- Robust Distortion-free Watermarks for Language Models

> https://arxiv.org/abs/2307.15593

加水印的方法 generator

- 准备阶段
  - 用一个key生成624个int_32的随机数序列W
  - 根据这个随机数序列生成一个[n, vocab_size]的tensor：xi，tensor的每一个元素都是从随机序列W中随机挑选的一个值
    - 和prob做操作的备选tensor集合，prob做操作的时候就是从n簇随机数中选择一个做操作
  - 从[0, n)中随机一个shift
- LLM生成文本阶段
  - 自回归生成new tokens，每生成一个new token的时候干预它的打分表logits
  - logits过softmax得到probs，shape：[1, vocab_size]
  - 从[n, vocab_size]中取一簇[1, vocab_size]形状的向量u，具体取哪簇根据(shift+i)%n

    注意：虽然因为shift的随机性，这个文本第一个选的簇是随机的；但是后面选的簇都是连续的；比如第一簇选的是i，第二簇就是i+1，一直往后排
  - vocab_size的每个维度用u**(1/prob)计算，取argmax决定到底选哪个token

检测水印的方法 detector

- 用key把随机序列W再计算一遍（和generator的完全一样）
- 根据这个随机数序列生成一个[n, vocab_size]的tensor：xi，tensor的每一个元素都是从随机序列W中随机挑选的一个值
  - **这里的xi和generator的xi完全一样**
- levenshtein函数：（**编辑距离**）
  - 编码后的文本和xi取k簇（连续的k簇）挨个算相异度，取最低值（取相异度最低的值作为返回结果，即找到一个连续的k簇使得它们的相似度最高）

    其实是遍历一遍备选，希望找到generator里到底是用的哪连续k簇
- 随机修改xi n_runs次，即生成n_runs（default=500）的xi_alternative，再用levenshtein计算相异度；因为这次是完全随机生成，所以按道理来讲，加了水印的话应该完全随机生成xi之后相异度变高了。

```
# assuming lower test values indicate presence of watermark
p_val += null_result <= test_result
```

感觉这个本质是"最大相似子序列长度之和匹配"