PDE: ENABLING WHITE-BOX METHODS TO USE PROPRIETARY MODELS FOR ZERO-SHOT LLM-GENERATED TEXT DETECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) can generate text almost indistinguishable from human-written one, highlighting the importance of machine-generated text detection. However, current zero-shot techniques face challenges as white-box methods are restricted to use weaker open-source LLMs, and black-box methods are limited by partial observations from stronger proprietary LLMs. It seems impossible to enable white-box methods to use proprietary models because the API-level access neither provides full predictive distributions nor inner embeddings. To break this deadlock, we propose *Probability Distribution Estimation (PDE)*, estimating full distributions from partial observations. Despite the simplicity of PDE, we successfully extend white-box methods like Entropy, Rank, Log-Rank, and Fast-DetectGPT to latest proprietary models. Experiments show that PDE (Fast-DetectGPT, GPT-3.5) achieves an average accuracy of about 0.95 across five latest source models, improving the accuracy by 51% relative to the remaining space of the baseline (as Table 1). It demonstrates that the latest LLMs can effectively detect their own outputs, suggesting advanced LLMs may be the best shield against themselves. We release our codes and data at https://github.com/xxx/xxxxxx.

Table 1: Detection accuracy measured in AUROC across five latest LLMs, where the baseline Fast-DetectGPT uses an open-source model but our PDE (Fast-DetectGPT) uses a proprietary model. The methods are evaluated on the diverse Mix3 dataset, combining XSum, Writing, and PubMed, with details presented in the main results (Table 2). The notion "↑" indicates the improvement relative to the remaining space, calculated by $(new - old)/(1.0 - old)$, following Bao et al. (2023).

Method	**ChatGPT**	**GPT-4**	**Claude-3** Sonnet	**Claude-3** Opus	**Gemini-1.5** Pro	**Avg.**
Fast-DetectGPT *(Open-Source: GPT-Neo-2.7B)*	0.9487	0.8999	0.9260	0.9468	0.8072	0.9057
PDE (Fast-DetectGPT) *(Proprietary: GPT-3.5)*	**0.9766** (↑ **54%**)	**0.9411** (↑ **41%**)	**0.9576** (↑ **43%**)	**0.9689** (↑ **42%**)	**0.9244** (↑ **61%**)	**0.9537** (↑ **51%**)

1 INTRODUCTION

Large language models (LLMs) (OpenAI, 2022; Team et al., 2023; Anthropic, 2024) can produce fluent and cogent text content, which is almost indistinguishable from human-written content (Ippolito et al., 2020; Shahid et al., 2022; Dugan et al., 2023). It powers the productivity of various industries such as journalism (Christian, 2023), social media (Yuan et al., 2022), and education (M Alshater, 2022), but at the same time causes various risks such as misinformation, disinformation, and plagiarism (Pan et al.; Weidinger et al., 2021; Meyer et al., 2023), thereby urging automatic detection tools for building trustworthy AI systems (Kaur et al., 2022; Sun et al., 2024). However, as long as the LLMs increase their ability, the detection of their generations becomes more difficult (Mireshghallah et al., 2023; Sadasivan et al., 2023; Tang et al., 2024).

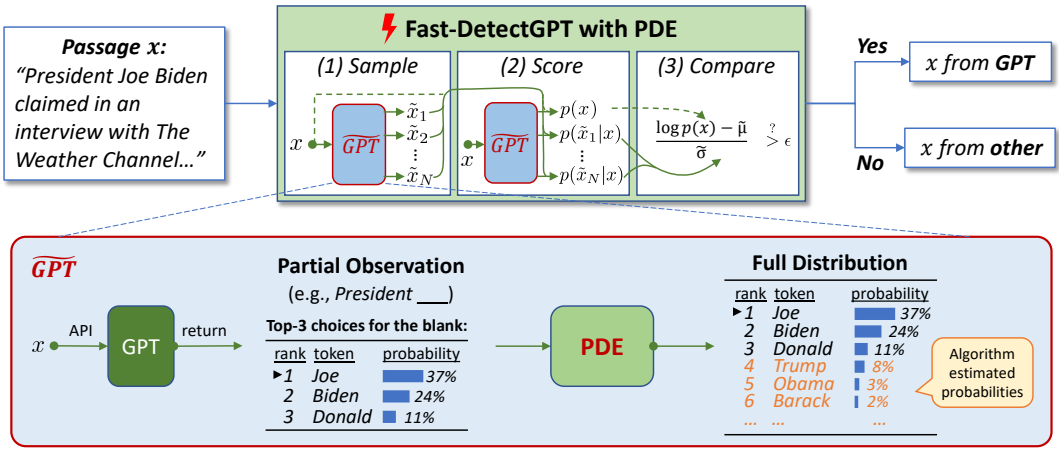


Figure 1: Fast-DetectGPT as an example to apply *Probability Distribution Estimation (PDE)*. The notion $\widetilde{\text{GPT}}$ refers to the model with estimated distribution, where the partial observation (top-K probabilities) returned by the model API is completed into a full distribution. The '*token*' column is just for reference, which is not necessary for calculating the metric (conditional probability curvature).

The most powerful LLMs are generally proprietary models, which only provide limit access through API. Consequently, the existing white-box methods that require 'full access' cannot be applied to these models, and instead, various black-box methods are developed. The black-box methods, such as DetectGPT (Mitchell et al., 2023) and DNA-GPT (Yang et al., 2023a), demonstrate competitive detection accuracies on advanced LLMs like GPT-3, ChatGPT, and GPT-4. However, these methods are less efficient and less robust compared to their white-box counterparts (Bao et al., 2023), because they rely on knowing the source model and need multiple evaluations or generations of text sequences. Instead of improving these black-box methods like Su et al. (2023), we turn back to white-box methods to see the potential of combining their strength with the power of latest LLMs. [1]

For example, Fast-DetectGPT (Bao et al., 2023) uses a fixed surrogate model to detect text from various source models including ChatGPT and GPT-4. The basic idea is to use a surrogate model to obtain token distributions to calculate a metric of conditional probability curvature, where the higher the metric, the more likely the input is machine-generated. Despite its simplicity, it achieves higher detection accuracies than the black-box methods. However, its detection accuracy on GPT-4 (AUROC 0.91) is significantly lower than on open-source models (an average of 0.99 on five models). We speculate that the lower accuracies on latest models are caused by the distribution mismatch between the small surrogate model and the large source models, which can potentially be addressed by using latest large models as the surrogate.

To enable white-box methods on proprietary models, we propose *Probability Distribution Estimation (PDE)* to estimate the full distribution from partial observation returned by model API. This observation includes the probabilities (logprobs) of the input tokens and a few (at least 1) top-ranking tokens on each token position, which we assume that a proprietary model gives. [2] Specifically, take Fast-DetectGPT as an example, illustrated in Figure 1. We first obtain the top probabilities from the GPT model, and then use these probabilities to estimate the distribution across the entire vocabulary. The basic idea is to find empirical correlation between the top probabilities and full vocabulary probabilities. To this end, we consider parameterized Geometric distribution, Zipfian distribution, and a MLP model trained over data to model the correlation. Using PDE, we also extend methods like Entropy, Rank, and LogRank to proprietary models.

Experiments show exceptionally strong results of these white-box methods using latest LLMs, where Rank, LogRank, and Fast-DetectGPT using PDE outperform their versions using open-source LLMs with a significant margins (more than 25% relative to the remaining space). We achieve the best detection accuracies for five source models, obtaining an average accuracy of 0.98 for ChatGPT,

[1] By 'latest', we refer to the most powerful proprietary models that are widely used, like ChatGPT.

[2] Provided by Completion API of popular models like Google's PaLM-2 (Anil et al., 2023), OpenAI's GPT-3 (Brown et al., 2020), GPT-3.5 (OpenAI, 2022), and GPT-4 (OpenAI, 2023).

0.94 for GPT-4, 0.96 for Claude-3 Sonnet, 0.97 for Claude-3 Opus, and 0.92 for Gemini-1.5 Pro in the stimulated white-box setting. These results indicate that latest LLMs can detect their generations with high accuracy.

To our knowledge, we are the first to investigate white-box detection methods on proprietary models, achieving new state-of-the-art detection accuracies among zero-shot methods, demonstrating that *the most powerful LLMs may be the strongest 'shield' against themselves*.

2 METHOD

We use the state-of-the-art Fast-DetectGPT (Bao et al., 2023) as an example to illustrate how to apply PDE to existing white-box methods in Section 2.2, and present three specific PDE algorithms in Section 2.3. We further discuss three more white-box methods with PDE in Section 2.4.

2.1 TASK AND SETTINGS

We are concentrating on the zero-shot detection of LLM-generated text, which we frame as a binary classification problem: determining whether a given text was created by a model or a human. A zero-shot detector usually uses a scoring model to produce a detection metric and makes a decision by comparing this metric to a predetermined threshold. The scoring model can either be the same as the source model or a different one, for example a fixed delegate model. The access level to this scoring model can vary, and methods can be categorized into black-box or white-box methods based on this (similar to Tang et al. (2024),).

Black-box methods use API-level access to interact with the scoring model, for example proprietary GPT-3.5, while **white-box methods** assume full access to the scoring model, for example open-source Neo-2.7B. In this definition, methods like Fast-DetectGPT (Bao et al., 2023) and PHD (Tulchinskii et al., 2024), which employ a delegate open-source model to identify generations from proprietary LLMs, are categorized as white-box methods. It is crucial to note that this definition differs from Yang et al. (2023b), which defines black-box and white-box based on the access level to the source model.

2.2 FAST-DETECTGPT WITH PDE

PDE does not change the overall framework of the baseline but only replaces the model distribution p_θ with estimated distribution $\tilde{p}_\theta$ when the missing part of p_θ is required. In the following description, we rehearsal the framework with the updated distribution.

Fast-DetectGPT posits that human and machine language generation differs in word selection based on context. While machines favor words with higher model probabilities, humans do not necessarily demonstrate such tendency. This discrepancy is quantified using a metric, naming conditional probability curvature. And we decide if a text is machine-generated by comparing the metric with a threshold ϵ, which is chosen according to specific scenarios.

Formally, given a text passage x, a proprietary model p_θ, and an estimated distribution $\tilde{p}_\theta$, the *conditional probability function* defined by Fast-DetectGPT is expressed as

$$p_\theta(\tilde{x}|x) = \prod_j p_\theta(\tilde{x}_j|x_{<j}), \tag{1}$$

which denotes the predictive distribution of the model taking x as the input. As a special case, when $\tilde{x}$ equals to x, $p_\theta(x|x) = p_\theta(x)$. Take the second word position ($j = 2$) of the passage in Figure 1 as an example. Its context is $x_{<2}$ = *'President'*, and the probabilities for possible choices $\tilde{x}_2 \in$ [*'Joe'*, *'Biden'*, *'Donald'*, ...] are $[0.37, 0.24, 0.11, ...]$. The tokens $\tilde{x}_j$ for different j are independent from each other given the input. Such conditional independence allows for efficient calculation in the algorithm.

Using the conditional probability function, the *conditional probability curvature* defined by Fast-DetectGPT is written as

$$\mathbf{d}(x, p_\theta) = \frac{\log p_\theta(x) - \tilde{\mu}}{\tilde{\sigma}}, \tag{2}$$

where the token likelihoods $p_\theta(x_j|x_{<j})$ are provided directly by the proprietary model, like $p_\theta(x_2 =$ *'Joe'*$|x_{<2} =$ *'President'*$) = 0.37$, no matter whether x_j is among the top-K choices. The expected score $\tilde{\mu}$ and the expected variance $\tilde{\sigma}^2$ are computed using the estimated distribution $\tilde{p}_\theta$ as

$$\begin{aligned}\tilde{\mu} &= \mathbb{E}_{\tilde{x}\sim\tilde{p}_\theta(\tilde{x}|x)}\left[\log\tilde{p}_\theta(\tilde{x}|x)\right] = \sum_j \mathbb{E}_{\tilde{x}_j\sim\tilde{p}_\theta(\tilde{x}_j|x_{<j})}\left[\log\tilde{p}_\theta(\tilde{x}_j|x_{<j})\right] = \sum_j \tilde{\mu}_j,\\ \tilde{\sigma}^2 &= \mathbb{E}_{\tilde{x}\sim\tilde{p}_\theta(\tilde{x}|x)}\left[\left(\log\tilde{p}_\theta(\tilde{x}|x)-\tilde{\mu}\right)^2\right] = \sum_j \mathbb{E}_{\tilde{x}_j\sim\tilde{p}_\theta(\tilde{x}_j|x_{<j})}\left[\left(\log\tilde{p}_\theta(\tilde{x}_j|x_{<j})-\tilde{\mu}_j\right)^2\right] = \sum_j \tilde{\sigma}_j^2.\end{aligned} \tag{3}$$

In the expectations, $\tilde{x} \sim \tilde{p}_\theta(\tilde{x}|x)$ denotes the sampling step while $\log\tilde{p}_\theta(\tilde{x}|x)$ denotes the scoring step as illustrated in Figure 1. The expectations over the whole sequence is calculated by accumulating all token-level expectations.

The token-level mean $\tilde{\mu}_j$ signifies the entropy of the predictive distribution at the j-th token. We calculate it analytically by enumerating all possible choices

$$\tilde{\mu}_j = \mathbb{E}_{\tilde{x}_j\sim\tilde{p}_\theta(\tilde{x}_j|x_{<j})}\left[\log\tilde{p}_\theta(\tilde{x}_j|x_{<j})\right] = \sum_{\tilde{x}_j}\tilde{p}_\theta(\tilde{x}_j|x_{<j})\log\tilde{p}_\theta(\tilde{x}_j|x_{<j}), \tag{4}$$

such as $\tilde{\mu}_2 = 0.37\cdot\log 0.37 + 0.24\cdot\log 0.24 + 0.11\cdot\log 0.11 + ...$

The token-level variance $\tilde{\sigma}_j^2$ is also computed analytically as

$$\tilde{\sigma}_j^2 = \mathbb{E}_{\tilde{x}_j\sim\tilde{p}_\theta(\tilde{x}_j|x_{<j})}\left[\left(\log\tilde{p}_\theta(\tilde{x}_j|x_{<j})-\tilde{\mu}_j\right)^2\right] = \sum_{\tilde{x}_j}\tilde{p}_\theta(\tilde{x}_j|x_{<j})\log^2\tilde{p}_\theta(\tilde{x}_j|x_{<j}) - \tilde{\mu}_j^2, \tag{5}$$

where the summation over $\tilde{x}_j$ is similarly achieved by enumerating all possible choices, like $\tilde{\sigma}_2^2 = 0.37\cdot\log^2 0.37 + 0.24\cdot\log^2 0.24 + ... - \tilde{\mu}_2^2$.

2.3 Probability Distribution Estimation (PDE)

Formally, given a text sequence x, the proprietary model $p_\theta(\tilde{x}|x)$ provides us the likelihood $p_\theta(x_j|x_{<j})$ and the top-K token probabilities $p_\theta(\tilde{x}_j^k|x_{<j})|_{k=1}^K$ on each token position j, where k denotes the rank. The problem is then formulated as estimating $p_\theta(\tilde{x}_j|x_{<j})$ over the whole vocabulary according to the given information.

However, we do not need token-probability pairs for metric calculation in general. Take Fast-DetectGPT as an example. The mean $\tilde{\mu}_j$ and variance $\tilde{\sigma}_j^2$ only depend on the probability values in the distribution. That is to say, we can calculate them using only the '*probability*' column in the full distribution in Figure 1, where the '*token*' column is not necessary. Such an advantage is not specific to Fast-DetectGPT. Other methods like Entropy, Rank, and Log-Rank can also be calculated from the probabilities only.

To simplify the discussion below, we denote the probabilities as $p(k)$, omitting the expression of the token $\tilde{x}_j^k$ and position j. Using the top-K ($K = 3$ typically) probabilities $p(k)|_{k=1}^K$, we estimate the rest $p(k)|_{k=K+1}^M$, where M denotes the size of the list. It is worth noting that M is not necessarily to be as large as the vocabulary size, because large ranks generally correspond to low probabilities. When the probabilities are small enough, their effects on the metric are ignorable.

Consequently, $\tilde{\mu}_j$ and $\tilde{\sigma}_j^2$ are rewritten as

$$\begin{aligned}\tilde{\mu}_j &= \sum_{\tilde{x}_j}\tilde{p}_\theta(\tilde{x}_j|x_{<j})\log\tilde{p}_\theta(\tilde{x}_j|x_{<j}) = \sum_k p(k)\log p(k),\\ \tilde{\sigma}_j^2 &= \sum_{\tilde{x}_j}\tilde{p}_\theta(\tilde{x}_j|x_{<j})\log^2\tilde{p}_\theta(\tilde{x}_j|x_{<j}) - \tilde{\mu}_j^2 = \sum_k p(k)\log^2 p(k) - \tilde{\mu}_j^2,\end{aligned} \tag{6}$$

where the summation over all possible tokens $\tilde{x}_j$ is thus converted into the summation over all possible ranks k.

The probability distribution across ranks generally follows a decaying pattern, where the larger models tend to have a higher top-1 probability and a bigger decay factor demonstrating a sharper distribution (Appendix A). We approximate the pattern using parameterized distributions, allocating the remaining probability mass (excluding top-K probabilities) to ranks larger than K.

Each parameterized distribution represents a family of distributions controlled by some parameters. Traditionally, we fit the distribution to data points to estimate the parameter, where we could use the top-K probabilities as the data points. However, this approach may under-utilize the observed information and infer top-K probabilities different from the real ones.

In this study, we consider the estimation problem in constraints. The basic constraints include: 1) *total probability constraint* – a summation of the probabilities equals 1. 2) *monotonic decrease constraint* – the probability of a larger rank is lower. Using the total probability constraint, we decide the probability mass to allocate to ranks larger than K. Using the monotone decrease constraint, we decide a suitable family of distributions. We discuss three specific estimation algorithms as follows.

Estimation Using Geometric Distribution. As the simplest decaying pattern, we consider exponential decay with a fixed decay factor, resulting in a Geometric distribution. We extend the distribution to multiple top probabilities and a limited range of k. We express the probabilities for ranks larger than K in near-Geometric distribution that

$$\begin{cases} p(k) = p_k, & \text{for } k \in [1..K] \\ p(k) = p_K \cdot \lambda^{k-K}, & \text{for } k \in [K+1..M] \\ \sum_{k=1}^{M} p(k) = 1, \end{cases} \tag{7}$$

where λ is a decay factor in $(0, 1)$, and M is the size of the rank list. We solve λ under the constraints as detailed in Appendix A.1.

Estimation Using Zipfian Distribution. Frequencies of words in natural languages usually adhere to Zipf's law (Zipf, 1946; 2013), where the word frequency and word rank follow a Zipfian distribution. Assuming that the word frequencies in a given context also comply with this law, we consider it as an alternative distribution for our estimation. Given the top-K probabilities p_k, we compute the probabilities of tokens with a ranking greater than K in a Zipfian distribution

$$\begin{cases} p(k) = p_k, & \text{for } k \in [1..K] \\ p(k) = p_K \cdot (\frac{\beta}{k-K+\beta})^{\alpha}, & \text{for } k \in [K+1..M] \\ \sum_{k=1}^{M} p(k) = 1, \end{cases} \tag{8}$$

where α and β are two positive parameters. M is the rank-list size. To solve the two parameters, we introduce an additional loss function to minimize their deviations from typical values. We identify the best α and β by searching a loss table $\text{Loss}(\alpha, \beta)$ as detailed in Appendix A.2.

Estimation Using a MLP Model. The Geometric and Zipfian algorithms both work on assumptions about the distributions. An alternate approach that does not rely on these assumptions involves modeling the distribution within a neural network. We consider the simple Multi-Layer Perceptron (MLP) model with a single hidden layer, which accepts the top-K probabilities and predicts the probabilities for the rest of the ranks. The distribution is expressed as

$$\begin{cases} p(k) = p_k, & \text{for } k \in [1..K] \\ p(k) = p_{\text{rest}} \cdot p_{\text{MLP}_\theta}(k-K), & \text{for } k \in [K+1..M] \\ \sum_{k=1}^{M} p(k) = 1, \end{cases} \tag{9}$$

where $p_{rest} = 1 - \sum_{k=1}^{K} p_k$ and p_{MLP_θ} represents the MLP predictive distribution. The MLP is defined in detail, including training and inference, in Appendix A.3.

2.4 Universality of PDE

PDE can also be used by other zero-shot detection methods like Entropy, Rank, and LogRank. For Entropy, the calculation is straight-forward by summing $p(k) \cdot \log p(k)$ over all items in the rank list. For Rank and LogRank, we decide the rank of current token by searching the closest $p(k)$ compared to current token probability $p_\theta(x_j|x_{<j})$. In this study, we only examine the basic white-box methods, leaving the integration of more sophisticated white-box methods and additional estimation algorithms to future research.

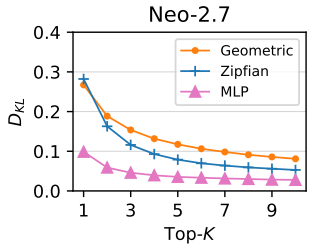


Figure 2: *KL divergence* against real distributions from Neo-2.7B.

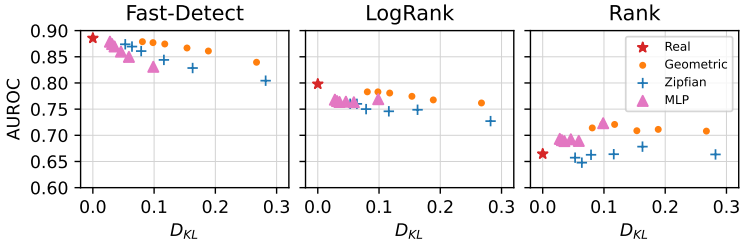


Figure 3: Correlation between *AUROC* and *KL divergence*, evaluated on XSum produced by GPT-4. We use the open-source model Neo-2.7B as the scoring model for PDE algorithms.

3 EXPERIMENTS

3.1 SETTINGS

We evaluate our methods on four datasets covering seven languages, five source models covering three model families, and four scoring models from small to large. We run each main experiment three times and report the median. The metrics used are described in Appendix B.1.

Datasets. We follow studies (Mitchell et al., 2023; Bao et al., 2023; Yang et al., 2023a; Zeng et al., 2024a) on the evaluation datasets and their settings: *XSum* for news (Narayan et al., 2018), *Writing* for story (Fan et al., 2018), and *PubMed* for technical question answering (Jin et al., 2019) , where for each dataset 150 human-written samples are randomly selected and corresponding LLM texts are generated using the same prefix (30 tokens). Additionally, we use *M4* (Wang et al., 2024) to incorporate various languages such as *Chinese*, *Russian*, *Urdu*, *Indonesian*, *Arabic*, and *Bulgarian* , where for each language 150 pairs are randomly selected from the ChatGPT subsets. To evaluate the detectors ability to handle diverse domains and languages, we combine XSum, Writing, and PubMed into a single dataset called *Mix3*, and the six language datasets into another called *Mix6*.

Source Models. We evaluate our detector on five latest LLMs from different companies, including *ChatGPT* (gpt-3.5-turbo) (OpenAI, 2022), *GPT-4* (gpt-4) (OpenAI, 2023), *Claude3 Sonnet* (claude-3-sonnet-20240229) and *Opus* (claude-3-opus-20240229) (Anthropic, 2024), *Gemini-1.5 Pro* (gemini-1.5-pro) (Team et al., 2023), where Opus is supposed at the same level as GPT-4 and Sonnet at the same level as ChatGPT. We use ChatCompletion API [3] of these models to prepare the datasets.

Scoring Models. A good scoring model can detect generations from a wide range of source models. We use OpenAI GPT series (from small to large) as the scoring model, inlcuding *Babbage* (babbage-002, 1.3B) and *Davinci* (davinci-002, 175B) (Brown et al., 2020), *GPT-3.5* (gpt-35-turbo-0301 or gpt-35-turbo-1106 almost equally, 175B) (OpenAI, 2022), and *GPT-4* (gpt-4-1106) (OpenAI, 2023) as scoring models using AzureOpenAI (see Appendix B.2). For comparison, we use *Neo-2.7* (gpt-neo-2.7B) (Black et al., 2021), *Phi2-2.7B* (Javaheripi et al., 2023), *Qwen2.5-7B* (Yang et al., 2024; Team, 2024), and *Llama3-8B* (Dubey et al., 2024) as representatives of open-source models, run locally on a Tesla A100 GPU.

Baselines. Among zero-shot detectors, we compare our methods with existing solutions such as *Fast-DetectGPT* (shortly Fast-Detect) (Bao et al., 2023), *DetectGPT* (Mitchell et al., 2023), *DNA-GPT* (Yang et al., 2023a), and simple baselines such as *Likelihood*, *Entropy*, *Rank*, and *Log-Rank* (Gehrmann et al., 2019; Solaiman et al., 2019; Ippolito et al., 2020). For other detectors, we compare our methods to commercial *GPTZero* (Tian & Cui, 2023).

Hyper-Parameters. By default, we use top-5 probabilities, a rank-list size of 1000 for Geometric and 100 for Zipfian and MLP, prompt4 from Table 7 for GPT-4 and prompt3 for other scoring models. These settings are ablated in Section 3.4 with one hyper-parameter changed each time, where we report the average accuracy over XSum, Writing, and PubMed.

[3] https://platform.openai.com/docs/guides/text-generation/chat-completions-api

Table 2: *Main results using PDE*, where the columns '*GPT-4*', '*Claude-3*', and '*Gemini-1.5*' display the AUROCs on the diverse mixture of the three datasets (Mix3), with the detailed results for GPT-4 and Gemini-1.5 in Table 4 and Claude-3 in Table 5 in Appendix C. We mark the best in each column in **bold**. Methods marked with ♢ require multiple evaluations or generations to detect one passage, thereby, consuming multiple times of cost and time. Additionally, we make a comparison of ACC in Table 6.

Method	ChatGPT			ChatGPT	GPT-4	Claude-3		Gemini-1.5	Avg.
	XSum	Writing	PubMed	Mix3	Mix3	Sonnet	Opus/Mix3	Pro/Mix3	Mix3
GPTZero	0.9843	0.9303	0.8403	0.9143	0.9009	-	-	-	-
Zero-Shot Detectors Using White-Box Open-Source LLMs									
Likelihood (Neo-2.7)	0.9578	0.9740	0.8775	0.9071	0.7690	0.8661	0.9030	0.7416	0.8373
Entropy (Neo-2.7)	0.3305	0.1902	0.2767	0.3136	0.4114	0.3466	0.3265	0.3959	0.3588
Rank (Neo-2.7)	0.7494	0.8064	0.5979	0.7044	0.6448	0.6888	0.7056	0.6260	0.6739
LogRank (Neo-2.7)	0.9582	0.9656	0.8687	0.9059	0.7626	0.8654	0.9042	0.7353	0.8347
DNA-GPT (Neo-2.7) ♢	0.9124	0.9425	0.7959	0.7192	0.6430	0.7080	0.7326	0.6438	0.6893
DetectGPT (T5-11B/Neo-2.7) ♢	0.8416	0.8811	0.7444	0.7826	0.6136	0.7967	0.7776	0.7406	0.7422
Fast-Detect (GPT-J/Neo-2.7)	0.9907	0.9916	0.9021	0.9487	0.8999	0.9260	0.9468	0.8072	0.9057
Fast-Detect (Phi2-2.7B)	0.8096	0.7245	0.8121	0.7627	0.5742	0.6957	0.7450	0.6164	0.6788
Fast-Detect (Qwen2.5-7B)	0.7808	0.8117	0.7887	0.7655	0.6862	0.7813	0.8119	0.6839	0.7458
Fast-Detect (Llama3-8B)	0.8508	0.8446	0.7941	0.7796	0.7269	0.8212	0.8510	0.7552	0.7868
Zero-Shot Detectors Using Black-Box Proprietary LLMs									
Likelihood (GPT-3.5)	0.9203	0.9925	0.9544	0.9246	0.8029	0.9023	0.9295	0.8043	0.8727
DNA-GPT (GPT-3.5) ♢	0.9260	0.9329	0.9304	0.8369	0.7748	0.7871	0.8383	0.7107	0.7896
PDE using Geometric									
Entropy (GPT-3.5)	0.3188	0.0463	0.1793	0.2160	0.4074	0.2582	0.2339	0.4144	0.3060
Rank (GPT-3.5)	0.8577	0.9845	0.8383	0.8533	0.7395	0.8473	0.8645	0.7406	0.8090
LogRank (GPT-3.5)	0.9240	0.9931	0.9532	0.9277	0.7870	0.9062	0.9336	0.7872	0.8683
Fast-Detect (Babbage)	**0.9908**	0.9904	0.9570	0.9764	0.8974	0.9438	0.9698	0.8083	0.9191
Fast-Detect (Davinci)	0.9900	**0.9976**	0.9421	0.9763	0.9131	0.9606	0.9742	0.8601	0.9369
Fast-Detect (GPT-3.5)	0.9808	0.9972	**0.9702**	0.9766	0.9411	0.9576	0.9689	**0.9244**	0.9537
Fast-Detect (GPT-4)	0.9815	0.9935	0.9564	0.9735	0.9647	0.9623	**0.9817**	0.8947	0.9554
PDE using Zipfian									
Fast-Detect (GPT-3.5)	0.9826	0.9956	0.9639	0.9647	0.9319	0.9475	0.9588	0.9161	0.9438
Fast-Detect (GPT-4)	0.9885	0.9917	0.9461	0.9768	**0.9719**	0.9613	0.9792	0.8991	0.9576
PDE using MLP									
Fast-Detect (GPT-3.5)	0.9819	0.9959	0.9676	0.9702	0.9342	0.9526	0.9634	0.9184	0.9478
Fast-Detect (GPT-4)	0.9869	0.9930	0.9528	**0.9771**	0.9705	**0.9631**	0.9807	0.9001	**0.9583**

3.2 The Effectiveness of PDE

We assess the effectiveness of PDE by comparing the estimated distributions to the real distributions, using Neo-2.7B as the scoring model. As Figure 2 shows, their Kullback–Leibler (KL) divergence decreases as long as more top probabilities are used. Overall, MLP obtains the lowest divergence while Geometric distribution has the highest divergence, suggesting the most accurate estimation of MLP. However, the correlation between the KL divergences and detection accuracies varies for different detection methods and estimation algorithms. As Figure 3 illustrates, the AUROC *declines* when the divergence increases for PDE (Fast-Detect and LogRank) with any of the estimation algorithms, while the AUROC *inclines* for PDE (Rank). Although Geometric distribution has larger divergences in general, it achieves higher or equal detection accuracies than other algorithms, suggesting the effect of PDE is beyond the similarity of estimated distributions.

It is worth noting that the accuracies achieved by PDE are not significantly lower than the baseline using real distribution for Fast-Detect and LogRank, and are even significantly higher than the baseline for Rank, suggesting the effectiveness and potential of PDE.

3.3 Main Results

Accuracy and Efficiency. We first compare PDE with existing black-box methods on their detection accuracy, speed, and cost. As Table 2 shows, Fast-Detect (GPT-3.5) surpasses both Likelihood (GPT-3.5) and DNA-GPT (GPT-3.5) with a significant margin across five source models (in AUROC). The basic method LogRank (GPT-3.5) also outperforms DNA-GPT on four-fifth source models, demonstrating the effectiveness and universality of PDE. Furthermore, PDE methods spend significantly less time and cost than DNA-GPT during detection process. Specifically, DNA-GPT takes a total of 1911 seconds across the three datasets, while PDE takes just 462 seconds, making the detection process 4.1 times faster. Since DNA-GPT creates 10 completions per passage but PDE

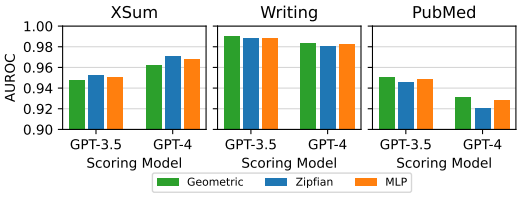


Figure 4: Ablation on *estimation algorithm*, where the AUROC is averaged across the five source models. Each dataset has its own preferred algorithm.

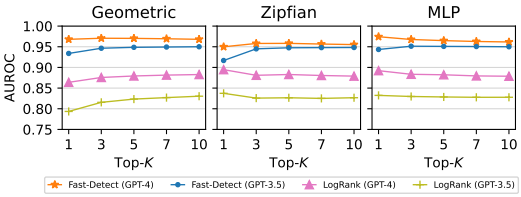


Figure 5: Ablation on *top-K*, where the AUROC is averaged across the datasets produced by GPT-4. Each line represents a combination of methods and scoring models.

merely echos the probabilities of the input, DNA-GPT ends up costing about 10 times more than PDE based on current pricing (where the cost per output token is twice of input token)[4].

Here, we skip comparison with DetectGPT in the black-box setting, because it requires 100x times API calls and is affirmed of inferior accuracies compared to DNA-GPT (Yang et al., 2023a) and Fast-Detect (Bao et al., 2023), which are already our baselines.

Latest LLMs are more accurate detectors. We then compare our method with other existing methods. Powered by PDE, the white-box methods such as Rank, LogRank, and Fast-Detect using GPT-3.5 significantly outperforms their open-source version using Neo-2.7, with a relative improvement of 41%, 21%, and 51%, respectively. PDE methods with smaller models also show competitive results. Fast-Detect (Babbage) outperforms Fast-Detect (Neo-2.7) on all five source models, even though Babbage (1.3B) is smaller than Neo-2.7 (2.7B). These results demonstrate that the latest LLMs with PDE are strong detectors.

Larger LLMs can be universal detectors. A recent study (Mireshghallah et al., 2023) reports that smaller LLMs are more efficient universal detectors, which can detect generations from various source models. However, our experimental results suggest that larger models can also be universal detectors, which perform better than smaller proprietary and open-source models across various source models. Specifically, GPT-3.5 (175B) reach an average AUROC of about 0.95, outperforming smaller Babbage (1.3B) by about ↑ 43% and Neo-2.7 (2.7B) by about ↑ 51%. These findings suggest that larger LLMs, when equipped with a right technique, can also serve as universal detectors.

3.4 Ablation Study

Ablation on Estimation Algorithm. As the main results in Table 2 shows, PDEs using Geometric, Zipfian, and MLP distributions do not show significant differences in their average accuracies. However, when we look into each dataset, we see different patterns as Figure 4 demonstrates. Specifically, Geometric distribution performs the best on Writing and PubMed, while Zipfian performs the best on XSum. MLP is more balanced, which performs the median on three datasets. These patterns hold with both GPT-3.5 and GPT-4 as the scoring model. These results suggest that different estimation algorithms may suit for different situations, which is also supported by experiments on languages shown in Table 3. In addition, we justify the necessity of PDE by comparing it with a Naive estimation algorithm in Appendix D.1

Ablation on Top-K. Intuitively, the greater the K value is, the more precise the estimated distribution is likely to be. We conduct an ablation study on two representative methods with two scoring models as shown in Figure 5. The results generally corroborate the intuition, particularly with Geometric distribution. However, Zipfian and MLP show exceptions, where LogRank (GPT-4 and GPT-3.5) with Zipfian and Fast-Detect and LogRank (GPT-4) with MLP obtain their highest accuracies with the top-1 setting. The results indicate that even with limited top-1 probabilities, PDE can still work effectively when a proper estimation algorithm is used.

Ablation on Rank-List Size. The size of the rank list is a crucial hyper-parameter affecting Geometric and Zipfian distributions. Larger size typically results in higher accuracy. Geometric distribution shows consistent increase, while Zipfian shows varying trends. Preliminary tests with MLP on sizes of 100 and 1000 suggest similar patterns to Zipfian. See Appendix D.2 for more details.

[4] https://azure.microsoft.com/en-us/pricing/details/cognitive-services/openai-service/

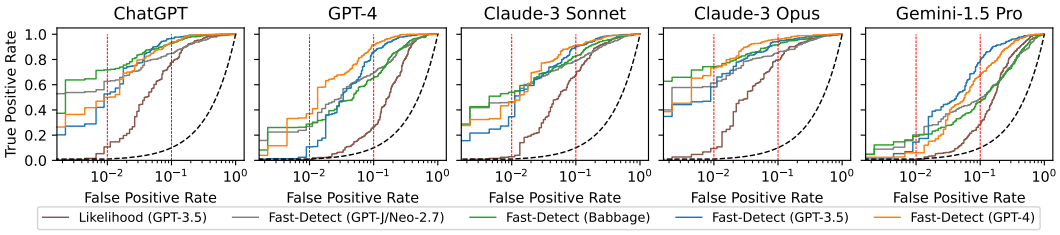


Figure 6: Robustness on *low false alarms*, where the red lines indicate false alarms of 1% and 10%. We draw the curves using Mix3 to simulate the real scenarios of using a single threshold to detect different domains. The dash lines denote the random classifier.

Table 3: Robustness over *languages*, where we use the same settings as the main experiments. Mix6 denotes the diverse mixture of the six languages.

Method	Chinese (Web QA)	Russian (RuATD)	Urdu (News)	Indonesian (News)	Arabic (Wikipedia)	Bulgarian (News)	Mix6
Fast-Detect (GPT-J/Neo-2.7B)	0.9319	0.8158	0.9630	0.9876	0.9121	0.9422	0.8862
Fast-Detect (Phi2-2.7B)	0.8024	0.6710	0.9049	0.9313	0.9155	0.7500	0.8205
Fast-Detect (Qwen2.5-7B)	0.5345	0.7065	0.9970	0.9687	0.8404	0.8992	0.8063
Fast-Detect (Llama3-8B)	0.8729	0.7984	0.9962	0.9922	0.9282	0.9723	0.8713
PDE using Geometric							
Fast-Detect (Babbage)	0.9814	0.7673	0.9894	0.9857	0.9916	0.9628	0.8950
Fast-Detect (Davinci)	**0.9938**	0.8030	0.9996	0.9991	0.9982	0.9842	0.9369
Fast-Detect (GPT-3.5)	0.9913	0.8555	**1.0000**	0.9996	**0.9999**	**0.9925**	**0.9774**

Ablation on Prompt. We examine how different prompts affect text detection accuracy in large models. The most sensitive model is GPT-4, with accuracies varying greatly with different prompts. GPT-3.5 is less sensitive, with its accuracies increasing slightly from with prompt0 to with prompt4. The base models Babbage and Davinci are least influenced by prompt changes. See Appendix D.3 for detailed description.

3.5 Analysis and Discussion

Robustness across Source Models and Domains. In real scenarios, we tend to use a constant threshold to detect text from various sources and domains. We evaluate the stability of PDE and other baselines across the source models and datasets using thresholds found on "out-of-domain" datasets. The results show that PDE consistently provides the highest accuracy across the source models and domains, showcasing its stability. See Appendix E.1 for detailed setup and discussion.

Robustness on Low False Alarms. In real scenarios, an effective detector is expected to have a high recall (true positive rate) with a low false alarm (false positive rate), thereby allowing it to identify most machine-generated text without misclassifying human-written text. We evaluate the capacity of the methods by contrasting their ROC curves. As illustrated in Figure 6, for a false alarm in the range of $(0.01, 0.1)$, Fast-Detect (GPT-4) performs the best except on ChatGPT generations, where Fast-Detect (Babbage) has the highest recalls. For a false alarm lower than 0.001, Fast-Detect (Babbage) performs consistently better than other methods, suggesting the advantage of it for low false alarm setting. Compared to baseline Likelihood (GPT-3.5) and Fast-Detect (Neo-2.7), PDE versions of Fast-Detect show consistent advantage.

Robustness over Languages. We assessment PDE methods on M4 datasets with six languages. As Table 3 shows, Fast-Detect (GPT-3.5) consistently outperforms the baselines as well as other proprietary LLMs. The system almost flawlessly identifies Urdu, Indonesian, and Arabic texts. However, the detection accuracy for Russian texts is much lower, indicating a potential under-training of the LLMs on this particular language. These findings imply that PDE with latest LLMs can be effective and reliable detectors across languages.

Robustness under Paraphrasing Attack. We test the performance of PDE against paraphrasing attack using DIPPER, with two settings: high lexical diversity and high order diversity. Fast-Detect (Babbage) outperforms Fast-Detect (Neo-2.7) in both, but is more affected by diverse lexicons. An unusual behavior of DIPPER is observed on XSum, which causes exceptionally low accuracy of Likelihood (GPT-3.5) baseline. PDE surpasses Fast-Detect (Neo-2.7) and trained detectors, prov-

ing its effectiveness. Larger LLMs are more vulnerable to increased diversity, which also reduce readability. See Appendix E.2 for detailed comparison and discussion.

Limitations. PDE does not support all white-box methods, especially the rare methods who use the inner embeddings instead of predictive distributions, like PHD (Tulchinskii et al., 2024). Additionally, not all proprietary models provide Completion API, therefore, PDE cannot use them as the scoring model.

Broader Impact. The estimation methods can potentially be applied to other scenarios. For example, ChatCompletion API also provides top-K probabilities. Using PDE, we can estimate the predictive distribution, which could be used to calculate some statistical metrics about the generated content. Such metrics could potentially be used to indicate, for instance, the level of hallucination in the content. Despite the effectiveness and potential of PDE, unsupervised use can lead to potential unfairness towards certain groups. For instance, detectors might display bias against non-native writers (Liang et al., 2023).

4 Related Work

In terms of using a proprietary model for scoring, our method is in line with existing black-box methods. However, it inherits existing white-box approaches.

Black-Box Methods for Zero-Shot Detection. Zero-shot detection in the black-box setting is challenging due to the restricted access of the model. DetectGPT (Mitchell et al., 2023) and its improvement NPR (Su et al., 2023) requires multiple evaluations of text sequences, while DNA-GPT (Yang et al., 2023a) requires multiple generations of text sequences, resulting in low speed and high cost. Another line of approaches treats detection as a question-answering task, but fail to reliably discern text generated by ChatGPT and GPT-4 (Bhattacharjee & Liu, 2024). These methods generally presume the knowing of the source model and use it to ascertain if a text was produced by it, which limits their usage to texts from uncertain origin. Unlike these approaches, our method minimizes the cost and dependence on known source.

White-Box Methods for Zero-Shot Detection. Zero-shot methods in the white-box setting analyze a variety of metrics from the model predictive distributions or output embeddings, including Entropy and Perplexity (Lavergne et al., 2008), Likelihood (Hashimoto et al., 2019; Solaiman et al., 2019), Rank and Log-Rank (Gehrmann et al., 2019), LRR (Su et al., 2023), Fast-DetectGPT (Bao et al., 2023), PHD (Tulchinskii et al., 2024), FourierGPT (Xu et al., 2024), and Binoculars (Hans et al.). These methods, except PHD requires output embeddings, can all be applied to proprietary models using PDE. However, in this study we focus on very basic Entropy, Rank, and Log-Rank, together with recent Fast-DetectGPT, leaving the rest for future exploration.

Other Detection Methods. Trained classifiers are the majority of black-box methods, which rely human-authored texts along with LLM-generated content for training (Bakhtin et al., 2019; Uchendu et al., 2020; Solaiman et al., 2019; Ippolito et al., 2020; Fagni et al., 2021; Solaiman et al., 2019; Fagni et al., 2021; Yan et al., 2023; Li et al., 2023; Zeng et al., 2024b) (Verma et al., 2024; Kushnareva et al., 2024). But they can become overly specialized to their training conditions such as specific domains, languages, or source models. White-box approaches like watermarking provide a different strategy, altering the LLMs decoding process to include unique text signatures (Kirchenbauer et al., 2023; Kuditipudi et al., 2023; Christ et al., 2024; Zhao et al., 2023b;a). However, they require proactive text generation injection, which is not allowed by proprietary models. In this paper, we investigate techniques that do not need training or proactive injection.

5 Conclusion

We proposed probability distribution estimation (PDE) to estimate full distributions from partial observations, enabling white-box text detection methods to proprietary models. Experiments show that PDE with latest LLMs performs significantly better than its counterparts in terms of accuracy, efficiency, and robustness, highlighting the benefits of using latest LLMs as detectors. Additional results on various white-box methods underscore the effectiveness of PDE, which may lead to a new direction of zero-shot detection in the black-box setting. To our knowledge, we are the first to enable white-box methods to proprietary LLMs.

ETHICAL STATEMEMT

High efficient and accurate machine-generated text detector can potentially contribute to trustworthy AI techniques, which can mitigate the risks bringing by the uncontrolled usage of LLMs. Such technology can potentially benefit text readers, policy makers, and media platforms in wide.

REFERENCES

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

Anthropic. The claude 3 model family: Opus, sonnet, haiku. `https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf`, 2024.

Anton Bakhtin, Sam Gross, Myle Ott, Yuntian Deng, Marc'Aurelio Ranzato, and Arthur Szlam. Real or fake? learning to discriminate machine from human generated text. *arXiv preprint arXiv:1906.03351*, 2019.

Guangsheng Bao, Yanbin Zhao, Zhiyang Teng, Linyi Yang, and Yue Zhang. Fast-detectgpt: Efficient zero-shot detection of machine-generated text via conditional probability curvature. In *The Twelfth International Conference on Learning Representations*, 2023.

Amrita Bhattacharjee and Huan Liu. Fighting fire with fire: can chatgpt detect ai-generated text? *ACM SIGKDD Explorations Newsletter*, 25(2):14–21, 2024.

Sid Black, Gao Leo, Phil Wang, Connor Leahy, and Stella Biderman. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, March 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Miranda Christ, Sam Gunn, and Or Zamir. Undetectable watermarks for language models. In *The Thirty Seventh Annual Conference on Learning Theory*, pp. 1125–1139. PMLR, 2024.

Jon Christian. Cnet secretly used ai on articles that didn't disclose that fact, staff say. *Futurusm, January*, 2023.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Liam Dugan, Daphne Ippolito, Arun Kirubarajan, Sherry Shi, and Chris Callison-Burch. Real or fake text?: Investigating human ability to detect boundaries between human-written and machine-generated text. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 12763–12771, 2023.

Tiziano Fagni, Fabrizio Falchi, Margherita Gambini, Antonio Martella, and Maurizio Tesconi. Tweepfake: About detecting deepfake tweets. *Plos one*, 16(5):e0251415, 2021.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2018.

Sebastian Gehrmann, Hendrik Strobelt, and Alexander M Rush. Gltr: Statistical detection and visualization of generated text. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 111–116, 2019.

Abhimanyu Hans, Avi Schwarzschild, Valeriia Cherepanova, Hamid Kazemi, Aniruddha Saha, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Spotting llms with binoculars: Zero-shot detection of machine-generated text. In *Forty-first International Conference on Machine Learning*.

Tatsunori B Hashimoto, Hugh Zhang, and Percy Liang. Unifying human and statistical evaluation for natural language generation. *arXiv preprint arXiv:1904.02792*, 2019.

Daphne Ippolito, Daniel Duckworth, Chris Callison-Burch, and Douglas Eck. Automatic detection of generated text is easiest when humans are fooled. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 1808–1822, 2020.

Mojan Javaheripi, Sébastien Bubeck, Marah Abdin, Jyoti Aneja, Sebastien Bubeck, Caio César Teodoro Mendes, Weizhu Chen, Allie Del Giorno, Ronen Eldan, Sivakanth Gopi, et al. Phi-2: The surprising power of small language models. *Microsoft Research Blog*, 1:3, 2023.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. Pubmedqa: A dataset for biomedical research question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2567–2577, 2019.

Davinder Kaur, Suleyman Uslu, Kaley J Rittichier, and Arjan Durresi. Trustworthy artificial intelligence: a review. *ACM Computing Surveys (CSUR)*, 55(2):1–38, 2022.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. *arXiv preprint arXiv:2301.10226*, 2023.

Kalpesh Krishna, Yixiao Song, Marzena Karpinska, John Wieting, and Mohit Iyyer. Paraphrasing evades detectors of ai-generated text, but retrieval is an effective defense. *Advances in Neural Information Processing Systems*, 36, 2024.

Rohith Kuditipudi, John Thickstun, Tatsunori Hashimoto, and Percy Liang. Robust distortion-free watermarks for language models. *arXiv preprint arXiv:2307.15593*, 2023.

Laida Kushnareva, Tatiana Gaintseva, German Magai, Serguei Barannikov, Dmitry Abulkhanov, Kristian Kuznetsov, Eduard Tulchinskii, Irina Piontkovskaya, and Sergey Nikolenko. Ai-generated text boundary detection with roft. In *1st Conference on Language Modeling (COLM)*, volume 2024, 2024.

Thomas Lavergne, Tanguy Urvoy, and François Yvon. Detecting fake content with relative entropy scoring. In *Proceedings of the 2008 International Conference on Uncovering Plagiarism, Authorship and Social Software Misuse-Volume 377*, pp. 27–31, 2008.

Yafu Li, Qintong Li, Leyang Cui, Wei Bi, Zhilin Wang, Longyue Wang, Linyi Yang, Shuming Shi, and Yue Zhang. Mage: Machine-generated text detection in the wild. *arXiv e-prints*, pp. arXiv–2305, 2023.

Weixin Liang, Mert Yuksekgonul, Yining Mao, Eric Wu, and James Zou. Gpt detectors are biased against non-native english writers. *Patterns*, 4(7), 2023.

Muneer M Alshater. Exploring the role of artificial intelligence in enhancing academic performance: A case study of chatgpt. *Available at SSRN*, 2022.

Jesse G Meyer, Ryan J Urbanowicz, Patrick CN Martin, Karen O'Connor, Ruowang Li, Pei-Chen Peng, Tiffani J Bright, Nicholas Tatonetti, Kyoung Jae Won, Graciela Gonzalez-Hernandez, et al. Chatgpt and large language models in academia: opportunities and challenges. *BioData Mining*, 16(1):20, 2023.

Niloofar Mireshghallah, Justus Mattern, Sicun Gao, Reza Shokri, and Taylor Berg-Kirkpatrick. Smaller language models are better black-box machine-generated text detectors. *arXiv preprint arXiv:2305.09859*, 2023.

Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D Manning, and Chelsea Finn. Detectgpt: Zero-shot machine-generated text detection using probability curvature. *arXiv preprint arXiv:2301.11305*, 2023.

Shashi Narayan, Shay B Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1797–1807, 2018.

OpenAI. ChatGPT. https://chat.openai.com/, December 2022.

OpenAI. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023.

Yikang Pan, Liangming Pan, Wenhu Chen, Preslav Nakov, Min-Yen Kan, and William Yang Wang. On the risk of misinformation pollution with large language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Steven T Piantadosi. Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic bulletin & review*, 21:1112–1130, 2014.

Vinu Sankar Sadasivan, Aounon Kumar, Sriram Balasubramanian, Wenxiao Wang, and Soheil Feizi. Can ai-generated text be reliably detected? *arXiv preprint arXiv:2303.11156*, 2023.

Wajiha Shahid, Yiran Li, Dakota Staples, Gulshan Amin, Saqib Hakak, and Ali Ghorbani. Are you a cyborg, bot or human?—a survey on detecting fake news spreaders. *IEEE Access*, 10: 27069–27083, 2022.

Irene Solaiman, Miles Brundage, Jack Clark, Amanda Askell, Ariel Herbert-Voss, Jeff Wu, Alec Radford, Gretchen Krueger, Jong Wook Kim, Sarah Kreps, et al. Release strategies and the social impacts of language models. *arXiv preprint arXiv:1908.09203*, 2019.

Jinyan Su, Terry Yue Zhuo, Di Wang, and Preslav Nakov. Detectllm: Leveraging log rank information for zero-shot detection of machine-generated text. *arXiv preprint arXiv:2306.05540*, 2023.

Lichao Sun, Yue Huang, Haoran Wang, Siyuan Wu, Qihui Zhang, Chujie Gao, Yixin Huang, Wenhan Lyu, Yixuan Zhang, Xiner Li, et al. Trustllm: Trustworthiness in large language models. *arXiv preprint arXiv:2401.05561*, 2024.

Kaito Taguchi, Yujie Gu, and Kouichi Sakurai. The impact of prompts on zero-shot detection of ai-generated text. *arXiv preprint arXiv:2403.20127*, 2024.

Ruixiang Tang, Yu-Neng Chuang, and Xia Hu. The science of detecting llm-generated text. *Communications of the ACM*, 67(4):50–59, 2024.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

Edward Tian and Alexander Cui. Gptzero: Towards detection of ai-generated text using zero-shot and supervised methods, 2023. URL https://gptzero.me.

Eduard Tulchinskii, Kristian Kuznetsov, Laida Kushnareva, Daniil Cherniavskii, Sergey Nikolenko, Evgeny Burnaev, Serguei Barannikov, and Irina Piontkovskaya. Intrinsic dimension estimation for robust detection of ai-generated texts. *Advances in Neural Information Processing Systems*, 36, 2024.

Adaku Uchendu, Thai Le, Kai Shu, and Dongwon Lee. Authorship attribution for neural text generation. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 8384–8395, 2020.

Vivek Verma, Eve Fleisig, Nicholas Tomlin, and Dan Klein. Ghostbuster: Detecting text ghostwritten by large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1702–1717, 2024.

Yuxia Wang, Jonibek Mansurov, Petar Ivanov, Jinyan Su, Artem Shelmanov, Akim Tsvigun, Chenxi Whitehouse, Osama Mohammed Afzal, Tarek Mahmoud, Toru Sasaki, et al. M4: Multi-generator, multi-domain, and multi-lingual black-box machine-generated text detection. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1369–1407, 2024.

Laura Weidinger, John Mellor, Maribeth Rauh, Conor Griffin, Jonathan Uesato, Po-Sen Huang, Myra Cheng, Mia Glaese, Borja Balle, Atoosa Kasirzadeh, et al. Ethical and social risks of harm from language models. *arXiv preprint arXiv:2112.04359*, 2021.

Yang Xu, Yu Wang, Hao An, Zhichen Liu, and Yongyuan Li. Detecting subtle differences between human and model languages using spectrum of relative likelihood. *arXiv preprint arXiv:2406.19874*, 2024.

Duanli Yan, Michael Fauss, Jiangang Hao, and Wenju Cui. Detection of ai-generated essays in writing assessment. *Psychological Testing and Assessment Modeling*, 65(2):125–144, 2023.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

Xianjun Yang, Wei Cheng, Linda Petzold, William Yang Wang, and Haifeng Chen. Dna-gpt: Divergent n-gram analysis for training-free detection of gpt-generated text. *arXiv preprint arXiv:2305.17359*, 2023a.

Xianjun Yang, Liangming Pan, Xuandong Zhao, Haifeng Chen, Linda Petzold, William Yang Wang, and Wei Cheng. A survey on detection of llms-generated content. *arXiv preprint arXiv:2310.15654*, 2023b.

Ann Yuan, Andy Coenen, Emily Reif, and Daphne Ippolito. Wordcraft: story writing with large language models. In *27th International Conference on Intelligent User Interfaces*, pp. 841–852, 2022.

Cong Zeng, Shengkun Tang, Xianjun Yang, Yuanzhou Chen, Yiyou Sun, Yao Li, Haifeng Chen, Wei Cheng, Dongkuan Xu, et al. Dlad: Improving logits-based detector without logits from black-box llms. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024a.

Cong Zeng, Shengkun Tang, Xianjun Yang, Yuanzhou Chen, Yiyou Sun, Yao Li, Haifeng Chen, Wei Cheng, Dongkuan Xu, et al. Improving logits-based detector without logits from black-box llms. *arXiv preprint arXiv:2406.05232*, 2024b.

Xuandong Zhao, Prabhanjan Ananth, Lei Li, and Yu-Xiang Wang. Provable robust watermarking for ai-generated text. *arXiv preprint arXiv:2306.17439*, 2023a.

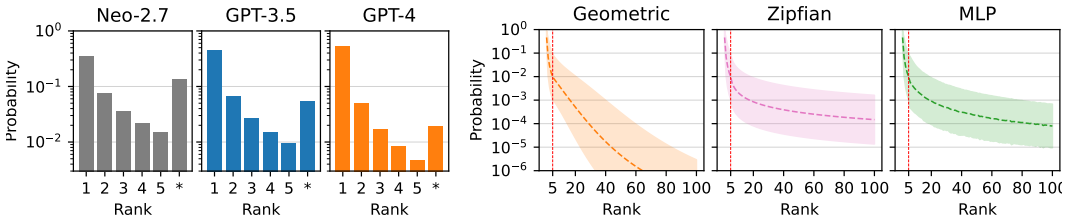
Xuandong Zhao, Yu-Xiang Wang, and Lei Li. Protecting language generation models via invisible watermarking. In *International Conference on Machine Learning*, pp. 42187–42199. PMLR, 2023b.

George Kingsley Zipf. The psychology of language. In *Encyclopedia of psychology*, pp. 332–341. Philosophical Library, 1946.

George Kingsley Zipf. *The psycho-biology of language: An introduction to dynamic philology*. Routledge, 2013.

A PROBABILITY DISTRIBUTION ESTIMATION (PDE)

As indicated in Figure 7a, the probability distribution across ranks generally follows a decaying pattern, where the larger models tend to have a higher top-1 probability and a bigger decay factor demonstrating a sharper distribution. We approximate the pattern using parameterized distributions, allocating the remaining probability mass (as '*' indicates) to ranks larger than K. We discuss three specific estimation algorithms with decaying patterns like Figure 7b.



(a) *Top-5 probabilities*, where '*' indicates the remaining probability mass.

(b) *Full distribution* completing the top-5 probabilities from GPT-3.5, estimated using the three algorithms.

Figure 7: *Decaying patterns* of averaged probability distribution over ranks, where the probabilities are shown in log scale and evaluated on XSum.

A.1 ESTIMATION USING GEOMETRIC DISTRIBUTION

As the simplest decaying pattern, we consider exponential decay with a fixed decay factor, resulting in a Geometric distribution. Considering only the top-1 probability p_1, the whole distribution could be estimated from p_1 using

$$p(k) = p_1 \cdot (1 - p_1)^{k-1}, \text{ for } k \in [1..\infty], \tag{10}$$

where the probability decays with a factor of $\lambda = (1 - p_1)$. However, this standard Geometric distribution only considers the top-1 probability and is defined over infinite k, which is not suit for a finite vocabulary.

Consequently, we extend the distribution to multiple top probabilities and a limited range of k. We express the probabilities for ranks larger than K in near-Geometric distribution that

$$\begin{cases} p(k) = p_k, & \text{for } k \in [1..K] \\ p(k) = p_K \cdot \lambda^{k-K}, & \text{for } k \in [K+1..M] \\ \sum_{k=1}^{M} p(k) = 1, \end{cases} \tag{11}$$

where λ is a decay factor in $(0, 1)$, and M is the size of the rank list.

Using the total probability constraint, we calculate the remaining probability mass for allocating

$$\sum_{k=K+1}^{M} p(k) = 1 - \sum_{k=1}^{K} p_k = p_{\text{rest}}, \tag{12}$$

Expanding $p(k)$ in the left expression, we obtain

$$p_K \cdot \sum_{k=1}^{M-K} \lambda^k = p_{\text{rest}}, \tag{13}$$

and rewritten as

$$\sum_{k=1}^{M-K} \lambda^k = \frac{(\lambda - \lambda^{M-K+1})}{1 - \lambda} = \frac{p_{\text{rest}}}{p_K}. \tag{14}$$

Assuming λ^{M-K+1} is close to zero, we reach an approximate solution

$$\lambda \approx \frac{p_{\text{rest}}}{p_K + p_{\text{rest}}}. \tag{15}$$

In practice, we first calculate the approximate solution, naming λ_0, and then check if ${\lambda_0}^{M-K+1}$ was close to zero. If it is not, we follow an iterative process to adjust λ until it converges

$$\lambda_{t+1} = 1 - \frac{(\lambda_t - \lambda_t^{M-K+1}) \cdot p_K}{p_{\text{rest}}}. \tag{16}$$

A.2 Estimation Using Zipfian Distribution

Frequencies of words in natural languages usually adhere to Zipf's law (Zipf, 1946; 2013), where the word frequency and word rank follow a Zipfian distribution

$$p(k) \propto \frac{1}{(k+\beta)^\alpha}, \text{ for } k \in [1..\infty]. \tag{17}$$

The parameters α and β are fitted to a specific corpus, with typical values of $\alpha = 1$ and $\beta = 2.7$ for English (Piantadosi, 2014).

Assuming that the word frequencies in a given context also comply with this law, we consider it as an alternative distribution for our estimation. Given the top-K probabilities p_k, we compute the probabilities of tokens with a ranking greater than K in a Zipfian distribution

$$\begin{cases} p(k) = p_k, & \text{for } k \in [1..K] \\ p(k) = p_K \cdot (\frac{\beta}{k-K+\beta})^\alpha, & \text{for } k \in [K+1..M] \\ \sum_{k=1}^{M} p(k) = 1, \end{cases} \tag{18}$$

where α and β are two positive parameters. M is the rank-list size.

Using the total probability mass as a constraint, we determine the probability mass for allocating.

$$\sum_{k=K+1}^{M} p(k) = 1 - \sum_{k=1}^{K} p_k = p_{\text{rest}}, \tag{19}$$

After expanding $p(k)$, we get

$$p_K \cdot \sum_{k=K+1}^{M} (\frac{\beta}{k-K+\beta})^\alpha = p_{\text{rest}}, \tag{20}$$

rewritten as

$$\sum_{k=K+1}^{M} (\frac{\beta}{k-K+\beta})^\alpha = \sum_{k=1}^{M-K} (\frac{\beta}{k+\beta})^\alpha = \frac{p_{\text{rest}}}{p_K}. \tag{21}$$

The equation has two unknown parameters, thereby having multiple possible solutions. Thus, we solve the parameters by minimizing a loss function

$$\text{Loss}(\alpha, \beta) = \left(\sum_{k=1}^{M-K} (\frac{\beta}{k+\beta})^\alpha - \frac{p_{\text{rest}}}{p_K} \right)^2 + 1.0 \cdot (\alpha - 1)^2 + 0.001 \cdot (\beta - 2.7)^2. \tag{22}$$

As an additional constraint, we expect the parameters not vary from their typical values too much. We determine a coefficients of 1.0 for α and a coefficient of 0.001 for β empirically.

To accelerate the optimization process, we construct a table $T[\alpha, \beta]$, which stores the pre-calculated summation values for each pair of α and β as

$$T[\alpha, \beta] = \sum_{k=1}^{M-K} (\frac{\beta}{k+\beta})^\alpha. \tag{23}$$

We enumerate $\alpha \in (0, 10)$ with a step of 0.1 and $\beta \in (0, 20)$ with a step of 0.2, resulting in a table with 10000 values. The ranges are empirically decided, which balance the coverage of the possible choices and the size of the table. During inference, we can efficiently compute the loss table $Loss(\alpha, \beta)$ from $T[\alpha, \beta]$ given p_{rest}/p_K. We then search the loss table to identify the best α and β that lead to the smallest loss.

A.3 Estimation Using a MLP Model

The Geometric and Zipfian algorithms both work on assumptions about the distributions. An alternate approach that does not rely on these assumptions involves modeling the distribution within a neural network. We consider the simple Multi-Layer Perceptron (MLP) model with a single hidden layer, which accepts the top-K probabilities and predicts the probabilities for the rest of the ranks. The distribution is expressed as

$$\begin{cases} p(k) = p_k, & \text{for } k \in [1..K] \\ p(k) = p_{\text{rest}} \cdot p_{\text{MLP}_\theta}(k-K), & \text{for } k \in [K+1..M] \\ \sum_{k=1}^{M} p(k) = 1, \end{cases} \tag{24}$$

where $p_{rest} = 1 - \sum_{k=1}^{K} p_k$ and p_{MLP_θ} represents the MLP predictive distribution. The MLP is defined as

$$p_{\text{MLP}_\theta} = \text{SOFTMAX}(\text{MLP}_\theta(x)), \tag{25}$$

where θ denotes the model parameters. The model inputs a vector with a size of K and outputs a distribution with a size of $M - K$. The input vector x is calculated from the top probabilities using

$$x_k = \log p_k, \text{ for } k \in [1..K]. \tag{26}$$

Training. We train the MLP model on probability distributions from an open-source LLM (e.g., GPT-Neo-2.7B), using the cross-entropy loss

$$\text{Loss} = -\sum_{k=1}^{M} p_k \log p(k) = -\sum_{k=K+1}^{M} p_k \log p_{\text{MLP}_\theta}(k-K) + C, \tag{27}$$

where p_k for $k \in [1..M]$ is the target distribution and $p(k)$ is the model distribution. C is the constant part.

Inference. We predict the distributions using the top-K probabilities from the proprietary LLMs, obtaining estimated distributions on all token positions. We do not enforce the monotonic decrease constraint during inference, but it generally follows the constraint because the training target is monotonic decrease distribution.

B Experimental Settings

B.1 Evaluation Metrics

AUROC. We measure the detection accuracy mainly in the area under the receiver operating characteristic (AUROC), which gives an overview of the detectors across all possible thresholds. AUROC values can range from 0.0 to 1.0, and this value mathematically signifies the likelihood of a randomly chosen machine-generated text having a higher predicted probability of being machine-generated compared to a randomly selected human-written text. An AUROC of 0.5 is indicative of a random classifier, while an AUROC of 1.0 suggests a flawless classifier.

Accuracy (ACC). As a complement, we report the ACC for some of the experiments. ACC denotes the ratio of the number of correct predictions to the total number of input samples, which works well only if there are equal number of positive and negative samples.

True Positive Rate (TPR) and **False Positive Rate (FPR).** In depth, we compare methods in TPR vs FPR on various thresholds. A high TPR indicates that the algorithm is effective at identifying positive cases, while a high FPR indicates that the algorithm often misclassifies negative cases as positive.

B.2 AzureOpenAI Settings

We use Completion API [5] [6] of these models via AzureOpenAI platform [7], with the following deployment settings. We echo the model to return the top probabilities of the provided texts, without producing any new tokens.

Different models are supported by different regions of Azure platform. We deploy babbage-002 and davinci-002 on the region of North Central US, gpt-35-turbo-0301 on the region of East US, and gpt-35-turbo-1106 and gpt-4-1106 on the region of West US. In addition, we also tried babbage-002 from OpenAI API endpoint, obtaining the same results as the one on AzureOpenAI endpoint.

C Main Results

Table 4: Main results on *GPT-4 and Gemini-1.5* generations, with the best AUROC marked in **bold**.

Method	GPT-4				Gemini-1.5 Pro			
	XSum	Writing	PubMed	Mix3	XSum	Writing	PubMed	Mix3
GPTZero	**0.9815**	0.8838	0.8193	0.9009	-	-	-	-
Zero-Shot Detectors Using Open-Source LLMs								
Likelihood (Neo-2.7)	0.7980	0.8553	0.8104	0.7690	0.8013	0.8364	0.7064	0.7416
Entropy (Neo-2.7)	0.4360	0.3702	0.3295	0.4114	0.4170	0.2945	0.3838	0.3959
Rank (Neo-2.7)	0.6644	0.7146	0.5965	0.6448	0.6711	0.6719	0.5688	0.6260
LogRank (Neo-2.7)	0.7975	0.8286	0.8003	0.7626	0.8022	0.8102	0.7006	0.7353
DNA-GPT (Neo-2.7)	0.7347	0.8032	0.7565	0.6430	0.7996	0.8133	0.6376	0.6438
DetectGPT (T5-11B/Neo-2.7)	0.5660	0.6217	0.6805	0.6136	0.7838	0.8256	0.6222	0.7406
Fast-Detect (GPT-J/Neo-2.7)	0.9067	0.9612	0.8503	0.8999	0.8571	0.8650	0.7075	0.8072
Fast-Detect (Phi2-2.7B)	0.4636	0.6463	0.6083	0.5742	0.6454	0.6324	0.5976	0.6164
Fast-Detect (Qwen2.5-7B)	0.6476	0.8202	0.6391	0.6862	0.7082	0.7723	0.6296	0.6839
Fast-Detect (Llama3-8B)	0.6615	0.8491	0.7556	0.7269	0.7786	0.9085	0.7065	0.7552
Zero-Shot Detectors Using Proprietary LLMs								
Likelihood (GPT-3.5)	0.6468	0.9570	0.9152	0.8029	0.7130	0.9644	0.8516	0.8043
DNA-GPT (GPT-3.5)	0.7952	0.8302	0.9092	0.7748	0.8036	0.6829	0.7738	0.7107
PDE using Geometric								
Entropy (GPT-3.5)	0.6353	0.2694	0.2376	0.4074	0.6237	0.2276	0.3361	0.4144
Rank (GPT-3.5)	0.6245	0.8719	0.8283	0.7395	0.6480	0.8793	0.7768	0.7406
LogRank (GPT-3.5)	0.6319	0.9323	0.9060	0.7870	0.7102	0.9421	0.8329	0.7872
Fast-Detect (Babbage)	0.9033	0.9264	0.9195	0.8974	0.8797	0.8316	0.7887	0.8083
Fast-Detect (Davinci)	0.9141	0.9798	0.8864	0.9131	0.9062	0.9516	0.7612	0.8601
Fast-Detect (GPT-3.5)	0.9035	**0.9957**	0.9467	0.9411	0.9221	**0.9840**	**0.9112**	**0.9244**
Fast-Detect (GPT-4)	0.9673	0.9901	**0.9534**	0.9647	0.9188	0.9506	0.8477	0.8947
PDE using Zipfian								
Fast-Detect (GPT-3.5)	0.9123	0.9931	0.9429	0.9319	0.9289	0.9809	0.9070	0.9161
Fast-Detect (GPT-4)	0.9797	0.9884	0.9436	**0.9719**	**0.9303**	0.9447	0.8336	0.8991
PDE using MLP								
Fast-Detect (GPT-3.5)	0.9076	0.9930	0.9464	0.9342	0.9257	0.9810	0.9103	0.9184
Fast-Detect (GPT-4)	0.9759	0.9893	0.9496	0.9705	0.9272	0.9495	0.8453	0.9001

[5] https://platform.openai.com/docs/guides/text-generation/completions-api

[6] Completion API with gpt-35-turbo-1106 and gpt-4-1106 require an AzureOpenAI API version of '2024-02-15-preview' or later, while others require '2023-09-15-preview' or later.

[7] https://azure.microsoft.com/en-us/products/ai-services/openai-service

Table 5: Main results on *Claude-3* generations, with the best AUROC marked in **bold**.

Method	Claude-3-Sonnet				Claude-3-Opus			
	XSum	Writing	PubMed	Mix3	XSum	Writing	PubMed	Mix3
Zero-Shot Detectors Using Open-Source LLMs								
Likelihood (Neo-2.7)	0.8862	0.9484	0.8360	0.8661	0.9322	0.9734	0.8603	0.9030
Entropy (Neo-2.7)	0.4146	0.2156	0.2989	0.3466	0.3871	0.1792	0.2910	0.3265
Rank (Neo-2.7)	0.7019	0.7812	0.6017	0.6888	0.7333	0.7950	0.6080	0.7056
LogRank (Neo-2.7)	0.8867	0.9401	0.8296	0.8654	0.9357	0.9679	0.8508	0.9042
DNA-GPT (Neo-2.7)	0.8558	0.9415	0.7647	0.7080	0.9424	0.9653	0.7806	0.7326
DetectGPT (T5-11B/Neo-2.7)	0.8150	0.8675	0.7347	0.7967	0.7718	0.8335	0.7752	0.7776
Fast-Detect (GPT-J/Neo-2.7)	0.9514	0.9763	0.8634	0.9260	0.9779	0.9832	0.8947	0.9468
Fast-Detect (Phi2-2.7B)	0.7536	0.6773	0.7144	0.6957	0.8080	0.7545	0.7322	0.7450
Fast-Detect (Qwen2.5-7B)	0.8595	0.8600	0.7346	0.7813	0.9097	0.8967	0.7572	0.8119
Fast-Detect (Llama3-8B)	0.9243	0.9198	0.7936	0.8212	0.9640	0.9377	0.8251	0.8510
Zero-Shot Detectors Using Proprietary LLMs								
Likelihood (GPT-3.5)	0.8364	0.9918	0.9299	0.9023	0.9191	0.9955	0.9467	0.9295
DNA-GPT (GPT-3.5)	0.7934	0.8587	0.8988	0.7871	0.9040	0.9362	0.8926	0.8383
PDE using Geometric								
Entropy (GPT-3.5)	0.4685	0.0565	0.1985	0.2582	0.4310	0.0381	0.1852	0.2339
Rank (GPT-3.5)	0.7801	0.9724	0.8476	0.8473	0.8263	0.9809	0.8524	0.8645
LogRank (GPT-3.5)	0.8502	0.9927	0.9265	0.9062	0.9302	**0.9966**	0.9414	0.9336
Fast-Detect (Babbage)	0.9508	0.9705	0.9111	0.9438	0.9874	0.9865	0.9298	0.9698
Fast-Detect (Davinci)	**0.9659**	**0.9939**	0.9084	0.9606	0.9940	0.9946	0.9262	0.9742
Fast-Detect (GPT-3.5)	0.9433	0.9930	**0.9552**	0.9576	0.9899	0.9829	**0.9686**	0.9689
Fast-Detect (GPT-4)	0.9523	0.9910	0.9424	0.9623	0.9930	0.9917	0.9580	**0.9817**
PDE using Zipfian								
Fast-Detect (GPT-3.5)	0.9455	0.9920	0.9510	0.9475	0.9914	0.9798	0.9627	0.9588
Fast-Detect (GPT-4)	0.9581	0.9889	0.9308	0.9613	**0.9958**	0.9901	0.9496	0.9792
PDE using MLP								
Fast-Detect (GPT-3.5)	0.9457	0.9925	0.9548	0.9526	0.9911	0.9811	0.9664	0.9634
Fast-Detect (GPT-4)	0.9574	0.9901	0.9382	**0.9631**	0.9953	0.9909	0.9543	0.9807

Table 6: *A comparison of ACC* in PDE and the major baselines on generations from *ChatGPT*, where we employ the optimal threshold either for each dataset or across all three datasets. The smaller average drop scales on PDE methods indicate that PDE offers a more consistent metric across datasets. In addition, we also assess optimal threshold across datasets and source models, which yields ACCs nearly identical (deviation less than 0.006 except Likelihood and DNA-GPT) to the optimal threshold across datasets.

Method	Best Threshold per Dataset				Best Threshold across Datasets				Drop
	XSum	Writing	PubMed	Avg.	XSum	Writing	PubMed	Avg.	Avg.
Fast-Detect (GPT-J/Neo-2.7)	0.9600	0.9633	0.8267	0.9167	0.9400	0.9367	0.7567	0.8778	-0.0389
Fast-Detect (Phi2-2.7B)	0.7467	0.6967	0.7600	0.7344	0.6767	0.6967	0.7467	0.7067	-0.0277
Fast-Detect (Qwen2.5-7B)	0.7367	0.7600	0.7267	0.7411	0.6433	0.7600	0.6867	0.6967	-0.0444
Fast-Detect (Llama3-8B)	0.8000	0.7700	0.7267	0.7656	0.6900	0.7667	0.6567	0.7044	-0.0612
Likelihood (GPT-3.5)	0.8767	0.9933	0.8933	0.9211	0.7533	0.9900	0.8633	0.8689	-0.0522
DNA-GPT (GPT-3.5)	0.8750	0.8592	0.8833	0.8725	0.8108	0.5986	0.7467	0.7187	-0.1538
PDE using Geometric									
Fast-Detect (Babbage)	0.9600	0.9433	0.9033	0.9356	**0.9600**	0.9100	0.8833	0.9178	-0.0178
Fast-Detect (Davinci)	**0.9667**	0.9800	0.8867	0.9444	0.9333	0.9700	0.8633	0.9222	-0.0222
Fast-Detect (GPT-3.5)	0.9633	**0.9967**	**0.9200**	**0.9600**	0.9033	**0.9967**	**0.9167**	**0.9389**	-0.0211
Fast-Detect (GPT-4)	0.9367	0.9733	0.8933	0.9344	0.9367	0.9400	0.8833	0.9200	**-0.0144**

D Ablation Study

D.1 Necessity of PDE

Readers may wonder the necessity of these estimation algorithms, given that the top-K probabilities provide the major information. To testify it, we consider a Naive approach to estimate the full distribution, where we assign zero probability to ranks larger than K. Using the Naive distribution, Fast-Detect (GPT-3.5) downgrades from 0.9630 (Geometric) to 0.9311 (Naive), indicating the necessity of a proper estimation algorithm.

Table 7: The prompts that we test for the ablation, from the empty prompt0 to simple prompt1 until complex prompt3 and prompt4. The changes are marked in *italic*.

Prompt	Content
prompt0	(Empty)
prompt1	*You serve as a valuable aide, capable of generating clear and persuasive pieces of writing given a certain context. Now, assume the role of an author and strive to finalize this article.*
prompt2	You serve as a valuable aide, capable of generating clear and persuasive pieces of writing given a certain context. Now, assume the role of an author and strive to finalize this article. *I operate as an entity utilizing GPT as the foundational large language model. I function in the capacity of a writer, authoring articles on a daily basis. Presented below is an example of an article I have crafted.*
prompt3	*System:* You serve as a valuable aide, capable of generating clear and persuasive pieces of writing given a certain context. Now, assume the role of an author and strive to finalize this article. *Assistant:* I operate as an entity utilizing GPT as the foundational large language model. I function in the capacity of a writer, authoring articles on a daily basis. Presented below is an example of an article I have crafted.
prompt4	*Assistant:* You serve as a valuable aide, capable of generating clear and persuasive pieces of writing given a certain context. Now, assume the role of an author and strive to finalize this article. *User:* I operate as an entity utilizing GPT as the foundational large language model. I function in the capacity of a writer, authoring articles on a daily basis. Presented below is an example of an article I have crafted.

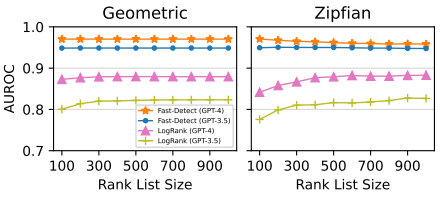


Figure 8: Ablation on *rank-list size*, where the AUROC is averaged across the three datasets produced by GPT-4.

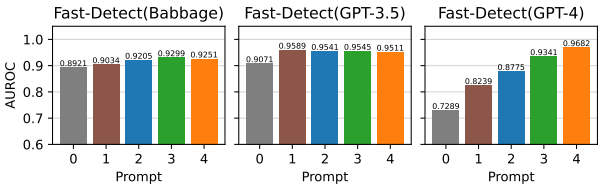


Figure 9: Ablation on *prompt*, where the AUROC is averaged across the three datasets produced by GPT-4. GPT-4 is most sensitive to prompts.

D.2 Ablation on Rank-List Size

The size M of the rank list is another important hyper-parameter. We assess its effects on Geometric and Zipfian distributions (skipping MLP because it requires a heavy training process for each setting). As demonstrated in Figure 8, overall PDE with a larger size obtains a higher accuracy. Geometric distribution shows a monotonic increasing trends, while Zipfian shows decreasing trends for Fast-Detect but increasing trends for LogRank, demonstrating an inconsistent pattern. Roughly, experiments with MLP on the sizes of 100 and 1000 suggest that it has the similar pattern as Zipfian.

D.3 Ablation on Prompt

Large models are sensitive to text context, necessitating a suitable prompt for optimal detection accuracy (Taguchi et al., 2024). We analyze the prompts featured in Table 7 in Appendix to ascertain their effect. We draft the prompts manually, starting with prompt3. Then we replace the '*System*' and '*Assistant*' roles with '*Assistant*' and '*User*' to produce prompt4 and we remove the roles to produce prompt2. We simplify prompt2 by removing its second paragraph to produce prompt1 and removing all content to produce the empty prompt0. In this study, we only experiment with several manually-drafted prompts, leaving a systematic exploration of the prompts for future.

As Figure 9 shows, GPT-4 is the most sensitive model among the four scoring models, with detection accuracy fluctuating between 0.7289 (prompt0) and 0.9682 (prompt4). In contrast, GPT-3.5 is less

sensitive, with a detection accuracy increasing from 0.9071 (prompt0) to 0.9589 (prompt1), but maintaining stability for the rest. The base model Babbage and Davinci are less influenced by the prompt, and we do not show them in the figure.

E Analysis and Discussion

E.1 Robustness across Source Models and Domains

Table 8: Robustness across *source models* measured in ACC, where we evaluate all source models using a threshold determined according to ChatGPT. All experiments are run on *Mix3*.

Method	ChatGPT	GPT-4	Claude-3		Gemini-1.5	All
			Sonnet	Opus	Pro	Avg.
Fast-Detect (GPT-J/Neo-2.7)	0.8778	0.7933	0.8389	0.8822	0.6867	0.8158
Fast-Detect (Phi2-2.7B)	0.7067	0.5500	0.6378	0.6622	0.6011	0.6316
Fast-Detect (Qwen2.5-7B)	0.6967	0.6222	0.6811	0.7011	0.6256	0.6653
Fast-Detect (Llama3-8B)	0.7044	0.6556	0.7244	0.7467	0.6844	0.7031
Likelihood (GPT-3.5)	0.8689	0.6806	0.8326	0.8605	0.6703	0.7826
DNA-GPT (GPT-3.5)	0.7187	0.6176	0.6942	0.7540	0.6372	0.6843
PDE using Geometric						
Fast-Detect (Babbage)	0.9178	0.7373	0.8461	0.9176	0.6562	0.8150
Fast-Detect (Davinci)	0.9222	0.8085	0.8973	0.9131	0.7363	0.8555
Fast-Detect (GPT-3.5)	**0.9389**	0.8650	0.8985	0.9197	**0.8554**	**0.8955**
Fast-Detect (GPT-4)	0.9200	**0.9043**	**0.9041**	**0.9276**	0.7973	0.8906

Table 9: Robustness across *domains* measured in ACC, where we cross-validate each dataset using a threshold determined according to other two datasets for each source model.

Method	ChatGPT				GPT-4			
	XSum	Writing	PubMed	Avg.	XSum	Writing	PubMed	Avg.
Fast-Detect (GPT-J/Neo-2.7)	0.8633	0.9300	0.7000	0.8311	0.7867	0.8833	0.6600	0.7767
Fast-Detect (Phi2-2.7B)	0.6800	0.6967	0.7467	0.7078	0.4767	0.6267	0.5667	0.5567
Fast-Detect (Qwen2.5-7B)	0.6200	0.7300	0.6167	0.6556	0.5733	0.7433	0.5500	0.6222
Fast-Detect (Llama3-8B)	0.6567	0.7500	0.5767	0.6611	0.5967	0.6933	0.5800	0.6233
Likelihood (GPT-3.5)	0.6933	0.9833	0.8233	0.8333	0.5503	0.8020	0.8300	0.7274
DNA-GPT (GPT-3.5)	0.5000	0.5106	0.7067	0.5724	0.4966	0.5000	0.6933	0.5633
PDE using Geometric								
Fast-Detect (Babbage)	0.9133	0.9100	**0.8800**	0.9011	0.8154	0.8033	0.8333	0.8174
Fast-Detect (Davinci)	0.8767	0.9633	0.8500	0.8967	0.8087	0.7800	0.8033	0.7974
Fast-Detect (GPT-3.5)	0.8833	**0.9967**	0.8267	**0.9022**	0.7819	**0.9396**	**0.8667**	0.8627
Fast-Detect (GPT-4)	**0.9367**	0.9367	0.8167	0.8967	**0.9200**	0.9195	0.8467	**0.8954**

In practice, we need to fix the decision threshold and detect text from various sources and domains. However, distributions of detection metric might be shifted between different source models or domains, resulting in high detection accuracy in one but low accuracy in another. In this section, we evaluate the robustness of PDE along with other strong baselines across different source models and domains.

Firstly, we examine the detection accuracy (in ACC) for each source model utilizing an optimal threshold identified on ChatGPT Mix3 dataset. As illustrated in Table 8, PDE consistently provides the highest ACCs across all source models. While the accuracy of Fast-Detect (GPT-3.5) and Fast-Detect (GPT-4) fluctuates between source models, their overall ACCs are closely matched. Fast-Detect (GPT-3.5) delivers the greatest ACC, which is approximately 8 points above the top baseline. This demonstrates the stability of PDE across numerous source models.

Subsequently, we assess the accuracy on each dataset using an optimal threshold established on the remaining two datasets for each source model. As exhibited in Table 9, we employ the source models of ChatGPT and GPT-4 as examples. PDE also delivers the highest ACCs on all datasets, further evidence of its robustness across various domains.

Table 10: Robustness under *paraphrasing attack* with diverse lexicons and orders, where we report the TPR (%) at an FPR level of 1%.

Method	ChatGPT + DIPPER (60 L)				ChatGPT + DIPPER (60 O)			
	XSum	Writing	PubMed	Avg.	XSum	Writing	PubMed	Avg.
Fast-Detect (GPT-J/Neo-2.7)	48.7	**65.3**	38.0	50.7	52.7	66.0	34.0	50.9
Fast-Detect (Phi2-2.7B)	6.7	15.3	8.0	10.0	0.7	0.7	5.3	1.8
Fast-Detect (Qwen2.5-7B)	16.0	35.3	8.0	19.8	5.3	10.7	6.0	7.3
Fast-Detect (Llama3-8B)	34.0	50.0	13.3	32.4	17.3	12.7	10.7	13.6
Likelihood (GPT-3.5)	0.7	11.3	9.3	6.9	0.7	71.3	18.7	30.2
PDE using Geometric								
Fast-Detect (Babbage)	**64.0**	58.7	**39.3**	**54.0**	**83.3**	80.0	**42.0**	**68.4**
Fast-Detect (Davinci)	16.7	48.0	28.7	31.1	51.3	88.7	33.3	57.8
Fast-Detect (GPT-3.5)	0.7	62.0	12.0	24.9	18.0	**95.3**	22.7	45.3

E.2 Robustness under Paraphrasing Attack

We assess the performance of PDE uder paraphrasing attack, utilizing DIPPER (Krishna et al., 2024) to rephrase the output generated by ChatGPT. Our testing encompasses two paraphrasing settings: high lexical diversity (60 L) and high order diversity (60 O). As indicated in Table 10, Fast-Detect (Babbage) surpasses Fast-Detect (Neo-2.7) in both settings, but is more significant influenced by diverse lexicons than by diverse orderings.

However, we also note unusual behavior of DIPPER on XSum, where the diverse lexical paraphrasing results in a surprisingly low detection accuracy for Likelihood (GPT-3.5) baseline. This is caused by the atypical trend that the paraphraser replaces common words with rare expressions, thus reducing the readability of the content. This odd distribution could be the cause of the unusually low accuracy of Fast-Detect (GPT-3.5) on XSum (60 L). PDE outperforms Fast-Detect (Neo-2.7) and the trained detectors, proving its efficacy in withstanding a paraphrasing attack.

Additionally, we find that larger LLMs are more susceptible to increased lexical and order diversity. Nonetheless, our review of the paraphrased articles indicates that this increased diversity also lowers the readability due to excessive use of unusual words and sequences, implying that there are some drawbacks of the attack.