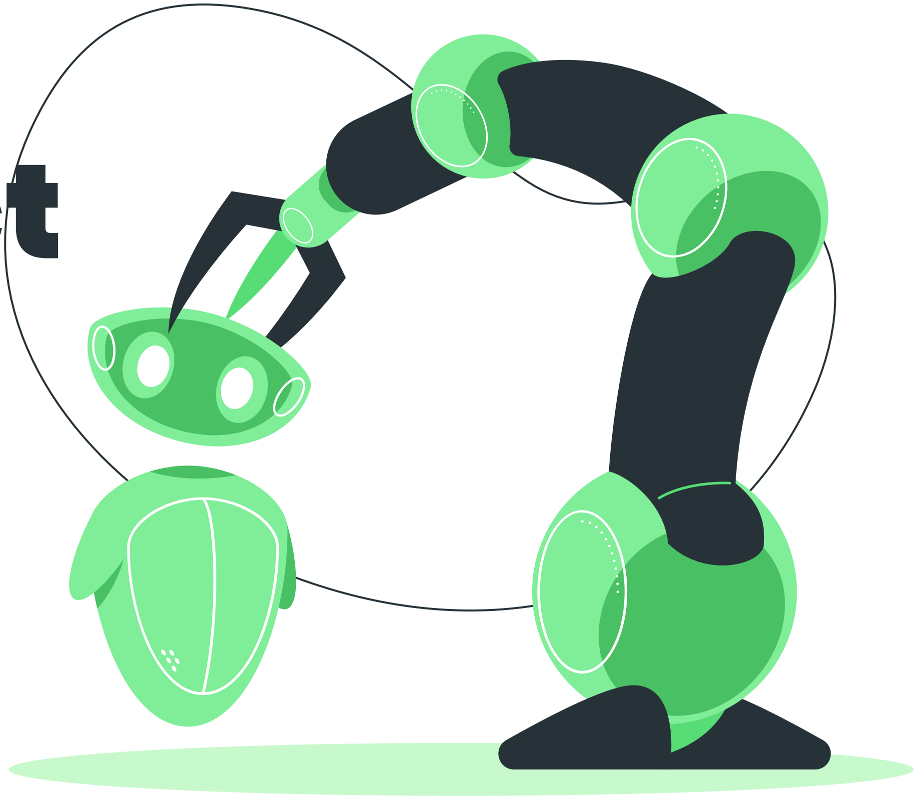
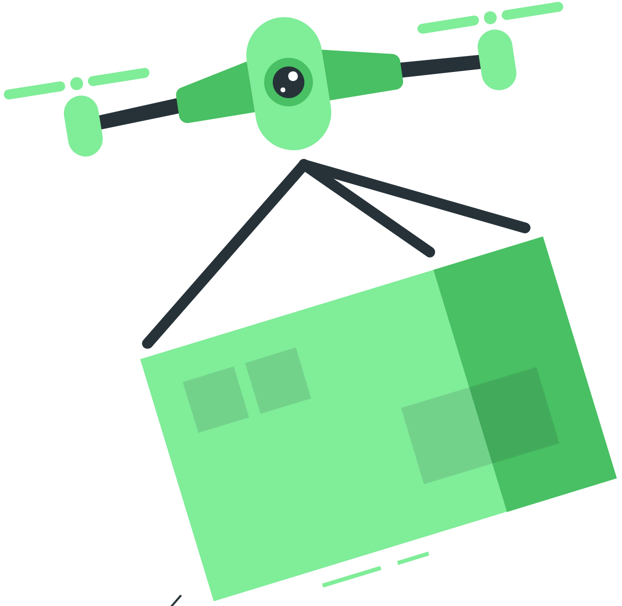
Final Project CSE 143



Karthik Chaparala, Jiancheng Xiong, Atharva Tawde, Swayam Shah

The Problem

- ★ AI generated content is rapidly increasing due to popular tools like ChatGPT
- ★ It is becoming harder to detect when something was written by AI
- ★ It's important to be able to distinguish between human-written and AI written text

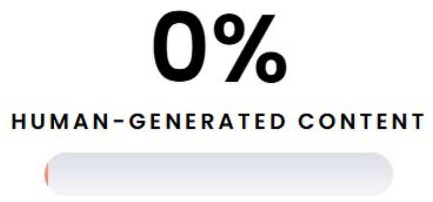


The Solutions?

- With the rise of AI, many have tried to develop a system that can detect writing that was generated by an AI tool
- These tools are unfortunately faulty, often flagging text as AI generated when it was written by a human and vice versa
  - People may lose opportunities despite doing their own work
  - Others get farther ahead without the skills necessary for future tasks

Our Goal

★ Our goal is to create a system that is more accurate than current services like GPT Zero in order to fulfill the need for AI detection without the increasingly common occurrence of a incorrect result



Our Approach: Data Collection

We collected two sets of data:

Human-Written

Articles from Wikipedia that were written in a uniform, academic style

AI Generated

Ollama generated articles on the same Wikipedia topics

Due to the type of data we collected, our model is more suited for the formal style used by Wikipedia writers, though it still remains accurate for normal “speaking” inputs.

Example Data

Wikipedia

“The Andromeda Galaxy is a barred spiral galaxy and is the nearest major galaxy to the Milky Way.”

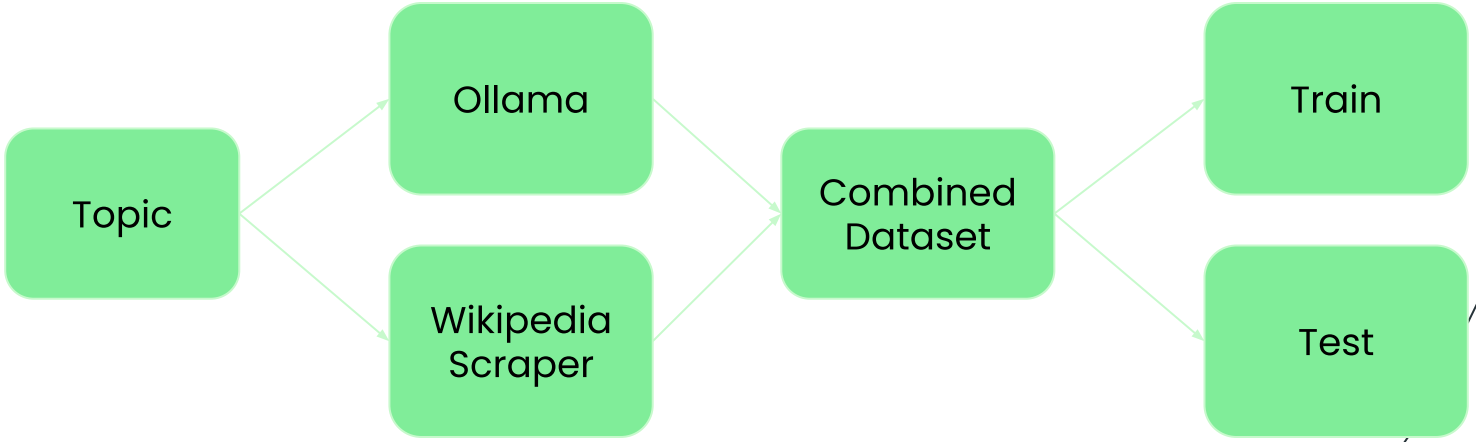
Ollama

“The Andromeda Galaxy, also known as Messier 31, M31, or NGC 224, is a spiral galaxy approximately 2.5 million light-years away in the constellation Andromeda.”

Our Approach: Data Collection

- In order to train our models (specifically the Logistic Regression and LSTM elements), we needed a labeled dataset of both human and AI generated content. In order to get a larg sample of human-written data, we turned to Wikipedia, and we would use Ollama (a localhost LLM) to generate AI-written data
- In order to ensure that both AI and human datasets would contain similar topics for data, file of Wikipedia article topics was created, and for each topic we would scrape the Wikip article, as well as prompt Ollama to generate a wikipedia-formatted article.
- In order to ensure that text artifacting wouldn't be a factor from our Wikipedia articles, as as to ensure consistency in the style of output for both human and AI generated, we removed references and selected works, as well as other Wikipedia exclusive sections tha would not generate.

Our Approach: Data Collection



Our Approach: Data Collection

```
Dante Alighieri
Venus (planet)
Bengal Tiger
The Great Barrier Reef
Ancient Rome
Sanskrit
Post-Impressionism
Astrophysics
Medieval Philosophy
Beethoven's Ninth Symphony
Climate Change
Artificial Intelligence
Solar System
Leonardo da Vinci
Theory of Relativity
Wright Brothers
Bioluminescence
American Revolution
Columbia University
Neanderthals
Human Genome Project
Hubble Space Telescope
Egyptian Mythology
Renaissance
Marie Curie
```

Some topics we have

Our Approach: Wikipedia

```python
import wikipediaapi
import os

def scrape_wikipedia_page(page_title, file_path, main_file_path):
    wiki = wikipediaapi.Wikipedia('AIvHumanproj', 'en')
    page = wiki.page(page_title)

    if not page.exists():
        print(f"The page '{page_title}' does not exist.")
        return

    # Extract text up to the "References" section
    text = page.text
    if "References" in text:
        text = text.split("References")[0]
    if "Selected works" in text:
        text = text.split("Selected works")[0]

    # Extract sentences
    lines = text.split('\n')
    cleaned_lines = [
        line.strip() for line in lines
    ]

    # Extract sentences from cleaned lines
```

```python
    sentences = '. '.join(cleaned_lines).split('. ')

    os.makedirs(os.path.dirname(file_path), exist_ok=True)

    with open(file_path, 'w', encoding='utf-8') as file:
        for sentence in sentences:
            if len(sentence) > 15:
                file.write("\"" + sentence.strip() + '.",0\n')

    with open(main_file_path, 'a', encoding='utf-8') as main:
        for sentence in sentences:
            if len(sentence) > 15:
                main.write("\"" + sentence.strip().replace('"', '').replace("'", "") + '.",0\n')

    print(f"Sentences saved to {file_path}")


# Read topics from 'topics.txt' and scrape each page
with open("topics.txt", 'r', encoding='utf-8') as topics_file:
    page_titles = [line.strip() for line in topics_file if line.strip()]
```

```python
with open("topics.txt", 'r', encoding='utf-8') as topics_file:
    page_titles = [line.strip() for line in topics_file if line.strip()]

output_directory = "human_written"

main_name = "MAIN_H.txt"
main_file_path = os.path.join(output_directory, main_name)

with open(main_file_path, 'w', encoding='utf-8') as main:
    pass

for title in page_titles:
    #output_file_path = f"{title}_H.txt"
    file_name = title.replace(' ', '_') + '_H.txt'
    file_path = os.path.join(output_directory, file_name)

    scrape_wikipedia_page(title, file_path, main_file_path)
```

Our Approach: AI generation

```python
import subprocess
import os

threshold = 15
model = "gemma2:27b"
input_file = "topics.txt"

def generate_from_file(input_file, model):
    try:
        # Open the main file to append content
        with open(f"ai_written/MAIN_A.txt", "a", encoding="utf-8") as main_file:

            # Read the topics from the input file
            with open(input_file, "r", encoding="utf-8") as file:
                topics = file.readlines()

            for topic in topics:
                topic = topic.strip()  # Remove leading/trailing spaces/newlines
                if not topic:  # Skip empty lines
                    continue

                #Generate content for the current topic
                prompt = f"Provide a clear and detailed explanation of the Wikipedia topic: {topic}. Do not include headers
                result = subprocess.run(
                    ["ollama", "run", model],
                    input=prompt,
                    text=True,
                    capture_output=True
                )

                # Check if the model returned an error
                if result.returncode != 0:
                    print(f"Error generating content for {topic}: {result.stderr}")
                    continue

                # Get the generated content
                generated_content = result.stdout

                # For testing purposes, we're using static content here
                # generated_content = f"Header. {topic}. This is an explanation of the topic {topic}. This is also another

                # Split the content into sentences (basic approach using periods as separators)
                sentences = generated_content.split(". ")
                sentences = [sentence.strip() + '.' for sentence in sentences if sentence and len(sentence) >= threshold]

                # Save the generated content to a separate file for the topic
                filename = f"{topic.replace(' ', '_')}_A.txt"
                with open(f"ai_written/{filename}", "w", encoding="utf-8") as topic_file:
                    for sentence in sentences:
```

```python
                        topic_file.write(f"\"{sentence}\"" + ",1\n")

                # Append the content to the main.txt file (one sentence per line)
                # main_file.write(f"### {topic} ###\n")
                for sentence in sentences:
                    sentence = sentence.replace('"', '').replace("'", "")
                    main_file.write(f"\"{sentence}\"" + ",1\n")
                # main_file.write("\n")

                print(f"Generated content for {topic} saved to '{filename}'.")

    except Exception as e:
        print(f"An error occurred: {e}")

if __name__ == "__main__":
    generate_from_file(input_file="topics.txt", model=model)
```

Example of Dataset

```
"Amelia Opie was an English novelist and poet who lived from 1769 to 1853.",1
"Born into a literary family in Norwich, England, she began writing poetry and prose at a young age.",1
"Her father was the Reverend John Opie, a prominent preacher and author of religious works.",1
"Opies early life was marked by tragedy, losing her mother when she was just a child.",1
"This experience deeply affected her writing, which often explored themes of loss, grief, and the comple
"She published her first collection of poems in 1793, titled Poems. Opie gained recognition for her nove
"Some of her most well-known works include The Fathers Secret (1808), Richmond (1815), and The Maid of H
"Her novels often featured strong female characters navigating societal expectations and moral dilemmas.
"Opies writing style was characterized by its clarity, simplicity, and emotional depth.",1
"She possessed a keen understanding of human nature and her stories resonated with readers for their rel
"Her versatility as a writer showcased her intellectual curiosity and her desire to explore different ge
"Nonetheless, she persevered and achieved considerable success, publishing over 50 novels and numerous c
"She is remembered today for her contributions to English literature and for paving the way for future g
"The Andromeda Galaxy, also known as Messier 31, M31, or NGC 224, is a spiral galaxy approximately 2.5 m
"Its the closest major galaxy to our own Milky Way and is so large its visible to the naked eye under da
"With an estimated trillion stars, its twice as massive as the Milky Way and spans about 220,000 light-y
"Andromeda has a central bulge containing older stars, surrounded by a disk with spiral arms populated b
"These spiral arms are sites of active star formation, giving rise to brilliant nebulae and star cluster
"Andromeda also harbors a supermassive black hole at its core, estimated to be over 100 million times th
"Like most large galaxies, it has satellite galaxies orbiting it, including M32, a dwarf elliptical gala
"Due to their gravitational attraction, Andromeda and the Milky Way are on a collision course, expected 
"This merger will result in a spectacular display of celestial fireworks as stars and gas clouds interac
"Observations of Andromeda have been crucial for understanding galactic evolution and structure, providi
"Its proximity and impressive size make it an ideal target for astronomers to study the properties and p
```

Basically just a CSV of “<text>”, [0-1]

Our Approach: Model Creation

For our models, we decided to do test 3 different approaches and create one comprehensive system using these 3 approaches.

1. **Content-based model:** We use a simple Logistic Regression model to analyze the content of the text. This model learns to identify patterns in the words and phrases used.
2. **Structural-based model:** We use a Random Forest model to analyze the structural features we extracted earlier. This model learns to identify patterns in the way the text is structured.
3. **LSTM model:** This is a deep learning model that learns to process sequences of text. It's particularly good at capturing long-range dependencies in the text.

Our Approach: Logistic Regression

We thought it would be best to start off simple, since this was a binary classification problem (classifying text as either AI-generated or human-written).

How it works:

1. **Feature Extraction:** Our `CountVectorizer` transforms text into numerical features, essentially counting the occurrences of each word.
2. **Model Training:** The Logistic Regression model learns to assign probabilities to each class (AI-generated or human-written) based on these numerical features.
3. **Prediction:** For a new piece of text, the model calculates the probability of it being AI-generated and compares it to a threshold to make a final prediction.

```
# Train content-based model
def train_content_model(X_train, y_train):
    model = LogisticRegression()
    model.fit(X_train, y_train)
    return model
```

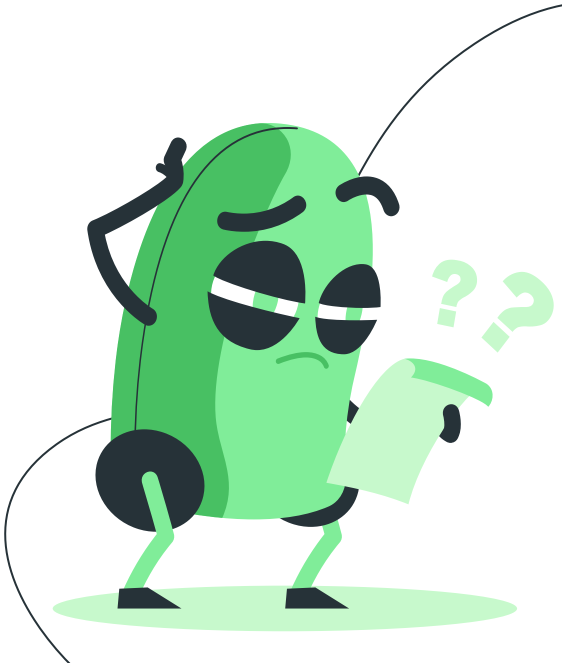
```
# Extract content-based features
vectorizer = CountVectorizer()
X_train_content = vectorizer.fit_transform(X_train_raw)
content_model = train_content_model(X_train_content, y_train)
```

The Problems

Initially, we decided to apply **Logistic Regression** directly to our dataset, expecting that a simpler approach would work. At first, the model showed a promising accuracy of around **85%** on the test set.

However, when we tested it with real hand-typed inputs, it struggled to differentiate between AI-generated and human-written text, especially for shorter prompts with limited information. Despite the high accuracy on the test set, the model failed to generalize well to the more nuanced cases.

So we had to try ***something else***.



Our Approach: Random Forest

This is a learning method that combines multiple decision trees to make more accurate predictions.

```python
# Train structural-based model
def train_structural_model(X_train, y_train):
    model = RandomForestClassifier()
    model.fit(X_train, y_train)
    return model
```

The Random Forest model leverages the structural features extracted from the text, such as sentence length, part-of-speech distribution, and other syntactic information. By combining multiple decision trees, the model can effectively classify text as AI-generated or human-written, even in the presence of complex patterns and noise.

```python
# Load spaCy language model
nlp = spacy.load("en_core_web_sm")

# Extract structural features from text
def extract_structural_features(texts):
    features = []
    for text in texts:
        doc = nlp(text)
        pos_counts = {pos: 0 for pos in ["NOUN", "VERB", "ADJ",
        "ADV", "PRON", "PUNCT"]}

        for token in doc:
            if token.pos_ in pos_counts:
                pos_counts[token.pos_] += 1

        # Sentence-level features
        sentence_length = len(doc)
        num_sentences = len(list(doc.sents))
        avg_sentence_length = sentence_length / num_sentences if
        num_sentences > 0 else 0

        # Add features as a list
        features.append([
            sentence_length,          # Total number of tokens
            num_sentences,            # Number of sentences
            avg_sentence_length,      # Average sentence length
            pos_counts["NOUN"],       # Number of nouns
            pos_counts["VERB"],       # Number of verbs
            pos_counts["ADJ"],        # Number of adjectives
            pos_counts["ADV"],        # Number of adverbs
            pos_counts["PRON"],       # Number of pronouns
            pos_counts["PUNCT"],      # Number of punctuations
        ])
    return np.array(features)
```

Our Approach: LSTM

Embedding Layer:

1. **Word Embeddings:** The tokenized input is fed into an embedding layer. This layer maps each token to a dense vector representation, capturing semantic and syntactic information.

Output Layer:

2. **Classification:** The final output of the LSTM is fed into a dense layer with a sigmoid activation function. This layer outputs a probability between 0 and 1, indicating the likelihood that the input text is AI-generated.

```
# Train LSTM model
def train_lstm_model(texts, labels, max_length=20):
    tokenizer = Tokenizer(num_words=10000)
    tokenizer.fit_on_texts(texts)
    sequences = tokenizer.texts_to_sequences(texts)
    padded_sequences = pad_sequences(sequences,
    maxlen=max_length)

    encoder = LabelEncoder()
    encoded_labels = encoder.fit_transform(labels)

    vocab_size = 10000
    embedding_dim = 128

    model = Sequential([
        Embedding(input_dim=vocab_size,
        output_dim=embedding_dim, input_length=max_length),
        LSTM(64),
        Dropout(0.2),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy',
    metrics=['accuracy'])
    model.fit(padded_sequences, encoded_labels, epochs=10,
    batch_size=32, validation_split=0.2)
    return model, tokenizer, encoder
```

Our Approach: LSTM

Sequential Processing: The LSTM processes the input sequence one token at a time, allowing it to capture the context of each token.

Long-Term Dependencies: Unlike simpler RNNs, LSTMs can learn long-range dependencies between words, making them effective for understanding complex language patterns.

By effectively capturing the nuances of text, the LSTM model contributes significantly to the overall accuracy of our AI vs. human-generated text classification system.

```
# Predict with LSTM model
def predict_lstm(model, tokenizer, encoder, text, max_length=20):
    sequence = tokenizer.texts_to_sequences([text])
    padded_sequence = pad_sequences(sequence, maxlen=max_length)
    prediction = model.predict(padded_sequence)
    return prediction[0][0]
```

Limitations & Things To Improve

Our model has some limitations;

- Because we trained our model on Wikipedia style writing, the system may disproportionately misclassify certain writing styles/patterns
- Classification for very short and very long texts remain difficult, as indicators may be spread too thin or not exist at all for shorter texts
- Ethical concerns: using such a system as basis for an accusation for misconduct happens frequently, when the model should be treated with some skepticism
- Because of the length of some Wikipedia articles (Minecraft's wikipedia article is over 600 sentences long), AI struggles to generate a fraction of the content. Because of this, we had to truncate a large amount of human-written data on topics.

Future improvements could be:

- Showing what parts of a text/input the model thinks is AI written (sentence by sentence rather than whole input)
- Increasing the type/styles of writing patterns we have in our dataset
- Adding a front end visualization of percentage AI use and hosting it as a website online

LIVE DEMO TIME