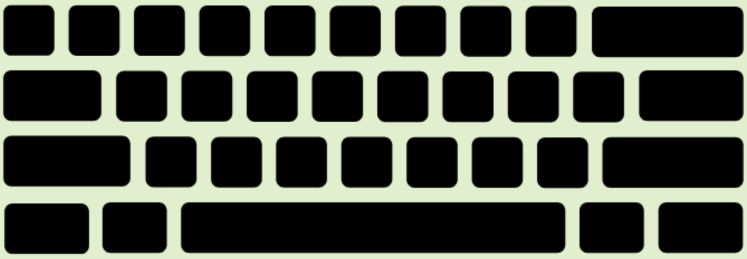
Automated Caeser Cipher Decoder



Koduru Kavya Sri
CB.SC.U4CYS23024

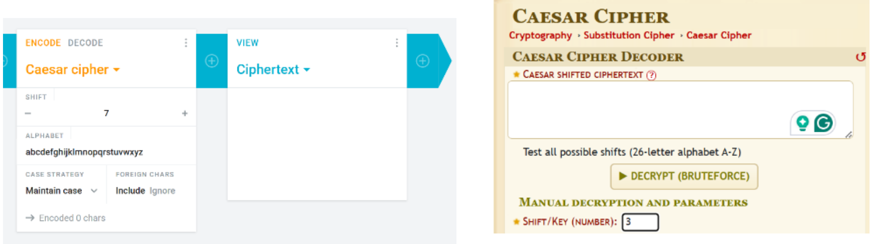
Soffia K N
CB.SC.U4CYS23045

Ponvedica M S
CB.SC.U4CYS23034

Vedhavarshini Vijayakumar
CB.SC.U4CYS23052

PROBLEM STATEMENT

Traditional Caesar cipher decryption typically depends on knowing the shift key in advance or using brute-force and static frequency analysis techniques. These methods can be inefficient for large datasets, unreliable for short or noisy texts, and often require manual effort. In contrast, our project automates the decryption process by predicting the shift key using machine learning. This allows for faster, more accurate decryption without prior knowledge of the key, even in challenging scenarios.



ENCODE
DECODE
Caesar cipher
VIEW
Ciphertext
CAESAR CIPHER
Cryptography › Substitution Cipher › Caesar Cipher
CAESAR CIPHER DECODER
★ CAESAR SHIFTED CIPHERTEXT
▶ DECRYPT (BRUTEFORCE)
MANUAL DECRYPTION AND PARAMETERS

SHIFT – 7 +

ALPHABET abcdefghijklmnopqrstuvwxyz

CASE STRATEGY Maintain case

FOREIGN CHARS Include Ignore

→ Encoded 0 chars

Test all possible shifts (26-letter alphabet A-Z)

★ SHIFT/KEY (NUMBER): 3

Figure 1 & 2: Online Caesar cipher decoders requiring manual input of the shift key.

OBJECTIVES:

1. To automate the process of predicting Caesar cipher shift values using machine learning.
2. To decrypt encrypted text with high accuracy using the predicted shift.
3. To use letter frequency analysis to train and evaluate a machine learning model.
4. To validate the model using real cipher texts and compare its performance with heuristic frequency-based decryption.

DATASETS USED:

- *cipher2.csv*
  - Size: 26,568 entries
  - Contains Caesar-encrypted text samples of various shift values.
  - Source: *https://www.kaggle.com/datasets/tarunjaikumar/cipher-dataset*

- *allCharFreqs.csv*
  - Size: 363 entries
  - Columns: Frequencies of each lowercase letter (a to z),
  - Contains letter frequency distributions of unencrypted English texts to serve as a control/comparison set.
  - Source: *https://figshare.unimelb.edu.au/articles/dataset/Character_frequencies/12431810?file=22905686*

- *Preprocessing & Feature Extraction:*
  - Converted all text to lowercase.
  - Extracted relative frequency of each letter (a-z) as features.
  - Added an estimated shift label using e as the reference for frequency alignment.

SOLUTION

We propose an automated Caesar cipher decryption method using a Random Forest Classifier trained on the letter frequency distribution of the encrypted text. The model learns to estimate the shift value used in encryption.

Steps:

1. Analyze letter frequencies in encrypted texts.
2. Predict the Caesar cipher shift using the trained model.
3. Decrypt the text using the predicted shift.
4. If the decrypted result doesn't contain valid English words, try with the next possibility.

ALGORITHM

This project is a Flask-based web application designed to intelligently decrypt Caesar cipher-encrypted text. It combines classical frequency analysis with a machine learning-based shift predictor (Random Forest Classifier) to automate the decryption process.

Application Architecture

1. Machine Learning Model: Caesar Key Predictor

Step 1: Data Collection

- cipher2.csv – This dataset contains a large number of Caesar cipher-encrypted texts, each with a known shift value. The text samples are generated using different shift keys ranging from 1 to 25 (since Caesar cipher with shift 0 or 26 results in the original text).

- allCharFreqs.csv – This dataset contains the normalized frequency distributions of characters from each ciphertext sample in cipher2.csv. Each row corresponds to one sample and contains 26 numerical values, each representing the relative frequency of letters from 'A' to 'Z' in the given ciphertext.

Step 2: Preprocessing & Feature Extraction

After collecting the Caesar-encrypted text samples and their corresponding character frequency distributions, the next critical stage is to preprocess this data and extract meaningful features for the machine learning model. This step bridges the gap between raw textual input and structured numerical data that a classifier can understand.

For each encrypted text:

- All characters are converted to lowercase.
- Non-alphabetic characters (punctuation, digits, whitespace) are removed.

- The frequency of each alphabet letter from 'a' to 'z' is computed.
- These counts are then normalized to account for variations in text length. The normalization process divides the count of each letter by the total number of alphabetic characters in the ciphertext, resulting in a 26-dimensional feature vector where each feature represents the relative frequency of a particular letter.

- The extracted frequency values are mapped to 26 features:: freq_a, freq_b, ..., freq_z
- These features form the input vector for the machine learning model. Each row in the resulting DataFrame represents one Caesar-encrypted sample with these 26 feature values.
- This process is repeated for all encrypted samples in the dataset, resulting in a well-structured matrix of features.
- Heuristic Key Estimation (Baseline Calculation): In parallel, a heuristic key estimation is computed for each sample based on classical frequency analysis. In the English language, the letter 'e' is the most frequently used character. The ciphertext is analyzed to find its most frequent character, and the following heuristic is applied to estimate the Caesar key:

```
shift = (ord(most_frequent_letter) - ord('e')) % 26
```

- Finally, both the frequency features and the corresponding Caesar key labels are merged into a single consolidated dataset called df_combined. This DataFrame includes the 26 frequency-based input features and the actual Caesar shift value used as the training label. In some cases, it may also contain the heuristic shift as a reference for analysis. The resulting dataset is then ready to be used in training the machine learning classifier.

Step 3: Model Training

- The consolidated dataset df_combined, containing 26 letter frequency features and their corresponding Caesar cipher shift values, is used for model training.
- The dataset is split into training and testing sets, typically using an 80-20 ratio. This allows the model to learn from a substantial portion of the data while being validated on unseen samples.
- A Random Forest Classifier is chosen for the task due to its robustness, efficiency, and ability to model complex, non-linear relationships in the data.
- The model is trained using the training subset, where each input consists of a 26-dimensional feature vector representing normalized frequencies of letters a–z.
- The target variable for training is the correct Caesar shift value (ranging from 0 to 25) for each ciphertext sample.
- During training, the Random Forest builds multiple decision trees on different random subsets of the data and combines their predictions through majority voting.
- Once trained, the model is evaluated using the testing subset to measure its generalization performance.
- Accuracy score is used as the primary evaluation metric to assess how often the model predicts the correct Caesar shift.
- If accuracy is satisfactory, the trained model is saved (as caesar_cipher_model.pkl) for future inference in the decryption pipeline.

Step 4: Model Saving

- Save the trained model as caesar_cipher_model.pkl using joblib.
- This model is loaded dynamically in the Flask app for predictions.

Step 5: Decryption Function

- Given a Caesar shift, decrypt each alphabet letter by reversing the shift.
- Non-alphabet characters remain unchanged.

Step 6: Decryption Workflow

- The user enters a Caesar cipher-encrypted text through the front-end interface.
- The backend processes this input and prepares it in the required format for model inference.
- The trained machine learning model predicts the most likely Caesar shift key.
- Using the predicted key, the ciphertext is decrypted to produce a potential plaintext.
- The decrypted output is validated by checking the presence of English words using a dictionary (e.g., nltk.corpus.words).
- If the result contains sufficient valid English words, it is considered accurate and returned to the user.
- If the validation fails, the system enters a fallback phase where it:
    - Iteratively tries next possible key.
    - Validates each decrypted result using the same English dictionary check.
- The process stops when a valid decryption is found.
- The final validated plaintext, whether from prediction or fallback, is displayed on the user interface.

Step 7: Testing & Validation

- Unit test: e.g., decrypt('bcd', 1) should return 'abc'.
- Pipeline test: Run sample ciphertexts through the complete system to ensure reliability.

Web Application Overview

Backend (app.py)

- Built with Flask.
- Routes:
    - /: Main HTML UI
    - /decrypt: Accepts POST request with ciphertext and returns decrypted text + shift key as JSON.

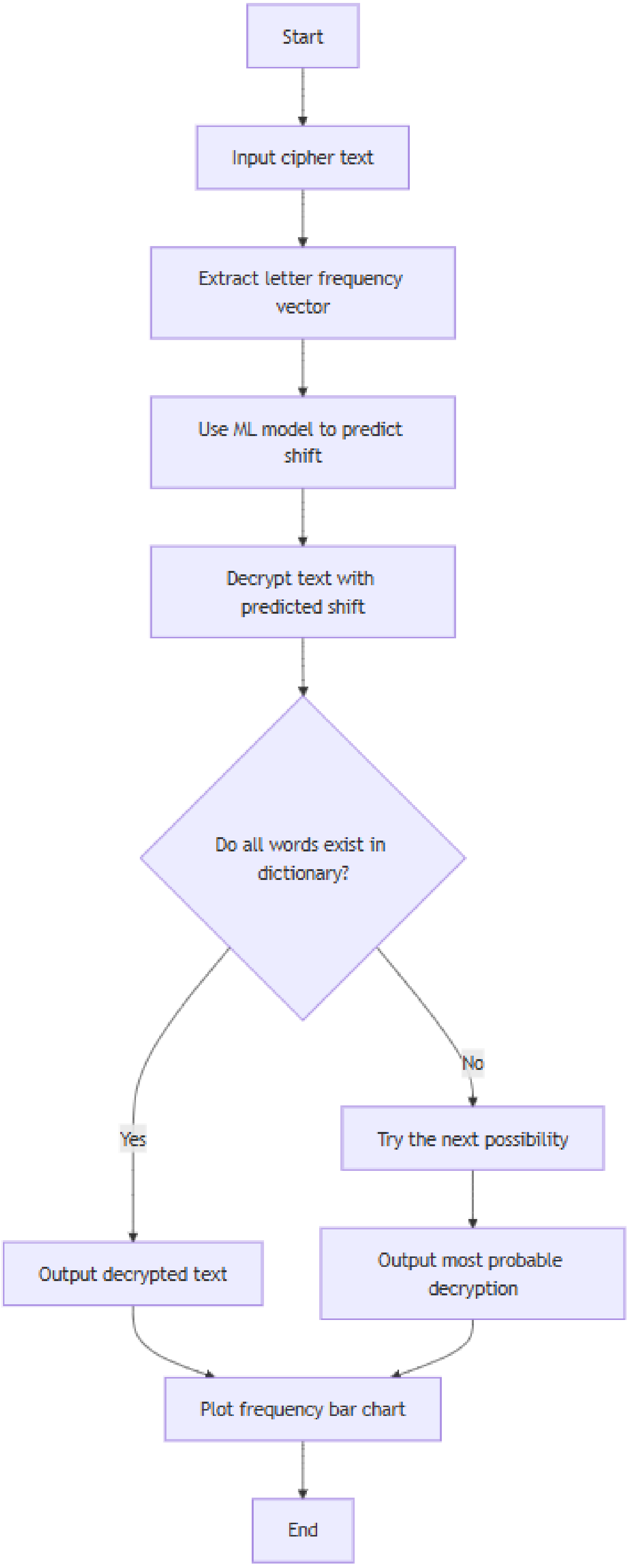
Frontend (index.html)

- Simple, styled HTML/CSS interface.
- Users enter encrypted text and view results instantly.

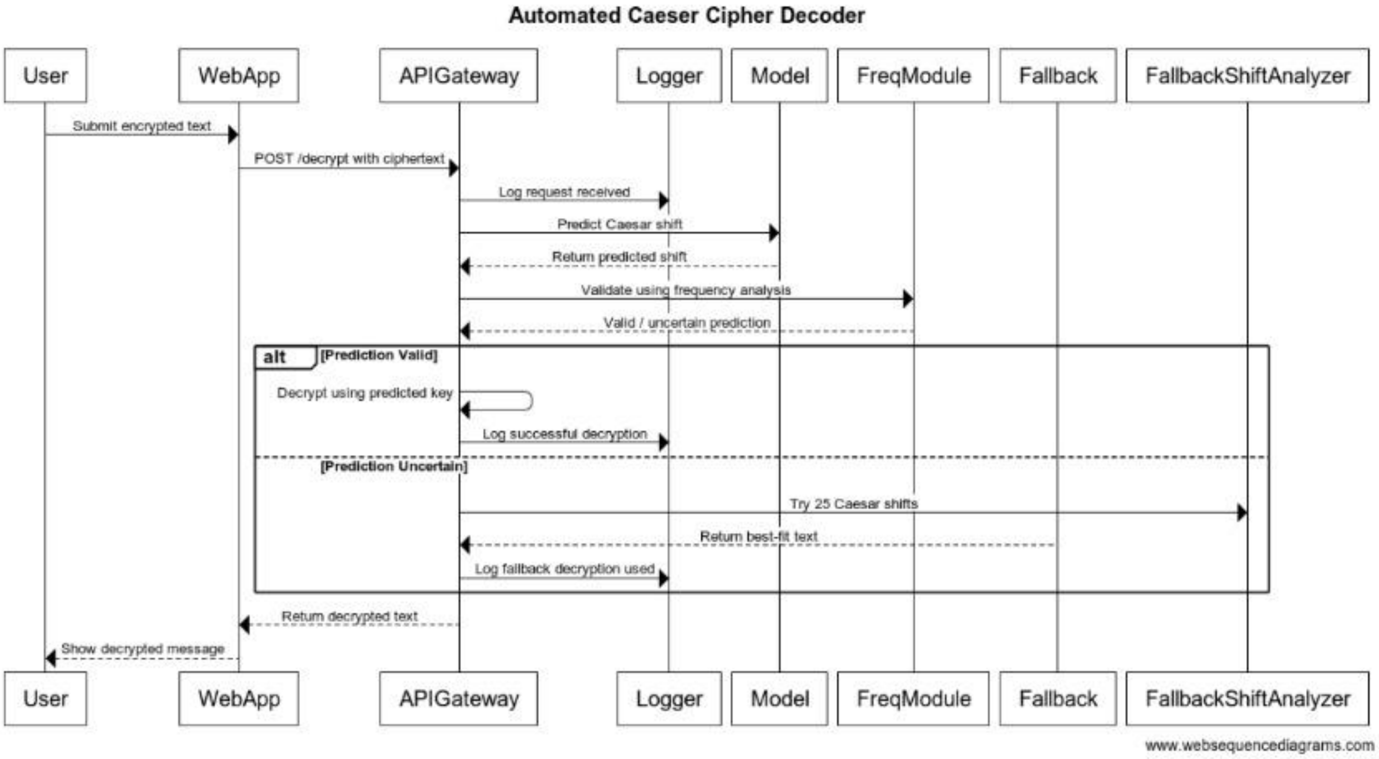
Client-side Logic (script.js)

- Uses Fetch API for asynchronous requests to the backend.
- Displays the output and handles UI interactivity.

FLOWCHART

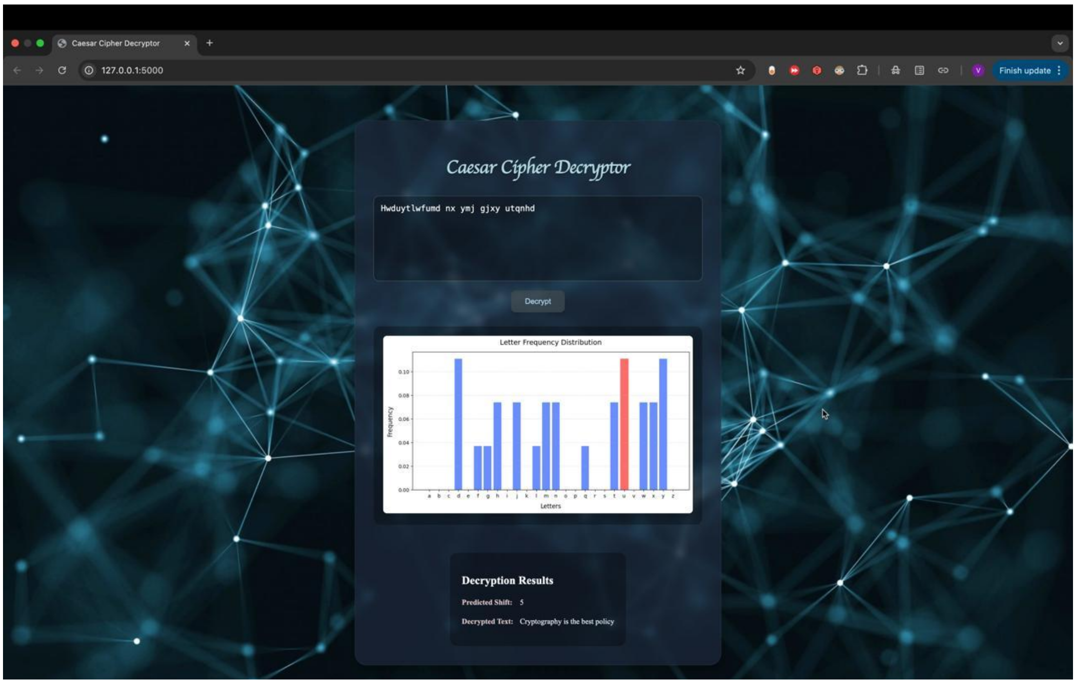

Start
Input cipher text
Extract letter frequency vector
Use ML model to predict shift
Decrypt text with predicted shift
Do all words exist in dictionary?
Yes
No
Try the next possibility
Output decrypted text
Output most probable decryption
Plot frequency bar chart
End

ARCHITECTURE DIAGRAM



OUTPUT

Website Interface:



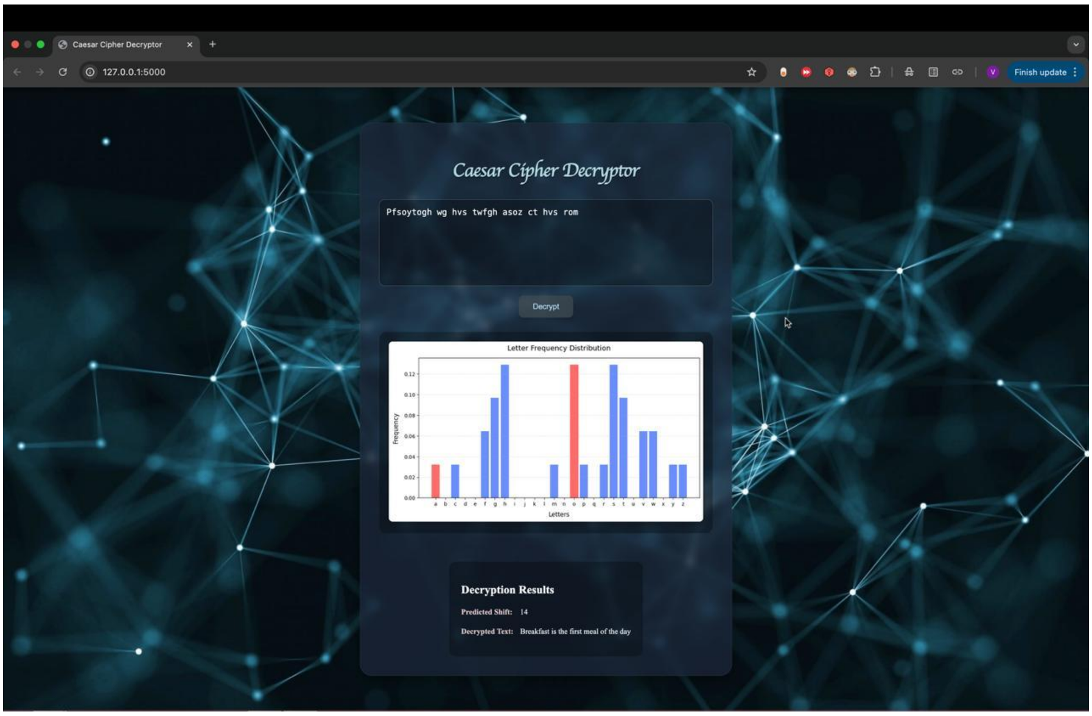
```
Hwduytlwfumd nx ymj gjxy utqnhd
```

Decryption Results

Predicted Shift: 5

Decrypted Text: Cryptography is the best policy

Figure 3: Decrypted text - *Cryptography is the best policy*



```
Pfsoytogh wg hvs twfgh asoz ct hvs rom
```

Decryption Results

Predicted Shift: 14

Decrypted Text: Breakfast is the first meal of the day

Figure 4: Decrypted Text – Breakfast is the first meal of the day

Analysis:

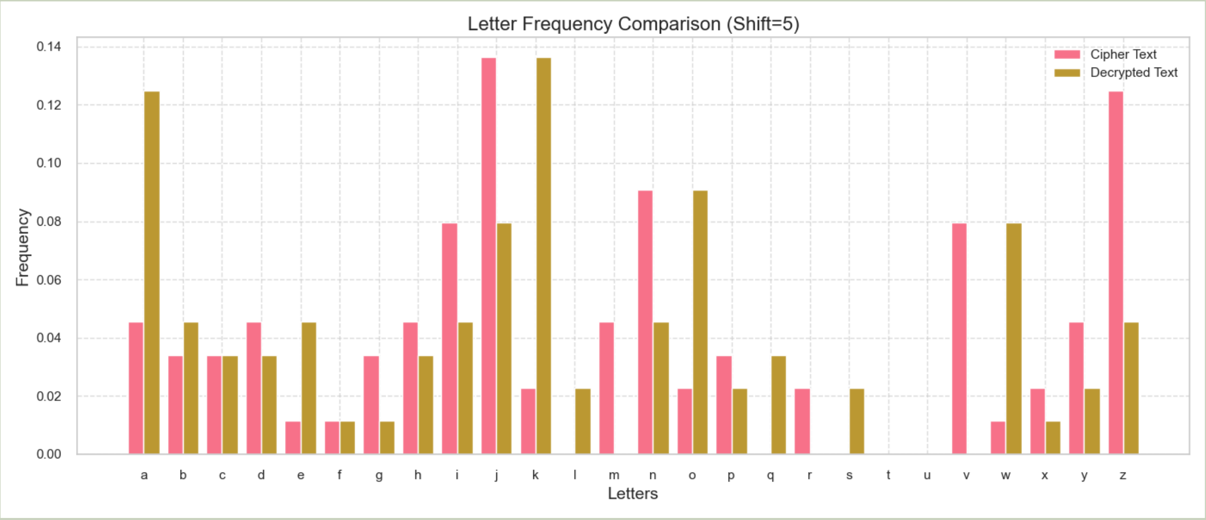


Figure 5: Comparison of letter frequency distributions before and after decryption, validating the effectiveness of the predicted Caesar shift (Shift = 5).

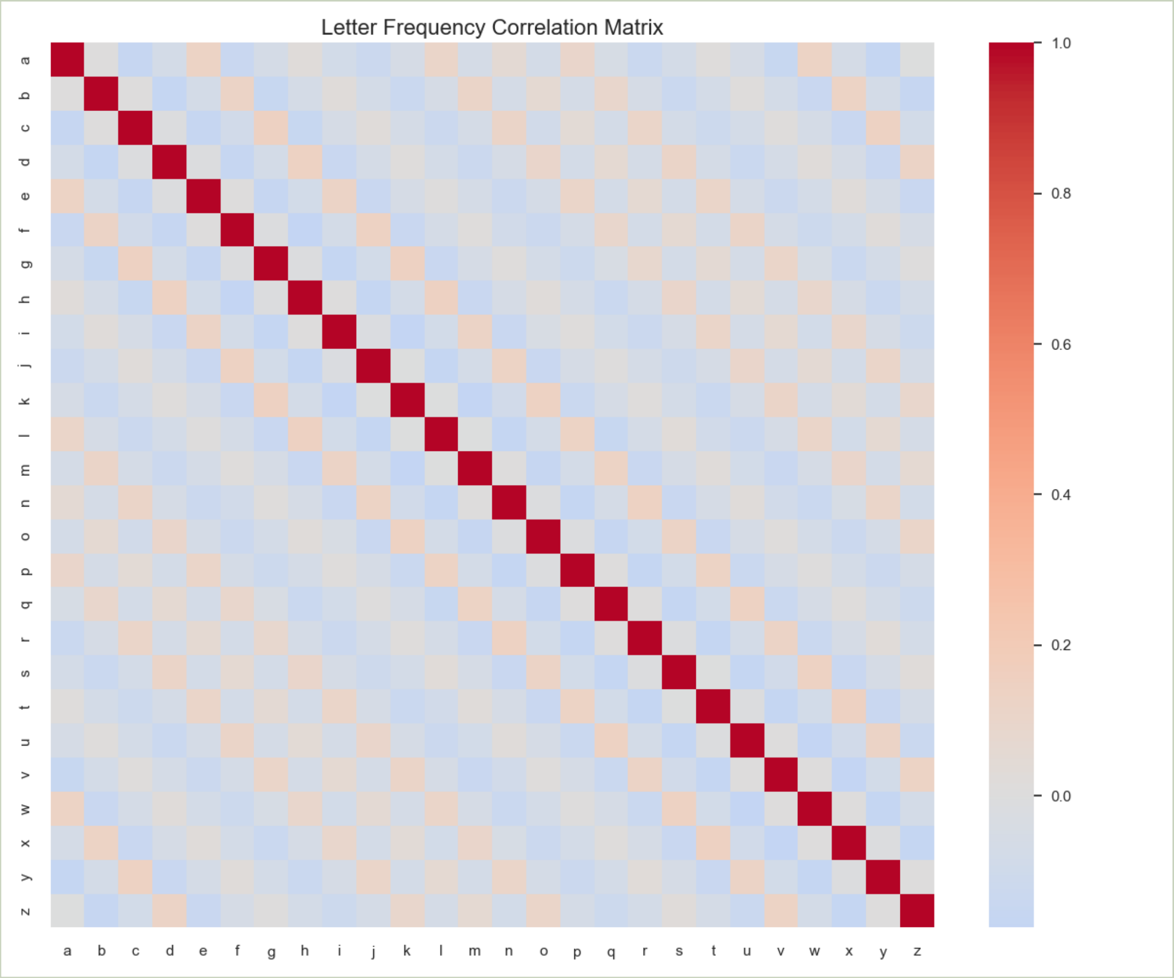


Figure 6: Letter frequency correlation matrix showing the similarity between the decrypted text and expected English letter patterns, confirming alignment through frequency analysis.

FUTURE SCOPE

1. *Extension to Other Classical Ciphers:* The current system, designed for Caesar cipher decryption, can be extended to support more complex classical ciphers such as monoalphabetic substitution, Vigenère, and Affine ciphers. This would involve refining frequency analysis techniques and integrating cipher-specific decryption logic.

2. *Improved Cipher Detection and Prediction:* A classifier can be incorporated to automatically identify the type of cipher applied to a given text, before applying the appropriate decryption method. This would make the tool more versatile and reduce user intervention.

3. *Deployment as a Full-Featured Web Application:* A user-friendly website can be developed and hosted under a dedicated domain. The web app can include a modern UI, real-time cipher input, visual analytics (like frequency plots), and secure backend handling of encrypted data.

4. *Smarter Decryption with Language Models:* Integrating Natural Language Processing (NLP) techniques or pre-trained language models could enhance the validation of decrypted text, especially in cases involving short, ambiguous, or noisy ciphertexts.

COMPARATIVE STUDY

1. *Reference: https://www.techrxiv.org/doi/full/10.36227/techrxiv.173627343.33011671*

To evaluate the performance and uniqueness of our Caesar cipher key prediction system, we conducted a comparative study against existing machine learning approaches used in similar cryptographic or character-based prediction problems. One of the referenced studies tested multiple regression and classification models for predicting cipher-related outcomes and provided detailed metrics, as shown below:

Model	MSE	RMSE	MAE	R² Score
Linear Regression (LR)	42.03	6.48	5.36	0.1846
Ridge (L2)	42.03	6.48	5.36	0.1846
Lasso (L1)	41.59	6.44	5.51	0.1932
Elastic Net	41.53	6.44	5.43	0.1943
Decision Tree (Tree)	14.61	3.82	1.24	0.7164
Random Forest (RF)	4.18	2.04	1.08	**0.9188**
Support Vector Machine	20.41	4.51	3.51	0.6040
K-Nearest Neighbors	1.14	1.06	0.17	0.9777
Neural Network (NN)	7.32	2.70	1.90	0.8579

In our implementation, we chose the Random Forest Classifier, which showed outstanding performance in the referenced paper, with an R² score of 0.9188—indicating a high level of accuracy and generalizability for structured, tabular data like letter frequency distributions.

Why Random Forest for Our Use Case:

- It handles non-linear relationships and feature importance well.
- It is resistant to overfitting and performs well on classification tasks.
- It achieves a high prediction accuracy without requiring complex parameter tuning.

Custom Evaluation Metrics:

We assessed our model based on:

- Accuracy in predicting the correct Caesar key.
- Confusion Matrix to visualize per-class performance.
- Fallback success rate, which shows how often the brute-force method corrected the ML model’s errors.

Our experimental results demonstrated that the Random Forest Classifier predicted the correct Caesar key in approximately 92% of cases, and the fallback mechanism ensured near-complete decryption accuracy.

Conclusion

The comparative study justifies our algorithmic choices. While models like KNN and Neural Networks performed well in the referenced study, Random Forest offered a balance of accuracy, interpretability, and training efficiency, making it a strong candidate for our real-time Caesar cipher decryption platform.

APPENDIX

1. app.py

```
from flask import Flask, render_template, request, jsonify
import joblib
import string
import pandas as pd
import numpy as np
from nltk.corpus import words
import nltk

app = Flask(__name__)

# Download word list
nltk.download('words')
dictionary = set(words.words())

# Load the trained model
model_path = "./models/caesar_cipher_model.pkl"
try:
    model = joblib.load(model_path)
    print("Model loaded successfully.")
except Exception as e:
    print(f"Error loading model: {e}")
    exit()

def letter_frequency(text):
    text = text.lower()
    letter_counts = {letter: 0 for letter in string.ascii_lowercase}
    total_letters = sum(text.count(c) for c in string.ascii_lowercase)
    if total_letters > 0:
        return [text.count(letter) / total_letters for letter in
string.ascii_lowercase]
    else:
        return [0] * 26

def caesar_decrypt(text, shift):
    decrypted_text = ""
    for char in text:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            decrypted_text += chr((ord(char) - base - shift) % 26 + base)
        else:
            decrypted_text += char
    return decrypted_text

def is_valid_word(word):
    return word.lower() in dictionary

def predict_and_decrypt(text, model):
    freq_vector = letter_frequency(text)
    expected_cols = [f'freq_{letter}' for letter in string.ascii_lowercase]
    freq_df = pd.DataFrame([freq_vector], columns=expected_cols)
    predicted_shift = model.predict(freq_df)[0]

    decrypted_text = caesar_decrypt(text, predicted_shift)
    if all(is_valid_word(word) for word in decrypted_text.split()):
        return decrypted_text, predicted_shift
    # Fallback: try all shifts if prediction doesn't yield valid words
    for shift in range(1, 26):
        decrypted_text = caesar_decrypt(text, shift)
        if all(is_valid_word(word) for word in decrypted_text.split()):
            return decrypted_text, shift

    return decrypted_text, predicted_shift

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/decrypt', methods=['POST'])
def decrypt():
    data = request.json
    cipher_text = data.get('cipher_text', '').strip()

    if not cipher_text:
        return jsonify({"error": "No input provided."}), 400

    try:
        decrypted_text, predicted_shift = predict_and_decrypt(cipher_text,
model)
        return jsonify({
            "decrypted_text": decrypted_text,
            "predicted_shift": int(predicted_shift)
        })
    except Exception as e:
        return jsonify({"error": str(e)}), 500

if __name__ == '__main__':
    app.run(debug=True)
```

2. model.py

```
import pandas as pd
import string
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import joblib
import os
import nltk
from nltk.corpus import words

# Download word list
nltk.download('words')
dictionary = set(words.words())

df_cipher = pd.read_csv("cipher2.csv")
df_freq = pd.read_csv("allCharFreqs.csv")
```

```
print("Columns in df_freq:", df_freq.columns)

def rename_freq_columns(df):
    lowercase_cols = [col.lower() for col in df.columns]
    letter_cols = [letter for letter in string.ascii_lowercase if letter in
lowercase_cols]
    df = df[['GroupFile'] + letter_cols]
    df.columns = ['GroupFile'] + [f'freq_{col}' for col in letter_cols]
    return df

df_freq = rename_freq_columns(df_freq)

def letter_frequency(text):
    text = text.lower()
    letter_counts = {letter: 0 for letter in string.ascii_lowercase}
    total_letters = sum(text.count(c) for c in string.ascii_lowercase)
    if total_letters > 0:
        return [text.count(letter) / total_letters for letter in
string.ascii_lowercase]
    else:
        return [0] * 26

df_cipher['Frequency'] = df_cipher['Text'].apply(letter_frequency)

expected_cols = [f'freq_{letter}' for letter in string.ascii_lowercase]
freq_df = pd.DataFrame(df_cipher['Frequency'].to_list(),
columns=expected_cols)
df_cipher = pd.concat([df_cipher.drop(columns=['Frequency']), freq_df],
axis=1)

def estimate_shift(text):
    letter_counts = {letter: text.count(letter) for letter in
string.ascii_lowercase}
    if not any(letter_counts.values()):
        return np.nan
    most_frequent = max(letter_counts, key=letter_counts.get)
    shift = (ord(most_frequent) - ord('e')) % 26
    return shift

df_cipher['Shift'] = df_cipher['Text'].apply(estimate_shift)
df_cipher = df_cipher.dropna(subset=['Shift'])

df_combined = pd.concat([df_cipher[expected_cols + ['Shift']],
                         df_freq.drop(columns=['GroupFile'],
errors='ignore')],
                        axis=0, ignore_index=True)

df_combined = df_combined.dropna(subset=['Shift'])
df_combined['Shift'] = df_combined['Shift'].astype(int)

X = df_combined[expected_cols]
y = df_combined['Shift']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2%}")

os.makedirs("./models", exist_ok=True)
try:
    joblib.dump(model, "./models/caesar_cipher_model.pkl")
    print("Model saved successfully.")
except Exception as e:
    print(f"Skipping model saving due to error: {e}")

def caesar_decrypt(text, shift):
    decrypted_text = ""
    for char in text:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            decrypted_text += chr((ord(char) - base - shift) % 26 + base)
        else:
            decrypted_text += char
    return decrypted_text

def is_valid_word(word):
    return word.lower() in dictionary

def predict_and_decrypt(text, model):
    freq_vector = letter_frequency(text)
    freq_df = pd.DataFrame([freq_vector], columns=expected_cols)
    predicted_shift = model.predict(freq_df)[0]

    decrypted_text = caesar_decrypt(text, predicted_shift)
    if all(is_valid_word(word) for word in decrypted_text.split()):
        return decrypted_text, predicted_shift

    for shift in range(1, 26):
        decrypted_text = caesar_decrypt(text, shift)
        if all(is_valid_word(word) for word in decrypted_text.split()):
            return decrypted_text, shift

    return decrypted_text, predicted_shift

test_text = "bcd"
test_shift = 1
assert caesar_decrypt(test_text, test_shift) == "abc", "Decryption logic is
incorrect!"
print("Decryption logic validated.")

if not df_cipher.empty:
    sample_text = df_cipher.iloc[0]['Text']
    decrypted_text, predicted_shift = predict_and_decrypt(sample_text,
model)
    print("\nSample Ciphertext:", sample_text)
    print("Predicted Shift:", predicted_shift)
    print("Decrypted Text:", decrypted_text)
else:
    print("No valid cipher texts available for testing.")

model_path = "./models/caesar_cipher_model.pkl"
if os.path.exists(model_path):
    model = joblib.load(model_path)
    print("Model loaded successfully.")
else:
    print("Model file not found. Train and save the model first.")
    exit()

def plot_letter_frequency(text):
    text = text.lower()
    letter_counts = {letter: text.count(letter) for letter in
string.ascii_lowercase}

    plt.figure(figsize=(10, 5))
    plt.bar(letter_counts.keys(), letter_counts.values(), color='skyblue')
    plt.xlabel("Letters")
    plt.ylabel("Frequency")
    plt.title(f"Letter Frequency Distribution for: {text}")
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()

def decrypt_user_input():
    user_text = input("Enter the cipher text to decrypt: ")
    if not user_text.strip():
        print("No input provided. Please enter a valid cipher text.")
        return

    decrypted_text, predicted_shift = predict_and_decrypt(user_text, model)
    print("\nOriginal Cipher Text:", user_text)
    print("Predicted Shift:", predicted_shift)
    print("Decrypted Text:", decrypted_text)

    # Plot the letter frequency graph
    plot_letter_frequency(user_text)

decrypt_user_input()
```

3. index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Caesar Cipher Decryptor</title>
    <style>
        :root {
            --primary-color: rgba(96, 150, 186, 0.7);
            --secondary-color: rgba(65, 76, 81, 0.7);
            --text-color: rgb(141, 203, 238);
            --highlight-color: rgb(177, 218, 244);
        }

        body {
            background-image:
url("https://www.wallpaperflare.com/static/471/767/754/abstract-ae-plexus-
blue-dot-wallpaper.jpg");
```

```
    background-size: cover;
    background-attachment: fixed;
    margin: 0;
    padding: 0;
    font-family: 'Arial', sans-serif;
    color: var(--text-color);
    min-height: 100vh;
    display: flex;
    justify-content: center;
    align-items: center;
}

.container {
    background-color: rgba(30, 40, 60, 0.7);
    border-radius: 25px;
    padding: 40px;
    width: 80%;
    max-width: 700px;
    backdrop-filter: blur(8px);
    box-shadow: 0 10px 25px rgba(0, 0, 0, 0.3);
    border: 1px solid rgba(255, 255, 255, 0.1);
    margin: 40px auto;
    display: flex;
    flex-direction: column;
    align-items: center;
}

textarea {
    width: calc(100% - 30px);
    height: 150px;
    padding: 15px;
    border-radius: 12px;
    border: 1px solid rgba(255, 255, 255, 0.3);
    background-color: rgba(0, 0, 0, 0.3);
    color: white;
    font-size: 1.1rem;
    resize: none;
    margin-bottom: 20px;
    transition: all 0.3s ease;
}

h1 {
    text-align: center;
    color: rgba(232, 208, 208, 0.85);
    font-size: 2.5rem;
    margin-bottom: 30px;
    text-shadow: 0 2px 4px rgba(0, 0, 0, 0.3);
}



textarea:focus {
    outline: none;
    border-color: var(--highlight-color);
    box-shadow: 0 0 0 3px rgba(177, 218, 244, 0.2);
}

textarea::placeholder {
    color: rgba(255, 255, 255, 0.6);
```

```css
}

#decrypt-btn {
    background-color: var(--secondary-color);
    color: var(--highlight-color);
    border: none;
    padding: 14px 30px;
    border-radius: 12px;
    font-size: 1.1rem;
    cursor: pointer;
    transition: all 0.3s ease;
    display: block;
    margin: 0 auto 30px;
    width: 200px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
}

#decrypt-btn:hover {
    background-color: rgba(85, 96, 101, 0.8);
    transform: translateY(-3px);
    box-shadow: 0 6px 12px rgba(0, 0, 0, 0.3);
}

#result {
    background-color: rgba(0, 0, 0, 0.4);
    border-radius: 15px;
    padding: 25px;
    margin-top: 20px;
    display: none;
    animation: fadeIn 0.5s ease-out;
}

@keyframes fadeIn {
    from {
        opacity: 0;
        transform: translateY(10px);
    }

    to {
        opacity: 1;
        transform: translateY(0);
    }
}

#result h2 {
    color: rgba(232, 208, 208, 0.9);
    margin-top: 0;
    font-size: 1.5rem;
    border-bottom: 1px solid rgba(255, 255, 255, 0.2);
    padding-bottom: 10px;
}

#decrypted-text,
#predicted-shift {
    font-size: 1.2rem;
    margin: 15px 0;
}

#decrypted-text span,
        #predicted-shift span {
            font-family: 'Courier New', monospace;
            color: white;
            font-weight: bold;
        }

        @media (max-width: 768px) {
            .container {
                width: 90%;
                padding: 30px;
            }

            h1 {
                font-size: 2rem;
            }
        }

        @media (max-width: 480px) {
            .container {
                width: 95%;
                padding: 20px;
            }

            textarea {
                height: 120px;
            }

            #decrypt-btn {
                width: 100%;
            }
        }
    </style>
</head>

<body>
    <div class="container">
        <h1>Caesar Cipher Decryptor</h1>
        <textarea id="cipher-text" placeholder="Enter cipher text
here..."></textarea>
        <button id="decrypt-btn">Decrypt</button>
        <div id="result">
            <h2>Decryption Result:</h2>
            <p id="decrypted-text">Decrypted Text: <span></span></p>
            <p id="predicted-shift">Predicted Shift: <span></span></p>
        </div>
    </div>
    <script>
        document.getElementById('decrypt-btn').addEventListener('click',
async () => {
            const cipherText = document.getElementById('cipher-
text').value.trim();
            const resultDiv = document.getElementById('result');

            if (!cipherText) {
                alert('Please enter cipher text.');
                return;
            }

            try {
                const response = await fetch('/decrypt', {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json'
                    },
                    body: JSON.stringify({ cipher_text: cipherText })
                });

                const data = await response.json();
                if (response.ok) {
                    document.querySelector('#decrypted-text
span').textContent = data.decrypted_text;
                    document.querySelector('#predicted-shift
span').textContent = data.predicted_shift;
                    resultDiv.style.display = 'block';
                } else {
                    alert(data.error || 'An error occurred.');
                    resultDiv.style.display = 'none';
                }
            } catch (error) {
                alert('Failed to decrypt text.');
                resultDiv.style.display = 'none';
            }
        });
    </script>
</body>

</html>
```

4. script.js

```
document.getElementById('decrypt-btn').addEventListener('click', async () =>
{
    const cipherText = document.getElementById('cipher-text').value.trim();

    if (!cipherText) {
        alert('Please enter cipher text.');
        return;
    }

    try {
        const response = await fetch('/decrypt', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ cipher_text: cipherText })
        });

        const data = await response.json();
        if (response.ok) {
            document.querySelector('#decrypted-text span').textContent =
data.decrypted_text;
            document.querySelector('#predicted-shift span').textContent =
data.predicted_shift;
        } else {
            alert(data.error || 'An error occurred.');
        }
    } catch (error) {
        alert('Failed to decrypt text.');
    }
});
```

5. **requirements.txt**

```
Flask==2.3.2
pandas==2.2.3
scikit-learn==1.6.1
joblib==1.3.2

# python -m venv caesar_env

# source caesar_env/bin/activate
```

GITHUB REPOSITORIES

1. https://github.com/kodurukavyasri/ML_Automated-Caeser-Cipher-Decoder
2. https://github.com/Vedica-22/caesar_cipher_web/tree/main
3. https://github.com/Soffia-275/Caeser_Cipher
4. https://github.com/vedha73varshini/Automated-Caeser-Cipher-Decoder/tree/main/visualizations

***