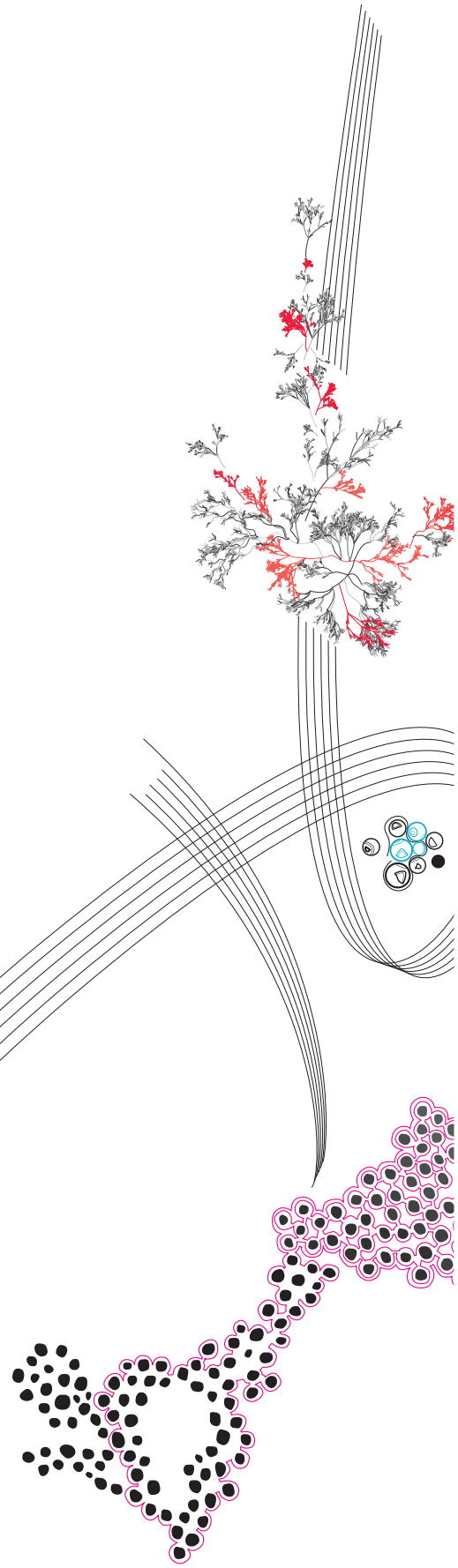


BSc Thesis Applied Mathematics

Stealing Part of a Production Language Model

Krystof Mitka

Supervisors: J. Goseling, L. Mariot, D. Paleka

June, 2024

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Abstract

The rapid advancement of large language models (LLMs) has led to their widespread deployment in various applications, often as black-box systems accessible only through APIs. This paper investigates the vulnerabilities of such models to model-stealing attacks, specifically focusing on extracting the full logit distributions of next-token predictions. By leveraging the bias map feature provided by APIs, we introduce a novel algorithm that efficiently recovers the complete logit distribution. Our contributions include the formulation of a class of algorithms that rely solely on the bias map, theoretical insights into their convergence and lower bounds, and the identification and analysis of a new state-of-the-art attack. We demonstrate the effectiveness of our approach through theoretical analysis and numerical experiments, highlighting the potential risks and implications for the security of proprietary language models.

1 Introduction

The current best-performing large language models are typically closed proprietary systems accessible only through APIs. Companies provide minimal information about their models size, architecture, training data, or training processes (OpenAI et al., 2024, Anil et al., 2023). However, recent research by Carlini et al., 2024 and Finlayson et al., 2024 has shown that it is possible to extract detailed information about these closed models by exploiting certain functionalities offered by APIs from companies like OpenAI and Google. These studies have successfully extracted the exact model dimensionality and even the entire last layer representation, known as the unembedding matrix. In response to these security breaches, companies have revised or restricted parts of their APIs to invalidate or significantly increase the difficulty and cost of such attacks, aiming to safeguard their models integrity and confidentiality.

To extract the model dimensionality and steal the unembedding layer, the attacks rely on collecting a large number of full logit distributions. A *full logit distribution* refers to the output vector produced by a neural network before the application of the softmax function. This vector contains the raw, unnormalized scores (logits) for each class in a classification task. In the context of large language models, the full logit distribution represents the scores assigned to each token in the model's vocabulary for a given input sequence. This paper focuses exclusively on efficiently extracting the next-token full logit distributions, as these are the primary target for attackers aiming to reverse-engineer parts of the model.

Related Work. The attacks formulated in related works (Carlini et al., 2024, Finlayson et al., 2024) relied on *two* API functionalities provided by the companies to steal a large number of full next-token logit distributions.

1. **Access to Log Probabilities.** The access to the log probabilities of some of the most likely tokens. This feature enables users to obtain detailed information about the model's predictions for the next token in a sequence.
2. **Biasing Tokens Using a Bias Map.** The ability to add a bias term to each logit before applying the softmax function. This bias map feature allows API users to influence the model's output by either censoring certain tokens or promoting others. For instance, users can increase the likelihood of specific tokens or completely suppress the generation of undesirable tokens.

The most efficient attacks, as detailed in Carlini et al., 2024; Finlayson et al., 2024, employ a combination of the aforementioned features. By biasing all tokens to appear among the

most likely tokens and subsequently collecting their log probabilities, attackers can reverse engineer these probabilities back to logits. However, following the publication of these attacks, both Google and OpenAI have removed the feature that allows viewing of biased log probabilities. Consequently, this paper focuses exclusively on algorithms that leverage the bias map feature to extract the complete logit distribution.

Contribution. This work introduces a state-of-the-art algorithm for extracting the full logit distribution by leveraging mathematical insights into the logit distribution and utilizing the logit bias map provided by the company's API. Concretely this work:

- Formulates a class of algorithms that rely exclusively on the bias map feature provided by the model's API. These algorithms are designed to efficiently extract the full logit distribution by strategically adjusting the bias terms for multiple tokens simultaneously.
- Offers novel observations and theoretical results regarding the convergence properties and lower bounds of these algorithms. This includes a detailed analysis of the conditions under which the algorithms converge and the minimum number of queries required to achieve a specified precision.
- Identifies and generalizes a new state-of-the-art attack that leverages the bias map feature to extract the complete logit distribution. This attack is shown to be more efficient than existing methods, requiring fewer queries to achieve the same level of precision.
- Analyzes this new attack in depth, providing a comprehensive explanation for its behavior. This includes a thorough numerical analysis, demonstrating the attack's effectiveness through empirical results and comparing its performance against existing methods.

2 General Algorithm Formulation

We consider a typical transformer architecture for a large language model (Vaswani et al., 2023). Let $\mathcal{P}(\mathcal{X})$ represent the space of all probability distributions over the vocabulary $\mathcal{X}$. Transformer models take N tokens as input from a vocabulary $\mathcal{X}$ of size v and output a probability distribution for the next token $\mathbf{q} \in \mathcal{P}(\mathcal{X})$, i.e., models of the form $f : \mathcal{X}^N \mapsto \mathcal{P}(\mathcal{X})$. The probability distribution, conditioned on the previous N tokens in the sequence, is computed by applying a softmax $\mathbb{R}^v \mapsto \mathcal{P}(\mathcal{X})$ to the full logit distribution vector $\mathbb{R}^v$ of the last token, $\mathbf{z}$.

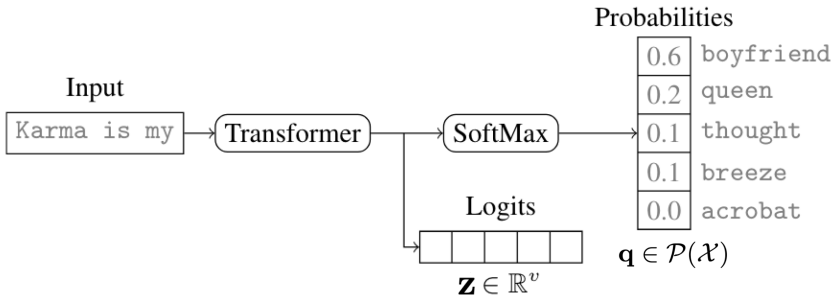


FIGURE 1: Diagram of the last layer of a transformer model.

Logits are the raw, unnormalized scores output by a neural network's final layer before applying the softmax function. The softmax function is defined as:

$$\text{softmax}(\mathbf{z}) = \left[\frac{e^{z_1}}{\sum_{i=1}^{v} e^{z_i}}, \ldots, \frac{e^{z_v}}{\sum_{i=1}^{v} e^{z_i}} \right]$$

We assume access to the model API provided by the owners of the model, such as OpenAI or Google, and we explore the class of attacks based on the following two assumptions.

Access to Bias Map. We assume that it is possible to set a bias term b_i for any token $\mathcal{X}_i$ in the vocabulary $\mathcal{X}$, where b_i can range from $-B$ to B. The number of tokens that can have a bias term set is limited to a constant $N_b \leq v$. We use vector notation $\mathbf{b} \in \mathbb{R}^v$, meaning the length of the bias vector is v. In cases where $N_b < v$, for all tokens that are not biased, we assume $b_i = 0$. The bias vector $\mathbf{b}$ is added to the logit vector $\mathbf{z}$ before applying the softmax function.

Access to Top Token. We constrain the information received from the API to only the token with the highest probability after applying the softmax function.

Define prompt p as a unique sequence of N tokens and let $g : \mathcal{X}^N \to \mathbb{R}^v$ be a function that outputs the logit vector $\mathbf{z}$ for the next token in the sequence. The Oracle $\mathcal{O}$ (API) is a black-box function that, given a prompt p and a bias vector $\mathbf{b}$, returns the token with the highest probability.

$$\mathcal{O}(p, b) \leftarrow \text{ArgMax}\left(\text{softmax}\left(g(p) + \mathbf{b}\right)\right).$$

Additionally, we assume that $\text{ArgMax}(\mathbf{v})$ returns the index of the coordinate with the largest value in some vector $\mathbf{v} \in \mathbb{R}^v$. In case two or more values are equal and the maximum, the ArgMax function will randomly select one of the indices corresponding to the maximum values. We refer to the indices of the vectors and tokens interchangeably.

Definition 1. *We say that a token is* ***sampled*** *if it is the token returned by the oracle's ArgMax function.*

Note that we can ensure the token with the highest logit value, which corresponds to the highest probability token, is consistently sampled by setting the temperature parameter to 0 or by configuring the top-k parameter to 1. The temperature parameter controls the randomness of the sampling process, with a value of 0 making the model deterministic by always selecting the highest logit value. The top-k parameter limits the sampling pool to the top k tokens, and setting it to 1 ensures only the token with the highest logit value is considered.

2.1 Logit normalization

To facilitate the implementation of the algorithm and ensure consistent scaling of the logit values, we will normalize the logit vectors to the interval $[0, 1]$. This normalization helps in simplifying the bias adjustments and maintaining a uniform scale for comparison across different logits.

Given a logit vector $\mathbf{z}$, we normalize it using the following procedure. Calculate the minimum and maximum values of the logit vector $\mathbf{z}$.

$$z_\delta = \min(\mathbf{z}), \quad z_\Delta = \max(\mathbf{z})$$

Normalize each component z_i of the logit vector $\mathbf{z}$ to the interval $[0, 1]$ using the formula:

$$\hat{z}_i = \frac{z_i - z_\delta}{z_\Delta - z_\delta}$$

where $\hat{z}_i$ is the normalized logit value.

In a realistic setting, we can assume knowledge of the width of the interval on which logits lie. Observations from API providers suggest that it suffices to take B as the width of this interval. Furthermore, to facilitate shifting the interval and simplify notation, we assume knowledge of the maximum logit value z_Δ.[1] Henceforth, we assume all logit vectors $\mathbf{z}$ are normalized to the interval $[0, 1]$.

2.2 Algorithm Framework

We will consider algorithms that modify the bias of multiple tokens simultaneously (Carlini et al., 2024). Specifically, at each step, we will adjust the biases of all N_b tokens. In scenarios where $N_b < v$, it is necessary to select subsets of tokens and repeat the procedure $\frac{v}{N_b}$ times. Prior to initiating the algorithm, we perform an initial query to the API without applying any bias map to identify the top token and set its index to 0 i.e. top token is z_0. We now introduce a general algorithm framework that is used to iteratively narrow down the possible values of $\mathbf{z}$, using lower bound vector $\mathbf{l} \in \mathbb{R}^{N_b}$ and upper bound vector $\mathbf{h} \in \mathbb{R}^{N_b}$, and give a concrete description of its implementation. The goal of all instances of this algorithm discussed in the following sections will be to come up with the function $\mathbf{f}(\mathbf{l}, \mathbf{h}, r)$, which sets all N_b bias terms at each step of the algorithm. The following algorithm framework is a vectorized and generalized version of the algorithm discussed in Carlini et al., 2024.

Algorithm 1 Learning logit differences with multi-token calls (Vectorized)

$\mathbf{l} \leftarrow \mathbf{0}, \mathbf{h} \leftarrow \mathbf{1}$
$\mathcal{C} = \{\mathbf{x} : z_0 - z_i \leq 1, \forall i\}$
$r \leftarrow 1$
while stop_condition$(\mathbf{l}, \mathbf{h}, r)$ is false **do**
    $\mathbf{b} \leftarrow \mathbf{f}(\mathbf{l}, \mathbf{h}, r)$
    $k \leftarrow \mathcal{O}(p, \mathbf{b})$
    $\mathcal{C} \leftarrow \mathcal{C} \cap \{\mathbf{x} : z_k + b_k \geq z_j + b_j, \forall j \neq k\}$
    $\mathbf{l} \leftarrow \mathbf{x}_{\min}$
    $\mathbf{h} \leftarrow \mathbf{x}_{\max}$
    $r \leftarrow r + 1$
end while
return $\mathbf{l}, \mathbf{h}$

At each step of Algorithm 1, we provide a specific bias vector $\mathbf{b} = \mathbf{f}(\mathbf{l}, \mathbf{h}, r)$ and sample a top token from the oracle, denoted as k. By the definition of $\mathcal{O}$, this results in $N_b - 1$ inequalities of the form $z_k + b_k \geq z_j + b_j$ for all $j \neq k$. We can rewrite these inequalities as $-z_k + z_j \leq b_k - b_j$. Therefore, at step n, we can represent the coefficients of logits as $A_n \in \mathbb{R}^{N_b - 1 \times N_b}$ and the resulting bound as $\mathbf{b}_n \in \mathbb{R}^{N_b - 1}$. Aggregating results after n steps,

[1] Note that OpenAI provides access to the log probabilities of the top-5 most likely tokens, which are not affected by the bias term after the introduction of the attacks. Consequently, we can infer the initial top-5 token logits, allowing us to determine the maximum value.

we let:

$$A = \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_n \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

So, the polytope $\mathcal{C}$ defined by this system of linear inequalities can be expressed as:

$$\mathcal{C} = \left\{ \mathbf{x} \in \mathbb{R}^{N_b} \mid A\mathbf{x} \leq \mathbf{b} \right\}$$

After updating the system for our polytope at each step with new information, we are interested in finding $\mathbf{x}_{\min}$ and $\mathbf{x}_{\max}$, where $\mathbf{x}_{\min}$ is the solution of the polytope where each component of the vector $\mathbf{x}$ is minimal and $\mathbf{x}_{\max}$ is the solution where each component is maximal. Formally, this can be defined as:

$$\mathbf{x}_{\min} = \left(\min_{\mathbf{x} \in \mathcal{C}} x_1, \min_{\mathbf{x} \in \mathcal{C}} x_2, \ldots, \min_{\mathbf{x} \in \mathcal{C}} x_n \right)$$

and

$$\mathbf{x}_{\max} = \left(\max_{\mathbf{x} \in \mathcal{C}} x_1, \max_{\mathbf{x} \in \mathcal{C}} x_2, \ldots, \max_{\mathbf{x} \in \mathcal{C}} x_n \right)$$

We can compute both $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ by considering the linear programming problem of finding the shortest path on a weighted graph (Carlini et al., 2024).

Generally, we consider two different stop conditions for the attacker's querying process.

1. **Fixed Budget Condition**: When the attacker has a budget of T requests for each batch of N_b tokens, the querying process stops after T rounds. Formally, this stop condition can be defined as:

$$\text{stop\_condition}(\mathbf{l}, \mathbf{h}, r) = \begin{cases} \text{True} & \text{if } r \geq T, \\ \text{False} & \text{otherwise.} \end{cases}$$

2. **Precision Condition**: When the attacker aims to attain a certain precision ϵ of the result and disregards the budget, the querying process stops when the largest interval so far is less than ϵ. Formally, this stop condition can be defined as:

$$\text{stop\_condition}(\mathbf{l}, \mathbf{h}, r) = \begin{cases} \text{True} & \text{if } \|\mathbf{h} - \mathbf{l}\|_\infty < \epsilon, \\ \text{False} & \text{otherwise.} \end{cases}$$

Later in numerical analysis, we will test the algorithms by comparing against the real value of the logit vector we are looking for. Therefore, we will add an argument $\mathbf{z}$ to the stop condition for this comparison. This inclusion ensures the stop condition takes into account the precision of the estimated logit vector in relation to the true logit vector $\mathbf{z}$.

3 Algorithm Theoretical Results

3.1 Lower bound

Before creating algorithms solving for the best bias term function $\mathbf{f}(\mathbf{l}, \mathbf{h}, r)$, we will check the theoretical lower bound both from the information theory perspective and by construct a perfect algorithm using the real logit vector z.

Initially, we consider a theoretical bound on the minimum number of queries required for an attacker to obtain N_b tokens from an information theory perspective. The following is a variation of a lemma from Carlini et al., 2024.

Lemma 1. *Assume the entries $P_1, \dots P_{N_b}$ of logit $\mathbf{z} \in \mathbb{R}^v$ are i.i.d. uniform over $[0, 1]$. To recover all N_b logits up to ∞-norm error ε, the number of queries to $\mathcal{O}(p, \cdot)$ we need is at least:*

$$\frac{-N_b \log_2(\varepsilon)}{\log_2(N_b)}.$$

For testing purposes in the following section we will consider logit vectors of length $N_b = 50$. In such setting we consider

$$\frac{-\log_2(\epsilon)}{\log_2(N_b)} \approx -0.588 \log_2(\epsilon)$$

queries per logit. For 6 bits of precision that yields 1.06 queries per logit.

As a different theoretical exercise, we can explore in how many steps can we converge assuming that the bias function has access to the true values of the logits we are trying to estimate, i.e., $\mathbf{f}(\mathbf{l}, \mathbf{h}, r, \mathbf{z})$, and seek a function that accelerates the algorithm's convergence.

Consider the isolated case of determining the value of a single logit. In this scenario, we can recover the logit by first biasing just below the logit value, $\frac{z_i - \epsilon}{2}$, and then just above it, $\frac{z_i + \epsilon}{2}$. This approach yields an interval around the logit z_i of $[z_i - \epsilon, z_i + \epsilon]$. Biasing in this manner would provide the desired bounds $\mathbf{l}$ and $\mathbf{h}$ in $2 * N_b$ steps.

An even more efficient result can be achieved by biasing all tokens around their true values z_i. Let $r_{mod} = r \mod N_b$ and define the bias function as

$$\mathbf{f}(\mathbf{l}, \mathbf{h}, \mathbf{z}, r) = \mathbf{1} - \mathbf{z} + \epsilon \cdot \mathbf{e}_{r_{mod}}$$

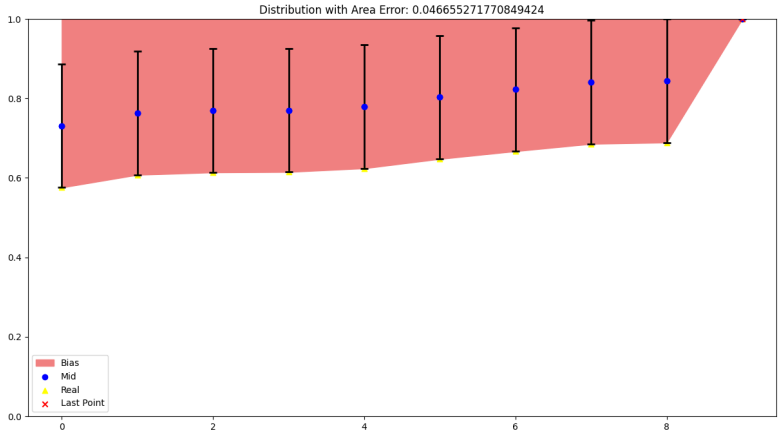
Where $\mathbf{e}_{r_{mod}}$ is zero everywhere except at the position r_{mod}. At position r_{mod} we add a perturbation to ensure it will be the output of the $ArgMax$. With each step we update the lower bound for logit at position r_{mod} as can be derived from inequality with token 0:

$$z_{r_{mod}} + b_i + \epsilon \geq 1 \implies z_{r_{mod}} \geq z_{r_{mod}} - \epsilon = l_i$$

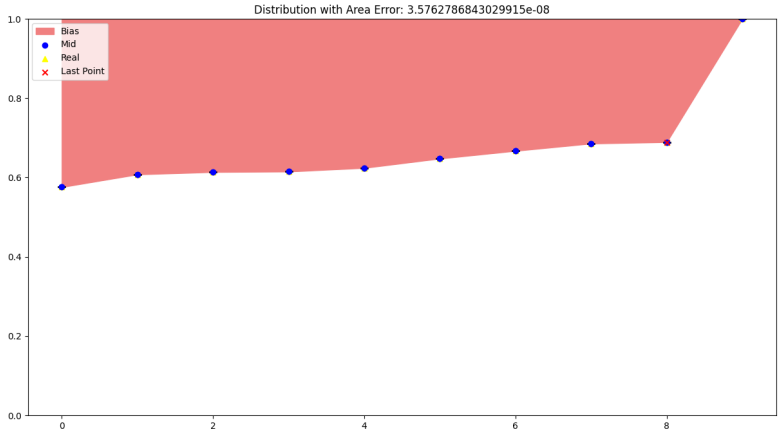
After obtaining lower bounds for all tokens, we bias token $z_0 = 1$ with the perturbation, which updates the upper bound of all tokens:

$$1 + \epsilon \geq z_i + b_i \implies 1 + \epsilon \geq z_i + (1 - z_i) \implies h_i = z_i + \epsilon \geq z_i \quad \forall i$$

Therefore after just N_b queries, which is essentially 1 query per logit, we are able to create an arbitrarily good bound $[z_i + \epsilon, z_i - \epsilon]$ around z_i.



(A) Perfect Algorithm at step $N_b - 1$



(B) Perfect Algorithm at step N_b

3.2 Convergence condition

Observation 1. *Assuming we have never sampled a token i, at each step we can only provide an upper bound for the unsampled token via the inequality $z_k + b_k - b_i \geq z_i$. This implies that our information about z_i is limited to adjusting the upper bound h_i, while the lower bound l_i remains unchanged.*

It follows that to achieve a certain ∞-norm precision around every logit, we need to sample each token at least once. The theorem also highlights issues related to the density of $\mathbb{R}$.[2] Specifically, consider the scenario where the bias function is set to $\frac{h_i+z_i}{2}$ for some token i. In this case, the interval around the logit z_i would be narrowed at each iteration, but only from above. Consequently, the upper bound h_i would approach z_i asymptotically, without any improvement in the lower bound l_i.

To prevent guesses that asymptotically approach the logit value, we will introduce a convergence condition. Recall from Algorithm 1 that, at the first step, the lower bound vector $\mathbf{l}$ is initialized to the zero vector $\mathbf{0}$, and the upper bound vector $\mathbf{h}$ is initialized to the vector $\mathbf{1}$. To achieve a specified precision ϵ around the true logit value z_i, it suffices to narrow the interval between the lower and upper bounds such that their width, $h_i - l_i$, equals ϵ. Consequently, at the initial and subsequent steps, we can restrict the possible values of the bounds by creating a partition that discretizes the interval $[0, 1]$ into points separated by ϵ. We consider a logit to be determined once its lower and upper bounds are separated by exactly ϵ. At this point, we cease further biasing of that particular logit.

Definition 2. *For a specific logit z_i, initially, the interval $[0, 1]$ is partitioned into points of size ϵ, forming the set called the **initial partition**:*

$$P_0^\epsilon = \{0, \epsilon, 2\epsilon, \ldots, k\epsilon\}$$

where $k = \left\lfloor \frac{1}{\epsilon} \right\rfloor$.

*At step n, the **partition at step** n for the interval $[l_i, h_i]$ of the logit z_i is a subset of the initial partition P_0^ϵ, consisting of points that are in sequence. Formally, at step n, the partition P_n^ϵ satisfies:*

$$P_n^\epsilon \subseteq P_0^\epsilon \quad \textit{and} \quad \forall x_j, x_{j+1} \in P_n^\epsilon,\, x_j < x_{j+1}$$

Definition 3. *At step n, the **interval length sum** is defined as the l_1-norm of the difference between the vectors of upper bounds $\mathbf{h}$ and lower bounds $\mathbf{l}$. Formally, the interval length sum E_n at step n is given by:*

$$E_n = \|\mathbf{h} - \mathbf{l}\|_1$$

where $\mathbf{l} = (l_1, l_2, \ldots, l_m)$ and $\mathbf{h} = (h_1, h_2, \ldots, h_m)$ are the vectors of lower and upper bounds of the intervals for logits at step n, respectively.

We split the interval around each logit to prevent guesses that approach the logit from above and we define a notion of error over all logits at each step n. Note that the interval sum error is a decreasing function with every step. Now we can formalize a sufficient condition for the bias function so that Algorithm 1 converges.

[2]Logits are typically represented using 32-bit floating-point numbers. According to the IEEE 754 standard, this provides a precision of 6 to 9 significant decimal digits, implying that logits are not exactly representable in $\mathbb{R}$ and are therefore not continuous. However, for the purposes of this analysis, we proceed under the assumption of continuity.

Theorem 1. *If at each step of Algorithm 1 we choose a bias vector* $\mathbf{b}$ *s.t. for at least one logit the bias is chosen* $b_i \in 1 - P_n^\epsilon$*, then for any* $\epsilon > 0$ *there exists* R*, number of rounds, such that* $\forall r \geq R$ *the interval length sum at round* n *satisfies* $E_n \leq \epsilon N_b$*.*

Proof. We consider two cases. First, assume that by setting $b_i \in 1 - P_n^\epsilon$, we sample the token 0. Then, we can update the h_i for all biased tokens by setting $h_i = 1 - b_i$, which narrows the interval by $h_i - (1 - b_i)$.

Otherwise, we must sample a biased token. Then we can update the lower bound l_i of the sampled token by setting $l_i = 1 - b_i$, as at minimum we will gain information about a better lower bound of the sampled token, which narrows the interval by $(1 - b_i) - l_i$. □

We can intuitively understand the convergence condition as follows: biasing at least one term in each round must divide the interval around the logit z_i into two sections, where only one section will remain viable after sampling the next token. For instance, the simple binary search algorithm introduced in Morris et al., 2023 achieves this condition by splitting the interval around each logit into equal halves. This is done by setting the bias to $1 - \frac{h_i + l_i}{2}$, which would be the midpoint of the partition P_n^ϵ in our case.

4 Solution Instances

We have established that if we bias each logit around its true value, we quickly converge to a narrow bound around each logit. Another way to understand this biasing strategy is that we have adjusted all logits so that their 'probability' of being sampled is $1/n$. Since the true values of the logits are unknown, our next best approach is to assume a probability distribution over the interval $[l_i, h_i]$ and treat each logit as random variable drawn from this distribution. In this section we explore different biasing techniques that use these assumptions.

4.1 StartOverN With Uniform Prior

We start by assuming uniform prior over the interval $[l_i, h_i]$ for each logit. A simple approach is to bias logits such that the probability of sampling token 0 (i.e. the maximum token) is $1/n$. This approach, introduced briefly in Carlini et al., 2024, has been the state-of-the-art method prior to the work presented in this paper. We formalize this approach and provide a derivation for a solution of the biasing term.

Theorem 2. *Given independent and identically distributed (i.i.d.) random variables* $P_1, P_2, \ldots, P_{n-1}$ *from a uniform distribution on an interval between* l_i *and* h_i*, and constants* $b_1, b_2, \ldots, b_{n-1}$*, the StartOverN algorithm aims to find* b_i *such that:*

$$P\left(\max(P_1 + b_1, P_2 + b_2, \ldots, P_{n-1} + b_{n-1}) \leq 1\right) = \frac{1}{n}$$

A solution to this problem is given by:

$$b_i = 1 - l_i + \left(\frac{1}{n}\right)^{\frac{1}{n-1}} (h_i - l_i)$$

Proof. We need

$$
\begin{aligned}
P\left(\max(P_1 + b_1, \ldots, P_{n-1} + b_{n-1}) \leq 1\right) &= \frac{1}{n} \\
P(P_1 + b_1 \leq 1) \cdot \ldots \cdot P(P_{n-1} + b_{n-1} \leq 1) &= \frac{1}{n} \\
P(P_1 \leq 1 - b_1) \cdot \ldots \cdot P(P_{n-1} \leq 1 - b_{n-1}) &= \frac{1}{n} \\
\prod_{i=1}^{n-1} \frac{1 - b_i - l_i}{h_i - l_i} &= \frac{1}{n}
\end{aligned}
$$

This is satisfied if $\forall i$:

$$
\begin{aligned}
\frac{1 - b_i - l_i}{h_i - l_i} &= \left(\frac{1}{n}\right)^{\frac{1}{n-1}} \\
1 - b_i - l_i &= \left(\frac{1}{n}\right)^{\frac{1}{n-1}} (h_i - l_i) \\
b_i &= 1 - l_i + \left(\frac{1}{n}\right)^{\frac{1}{n-1}} (h_i - l_i)
\end{aligned}
$$

Hence, $b_i = 1 - l_i + \left(\frac{1}{n}\right)^{\frac{1}{n-1}} (h_i - l_i)$ is a solution. □

In the vector notation of the biasing function we would therefore write:

$$
\mathbf{f}(\mathbf{l}, \mathbf{h}, r) = 1 - \mathbf{l} + \left(\frac{1}{n}\right)^{\frac{1}{n-1}} (\mathbf{h} - \mathbf{l})
$$

For the readers coming from Carlini et al., 2024 we note that the solution can be rewritten using an equivalent scaling term:

$$
\left(\frac{1}{n}\right)^{\frac{1}{n-1}} = \exp\left(\frac{-\log(n)}{n-1}\right)
$$

4.2 Finding parameters for normal distribution

In the previous derivation, we assumed a uniform distribution for all logits in the full output token vector. In this section, we hypothesize that logits follow a normal distribution.

We collected 52 logit distributions from the LLaMA-7b model for single character prompts, specifically using characters of the alphabet A-Z (26) and a-z (26). Each logit distribution was normalized with an assumed width of 40, and then the 52 logit distributions were flattened into a single vector. In Figure 3, we plot the histogram of the logit distribution with 100 bins in the interval $[0, 1]$.

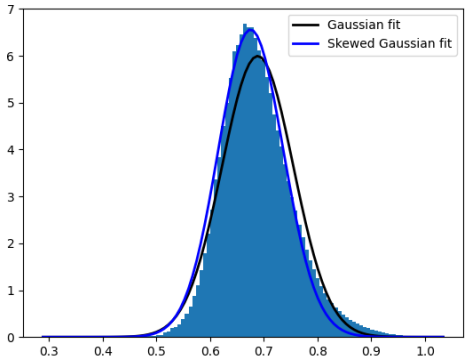


FIGURE 3: Logit Distribution Fitting

As seen in Figure 3, the logits can be reasonably well modeled using a normal distribution approximation. For our numerical experiments, we collected $\mu = 0.688$ and $\sigma = 0.066$, which we use extensively in the following tests. We note that while the following algorithms assume knowledge of the μ and σ of the logit distributions we are trying to steal, in practice, these parameters can be inferred by either transferring them from smaller models or estimating them from a small number of logit distributions already collected.

4.3 StartOverN With Normal Distribution Prior

Now we can formalize an algorithm using the normal distribution prior. We consider each logit as a random variable drawn from a normal distribution using the parameters μ and σ^2, which we have derived in previous section. At each step of the algorithm we bound the logit z_i using l_i and h_i meaning the logits are actually drawn from a truncated normal distribution over the interval $[l_i, h_i]$.

Formally, we note

$P_i \sim N(\mu, \sigma^2)$ truncated s.t. $l_i \leq P_i \leq h_i$ where $0 \leq l_i \leq h_i \leq 1$ and Φ is the CDF of $N(\mu, \sigma^2)$.

$$\Phi_{P_i}(x) = P(P_i \leq x) = \frac{\Phi(x) - \Phi(l_i)}{\Phi(h_i) - \Phi(l_i)}. \tag{1}$$

Theorem 3. *Given independent and identically distributed (i.i.d.) random variables $P_1, P_2, \ldots, P_{n-1}$ from a truncated normal distribution with truncation points l_i and h_i, and constants $b_1, b_2, \ldots, b_{n-1}$, the StartOverN algorithm aims to find b_i such that:*

$$P\left(\max(P_1 + b_1, P_2 + b_2, \ldots, P_{n-1} + b_{n-1}) \leq 1\right) = \frac{1}{n}$$

A solution to this problem is given by:

$$b_i = 1 - \Phi^{-1}\left(\left(\frac{1}{n}\right)^{\frac{1}{n-1}} (\Phi(h_i) - \Phi(l_i)) + \Phi(l_i)\right)$$

Proof. We need

$$\begin{aligned} P\left(\max(P_1 + b_1, \ldots, P_{n-1} + b_{n-1}) \leq 1\right) &= \frac{1}{n} \\ P(P_1 + b_1 \leq 1) \cdot \ldots \cdot P(P_{n-1} + b_{n-1} \leq 1) &= \frac{1}{n} \\ P(P_1 \leq 1 - b_1) \cdot \ldots \cdot P(P_{n-1} \leq 1 - b_{n-1}) &= \frac{1}{n} \\ \prod_{i=1}^{n-1} \frac{\Phi(1 - b_i) - \Phi(l_i)}{\Phi(h_i) - \Phi(l_i)} &= \frac{1}{n} \end{aligned}$$

This is satisfied if $\forall i$:

$$\begin{aligned} \frac{\Phi(1 - b_i) - \Phi(l_i)}{\Phi(h_i) - \Phi(l_i)} &= \left(\frac{1}{n}\right)^{\frac{1}{n-1}} \\ \Phi(1 - b_i) - \Phi(l_i) &= \left(\frac{1}{n}\right)^{\frac{1}{n-1}} (\Phi(h_i) - \Phi(l_i)) \\ \Phi(1 - b_i) &= \left(\frac{1}{n}\right)^{\frac{1}{n-1}} (\Phi(h_i) - \Phi(l_i)) + \Phi(l_i) \\ 1 - b_i &= \Phi^{-1}\left(\left(\frac{1}{n}\right)^{\frac{1}{n-1}} (\Phi(h_i) - \Phi(l_i)) + \Phi(l_i)\right) \\ b_i &= 1 - \Phi^{-1}\left(\left(\frac{1}{n}\right)^{\frac{1}{n-1}} (\Phi(h_i) - \Phi(l_i)) + \Phi(l_i)\right) \end{aligned}$$

Hence, $b_i = 1 - \Phi^{-1}\left(\left(\frac{1}{n}\right)^{\frac{1}{n-1}} (\Phi(h_i) - \Phi(l_i)) + \Phi(l_i)\right)$ is a solution. □

We have derived the bias term in similar fashion to the uniform prior. Again we write the vector notation of the biasing function as:

$$\mathbf{f}(\mathbf{l}, \mathbf{h}, r) = 1 - \Phi^{-1}\left(\left(\frac{1}{n}\right)^{\frac{1}{n-1}} (\Phi(\mathbf{h}) - \Phi(\mathbf{l})) + \Phi(\mathbf{l})\right).$$

The generalized form of the function for any distribution truncated on the interval $[a, b]$, where the CDF is defined as:

$$F_{[a,b]}(x) = \frac{F(x) - F(a)}{F(b) - F(a)}, \quad \text{for } a \leq x \leq b.$$

This can be analogously formulated as:

$$\mathbf{f}(\mathbf{l}, \mathbf{h}, r) = 1 - F^{-1}\left(\left(\frac{1}{n}\right)^{\frac{1}{n-1}} (F(\mathbf{h}) - F(\mathbf{l})) + F(\mathbf{l})\right)$$

This result can be useful when considering different distributional assumptions for the logit distribution, such as a skewed normal distribution, which can lead to further performance boosts.

4.4 EverythingOverN With Normal Distribution Prior

Recall the perfect algorithm formulated in Section 3.1. One way to interpret the biasing strategy of the algorithm is that we have adjusted all logits such that their probability of being sampled is $\frac{1}{n}$. In previous sections, we have demonstrated that under the assumption of truncated distributions, it is always possible to find a bias such that the probability of sampling the *top token* is $\frac{1}{n}$. We now turn our attention to extending this strategy to bias tokens such that each token has a probability of $\frac{1}{n}$ of being sampled. Specifically, we aim to set the bias vector such that the probability of sampling any given token is $\frac{1}{n}$, under the assumption that logits are drawn from a truncated normal distribution.

In this section we formalize the approach and provide an introduction of the different methods used to find a solution to the problem.

4.4.1 Formal definition

Suppose we have n different random variables $P_1, P_2, \ldots, P_n$. We are interested in finding shift constants $b_1, b_2, \ldots, b_n$ such that the order statistics of the shifted random variables, $P_1 + b_1, P_2 + b_2, \ldots, P_n + b_n$, ensure that the probability of any particular shifted random variable being the maximum is equal for all variables. Specifically, we want to ensure that

$$P\left(X_{(n)} = P_i + b_i\right) = \frac{1}{n}$$

for all $i = 1, 2, \ldots, n$, where $X_{(n)}$ denotes the maximum of the n shifted random variables.

Equivalently we can formulate the problem as find all r_i s.t.

$$\begin{aligned} P\left(\max_{j \neq i}(P_j + b_j) \leq P_i + b_i\right) &= \frac{1}{n} \quad \forall i \\ \prod_{j \neq i} P(P_j + b_j \leq P_i + b_i) &= \frac{1}{n} \quad \forall i \\ \prod_{j \neq i} P(P_j \leq P_i + b_i - b_j) &= \frac{1}{n} \quad \forall i \end{aligned}$$

As shown in Equation 2, the probability can be computed using the integral of the product of the PDFs.

$$\prod_{j \neq i} \frac{1}{\sigma^2 \left(\Phi\left(h_j\right) - \Phi\left(l_j\right)\right)\left(\Phi\left(h_i\right) - \Phi\left(l_i\right)\right)} \int_{l_i}^{h_i} \phi\left(y\right)\left(\Phi\left(y + b_i - b_j\right) - \Phi\left(l_j\right)\right) dy = \frac{1}{n} \quad \forall i$$

We note that the intervals $[l_i, h_i]$ can be disjoint, and any two disjoint intervals would result in the equation being equal to zero.

4.4.2 Approach to solution

By brute-force enumeration of all possible combinations of shifts, we have demonstrated that for up to four normally distributed random variables truncated on the interval $[0, 1]$, it is possible to find the shift constants such that the above equations hold. Our next

approach involved providing an initial guess, calculated by fixing one random variable and then taking the difference of means of the remaining variables as the guess. We define the loss as the ∞-norm of the difference between the computed probability of random variables being the maximum and the desired probability $1/n$. We experimented with two types of algorithms. First, we employed a gradient-based search on the loss, which showed promise but was prone to occasionally getting stuck in local minima. Second, we utilized a hyper-sphere random search to find an optimal solution.

It is important to note that the optimization problem has two bottlenecks that we have observed. First, it suffers from the curse of dimensionality, where the number of possible combinations of shifts grows exponentially with the number of variables. Second, in cases where we attempt to align many random variables truncated on very different intervals, an exact solution may, in fact, be impossible. This impossibility can be illustrated in extreme cases where we are faced with aligning random variables where some have orders of magnitude narrower bounds than others, causing the truncated distributions to be effectively very different from the initial distribution.

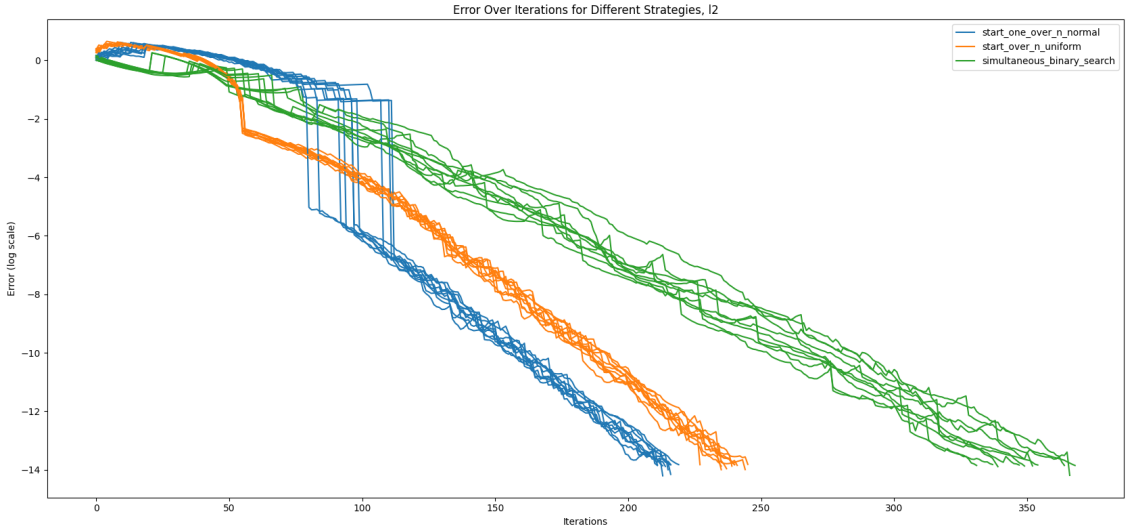
5 Numerical analysis

Notice that at each iteration of Algorithm 1, we update the lower bound $\mathbf{l}$ and upper bound $\mathbf{h}$. Upon reaching a specified stopping condition, we return both bounds as the best estimates for all logits. This approach implies that, up to this point, we have not made any concrete *guess* about the actual values of the logits. While a straightforward solution might involve taking the midpoint, i.e., $\frac{\mathbf{h}+\mathbf{l}}{2}$, as demonstrated in this section, this strategy may be suboptimal in scenarios where the attacker is constrained by a finite query budget T.

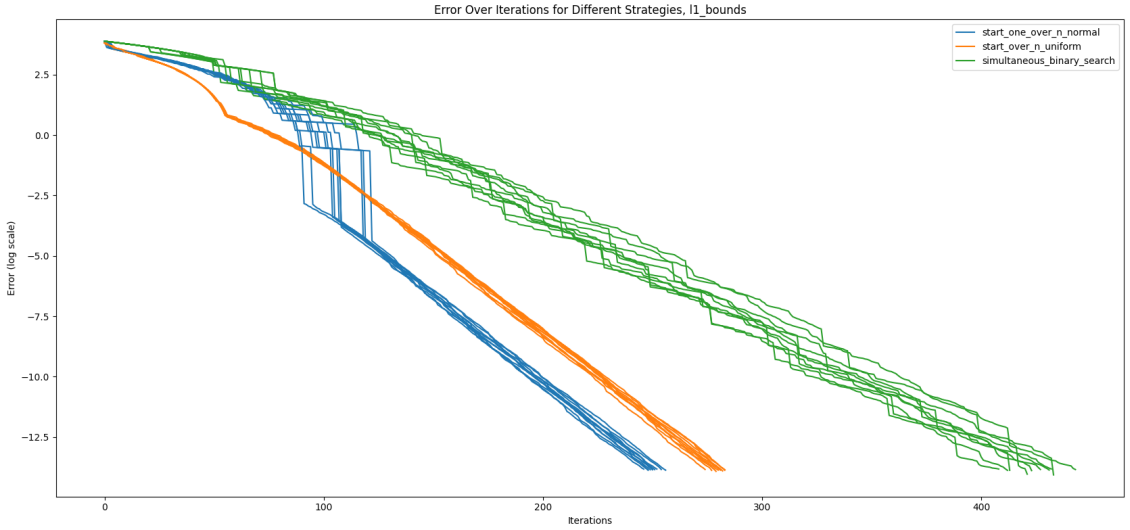
Definition 4. *Let $\mathbf{l}, \mathbf{h} \in \mathbb{R}^{N_b}$ be vectors representing the lower and upper bounds, respectively. A **guess** is a function $g : \mathbb{R}^{N_b} \times \mathbb{R}^{N_b} \to \mathbb{R}^{N_b}$ that maps $\mathbf{l}$ and $\mathbf{h}$ to a vector $\mathbf{g} \in \mathbb{R}^{N_b}$ such that $\mathbf{l} \leq \mathbf{g} \leq \mathbf{h}$ element-wise.*

We have already discussed *errors* that were defined as the $l_1 - norm$ and $l_\infty - norm$ of the bounds, that is $\|\mathbf{h} - \mathbf{l}\|_1$ and $\|\mathbf{h} - \mathbf{l}\|_\infty$ respectively. For testing purposes of how our algorithms are performing we can analogously define an error in terms of the true value of the logits z and our *guess*, similarly we consider $\|\mathbf{z} - \mathbf{g}\|_2$. In all results we also report on the performance of the simple *simultaneous-binary-search* algorithm, which is defined by the biasing function $\mathbf{f}(\mathbf{l}, \mathbf{h}, r) = 1 - \frac{\mathbf{l}+\mathbf{h}}{2}$.

Using $\|\mathbf{h} - \mathbf{l}\|_1$ and $\|\mathbf{z} - \mathbf{g}_m\|_2$, we evaluate the performance of the *simultaneous-binary-search*, *start-over-n-with-uniform-prior*, and *start-over-n-with-normal-distribution* algorithms on the 10 logit distributions we have collected from LLaMa-7b, as shown in Figure 4. We assess the algorithms using the mid guess, defined as $\mathbf{g}_m = \frac{\mathbf{h}+\mathbf{l}}{2}$, and plot the logarithm of the error. To illustrate an interesting phenomenon, we draw error lines for 10 different logit vectors for each algorithm, instead of reporting confidence intervals.



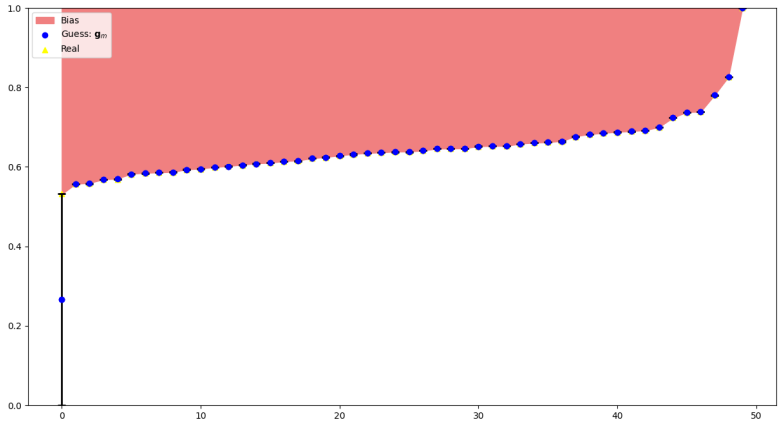
(A) $\|\mathbf{z} - \mathbf{g}_m\|_2$, $N_b = 50$



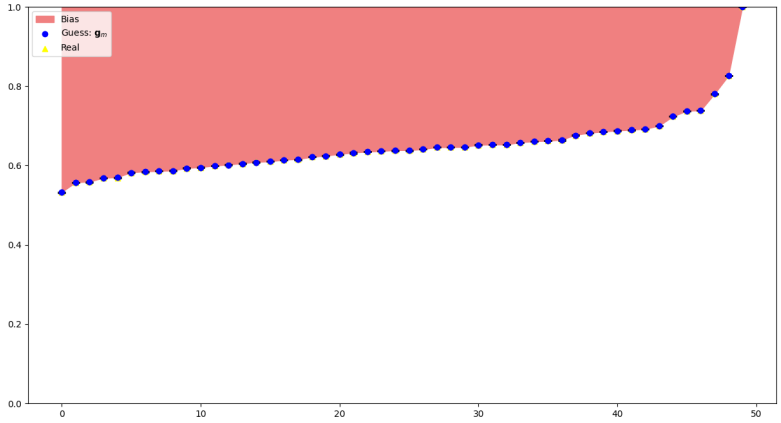
(B) $\|\mathbf{h} - \mathbf{l}\|_1$, $N_b = 50$

FIGURE 4: Visualization of algorithms performance

As observed in Figure 4, our new algorithm, *start-over-n-with-normal-distribution*, outperforms both the state-of-the-art *start-over-n-with-uniform-prior* and *simultaneous-binary-search*. Interestingly, the algorithm exhibits a 'jump' around the 100th iteration mark. To better understand this behavior, we inspect the bounds at the steps immediately before and after the jump, as shown in Figure 5.



(A) Example of logits at iteration 98 with blue $\mathbf{g}_m$



(B) Example of logits at iteration 99 with blue $\mathbf{g}_m$

FIGURE 5: Visualization of jump behaviour

As seen in the graphs in Figure 5, only the upper bound of the last logit, z_{last}, was updated before the jump. Referencing Observation 1, if a logit z_i has never been sampled, we can only update its upper bound h_i. This explains the observed jump. The algorithm spends the first 98 iterations before sampling the last token, z_{last}. Once the last token is sampled, it quickly establishes a precise lower bound, l_i. This behavior is analogous to the perfect algorithm described in Section 3.4, where at step $N_b - 1$, the lower bounds l_i of all logits are close to the true values z_i, and at step N_b, all upper bounds h_i are updated simultaneously.

This implies that for $\|\mathbf{h} - \mathbf{l}\|_1$, the error at iteration 98 is dominated by the difference $h_{last} - l_{last}$, and for $\|\mathbf{z} - \mathbf{g}_m\|_2$, by the difference $z_{last} - g_{m_{last}}$. Hence, we observe a dramatic change in the error graph.

One way to correct for jumps is to define a new *weird* error metric that uses a combination of bounds and the actual values.

$$weird(\mathbf{l}, \mathbf{h}, \mathbf{z}) = \sum_{i=1}^{N_b} \min(h_i - z_i, z_i - l_i)$$

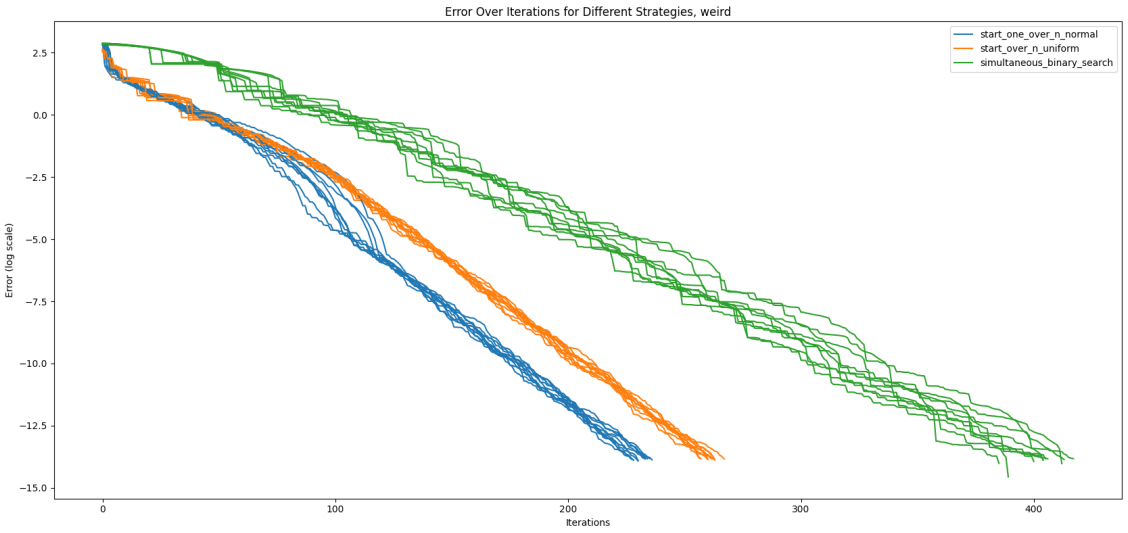


FIGURE 6: $weird(\mathbf{l}, \mathbf{h}, \mathbf{z})$, $N_b = 50$

As shown in Figure 6, using the newly defined error metric, the jumps have disappeared. While this error metric addresses the problem, it has a drawback: if the attacker has a low budget of queries T, our new attack would underperform until about the 100th iteration.

A slightly better approach would be to adjust our guess by leveraging our understanding of the reason for the jump. One general solution is to weight how far the lower bound l_i is from its initial value 0, and how far the upper bound h_i is from its initial value 1. This ensures that we are closer to the respective bound in proportion to how far we are from the initial value. We achieve this with the following definition.

$$\mathbf{m} = \frac{\mathbf{h} + \mathbf{l}}{2}$$

$$\mathbf{r} = \frac{\mathbf{h} - \mathbf{l}}{2}$$

$$\mathbf{g}_w = \mathbf{m} + \left(\frac{1 - \mathbf{h}}{\mathbf{l} + (1 - \mathbf{h})}\right)\mathbf{r} - \left(\frac{\mathbf{l}}{\mathbf{l} + (1 - \mathbf{h})}\right)\mathbf{r}$$

We start by centering our guess at the middle value of the bounds, and then weighting how far we have moved from the initial value: $1 - \mathbf{h}$ for the upper bound and $\mathbf{l}$ for the lower bound, divided by the total amount we have moved from the initial values, $\mathbf{l} + (1 - \mathbf{h})$. Using these weights, we scale the radius $\mathbf{r}$ of our interval.

When we examine the state of our *start-over-n-with-normal-prior* algorithm before sampling the last token, i.e., before the jump, we observe that the weighted guess is correctly

aligned with its true value. This is because, for tokens that have not been sampled yet, the weighted error equals the upper bound h_i, which means $\mathbf{g}_w = \mathbf{h}$ for all unsampled tokens.

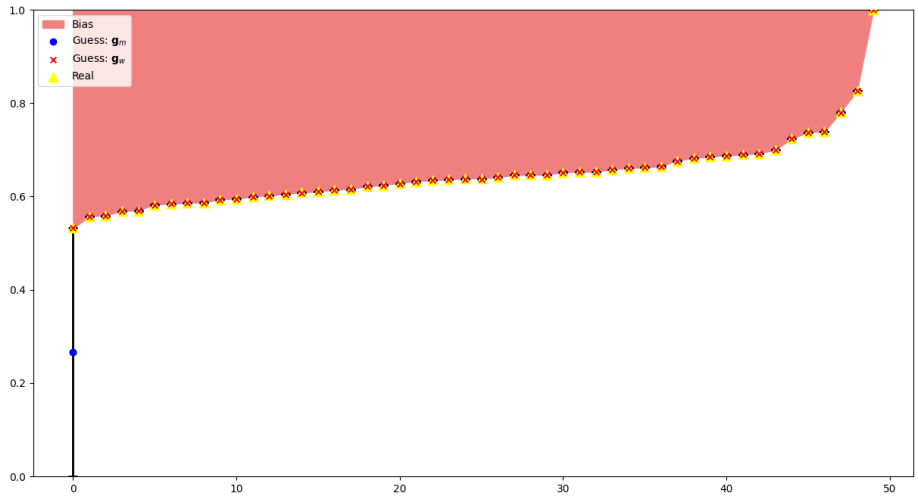


FIGURE 7: Example of logits at iteration 98 with blue $\mathbf{g}_m$ and red $\mathbf{g}_w$

Finally, we run the algorithms using the weighted guess $\mathbf{g}_w$ with $\|\mathbf{z} - \mathbf{g}_w\|_2$ to inspect its stability in Figure 8. Using the weighted guess we are now on par with *start-over-n-with-uniform-prior* during the first 100 iterations.

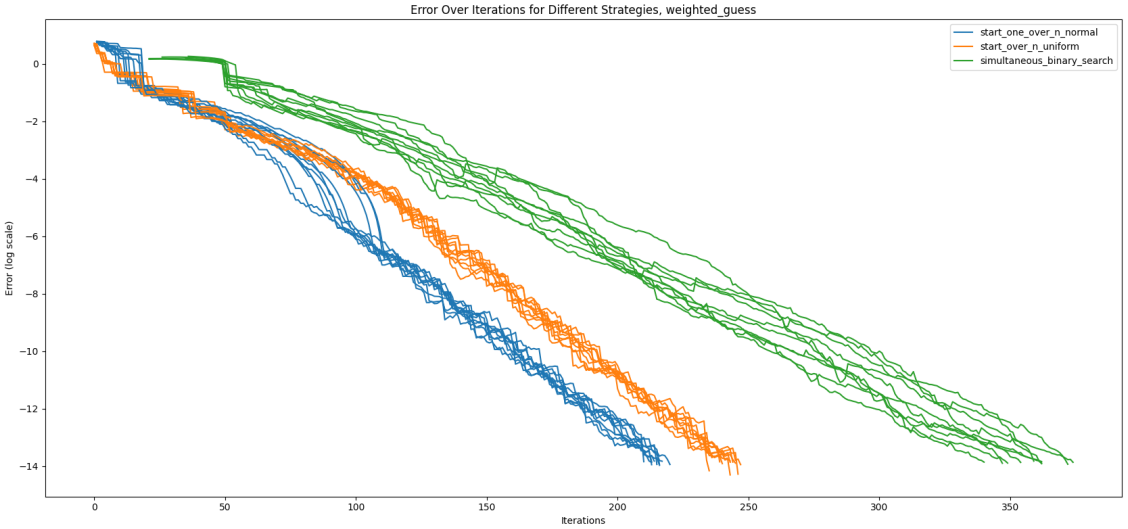


FIGURE 8: $\|\mathbf{z} - \mathbf{g}_w\|_2$, $N_b = 50$

We hypothesize that error jumps are an essential characteristic of highly performant algorithms, as they indicate precise boundary estimation.

5.1 Performance

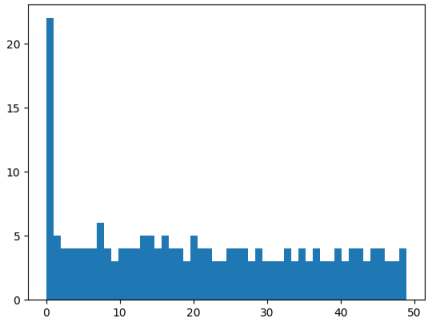
We evaluate the performance of various algorithms designed to extract logit distributions from large language models. We compare the algorithms based on the number of queries required to achieve a specified precision and report on the queries per logit metric. We measure their performance using the $l2$-norm of the difference between the true logit vector $\mathbf{z}$ and the weighted guess $\mathbf{g}_w$ by $\|\mathbf{z} - \mathbf{g}_w\|_2$.

Name	**Bias Function $\mathbf{f}(\mathbf{l}, \mathbf{h})$**	**Queries per logit**
Simultaneous Binary Search	$1 - \frac{\mathbf{l}+\mathbf{h}}{2}$	3.12
Start Over N with Uniform Prior	$1 - \mathbf{l} + \left(\frac{1}{n}\right)^{\frac{1}{n-1}} (\mathbf{h} - \mathbf{l})$	2.42
Start Over N with Normal Prior	$1 - \Phi^{-1}\left(\left(\frac{1}{n}\right)^{\frac{1}{n-1}} (\Phi(\mathbf{h}) - \Phi(\mathbf{l})) + \Phi(\mathbf{l})\right)$	**1.98**

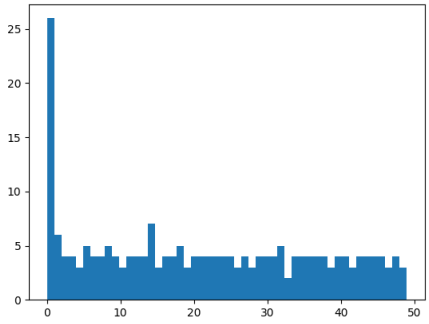
TABLE 1: Comparison of Algorithms with Precision 10^{-6} and $\mathbf{g}_w$

5.2 Frequency analysis

Interestingly, upon plotting the logit sampling frequency at the conclusion of the StartOverN algorithms, we observed that both the uniform prior and normal prior algorithms exhibit a higher sampling frequency for the first token compared to the other tokens. This indicates that, in practice, the algorithms do not achieve the intended biasing of the top token to have a $1/n$ probability of being sampled.



(A) Logit sampling frequency at the end of Start Over N with Normal Prior with precise Normal logit data



(B) Logit sampling frequency at the end of Start Over N with Normal Prior with Actual logit data

6 Future work

6.1 SomeOverN with Normal prior

In the StartOverN algorithm, we adjusted each token's probability such that the token with the highest logit value, denoted as token 0, would be sampled with a probability of $\frac{1}{n}$. This simplification was feasible because we set the highest logit to a constant value, thereby avoiding the need to draw it from a distribution. In this section, we extend the problem to a more general case where the logit values are sampled from a truncated normal distribution within the interval $[l_i, h_i]$.

Consider independent and identically distributed (i.i.d.) random variables $P_1, P_2, \ldots, P_{n-1}$ from a truncated normal distribution with truncation points l_i and h_i, and constants $r_1, r_2, \ldots, r_{n-1}$. The objective of the Some $1/n$ algorithm is to find r_i such that:

$$P\left(\max(P_1 + r_1, P_2 + r_2, \ldots, P_{n-1} + r_{n-1}) \leq P_i + r_i\right) = \frac{1}{n}$$

We seek this solution by setting r_i such that:

$$P(P_i + r_i \leq P_n) = P(P_i \leq P_n - r_i) = \int_{l_i}^{h_i} F_i(y - r_i) f_n(y)\, dy = \left(\frac{1}{n}\right)^{\frac{1}{n-1}}$$

This can be naively achieved by iterating over all $r_i \in [l_n - h_i, h_n - l_i]$.

However, this approach encounters difficulties. Setting $P(P_i + r_i \leq P_n) = \left(\frac{1}{n}\right)^{\frac{1}{n-1}}$ does not ensure that the maximum value is equal to $\frac{1}{n}$, indicating that the variables are not independent. This dependency is not immediately apparent, even when considering papers that compute the Kullback-Leibler divergence (D_{KL}) of the distributions. Despite this, it seems plausible that a solution exists, as we do not require other distributions to have the same probability of being the maximum.

6.2 Frequency Exhaustion Algorithm with Temperature and Frequency Penalty

A novel algorithm that does not rely on the bias map can be formulated as follows:

Set High Temperature: Use the highest possible temperature to ensure a more uniform probability distribution. This increases the likelihood of sampling less probable tokens.

Create Special Prompts: Design prompts that contain tokens likely to be top tokens in the distribution (e.g., "aaaaa" will likely have "a" as a high-probability token). Apply frequency penalties to these top tokens to negatively bias them, thereby increasing the visibility of tokens with lower probabilities.

Repeated Prompting: Prompt the model X times with the same prompt sequence to gather a comprehensive set of token outputs.

Collect Frequencies: Record the frequencies of different tokens from the X repetitions. This frequency list serves as an approximation of the logit distribution.

Solve for True Logits: Use the frequency list obtained from the high-temperature sampling to infer the true logits of the model.

7 Conclusion

Model stealing, involves replicating a machine learning model's functionality without direct access to its internal parameters. This is particularly concerning for large language models (LLMs) deployed as black-box systems via APIs, where attackers can exploit the model's outputs to infer its underlying structure. The ability to extract full logit distributions, which represent the raw, unnormalized scores for each token, is a critical step in reverse-engineering these models.

In this thesis, we have explored various algorithms for extracting logit distributions from large language models, focusing on the challenges posed by the curse of dimensionality and the need for efficient query strategies. Our numerical analysis demonstrated the effectiveness of the *start-over-n-with-normal-distribution* algorithm, which outperformed existing methods such as *start-over-n-with-uniform-prior* and *simultaneous-binary-search*. We introduced a novel error metric, $weird(\mathbf{l}, \mathbf{h}, \mathbf{z})$, to address the issue of error jumps, and proposed a weighted guess strategy to improve performance under constrained query budgets. Our findings suggest that while error jumps are indicative of precise boundary guessing, they can be mitigated through careful algorithm design. Future work could explore more sophisticated biasing functions and alternative error metrics to further enhance the accuracy and efficiency of logit extraction algorithms.

A $P(X \leq Y)$ where X, Y from truncated normal distribution

Assume X and Y are independent random variables from the same normal distribution $N(\mu, \sigma^2)$, but truncated to different intervals:

- X is truncated to the interval $[l_X, h_X]$.
- Y is truncated to the interval $[l_Y, h_Y]$.

For a truncated normal distribution X with bounds $[a, b]$, the probability density function (PDF) is:

$$f_X(x) = \frac{\phi\left(\frac{x-\mu}{\sigma}\right)}{\sigma\left(\Phi\left(\frac{b-\mu}{\sigma}\right) - \Phi\left(\frac{a-\mu}{\sigma}\right)\right)}$$

where $\phi(\cdot)$ is the PDF and $\Phi(\cdot)$ is the cumulative distribution function (CDF) of the standard normal distribution.

To compute $P(X \leq Y)$:

$$P(X \leq Y) = \int_{l_Y}^{h_Y} \int_{l_X}^{y} f_X(x) f_Y(y)\, dx\, dy$$

Substituting the PDFs of X and Y:

$$P(X \leq Y) = \int_{l_Y}^{h_Y} \int_{l_X}^{y} \frac{\phi\left(\frac{x-\mu}{\sigma}\right)}{\sigma\left(\Phi\left(\frac{h_X-\mu}{\sigma}\right) - \Phi\left(\frac{l_X-\mu}{\sigma}\right)\right)} \cdot \frac{\phi\left(\frac{y-\mu}{\sigma}\right)}{\sigma\left(\Phi\left(\frac{h_Y-\mu}{\sigma}\right) - \Phi\left(\frac{l_Y-\mu}{\sigma}\right)\right)} dx\, dy$$

This can be simplified by factoring out the normalization constants:

$$P(X \leq Y) = \frac{1}{\sigma^2\left(\Phi\left(\frac{h_X-\mu}{\sigma}\right) - \Phi\left(\frac{l_X-\mu}{\sigma}\right)\right)\left(\Phi\left(\frac{h_Y-\mu}{\sigma}\right) - \Phi\left(\frac{l_Y-\mu}{\sigma}\right)\right)} \int_{l_Y}^{h_Y} \phi\left(\frac{y-\mu}{\sigma}\right) \left(\int_{l_X}^{y} \phi\left(\frac{x-\mu}{\sigma}\right) dx\right) dy$$

Evaluate the inner integral with respect to x:

$$\int_{l_X}^{y} \phi\left(\frac{x-\mu}{\sigma}\right) dx = \int_{l_X}^{y} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx$$

This is the CDF of the normal distribution, evaluated from l_X to y:

$$\int_{l_X}^{y} \phi\left(\frac{x-\mu}{\sigma}\right) dx = \Phi\left(\frac{y-\mu}{\sigma}\right) - \Phi\left(\frac{l_X-\mu}{\sigma}\right)$$

Now substitute back into the outer integral:

$$P(X \leq Y) = \frac{1}{\sigma^2\left(\Phi\left(\frac{h_X-\mu}{\sigma}\right) - \Phi\left(\frac{l_X-\mu}{\sigma}\right)\right)\left(\Phi\left(\frac{h_Y-\mu}{\sigma}\right) - \Phi\left(\frac{l_Y-\mu}{\sigma}\right)\right)} \int_{l_Y}^{h_Y} \phi\left(\frac{y-\mu}{\sigma}\right) \left(\Phi\left(\frac{y-\mu}{\sigma}\right) - \Phi\left(\frac{l_X-\mu}{\sigma}\right)\right) dy$$

Assuming PDF and CDF defined in terms of the correct μ and σ:

$$P(X \leq Y) = \frac{1}{\sigma^2 \left(\Phi\left(h_X\right) - \Phi\left(l_X\right)\right)\left(\Phi\left(h_Y\right) - \Phi\left(l_Y\right)\right)} \int_{l_Y}^{h_Y} \phi\left(y\right)\left(\Phi\left(y\right) - \Phi\left(l_X\right)\right) dy$$

For some constant r, relevant to your needs, this becomes:

$$P(X \leq Y + r) = \frac{1}{\sigma^2 \left(\Phi\left(h_X\right) - \Phi\left(l_X\right)\right)\left(\Phi\left(h_Y\right) - \Phi\left(l_Y\right)\right)} \int_{l_Y}^{h_Y} \phi\left(y\right)\left(\Phi\left(y + r\right) - \Phi\left(l_X\right)\right) dy \tag{2}$$

Alternatively, knowing both the PDF and CDF of the truncated distributions for X and Y, we can write the equation as follows:

$$P(X \leq Y + r) = \int_{-\infty}^{\infty} F_X(y + r) f_Y(y)\, dy$$

$$P(X \leq Y + r) = \int_{l_Y}^{h_Y} F_X(y + r) f_Y(y)\, dy$$

$$P(P_i \leq P_n - r_i) = \int_{l_i}^{h_i} F_i(y - r_i) f_n(y)\, dy = 1/n$$

References

Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., Shakeri, S., Taropa, E., Bailey, P., Chen, Z., Chu, E., Clark, J. H., Shafey, L. E., Huang, Y., Meier-Hellstern, K., Mishra, G., Moreira, E., Omernick, M., Robinson, K., ... Wu, Y. (2023, September). PaLM 2 Technical Report [arXiv:2305.10403 [cs]]. https://doi.org/10.48550/arXiv.2305.10403

Carlini, N., Paleka, D., Dvijotham, K. D., Steinke, T., Hayase, J., Cooper, A. F., Lee, K., Jagielski, M., Nasr, M., Conmy, A., Wallace, E., Rolnick, D., & Tramèr, F. (2024, March). Stealing Part of a Production Language Model [arXiv:2403.06634 [cs]]. Retrieved May 1, 2024, from http://arxiv.org/abs/2403.06634

Finlayson, M., Ren, X., & Swayamdipta, S. (2024, March). Logits of API-Protected LLMs Leak Proprietary Information [arXiv:2403.09539 [cs]]. https://doi.org/10.48550/arXiv.2403.09539

Morris, J. X., Zhao, W., Chiu, J. T., Shmatikov, V., & Rush, A. M. (2023, November). Language Model Inversion [arXiv:2311.13647 [cs]]. https://doi.org/10.48550/arXiv.2311.13647

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., ... Zoph, B. (2024, March). GPT-4 Technical Report [arXiv:2303.08774 [cs]]. https://doi.org/10.48550/arXiv.2303.08774

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023, August). Attention Is All You Need [arXiv:1706.03762 [cs]]. https://doi.org/10.48550/arXiv.1706.03762