CS772 Project Report: Improved Detection of Machine-Generated Text using Bayesian Surrogates

Omkar Chavan (210280)
Department of Mechanical Engineering
Indian Institute of Technology, Kanpur
chavan21@iitk.ac.in

Saranya Pal (210930)
Department of Chemistry
Indian Institute of Technology, Kanpur
saranyap21@iitk.ac.in

Vaibhav Shaily (211139)
Department of Aerospace Engineering
Indian Institute of Technology, Kanpur
vaibhavs21@iitk.ac.in

Aditya Nikam (218070066)
Department of Mechanical Engineering
Indian Institute of Technology, Kanpur
adityarn21@iitk.ac.in

Abstract

As part of a course project in Probabilistic Machine Learning, we addressed the problem of distinguishing machine-generated texts from human-written ones. To this end, we implemented the methodology outlined in a provided seed paper, for which no official implementation was available. Building upon the original approach, we introduced improvements to the Perturbation Generator and modified the kernel function used in the Gaussian Process model. In total, we implemented five algorithms: two based on the original paper and three variants proposed by us.

1 Problem Overview

With the rise of large language models (**LLMs**), it is important to detect such machine generated texts for preventing their misuse which may result in some serious social problems. In the past efforts have been made to train detectors on specific dataset, but fail to generalize on unseen dataset. Recently, Bayesian surrogate models have been used for selecting samples based on uncertainty and then interpolate scores from samples to other samples. Our project is based on the paper "Efficient Detection of LLM-generated Texts with a Bayesian Surrogate Model." It focuses on detecting and classifying LLM-generated text from human-written text, a challenge given LLMs' ability to produce highly realistic text. In this paper, we implement the above mentioned method and also explore other variation of it. Specifically, we introduce a diffusion-based perturbation mechanism to generate more informative candidate texts and modify the kernel function in the Bayesian surrogate model to better capture the underlying score landscape.

2 Introduction

Large language models (LLMs) exhibit a remarkable ability to mimic human language, generating text that is coherent, articulate, and persuasive. However, despite their fluency, the content they produce may include factual inaccuracies and fabricated references. As LLMs become increasingly popular tools for simplifying writing and communication tasks, their misuse has also grown, ranging from the creation of misleading fake news to academic dishonesty. These harmful applications pose serious societal risks, making it essential for the research community to address the challenges associated with responsible LLM use.

Some approaches rely on training supervised classifiers, but these often struggle with overfitting and fail to generalize effectively to unseen data. In contrast, zero-shot methods for detecting LLM-generated text avoid these limitations by utilizing the source LLM itself to evaluate its outputs

[1, 2]. These methods typically assess the average per-token log probability of a given text, yet their real-world detection performance often remains inadequate.

Zero-shot methods for detecting LLM-generated text avoid the limitations of supervised classifiers by leveraging the source language model itself to evaluate its outputs [5, 1]. These techniques typically assess the average per-token log probability of a given text. However, their detection performance is often inadequate in practice. In this paper we aim to implement and also explore some other variations of the method given in [3] which uses Bayesian surrogate model to improve query efficiency. The key idea is to replace the large number of queries with the help of Gaussian Process (**GP**) as a surrogate model. Here a small set of typical samples are selected to query the LLM, then bayesian uncertainty is used to decide which samples are most informative . Based on the updated GP model, the scores are evaluated for the rest of the purturbations.

3 Related Work

Previous approaches to detecting machine-generated text based on zero-shot detection [1, 4, 5], which require access to the source model to extract features such as output logits or loss values. For example, [5] proposed that higher per-token log probabilities are indicative of AI-generated content. When direct access to the source model is not available, these methods often resort to using proxy models. However, significant performance gaps may arise due to discrepancies between the proxy and source models.

DetectGPT [4] introduces a more effective zero-shot approach by analyzing the probability curvature of the language model. It perturbs the candidate text multiple times and uses the source LLM to score these variants, enabling the calculation of detection metrics. It is based on the hypothesis that the machine generated texts tend to lie in regions of high probability curvature in the models output distribution and in contrast human written texts do not lie at such peaks. This method works with any LLM as long as it can output log probabilities. This method also requires creating multiple purturbations of a candidate text. While it performs well on some models, but DetectGPT requires hundreds of LLM queries per sample, making it computationally expensive. This hinders its scalability to more advanced models such as LLaMA, ChatGPT and GPT-4. Some work has been done to improve query efficiency of DetectGPT. One of which is [3] which states that the inefficiency in DetectGPT arises from the use of random perturbations for log probabiliy curvature estimation. It uses uncertainty from GP model to identify most useful perturbations and also interpolate the log probabilities for other samples.

4 Methodology

Our method is built on the DetectGPT framework, using a diffusion-based perturbation generator and a modified kernel for the surrogate model for capturing similarities between candidate texts. A **GP** model is used as a surrogate model, which has two tasks: (i) fitting a function for log probability (ii) identifying most important perturbations for quering LLM.

4.1 Baseline: DetectGPT with Vanilla Perturbation

We took the DetectGPT approach as our baseline method, which detects machine-generated text by analyzing how the log-probability of a text changes under small perturbations. It utilizes the following measure to determine if a text passage x is generated form a LLM p_θ.

$$\log p_\theta(x) - \mathbb{E}_{\tilde{x}\sim q(\cdot|x)}[\log p_\theta(\tilde{x})] \tag{1}$$

where $q(\cdot|x)$ is a purturbation distruibutionsupported on semantic neighbourhood of a candidate text x. Here $q(\cdot|x)$ can be defined with manual rephrasing while maintaining semantic similarity. To avoid human intervention, models like T5 [**?**] can be used for rephrasing.

- **Perturbation Generation:** For a given input text, a paraphrasing model (like T5) generates multiple paraphrased versions. We use the T5ForConditionalGeneration model. Given an input text x, it generates a set of N semantically similar perturbations.

$$\mathcal{P}(x) = \{x_1, x_2, \ldots, x_N\} \tag{2}$$

  where each x_i is a paraphrase of x.

- **Log-Probability Calculation:** The log-probabilities of each token of both the original and perturbed texts are hen computed using a source language model (e.g., GPT-2). We normalized these probablities on token counts to account for length differences. For a sequence $x = (x_1, x_2, \ldots, x_n)$, the normalized log-probability is given by:

$$\log p_\theta(x) = \frac{1}{n-1} \sum_{t=1}^{n-1} \log p_\theta(x_{t+1}|x_{1:t}) \tag{3}$$

- **Detection Score:** We defined the detection score as the difference between the log-probability of the original text and the mean log-probability of its perturbations. Higher values mean a higher likelihood of text being machine-generated.

Algorithm 1 DetectGPT

```
1: **Input:** Text passage $x$, LLM $p_\theta$, perturbation model $q(\cdot|x)$, kernel $k(x, x')$, hyperparameters
   $\alpha, \beta, \sigma$ sample sizes $N, T, S$, detection threshold $\delta$
2: **Output:** True or False indicating whether $x$ is generated by $p_\theta$
3: Generate perturbations $X = \{x_i\}_{i=0}^{N}$ using $q(\cdot|x)$
4: Randomly initialize typical set $X_t$ and selection set $X^*$
5: $y_t \leftarrow \log p_\theta(X_t)$
6: Append selected sample and score to $X_t$ and $y_t$ respectively
7: Estimate detection measure $\ell(x, p_\theta, q)$ using the final GP model
8: **return** True if $\ell(x, p_\theta, q) > \delta$ else False
```

4.2 Bayesian Surrogate Model with Modified Kernel

To interpolate the log-probability distribution over the space of perturbations, we use a Bayesian surrogate model:

- **Gaussian Process Regression:** A Gaussian Process (GP) surrogate is trained on a selected set of perturbations, whose log-probabilities are found. The GP models the relationship between perturbations and their log-probabilities for uncertainty estimation and active sampling. The GP is defined as:

$$f \sim \mathcal{GP}(0, k(\cdot, \cdot)) \tag{4}$$

  where k is a custom kernel based on BERTScore similarity:

$$k(x_i, x_j) = \alpha \cdot S_{ij} + \beta \tag{5}$$

  Here, α and β are trainable hyperparameters.

  Given observed log-probabilities $\mathbf{y}$ at samples $\mathbf{X}$, the GP posterior for new points $\mathbf{X}^*$ is:

$$p(f^*|\mathbf{X}^*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(K_{X^*X}[K_{XX} + \sigma^2 I]^{-1}\mathbf{y}, K_{X^*X^*} - (K_{X^*X}[K_{XX} + \sigma^2 I]^{-1}K_{XX*}) \tag{6}$$

- **Similarity-Based Kernel:** Instead of a standard kernel, we implement a custom kernel based on BERTScore similarity between perturbations. To capture the relationships between candidate texts, we compute a semantic similarity matrix S using BERTScore. Each entry S_{ij} represents the semantic similarity between texts x_i and x_j:

$$S_{ij} = BERTScore(x_i, x_j) \tag{7}$$

- **Active Sampling:** The uncertainty estimated by GP help us select new perturbations to query. At each iteration, the perturbation with the highest predictive uncertainty is selected, and its log-probability is computed. At each iteration, we select the candidate with the highest predictive variance:

$$x_{t+1} = \arg \max_{x \in \mathcal{P}(x)} \sigma_t(x) \tag{8}$$

  where $\sigma_t(x)$ is the predictive standard deviation from the GP at iteration t. This process continues until a threshold ϵ is reached.

- **Hyperparameter tuning:** The hyperparameters α and β are tunned by optimizing log marginal likelihood of targets y.

$$\log p(\mathbf{y} \mid X, \alpha, \beta, \sigma^2) \propto -\left[\mathbf{y}^\top (K_{XX} + \sigma^2 I)^{-1}\mathbf{y} + \log \left|K_{XX} + \sigma^2 I\right|\right] \tag{9}$$

4.2.1 End-to-End Detection Pipeline

The complete detection process is as follows:

1. **Input:** Given a candidate text, generate N perturbations using either vanilla T5-based paraphrasing or the diffusion-based refinement process.
2. **Log-Probability Queries:** Compute per-token normalized log-probabilities for the original and new perturbations using the source language model (e.g., GPT-2).
3. **Similarity Matrix:** Calculate the BERTScore similarity matrix among all texts (original + perturbations).
4. **GP Surrogate Training:** Initialize the GP with the log-probabilities of the original text and one perturbation. Iteratively expand the training set by selecting the most uncertain perturbation, querying its log-probability, and updating the GP.
5. **Score Calculation:** After the uncertainty falls below the threshold, the GP predicts log-probabilities for all candidates. The detection score is computed as follows:

$$Score(x) = \frac{\mu_x - \frac{1}{N}\sum_{i=1}^{N}\mu_{x_i}}{\sigma_{x_i}} \tag{10}$$

   Where:

   - μ_x = Original text's predicted log-probability
   - μ_{x_i} = Perturbed samples' log-probabilities
   - σ_{x_i} = Standard deviation of perturbed log-probabilities
   - N = Number of perturbations

6. **Interpretation:** A higher normalized score indicates a higher likelihood that the input text is machine-generated.

Algorithm 2

```
1: Input: Text passage $x$, LLM $p_\theta$, perturbation model $q(\cdot|x)$, kernel $k(x, x')$, hyperparameters
   $\alpha, \beta$, sample sizes $N, T, S$, detection threshold $\delta$
2: Output: True or False indicating whether $x$ is generated by $p_\theta$
3: Generate perturbations $X = \{x_i\}_{i=0}^{N}$ using $q(\cdot|x)$
4: Randomly initialize typical set $X_t$ and selection set $X^*$
5: $y_t \leftarrow \log p_\theta(X_t)$
6: while $|X_t| < T$ or early stop criteria not met do
7:     Optimize $\alpha, \beta$ using Eq. (9) with $X_t$ and $y_t$
8:     Estimate predictive covariance $\Sigma^*$ for $X^*$ using Eq. (6)
9:     Identify sample in $X^*$ with highest uncertainty (largest diagonal of $\Sigma^*$)
10:    Score selected sample with LLM $p_\theta$
11:    Append selected sample and score to $X_t$ and $y_t$ respectively
12: end while
13: Estimate detection measure $\ell(x, p_\theta, q)$ using the final GP model
14: return True if $\ell(x, p_\theta, q) > \delta$ else False
```

4.3 Summary Table

Step	Description
Perturbation Generation	T5-based paraphrasing (vanilla) or diffusion-guided refinement (proposed)
Log-Probability Calculation	Per-token normalized log-probability via source model (GPT-2)
Similarity Matrix	BERTScore-based precomputed similarity between all candidates
Surrogate Model	Gaussian Process with custom BERTScore kernel
Active Sampling	Query perturbations with highest model uncertainty
Detection Score	Normalized difference between original and mean perturbed log-probabilities

Table 1: Overview of the detection pipeline steps.

This methodology enables robust, query-efficient detection of machine-generated text, leveraging both advanced ies and uncertainty-aware Bayesian modeling for improved generalization and performance under limited query budgets.

4.4 Proposed Enhancements

4.4.1 Diffusion-Based Perturbation

We introduced a diffusion-inspired refinement process to generate more informative perturbations. This framework presents a Bayesian uncertainty-based text detector that leverages language model log-probabilities, paraphrastic perturbations, and Gaussian Process Regression (GPR) with a BERTScore-based kernel. Perturbations are refined via a diffusion-inspired process involving paraphrasing followed by a reconstruction step. The framework adaptively samples the most informative perturbations using the variance estimated by the Gaussian Process to detect shifts in model confidence. An important point to note is that it is NOT a diffusion model that we are using. Instead, the process of noising, followed by de-noising is inspired from the diffusion process:

- **Diffusion Refiner:** Each perturbation undergoes a two-step process:
  1. *Noising:* The text is paraphrased using a paraphraser model (e.g., a ChatGPT-paraphraser fine-tuned on T5), introducing controlled "noise".
  2. *Denoising/Refinement:* The noised text is then passed through a text-to-text generation model (For our case, we use google/flan-t5-small) to recreate a refined version, simulating a diffusion process.
- **Batch Processing:** We apply this refinement to a set of texts to produce texts that are similar yet structurally different and the results are perturbations with improved qualt

Algorithm 3

```
1: **Input:** Text passage $x$, LLM $p_\theta$, perturbation model $q(\cdot|x)$, kernel $k(x, x')$, hyperparameters $\alpha, \beta$, total samples $N$, max queries $T$, uncertainty threshold $\delta$
2: **Output:** True or False indicating whether $x$ is generated by $p_\theta$
3: Generate perturbations $X = \{x_i\}_{i=1}^{N-1}$ using $q(\cdot|x)$
4: Optionally refine $X$ using diffusion: $x_i \leftarrow Refiner(Paraphraser(x_i))$
5: Prepend $x_0 = x$ to $X$
6: Compute similarity matrix $S$ between all $x_i$ using BERTScore
7: Initialize training indices $I_t \leftarrow \{0, 1\}$ and scores $y_t \leftarrow \log p_\theta(\{x_0, x_1\})$
8: **while** $|I_t| < T$ and $\max Var_{GP}(X^*) > \delta$ **do**
9:     Fit GP on $(I_t, y_t)$ with kernel $k(x_i, x_j) = \alpha S_{ij} + \beta$
10:    Compute predictive variance over $X^* = \{x_j \notin I_t\}$
11:    Select $x^* = \arg\max_{x_j \in X^*} Var_{GP}(x_j)$
12:    Query $p_\theta$ to get $y^* = \log p_\theta(x^*)$
13:    Append index and score: $I_t \leftarrow I_t \cup \{x^*\}$, $y_t \leftarrow y_t \cup \{y^*\}$
14: **end while**
15: Fit final GP and obtain posterior means $\mu$
16: Compute detection score: $\ell(x, p_\theta, q) = \frac{\mu_0 - \bar{\mu}_{1:N-1}}{\max(std(\mu_{1:N-1}), \varepsilon)}$
17: **return** True if $\ell(x, p_\theta, q) > \delta$ else False
```

So, it is evident that all the remaining parts are kept exactly the same as per the implementation of the paper. Only the Perturbation Generator is improved using a Diffusion-guided process.

4.4.2 Hybrid BERTScore and ARD Kernel

In the Bayesian detector, the kernel function plays a central role in modeling the similarity between perturbed versions of the original text. Rather than relying purely on surface-level features like token overlap, we blend semantic similarity (via BERTScore) and distributional similarity (via RBF on BERT embeddings) into a single, learnable kernel. This hybrid approach enables more expressive uncertainty modeling over text samples.

Composite Kernel Description

The kernel function $k(x, x')$ used in our Bayesian detector is a combination of two components: a similarity measure based on BERTScore and a radial basis function (RBF) kernel operating on BERT embeddings. Formally, the composite kernel is defined as:

$$k(x, x') = \alpha \cdot k_{BERTScore}(x, x') + \beta \cdot k_{ARD-RBF}(\phi(x), \phi(x')) \tag{11}$$

where:

- $\alpha, \beta > 0$ are scalar hyperparameters (learned during training via marginal likelihood maximization).
- $k_{BERTScore}(x, x')$ is a precomputed similarity value between texts x and x' using the BERTScore metric.
- $\phi(x) \in \mathbb{R}^d$ denotes the BERT embedding of the text x, obtained by mean pooling the final hidden states.

The second component, $k_{ARD-RBF}$, is an automatic relevance determination (ARD) variant of the standard RBF kernel and is defined as:

$$k_{ARD-RBF}(\phi(x), \phi(x')) = \exp\left(-\frac{1}{2}\sum_{j=1}^{d}\frac{(\phi_j(x) - \phi_j(x'))^2}{\ell_j^2}\right) \tag{12}$$

Here, ℓ_j is a separate lengthscale parameter for each dimension j of the BERT embedding, allowing the kernel to adaptively weigh each latent feature's contribution.

This composite kernel enables the Gaussian Process surrogate model to capture both high-level semantic similarity and finer-grained geometric relationships between perturbed text samples.

4.4.3 Merging both the Improvements

Improved Kernel is used....

5 Experimental Setup

5.1 Datasets

We used the **LLM-Human Classification Data** dataset, which contains 29,145 samples labeled as either LLM-generated or human written. For computational efficiency, we sampled 581 LLM-generated and 525 human written texts, resulting in a balanced evaluation set of 1,106 samples.

5.2 Evaluation Metrics

The main evaluation metric that we used is **accuracy**, which was computed after thresholding the detection score, to classify every text as LLM-generated or human-written. The optimal threshold and direction (> or <) were selected to maximize validation accuracy. Confusion matrices were also used for more analysis.

Algorithm 4

```
1: Input: Text passage $x$, LLM $p_\theta$, perturbation model $q(\cdot|x)$, kernel $k(x, x')$, hyperparameters $\alpha, \beta$, sample sizes $N, T, S$, detection threshold $\delta$.
2: Output: True or False indicating whether $x$ is generated by $p_\theta$
3: Generate perturbations $X = \{x_i\}_{i=0}^N$ using $q(\cdot|x)$
4: Randomly initialize typical set $X_t$ and selection set $X^*$
5: $y_t \leftarrow \log p_\theta(X_t)$
6: Define composite kernel:
       $$k(x, x') = \alpha \cdot BERTScore(x, x') + \beta \cdot RBF(\phi(x), \phi(x'))$$
7: where $\phi(x)$ is the BERT embedding of $x$, and RBF is an ARD kernel with lengthscale per dimension
8: while $|X_t| < T$ or early stop criteria not met do
9:     Optimize $\alpha, \beta$ using Eq. (9) with $X_t$ and $y_t$
10:    Estimate predictive covariance $\Sigma^*$ for $X^*$ using Eq. (6)
11:    Identify sample in $X^*$ with highest uncertainty (largest diagonal of $\Sigma^*$)
12:    Score selected sample with LLM $p_\theta$
13:    Append selected sample and score to $X_t$ and $y_t$ respectively
14: end while
15: Estimate detection measure $\ell(x, p_\theta, q)$ using the final GP model
16: return True if $\ell(x, p_\theta, q) > \delta$ else False
```

Algorithm 5 Bayesian Detector with Diffusion Perturbation and Composite Kernel

```
1: Input: Text passage $x$, LLM $p_\theta$, perturbation model $q(\cdot|x)$, hyperparameters $\alpha, \beta$, total samples $N$, max queries $T$, uncertainty threshold $\delta$
2: Output: True or False indicating whether $x$ is generated by $p_\theta$
3: Generate perturbations $X = \{x_i\}_{i=1}^{N-1}$ using $q(\cdot|x)$
4: Refine each $x_i$ with diffusion: $x_i \leftarrow Refiner(Paraphraser(x_i))$
5: Prepend $x_0 = x$ to $X$
6: Compute BERTScore similarity matrix $S$ between all $x_i$
7: Compute BERT embeddings $\phi(x_i)$ for all $x_i$
8: Define composite kernel:
       $$k(x_i, x_j) = \alpha \cdot S_{ij} + \beta \cdot \exp\left(-\frac{1}{2}(\phi(x_i) - \phi(x_j))^\top \Lambda^{-1}(\phi(x_i) - \phi(x_j))\right)$$
9: Initialize training indices $I_t \leftarrow \{0, 1\}$ and scores $y_t \leftarrow \log p_\theta(\{x_0, x_1\})$
10: while $|I_t| < T$ and $\max Var_{GP}(X^*) > \delta$ do
11:     Fit GP on $(I_t, y_t)$ using kernel $k$
12:     Compute predictive variance over $X^* = \{x_j \notin I_t\}$
13:     Select $x^* = \arg\max_{x_j \in X^*} Var_{GP}(x_j)$
14:     Query $p_\theta$ to get $y^* = \log p_\theta(x^*)$
15:     Update $I_t \leftarrow I_t \cup \{x^{\}}$ and $y_t \leftarrow y_t \cup \{y^{\}}$
16: end while
17: Fit final GP and obtain posterior means $\mu$
18: Compute detection score: $\ell(x, p_\theta, q) = \frac{\mu_0 - \bar{\mu}1:N-1}{\max(std(\mu1:N-1), \varepsilon)}$
19: return True if $\ell(x, p_\theta, q) > \delta$ else False
```

5.3 Implementation Details

- **Models Used:** T5 (for paraphrasing), ChatGPT-paraphraser (for diffusion noising), TrOCR or T5 (for diffusion denoising), GPT-2 (as source model for log-probabilities), BERT (for similarity computation).
- **Libraries:** PyTorch, HuggingFace Transformers, GPyTorch, BERTScore.
- **Parameters:** Number of perturbations (N), query budget, uncertainty threshold, diffusion strength, and number of diffusion steps are configurable.
- **Efficiency:** Progress bars (`tqdm`) and caching are used for similarity computations and model queries to optimize runtime.

Our work can be found at: https://github.com/DarcShelly/CS772

6 Results and Discussion

6.1 Model Performance Overview

We evaluated the proposed detection approaches on a dataset of approximately 1100 samples. Given the modest dataset size and the use of small, basic models, the results should be interpreted as qualitative trends rather than definitive quantitative benchmarks. The primary goal was to compare the impact of different perturbation and kernel strategies on detection accuracy.

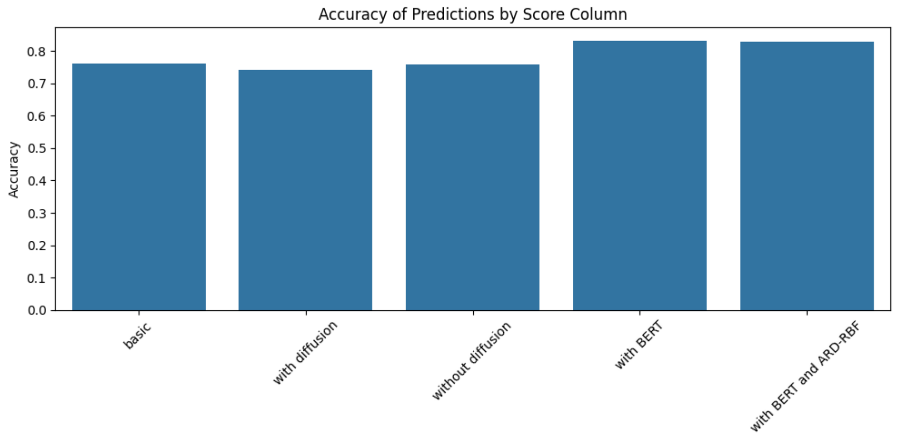

Figure 1: Accuracy of Predictions by Model Configuration

Figure 1 shows the overall accuracy for each configuration:

- **Basic**: Standard DetectGPT with vanilla perturbations.
- **With diffusion**: DetectGPT with diffusion-inspired perturbation refinement.
- **Without diffusion**: Baseline without the diffusion-based process.
- **With BERT**: Kernel replaced by BERTScore-based similarity.
- **With BERT and ARD-RBF**: BERTScore kernel with Automatic Relevance Determination and RBF enhancements.

The baseline and non-diffusion models achieved accuracies around 76%. The diffusion-based approach slightly reduced accuracy, likely due to the added noise not always producing more informative perturbations. Incorporating BERTScore for kernel similarity led to a marked improvement, raising accuracy to over 82%. The ARD-RBF kernel variant performed comparably, suggesting the main gains stem from semantic similarity rather than kernel complexity.

6.2 Confusion Matrix Analysis

To further analyze classification behavior, we present normalized confusion matrices for five representative configurations:

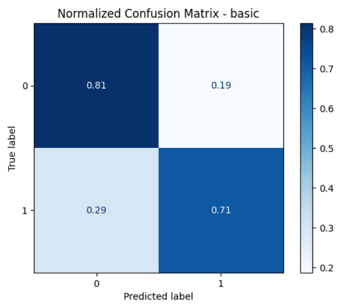

Figure 2: Normalized Confusion Matrix: Basic

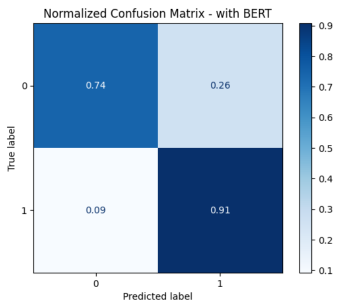

Figure 3: Normalized Confusion Matrix: With BERT

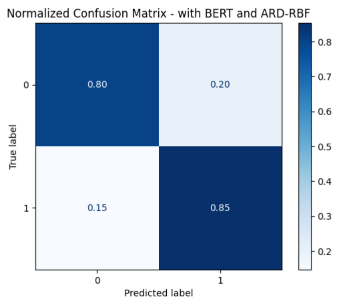

Figure 4: Normalized Confusion Matrix: With BERT and ARD-RBF

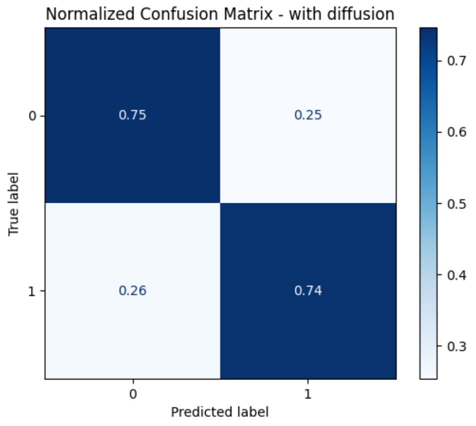

Figure 5: Confusion Matrix: Code with Diffusion Denoising

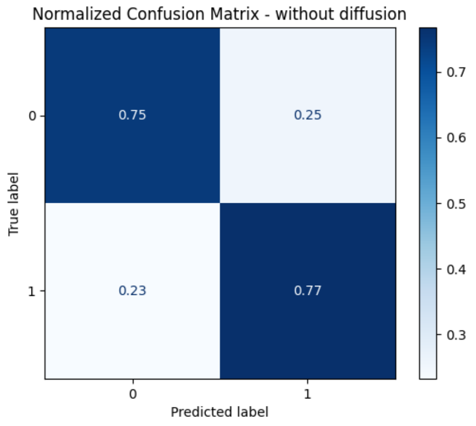

Figure 6: Confusion Matrix: Code without Diffusion Denoising

- **Basic** (Fig. 2): True positive rates were 81% for human-written text and 71% for machine-generated text, with a notable 29% false negative rate for the latter.
- **With BERT** (Fig. 3): Machine-generated text detection improved dramatically to 91% true positive rate, though human text detection dropped slightly to 74%.
- **With BERT and ARD-RBF** (Fig. 4): Performance balanced out, with 80% true positive for human and 85% for machine-generated text, reducing false negatives for both classes.
- **With diffusion**: This configuration achieved 75% true positive rate for human text and 74% for machine-generated text, showing a more balanced but slightly lower overall performance compared to the basic model.
- **Without diffusion**: This approach maintained similar performance for human text detection (75%) but showed a small improvement in machine-generated text detection (77%) compared to the diffusion-based approach.

6.3 Kernel Optimization and Diffusion Effects

Replacing the standard kernel with a BERTScore-based similarity matrix enabled the model to better capture semantic relationships between perturbations. This led to a significant boost in distinguishing

LLM-generated from human-written text, as shown by both accuracy and confusion matrices. The ARD-RBF kernel offered further refinement, balancing class-wise detection rates.

The diffusion-inspired perturbation process showed mixed results in our experiments. While it maintained reasonable detection rates (74% for machine-generated text), it did not outperform the model without diffusion (77% for machine-generated text). This suggests that the added noise in the diffusion process may not always have produced more informative or diverse paraphrases in this small-scale setting.

6.4 Limitations and Qualitative Considerations

- The dataset size (1100 samples) limits statistical significance; observed trends are qualitative.
- All models were intentionally kept small and basic for proof-of-concept, not for maximizing accuracy.
- The diffusion-based perturbation mechanism may require further tuning or larger models to fully realize its potential.
- Results are best interpreted as a demonstration of the value of semantic kernel optimization (via BERTScore) and active sampling with Gaussian Processes for efficient, query-light LLM text detection.

6.5 Computational Efficiency Analysis

In addition to detection accuracy, computational efficiency is a critical consideration for practical deployment. We measured the execution time required for each approach, separating the main computational components for clarity:

Configuration	Main Processing (s)	Prediction/Activation (s)	Total Time (s)
Basic (original)	–	–	5.6
BERTScore kernel	19.1 (BERTScore calc)	3.0 (activation)	22.1
BERT with ARD-RBF	22.3 (BERTScore calc)	2.3 (activation)	24.6
Diffusion	24.7 (diffusion perturbation)	0.3 (score calc)	25.0

Table 2: Computation time for different model configurations.

The results in Table 2 show that the original baseline method is by far the fastest, completing in just 5.6 seconds. Enhanced methods that rely on BERTScore for kernel optimization require significantly more time, primarily due to the semantic similarity computations. The BERTScore kernel approach takes 19.1 seconds for BERTScore calculation and 3.0 seconds for activation, while the ARD-RBF variant is slightly slower overall.

The diffusion-based perturbation method is the most computationally intensive, with 24.7 seconds spent on generating refined perturbations, though its prediction score calculation is very fast (0.3 seconds). Despite its high computational cost, this method did not yield improved accuracy over the BERT-based approaches.

In summary, while BERT-based and diffusion-based enhancements offer improved detection accuracy, they come at a substantial computational cost compared to the original method. This trade-off should be considered when selecting an approach for real-world or large-scale deployments.

6.6 Summary

Overall, the experiments highlight three key findings: (1) the importance of semantic similarity measures (BERTScore) in kernel design for Bayesian surrogate models, (2) the mixed effectiveness of diffusion-inspired perturbation refinement, and (3) the demonstration that even simple models can benefit from active sampling and kernel optimization when data and query budgets are limited.

7 Conclusion and Future work

In this project, we explore and enhance state-of-the-art methods for detecting text generated by large language models (LLMs), addressing the critical societal need for responsible AI use. We used an enhanced version of DetectGPT which is based on a Bayesian surrogate model to reduce the number of queries required. We also tried to explore the model by using the diffusion-based perturbation mechanism for more diverse and informative text variants, improving the robustness of the detection process without sacrificing semantic similarity. Some changes were made in the GP model where the RBF+BERTScore kernel showed more promising results compared to the BERTScore kernel. The limitation of this method is that it needs source LLM log probabilities which limits its use case to open source LLMs.

Together, these enhancements led to a more efficient and potentially more accurate detection pipeline that reduces computational cost while maintaining strong performance. The implementation demonstrates the potential of combining uncertainty estimation with semantic-aware similarity measures to detect LLM-generated text effectively. This contributes a promising direction for scalable, generalizable AI-generated content detection systems in real-world applications. Possible future work may include exploring different types of perturbation techniques, using different Bayesian model other that GP, using different types of kernels for GP.

8 Acknowledgements

We would like to express my sincere gratitude to Professor Piyush Rai for offering this course and for his insightful lectures that laid a strong foundation for this project. We are also thankful to the teaching assistants for their support and valuable feedback throughout the semester.

We are grateful to the creators and maintainers of the open-source tools and models (such as T5, BERTScore, and GPy) that were crucial for building and evaluating the Bayesian detection pipeline described in this report.

9 Disclaimer

We confirm that this project is original work done specifically for this coursse (CS772) and has not been submitted for any other course or internship, nor have we received credit for it elsewhere.

References

[1] Sebastian Gehrmann, Hendrik Strobelt, and Alexander M. Rush. Gltr: Statistical detection and visualization of generated text. `https://arxiv.org/abs/1906.04043`, 2019. arXiv:1906.04043.

[2] 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 (ACL)*, pages 1808–1822, 2020.

[3] Yibo Miao, Hongcheng Gao, Hao Zhang, and Zhijie Deng. Efficient detection of llm-generated texts with a bayesian surrogate model. *arXiv preprint arXiv:2305.16617*, 2023.

[4] Eric Mitchell, Yoonho Lin, Antoine Lee, Xuechen Chen, Christopher D. Manning, and Chelsea Finn. Detectgpt: Zero-shot machine-generated text detection using probability curvature. *arXiv preprint arXiv:2301.11305*, 2023.

[5] Irene Solaiman, Miles Brundage, Jack Clark, Amanda Askell, Ariel Herbert-Voss, Jeffrey Wu, Alec Radford, Gretchen Krueger, Jenna Kim, Miles McCain, et al. Release strategies and the social impacts of language models. `https://openai.com/research/release-strategies`, 2019. Accessed: 2023-12-01.