

IndiaAI CyberGuard AI Hackathon – Stage 1 Submission

Project Title

NLP-based Complaint Classification System for Cybercrime Reporting

Team Details

- **Team Name:** TechTitan
 - **Team Member(s):** Pawan Srivastava
 - **Contact Information:**
 - **Email:** srivastavapawan103@gmail.com
 - **Phone:** +91 8856925900
-

Project Name :

CyberAid AI: Multilingual Complaint Classification and Guidance System

Presented to

**IndiaAI CyberGuard AI Hackathon
Ministry of Electronics and Information Technology (MeitY),
Government of India**

Submission Date

21/10/2024

1. Introduction

The **IndiaAI CyberGuard AI Hackathon** is a strategic initiative aimed at enhancing cybersecurity through AI-driven solutions. Our project leverages **Natural Language Processing (NLP)** and machine learning to automate the classification of cybercrime complaints, providing a scalable solution for the **National Cyber Crime Reporting Portal (NCRP)**. With thousands of complaints being submitted daily, a manual system would be inefficient and slow. The goal of this project is to automate complaint categorization and offer users tailored advice based on complaint details.

This report discusses the methodology, implementation, and results of the developed **NLP model**, which uses **FastText embeddings**, a **Feed-Forward Neural Network (FFNN)**, and a **Flask-based web interface**. Additionally, it outlines the findings from our analysis, model evaluation metrics, and implementation plan for further development and deployment.

2. Problem Understanding

2.1 Context and Objective

With cybercrimes on the rise, it is crucial to develop automated systems to handle complaints efficiently. The primary objective of this project is to develop an **NLP-based classification system** that categorizes complaints into predefined **categories** and **subcategories**, and provides users with **actionable advice** based on the classification.

2.2 Key Challenges

- **Ambiguity in User Input:** Complaints often contain unstructured and ambiguous text, making it difficult to accurately classify.
- **Data Imbalance:** Some categories have significantly more data than others, which could lead to model bias.
- **Scalability:** The model must handle a large number of daily complaints while maintaining high accuracy and speed.

3. Methodology

3.1 Data Processing

1. Dataset Overview:

The datasets used for this project are train.csv and test.csv, containing the following key columns:

- **CrimeAdditionalInfo:** The textual complaint data submitted by users.
- **Category:** A broad classification of the complaint (e.g., Financial Fraud, Cyber Terrorism).
- **Subcategory:** A specific classification within the main category (e.g., Phishing, Malware).

2. Text Preprocessing:

- **Null Value Handling:** Missing values in CrimeAdditionalInfo were replaced with the placeholder "**unknown**".
- **Cleaning:** The text was cleaned by removing special characters, punctuation, and numbers using regular expressions.
- **Tokenization & Lemmatization:** Text was tokenized into words and lemmatized to standardize forms (e.g., "running" to "run").
- **Stopword Removal:** Common stopwords (e.g., "the", "is", "on") were removed to reduce noise.

3. Feature Engineering:

- **FastText Embeddings:** We trained a **FastText model** on the complaint data to capture semantic meaning in the text. The model generates embeddings that represent each complaint as a vector in a continuous vector space.

3.2 Model Development

1. Feed-Forward Neural Network (FFNN):

- **Input Layer:** FastText embeddings as input features.
- **Hidden Layers:**
 - Multiple Dense layers (ReLU activation), BatchNormalization, and Dropout for regularization to prevent overfitting.
 - Three Dense layers with ReLU activation, BatchNormalization, and Dropout to prevent overfitting.
- **Output Layers:**
 - **Category Output:** Softmax activation for multi-class classification.

- **Subcategory Output:** Softmax activation for multi-class classification.
 - **Loss Function:** **Sparse Categorical Crossentropy** for both category and subcategory outputs.
 - **Optimizer:** **Adam** with a learning rate of 0.001, adjusted with learning rate reduction on plateau.
 - **Early Stopping:** To avoid overfitting, early stopping was used to halt training when validation loss stopped improving.
2. **Training and Evaluation:**
- Split the dataset into **80% training**, **10% validation**, and **10% testing**.
 - We utilized **accuracy**, **precision**, **recall**, and **F1-score** as evaluation metrics.

3.3 Web Application

1. **Flask Framework:**
 - Developed a web application to handle user inputs, classify complaints, and display predictions.
2. **Multilingual Interface:**
 - Used the Google Translate API to provide outputs in the user's input language.

4. Results and Observations

4.1 Model Evaluation

- **Category Classification:**
 - **Accuracy: 75.35%**
 - **F1-Score: 0.7084** (Weighted Average)
- **Subcategory Classification:**
 - **Accuracy: 54.83%**
 - **F1-Score: 0.5177** (Weighted Average)

4.2 Significant Findings

1. **Key Drivers of Correct Predictions:**
 - Complaints that contained clear, specific details resulted in higher classification accuracy.
 - **Financial fraud** complaints (such as credit card fraud and phishing) were more accurately classified due to the high volume of training data for these types.
2. **Challenges / Misclassifications:**
 - Ambiguous descriptions (e.g., "online fraud" vs. "online scam") led to incorrect predictions.
 - Subcategories with fewer instances, like **Cyber Terrorism**, showed lower classification accuracy due to data imbalance.

4.3 Text Classification Accuracy

- **Key Drivers:**
 - Accurate classification was driven by the quality and specificity of complaint descriptions.
 - Complaints with generic or ambiguous content were often misclassified.
- **Incorrect Predictions:**
 - Complaints that had overlapping subcategories (e.g., **Phishing** vs **Business Email Compromise**) showed confusion in classification, especially where descriptions lacked detailed context.

5. Design Methodology

1. Text Processing Pipeline:

- The entire pipeline was designed to ensure that the model could handle a variety of complaint formats. The preprocessing steps, including tokenization, stopword removal, and lemmatization, were essential in reducing the noise from the text and extracting meaningful features.

2. FastText Embeddings:

- We chose **FastText** for word embeddings because it captures subword information, making it robust to rare words and misspellings, a common issue in user-submitted complaints.

3. Multi-task Learning:

- The FFNN was designed to solve a multi-task problem, where we simultaneously predict both the **category** and **subcategory** of each complaint. This approach allows for better generalization and makes the model more adaptable to a wide range of complaints.

4. Regularization Techniques:

- **BatchNormalization** and **Dropout** were applied to the hidden layers to reduce overfitting and improve the model's robustness to unseen data.

6. Significant Findings from NLP Analysis (500 words)

6.1 Commonly Recurring Themes/Topics

Through our analysis of the complaints, several recurring themes were identified:

- **Financial Frauds:** The most frequent complaints involved various forms of financial fraud, such as credit card fraud, phishing, and email scams. This aligns with the broader trend of cybercrime affecting individuals' financial security.
- **Hacking and Malware:** Complaints related to hacking (e.g., unauthorized access to accounts or data breaches) were also common, particularly in business settings.
- **Cyber Terrorism:** Although less frequent, complaints in this category were significant due to their high severity. They often involved attempts to disrupt critical infrastructure or steal sensitive government data.

6.2 Sentiment Analysis

While sentiment analysis was not the focus of this project, future iterations of this system could include sentiment analysis to better understand the urgency or severity of complaints based on the emotional tone of the text.

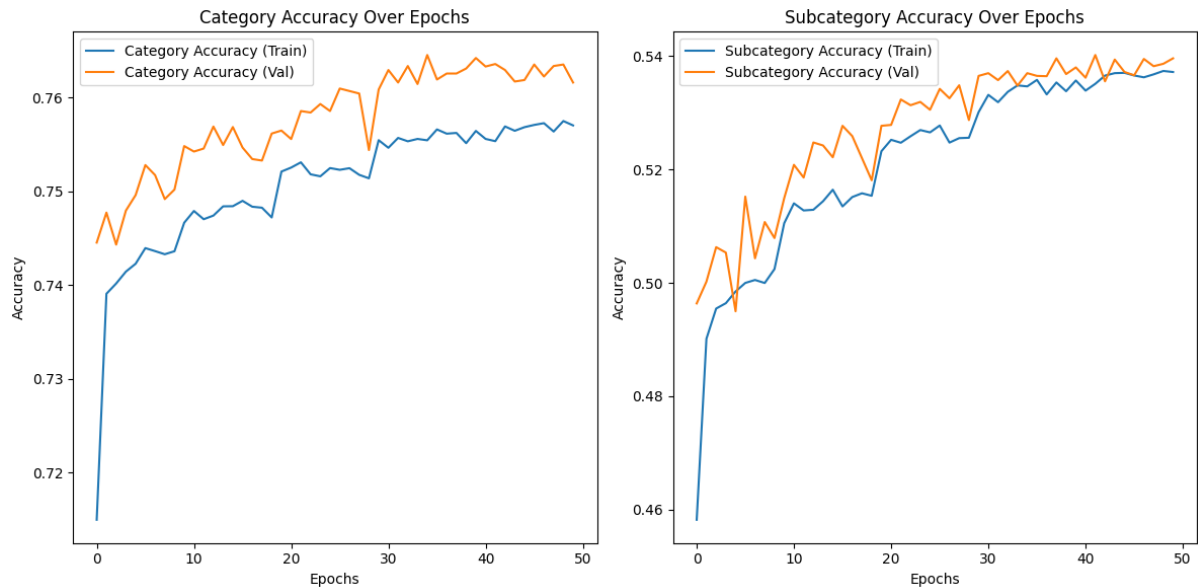
6.3 Visualization

We created **confusion matrices** to visualize misclassifications between categories and subcategories. These matrices revealed:

- High misclassification between **Phishing** and **Business Email Compromise** due to their similarity in attack methods.
- Lower accuracy in rare subcategories, such as **Cyber Terrorism**, suggesting the need for more training data.

Visualisation :

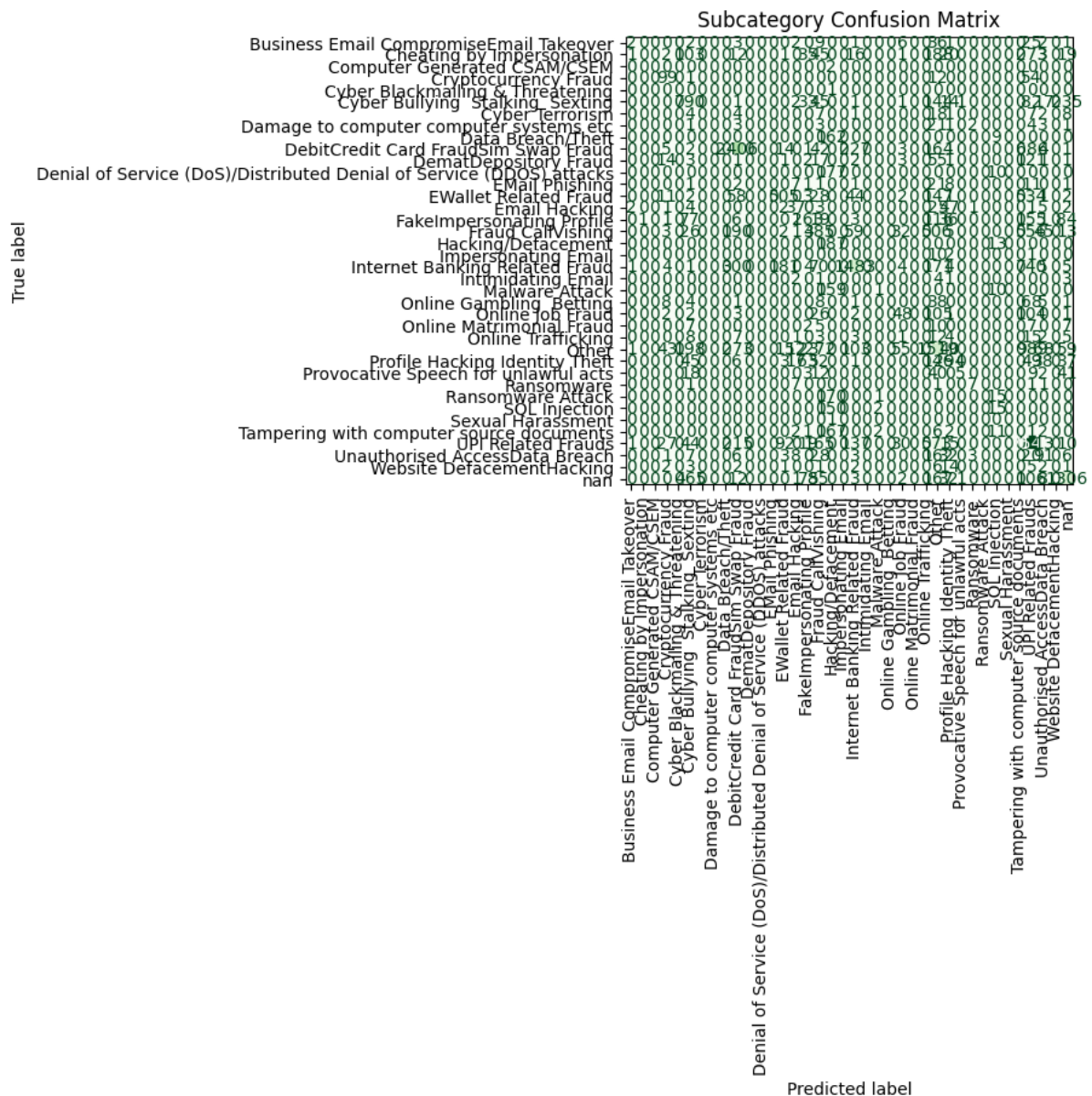
1. Training and Validation Accuracy Plots



2. Confusion Matrix for Category :

		Category Confusion Matrix															
True label	Any Other Cyber Crime	763	2	0	22	0	0	90	0	23	58	0	435	0	0	0	0
	Child Pornography CPChild Sexual Abuse Material CSAM	3	10	0	0	0	0	0	0	21	0	68	0	0	0	21	
	Crime Against Women & Children	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	
	Cryptocurrency Crime	0	0	0	67	0	0	0	0	98	0	1	0	0	0	0	
	Cyber Attack/ Dependent Crimes	0	0	0	0	126	1	0	0	0	0	0	0	0	0	0	
	Cyber Terrorism	13	0	0	0	0	0	2	0	21	0	16	0	0	0	0	
	Hacking Damage to computercomputer system etc	105	0	2	0	0	153	0	163	0	168	1	0	0	0	0	
	Online Cyber Trafficking	5	0	0	0	0	0	3	0	34	0	19	0	0	0	0	
	Online Financial Fraud	479	0	0	26	0	0	77	0	803	0	280	0	0	0	0	
	Online Gambling Betting	12	0	0	2	0	0	4	0	106	0	10	0	0	0	0	
	Online and Social Media Related Crime	280	2	0	0	0	0	86	0	1325	0	2433	0	0	0	13	
	Ransomware	1	0	0	0	0	0	11	0	1	0	2	3	0	0	0	
	RapeGang Rape RGRSexually Abusive Content	6	5	0	0	0	0	0	0	11	0	59	0	825	0	6	
	Sexually Explicit Act	34	5	0	0	0	0	7	0	102	0	378	0	0	0	9	
	Sexually Obscene material	36	4	0	0	0	0	4	0	78	0	528	0	0	0	16	
		Predicted label															
		Any Other Cyber Crime	Child Pornography CPChild Sexual Abuse Material CSAM	Crime Against Women & Children	Cryptocurrency Crime	Cyber Attack/ Dependent Crimes	Cyber Terrorism	Hacking Damage to computercomputer system etc	Online Cyber Trafficking	Online Financial Fraud	Online Gambling Betting	Online and Social Media Related Crime	Ransomware	RapeGang Rape RGRSexually Abusive Content	Sexually Explicit Act	Sexually Obscene material	

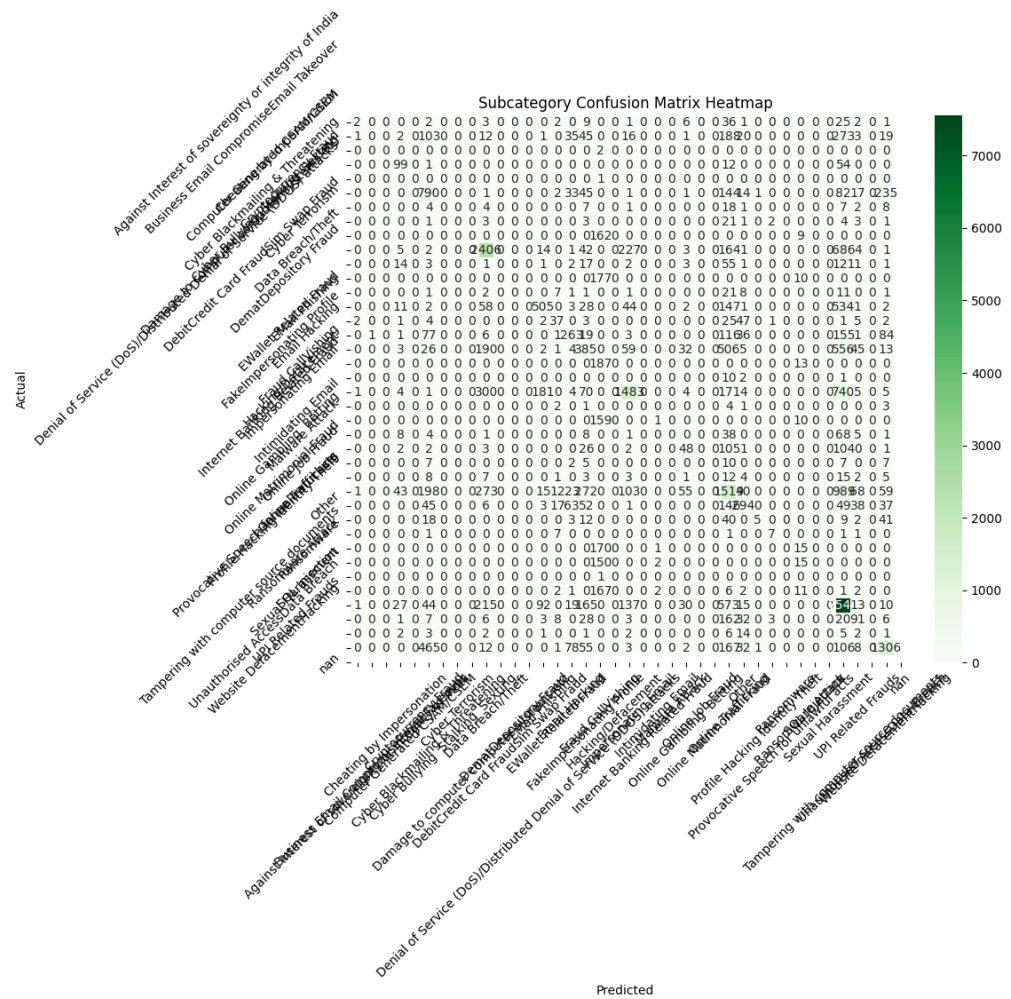
3. Confusion Matrix for Sub - Category :



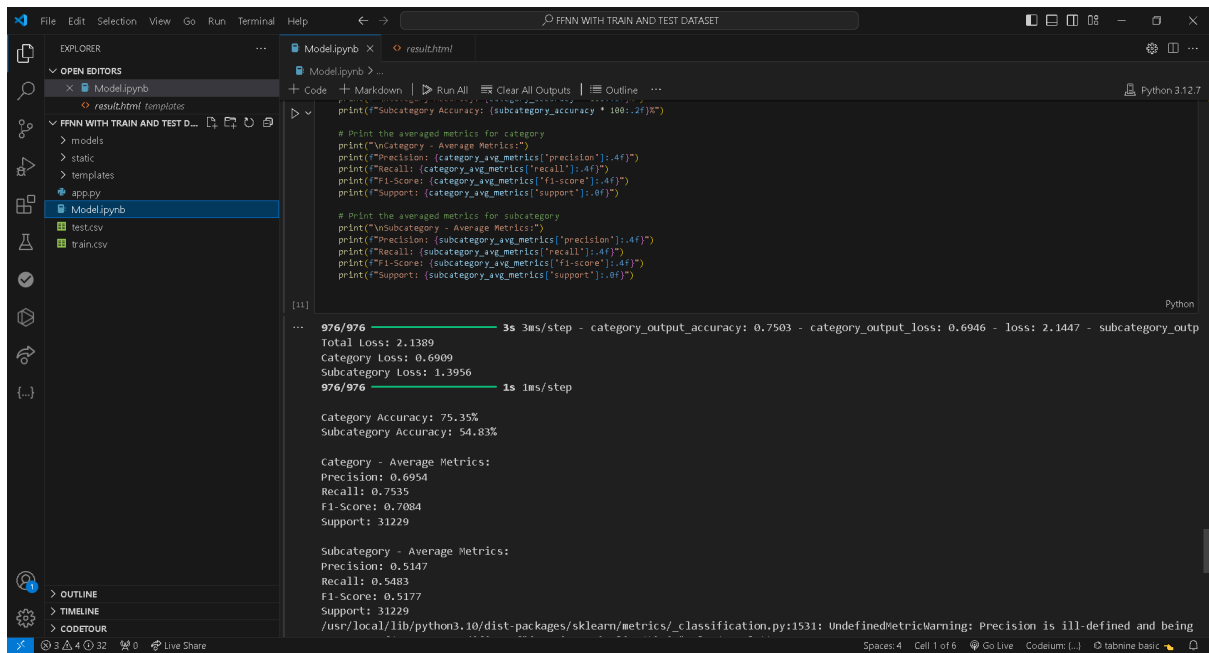
4. HeatMap for Catoegry :



5. HeatMap for Sub – Category :



Screenshots and Code



The screenshot shows a Jupyter Notebook interface with the following content:

```
Model.ipynb X result.html
Model.ipynb > ...
+ Code + Markdown | ▶ Run All | Clear All Outputs | Outline ...
Python 3.12.7

# Print the averaged metrics for category
print("\nCategory - Average Metrics:")
print("Precision: (category_avg_metrics['precision']:.4f)")
print("Recall: (category_avg_metrics['recall']:.4f)")
print("F1-Score: (category_avg_metrics['f1-score']:.4f)")
print("Support: (category_avg_metrics['support']:.4f)")

# Print the averaged metrics for subcategory
print("\nSubcategory - Average Metrics:")
print("Precision: (subcategory_avg_metrics['precision']:.4f)")
print("Recall: (subcategory_avg_metrics['recall']:.4f)")
print("F1-Score: (subcategory_avg_metrics['f1-score']:.4f)")
print("Support: (subcategory_avg_metrics['support']:.4f)")

[11]

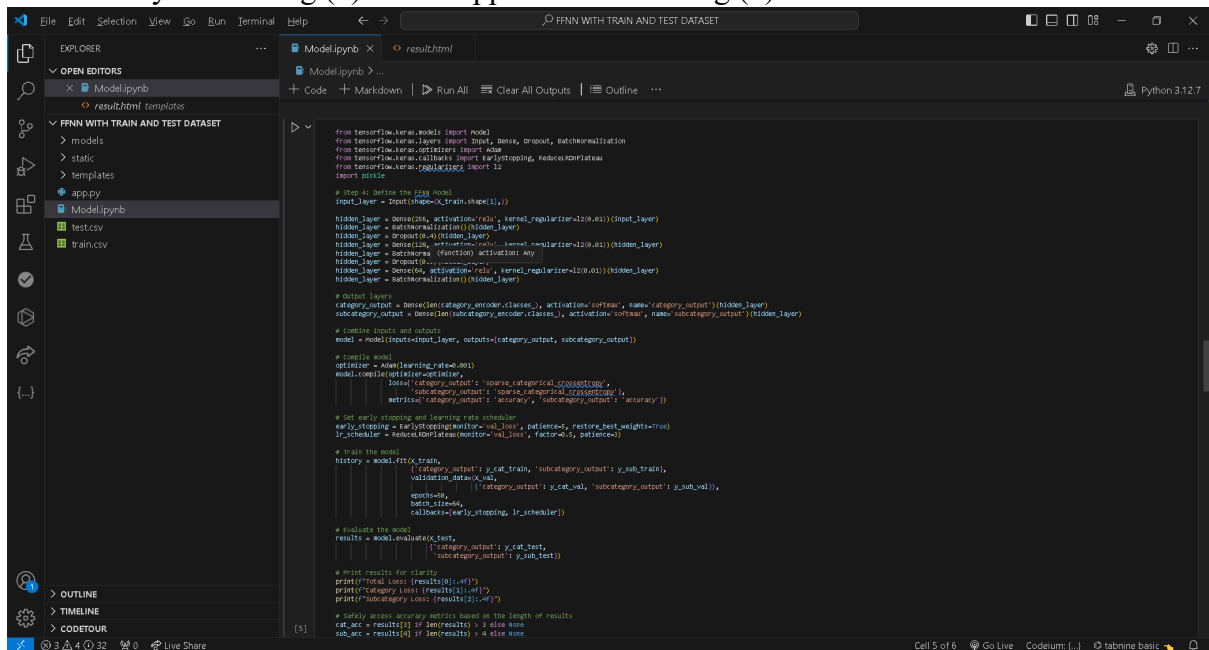
... 976/976 3s 3ms/step - category_output_accuracy: 0.7503 - category_output_loss: 0.6946 - loss: 2.1447 - subcategory_outp
Total Loss: 2.1389
Category Loss: 0.6909
Subcategory Loss: 1.3956
976/976 1s 1ms/step

Category Accuracy: 75.35%
Subcategory Accuracy: 54.83%

Category - Average Metrics:
Precision: 0.6954
Recall: 0.7535
F1-Score: 0.7084
Support: 31229

Subcategory - Average Metrics:
Precision: 0.5147
Recall: 0.5483
F1-Score: 0.5177
Support: 31229
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being
```

Accuracy Score Img (1) / Web App Source code Img (2)



The screenshot shows a Jupyter Notebook interface with the following content:

```
Model.ipynb X result.html
Model.ipynb > ...
+ Code + Markdown | ▶ Run All | Clear All Outputs | Outline ...
Python 3.12.7

from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.metrics import MeanMetricWrapper

# Step 1: Define the input layer
input_layer = Input(shape=(X_train.shape[1],))

# Hidden layers
hidden_layer = Dense(128, activation='relu', kernel_regularizer=l2(0.01))(input_layer)
hidden_layer = BatchNormalization()(hidden_layer)
hidden_layer = Dropout(0.5)(hidden_layer)
hidden_layer = Dense(128, activation='relu', kernel_regularizer=l2(0.01))(hidden_layer)
hidden_layer = BatchNormalization()(hidden_layer)
hidden_layer = Dropout(0.5)(hidden_layer)
hidden_layer = Dense(64, activation='relu', kernel_regularizer=l2(0.01))(hidden_layer)
hidden_layer = BatchNormalization()(hidden_layer)

# Output layers
category_output = Dense(len(category_encoder.classes_), activation='softmax', name='category_output')(hidden_layer)
subcategory_output = Dense(len(subcategory_encoder.classes_), activation='softmax', name='subcategory_output')(hidden_layer)

# Combine inputs and outputs
model = Model(inputs=input_layer, outputs=[category_output, subcategory_output])

# compile model
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer,
              loss={'category_output': 'categorical_crossentropy',
                   'subcategory_output': 'categorical_crossentropy'},
              metrics={'category_output': 'accuracy', 'subcategory_output': 'accuracy'})

# Set early stopping and learning rate scheduler
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5)

# Train the model
history = model.fit(X_train,
                    {'category_output': y_cat_train, 'subcategory_output': y_sub_train},
                    validation_data=(X_val,
                                   {'category_output': y_cat_val, 'subcategory_output': y_sub_val}),
                    epochs=epochs,
                    batch_size=batch_size,
                    callbacks=[early_stopping, lr_scheduler])

# Evaluate the model
results = model.evaluate(X_test,
                        {'category_output': y_cat_test,
                         'subcategory_output': y_sub_test})

# Print results for clarity
print(f"Total Loss: {results[0]:.4f}")
print(f"Category Loss: {results[1]:.4f}")
print(f"Subcategory Loss: {results[2]:.4f}")

# Safely access accuracy metrics based on the length of results
cat_acc = results[3] if len(results) > 3 else None
sub_acc = results[4] if len(results) > 4 else None
```

```
File Edit Selection View Go Run Terminal Help
FFNN WITH TRAIN AND TEST DATASET

Model.ipynb x app.py 1
Model.ipynb > from sklearn.metrics import classification_report
+ Code + Markdown | Run All | Clear All Outputs | Outline ...
Data preparation complete. Ready for model training!

from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.regularizers import l2
import pickle

# Step 4: Define the FFNN Model
input_layer = Input(shape=(X_train.shape[1],))

hidden_layer = Dense(256, activation='relu', kernel_regularizer=l2(0.01))(input_layer)
hidden_layer = BatchNormalization()(hidden_layer)
hidden_layer = Dropout(0.4)(hidden_layer)
hidden_layer = Dense(128, activation='relu', kernel_regularizer=l2(0.01))(hidden_layer)
hidden_layer = BatchNormalization()(hidden_layer)
hidden_layer = Dropout(0.3)(hidden_layer)
hidden_layer = Dense(64, activation='relu', kernel_regularizer=l2(0.01))(hidden_layer)
hidden_layer = BatchNormalization()(hidden_layer)

# Output layers
category_output = Dense(len(category_encoder.classes_), activation='softmax', name='category_output')(hidden_layer)
subcategory_output = Dense(len(subcategory_encoder.classes_), activation='softmax', name='subcategory_output')(hidden_layer)

# Combine inputs and outputs
model = Model(inputs=input_layer, outputs=[category_output, subcategory_output])

# Compile model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
```

Model Code img (3) / Crime Prediction for Complaint in English img(4)

Complaint Classification x +

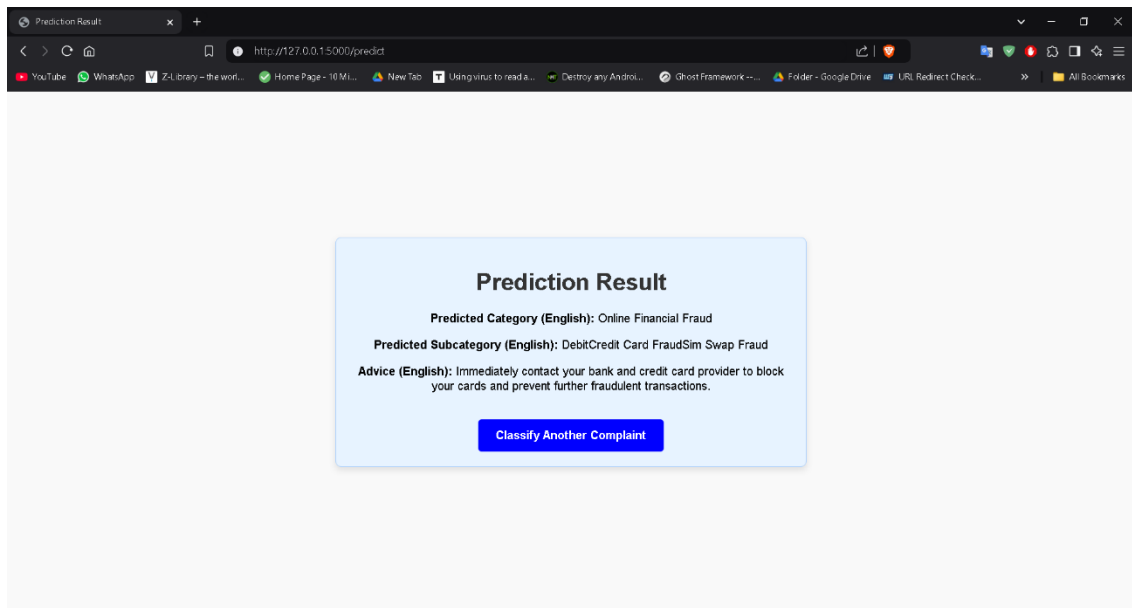
http://127.0.0.1:5000

Enter Complaint Details for Classification

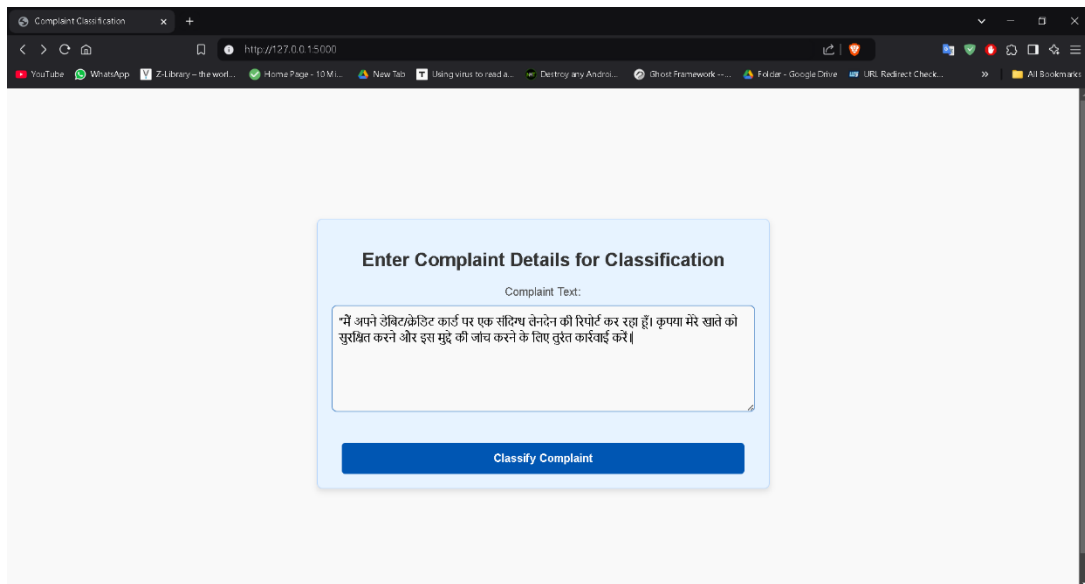
Complaint Text:

I received a notification about an unauthorized transaction on my credit card.
Please investigate this fraudulent activity immediately.

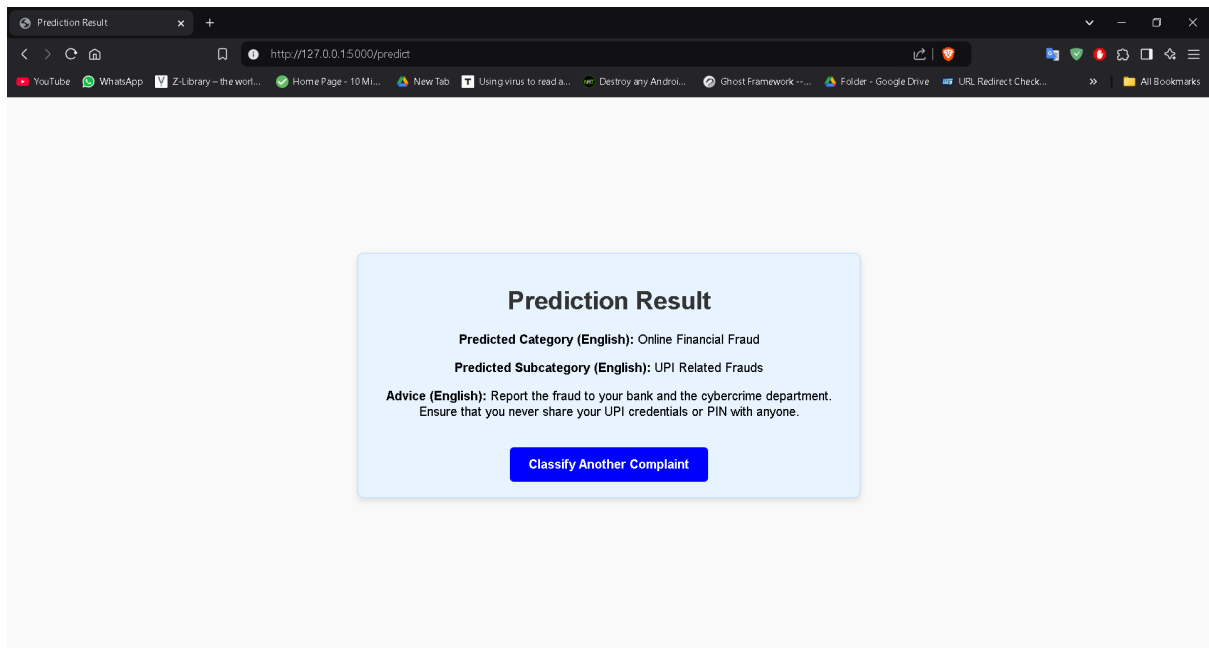
Classify Complaint



Prediction Result for Img (4) (complain in English) Img(5) / Complain in Regional Lngauge Img(6)



Prediction Result for Img (6) (complain in Regional Lnagugae) Img(7)



The screenshot shows a web browser window with a single tab titled "Prediction Result". The address bar displays the URL "http://127.0.0.1:5000/predict". The browser's taskbar at the bottom shows several open applications, including YouTube, WhatsApp, Z-Library, Home Page, New Tab, Using virus to read a..., Destroy any Androi..., Ghost Framework, Folder - Google Drive, URL Redirect Check..., and All Bookmarks.

The main content area of the browser displays a light blue box with the following text:

Prediction Result

Predicted Category (English): Online Financial Fraud

Predicted Subcategory (English): UPI Related Frauds

Advice (English): Report the fraud to your bank and the cybercrime department.
Ensure that you never share your UPI credentials or PIN with anyone.

Below the text is a blue button with the text "Classify Another Complaint".

7. Implementation Plan (200 words)

7.1 Deployment Strategy

We will deploy the system on a **cloud platform** (e.g., **AWS** or **Google Cloud**) to ensure scalability and handle large volumes of user complaints. The web application will be containerized using **Docker** for easy maintenance and seamless updates. A **REST API** will be developed for real-time predictions, and the backend will be optimized for fast query processing.

7.2 Future Enhancements

- **Advanced NLP Models:** To improve the accuracy, we will integrate more sophisticated models like **BERT** or **CyberBERT**, specifically fine-tuned for cybersecurity-related text.
- **Regional Language Support:** We will expand the dataset to include complaints in regional languages, ensuring the model is accessible to a broader demographic.
- **Continuous Learning:** The system will be updated periodically with new complaint data, and user feedback will be integrated into the training process to improve predictions over time.

8. References

Libraries and Tools

1. **TensorFlow and Keras:** Used for building and training the FFNN model.
2. **Gensim:** Used for generating FastText embeddings.
3. **Flask:** Used for building the web application.
4. **Google Translate API:** Enabled multilingual support for user inputs and advice outputs.
5. **NLTK:** Preprocessed text using stopwords removal and lemmatization.
6. **langdetect:** Detected the language of user inputs.

External Resources

1. **Cybersecurity Guidelines:** Consulted cybersecurity guidelines to align classification categories with real-world scenarios.
2. **Documentation:**
 - TensorFlow, Keras, and Flask official documentation.
 - FastText model implementation guides.

9. Plagiarism Declaration

We affirm that all work presented in this project is **original** or **properly cited**. All external libraries, models, and tools used are acknowledged, and the project adheres to ethical AI development principles.