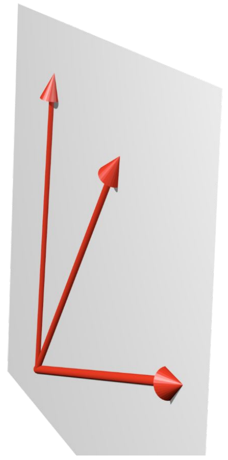
Dimension of the **vector space** of its rows or columns

Rank can be **less** than the matrix **dimension**

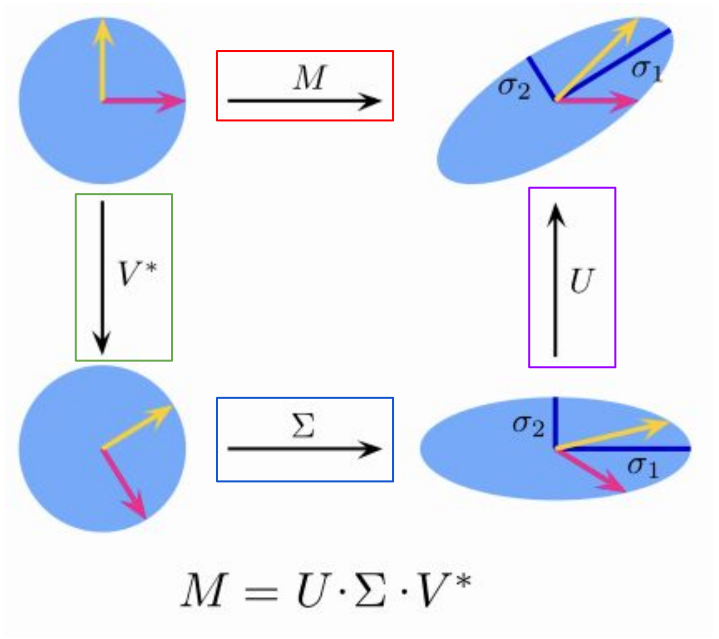
$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Rank = 2

Column 3 == column 1 + column 2
Row 2 == row 3



Singular Value Decomposition (SVD)



Matrices can be thought of as **transformations**

All matrix transformations are made of 3 components:

1. a **rotation**
2. a **stretching**
3. another **rotation**

SVD **breaks down** a matrix into these **3 components**

SVD formula: **M = U Σ V***

M is any *m x n* matrix

U is an *m x m* matrix

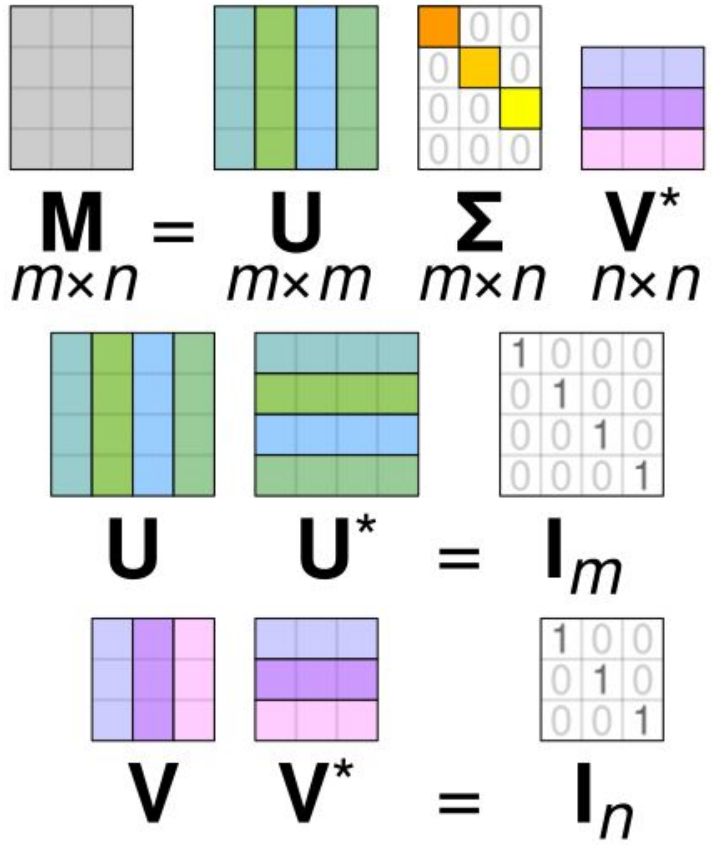
- Rotation in *m*-dimensional space

Σ is an *m x n* matrix

- Projection from *m*- to *n*-dim. space

V* is an *n x n* matrix

- Rotation in *n*-dimensional space



$\mathbf{M}_{m\times n} = \mathbf{U}_{m\times m}\,\mathbf{\Sigma}_{m\times n}\,\mathbf{V}^*_{n\times n}$
$\mathbf{U}\,\mathbf{U}^* = \mathbf{I}_m$
$\mathbf{V}\,\mathbf{V}^* = \mathbf{I}_n$

SVD formula: **M = U Σ V***

M is any *m x n* matrix

U is an *m x m* matrix

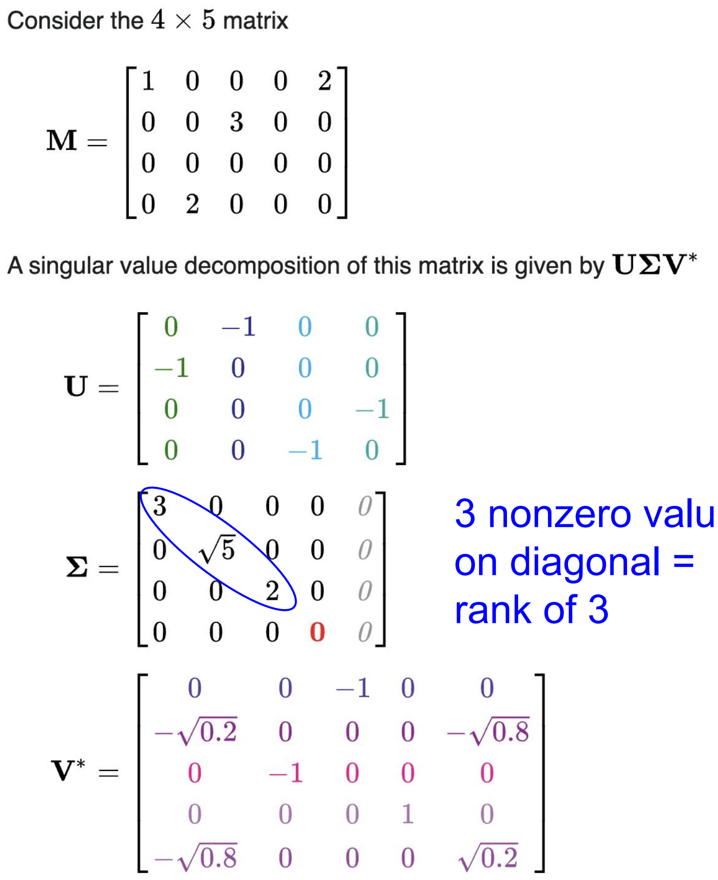
- Rotation in *m*-dimensional space

Σ is an *m x n* matrix

- **Projection from *m*- to *n*-dim. space**

V* is an *n x n* matrix

- Rotation in *n*-dimensional space



Consider the 4×5 matrix

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \end{bmatrix}$$

A singular value decomposition of this matrix is given by $\mathbf{U\Sigma V^*}$

$$\mathbf{U} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{5} & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{V}^* = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ -\sqrt{0.2} & 0 & 0 & 0 & -\sqrt{0.8} \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2} \end{bmatrix}$$

3 nonzero values on diagonal = rank of 3

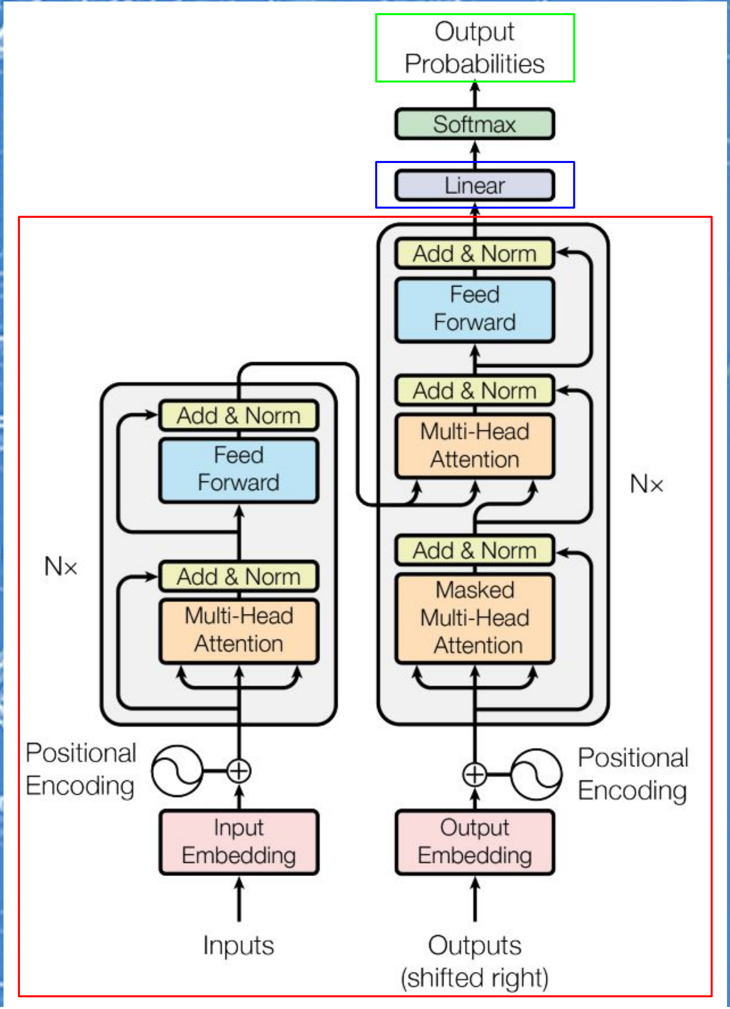
The floor plan

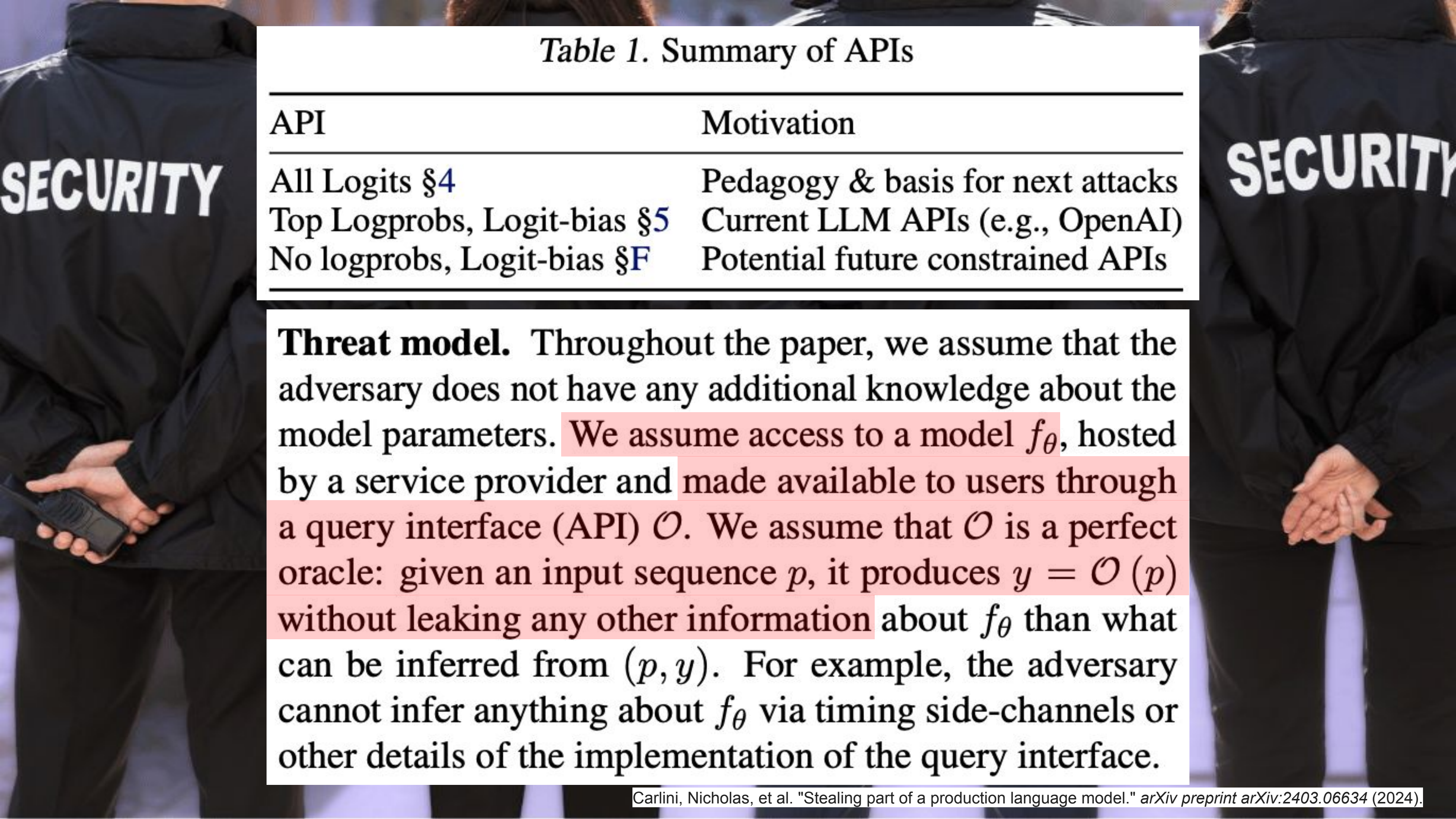
We study models that take a sequence of tokens drawn from a vocabulary $\mathcal{X}$ as input. Let $\mathcal{P}(\mathcal{X})$ denote the space of probability distributions over $\mathcal{X}$. We study parameterized models $f_\theta : \mathcal{X}^N \rightarrow \mathcal{P}(\mathcal{X})$ that produce a probability distribution over the next output token, given an input sequence of N tokens. The model has the following structure:

$$\underline{f_\theta(p)} = \text{softmax}(\underline{\mathbf{W}} \cdot \underline{g_\theta(p)}), \qquad (1)$$

$g_\theta(p)$ produces an *h*-dimensional output vector
- *h* is the hidden dimension
- LLaMA uses hidden dimension between 4096 and 8192

W converts this into an l-dimensional probability vector
- 1 element per token in the vocabulary
- GPT-4 has a 100,000 token vocab
- **W** is an l x *h* “embedding projection matrix”





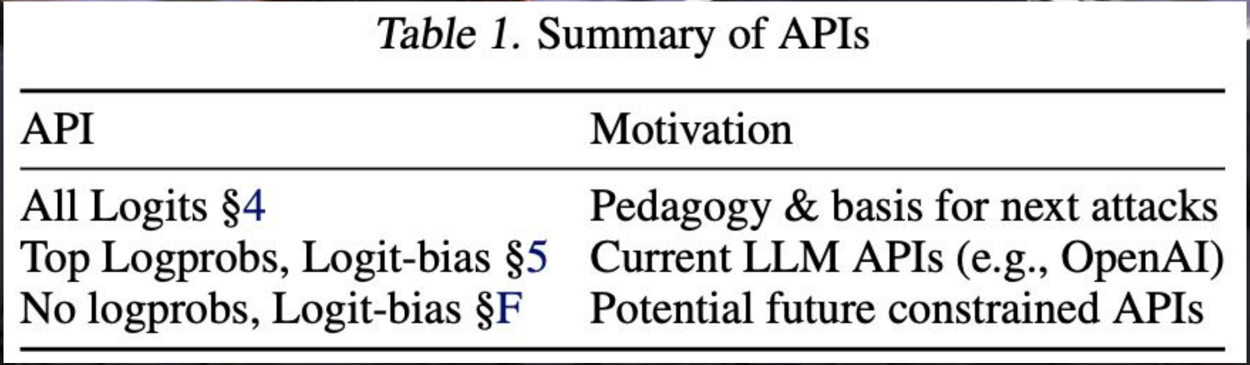
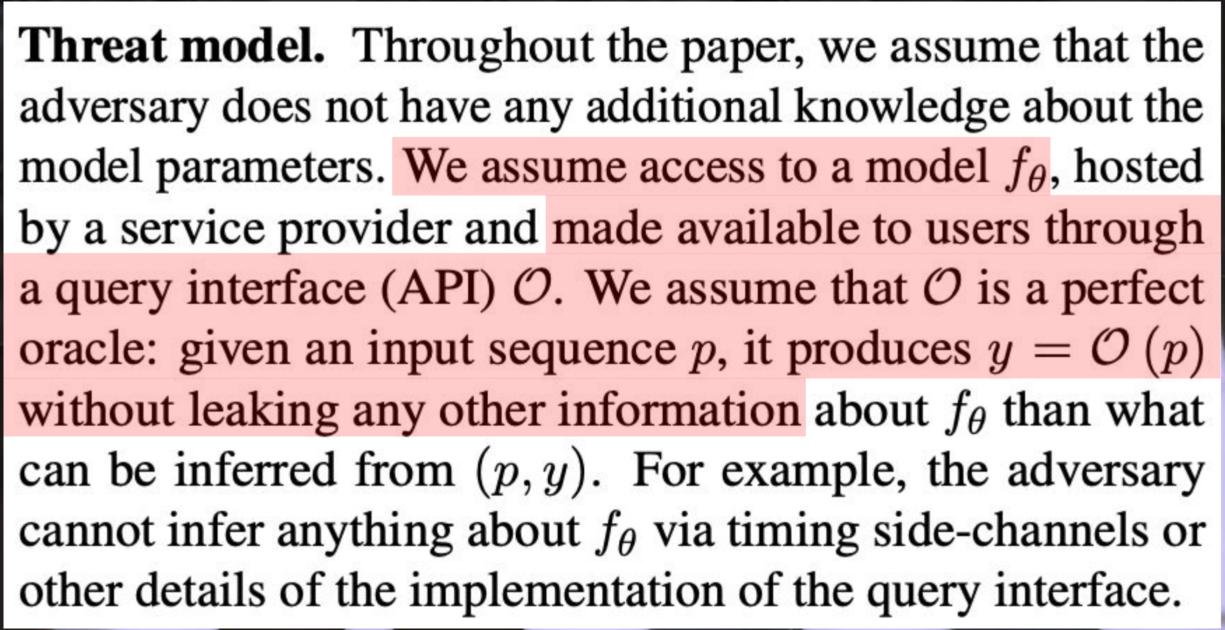


Table 1. Summary of APIs

API	Motivation
All Logits §4	Pedagogy & basis for next attacks
Top Logprobs, Logit-bias §5	Current LLM APIs (e.g., OpenAI)
No logprobs, Logit-bias §F	Potential future constrained APIs

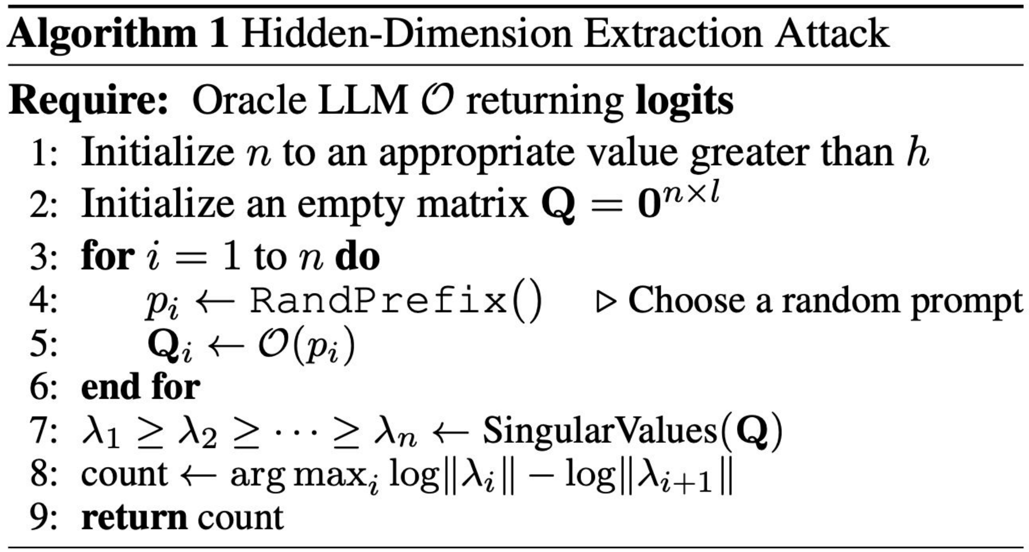

Threat model. Throughout the paper, we assume that the adversary does not have any additional knowledge about the model parameters. We assume access to a model f_θ, hosted by a service provider and made available to users through a query interface (API) $\mathcal{O}$. We assume that $\mathcal{O}$ is a perfect oracle: given an input sequence p, it produces $y = \mathcal{O}(p)$ without leaking any other information about f_θ than what can be inferred from (p, y). For example, the adversary cannot infer anything about f_θ via timing side-channels or other details of the implementation of the query interface.

Carlini, Nicholas, et al. "Stealing part of a production language model." *arXiv preprint arXiv:2403.06634* (2024).

The first heist

Assume the **API** returns **logits** (log probabilities) for **every token** of the vocab

Under these conditions, we can recover the model's **hidden dimension**



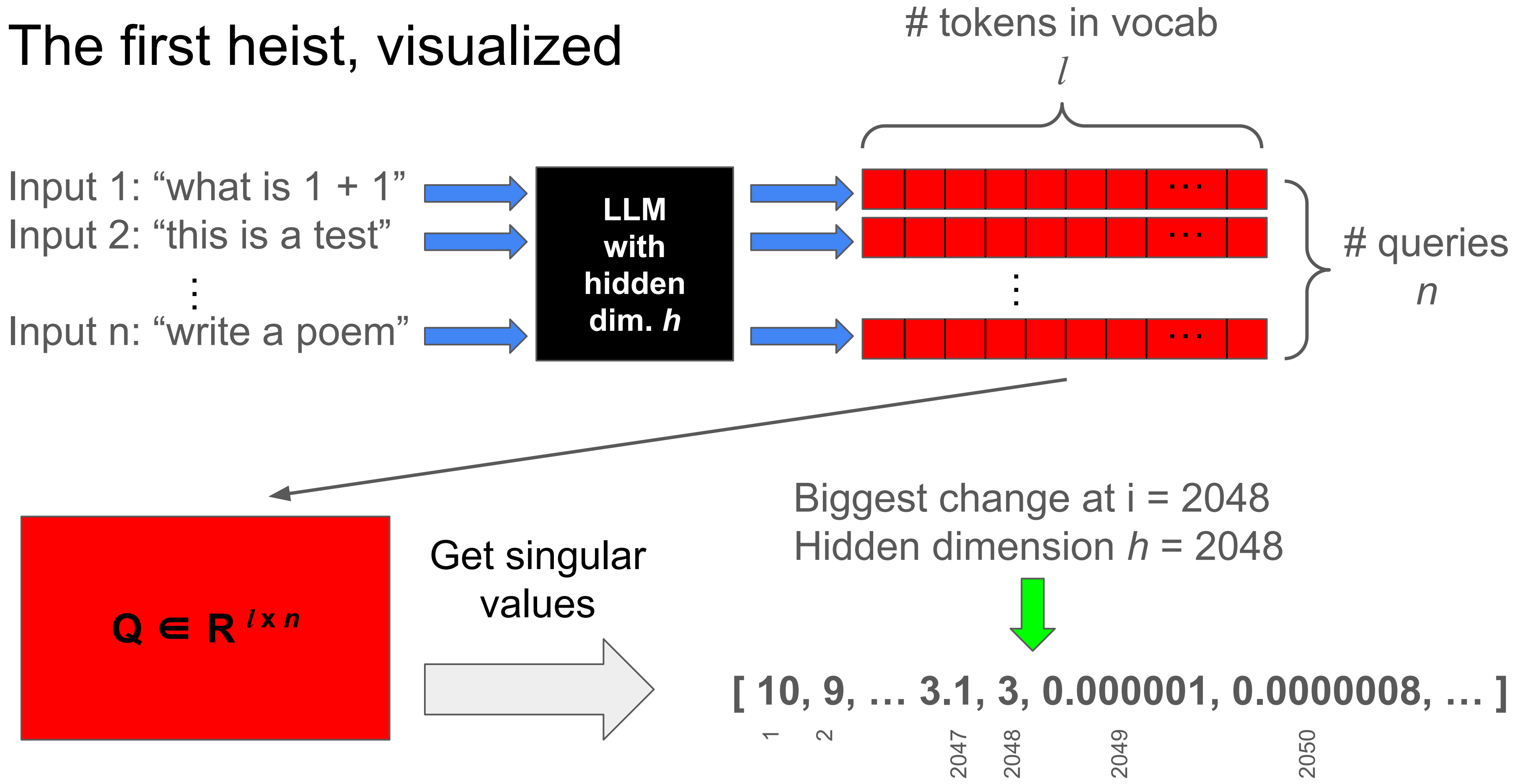
Algorithm 1 Hidden-Dimension Extraction Attack

```
Require: Oracle LLM $\mathcal{O}$ returning logits
1: Initialize $n$ to an appropriate value greater than $h$
2: Initialize an empty matrix $\mathbf{Q} = \mathbf{0}^{n \times l}$
3: for $i = 1$ to $n$ do
4:     $p_i \leftarrow$ RandPrefix()    ▷ Choose a random prompt
5:     $\mathbf{Q}_i \leftarrow \mathcal{O}(p_i)$
6: end for
7: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \leftarrow$ SingularValues($\mathbf{Q}$)
8: count $\leftarrow \arg\max_i \log\|\lambda_i\| - \log\|\lambda_{i+1}\|$
9: return count
```

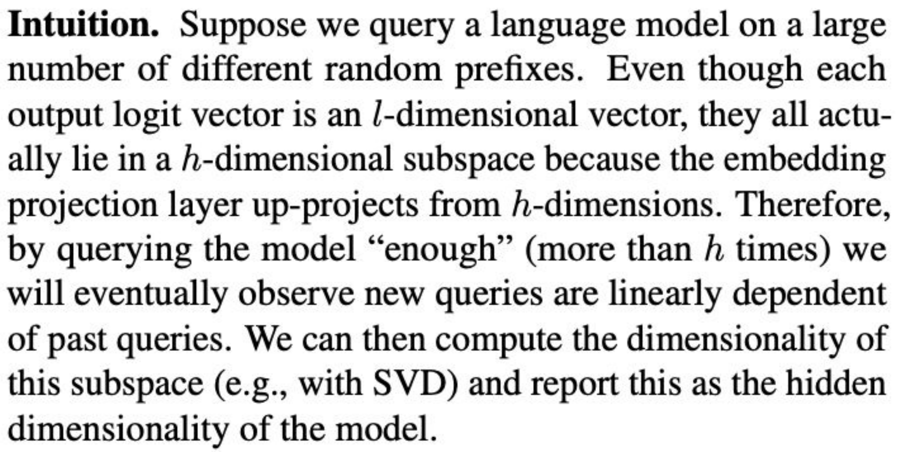
1. Query the model *n* times with random inputs
2. Combine all *n* queries into a matrix **Q** (e.g 3,000 queries x 100,000 tokens)
3. Perform SVD on **Q**
4. The model's hidden dimension == index of largest difference between consecutive singular values

Carlini, Nicholas, et al. "Stealing part of a production language model." *arXiv preprint arXiv:2403.06634* (2024).

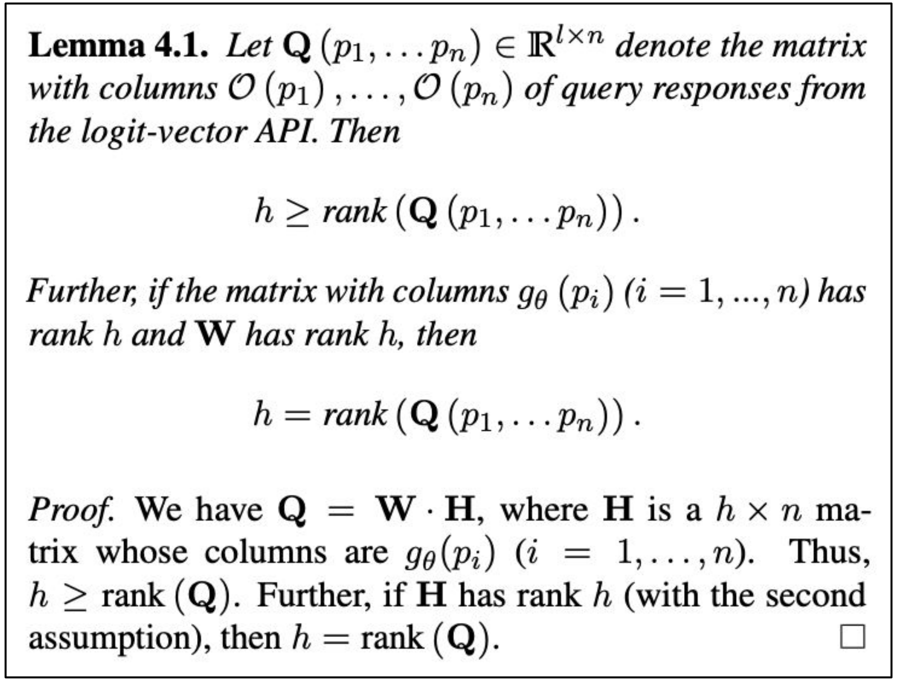
The first heist, visualized



Why does this work?



Intuition. Suppose we query a language model on a large number of different random prefixes. Even though each output logit vector is an l-dimensional vector, they all actually lie in a h-dimensional subspace because the embedding projection layer up-projects from h-dimensions. Therefore, by querying the model "enough" (more than h times) we will eventually observe new queries are linearly dependent of past queries. We can then compute the dimensionality of this subspace (e.g., with SVD) and report this as the hidden dimensionality of the model.



Lemma 4.1. *Let* $\mathbf{Q}\left(p_1, \ldots p_n\right) \in \mathbb{R}^{l \times n}$ *denote the matrix with columns* $\mathcal{O}\left(p_1\right), \ldots, \mathcal{O}\left(p_n\right)$ *of query responses from the logit-vector API. Then*

$$h \geq \mathit{rank}\left(\mathbf{Q}\left(p_1, \ldots p_n\right)\right).$$

Further, if the matrix with columns $g_\theta\left(p_i\right)$ $(i = 1, ..., n)$ *has rank* h *and* $\mathbf{W}$ *has rank* h*, then*

$$h = \mathit{rank}\left(\mathbf{Q}\left(p_1, \ldots p_n\right)\right).$$

Proof. We have $\mathbf{Q} = \mathbf{W} \cdot \mathbf{H}$, where $\mathbf{H}$ is a $h \times n$ matrix whose columns are $g_\theta(p_i)$ $(i = 1, \ldots, n)$. Thus, $h \geq \operatorname{rank}\left(\mathbf{Q}\right)$. Further, if $\mathbf{H}$ has rank h (with the second assumption), then $h = \operatorname{rank}\left(\mathbf{Q}\right)$. □

LLM hidden dimension is a bottleneck; outputs cannot have greater rank than the weights that produce them

Pythia 1.4B hidden rank heist

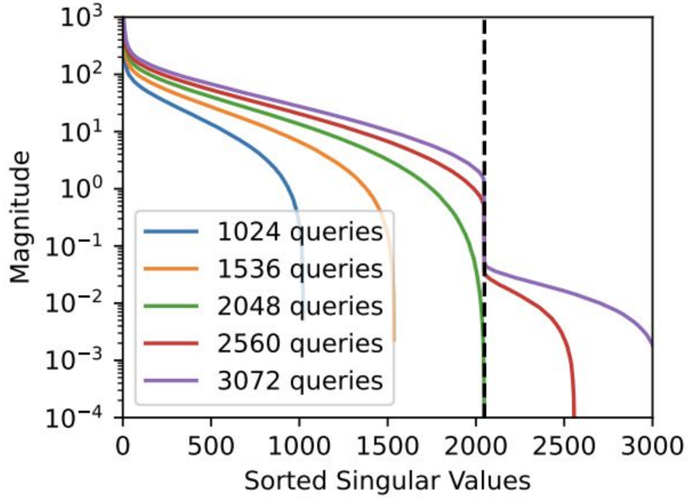


Figure 1. SVD can recover the hidden dimensionality of a model when the final output layer dimension is greater than the hidden dimension. Here we extract the hidden dimension (2048) of the Pythia 1.4B model. We can precisely identify the size by obtaining slightly over 2048 full logit vectors.

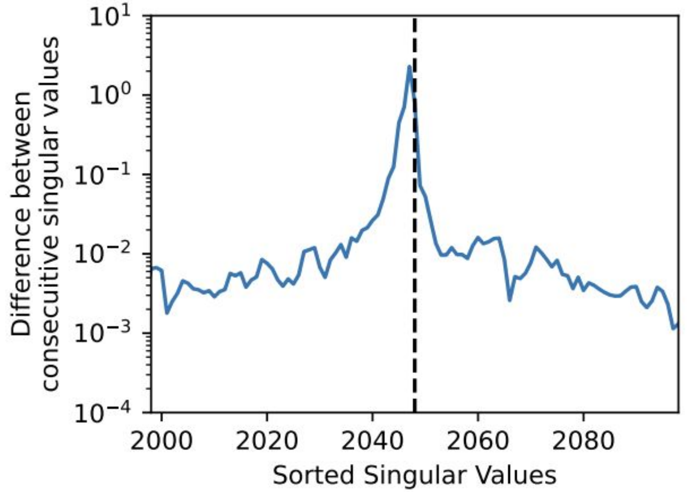


Figure 2. Our extraction attack recovers the hidden dimension by identifying a sharp drop in singular values, visualized as a spike in the difference between consecutive singular values. On Pythia-1.4B, a 2048 dimensional model, the spike occurs at 2047 values.

Stealing other model ranks

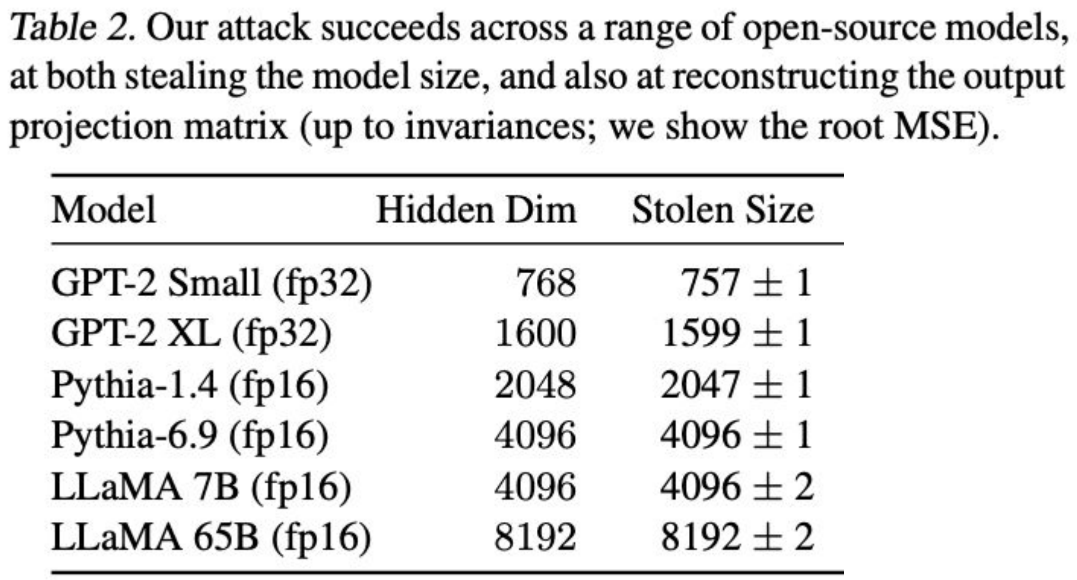


Table 2. Our attack succeeds across a range of open-source models, at both stealing the model size, and also at reconstructing the output projection matrix (up to invariances; we show the root MSE).

Model	Hidden Dim	Stolen Size
GPT-2 Small (fp32)	768	757 ± 1
GPT-2 XL (fp32)	1600	1599 ± 1
Pythia-1.4 (fp16)	2048	2047 ± 1
Pythia-6.9 (fp16)	4096	4096 ± 1
LLaMA 7B (fp16)	4096	4096 ± 2
LLaMA 65B (fp16)	8192	8192 ± 2

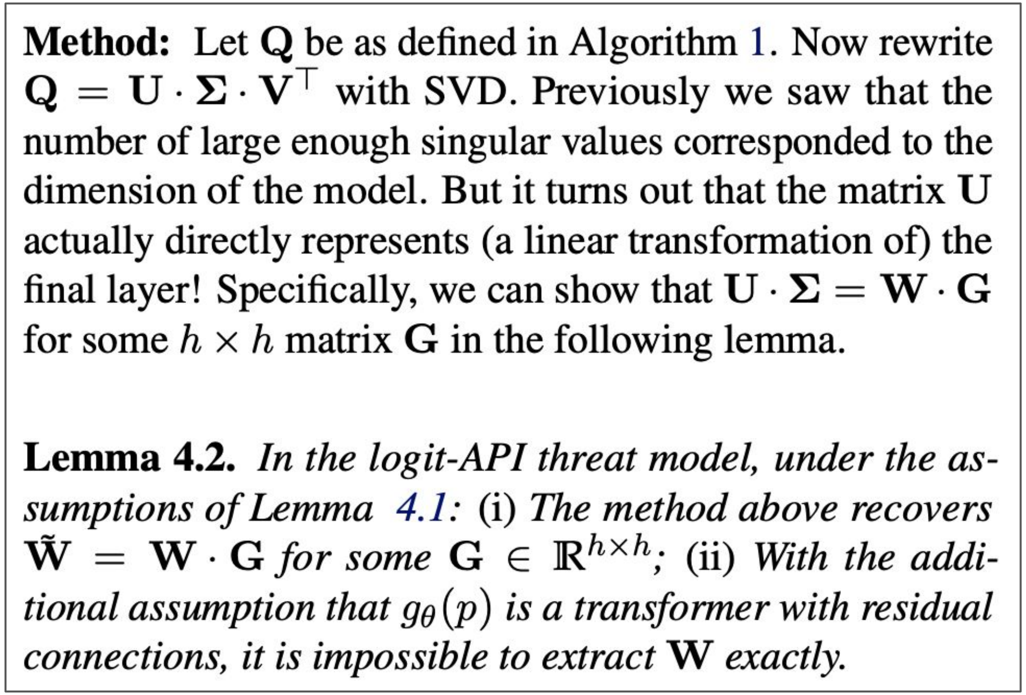
The attack is successful on every attempted model

The attack is “wrong” on GPT-2 Small because the model actually has a smaller rank than expected

- Linearly dependent weights

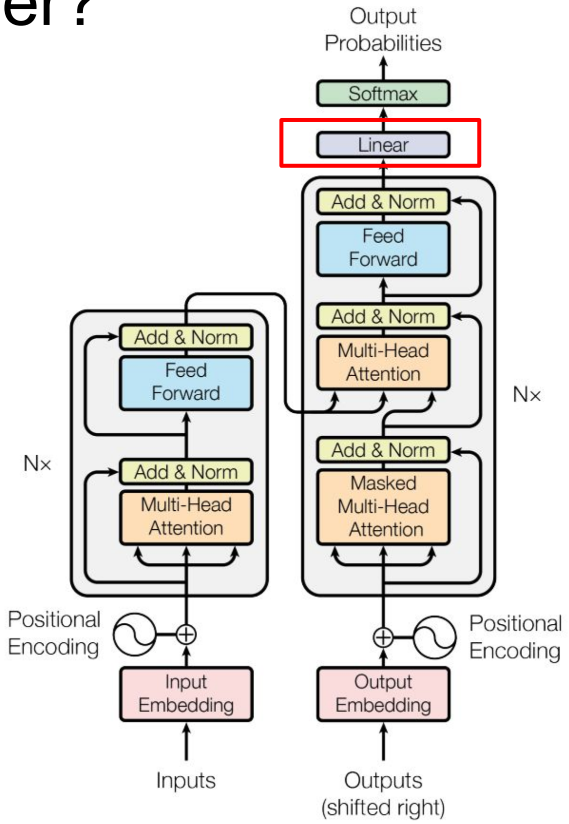
Carlini, Nicholas, et al. "Stealing part of a production language model." *arXiv preprint arXiv:2403.06634* (2024).

Can we steal the entire output linear layer?



Method: Let $\mathbf{Q}$ be as defined in Algorithm 1. Now rewrite $\mathbf{Q} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^\top$ with SVD. Previously we saw that the number of large enough singular values corresponded to the dimension of the model. But it turns out that the matrix $\mathbf{U}$ actually directly represents (a linear transformation of) the final layer! Specifically, we can show that $\mathbf{U} \cdot \mathbf{\Sigma} = \mathbf{W} \cdot \mathbf{G}$ for some $h \times h$ matrix $\mathbf{G}$ in the following lemma.

Lemma 4.2. *In the logit-API threat model, under the assumptions of Lemma 4.1:* (i) *The method above recovers* $\tilde{\mathbf{W}} = \mathbf{W} \cdot \mathbf{G}$ *for some* $\mathbf{G} \in \mathbb{R}^{h \times h}$; (ii) *With the additional assumption that* $g_\theta(p)$ *is a transformer with residual connections, it is impossible to extract* $\mathbf{W}$ *exactly.*



… in theory, yes!

Carlini, Nicholas, et al. "Stealing part of a production language model." *arXiv preprint arXiv:2403.06634* (2024).

Stealing the output linear layer

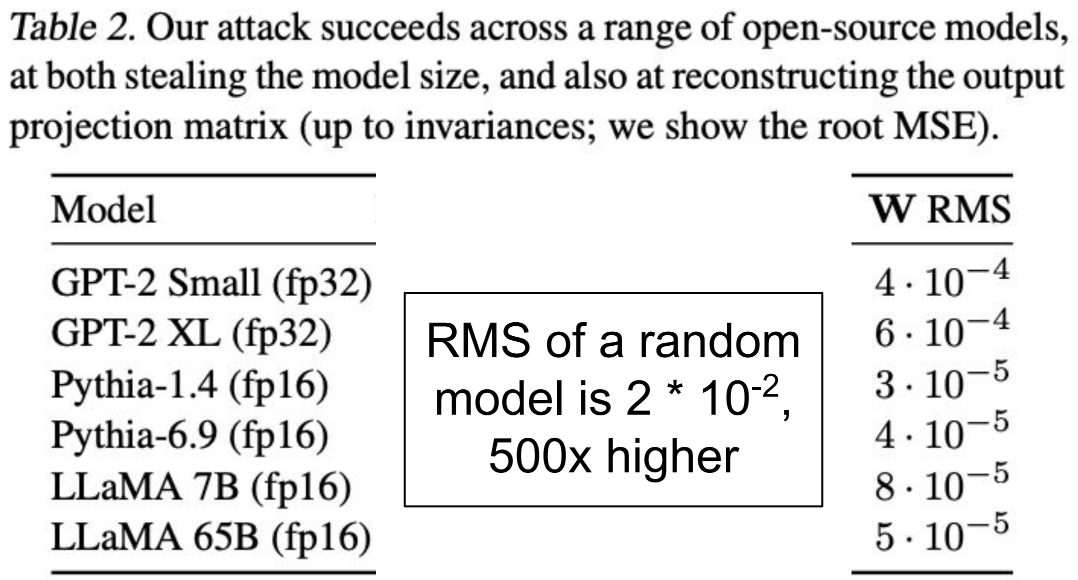


Table 2. Our attack succeeds across a range of open-source models, at both stealing the model size, and also at reconstructing the output projection matrix (up to invariances; we show the root MSE).

Model	**W** RMS
GPT-2 Small (fp32)	$4 \cdot 10^{-4}$
GPT-2 XL (fp32)	$6 \cdot 10^{-4}$
Pythia-1.4 (fp16)	$3 \cdot 10^{-5}$
Pythia-6.9 (fp16)	$4 \cdot 10^{-5}$
LLaMA 7B (fp16)	$8 \cdot 10^{-5}$
LLaMA 65B (fp16)	$5 \cdot 10^{-5}$

RMS of a random model is $2 * 10^{-2}$, 500x higher

We know that SVD output matrix **U** * **Σ** equals the linear layer **W** * some affine transformation matrix **G**

Solve the least squares system for **G** to approximate **W**

Report distance between real and calculated **W**

Carlini, Nicholas, et al. "Stealing part of a production language model." *arXiv preprint arXiv:2403.06634* (2024).

Let’s be more realistic…

The above attack makes a significant assumption:

The adversary can directly observe the complete logit vector for each input.

In practice, this is not true.

No production model we are aware of provides such an API.

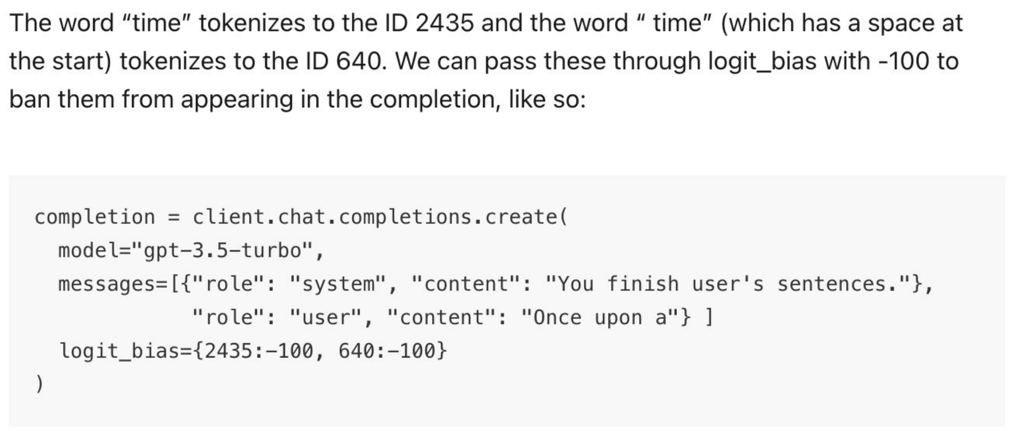
Instead, for example, they provide a way for users to get the top-K token log probabilities.

Then, what can we do?

We use Logit Bias !

What is logit bias?

Logit bias is an optional API parameter that modifies the likelihood of specified tokens appearing in a model generated output.



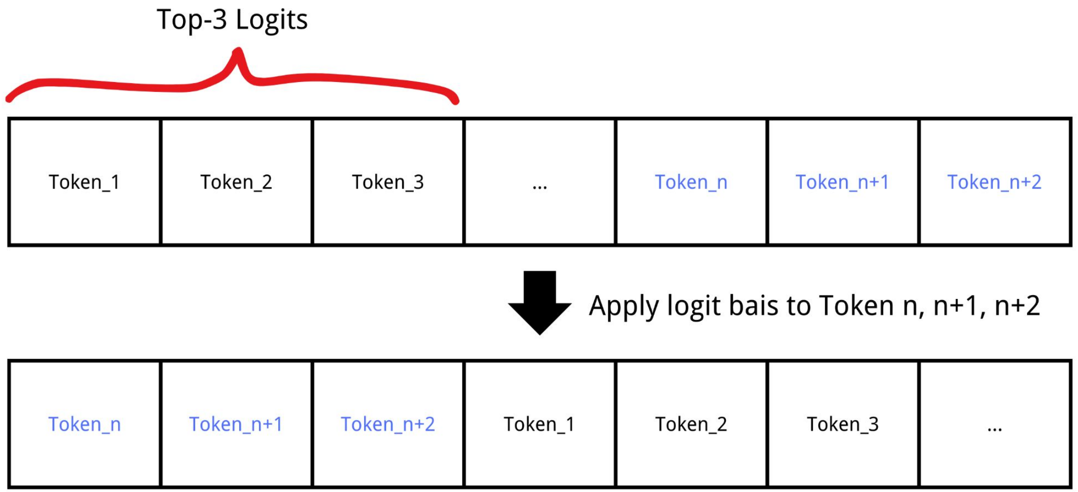
The word “time” tokenizes to the ID 2435 and the word “ time” (which has a space at the start) tokenizes to the ID 640. We can pass these through logit_bias with -100 to ban them from appearing in the completion, like so:

```
completion = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[{"role": "system", "content": "You finish user's sentences."},
             "role": "user", "content": "Once upon a"} ]
  logit_bias={2435:-100, 640:-100}
)
```

Cited from: https://help.openai.com/en/articles/5247780-using-logit-bias-to-alter-token-probability-with-the-openai-api

The second heist

Suppose that the API returned the top 3 logits. Then we could recover the complete logit vector for an arbitrary prompt by cycling through different 3 token sets with logit bias and measuring the top 3 logits each time.



Not Prob! Log Prob!!!

Most production model APIs return logprobs (the log of the softmax output of the model), for numerical stability and to simplify the cross-entropy loss.

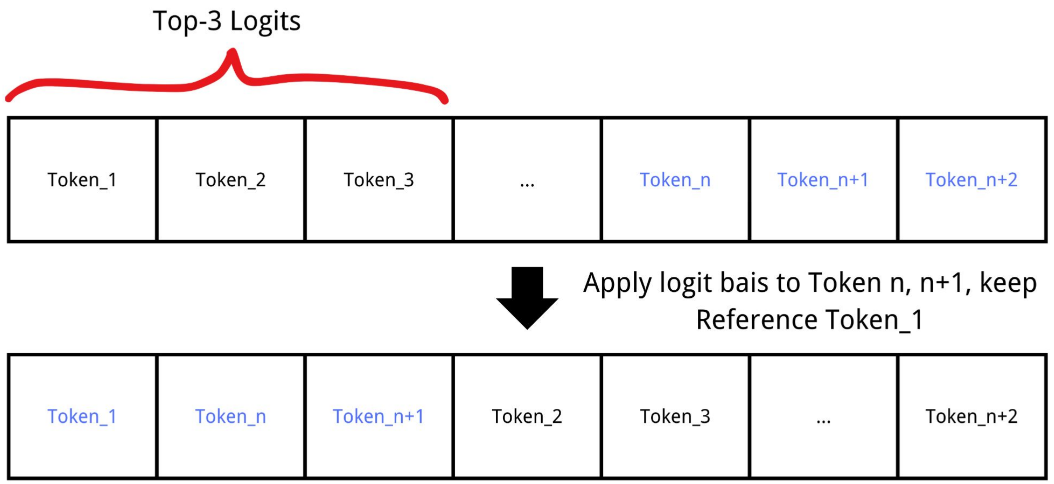
$$\log(\text{Softmax}(z_i)) = z_i - \log(\sum_j e^{z_j})$$

When we apply a logit bias B to the i-th token and observe that token's logprob, we get the value:

$$y_i^B = z_i + B - \log\left(\sum_{j \neq i} \exp(z_j) + \exp(z_i + B)\right)$$

The second heist

Since we can observe 3 logprobs, we can compare the reference token R to 2 tokens per query, by adding a large bias that pushes all 2 tokens into the top 3 (along with the reference token).



“Reference” token

For token i with bias B:

$$y_i^B = z_i + B - \log\left(\sum_{j\neq i}\exp(z_j) + \exp(z_i + B)\right)$$

For the reference token R without bias:

$$y_R^B = z_R - \log\left(\sum_{j\neq i}\exp(z_j) + \exp(z_i + B)\right)$$

Compute the difference between y_i and y_R:

$$y_R^B - y_i^B = (z_R - \log(*)) - (z_i + B - \log(*))$$

$$= z_R - z_i - B$$

The ultimate heist!

Place two further restrictions on the logit bias API:

Only can see the most likely token’s logprob (Top-1).

Each logit bias B is constrained to be in {−1, 0}.

Solution:

Query the model twice.

Once without logit bias, and once with a logit bias of −1 for token t.

Then the top token will be slightly more likely with a bias of −1.

How “slight” depending on the value of tokent’s logprob.

Even can attack model without logprob access!

So, what is the cost?

Bits of precision: The average number of bits of agreement between the true logit vector and the recovered logit vector.

Queries per logit: The average number of queries required to recover one full logit vector.

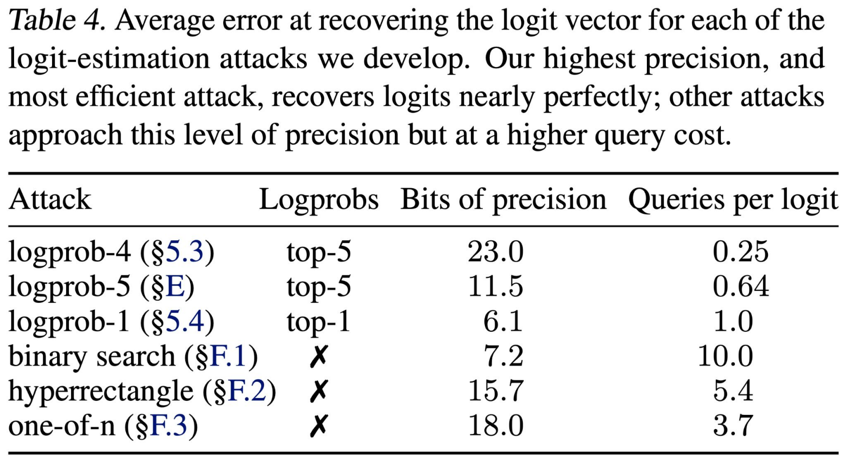


Table 4. Average error at recovering the logit vector for each of the logit-estimation attacks we develop. Our highest precision, and most efficient attack, recovers logits nearly perfectly; other attacks approach this level of precision but at a higher query cost.

Attack	Logprobs	Bits of precision	Queries per logit
logprob-4 (§5.3)	top-5	23.0	0.25
logprob-5 (§E)	top-5	11.5	0.64
logprob-1 (§5.4)	top-1	6.1	1.0
binary search (§F.1)	✗	7.2	10.0
hyperrectangle (§F.2)	✗	15.7	5.4
one-of-n (§F.3)	✗	18.0	3.7

Theoretical improvements are not always practical.

It works!

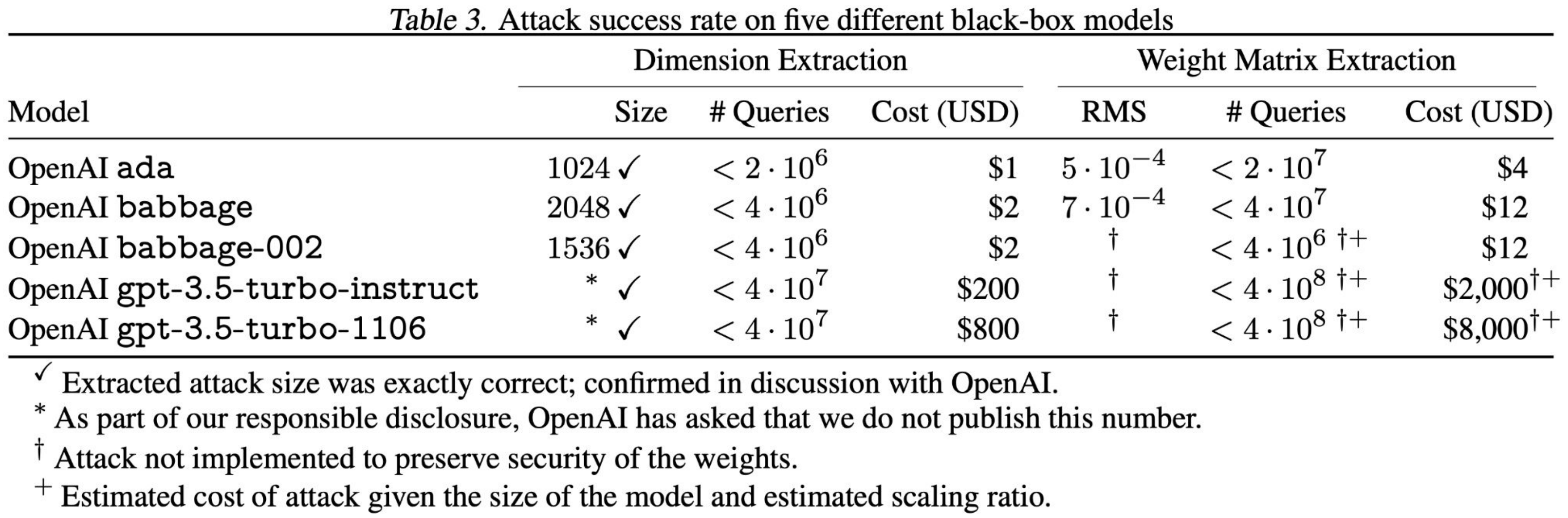


Table 3. Attack success rate on five different black-box models

	Dimension Extraction			Weight Matrix Extraction		
Model	Size	# Queries	Cost (USD)	RMS	# Queries	Cost (USD)
OpenAI `ada`	1024 ✓	$< 2 \cdot 10^6$	$1	$5 \cdot 10^{-4}$	$< 2 \cdot 10^7$	$4
OpenAI `babbage`	2048 ✓	$< 4 \cdot 10^6$	$2	$7 \cdot 10^{-4}$	$< 4 \cdot 10^7$	$12
OpenAI `babbage-002`	1536 ✓	$< 4 \cdot 10^6$	$2	†	$< 4 \cdot 10^6$ †+	$12
OpenAI `gpt-3.5-turbo-instruct`	* ✓	$< 4 \cdot 10^7$	$200	†	$< 4 \cdot 10^8$ †+	$2,000†+
OpenAI `gpt-3.5-turbo-1106`	* ✓	$< 4 \cdot 10^7$	$800	†	$< 4 \cdot 10^8$ †+	$8,000†+

✓ Extracted attack size was exactly correct; confirmed in discussion with OpenAI.
* As part of our responsible disclosure, OpenAI has asked that we do not publish this number.
† Attack not implemented to preserve security of the weights.
+ Estimated cost of attack given the size of the model and estimated scaling ratio.

1. Use 4-logprob attack, because it is most query efficient and most precise method.

2. The RMS between a randomly initialized model and the actual weights is 2·10^−2

Protect the bank!

We can do something to defend the attack.

Prevention

Remove logit bias.

Replace logit bias with a block-list.

Architectural changes.

Post-hoc altering the architecture.

Mitigation

Logit bias XOR logprobs.

Noise addition.

Rate limits on logit bias.

Detect malicious queries.

Long way to go…

Extending this attack beyond a single layer, finding methods that can be used for nonlinear layers.

Removing the logit bias assumption, other API parameters could give alternative avenues for learning logit information.

Exploiting the stolen weights, the stolen embedding projection layer might improve other attacks against the model.

We have seen the sunrise

While there appear to be no immediate practical consequences of stealing one layer of a production language model.

It represents the first time that any precise information about a deployed transformer model has been stolen.

07/11/2022 Colorado Louisville

We have seen the sunrise

“Adversarial ML had somewhat of a bad reputation for a few years. It seemed like none of the attacks we were working on actually worked in practice.

...

This paper shows—again—that all the work the adversarial ML community has been doing over the past few years can directly transfer over to this new age of language models we’re living in.”

Cited from: https://not-just-memorization.github.io/partial-model-stealing.html