Supplementary Material for Baselines for Identifying Watermarked Large Language Models

A. Proof of Theorem 3.8

Proof. Let f_s be a pseudorandom function, which exists as one-way functions exist. Consider the Kirchenbauer watermark W_s that generates a pseudorandom number $r \in [0, 1]$ by applying f_s to the previous tokens. The next token is then chosen by using r to select the next token from the LLM logits.

This is strong quality-preserving, as otherwise an adversary that could distinguish a watermarked from unmarked language model could be used to distinguish f_s from a random function. Since W_s is deterministic for any given seed, it can be detected by rerunning the watermarked LLM and observing if it returns the same output. □

B. Proof of Theorem 3.10

Proof. Consider the distribution of next-token probabilities for 0 in a text.

If the detector correctly distinguishes between positive and negative distributions is at most $\frac{1}{2} + p$, we can use the Sadasivan bound from to bound the total variation distance between R^n and $W(R)^n$:

$$\frac{1}{2} + p \leq \frac{1}{2} + \text{TV}(R^n, W(R)^n) - \frac{\text{TV}(R^n, W(R)^n}{2}$$

$$\Rightarrow \text{TV}(R^n, W(R)^n) \geq 1 - \sqrt{1-p}$$

Suppose that the adversary has the ability to not only sample the generator, but also obtain its probabilities for the next token. Consider the average variation distance from uniform of the next token from the watermarked generator, over a uniformly random in $(0, \ldots, n-1)$ number of uniformly randomly generated previous tokens. By the subadditivity of the total variation measure, the average variation distance must be at least $\frac{1-\sqrt{1-p}}{n}$. Since it is bounded in $[0, 0.5]$, at least $\frac{2-2\sqrt{1-p}}{n}$ of the sampled probabilities must be at least $\frac{1-\sqrt{1-p}}{n}$. To ensure that with probability $1 - \Delta/2$ the adversary has sampled at least one such probability, it must take at least m samples, with $(1 - \frac{2-2\sqrt{1-p}}{n})^m \leq \Delta/2$, and so

$$m \leq \frac{\log(\Delta/2)}{\log(1 - \frac{2-2\sqrt{1-p}}{n})}$$

Since the adversary cannot sample probabilities, it must repeatedly sample a certain token. Let k be the number of samples it takes from each particular token, and let the adversary classify the sample depending on whether the proportion of '0' generations differs from $1/2$ by at least q. Using the two-sided Chernoff bounds, we can get the probability of any particular sample from the uniform generator being misclassified. We then use union bounds to get the total probability of the uniform generator being misclassified, and use it to get bounds on k and q:

$$2m \exp(-k((0.5+q)\log(1+2q) + (0.5-q)\log(1-2q))) \leq \Delta$$

We use a similar method to obtain the probability that a token with variation from uniform $v = \frac{1-\sqrt{1-p}}{n}$ avoids detection:

$$\exp(-k((0.5+q)\log(\frac{0.5+q}{0.5+v}) + (0.5-q)\log(\frac{0.5-q}{0.5-v}))) \leq \Delta/2$$

To get the q which requires the fewest samples, we set these bounds to be equal. Doing this, we get a detection algorithm polynomial that takes $O(n \log(\frac{n}{pv\Delta} \log(\frac{1}{\Delta})))$ which correctly classifies both the random and any watermarked generator with probability at least $1 - \Delta$.

□

C. Understanding Language Model Output and Probability Distributions

A watermark can be characterized and detected by how it affects the logits distribution of the underlying LLM. As such, our algorithms for watermark detection are centered heavily on analyzing shifts in language model output as well as logit and probability distributions. Therefore, it is critical to gain intuition for how these distributions usually behave.

C.1. Random Bit Generation

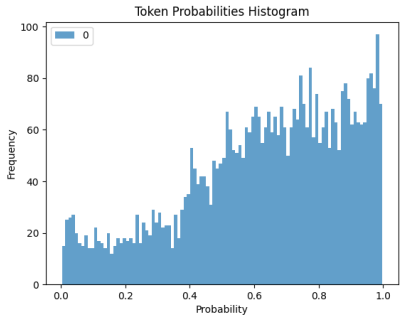
A seemingly simple case casts LLMs as random bit generators. Ideally, a LLM can generate bits uniformly at random when prompted, and so the identification mechanism in Theorem 3.10 would apply. We attempted random bit generation with OpenAI models. For each model, we use the following prompt:

```
"""Choose two digits, and generate a
uniformly random string of those digits.
Previous digits should have no influence
on future digits:"""
```

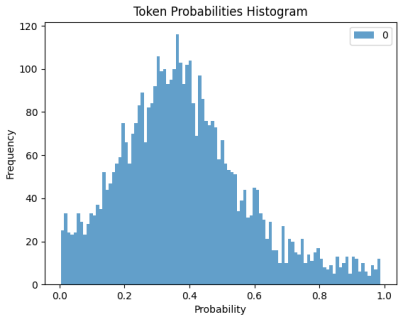
This prefix was followed by a fixed sequence of 20 '0's and '1's, produced by a Python random number generator.

We let each model generate 100 tokens. For each token, if '0' and '1' were both a top-5 likely token, the probability of generating '0' was recorded. This procedure was repeated across multiple generations. A graph of the recorded probabilities for each model is displayed in Figure 1.

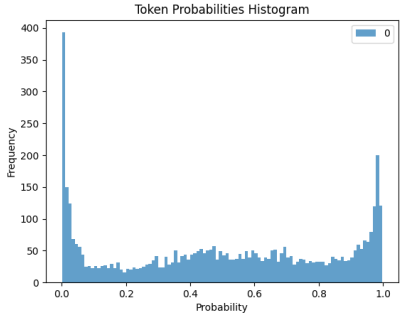
These distributions fail tests for normality and unsurprisingly, the corresponding generated bits are far from uniform. Surprisingly, the qualitative output probability distributions of each model are strikingly different. Ada (1a) produces a roughly monotonically increasing distribution, babbage (1b) produces a roughly truncated normal distribution, curie (1c) produces a distribution with probability mass concentrated around 0 and 1, and davinci (1d) produces a trimodal distribution with peaks around 0, 0.5, and 1.



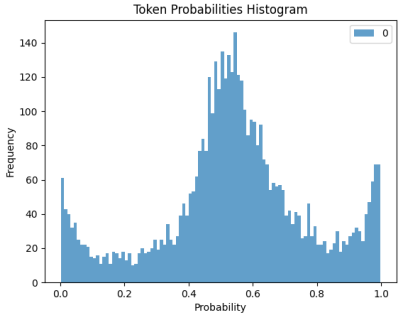
(a) Ada probabilities of generating 0.



(b) Babbage probabilities of generating 0.

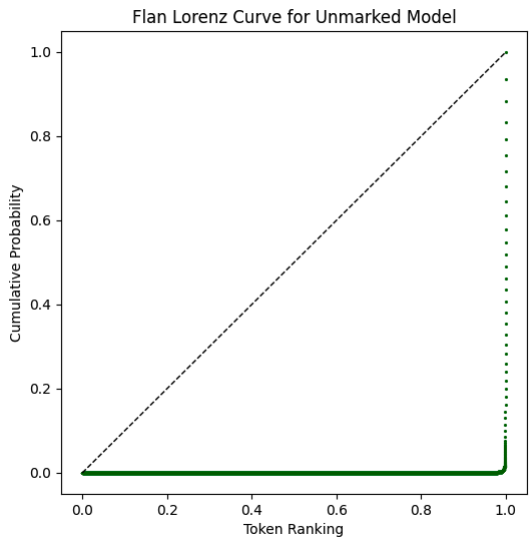


(c) Curie probabilities of generating 0.

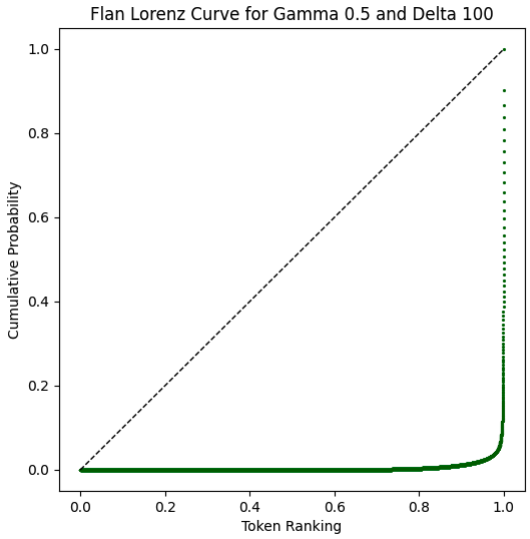


(d) Davinci probabilities of generating 0.

Figure 1. Comparison of OpenAI engine behavior on a simple random bit generation task. The x-axis displays the retrieved probability of generating 0, and the y-displays the frequency of each probability bucket over multiple generations.



(a) Lorenz curve for a unmarked Flan-T5-XXL language model. Most of the probability mass is concentrated in a few top tokens, as visualized by the sharp spike towards the right of the Lorenz curve.



(b) Lorenz curve for a Flan-T5-XXL model affected by a Kirchenbauer watermark with parameters $\gamma = 0.5$ and $\delta = 100$. Notice that the Lorenz curve is slightly smoother under this setting, due to the δ application on low-probability tokens.

Figure 2. Examples of ranked probability Lorenz curves of the first token generated by Flan-T5-XXL under different Kirchenbauer watermarking strengths. The dashed line represents a perfectly uniform distribution. In both watermarking settings, the majority of the probability mass is concentrated in the top few tokens.

C.2. Ranked Probability Lorenz Curves

A ranked probability Lorenz curve is constructed with x-axis listing tokens sorted from lowest to highest probability, and the y-axis displaying the probabilities of each token. Due to the sorted construction of the x-axis, the ranked token Lorenz curve is monotonically increasing.

The Lorenz curve is an effective tool for understanding the effects of a watermark from **?**. Such a watermark adds a constant term δ to a randomly selected subset of green list token logits. In the ranked token Lorenz curve, this is notably reflected by a smoothing effect, as seen on the right of Figure 2. This indicates that a portion of low-probability tokens have experienced a δ-increase.

To rigorize this notion of smoothness, one can compute the Gini coefficient G of the Lorenz curve:

$$G = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} |x_i - x_j|}{2n^2 \overline{x}}$$

Here x_i, x_j are the probabilities of i-th and j-th tokens on the curve, indexed by the ordered ranking, and $\overline{x}$ is the average probability. Traditionally in economics, G is used to measure the inequality of a distribution. High G suggests more inequality, reflected in unmarked distributions, while low G suggests less inequality and a smoother distribution, suggesting the presence of a watermark.

Recovering Logits from Sampling In practice, exact logits may not be available for analysis, for example when interacting with ChatGPT. In this case, we approximate token probabilities by sampling a large number of tokens from a language model, and calculating empirical probabilities.

C.3. Random Number Generation

In the case of a publicly hosted API, oftentimes logit data is not directly accessible. As a suitable approximation, we instead consider the distribution of tokens from a small subset of the original vocabulary. This enables us to analyze the shifting

behavior of a LLM before and applying a watermark, without requiring access to output logits.

Specifically, we treat LLMs as random number generators, asking them to generate integers from 1 to 100, inclusive. Figure 3 displays an example 10,000-sample distribution from Alpaca-LoRA using the following prompt:

```
"""Below is an instruction that
describes a task. Write a response that
appropriately completes the request.

### Instruction:
Generate a random number between
1 and 100.

### Response:"""
```

While this is a natural task to restrict the output token set of a model, it is certainly not the only task that would do so. For example, asking a LLM to provide a synonym for a given input word, such as “intelligent”, that starts with a specific letter, such as “c”, would also severely restrict the output distribution to a subset resembling something like {“clever”, “canny”, “crafty”, “calculating”, “cunning”}.

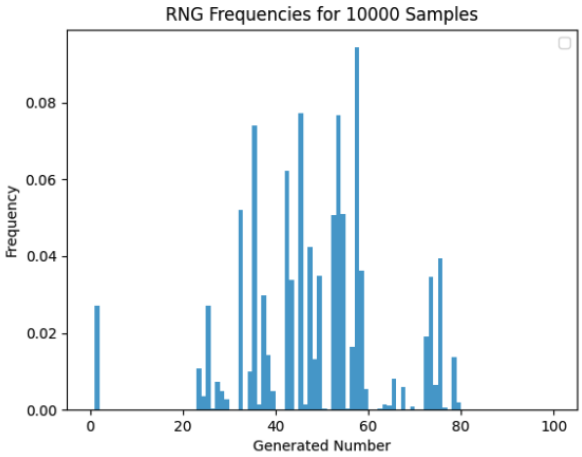


Figure 3. Example of a 10,000-sample RNG distribution generated by Alpaca-LoRA. Clearly, the distribution is far from uniform and exhibits idiosyncratic generations resulting from the training set.

A key benefit of the random number generation task over other alternatives, however, is that the output space for any model is fairly consistent between models, generating integers between 1 and 100, regardless of model capacity. While the distribution of numbers is certainly expected to change across models, the range of outputs is relatively more stable.