University of Pisa
Department of Information Engineering

Business & Project Management

Cheating Detection in Interviews using GPT Reverse Engineering

Malik Muhammad Uzair
Nedal Adham Ahmed Ezzeldeen Mohamed Elazaby

Year 2023/2024

Contents

1 Introduction

1.1 Background

Within the world of academic evaluations and professional interviews, maintaining fairness and ensuring the authenticity of a candidate's responses has always been paramount. The credibility of recruitment processes and the value of certifications rely on the assurance that the answers provided reflect an individual's true knowledge and ability. However, the rise of advanced AI-driven language models has blurred the line between responses crafted by humans and those generated by machines.

Picture a scenario where an individual is being evaluated for a prestigious position or certification based on their spoken answers during an interview or oral exam. While the candidate may seem to articulate thoughtful and coherent responses, there is the hidden possibility that those answers were crafted with the help of an AI, unbeknownst to the examiners. Such a revelation would jeopardize the integrity of the entire evaluation process, undermining the fairness of the competition and devaluing the efforts of candidates who genuinely rely on their own skills and knowledge.

The shift towards remote interviews and virtual evaluations, accelerated by global events, has further complicated the issue. With face-to-face supervision reduced and a growing dependence on online platforms, individuals now have greater opportunities to exploit technology by generating AI-assisted answers. This presents a new layer of complexity in ensuring that assessments are conducted fairly and authentically.

In this report, we will explore the development and methodology behind our AI Detection Model for Interviews and Oral Exams. We will examine the technology's potential to distinguish between AI-generated and human-generated responses, and its critical role in preserving the integrity and fairness of the assessment process across various professional and academic settings.

1.2 Objectives

The primary objective of this project is to develop a system that can:

1. Record the audio of interview/exam responses.
2. Convert the audio to text.
3. Analyze the text using a GPT detection model.
4. Determine the likelihood that the responses are AI-generated.

This system aims to enhance the integrity of the interview/exam process by identifying candidates who may be using AI to generate their responses.

1.3 Alternative Uses

1.3.1 Customer Support and Call Centers

Scenario: In outsourced customer support, companies may face situations where agents use AI-generated responses to handle customer queries, rather than relying on their own expertise.

Application: The system can assess agent responses in real-time or after calls, ensuring that agents are not overly dependent on AI tools to respond to customers.

Benefit: It ensures genuine human interaction and the authenticity of responses, improving customer satisfaction.

2 Methodology

2.1 Theoretical Background

We utilize a reverse engineering approach with GPT models to detect AI-generated text. The theoretical foundation of our system is based on zero-shot learning, which does not require labeled training data. Instead, it uses the raw log probabilities computed by a generative model to determine if a passage was likely generated by an AI.

The core of our detection mechanism is inspired by the DetectGPT method, which hypothesizes that texts generated by GPT models lie in areas of negative curvature of the log probability function. By introducing perturbations to the text and analyzing the changes in log probabilities, we can distinguish between human-generated and AI-generated text.

2.2 Implementation Steps

The AI detection system was implemented using a combination of speech recognition, API integration of ready-made code to detect AI-generated text, and a graphical user interface (GUI) to provide a comprehensive tool for detecting AI-generated responses in interviews and oral exams. The following steps outline the methodology:

2.2.1 Audio Recording and Speech Recognition

- **Process:** Audio responses from interviews and oral exams are recorded using the microphone or from the computer itself. The `speech_recognition` library is utilized to convert spoken words into text.
- **Technology Used:** The `speech_recognition` library handles audio capture and transcription, adjusting for ambient noise and recognizing speech using Google's Speech-to-Text API.

2.2.2 Real-Time Listening

- **Process:** The system listens to the audio input in real-time while the user is speaking. A separate thread manages this process to ensure the UI remains responsive.
- **Implementation:** The listening process continues until the user stops it manually. The captured audio is transcribed into text, which is accumulated and displayed in the UI.

2.2.3 AI Text Detection using Hugging Face API

- **Process:** After capturing the entire response, the text is sent to the Hugging Face API for analysis. The API endpoint used is specifically designed to detect whether the text was generated by AI or a human.

- **Technology Used:** The Hugging Face API, particularly the model akshayvkt/detect-ai-text, is used for this purpose. The API is queried with the transcribed text, and the response includes probabilities indicating the likelihood of the text being AI-generated.

2.2.4 API Interaction and Error Handling

- **Process:** The system interacts with the Hugging Face API to fetch results. Errors and exceptions during API requests are handled gracefully, with error messages displayed in the UI.
- **Implementation:** The `requests` library is used to handle API calls, with robust error checking to ensure the system can manage network issues and invalid responses.

2.2.5 Result Interpretation and Display

- **Process:** The results from the API are analyzed to determine the likelihood that the response was generated by AI. The system extracts relevant scores from the API response and compares them.
- **Outcome:** The results are presented to the user with a clear message indicating whether the text is more likely to be human-generated or AI-generated.

2.2.6 System Operation and Maintenance

- **Process:** The system includes options for starting and stopping the listening process, updating the UI with real-time feedback, and quitting the application.
- **Implementation:** The Tkinter buttons and status updates provide a user-friendly experience, allowing for straightforward operation and easy monitoring of the system's status.

2.2.7 User Interface Setup

- **Technology Used:** The user interface (UI) was developed using the Tkinter library, a standard GUI toolkit for Python. The UI includes text display areas and control buttons for starting, stopping, and quitting the application.
- **Components:** The interface features a text display for status updates, buttons for controlling the listening process, and labels for status messages.

2.3 Drawbacks

AI detection systems can face several general drawbacks, such as issues with false positives and negatives, which can lead to misclassification of human and AI-generated content. The effectiveness of these systems is heavily dependent on the quality and diversity of the training data, which may affect their performance across different contexts. Additionally, as AI technology evolves, existing detection models might become outdated, requiring constant updates and retraining to stay effective. Privacy concerns also arise, as analyzing personal content necessitates robust data protection measures.

There are also specific limitations such as dependency on the Hugging Face API's availability and potential stability issues. The real-time performance of the system can be affected by delays in audio processing and API responses. Moreover, while basic error handling is implemented, it may not cover all edge cases, leading to incomplete analysis in some situations. Thread management for real-time listening needs to be carefully managed to avoid performance problems or crashes.

3 Application

The layout of the application is designed to be user-friendly and efficient, divided into two main sections: audio recording and result display.

In the first section, users can start and stop the audio recording process, capturing responses from interviews and oral exams. The second section displays the transcribed text and the result of the AI detection analysis. This setup allows users to easily monitor and control the recording process while immediately reviewing the detection outcomes.

We've recently developed a robust graphical application aimed at enhancing the integrity of interview and exam processes by detecting AI-generated responses. This tool simplifies the audio recording and analysis workflow, providing real-time feedback on whether responses are likely AI-generated. The application's design ensures ease of use, making it a valuable asset for interviewers and examiners.

Technology Stack Our application is built using Python and leverages the Tkinter library for creating a cross-platform graphical user interface. Tkinter provides a straightforward way to develop an intuitive and interactive interface for users.

Paradigm The application follows a client-server paradigm. The server, represented by the backend system, processes audio input, transcribes it into text, and sends it to the Hugging Face API for AI detection. The client, which is the Tkinter-based frontend, handles user interactions, such as starting and stopping recordings, and displays the results. The client communicates with the server to receive and present the analysis outcomes.

3.1 Backend

The backend of our system is implemented using Python and consists of the following components:

- **Speech Recognition**: Utilizes the `speech_recognition` library to capture and convert audio to text. The system listens to responses through the microphone and processes the audio in real-time, transcribing spoken words into written text.

```
import speech_recognition as sr

class SpeechRecognitionHandler:
    def __init__(self):
        self.recognizer = sr.Recognizer()
        self.microphone = sr.Microphone()
        self.listening = False
        self.full_answer = ""  # Holds the entire response
    def listen_for_full_answer(self):
        self.full_answer = ""
        print("Listening for full answer...")  # Debugging statement
        try:
            with self.microphone as mic:
                self.recognizer.adjust_for_ambient_noise(mic, duration=0.2)
                print("Listening...")  # Debugging statement

                while self.listening:
                    audio = self.recognizer.listen(mic)
                    try:
                        text = self.recognizer.recognize_google(audio)
                        self.full_answer += text + " "
                        print(f"Partial answer: {text}")
                    except sr.UnknownValueError:
                        print("Could not understand audio.")
                    except sr.RequestError as e:
                        print(f"RequestError: {e}")

        except Exception as e:
            print(f"Error during listening: {e}")  # Debugging statement

    def get_full_answer(self):
        return self.full_answer
```

- **AI Detection API**: Integrates with the Hugging Face API for GPT detection. Once the audio is converted to text, the system sends this text to the API. The API then analyzes the text and returns a response indicating the likelihood that the content is generated by AI.

```
import requests

API_URL="https://api-inference.huggingface.co/models/akshayvkt/detect-ai-text"
HEADERS = {"Authorization": "Bearer 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"}

def is_model_ready():
    try:
        response = requests.get(API_URL, headers=HEADERS, timeout=10)
        response.raise_for_status()
        data = response.json()
        print(f"Model status response: {data}")  # Debugging statement
        return "error" not in data
    except requests.exceptions.RequestException as e:
        print(f"Exception during API request: {e}")  # Debugging statement
        return False

def query(payload):
    try:
        response = requests.post(API_URL, headers=HEADERS, json=payload,
        timeout=10)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"Exception during API request: {e}")  # Debugging statement
        return {"error": str(e)}

def analyze_answer(full_answer):
    if not full_answer:
        return "No answer detected."

    # Call the API with the full answer
    output = query({"inputs": full_answer.strip()})
    print(f"API Output: {output}")  # Debugging statement

    # Extract the labels and scores
    if isinstance(output, list) and len(output) > 0:
        labels_and_scores = output[0]
        human_score = None
        ai_score = None

        # Find scores for human and AI
        for item in labels_and_scores:
            if item['label'].lower() == 'human':
                human_score = item['score']
            elif item['label'].lower() == 'ai':
                ai_score = item['score']

        # If both scores are found, round and compare them
        if human_score is not None and ai_score is not None:
            human_score_rounded = round(human_score * 100, 2)
            ai_score_rounded = round(ai_score * 100, 2)

            if human_score > ai_score:
                message = f"Text is mostly by a human (Human:
                {human_score_rounded}%, AI: {ai_score_rounded}%)"
            else:
                message = f"Text is mostly by AI (AI: {ai_score_rounded}%,
                Human: {human_score_rounded}%)"
        else:
            message = "Unable to detect scores properly from API output."
    else:
        message = "Unexpected API output format."

    print(message)  # Debugging statement
    return message
```

3.2 Frontend

The frontend of the system is implemented using Python's Tkinter library to provide a graphical user interface (GUI). The main components of the frontend include:

3.2.1 User Interface Elements

- **Text Display Area:** A section for showing status messages.
- **Buttons**

3.2.2 Functional Components

- **Start Listening:** Initiates the audio capture and transcription process.
- **Stop Listening:** Halts audio capture and proceeds with analyzing the recorded text.
- **Quit:** Closes the application.

3.2.3 Real-Time Updates

The application provides real-time feedback to the user regarding the model's readiness and the status of audio capture and analysis.

This setup ensures a user-friendly experience for interacting with the cheating detection system, allowing users to start, stop, and monitor the process with ease.

3.3 Frontend Code

```
import tkinter as tk
from tkinter import messagebox
import threading

class CheatingDetectionApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Cheating Detection in Interviews")

        self.text_display = tk.Text(root, wrap=tk.WORD, height=10, width=50)
        self.text_display.pack(pady=10)

        self.start_button = tk.Button(root, text="Start Listening",
        command=self.start_listening)
        self.start_button.pack(side=tk.LEFT, padx=10)

        self.stop_button = tk.Button(root, text="Stop Listening",
        command=self.stop_listening, state=tk.DISABLED)
        self.stop_button.pack(side=tk.LEFT, padx=10)

        self.quit_button = tk.Button(root, text="Quit",
        command=self.quit_application)
        self.quit_button.pack(side=tk.RIGHT, padx=10)

        self.listening = False
        self.model_ready = False
        self.full_answer = ""  # Holds the entire response

        self.update_status_label("Model is loading...")
        self.root.after(1000, self.check_model_status)

    def check_model_status(self):
        if is_model_ready():
            self.model_ready = True
            self.update_status_label("Model is ready. Press 'Start Listening'
            to begin.")
        else:
            self.update_status_label("Model is still loading... Retrying in 5
            seconds.")
            self.root.after(5000, self.check_model_status)  # Check every 5
            seconds

    def update_status_label(self, message):
        self.text_display.delete(1.0, tk.END)
        self.text_display.insert(tk.END, message + "\n")
        self.root.update_idletasks()

    def start_listening(self):
        if not self.model_ready:
            self.update_status_label("Model is not ready yet.")
            return

        self.listening = True
        self.start_button.config(state=tk.DISABLED)
        self.stop_button.config(state=tk.NORMAL)
        self.update_status_label("Listening for your answer...")

        self.listening_thread =
        threading.Thread(target=self.listen_for_full_answer)
        self.listening_thread.start()

    def stop_listening(self):
        self.listening = False
        self.start_button.config(state=tk.NORMAL)
        self.stop_button.config(state=tk.DISABLED)
        self.update_status_label("Stopped listening. Processing your
        answer...")

        # Analyze the full answer
        self.analyze_answer()
```

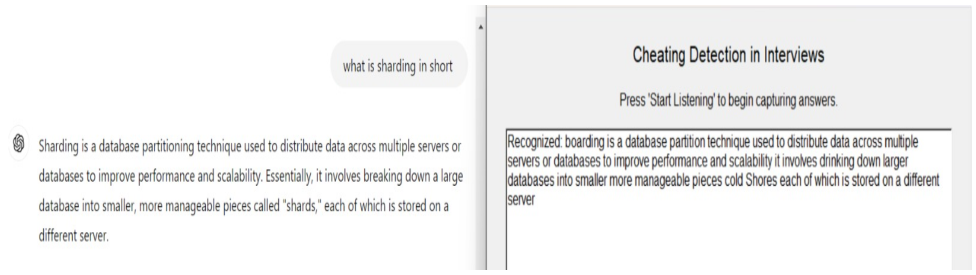
```
def listen_for_full_answer(self):
    self.full_answer = ""
    try:
        with self.microphone as mic:
            self.recognizer.adjust_for_ambient_noise(mic, duration=0.2)

            while self.listening:
                audio = self.recognizer.listen(mic)
                try:
                    text = self.recognizer.recognize_google(audio)
                    self.full_answer += text + " "
                    self.update_status_label(f"Recognized:
                    {self.full_answer.strip()}")
                except sr.UnknownValueError:
                    print("Could not understand audio.")
                except sr.RequestError as e:
                    self.update_status_label(f"Could not request results;
                    {e}")

def analyze_answer(self):
    if not self.full_answer:
        self.update_status_label("No answer detected.")
        return

    # Call the API with the full answer
    output = query({"inputs": self.full_answer.strip()})

    # Extract the labels and scores
    if isinstance(output, list) and len(output) > 0:
        labels_and_scores = output[0]
        human_score = None
        ai_score = None

        # Find scores for human and AI
        for item in labels_and_scores:
            if item['label'].lower() == 'human':
                human_score = item['score']
            elif item['label'].lower() == 'ai':
                ai_score = item['score']

        # If both scores are found, round and compare them
        if human_score is not None and ai_score is not None:
            human_score_rounded = round(human_score * 100, 2)
            ai_score_rounded = round(ai_score * 100, 2)

            if human_score > ai_score:
                message = f"Text is mostly by a human (Human:
                        {human_score_rounded}%, AI: {ai_score_rounded}%)"
                    else:
                        message = f"Text is mostly by AI (AI:
                        {ai_score_rounded}%, Human: {human_score_rounded}%)"
                else:
                    message = "Unable to detect scores properly from API output."
            else:
                message = "Unexpected API output format."


            self.update_status_label(message)


        def quit_application(self):
            self.listening = False
            self.root.destroy()


if __name__ == "__main__":
    root = tk.Tk()
    app = CheatingDetectionApp(root)
    root.mainloop()
```

4 Results

We conducted several tests to evaluate the system's effectiveness. The results were analyzed by comparing the system's recognition of human-generated text versus AI-generated text using a speech-to-text mechanism and an AI text-detection model.

4.1 Test 1: AI-Generated Response

We tested a scenario where the input response was predominantly generated by an AI model. As shown in Figure 2, our system accurately detected this, identifying the text as being **91.24% AI-generated**, with only **8.76% attributed to human input**. This demonstrates the system's ability to discern AI-driven text accurately when significant portions of the response originate from an artificial intelligence source.



what is sharding in short

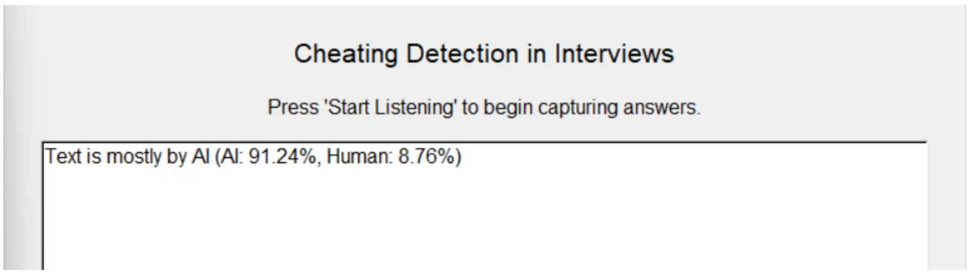
Sharding is a database partitioning technique used to distribute data across multiple servers or databases to improve performance and scalability. Essentially, it involves breaking down a large database into smaller, more manageable pieces called "shards," each of which is stored on a different server.

Cheating Detection in Interviews

Press 'Start Listening' to begin capturing answers.

Recognized: boarding is a database partition technique used to distribute data across multiple servers or databases to improve performance and scalability it involves drinking down larger databases into smaller more manageable pieces cold Shores each of which is stored on a different server

Figure 1: AI-Generated Response Detection Test


Cheating Detection in Interviews

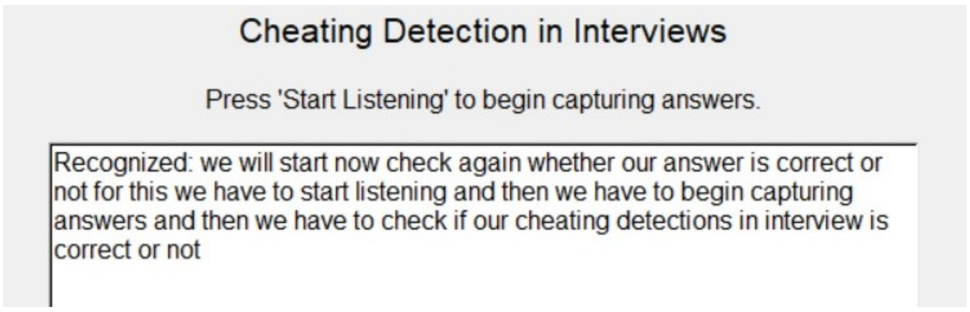
Press 'Start Listening' to begin capturing answers.

Text is mostly by AI (AI: 91.24%, Human: 8.76%)

Figure 2: AI-Generated Response Detection Result

4.2 Test 2: Human-Generated Response

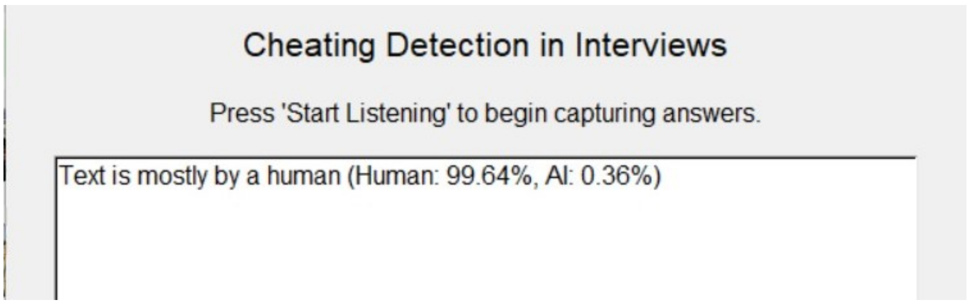
A second test was conducted using a response that was entirely human-generated. As seen in Figure 4, our system effectively detected this as well, showing that the text was **99.64% human-generated** with only **0.36% recognized as AI-generated text**. This confirms that the system can correctly differentiate human speech and minimizes the false detection of AI involvement.


Cheating Detection in Interviews

Press 'Start Listening' to begin capturing answers.

Recognized: we will start now check again whether our answer is correct or not for this we have to start listening and then we have to begin capturing answers and then we have to check if our cheating detections in interview is correct or not

Figure 3: Human-Generated Response Detection Test


Cheating Detection in Interviews

Press 'Start Listening' to begin capturing answers.

Text is mostly by a human (Human: 99.64%, AI: 0.36%)

Figure 4: Human-Generated Response Detection Result

These results indicate that the system is capable of reliably distinguishing between AI and human-generated text, making it a valuable tool for ensuring authenticity in spoken responses during interviews. Our tests affirm that the accuracy is high when identifying both AI-driven and human-originated responses.

5 Conclusion

In conclusion, our Cheating Detection System represents a significant leap forward in the effort to identify AI-generated content in interviews and oral exams. Although the system is still in its early stages, it lays the groundwork for a more accurate and efficient detection mechanism. Currently, the system requires human supervision to refine its accuracy and adapt to various contexts. This collaborative effort will allow the system to improve and evolve over time.

Our methodology, while cutting-edge, is part of a rapidly evolving field that holds great potential for further research and development. We anticipate advancements that will reduce the dependency on resource-heavy GPT models, thereby addressing memory constraints and boosting efficiency. Despite being in the developmental phase, our system already showcases the ability to detect AI-generated content with notable precision. As technology continues to advance, integrating human expertise with innovative solutions promises to enhance academic integrity and expand the system's applications. The future of AI detection systems is promising, with ongoing advancements set to make these tools even more effective and accessible.

A Appendices

A.1 Code Listing

The complete code for the backend is provided in Section 3.1 and 3.2. This code is essential for the operation of the system and can be further customized based on specific requirements.

A.2 References

- DetectGPT: Zero-Shot Machine-Generated Text Detection using Probability Curvature. `https://example.com/paper`
- Hugging Face API Documentation. `https://huggingface.co/docs`
- Python `speech_recognition` Library. `https://pypi.org/project/SpeechRecognition/`