# Fast-DetectGPT

## Understanding sampling-based zero-shot AI-generated Text Detection

Krystof Bobek
Mara Dragomir
Mahdi Rahimi
Jose Garcia
Salvador Torpes

# Table of Contents

# Introduction

# The Task

Classifying input text as human or AI-generated

With roots in blues rock and psychedelic rock, the bands that created heavy metal developed a thick, massive sound, characterized by highly amplified distortion, extended guitar solos, emphatic beats, and overall loudness.

Which example is human and which is generated by a LLM?

Metal music is a genre of rock music that typically features heavy, distorted guitar riffs and fast-paced tempos. It often incorporates elements of other musical genres such as punk, hardcore, and classical music, and is known for its aggressive, energetic sound.

# The Task

Classifying input text as human or AI-generated

Human

With roots in blues rock and psychedelic rock, the bands that created heavy metal developed a thick, massive sound, characterized by highly amplified distortion, extended guitar solos, emphatic beats, and overall loudness.

LLM

Metal music is a genre of rock music that typically features heavy, distorted guitar riffs and fast-paced tempos. It often incorporates elements of other musical genres such as punk, hardcore, and classical music, and is known for its aggressive, energetic sound.

Difficult task for humans, but maybe not so much for AI

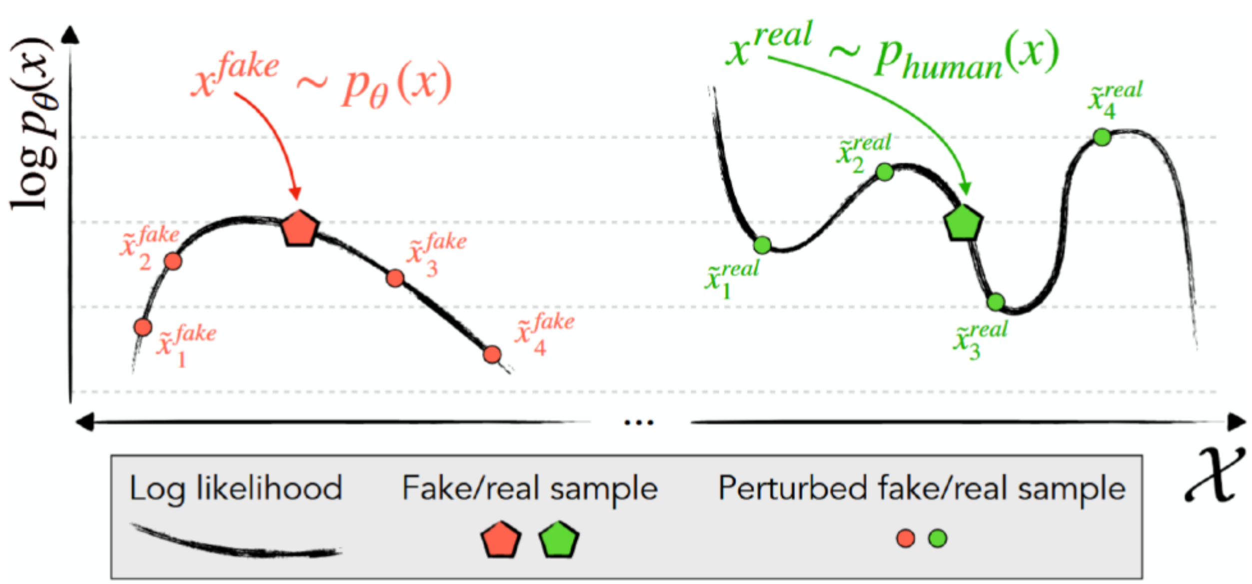
# Motivation

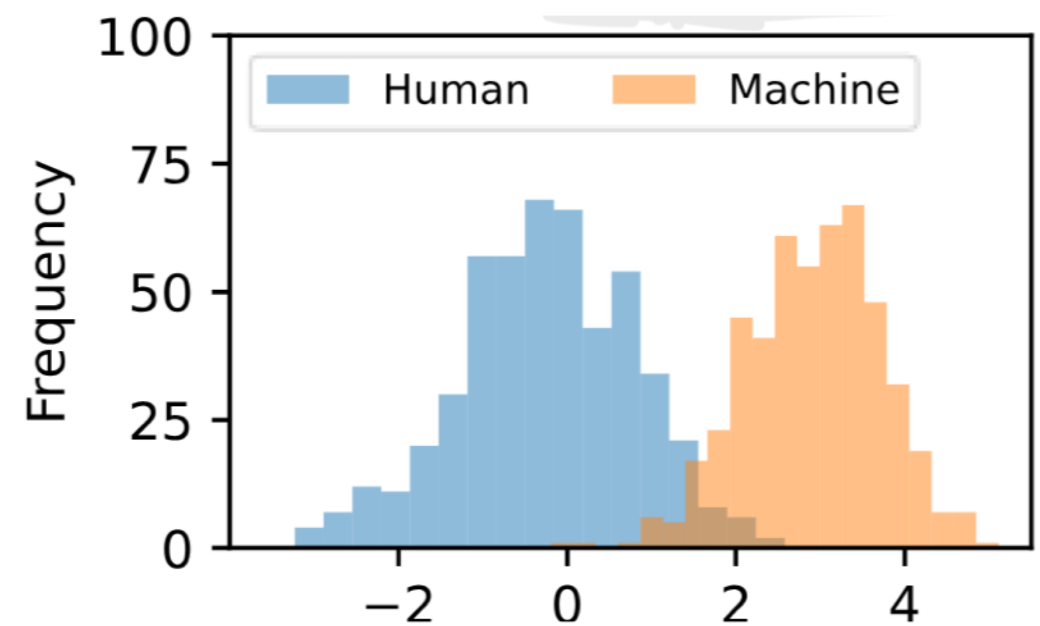
Why is it important to detect AI-generated content?

- Preventing mis- and disinformation
- Preventing plagiarism
- Transparency
- Preserving human creativity and intellect

# The Idea

DetectGPT & Fast DetectGPT




The probability curvature of a text sample fed to a model can show whether the text is written by a human or by the model itself

# Methodology

# Models

# Models

FastDetectGPT uses three different Large Language Models in its pipeline:

**Source Model**

Source model is the one used to generate the datasets: we feed the model with the first 30 tokens of a human written sentence and then we do decoding up until we have a sentence that is the size of the original one.

**Sampling Model**

Sampling model is the one used to get the probability distribution of tokens over each position in the sentence that will then be used to sample from and generate the 'perturbations' of the sentence

**Scoring Model**

Scoring model is the one used to get the probability distribution of tokens over each position in the sentence that will then be used to compute the loss of the sentence

# Models

FastDetectGPT uses three different Large Language Models in its pipeline:

**Source Model**

- Used for dataset creation
- Takes a collection of human sentences as input, and generates a corresponding AI sentence for each using the first 30 tokens of the sentence

**Sampling Model**

- Used for perturbing the input sentences
- Samples from the probability distribution of tokens over each position in the sentence

**Scoring Model**

- Used for the probability curvature
- Uses the probability distribution of tokens over each position in the sentence to compute the loss
- Classifies the sentence via a threshold

# Settings

The experiments of FastDetectGPT can be done under a black box or a white box setting.

**Black Box**

An experiment under the black box setting is defined by having **different** source and scoring models - real life setting where we don't know it the text was AI generated

**White Box**

An experiment under the black box setting is defined by having the **same** source and scoring models - we know the model we are trying to detect

In both setting sampling and scoring model may differ and that might lead to performance changes.

# Settings

The experiments of FastDetectGPT can be done under a black box or a white box setting.

(Our Focus)

**Black Box**

where the source and scoring models can be different (the model doesn't know what model generated the text)

**White Box**

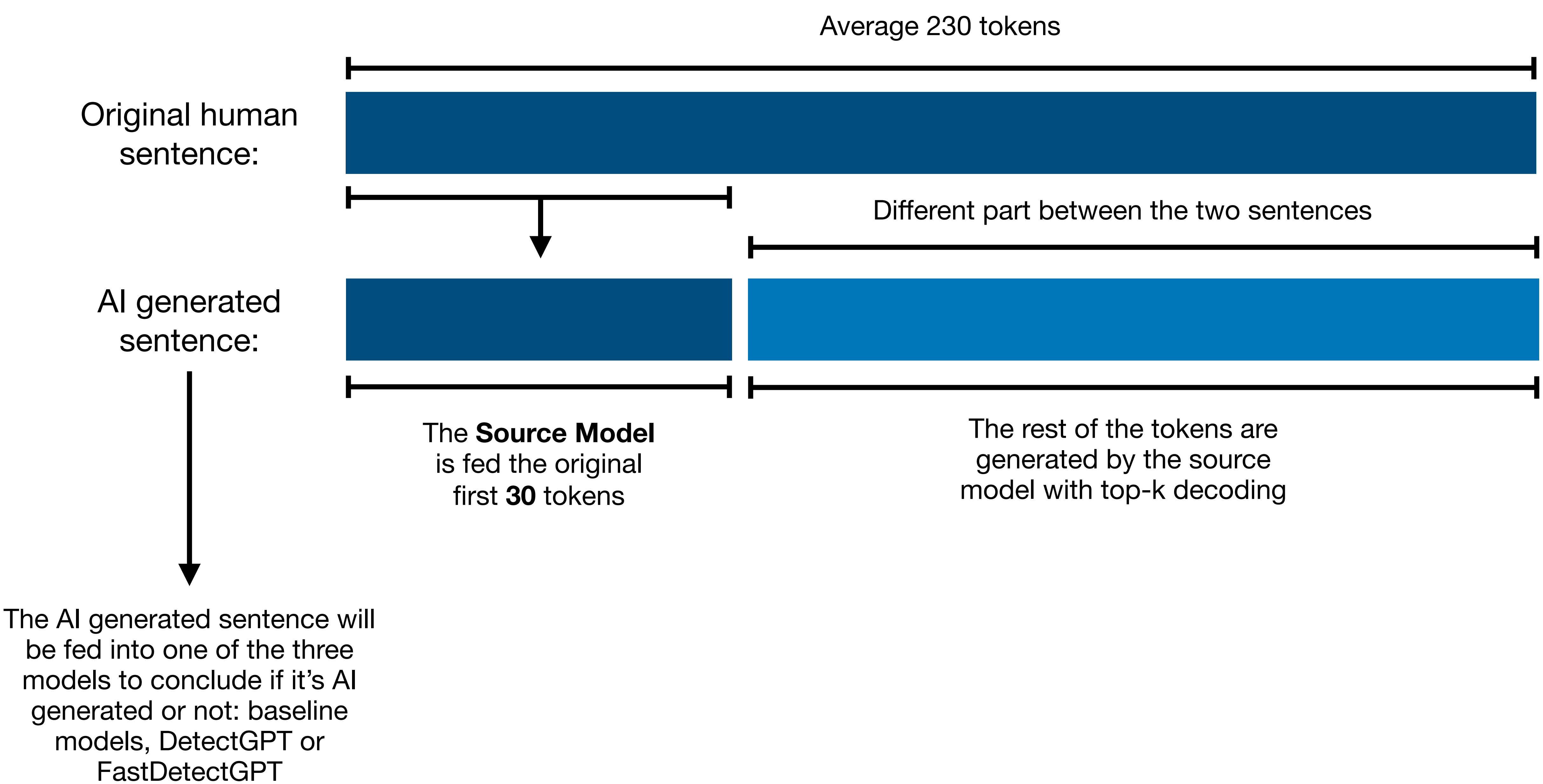
where the source and scoring models are the same (the model classifies whether a text is generated by itself)

The sampling and scoring model may differ and that might lead to performance changes

# Dataset Generation

# Generating AI text

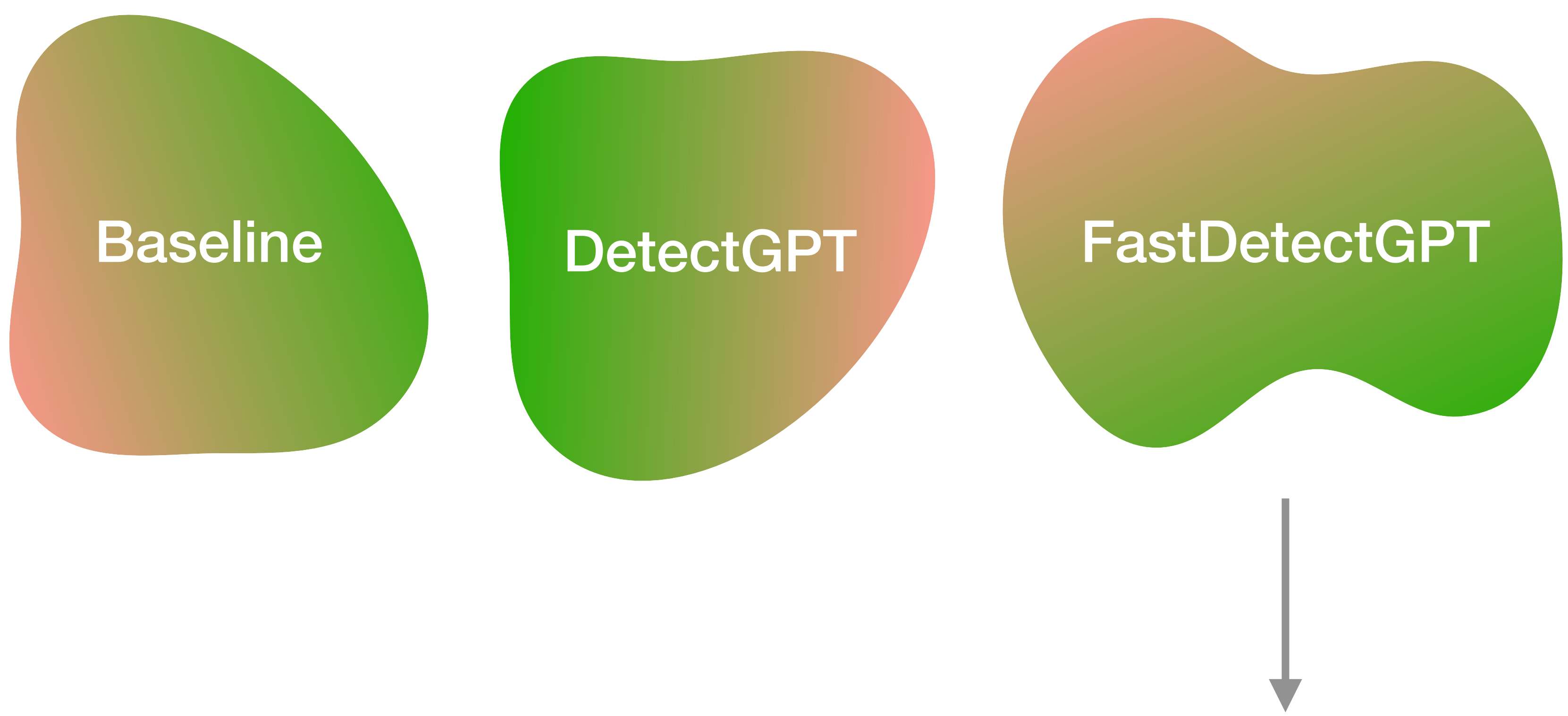
Considering we have a human text dataset with multiple entries, building the AI text will consist of building an AI equivalent sentence for each entry, as follows:



# Methods

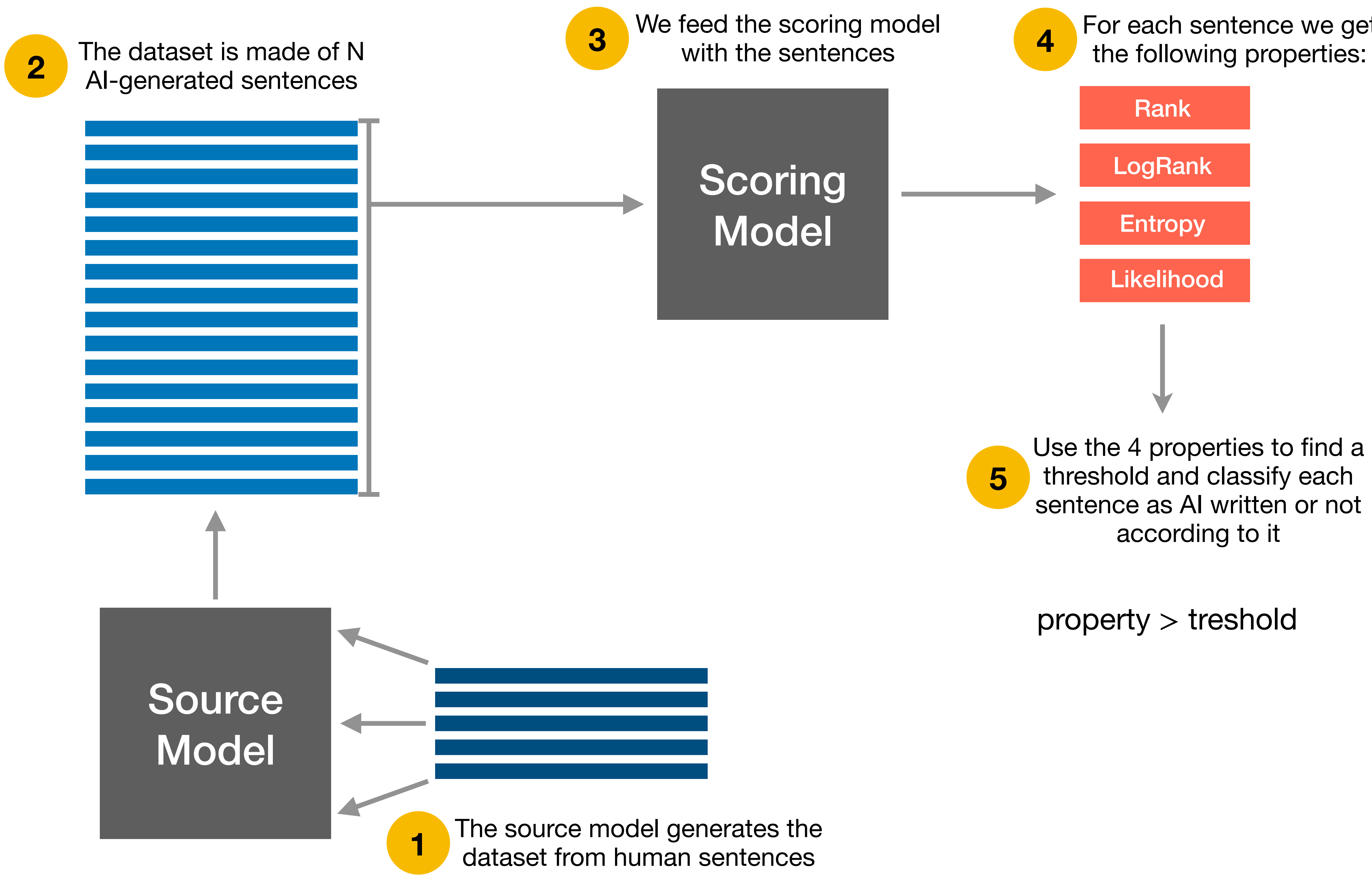
# Three Different Methods for Detection

We have studied three different methods for identifying AI-written text:

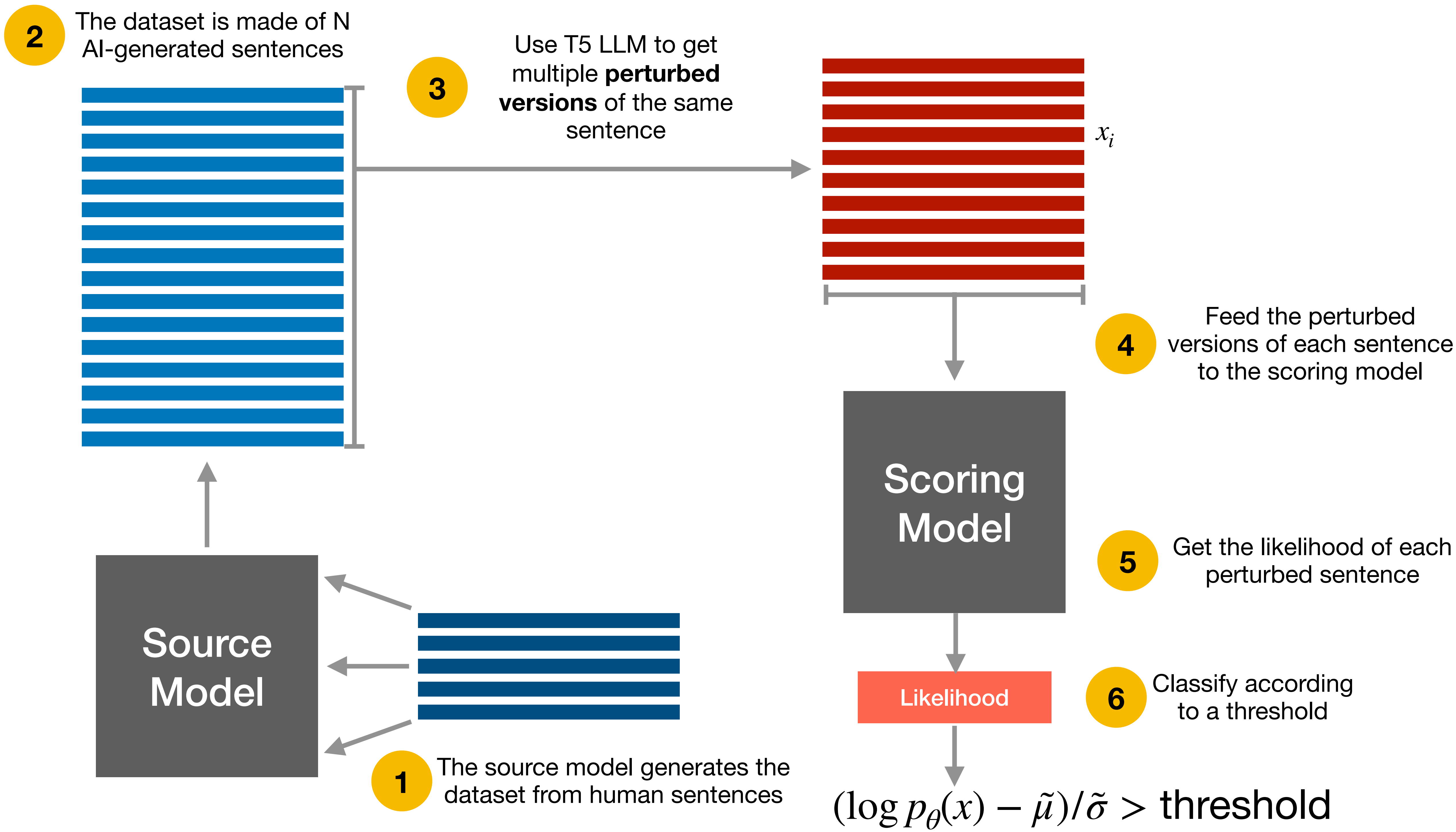



Our work is focused on this one

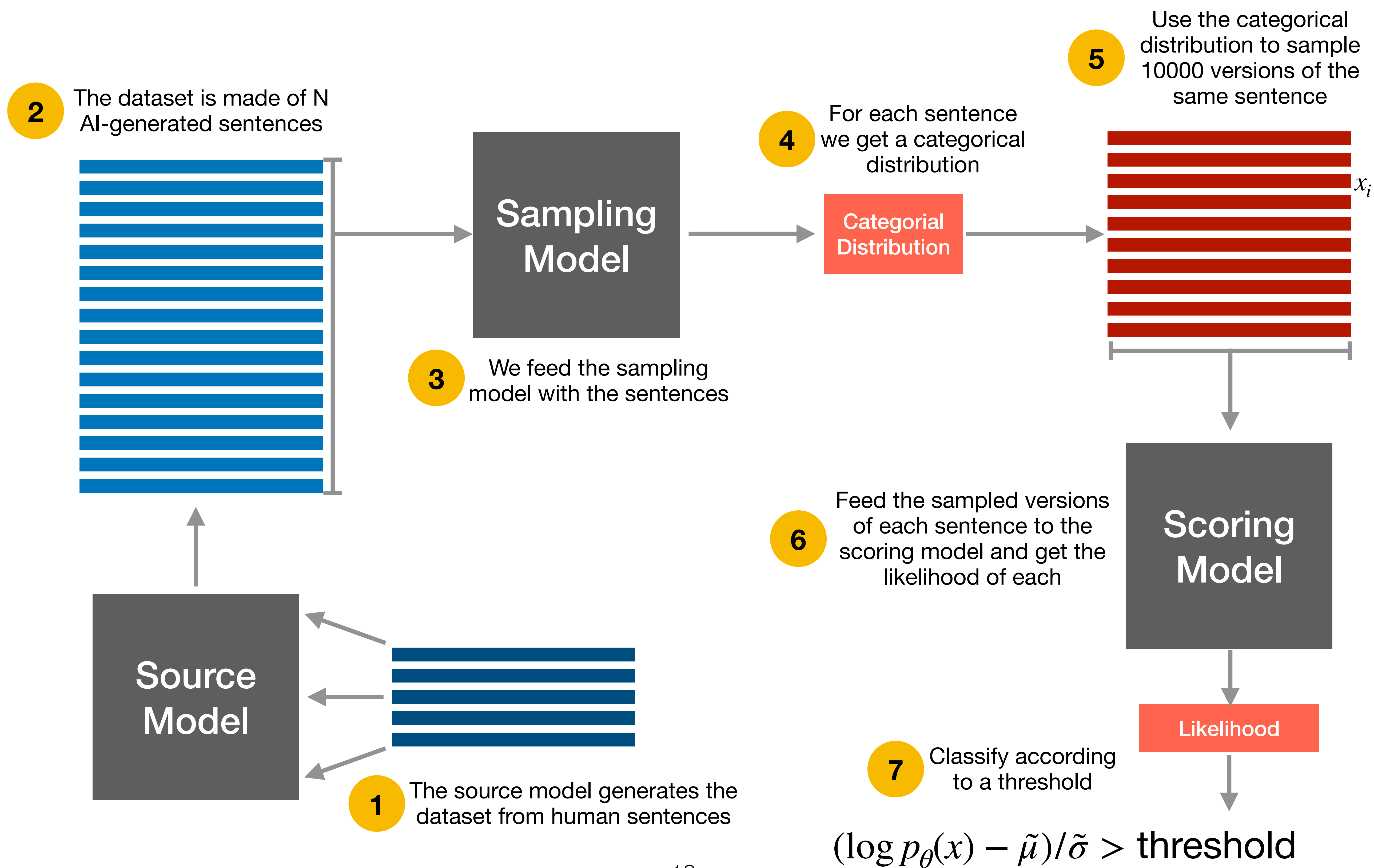
# Baseline Methods



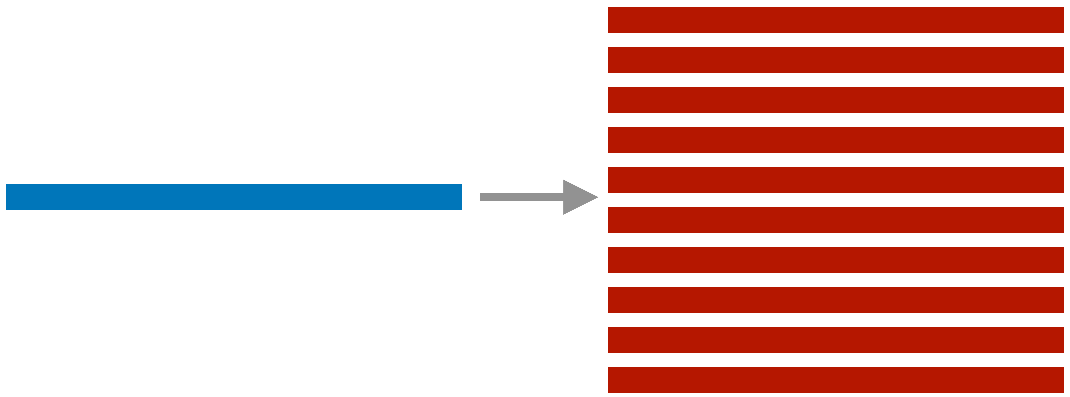
# DetectGPT


2
The dataset is made of N AI-generated sentences
3
Use T5 LLM to get multiple **perturbed versions** of the same sentence
$x_i$
4
Feed the perturbed versions of each sentence to the scoring model
Scoring Model
5
Get the likelihood of each perturbed sentence
Likelihood
6
Classify according to a threshold
Source Model
1
The source model generates the dataset from human sentences
$(\log p_\theta(x) - \tilde{\mu})/\tilde{\sigma} >$ threshold

# FastDetectGPT



# FastDetectGPT



1 AI generated sentence from the dataset originates 10000 samples and with those samples we compute this

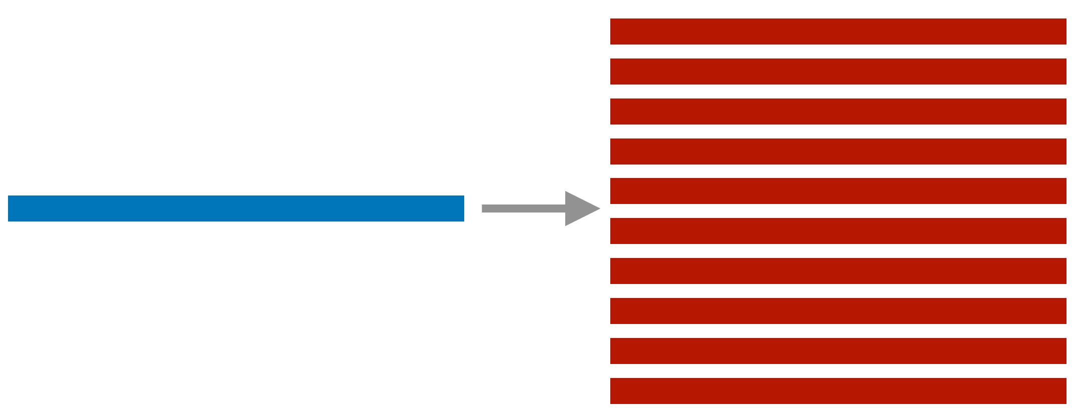
X

# FastDetectGPT

**Why** are we using this log-likelihood based sum?

| | |
|---|---|
| $\log p_\theta(x_i)$ | Is the log likelihood of the sentence $x_i$ under the scoring model (sum of each token's log probability) |
| $\tilde{\mu}$ | Is the mean of the log likelihood of each of the 10000 sample $x_i$ |
| $\tilde{\sigma}$ | Is the standard deviation of the log likelihood of each of the 10000 sample $x_i$ |

$(\log p_\theta(x) - \tilde{\mu})/\tilde{\sigma} >$ threshold

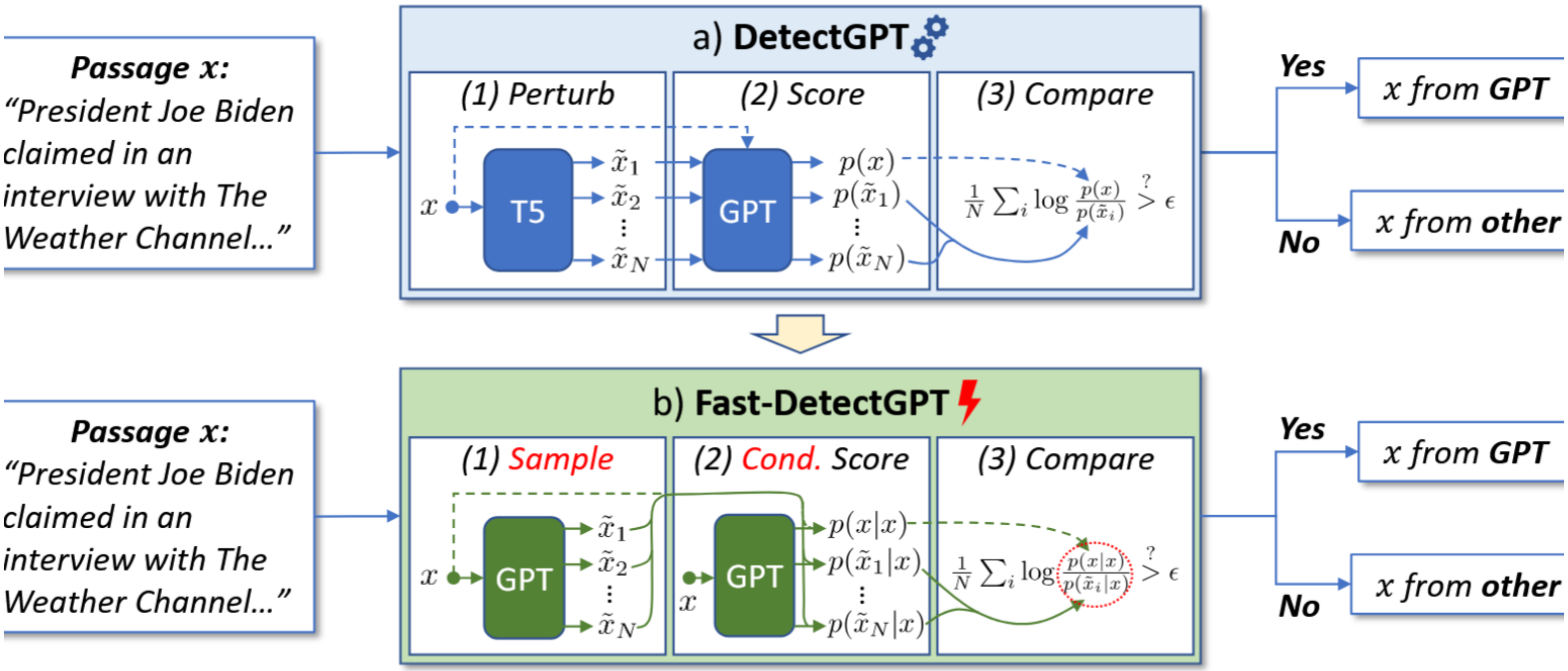


**1** AI generated sentence from the dataset originates **10000** samples and with those samples we compute this

The higher this is, the more likely it is for the sentence to be generated by an AI model since it's log-probability is similar to the sampled ones

# The Method

DetectGPT & Fast DetectGPT




Idea: perturb input text, freed to model & observe probability curvature

# Baseline Models

### Baselines = Threshold-based

```python
# Baselines compute a single score
score = get_likelihood(model, text)  # e.g., -2.5
score = get_rank(model, text)        # e.g., -150.2

# Then use a threshold for classification
if score > threshold:
    prediction = "AI-generated"
else:
    prediction = "Human-written"
```

### DetectGPT/Fast-DetectGPT = Comparison-based

```python
# DetectGPT compares original vs perturbations
original_ll = get_likelihood(model, original_text)           # e.g., -2.5
perturbation_lls = [get_likelihood(model, pert) for pert in perturbations]  # e.g

# Z-score comparison (relative measure)
mean_pert = mean(perturbation_lls)  # e.g., -2.93
std_pert = std(perturbation_lls)    # e.g., 0.15
z_score = (original_ll - mean_pert) / std_pert  # (-2.5 - (-2.93)) / 0.15 = 2.87

# Higher z-score = more likely AI-generated
```

### Key Difference:

**Threshold approach:**

- **Absolute judgment**: "This text has likelihood -2.5, is that high or low?"
- **Problem**: What's "high" depends on the model, domain, text length, etc.

**Comparison approach:**

X

# FastDetectGPT

Here follows the pseudo-code for FastDetectGPT:

**Algorithm 1** Fast-DetectGPT machine-generated text detection.

**Input:** passage $x$, sampling model $q_\varphi$, scoring model $p_\theta$, and decision threshold $\epsilon$
**Output:** True – probably machine-generated, False – probably human-written.

```
function FastDetectGPT(x, q_φ, p_θ)
    x̃_i ~ q_φ(x̃|x), i ∈ [1..N]                      ▷ Conditional sampling
    μ̃ ← (1/N) Σ_i log p_θ(x̃_i|x)                     ▷ Estimate the mean
    σ̃² ← (1/(N−1)) Σ_i (log p_θ(x̃_i|x) − μ̃)²        ▷ Estimate the variance
    d̂_x ← (log p_θ(x) − μ̃)/σ̃                         ▷ Estimate conditional probability curvature
    return d̂_x > ε
end function
```

$$\sum_i (\log p_\theta(x_i) - \tilde{\mu})/\tilde{\sigma} > \text{threshold}$$

# Sampling and Scoring

# Sampling and Scoring

Both sampling and scoring require the token distribution at each position in the sentence - sampling for alternative token selection, scoring for calculation of conditional probability curvature at each token

## Sampling

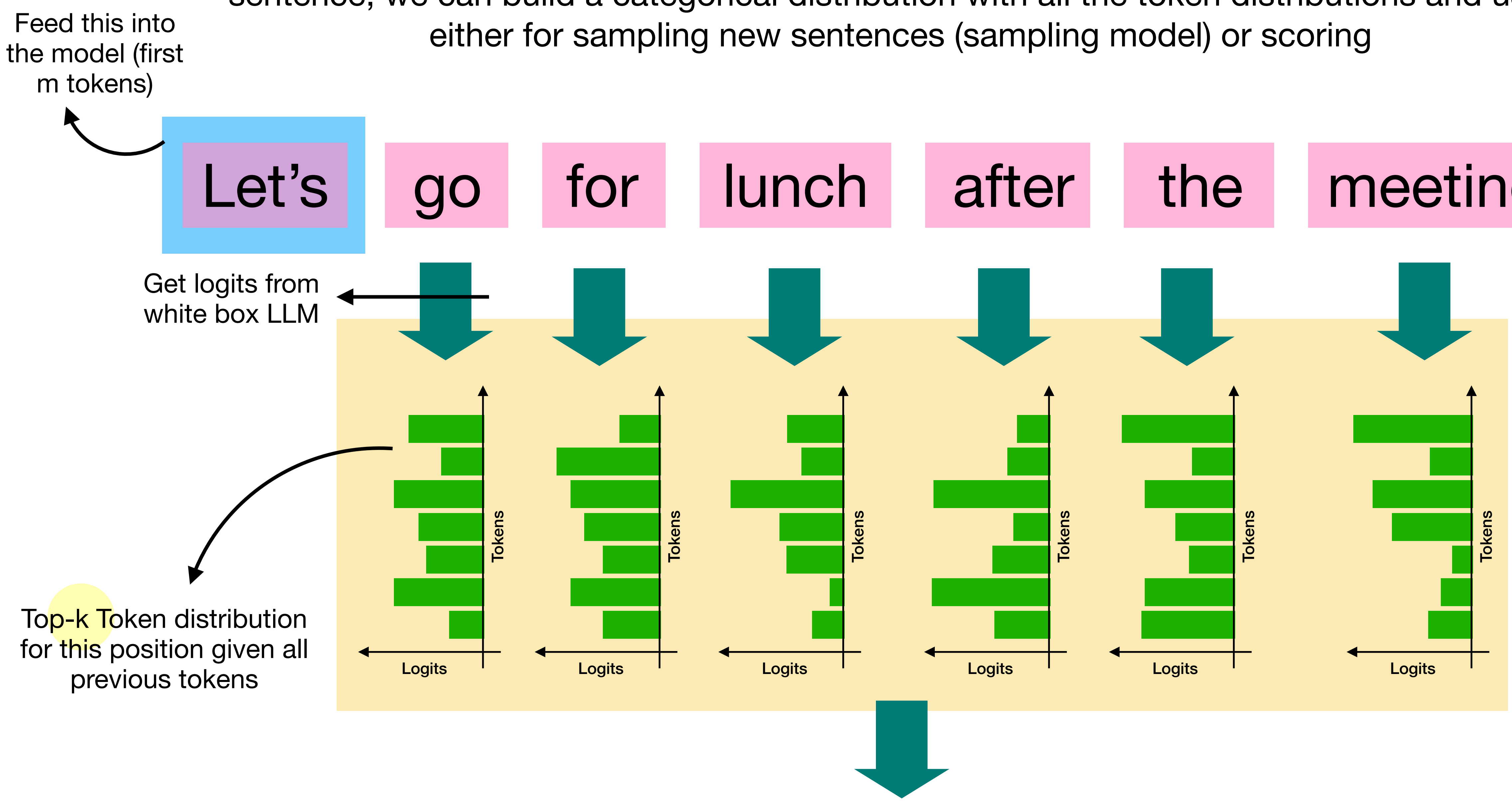
Generates alternative sequences by sampling from the model's token distribution

## Scoring

Scoring model used to compute conditional-probability curvature at each token; higher values mean the sequence is more likely machine-generated

# Obtaining token distributions

We feed an LLM model the first m tokens of the original sentence in order to get logits for the next positions. After we have a probability distribution over each position on the sentence, we can build a categorical distribution with all the token distributions and use it either for sampling new sentences (sampling model) or scoring

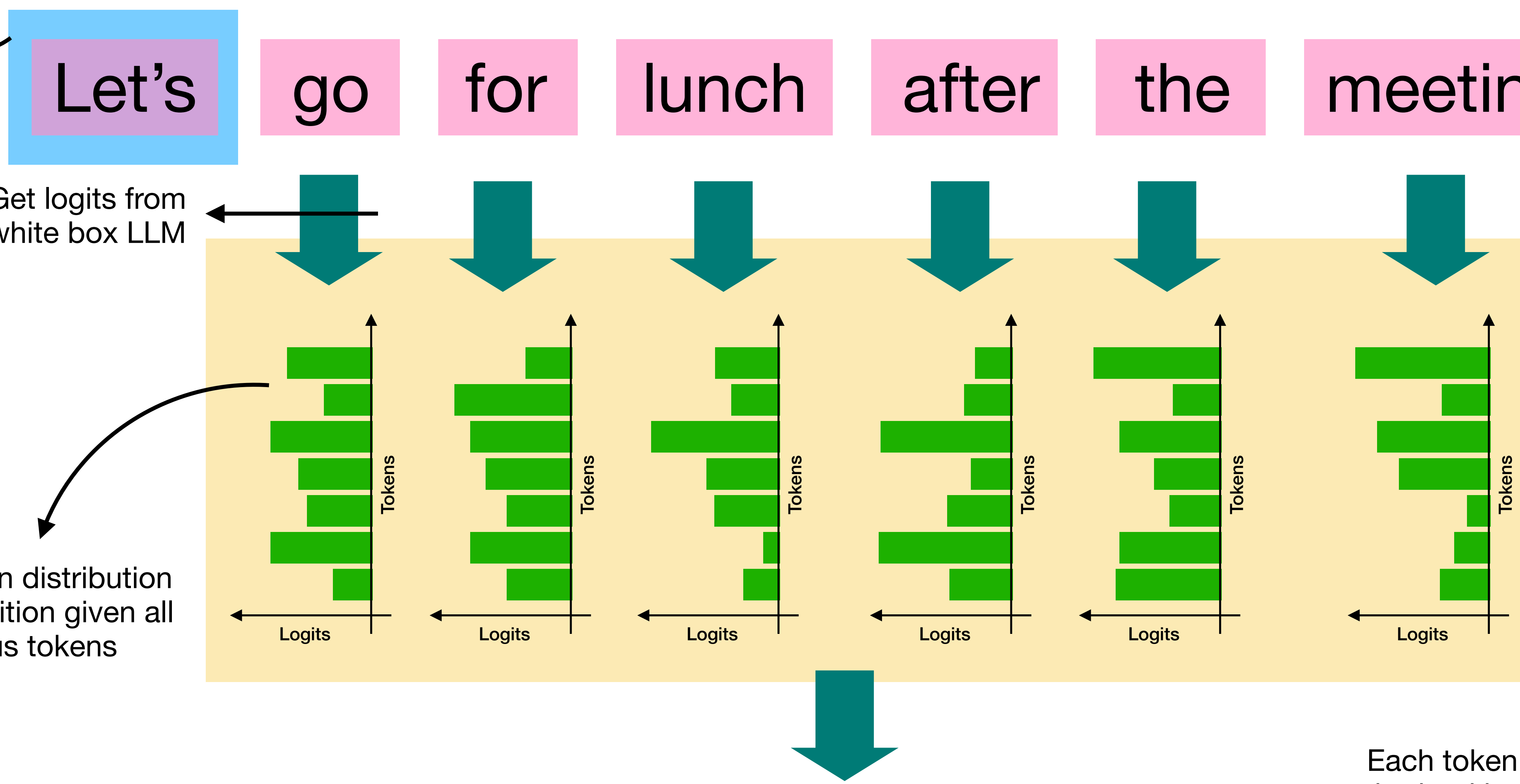



Build a categorical distribution with all token distributions for the sampling model and for the scoring model

# Obtaining token distributions

We feed an LLM model the first m tokens of the original sentence in order to get logits for the next positions. After we have a probability distribution over each position on the sentence, we can build a categorical distribution with all the token distributions and use it either for sampling new sentences (sampling model) or scoring

Feed this into the model (first m tokens)




Get logits from white box LLM

Top-k Token distribution for this position given all previous tokens

Build a categorical distribution with all token distributions for the sampling model and for the scoring model

Each token's distributions is obtained based on the first m tokens of the original sentence

Probability of a generated sample

X

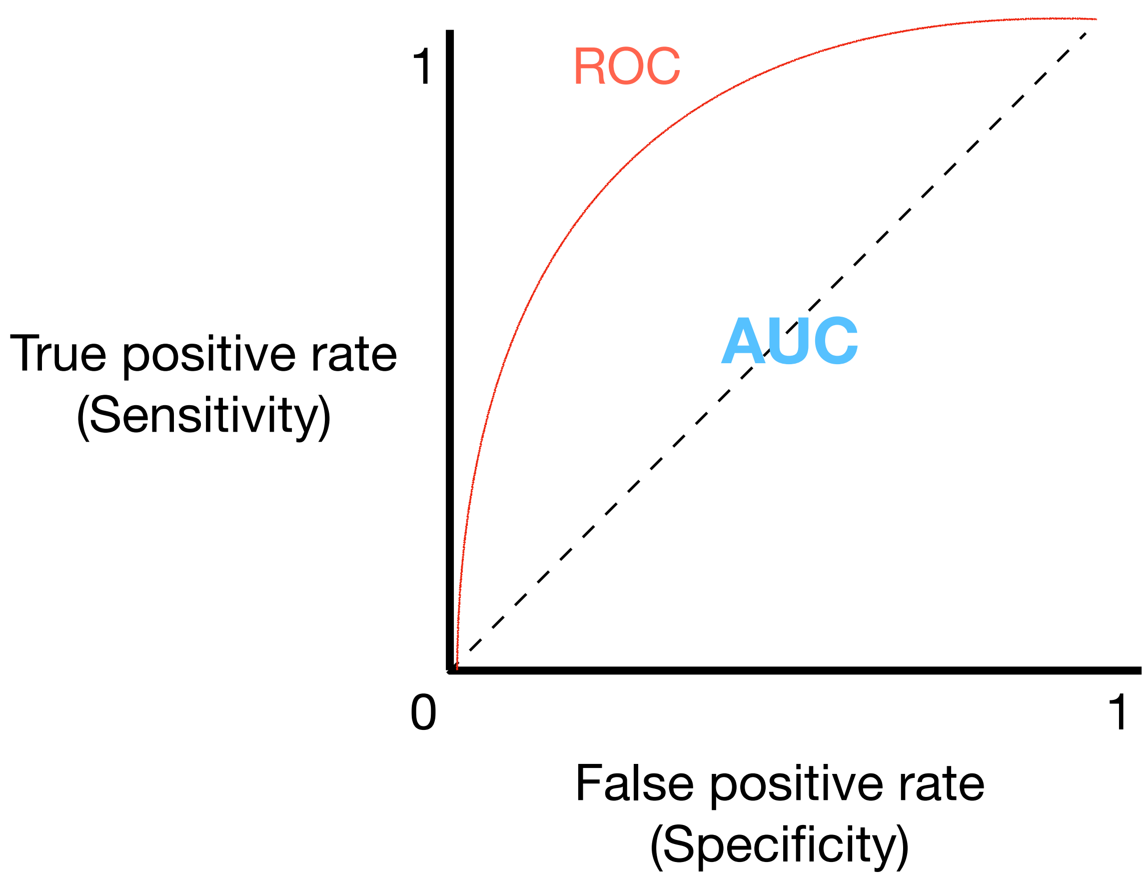
$$p_\theta\left(x^{\text{generated}} \,|\, x^{\text{original}}\right) = \prod_j p_\theta\left(x_j^{\text{generated}} \,|\, x_{<m}^{\text{original}}\right)$$

The process of obtaining the tokens distribution for each position is done twice*: once for the Sampling Model and once for the Scoring Model

*Except for white box setting where both the models are identical.

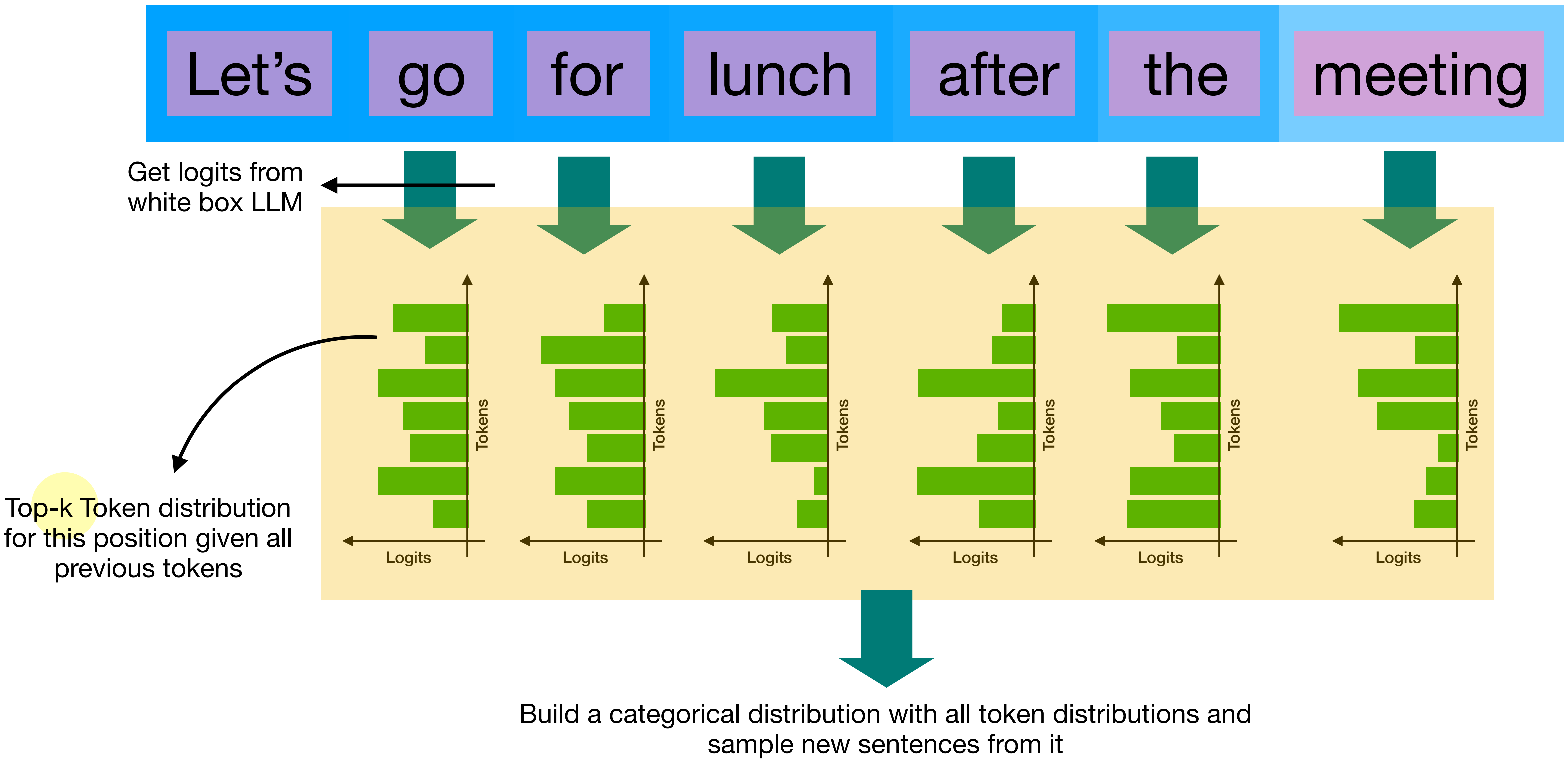
# Main Metric

## Area Under Receiver Operating Characteristic

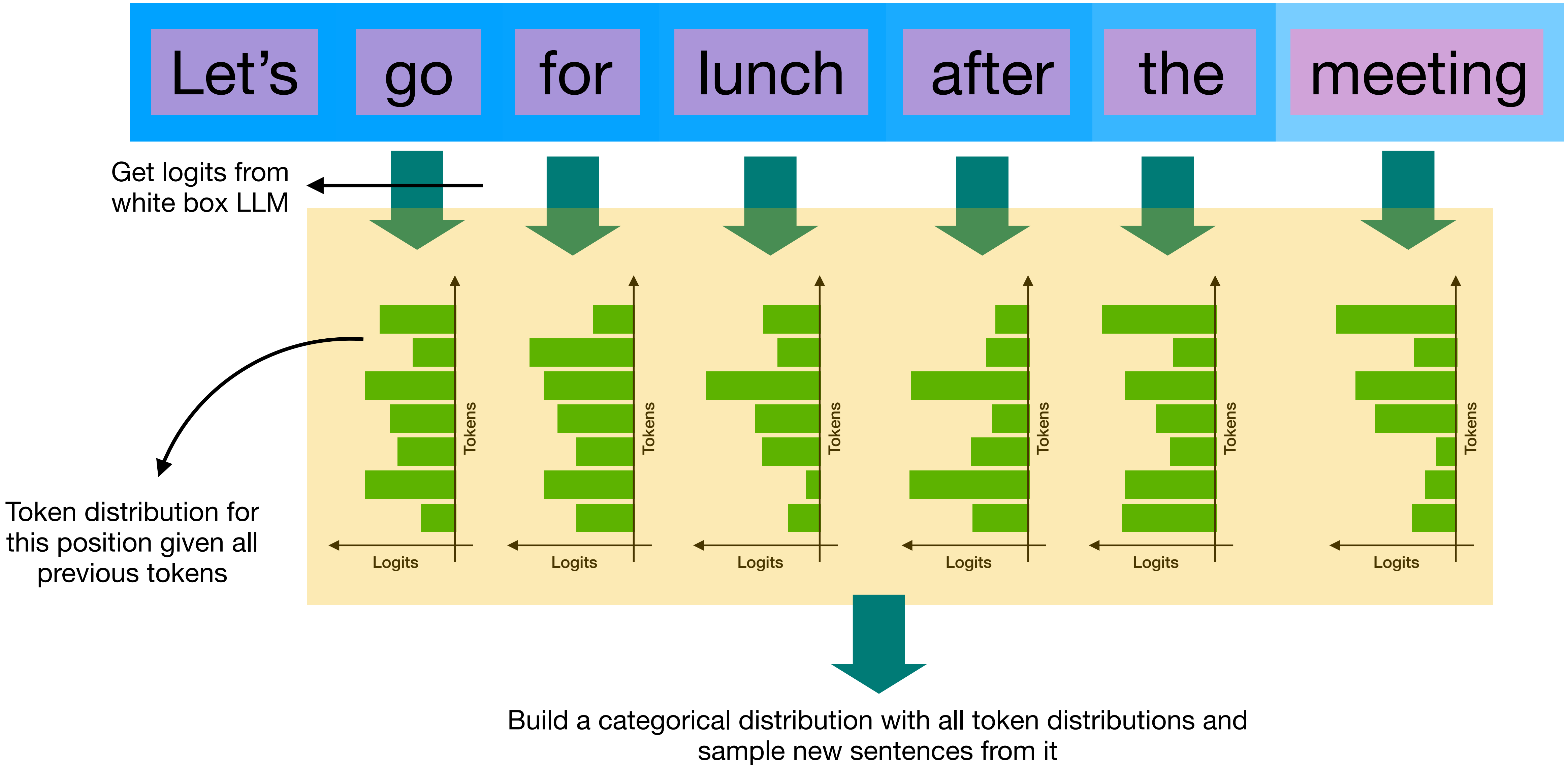


# How does sampling work? (Wrong)

The goal is to build multiple versions of the same sentence (perturbed versions) and sampling is the way to do it. We feed an LLM model each sub-sentence of the original sentence in order to get logits for the next token. After we have a probability distribution over each token position on the sentence, we can build a categorical distribution with all the token distributions and sample from it in order to get a new sentence.




X


Let's
go
for
lunch
after
the
meeting
Get logits from white box LLM
Token distribution for this position given all previous tokens
Tokens
Logits
Build a categorical distribution with all token distributions and sample new sentences from it

Let’s go for lunch after the meeting

X

# Experiments and Results

# New Models and Datasets

# Reproducibility Study

Small Open-Weight Models:

- DeepSeek R1 Distill Llama
- Microsoft Phi 2
- Mistral 7B Instruct v0.2

Metric: AUROC

Note: DetectGPT could only be used one time due to computational constraints

| Method | XSum | | | |
|---|---|---|---|---|
| | **R1-8B** | **Phi-2** | **Mistral** | **Avg.** |
| Likelihood | 0.9999 | 0.8782 | 0.9665 | 0.9482 |
| Entropy | 0.1645 | 0.5411 | 0.4558 | 0.3871 |
| LogRank | **1.0000** | 0.9051 | 0.9596 | 0.9549 |
| DetectGPT | 0.8560 | - | - | - |
| ★ Fast-DetectGPT | **1.0000** | **0.9769** | **0.9989** | **0.9919** |

| Method | HC3 | | | |
|---|---|---|---|---|
| | **R1-8B** | **Phi-2** | **Mistral** | **Avg.** |
| Likelihood | 0.8391 | 0.7139 | 0.7486 | 0.7672 |
| Entropy | 0.3521 | 0.4358 | 0.4235 | 0.4038 |
| LogRank | 0.8275 | 0.7224 | 0.7393 | 0.7631 |
| ★ Fast-DetectGPT | **0.9710** | **0.8347** | **0.8963** | **0.9007** |

# Speed Comparison

Fast-DetectGPT is

**118.9x**

faster than DetectGPT

on XSum with DeepSeek R1 Distill Llama

# Close-Weight API Model

| Method | GPT 4o mini | | |
|---|---|---|---|
| | XSum ★ | HC3 | **Avg.** |
| Fast-DetectGPT | **0.8051** | 0.6026 | 0.7039 |

- Lower performance due to top-20 probability restriction

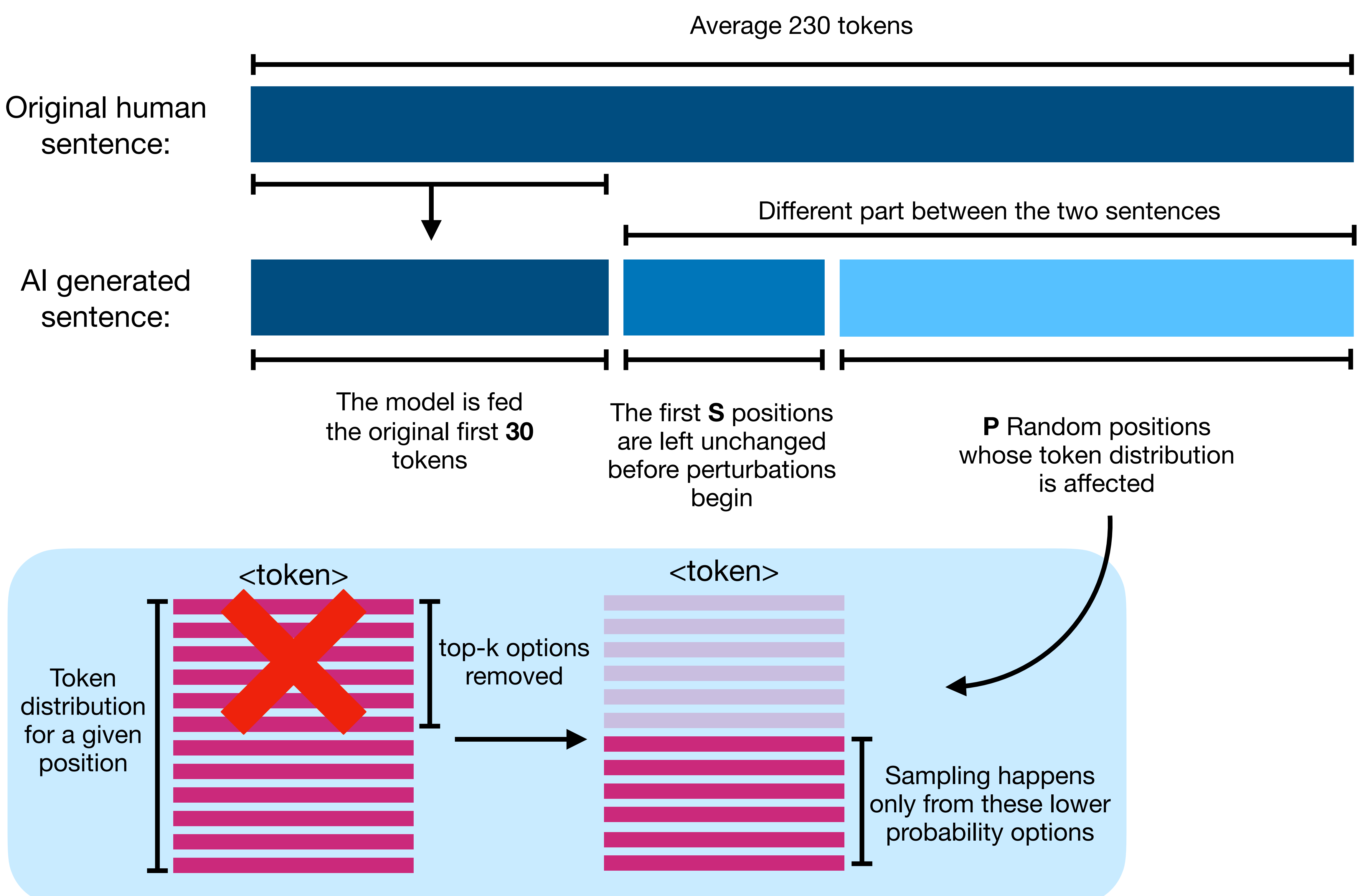
# Low-Resource Language

| Method | Underrepresented language | | | |
|---|---|---|---|---|
| | **R1-8B** | **Phi-2** | **Mistral** | **Avg.** |
| Likelihood | **1.0000** | 0.9976 | 0.9972 | 0.9983 |
| Entropy | 0.0010 | 0.0968 | 0.2730 | 0.1236 |
| ★ LogRank | **1.0000** | **0.9984** | 0.9974 | **0.9986** |
| Fast-DetectGPT | 0.9994 | 0.9825 | **0.9995** | 0.9938 |

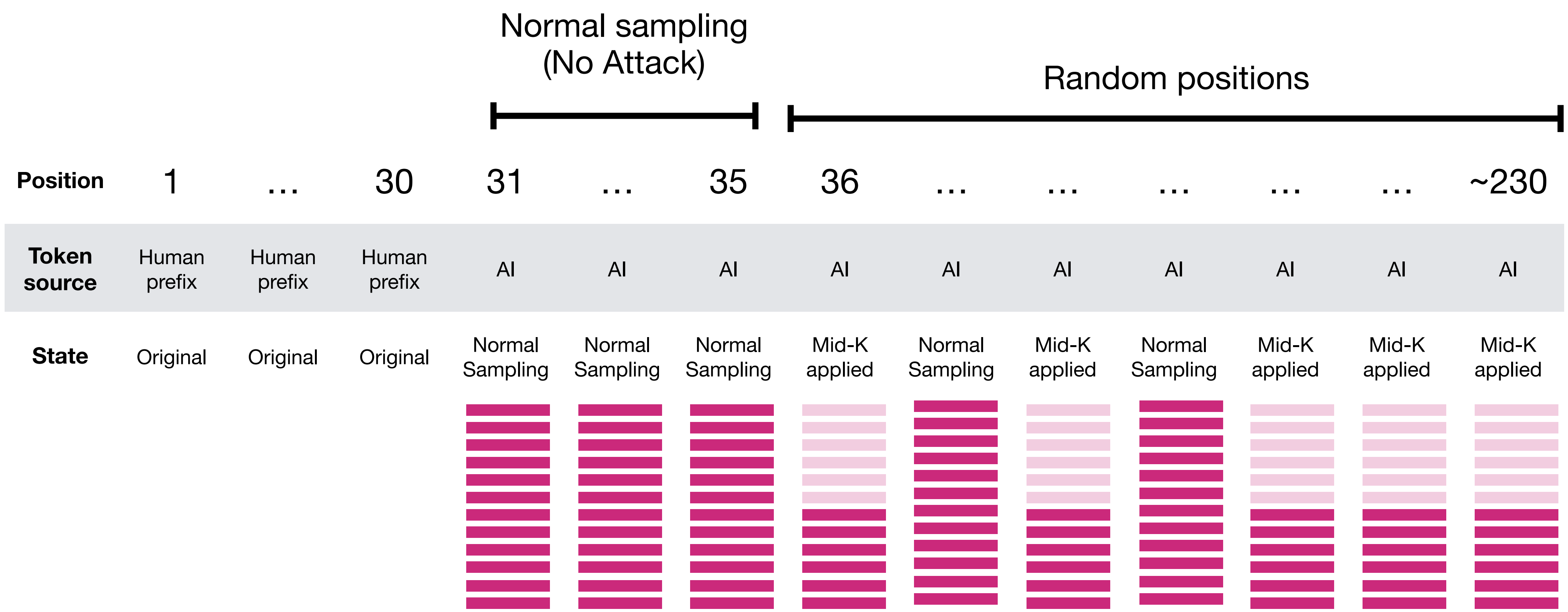
Tested on Scottish Gaelic subset from XL-Sum
+ Competitive scores, overall robustness

# Mid-K Extension

# How Mid-K Works



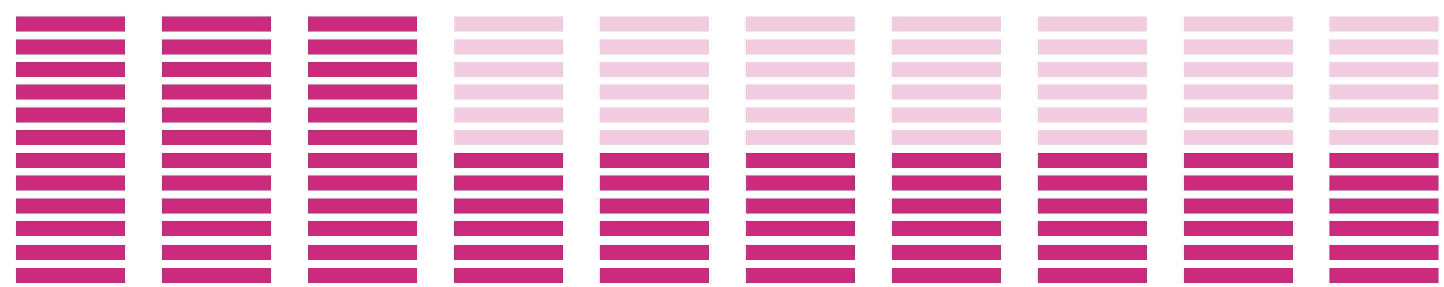
# Mid-K in Action




- Key parameters:
  - **P**: Number of random positions where attack is applied
  - **K**: Number of top tokens removed at each position
  - **S**: Start position (attack begins after S tokens)

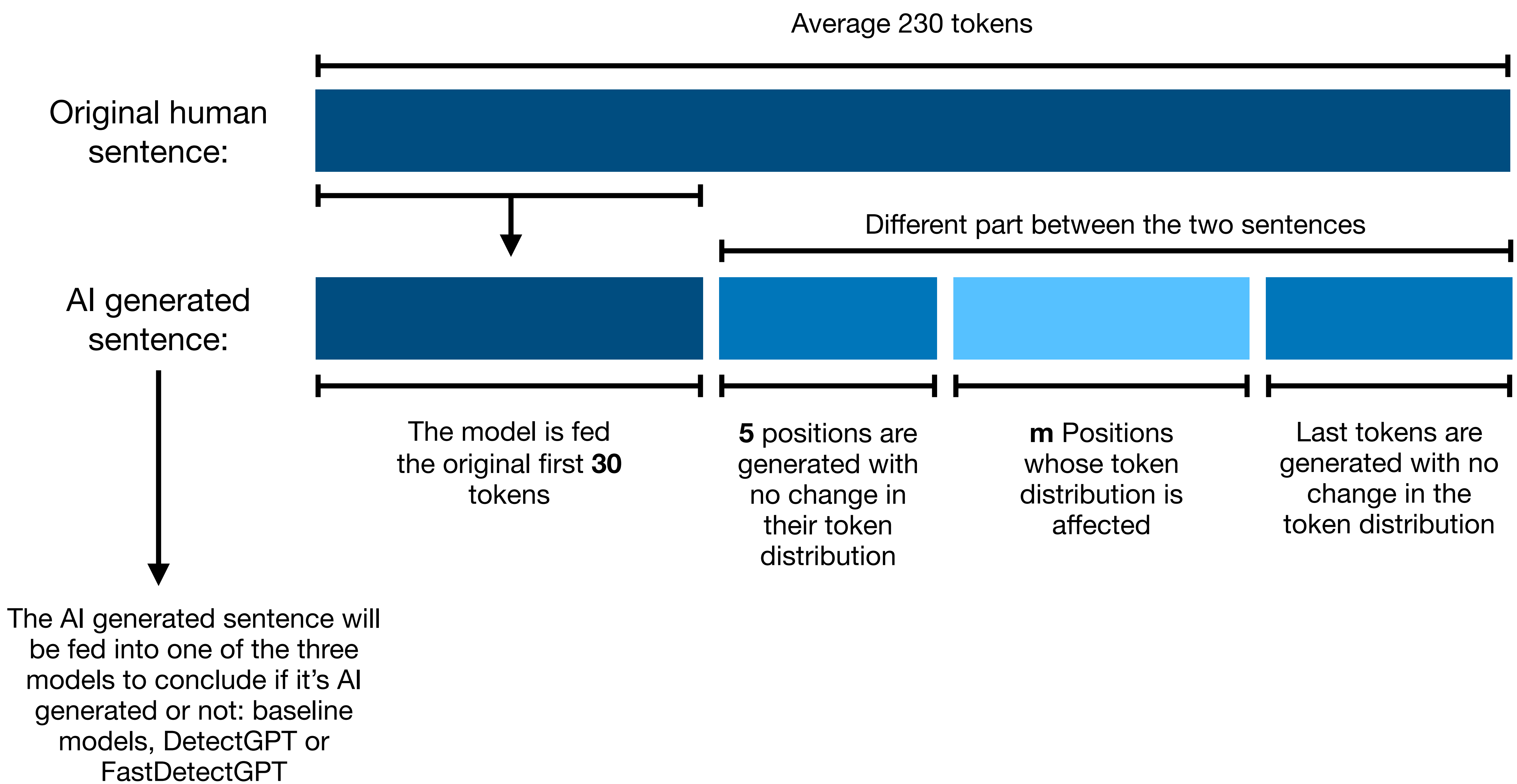
# Examples

| **Position** | 1 | … | 30 | 31 | … | 35 | 36 | … | … | … | … | … | ~250 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Token** | <token> | | <token> | <token> | <token> | <token> | <token> | <token> | <token> | <token> | <token> | <token> | <token> |
| **State** | Original | Original | Original | Sampled | Sampled | Sampled | Sampled | Sampled | Sampled | Sampled | Sampled | Sampled | Sampled |



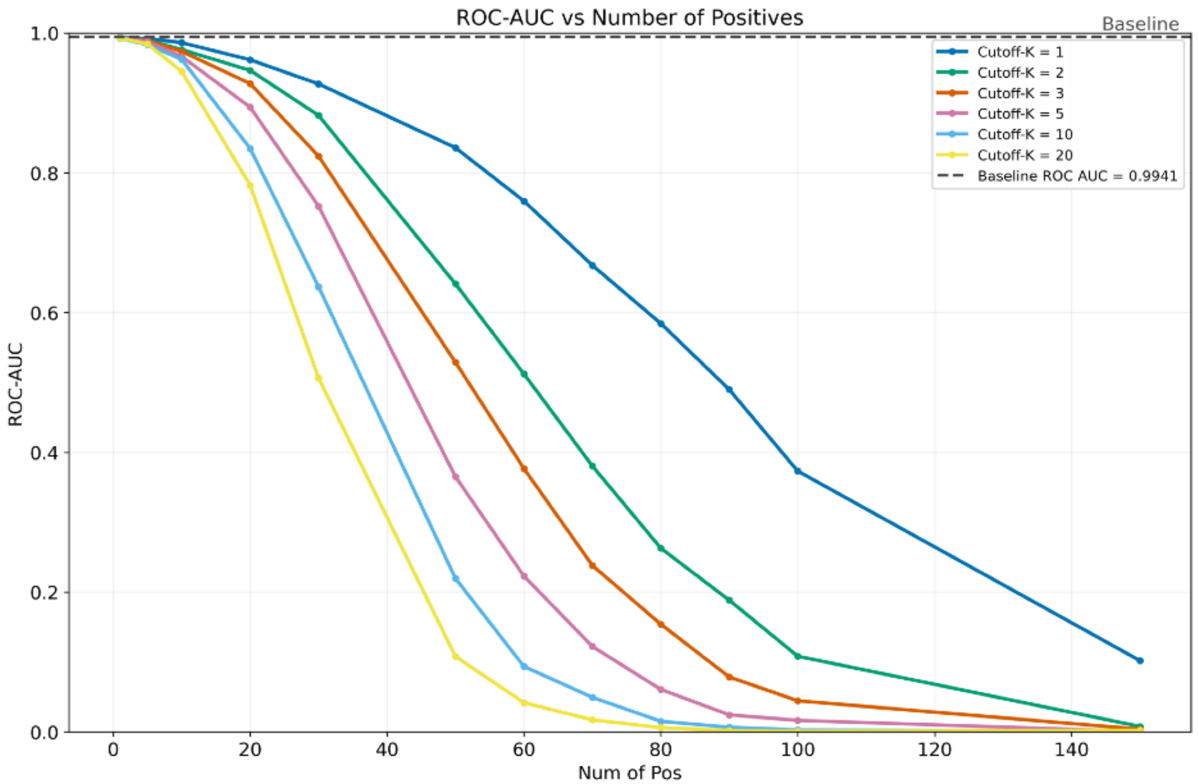
X

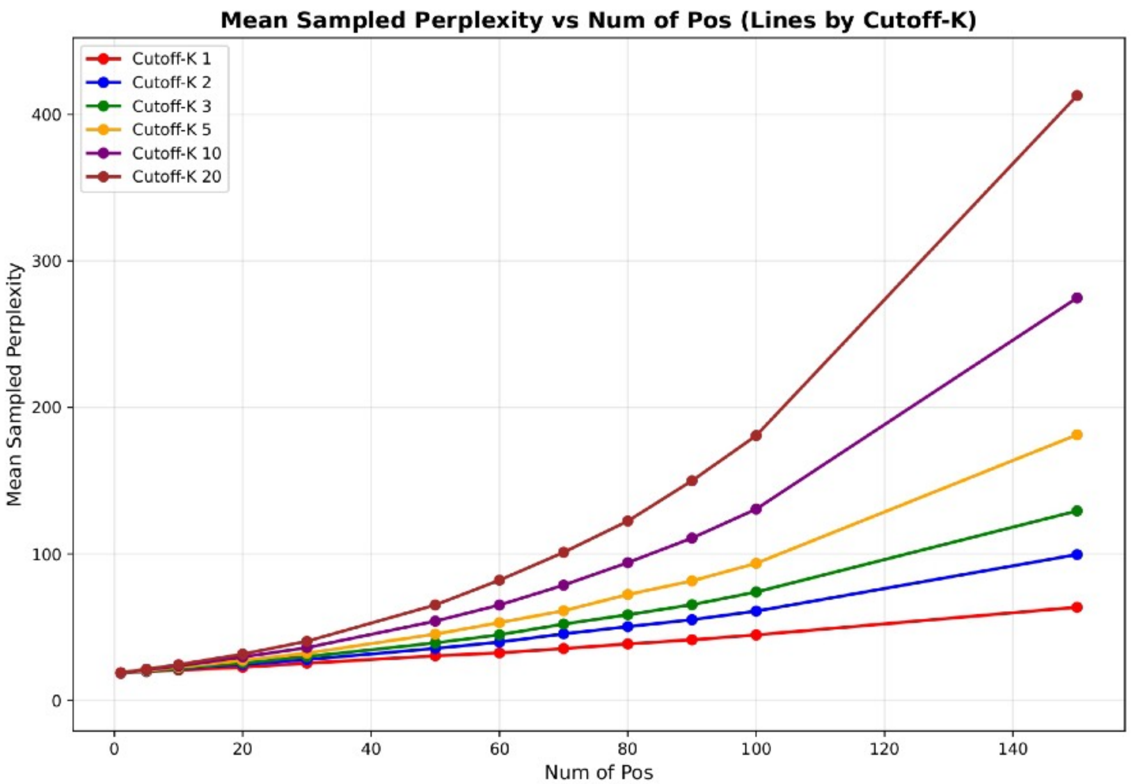
# Mid-k dataset generation




X

# Effect of Mid-K Attack

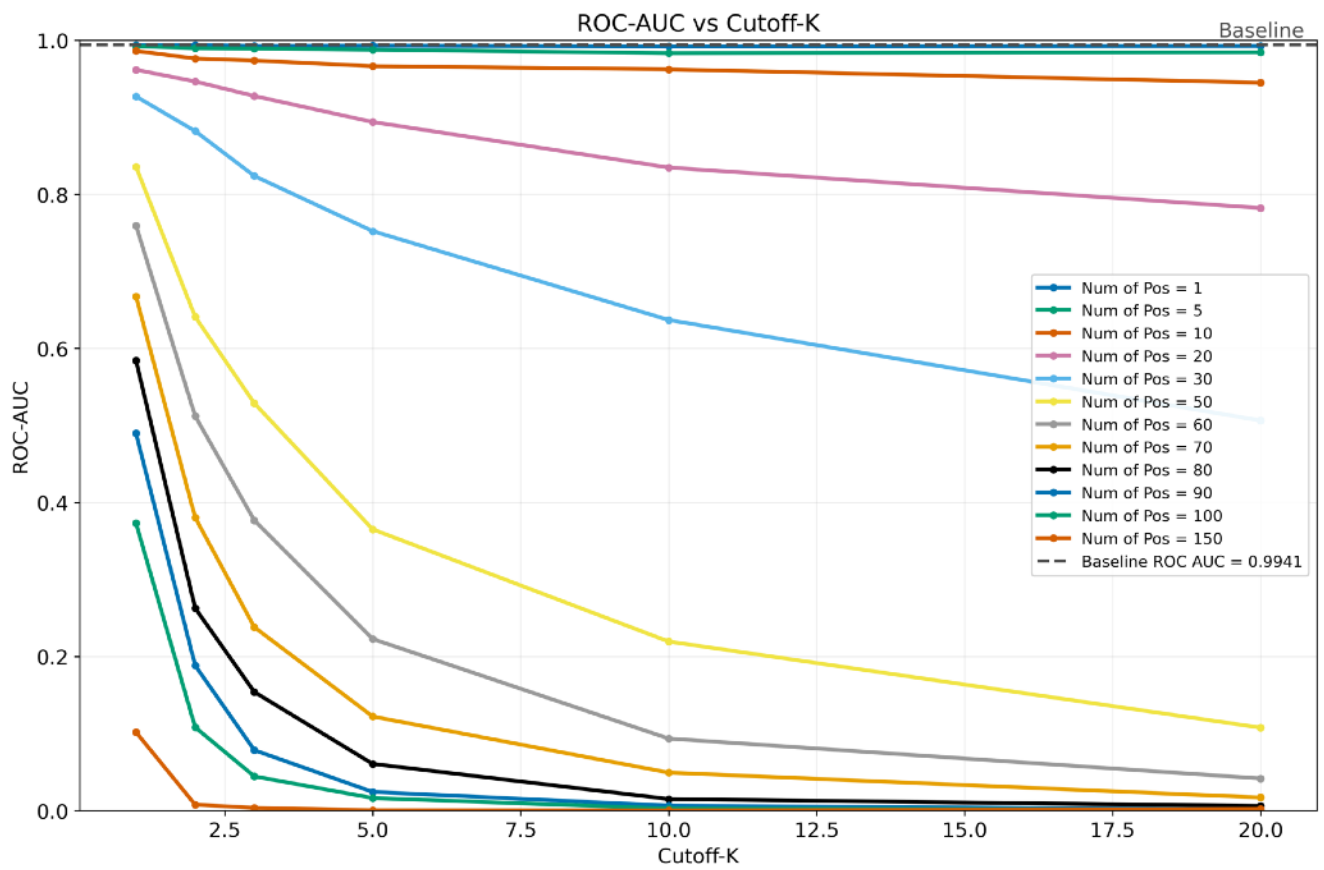



Setup:

- Model: GPT-2
- Dataset: XSum
- Parameters: P ∈ {1, 5, 10, 20, 30, 50, 60, 70, 80, 90, 100, 150}
- K ∈ {1, 2, 5, 10, 20}, S = 5

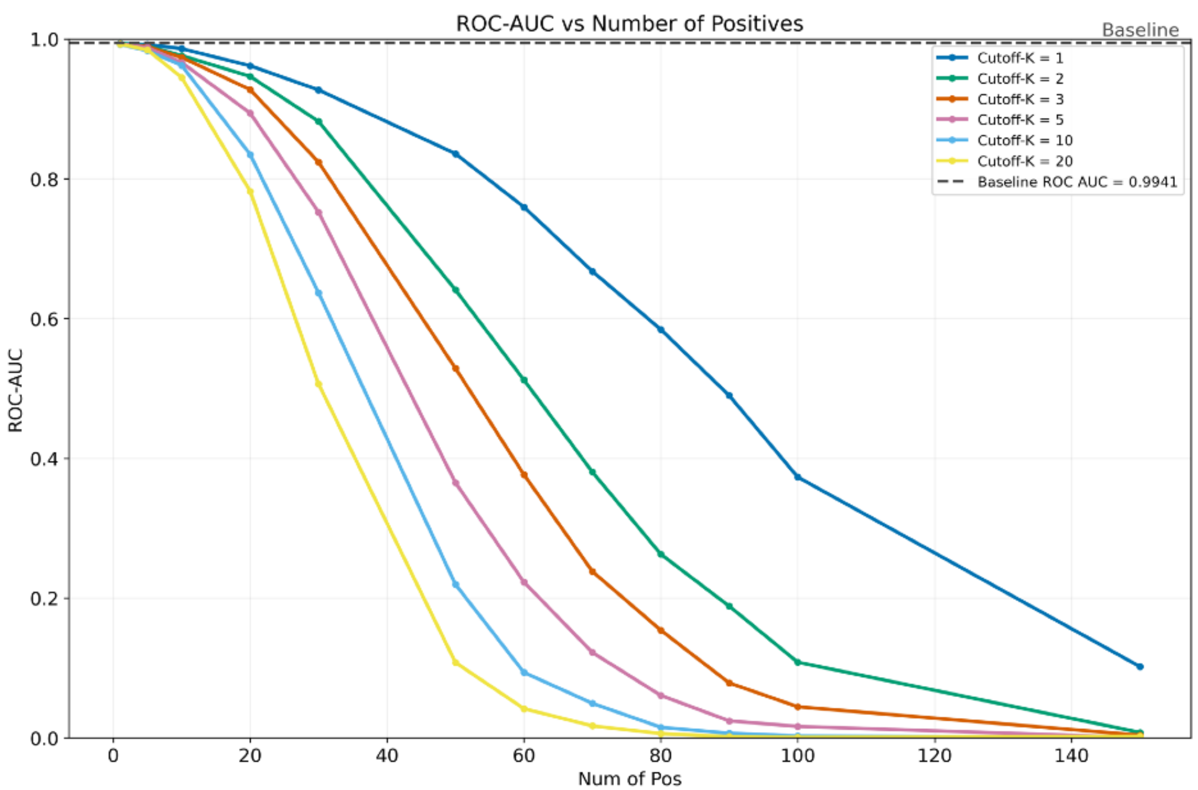
# Effect of Mid-K Attack

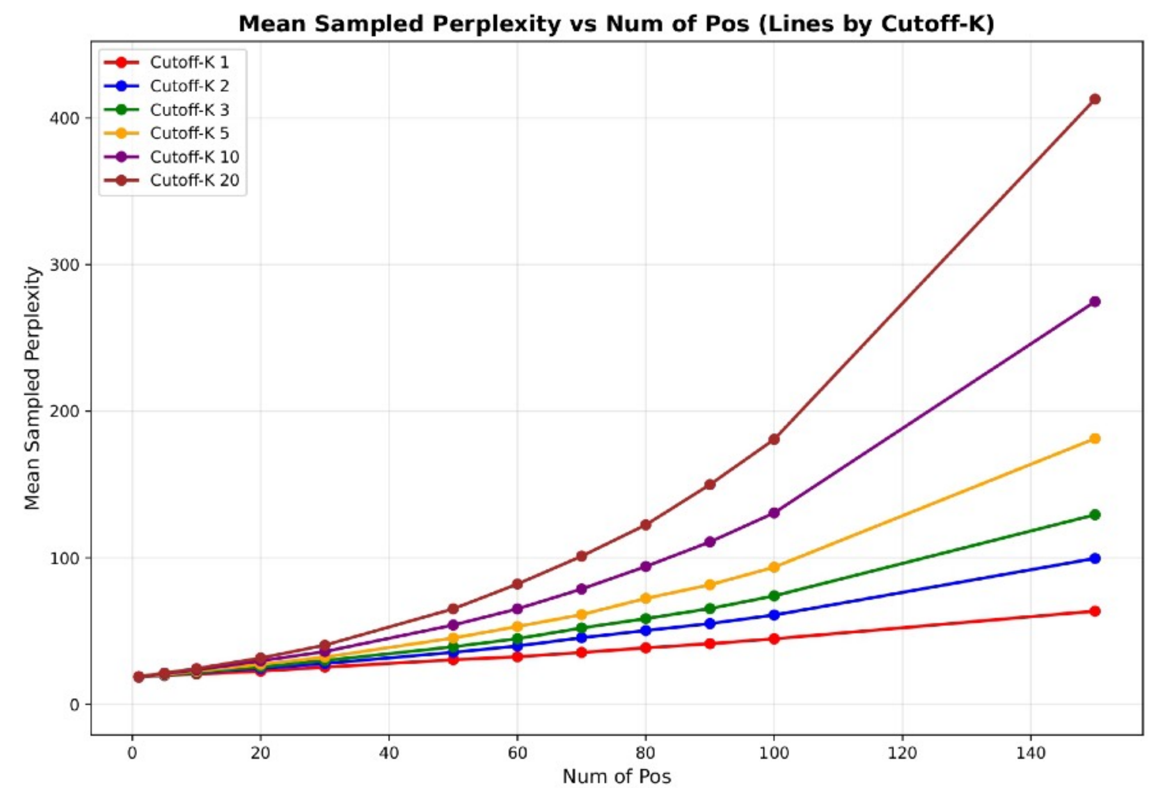



Setup:

- Model: GPT-2
- Dataset: XSum
- Parameters: P ∈ {1, 5, 10, 20, 30, 50, 60, 70, 80, 90, 100, 150}
- K ∈ {1, 2, 5, 10, 20}, S = 5

# Effect of Mid-K Attack




Using a dataset where the AI generated sentences have tokens with lower log probability brings the whole log-likelihood of the generated sentence down. This will trick FastDetectGPT because it focuses on concluding that sentences with high log-probability are AI-generated. In this case, they are still AI-generated but the log-probability is lower, that's why the AUROC goes down as we increase the number of positions with removed top-m logits and as we increase m.

# Qualitative Examples

**Fixed part:** "The false message said: "Breaking: Two Explosions..."

| | |
|---|---|
| Human | in the White House and Barack Obama is injured." US markets were spooked by the tweet |
| AI (No Attack) | At University of Michigan Student Residences, Campus Police Service (APS) has confirmed at least one person was hurt. |
| Mild (P=60, K=2) | At University of Michigan Student Residences, Campus Police Service (APS) has confirmed at 7:18 a.b.'s report a report by Campus Police that three occupants were shot dead |
| Wide (P=150, K=1) | At University of Michigan Student Residences, Campus Fires. We apologize if we didn't find the wrong tweet. The University has since been informed. |
| Strong (P=150, K=20) | are Found at a School; Parents Support Public Alert: Close Education Week Hours Soon With Two Public Works, Infrastructure Flashed". http@worldc.en |

# Perplexity

| AUROC | | Number of positions with top-m logits being removed | | | |
|---|---|---|---|---|---|
| | | 5 | 10 | 20 | 30 |
| Number of logits removed from the probability distribution (m) | 1 | | | | |
| | 3 | | | | |
| | 5 | | | | |
| | 10 | | | | |
| | 20 | | | | |

# Conclusion

# Takeaways

## Fast-DetectGPT:

- Outperforms baselines on newly tested models ✓
- Two orders of magnitude faster than DetectGPT (not 340 times though) ✓
- Vulnerable towards adversarial attacks ✗
- Performance depends on full logits distribution availability ✗

# Thank you!