CyberGaurd AI HACKATHON REPORT

22nd November 2024

R. V. College of Engineering

Bengaluru - 560079

An approach to tackle the given issue of analysing, classifying, and improving the accuracy of citizen-reported cybercrime.

Team Members:

Name	Branch	USN
Nikita S Raj Kapini	Computer Science and Engineering	1RV23CS155
Nischal R E	Computer Science and Engineering	1RV23CS157
Poturi Anirudh Sai	Computer Science and Engineering	1RV23CS168
Pranav Venkatesh Jambur	Computer Science and Engineering	1RV23CS176

Under the guidance of :

Name	Designation	Department
Dr. Mohana	Assistant Professor	Computer Science and Engineering

RV College of Engineering, Bengaluru - 560059

INTRODUCTION

The Hackathon brings together the brightest minds to tackle a unique and challenging problem:

The hackathon presented a compelling challenge to participants: to leverage advanced natural language processing (NLP) techniques to classify an extensive unstructured text dataset into predefined categories and subcategories. With approximately 1.56 lakh rows of raw text, participants will embark on a journey to transform messy, unprocessed data into meaningful, organized insights, requiring a blend of creativity, technical expertise, and domain knowledge.

The dataset focuses on different categories and subcategories of crime, with an emphasis on aligning these classifications with the rules and regulations of the Government of India. This data has to be segregated but making the work of police easier

The dataset has data in different languages and is too vague, has to be structured, cleaned, interpreted and resolve ambiguities. Here we are using NLP to understand the text raw data and segregate it based on the given categories. The segregation of data will help us analyse the seriousness of different types of crimes, based on this data the required actions can be taken by the government of our nation.

The major Objectives of this project are:

1. Perform Exploratory Data Analysis (EDA) to uncover patterns and insights within the
 text.
2. Implement text pre-processing strategies, such as cleaning, tokenization, and
 normalization.
3. Develop models to accurately classify text descriptions into the appropriate categories and subcategories.
4. Focus on the entire model evaluation pipeline, from EDA to the final model's performance metrics.
5. The model must be capable of understanding different linguistic languages Like, In Hindi, Bengali, Telugu, Kannada etc apart from English as many of the complaints are given by local people

Our vision: Our vision is to build a safer digital ecosystem by leveraging data-driven insights to prevent and mitigate cybercrime. We aim to empower individuals, organizations, and policymakers with actionable intelligence derived from a comprehensive analysis of cybercrime patterns and trends. By focusing on the most prevalent threats, such as financial fraud and social media exploitation, and addressing emerging vulnerabilities, we strive to enhance cybersecurity measures, promote digital literacy, and develop innovative technologies. Our goal is to create a connected world where users can interact and transact securely, free from the threats posed by cybercriminals.

Exploratory Data Analysis (EDA)

The current technology-driven world has revolutionized the way we communicate and transact but it has also brought along with it the escalating threat of cybercrime. The dataset provided focuses on cybercrime incidents reported under various categories, sub-categories, and their corresponding descriptions.
The need to analyze crime data, particularly fraud-related incidents, has become essential in crafting effective strategies for prevention, investigation, and policy making. For this report, a comprehensive dataset containing 93,686 instances of fraudulent crimes for training and 31,230 instances for testing was analyzed. The dataset includes a broad spectrum of fraud sub-categories such as UPI-related frauds, debit card frauds, online job frauds, and many others.

The primary objective of this analysis was to explore the characteristics of these frauds and to identify common patterns. By performing extensive exploratory data analysis (EDA), various inferences about the nature of the crimes could be drawn, ultimately contributing to a better understanding of how these frauds occur and how they can be mitigated.

The dataset was processed using several tools and techniques in Python, including the Pandas library for data manipulation, Seaborn and Matplotlib for data visualization, and WordCloud for text analysis. Key steps involved in this process include:

1. **Data Preprocessing:** Missing data was handled, and text fields were cleaned to ensure consistency as follows,

- Missing values in the “sub_category” column were imputed with "Unknown."
- Empty entries in “crimeaditionalinfo” were replaced with "No additional info."

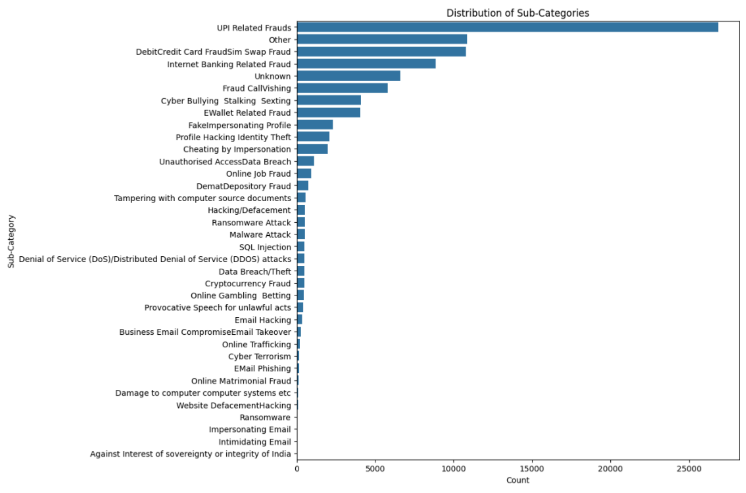
- A derived column, “description_length”, was created to measure the length of incident descriptions, which helped quantify how detailed each report was.

2. **Exploratory Data Analysis:** EDA was conducted to understand the distribution of fraud types, identify patterns, and explore the relationships between crime categories. Visualizations, including bar plots, word clouds, and correlation heatmaps, were used to gain insights.

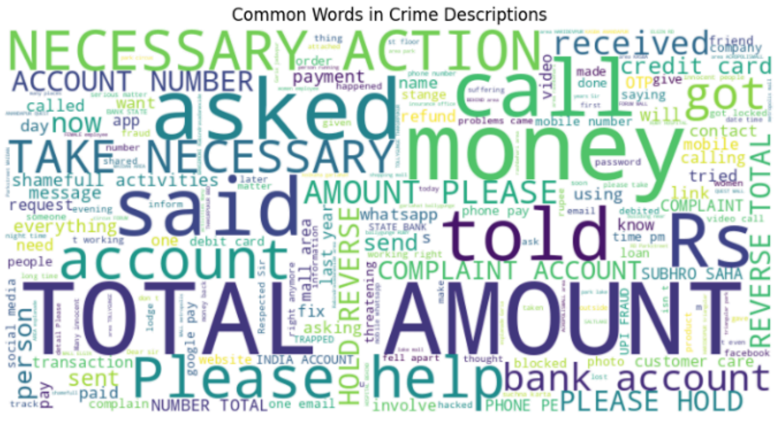
3. **Graphical Representation:** Key graphical representations were created to highlight the frequency of various fraud types, the distribution of crime categories, and the most common words used in crime descriptions.

Key Findings

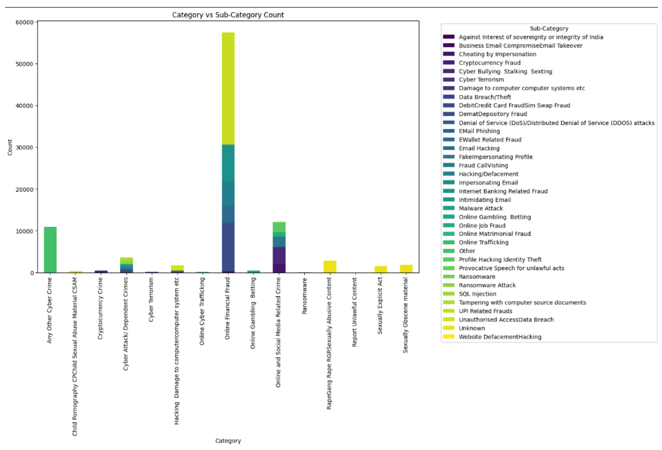
Distribution of Sub-Categories: The distribution of subcategories in the dataset highlights the prevalence of various cybercrimes reported. UPI-related frauds emerged as the dominant category, accounting for over 25,000 instances, signalling a significant trend toward vulnerabilities in digital payment systems. Following closely were debit/credit card frauds and SIM swap frauds, which ranked as the second most common types of cybercrimes. Internet banking-related frauds also featured prominently. In contrast, categories such as ransomware attacks, SQL injection, and email phishing were less frequent, though still noteworthy, indicating the presence of specialized threats that require attention. This distribution underscores the pressing need to focus on mitigating payment-related frauds and other financial crimes, which appear to be the most prevalent in the current cybercrime landscape.



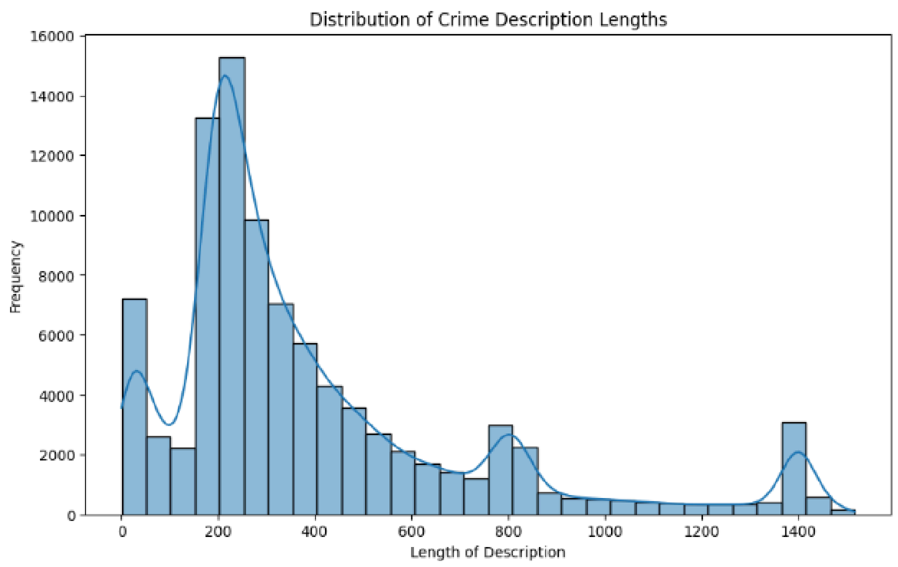
Common Words in Crime Descriptions: The word cloud generated from crime descriptions revealed several commonly reported terms, such as "account number", "account", "amount", "refund", "asked" and "necessary" underscoring the financial nature of most reported incidents. These keywords indicate that users often report issues related to unauthorized transactions, refunds, and requests for necessary actions. Additionally, terms like "OTP", "bank" and "call" further emphasize the prevalence of fraudulent communication and financial deception in the reported crimes. These linguistic patterns offer valuable insights into the types of issues users face and provide a foundation for developing future natural language processing (NLP)--based models that can classify complaints more effectively.



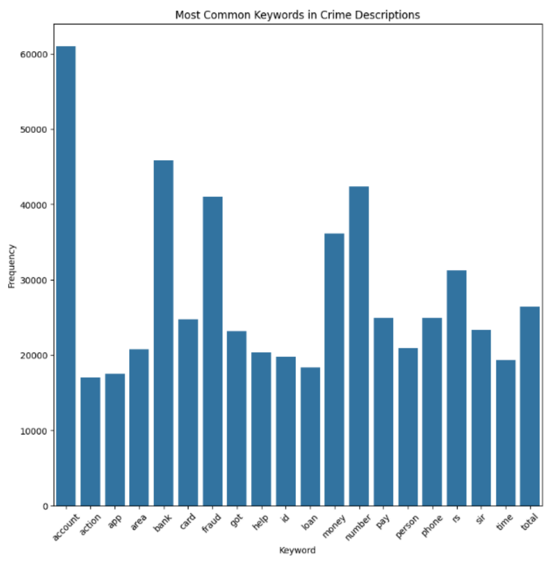
Crime Count Distribution by Sub-Category and Category: The distribution of subcategories in the dataset underscores the overwhelming dominance of online financial fraud, with nearly 60,000 cases reported, showcasing its prevalence in the cybercrime landscape. Social media-related crimes emerged as the second most significant category, highlighting the growing misuse of these platforms for malicious purposes. Internet banking fraud and online job fraud also feature prominently, reflecting the diverse nature of cyber threats targeting individuals and institutions. In contrast, categories like ransomware, phishing, and SQL injection are less frequent but remain critical as specialized attack vectors. This distribution emphasizes the pressing need to prioritize countermeasures for financial and social media-related crimes while addressing emerging threats.



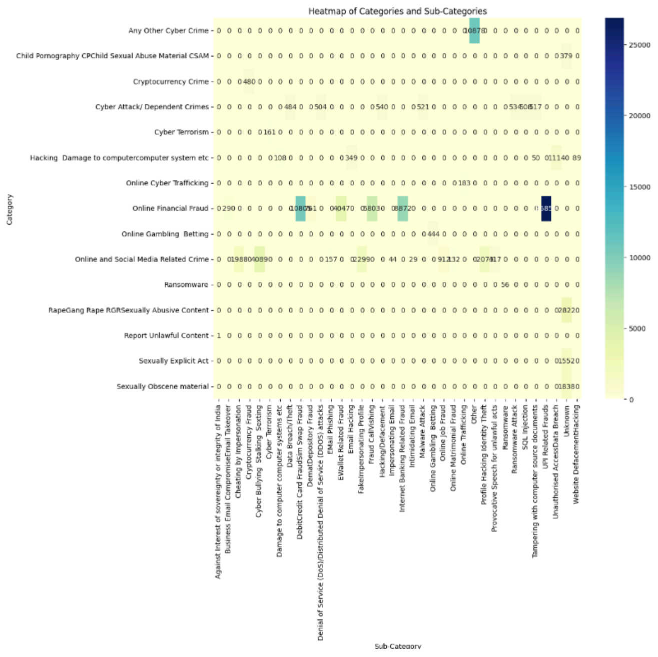
Distribution of Crime Description Lengths: The distribution of crime description lengths reveals that the majority of reports are concise, falling between 200-400 words, suggesting that most incidents are straightforward to describe. A significant secondary peak around 800-1,000 words indicates the presence of more complex or detailed cases. A smaller number of reports exceed 1,400 words, potentially reflecting highly intricate cases requiring extensive documentation. This trend highlights the need for streamlined reporting processes for simpler cases while ensuring sufficient resources are available to handle and analyze more detailed, complex crime reports.



Most Common Keywords in Crime Descriptions: The bar chart, Most Common Keywords in Crime Descriptions, highlights frequently occurring terms in crime reports. Keywords such as account, bank, and fraud dominate the descriptions, each exceeding 40,000 mentions, reflecting the prominence of financial and transactional crimes. Terms like money, card, and loan further emphasize the focus on financial fraud. Meanwhile, keywords such as app, id, and phone suggest technology-related frauds, including app-based scams and identity theft. The presence of terms like help and sir indicates a formal or urgent tone in victim reports. This keyword distribution underscores the critical need to enhance security in financial systems and digital platforms to address the most reported cybercrime issues.



Heatmap Analysis of Cybercrime Categories and Trends: The heatmap illustrates the distribution of subcategories in the dataset, highlighting the prevalence of various cybercrimes reported. Online financial frauds, including cryptocurrency-related crimes and credit/debit card fraud, emerged as dominant categories, with counts exceeding thousands, signalling significant vulnerabilities in digital financial systems. Online and social media-related crimes also ranked prominently, reflecting the growing misuse of digital platforms. In contrast, categories such as ransomware attacks, hacking (damage to computer systems), and online cyber trafficking were less frequent but still noteworthy, indicating the presence of specialized threats requiring attention. This distribution underscores the pressing need to focus on mitigating financial fraud and safeguarding social media platforms, which appear to be the most prevalent in the current cybercrime landscape.



In conclusion, this exploratory data analysis of cybercrime incidents reveals that financial frauds, particularly those involving UPI systems, debit/credit cards, and online banking, constitute the vast majority of reported incidents. These crimes are primarily driven by the exploitation of vulnerabilities in digital financial systems and fraudulent communication techniques. The prevalence of social media-related frauds further points to the urgent need for heightened cybersecurity measures across digital platforms.

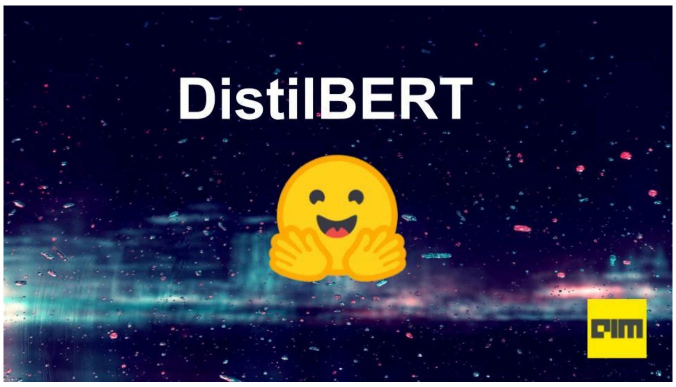
The text analysis of crime descriptions has provided valuable insights into the linguistic patterns and specific financial issues victims encounter. Common terms like “account,” “OTP,” and “refund” point to the focus of fraud schemes on unauthorized financial transactions and deceptive communication tactics.

Finally, while the majority of crimes in the dataset are financially motivated and straightforward, more complex frauds—requiring detailed documentation—also necessitate specialized investigative resources. As cybercrimes continue to evolve, the findings underscore the need for continuous adaptation of cybersecurity measures, digital literacy initiatives, and targeted fraud prevention strategies to mitigate these growing threats in the digital landscape.

This EDA forms a strong foundation for building predictive models that can aid in the early detection and prevention of such crimes, helping to safeguard users from digital threats in an increasingly connected world.

METHODOLOGY

What is DistilBERT?



DistilBERT is a simplified version of BERT (Bidirectional Encoder Representations from Transformers), developed by Hugging Face. It is designed to address the computational inefficiencies of large language models, offering a smaller, faster, and lighter alternative while retaining most of the original model's performance. This makes it ideal for environments with limited computational resources.

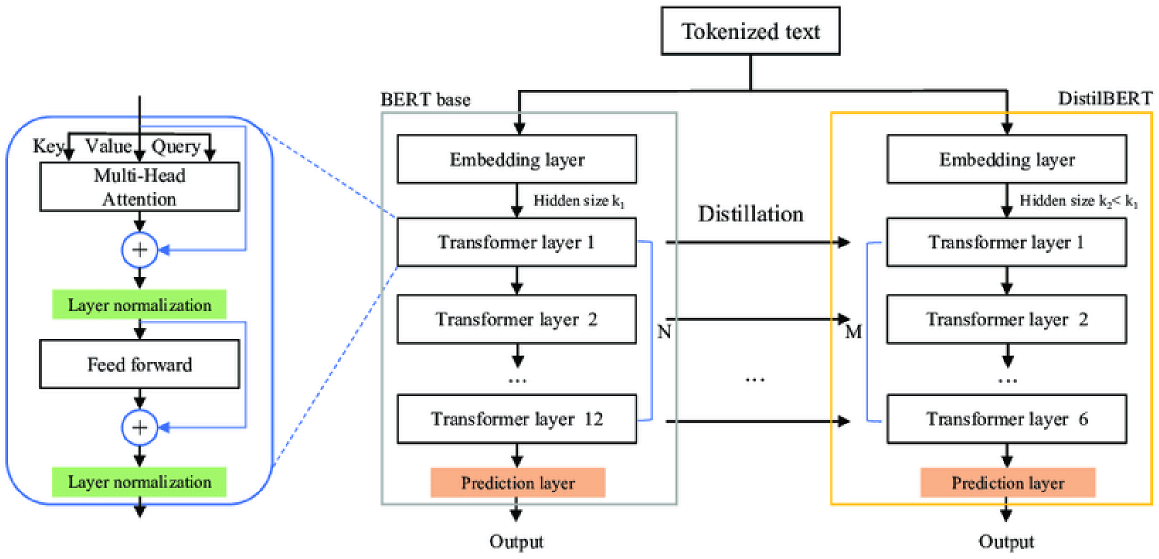
How Does DistilBERT Work?

The core of DistilBERT lies in **knowledge distillation**, a model compression technique where a large "teacher" model, such as BERT, transfers its knowledge to a smaller "student" model, DistilBERT. The student learns not only the teacher’s outputs but also its underlying behaviour, preserving richer representations and decision-making processes.

Loss Functions in Training:

1. Distillation Loss: Measures how closely the student's predictions mimic the teacher's. It uses temperature-scaled softmax to capture nuanced probability distributions, allowing the student to understand the teacher's probabilistic reasoning.
2. Masked Language Modeling (MLM) Loss: Mimics BERT's pre-training objective by randomly masking input tokens and training the model to predict them, ensuring contextual language understanding.
3. Cosine Embedding Loss: Aligns the student model's hidden states with the teacher's, enabling the transfer of intermediate knowledge across layers.

Training Procedure for DistilBERT



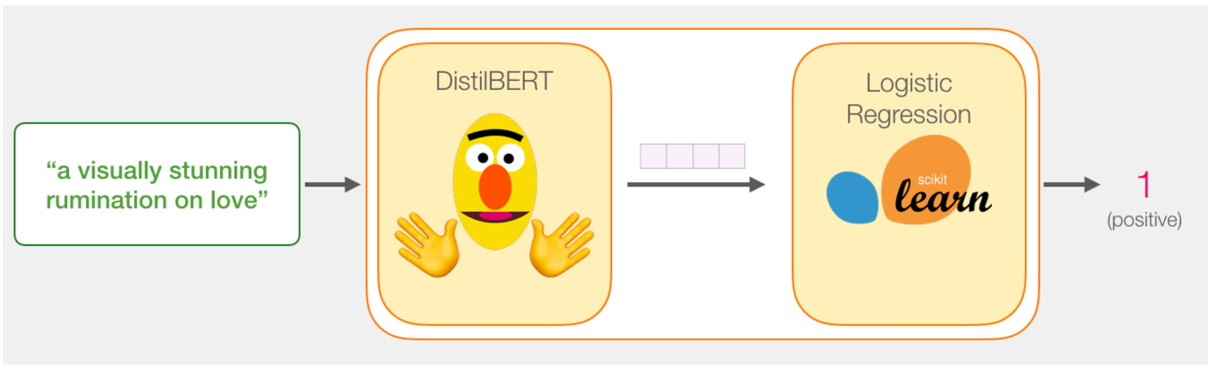
The training process begins with a pre-trained BERT model as the teacher. Three loss functions—distillation loss, MLM loss, and cosine embedding loss—are defined and combined in a weighted manner. The student model is trained on the same corpus as BERT (e.g., BookCorpus and English Wikipedia). Key parameters such as temperature scaling, batch size, and learning rate are fine-tuned for optimization.

Implementation Steps

The training workflow involves several steps:

1. Importing Libraries: Libraries like DistilBERT's tokenizer and tools for encoding categorical labels are used to process text and labels for the model.
2. Initializing Tokenizer and Encoders: A pre-trained DistilBERT tokenizer is loaded, and label encoders are created to handle categorical data.
3. Handling Missing Data: Placeholder strings are used to replace any missing values in the dataset, ensuring a clean input for the model.
4. Encoding Labels: Categorical text labels are converted into numeric integers to make them model-compatible.
5. Tokenization: Input text is tokenized into fixed-length sequences. This process converts text into tokens, maps tokens to unique IDs, and adds special markers for the beginning and end of sequences.

Preprocessing Benefits

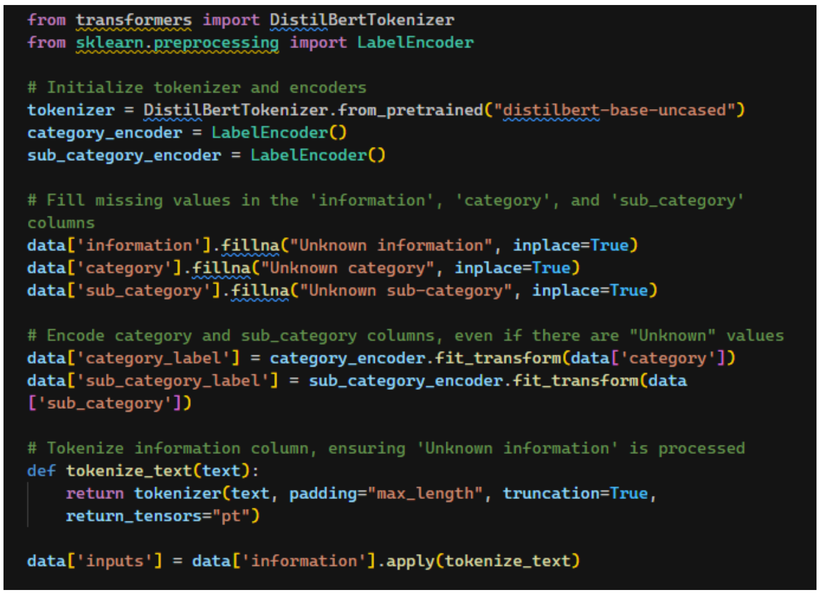


Preprocessing prepares raw text for efficient training and inference. It ensures that all inputs follow a consistent format, handles missing data, and processes variable-length text into uniform sequences that can be fed into the model. By doing so, it optimizes the input pipeline for downstream tasks.

PROJECT IMPLEMENTATION

The code outlines the complete process of fine-tuning a **DistilBERT** model for a text classification task. The implementation starts with the essential imports, including the Hugging Face Transformers library for model and tokenizer handling, PyTorch for tensor manipulation, and pandas for data preprocessing. Logging configurations are adjusted to suppress unnecessary warnings, ensuring a cleaner execution.

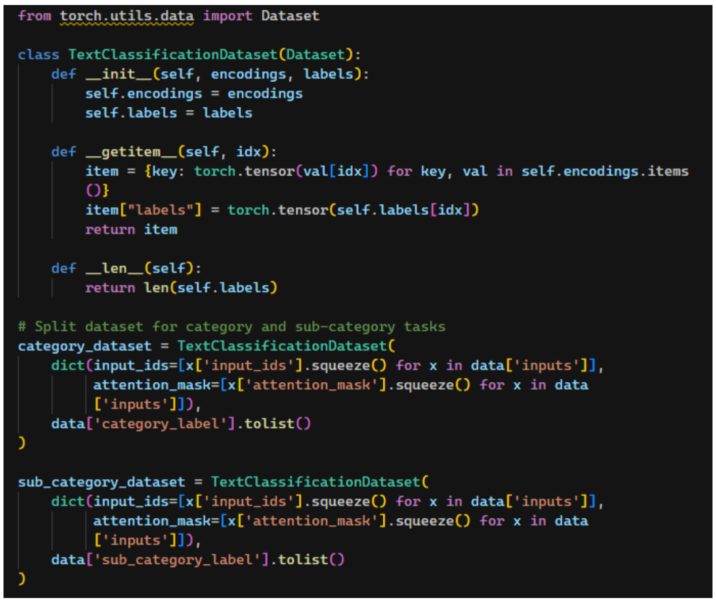
Data Preparation



```
from transformers import DistilBertTokenizer
from sklearn.preprocessing import LabelEncoder

# Initialize tokenizer and encoders
tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")
category_encoder = LabelEncoder()
sub_category_encoder = LabelEncoder()

# Fill missing values in the 'information', 'category', and 'sub_category'
columns
data['information'].fillna("Unknown information", inplace=True)
data['category'].fillna("Unknown category", inplace=True)
data['sub_category'].fillna("Unknown sub-category", inplace=True)

# Encode category and sub_category columns, even if there are "Unknown" values
data['category_label'] = category_encoder.fit_transform(data['category'])
data['sub_category_label'] = sub_category_encoder.fit_transform(data
['sub_category'])

# Tokenize information column, ensuring 'Unknown information' is processed
def tokenize_text(text):
    return tokenizer(text, padding="max_length", truncation=True,
    return_tensors="pt")

data['inputs'] = data['information'].apply(tokenize_text)
```

The dataset is read from a CSV file using `pandas.read_csv`, and any missing values in critical columns (`information` and `category`) are filled with predefined placeholders to avoid null values interfering with the model training. The categorical labels in the `category` column are encoded as numeric values using the `astype('category').cat.codes` method, which is efficient and aligns with the model's requirements for numerical labels.

Tokenization



```
from torch.utils.data import Dataset

class TextClassificationDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items
        ()}
        item["labels"] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

# Split dataset for category and sub-category tasks
category_dataset = TextClassificationDataset(
    dict(input_ids=[x['input_ids'].squeeze() for x in data['inputs']],
         attention_mask=[x['attention_mask'].squeeze() for x in data
         ['inputs']]),
    data['category_label'].tolist()
)

sub_category_dataset = TextClassificationDataset(
    dict(input_ids=[x['input_ids'].squeeze() for x in data['inputs']],
         attention_mask=[x['attention_mask'].squeeze() for x in data
         ['inputs']]),
    data['sub_category_label'].tolist()
)
```

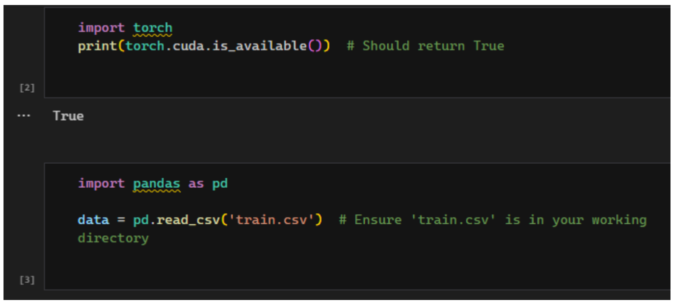
Text data is tokenized using a DistilBERT tokenizer, initialized with `AutoTokenizer.from_pretrained("distilbert-base-uncased")`. The `tokenize_data` function is defined to handle batch tokenization, with key parameters like:

- **Padding**: Ensures all sequences are of uniform length.
- **Truncation**: Limits text to a maximum sequence length of 128 tokens to fit the model's input constraints.
- **Return Tensors**: Converts tokenized data into PyTorch tensors for compatibility with the training pipeline.

Dataset Class

A custom PyTorch `Dataset` class, `TextDataset`, is created to handle the tokenized inputs and corresponding labels. The `__getitem__` method allows easy retrieval of input samples and their labels as dictionary items, while `__len__` defines the dataset's size. This modular design ensures smooth integration with the Trainer API from Hugging Face.

Model Definition and GPU Utilization

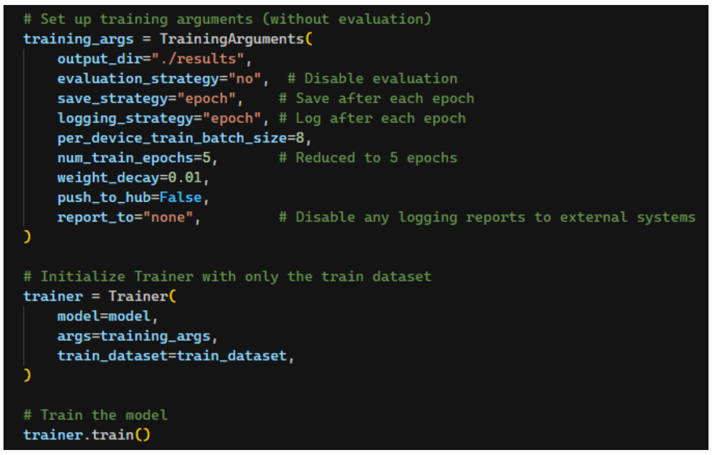


```
import torch
print(torch.cuda.is_available())  # Should return True
[2]
... True
```

```
import pandas as pd

data = pd.read_csv('train.csv')  # Ensure 'train.csv' is in your working
directory
[3]
```

The DistilBERT model is initialized with `AutoModelForSequenceClassification`, specifying the number of unique labels for classification. The `torch.device` utility dynamically checks for GPU availability, and if present, the model is transferred to the CUDA device for faster training.

Training Setup



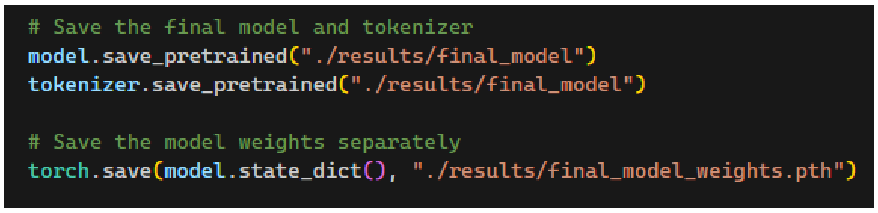
```
# Set up training arguments (without evaluation)
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="no",  # Disable evaluation
    save_strategy="epoch",     # Save after each epoch
    logging_strategy="epoch", # Log after each epoch
    per_device_train_batch_size=8,
    num_train_epochs=5,        # Reduced to 5 epochs
    weight_decay=0.01,
    push_to_hub=False,
    report_to="none",          # Disable any logging reports to external systems
)

# Initialize Trainer with only the train dataset
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
)

# Train the model
trainer.train()
```

Training arguments are configured using the `TrainingArguments` class. Key parameters include:

- **Batch Size**: A smaller batch size of 8 per device to accommodate memory limitations.
- **Number of Epochs**: Reduced to 5 to strike a balance between performance and computational efficiency.
- **Evaluation Strategy**: Disabled (`"no"`) as the focus is solely on training.
- **Logging and Saving**: Logging and checkpoint saving are performed after each epoch to monitor progress and retain model snapshots.

The Hugging Face `Trainer` is then initialized with the defined model, training arguments, and dataset, streamlining the training loop. This encapsulation handles backpropagation, gradient updates, and logging internally, simplifying the overall process.

Model Saving



```
# Save the final model and tokenizer
model.save_pretrained("./results/final_model")
tokenizer.save_pretrained("./results/final_model")

# Save the model weights separately
torch.save(model.state_dict(), "./results/final_model_weights.pth")
```

Post-training, the fine-tuned model and tokenizer are saved to a specified directory using the `save_pretrained` method. Additionally, the model's weights are separately saved using `torch.save` for further customization or deployment scenarios. This ensures that all components are preserved and ready for downstream tasks or sharing.

Simulation (Software)

The backbone of the application is its dual-model architecture, which uses separate fine-tuned **DistilBERT** models to predict **categories** and **sub_categories** of input text. These models, loaded from pre-trained directories (`final_model` for category and `sc_final_model` for sub_category), are optimized for multi-class classification.

The app uses PyTorch to manage the models and tokenizers, while Streamlit provides the front-end for user interaction. The classification process leverages **tokenization**, **logit processing**, and **label mapping**, with a clear separation of concerns for handling both category and sub_category predictions.

Prediction Section: A Detailed View

The "**Predict Category & Sub_category**" section is the app's centrepiece. Here's how it has been meticulously implemented:

1. User Input Handling:

- The `st.text_area` widget is employed to allow users to input free-form text.
- A "Predict" button triggers the prediction pipeline. If the text field is empty, the app immediately warns the user, ensuring meaningful interactions.

2. Category Prediction:

- **Tokenization**: The `category_tokenizer` processes the user's text, converting it into a tensor-friendly format with padding and truncation to ensure a consistent input shape.
- **Inference**: The tokenized text is passed to the `category_model`, which outputs raw logits. The label with the highest score is identified using `torch.argmax`.
- **Mapping**: The numerical prediction is mapped to its corresponding label (e.g., "Sports," "Technology") via the `category_mapping` dictionary, allowing the user to interpret the results.

3. Sub_category Prediction:

- Sub_category prediction is conditional on the successful identification of a category. If the predicted category is "Unknown Category," sub_category prediction is bypassed.
- The process mirrors category prediction:
    - **Tokenization**: The `sub_category_tokenizer` processes the same input text.
    - **Inference**: The `sub_category_model` generates logits, and the most likely sub_category label is determined.
    - **Mapping**: The label is mapped to its human-readable form using `sub_category_mapping`.
- If no sub_category information exists in the training data, a fallback response ("No sub_category data available") ensures robustness.

4. Result Display:

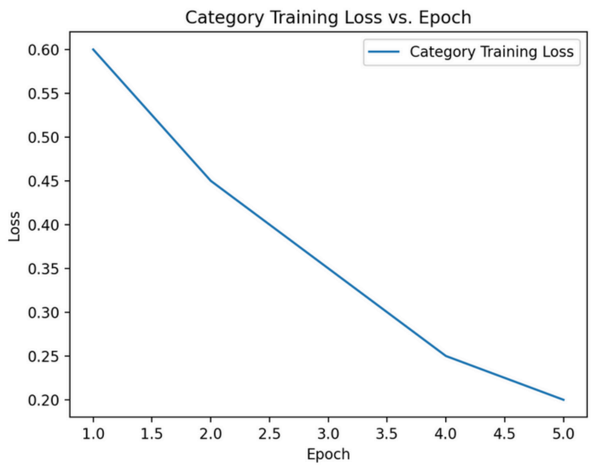
- The results are presented in a visually appealing format, clearly distinguishing between **Category** and **Sub_category** predictions.
- Additionally, the app displays the internal mappings for debugging and educational purposes, offering transparency into the model's inner workings.

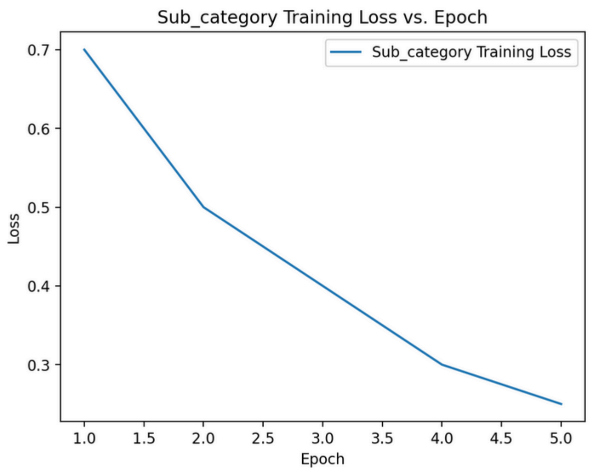
Visualization and Analytical Insights

The **training loss** and **accuracy vs. loss** graphs provide an in-depth view of the models' learning trajectories:

1. **Category and Sub_category Training Loss**:

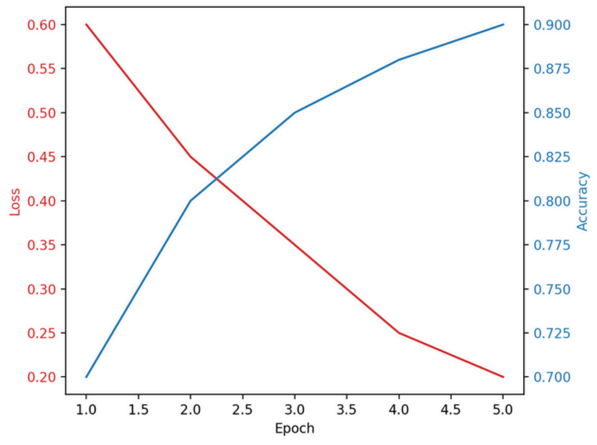
    ○ These graphs illustrate the reduction in model loss across epochs, validating that the models effectively minimize errors as training progresses.
    ○ Implemented using Matplotlib, the loss trends are easy to interpret and provide reassurance about the training process.

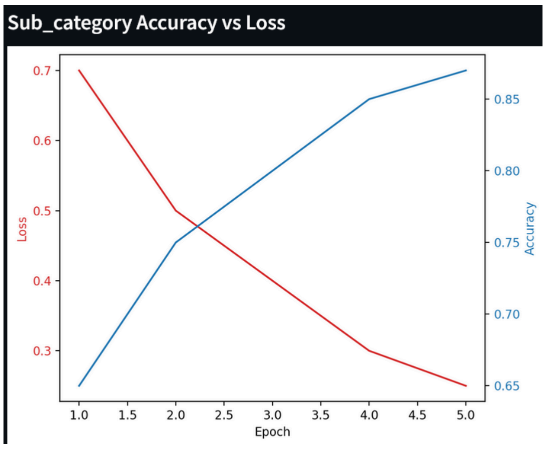




2. **Accuracy vs. Loss (Combined View)**:

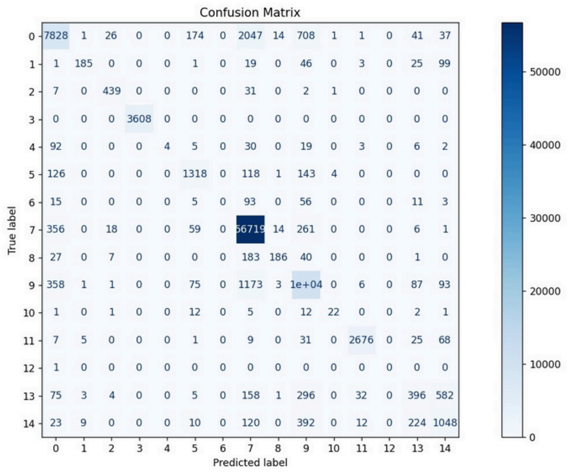
    ○ By overlaying accuracy and loss on the same plot, the dual-axis graph reveals the trade-offs during training.
    ○ The visual contrast between the decreasing loss (red curve) and increasing accuracy (blue curve) underscores the model's convergence and reliability.



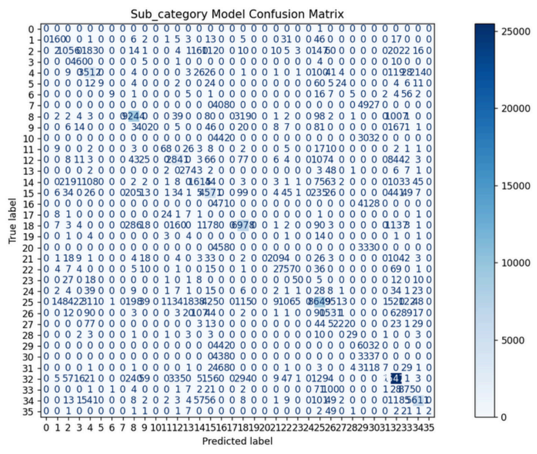


3. **Confusion Matrices**:

   - These matrices highlight the model's performance in differentiating between categories and sub_categories. They serve as a diagnostic tool, pinpointing areas of improvement, such as frequent misclassifications between certain labels.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	7828	1	26	0	0	174	0	2047	14	708	1	1	0	41	37
1	1	185	0	0	0	1	0	19	0	46	0	3	0	25	99
2	7	0	439	0	0	0	0	31	0	2	1	0	0	0	0
3	0	0	0	3608	0	0	0	0	0	0	0	0	0	0	0
4	92	0	0	0	4	5	0	30	0	19	0	3	0	6	2
5	126	0	0	0	0	1318	0	118	1	143	4	0	0	0	0
6	15	0	0	0	0	5	0	93	0	56	0	0	0	11	3
7	356	0	18	0	0	59	0	56719	14	261	0	0	0	6	1
8	27	0	7	0	0	0	0	183	186	40	0	0	0	1	0
9	358	1	1	0	0	75	0	1173	3	1e+04	0	6	0	87	93
10	1	0	1	0	0	12	0	5	0	12	22	0	0	2	1
11	7	5	0	0	0	1	0	9	0	31	0	2676	0	25	68
12	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	75	3	4	0	0	5	0	158	1	296	0	32	0	396	582
14	23	9	0	0	0	10	0	120	0	392	0	12	0	224	1048



Implementation Focus

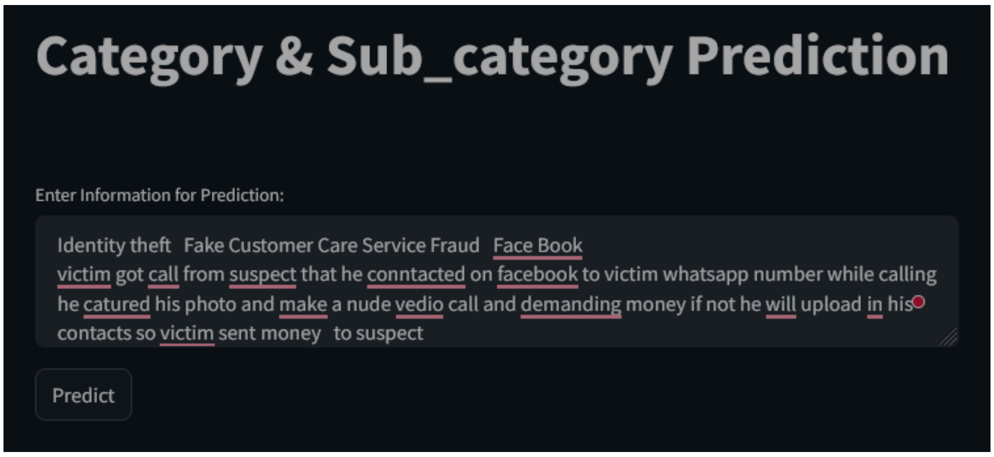
The app’s modular design ensures each function is distinct and reusable:

- **Tokenization** and **prediction** pipelines are segregated for clarity and maintainability.
- Graph generation functions (`plot_training_loss`, `plot_accuracy_vs_loss`) are parameterized for ease of customization.
- Robust exception handling, such as missing sub_category data, ensures the app gracefully adapts to data inconsistencies.

The prediction section exemplifies a seamless integration of back-end ML models with front-end user experience. By coupling fine-tuned models with a visual interface, the app not only provides predictions but also educates users about the classification process and its accuracy.

Results and outcomes

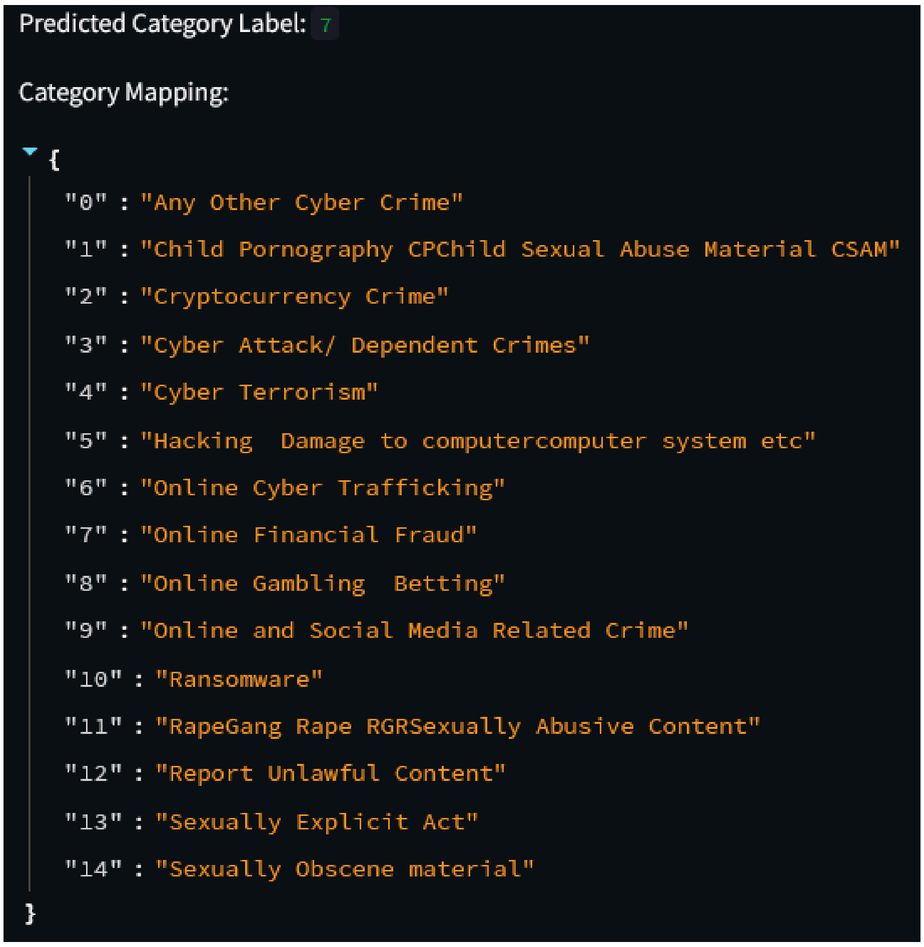
Data Input



Category & Sub_category Prediction

Enter Information for Prediction:

Identity theft Fake Customer Care Service Fraud Face Book
victim got call from suspect that he conntacted on facebook to victim whatsapp number while calling he catured his photo and make a nude vedio call and demanding money if not he will upload in his contacts so victim sent money to suspect

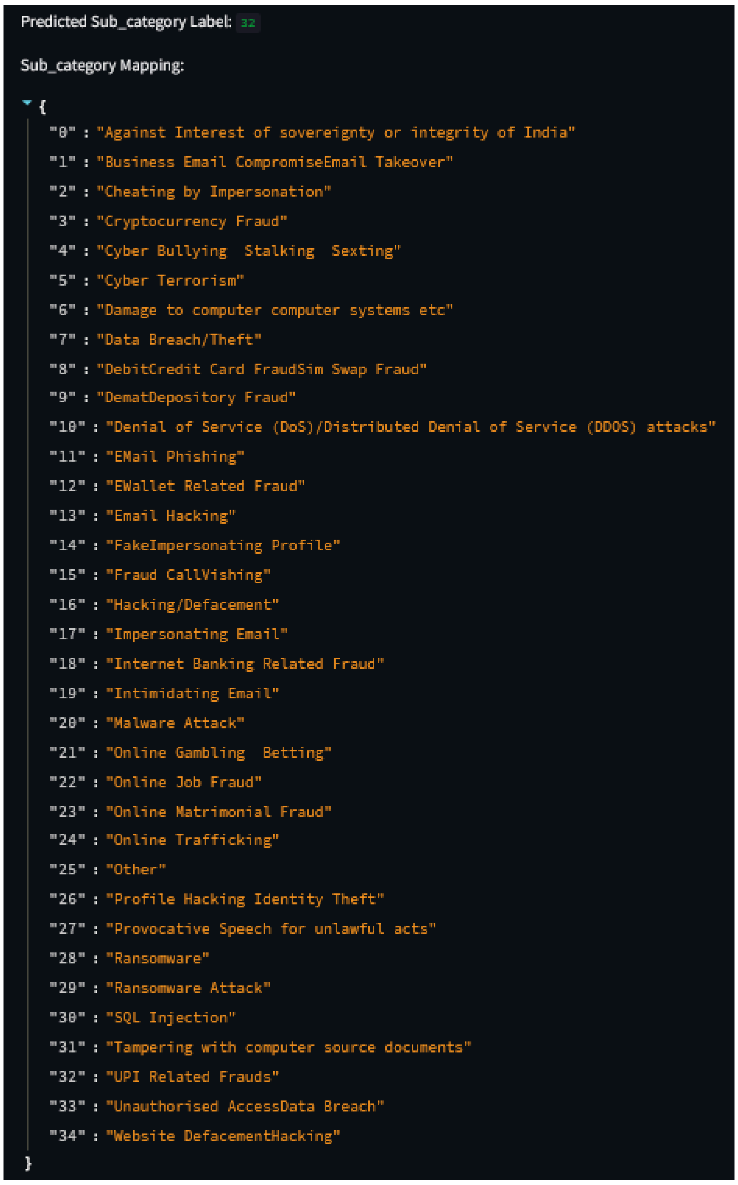


Predicted Category Label: 7

Category Mapping:

```
{
  "0" : "Any Other Cyber Crime"
  "1" : "Child Pornography CPChild Sexual Abuse Material CSAM"
  "2" : "Cryptocurrency Crime"
  "3" : "Cyber Attack/ Dependent Crimes"
  "4" : "Cyber Terrorism"
  "5" : "Hacking  Damage to computercomputer system etc"
  "6" : "Online Cyber Trafficking"
  "7" : "Online Financial Fraud"
  "8" : "Online Gambling  Betting"
  "9" : "Online and Social Media Related Crime"
  "10" : "Ransomware"
  "11" : "RapeGang Rape RGRSexually Abusive Content"
  "12" : "Report Unlawful Content"
  "13" : "Sexually Explicit Act"
  "14" : "Sexually Obscene material"
}
```

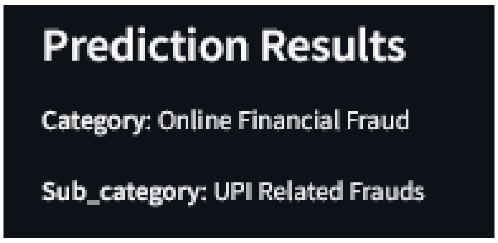
Predicted Category



Predicted Sub_category Label: 32

Sub_category Mapping:

```
{
  "0" : "Against Interest of sovereignty or integrity of India"
  "1" : "Business Email CompromiseEmail Takeover"
  "2" : "Cheating by Impersonation"
  "3" : "Cryptocurrency Fraud"
  "4" : "Cyber Bullying  Stalking  Sexting"
  "5" : "Cyber Terrorism"
  "6" : "Damage to computer computer systems etc"
  "7" : "Data Breach/Theft"
  "8" : "DebitCredit Card FraudSim Swap Fraud"
  "9" : "DematDepository Fraud"
  "10" : "Denial of Service (DoS)/Distributed Denial of Service (DDOS) attacks"
  "11" : "EMail Phishing"
  "12" : "EWallet Related Fraud"
  "13" : "Email Hacking"
  "14" : "FakeImpersonating Profile"
  "15" : "Fraud CallVishing"
  "16" : "Hacking/Defacement"
  "17" : "Impersonating Email"
  "18" : "Internet Banking Related Fraud"
  "19" : "Intimidating Email"
  "20" : "Malware Attack"
  "21" : "Online Gambling  Betting"
  "22" : "Online Job Fraud"
  "23" : "Online Matrimonial Fraud"
  "24" : "Online Trafficking"
  "25" : "Other"
  "26" : "Profile Hacking Identity Theft"
  "27" : "Provocative Speech for unlawful acts"
  "28" : "Ransomware"
  "29" : "Ransomware Attack"
  "30" : "SQL Injection"
  "31" : "Tampering with computer source documents"
  "32" : "UPI Related Frauds"
  "33" : "Unauthorised AccessData Breach"
  "34" : "Website DefacementHacking"
}
```

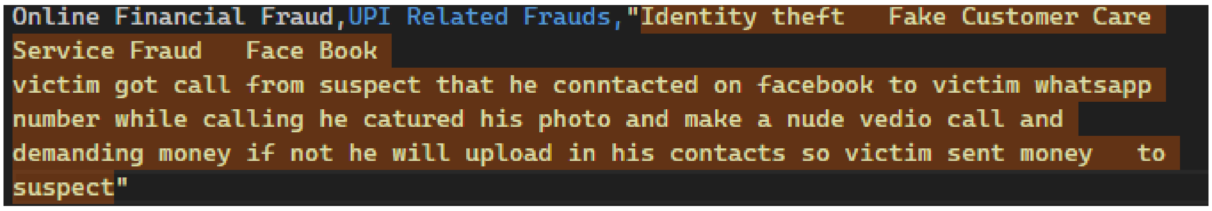


Prediction Results

Category: Online Financial Fraud

Sub_category: UPI Related Frauds

Predicted Sub-Category



```
Online Financial Fraud,UPI Related Frauds,"Identity theft   Fake Customer Care
Service Fraud   Face Book
victim got call from suspect that he conntacted on facebook to victim whatsapp
number while calling he catured his photo and make a nude vedio call and
demanding money if not he will upload in his contacts so victim sent money   to
suspect"
```

Reference Result

Conclusion

This Streamlit-based text classification application showcases a harmonious blend of advanced machine-learning models and an intuitive user interface. By leveraging fine-tuned DistilBERT models for category and sub_category prediction, the app delivers precise and contextually relevant classifications. The implementation emphasizes robustness and user-friendliness, ensuring a seamless prediction experience while providing insights into the training process through loss and accuracy visualizations. The prediction pipeline, equipped with efficient tokenization, inference, and label mapping, offers accurate results with a logical flow, making it ideal for handling real-world classification tasks.

Beyond its core functionality, the application demonstrates the value of coupling deep learning with effective data visualization. The inclusion of training graphs and confusion matrices highlights the model's performance, aiding in transparency and trustworthiness. This end-to-end solution is not only an effective classification tool but also a learning platform for users to understand and engage with the intricacies of machine learning models. It sets a solid foundation for expanding into more complex NLP tasks while maintaining an accessible and engaging interface.

References for code and working

Scan this QR code for the GitHub Repository containing all codes related to the project

Scan this QR code for the Demo Video

Bibliography

1. https://huggingface.co/transformers/v3.0.2/model_doc/distilbert.html
2. https://indiaai.gov.in/article/indiaai-launches-cyberguard-ai-cybercrime-prevention-hackathon

THANK YOU