3D Environment Based Intelligent Systems Engineering of Advanced Systems (B.A.T.M.A.N.)

Project Report

Team AI/UC

Presented By:

Chandra Prakash, Matriculation No. 6896945
Ilyaz Khan Mohammed, Matriculation No. 6863945
Kirtika Yadav, Matriculation No. 6882579
Kruthika Hosaballi YajnanarayanaBhat, Matriculation No. 6882491
Yashwanth Tadakamalla, Matriculation No. 6896890

Supervisor: Aschot Kharatyan

Paderborn, 31.03.2021

FRAUNHOFER IEM
Zukunftsmeile 1
D-33102 Paderborn

TABLE OF CONTENT

1 INTRODUCTION

1.1 Problem Statement:

In this growing world where Automation is everywhere making human life easier wherever it is possible. there has been a significant shift in the way systems are being perceived from a development perspective. Over the past few decades, we have seen the significant shift the technologies and how the systems are becoming more and more sophisticated in their functionalities and the way the process. Systems have reached a new level that could not have been imagined a few years ago. This states that the systems are becoming more and more complex and more connected to each other systems. What does this complexity mean in terms of system design and development? It increases the work of the system developers. While developing any system the developers need different kind of knowledge for accomplishing the required tasks like design, security, planning etc. This demonstrates that currently here we need a system or application that helps the developers while developing a system by providing the required information related to security and other aspects. It would be very hectic work for developers to find various resources to gain the knowledge about the system.

1.2 Objective:

There are many possibilities that an autonomous system can be hacked and do things which can create harm and fail the system. So, here we proposed an application "BATMAN THREAT EXTRACTOR" which makes the life of the developers easy by providing the required knowledge about security while developing an autonomous system. By using this application, the developers can gain the knowledge about the threats / attacks that might occur in future to his system. The developers can know how the threat is performed or implemented and what are the consequences that occur if a threat take. By this information, the developers can build a system which is not vulnerable to any kind of security attacks. Developers can see the simulation of the attack/threats occurred and can even identify new threats using the sub application created by UNITY.

1.3 Approach:

For our aim to accomplish we have researched on few Natural Language Processing models. We need these models in our project because here we are collecting the unstructured data from the internet, blogs, and some published papers. NLP pretrained models are used to extract the insights from those unstructured data. So, we have selected 3 state-of-art NLP models. They are *spaCy* which is an open-source software library for advanced natural language processing, and Bidirectional Encoder Representations from Transformers (*BERT*) is a Transformer-based machine learning technique for natural language processing (NLP) pre-training developed by google and *XLnet* is a generalized autoregressive. pretraining method. These are all three pretrained models we used in our research phase. Here we are concentrating on Named entity recognition. We implemented NER on all the three models and compared with each other to choose the best model for our final implementation and the UNITY application that visualizes and configures the reference scenarios related to threats / attacks is made using UNITY 3d engine and using some unity assets. In further sections

we will explain you detail how we implemented and how we choose and which technologies and tools we used for web application and UNITY application.

2 MODEL EXPLANATION

2.1 Dataset Preparation

To prepare a Dataset, the data related to autonomous security, cyber security in unstructured form was collected from sources like articles, journal papers, IEE papers, news blogs etc. These datasets required preprocessing according to the model by which the pretrained models would learn and was trained to predict the output.

After collecting the data, we start preprocessing of data. Here each word of the sentence needs to be tagged with respective Parts of speech and a tag. Part of Speech (hereby referred to as POS) Tags are useful for building parse trees, which are used in building NERs (most named entities are Nouns) and extracting relations between words. POS Tagging is also essential for building lemmatizers which are used to reduce a word to its root form. POS tagging is the process of marking up a word in a corpus to a corresponding part of a speech tag, based on its context and definition. This task is not straightforward, as a particular word may have a different part of speech based on the context in which the word is used. By knowing the word is whether a noun, a verb, an adjective and so on. Knowing the role of each word in the sentence will help us start to figure out what the sentence is talking about. For tagging we have used BIO system where 'B' stands for "Beginning" and 'I' stands for "Interior" and 'E' stand for "End" For example "United States of America" as the name of a country.

We will tag It like:

United (B-Country)
States(I-Country)
Of(I-Country)
America (E- Country)

Here is an example:

Sentence #	Word	POS	Tag
Sentence:1	A	DT	O
	hacker	NN	B-PER
	who	WP	O
	swipes	VBZ	O
	a	DT	O
	relatively	RB	O
	inexpensive	JJ	O
	Proxmark	NNP	B-ATS
	RFID	NNP	I-ATS
	reader	NN	E-ATS
	/	SYM	O
	transmitter	NN	B-ATS
	device	NN	E-ATS
	near	IN	O
	the	DT	O
	key	JJ	B-TAR

Figure 2.1: Dataset table

Here as mentioned above the data is preprocessed by adding the respective POS and tags. Here we can see different types like TMS, PER, ATS etc.
We have used custom data for application, so we also made custom tags for our dataset.
We have used 19 tags. They are:

- Person- PER
- Animal- AML
- Technical Mobility System - TMS
- Another Technical System - ATS
- Attack – ATK
- Consequences - CONS
- Target- TAR
- HAZARD - HAZ
- Brute Force - SBF
- Denial Of Service - DOS
- Spoofing - SPF
- Natural Cause - NC
- Information disclosure - IFD
- Tampering - TMP
- Relay Attack - RA
- Hijacking- HJK
- Phishing - PHG
- Physical Cause - PC
- Cyber Attack – CYB

Testing Data:
For testing the model, we separately collected the sentences and articles and tagged them accordingly to the train data. But this data won't be used to train the model. So, the model can predict on new data.

2.1.1 Classification Dataset

- Explored research papers, articles to retrieve news related to automobiles threat and non-threat news. The dataset has a size of 200 news related to threat and non-threat. This is used to train BERT model for news classification.
- Test dataset is also prepared with a size of 20 sentences to check the performance.

2.1.2 Named Entity Recognition:

Named Entity Recognition (NER) – sometimes referred to as entity chunking, extraction, or identification. The task of the NER is to identify and categorize the key information (entities) that present in text [1]. An entity can be any word or series of words that constantly representing the same thing. Every detected entity is classified into a predetermined category. For example, an NER machine learning model may detect the word "Ford" in a text and classify it as a "company".

NER is a form of natural language processing (NLP), a subfield of artificial intelligence. NLP is concerned with computers processing and analysing natural language.

How NER works?

Figure 2.1.2: Example of NER

The core of any NER model is a twostep process [1]:

1. Detect a named entity.
2. Categorize the entity.

In Step1, it involves detecting a word or string of words that can be form an entity. Each word represents a token "The Great Lakes" is a string of three tokens that represents one entity. Inside-outside-beginning tagging is a common way of indicating where entities begin and end [1].

In the Step2, it requires the creation of entity categories. For Example

- Person – Yashwanth, Andrew, Damon
- Organization – Google, University of Oxford
- Time – 2006,15:44pm
- Location – Paderborn, Machu Picchu

Applications of NER:

- Human resources -- Speed up the hiring process by summarizing applicants' CVs.
- Search and recommendation engines -- Improve the speed and relevance of search results and recommendations by summarizing descriptive text, reviews, and discussions [1]
- Content Classification --surface content more easily and gain insights into trends by identifying the subjects and themes of blog posts and news articles.

- **Dataset Pre-Processing and Construction Applications**

It was a hazard to manually create databases. It was an Idea to automate most of it. In the exploration, Two Application were developed.

1. Classified Data Conversion Application
2. Dataset Construction Application

Following is the process diagram for Dataset Preparation Process:

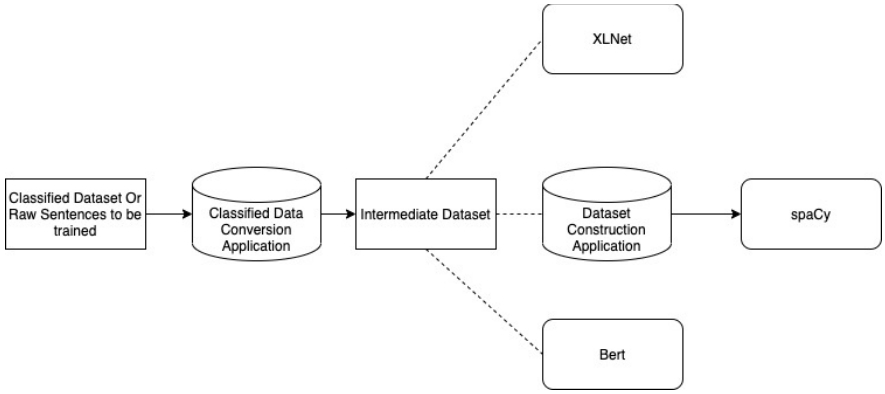


Figure 2.1.2 (a): Dataset Preparation Process Diagram

For the development of these applications JavaFX Technology is used.

JavaFX

JavaFX is used for developing rich client applications. It provided a feasible GUI. It is compatible with all Java supporting devices. So, Platform indecency is also taken care with it. JavaFX provides the development of Java applications with a modern, hardware-accelerated user interface that is highly portable.

Some Highlights about JavaFX:

JavaFX supports Java Lambda expressions in action-triggering events.
Using Lambda expressions, JavaFX promotes code readability and just needs fewer lines of code.
JavaFX is a good alternative GUI for Android and iOS, meaning JavaFX is portable now.
JavaFX uses hardware accelerated graphics pipeline for the rendering, known as *Prism*.
JavaFX has a platform dependent *Glass* windowing toolkit layer to connect to the native operating system. It uses the operating system's event queue to schedule thread usage. Also, it asynchronously handles windows, events, timers.

Architecture Diagram of Dataset Preparation Applications:

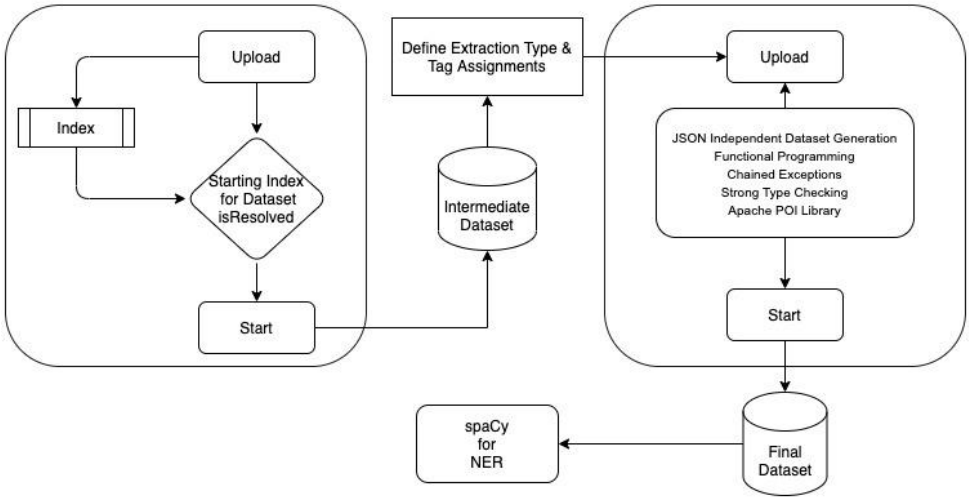


Figure 2.1.2 (b): Architecure of Dataset Preparation Application

2.1.3 Classified Data Conversion Application

It helps in generating a dataset to further process with SpaCy, XLNet and BERT. We take input from Classified Dataset or New Raw Input. Then it is processed to generate an intermediate Dataset.

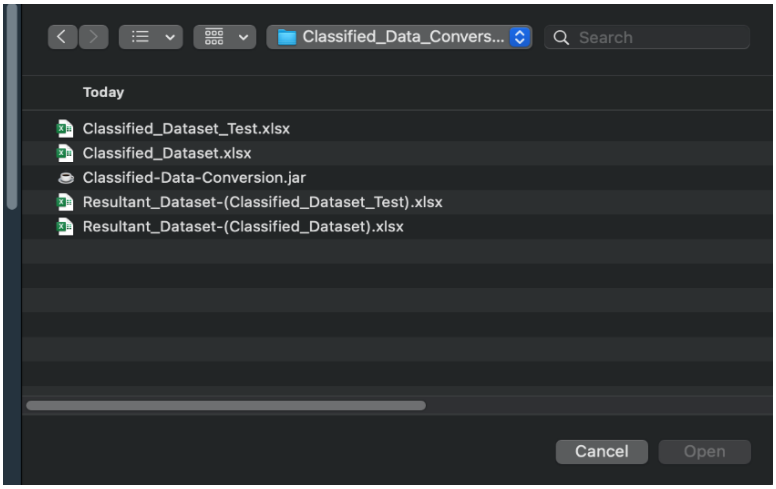
It can be used with the following steps:

1. Open the "Classified-Data-Conversion.jar"

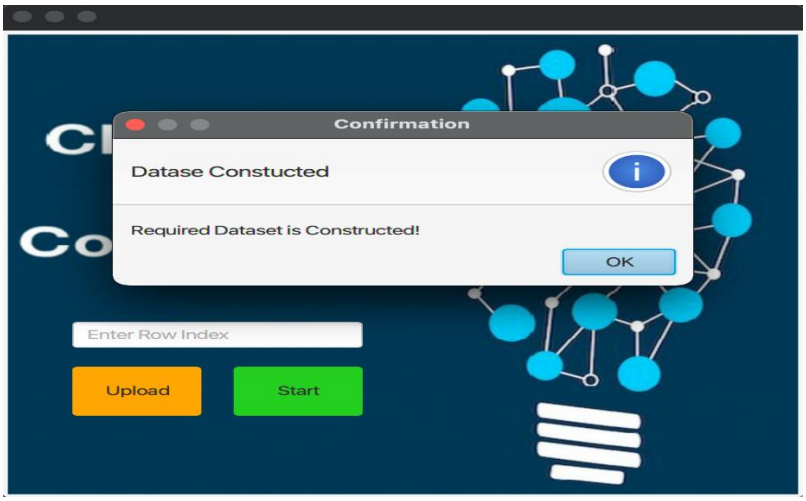
   In any case if it doesn't open, the please open console and go to the folder location where it exists. After that please execute the following command on console: java -jar Classified-Data-Conversion.jar.

2. Click on Upload.

3. Select the Classified Dataset or Custom file where raw data is available
4. Click on Start



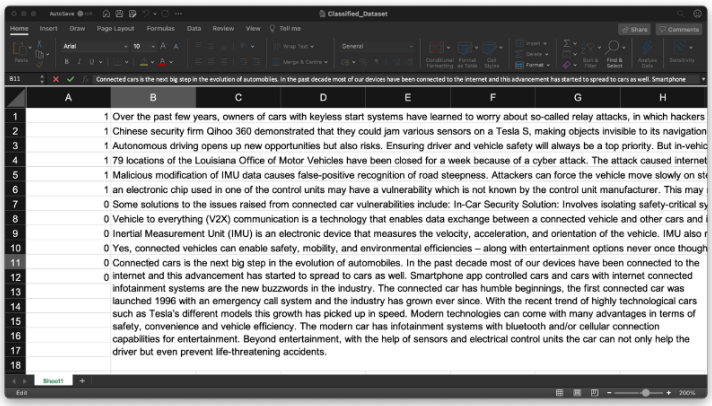
5.Collect the generated Dataset



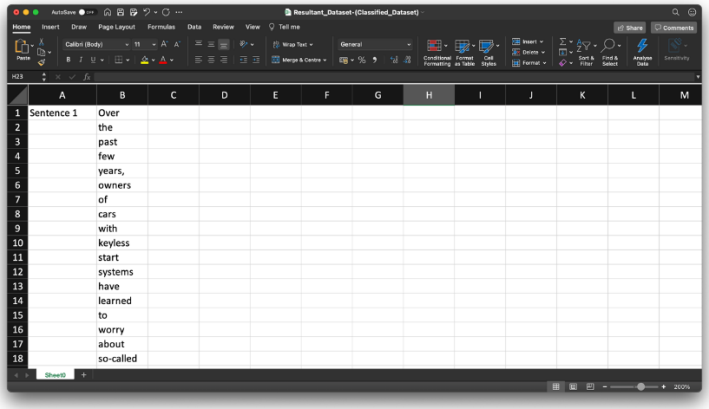
6. Collect the generated Dataset

This will create intermediate Dataset, which further can be prepared for NER training to spaCy, XLNet and BERT.

Input: Classified Dataset or New Raw Source Data



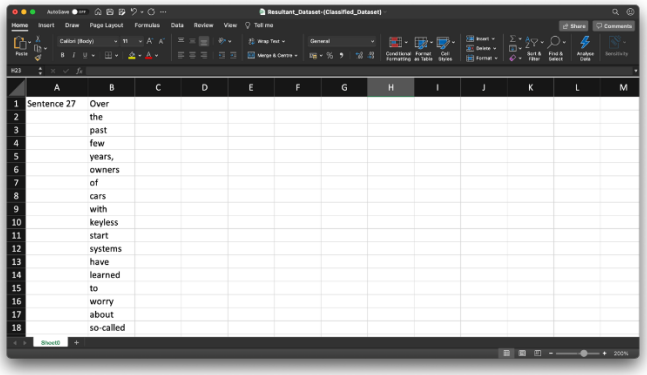
Output: Intermediate Dataset



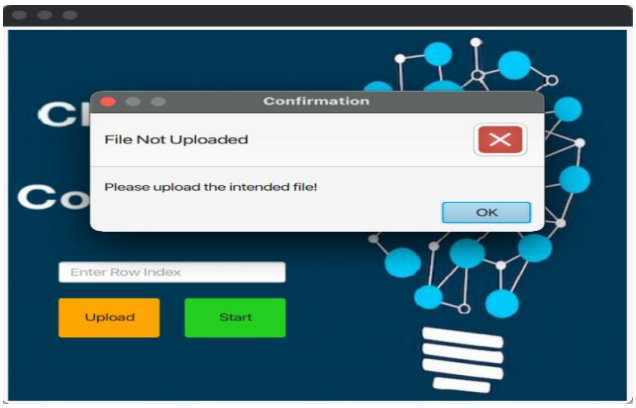
	A	B
1	Sentence 1	Over
2		the
3		past
4		few
5		years,
6		owners
7		of
8		cars
9		with
10		keyless
11		start
12		systems
13		have
14		learned
15		to
16		worry
17		about
18		so-called

There is a special situation when we required to add new data to existing intermediate dataset. If we create add new dataset ordinary. Then new sentences will start from 1 index. It again kills a lot of time to order them. In order to resolve that Index feature was brought in. This is like "**Start From Anywhere**" feature for Dataset Preparation. In index input we can provide the starting index. i.e. There is 27 value added in following snippet.

This will create a intermediate dataset with starting index as 27 (Provided input). Now this data can be simply added to existing dataset and processed for training.



If by any chance Input is not provided and directly clocked on start. Then it prompts an Information to upload the input.



Some Important working code snippet for “Classified Data Conversion Application”:

An insight of Functional Programming, Lambda Expression and Addtion of Instruction Types

```
start.setOnAction(e -> {
    try {
        Boolean hadFile = dc.construct();
        if(hadFile) {
            Alert alert = new Alert(AlertType.INFORMATION);
            alert.setTitle("Confirmation");
            alert.setHeaderText("Datase Constucted");
            alert.setContentText("Required Dataset is Constructed!");
            alert.showAndWait();
        }
        else
        {
            Alert alert = new Alert(AlertType.ERROR);
            alert.setTitle("Confirmation");
            alert.setHeaderText("File Not Uploaded");
            alert.setContentText("Please upload the intended file!");
            alert.showAndWait();
        }
    } catch (IOException e1) {
        e1.printStackTrace();
    }
});
```

To generate results:

```
private static void generateResult(File sourceFile) throws IOException {
    try (FileInputStream file = new FileInputStream(sourceFile);
          XSSFWorkbook newWorkbook = new XSSFWorkbook();
          XSSFWorkbook workbook = new XSSFWorkbook(file);) {
        XSSFSheet sheet = workbook.getSheetAt(0);
        Iterator<Row> rowIterator = sheet.iterator();
        XSSFSheet newSheet = newWorkbook.createSheet();

        int newRowCount = 0;
        XSSFRow newRow;
        while (rowIterator.hasNext())
        {
            try {
                Row row = rowIterator.next();
                String eachRowValue = row.getCell(1).toString();
                newRow = newSheet.createRow(newRowCount);
                newRow.createCell(0).setCellValue("Sentence "+ sentenceCount);
                sentenceCount++;

                String[] tokenisezWords = eachRowValue.split(" ");

                for(String eachWord : tokenisezWords) {
                    newRow.createCell(1).setCellValue(eachWord.replaceAll("[.]", ""));
                    newRowCount++;
                    newRow = newSheet.createRow(newRowCount);
                }
            }catch (Exception e) {
                e.printStackTrace();
            }
        }
        saveResult(sourceFile.getName() ,newWorkbook);
    }
}
```

To save results:

```
private static void saveResult(String targetFileName, XSSFWorkbook newWorkbook) throws IOException {
    String refiledTargetName = "Resultant_Dataset~("+ targetFileName.split("\\.")[0] + ").xlsx";
    try(FileOutputStream fileOut = new FileOutputStream(Paths.get(refiledTargetName).toString())){
        newWorkbook.write(fileOut);
    }
}
```

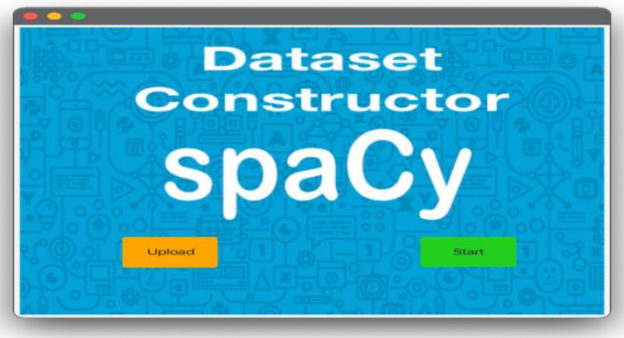
Git Link for Application [1]

Source Code Link [2]

2.1.4 Dataset Construction Application

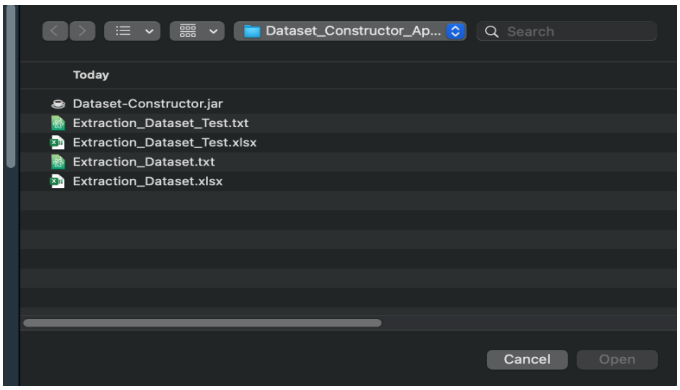
1.Open the “Dataset-Constructor.jar”
In any case if it doesn’t open, the please open console and go to the folder location where it exists. After that please execute the following command on console: java -jar Dataset-Constructor.jar

2.Click on Upload



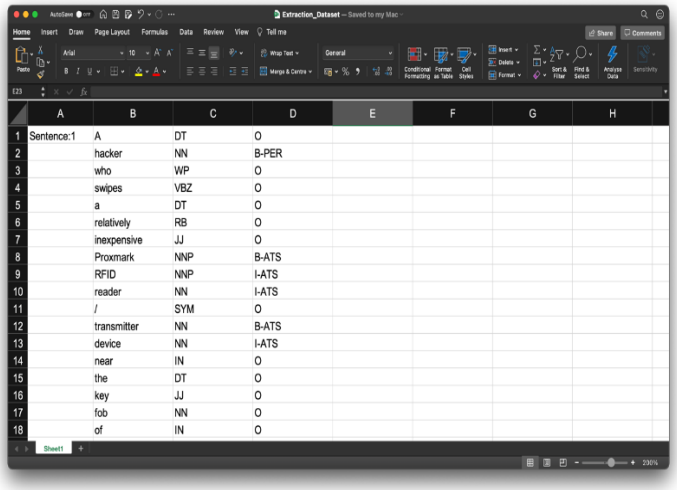
3. Select the Prepared Common Dataset in order to generate Dataset for spaCy training

4. Click on Start



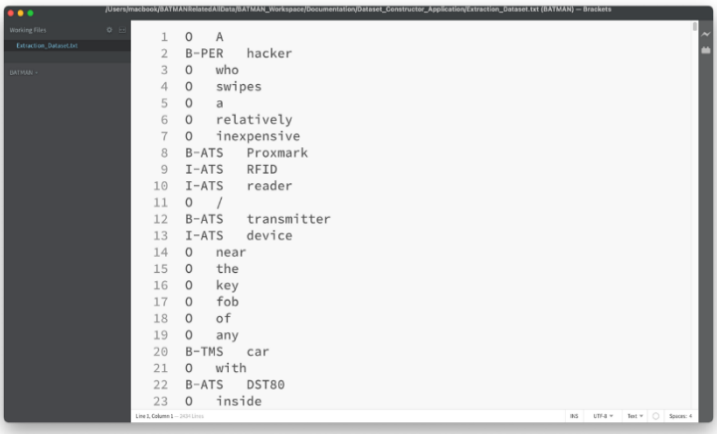
5. Once the Dataset is successfully created then a new window is prompted to inform the user that everything went well

Input: Intermediate Dataset after Defining Extraction Type and Tag Assignments



	A	B	C	D	E	F	G	H
1	Sentence:1	A	DT	O				
2		hacker	NN	B-PER				
3		who	WP	O				
4		swipes	VBZ	O				
5		a	DT	O				
6		relatively	RB	O				
7		inexpensive	JJ	O				
8		Proxmark	NNP	B-ATS				
9		RFID	NNP	I-ATS				
10		reader	NN	I-ATS				
11		/	SYM	O				
12		transmitter	NN	B-ATS				
13		device	NN	I-ATS				
14		near	IN	O				
15		the	DT	O				
16		key	JJ	O				
17		fob	NN	O				
18		of	IN	O				

Output: Final Dataset Compatible to spaCy



```
1  O   A
2  B-PER   hacker
3  O   who
4  O   swipes
5  O   a
6  O   relatively
7  O   inexpensive
8  B-ATS   Proxmark
9  I-ATS   RFID
10 I-ATS   reader
11 O   /
12 B-ATS   transmitter
13 I-ATS   device
14 O   near
15 O   the
16 O   key
17 O   fob
18 O   of
19 O   any
20 B-TMS   car
21 O   with
22 B-ATS   DST80
23 O   inside
```

If by any chance Input is not provided and directly clocked on start. Then it prompts an Information to upload the input.

To generate results:

```
private static void generateResult(File sourceFile) throws IOException {
    try (InputStream fileInputStream = new FileInputStream(sourceFile);
            XSSFWorkbook workbook = new XSSFWorkbook(fileInputStream)) {
        XSSFSheet sheet = workbook.getSheetAt(0);
        Iterator<Row> rowIterator = sheet.iterator();
        StringBuilder stringBuider = new StringBuilder();
        while (rowIterator.hasNext())
        {
            try {
                Row row = rowIterator.next();
                Cell sentenceColumnCell = row.getCell(0);
                if(Objects.nonNull(sentenceColumnCell)) {
                    String sentenceColumn = row.getCell(0).getStringCellValue();
                    if(Objects.nonNull(sentenceColumn) && sentenceColumn.startsWith("Sentence")) {
                        if(stringBuider.length() != 0)
                            stringBuider.append("\n");
                    }
                }
                String tagColumn = getCellStringValue(row.getCell(3));
                String wordColumn = getCellStringValue(row.getCell(1));
                stringBuider.append(tagColumn + "\t" + wordColumn + "\n");
                System.out.println(row.getRowNum());
                System.out.println(tagColumn + "\t" + wordColumn + "\n");
            }catch (Exception e) {
                e.printStackTrace();
            }
        }
        saveResult(sourceFile.getName() ,stringBuider);
    }
}
```

To Handle Exceptions and Type Checking:

```
private static String getCellStringValue(Cell cell) {
    Objects.requireNonNullElse(cell, "");
    String cellStringValue = "";
    try{
        cellStringValue = cell.getStringCellValue();
    } catch (IllegalStateException ex1) {
        try {
            Number cellNumberVlaue = cell.getNumericCellValue();
            if((cellNumberVlaue.doubleValue() % 1) == 0) {
                cellStringValue = String.valueOf(cellNumberVlaue.intValue());
            }
            else {
                cellStringValue = cellNumberVlaue.toString();
            }
        } catch (IllegalStateException ex2) {
            try {
                Boolean cellBooleanVlaue = cell.getBooleanCellValue();
                cellStringValue = cellBooleanVlaue.toString();
            } catch(IllegalStateException ex3) {
                Date cellDateValue = cell.getDateCellValue();
                cellStringValue = cellDateValue.toString();
            }
        }
    }
    return cellStringValue;
}
```

To save results:

```
private static void saveResult(String targetFileName, StringBuilder stringBuider) throws IOException {
    String refiledTargetName = targetFileName.split("\\.")[0] + ".txt";
    Files.write(Paths.get(refiledTargetName), Arrays.asList(stringBuider.toString()));
}
```

Additional information regarding Dataset Applications development:

- Build & Deployed Version: Java 15
- Maven Project
- Apache POI Library is used
- Functional Programming & Lambda Expression
- Chain of Exceptions code is written to have a strong Type Checking and Exception Handling

Git Link for Application [3]

Source Code Link [4]

2.1.5 Data Pre-Processing and POS Tagging Script in Python

File Name and file path: Dataset_Preparation_by_breaking_sentences_and_tagging.ipynb for code and for dataset test_dataset.csv in AI-UC_2021->Dataset->Dataset_Preparation_And_Preprocessor_Scripts-> Dataset_Preparation_by_converting_Csv_to_txt folder.

Task: Given the news content, we need to pre-process, tokenize, and add POS tags to the resultant CSV, which will be further used for training BERT, XLnet and Spacy models by following project code standards (written in python).

Input: Dataset csv containing sentences to be pre-processed for classification by POS tagging the significant words in the sentence.

Output: Tokenized significant words with POS tagging's

Description: Given the news content/data, we pre-processed the data, by using NLTK library of python. NLTK is a NLP utility library containing built-in NLP methods and technologies. It provides text processing libraries for classification, tokenizing, stemming etc. We tokenized the data using *nltk.word_tokenize()* method of NLTK. This gives us the breakdown of the news sentences, and makes the data ready for tagging.Now we apply POS tagging to these tokens, the method used for POS tagging is *nltk.pos_tag(words).* We do this to prepare and enable the dataset to be ready to feed into BERT and XLNET models. Steps for the execution are as follows:

1. Upload the code in Jupyter or Google Colab.
2. Create a directory for data using !mkdir data and upload the dataset that needs to be pre-processed for NER dataset.
3. Mention the dataset name in read.csv command.
4. Execute the complete code.

```
!mkdir data
```

```
mkdir: cannot create directory 'data': File exists
```

```
[ ] import pandas as pd
    import numpy as np
    from tqdm import tqdm, trange

    data = pd.read_csv("data/test_dataset.csv", engine='python', encoding="utf8").fillna(method="ffill")
    outputFile = open(r'data/final_file_9.csv', "w+")
```

Figure 2.1.5: Creating data directory for uploading of raw sentence dataset and mentioning file name.

	Standard	Standard
1	1	A hacker who swipes a relatively inexpensive Proxmark
2	1	Over the past few years, owners of cars with keyless
3	0	After the first big round of e-commerce hacks, Bill
4	0	security experts are constantly exploring new defense
5	1	Chinese security firm Qihoo 360 demonstrated that the
6	1	Attackers can evade state-of-the-art face recognition
7	1	Those automated taxis or delivery vehicles could be v
8	1	When hackers demonstrated that vehicles on the roads

Figure 2.1.5 (a): Input File for Data Pre-processor script.

	Standard	Standard	Standard
1	Sentence1	Over	IN
2		the	DT
3		past	JJ
4		few	JJ
5		years	NNS
6		,	,
7		owners	NNS
8		of	IN
9		cars	NNS
10		with	IN
11		keyless	JJ
12		start	NN
13		systems	NNS
14		have	VBP
15		learned	VBN
16		to	TO
17		worry	VB
18		about	IN
19		so-called	JJ
20		relay	NN
21		attacks	NNS
22		,	,
23		in	IN
24		which	WDT
25		hackers	NNS
26		exploit	VBP
27		radio-enabled	JJ
28		keys	NNS
29		to	TO
30		steal	VB

Figure 2.1.5 (b): Output file for data pre-processor script.

Git Path: In Main Branch follow link: [5]

2.1.6 Generic Data Processing Script in Python

File Name and file path: csv_to_txt.ipynb for code and for dataset name is DemoDataSet.csv Path:AI-UC_2021->Dataset->Dataset_Preparation_And_Preprocessor_Scripts-> Dataset_Preparation_by_converting_Csv_to_txt folder.

Task: convert a data csv file to a Spacy supported format, i.e., text format with some of the features cropped and the attributes re-arranged to be compatible with Spacy's framework.

Input: CSV file from BERT etc. with the columns/features consisting of words, sentences, classification category, POS tags.

Output: .txt file with re-arranged features and removed pos-tags, making sure the special characters are also intact and proper tokenization and data integrity is intact.

Description: We have written a utility for converting a column based .csv file into a .txt format, such that the txt file is compatible to be used with the Spacy models for further classifications. The major challenge was to first split the input csv files' data lines containing "[,] comma as a data" as well as separator, and create valid data lists for further processing. Another challenge was to filter out the POS-tags from the data lists and then rearrange the columns/features to make the output file, compatible to be sent as a valid input to Spacy model framework. Steps for execution of code is as follows:

1. Upload the code in Jupyter or Google Colab.
2. Create a directory for data using !mkdir data and upload the dataset that needs to be pre-processed for NER dataset
3. Mention the dataset name in read.csv command.
4. Execute the complete code.

```
import re
import ntpath
```

```
[ ] import pandas as pd
    import csv
```

```
[ ] !mkdir data
```

```
[ ]
    import numpy as np
    from tqdm import tqdm, trange

    data = pd.read_csv("data/DemoDataSet.csv", engine='python', encoding="utf8").fillna(method="ffill")
```

Figure 2.1.6: Creating data directory for uploading of Prepared NER dataset and mentioning file name for conversion of csv to text file for specific model used for NER task to be trained.

	Standard	Standard	Standard	Standard
1	Sentence #	Word	POS	Tag
2	Sentence:1	A	DT	O
3		hacker	NN	B-PER
4		who	WP	O
5		swipes	VBZ	O
6		a	DT	O
7		relatively	RB	O
8		inexpensive	JJ	O
9		Proxmark	NNP	B-ATS
10		RFID	NNP	I-ATS
11		reader	NN	E-ATS
12		/	SYM	O
13		transmitter	NN	B-ATS
14		device	NN	E-ATS
15		near	IN	O
16		the	DT	O
17		key	JJ	B-TAR
18		fob	NN	E-TAR
19		of	IN	O
20		any	DT	O
21		car	NN	B-TMS
22		with	IN	O
23		DST80	NNP	B-ATS
24		inside	RB	O
25		can	MD	O
26		gain	VB	O
27		enough	JJ	O
28		information	NN	O
29		to	TO	O
30		derive	VB	B-CONS

Figure 2.1.6 (a): Input file for csv to txt script.

```
O       A
B-PER   hacker
O       who
O       swipes
O       a
O       relatively
O       inexpensive
B-ATS   Proxmark
I-ATS   RFID
E-ATS   reader
O       /
B-ATS   transmitter
E-ATS   device
O       near
O       the
B-TAR   key
E-TAR   fob
O       of
O       any
B-TMS   car
O       with
B-ATS   DST80
O       inside
O       can
O       gain
O       enough
O       information
O       to
B-CONS  derive
I-CONS  its
I-CONS  secret
I-CONS  cryptographic
E-CONS  value
O       .
O       That
O       ,
O       in
O       turn
O       ,
O       would
O       allow
O       the
B-PER   attacker
O       to
O       use
O       the
O       same
B-ATS   Proxmark
E-ATS   device
O       to
B-ATK   impersonate
I-ATK   the
E-ATK   key
```

Figure 2.1.6 (b): Output File after converting csv to txt for spacy model.

Git Path: In main Branch follow link: [6]

2.2 Implementation of Models for NER and Comparison of Models

2.2.1 XLnet

Why XLnet?

As we have seen above the one of the pretrained model "BERT" which stand for "Bidirectional Encoder Representations Form Transformers". It is a neural network architecture that can model bidirectional contexts in text data using Transformers.
As BERT works in bidirectional and implements traditional methods to predict the current token given previous n tokens, or predict the current token given all tokens after it. The Bidirectionality is achieved by a phenomenon named as "Masked Language Modelling". In Masked Language Modeling (MLM) we can achieve bidirectionality by making 15% of tokens in the input sentences as masked. The masking will be done in random order. Then the transformer is trained to predict the masked words [2]. By this technique there are few disadvantages/limitations to BERT.

The two main limitations of BERT algorithm are:

1. BERT corrupts the input data with masks and suffers from pretrain finetune discrepancy.
2. BERT neglects the dependency between masked positions. For example, consider the sentence "United States of America is a country" and input to BERT to be "[Mask] [Mask] of [MASK] is a country". The objective of BERT would be.

- "log p (United | is a country) + log p(States | is a country) + log p(America | is a country).

From this function we can say that there is no dependency between learnings "United", "States" and "America". So, BERT can result in a prediction like "United York America is a city" [2]. We can overcome all these limitations by using another pretrained model "XLNET". Before we dive into what is XLNET we need to know some key integrations that are integrated from Transformer -XL [2].

What is Transformer-XL Architecture?

XLNET integrates few ideas from Transformer-XL, which is the state-of-the-art autoregressive model into pretraining. Basically, Transformer models are used for language translation purposes by google. It is an encoder-decoder model where you can map one sequence to another i.e., English to French. While translating a sentence in English to French, the decoder needs to look at the entire English sentence to selectively extract the needed information from it at any point of time. So, all the hidden states of the encoder are made available to the decoder. So, there will be no masking used in transformer-XL.

But how does the decoder know which hidden states it should look up at any point?

This can be done by weighting each of the hidden states of the encoder. These weights are determined by using a simple feed forward neural network. These are called attention weights, or values in the terminology. We can find some the terminologies that are useful are:

- Query (Q) – decoder's hidden state.
- Keys (K) – encoder's hidden state.
- Values (V) – attention weights when processing a query.

There are two main things that are integrated from Transformer-XL to XLNET.

1. Positional Encoding – which helps in keep track of the positions of each token in a sequence.
2. Segment recurrence – it caches the hidden state of first segment in memory in each layer and update attention respectively. It helps us to reuse of memory for each segment.

Now, we are good to know about XLNET.

What is XLNET?

XLNet is a new unsupervised language representation learning method based on a novel generalized permutation language modeling objective. [git] XLNet is a generalized autoregressive pretraining method that.

- Enables learning bidirectional contexts by maximizing the expected likelihood over all permutations of the factorization order.
- overcomes the limitations of BERT by its autoregressive formulation.

- Relative positional embeddings and recurrence mechanism from Transformer XL

Additionally, XLNet employs Transformer-XL as the backbone model, exhibiting excellent performance for language tasks involving long context. Overall, XLNet achieves state-of-the-art (SOTA) results on various downstream language tasks including question answering, natural language inference, sentiment analysis, and document ranking.

Requirements:

Permutation Language Modeling (PLM)
PLM is the idea of capturing bidirectional context by training an autoregressive model on all possible permutation of words in a sentence. Instead of fixed left-right or right-left modelling.[2]

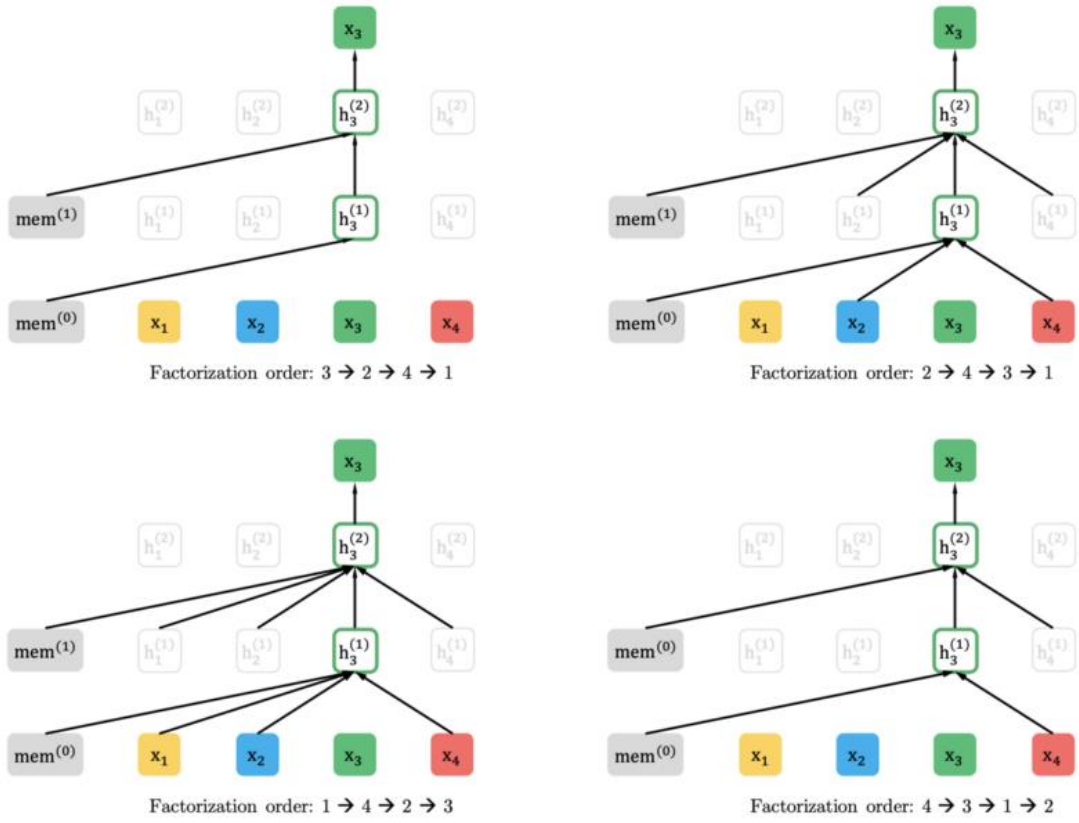


Figure 2.2.1: Illustration of PLM objective for predicting x3 given the same input sequence x but with different factorization orders. [2]

The above diagram illustrates PLM. Let us consider that we are learning x3 (the token at the 3rd position in the sentence). PLM trains an autoregressive model with various permutations of the tokens in the sentence, so that in the end of all such permutations, we would have learnt x3, given all other words in the sentence. In the above illustration, we can see that the next layer takes as inputs only the tokens preceding x3 in the permutation sequence [2]. This way, autoregression is also achieved.

NER PIPELINE:

Figure 2.2.1 (a): XLNet pipeline

Building an NLP Pipeline For NER:

Step1: Sentence segmentation:

The first step in the process is Sentence segmentation. Here we break the text apart into separate sentences. For example

1. A Hacker who swipes a relatively inexpensive Proxmark RFID reader device near the car fob of any car with DST80 inside can gain enough information to derive its secret cryptographic value.
2. That, in turn, would allow the attacker to use the same Proxmark device to impersonate the key inside the car, disabling the immobilizer and letting them start the engine.

We can assume that each sentence in English is a separate thought or an idea.
Coding a Coding a Sentence Segmentation model can be as simple as splitting apart sentences whenever you see a punctuation mark. But modern NLP pipelines often use more complex techniques [3].

Step2: Tokenization:

Now we will import the XLNet Tokenizer, which we use to convert our text into tokens that correspond to XLNet's vocabulary. The XLNet implementation comes with a pretrained tokenizer and a defined vocabulary. We load the one related to the smallest pre-trained model "xlnet-base-cased". We use cased variate since it is well suited for NER.

Since XLNet tokenizer will split tokens in sub word tokens. For example, 'gunships' will be split in the two tokens 'guns' and '##hips'. We have to deal with the issue of splitting our token-level labels to related sub tokens[3].

XLNet requires specifically formatted inputs. For each generated tokenized input sentence, we will create:

- input ids: a sequence of integers identifying each input token to its index number in the XLNet tokenizer Vocabulary
- Segment mask: (optional) a sequence of 1s and 0s used to identify whether the input is one sentence or two sentences long. For one sentence inputs, this is simply a sequence of 0s. For two sentence inputs, there is a 0 for each token of the first sentence, followed by a 1 for each token of the second sentence.
- Attention mask: (optional) a sequence of 1s and 0s, with 1s for all input tokens and 0s for all padding tokens.

Although we can have variable length input sentences, XLNet do not need our input arrays to be the same size. So, here we first choose a maximum sentence length, and then we pad and truncating our inputs until every input sequence is of the same length [3]

To “pad” our inputs in this context it means that if a sentence is shorter than the maximum sentence length, we simply add 0s to the end of the sequence until it is the maximum sentence length. If a sentence is longer that the maximum sentence length, then we simply truncate the end of the sequence, discarding anything that does not fit into our maximum sentence length [3].

We pad and truncate our sequences so that they are all become of length MAX_LEN. We have a utility “pad_sequences” that we use from Keras. It simply handles the truncating and padding the lists.

Then we create the attention masks. Which creates a mask of 1s for each token followed by 0s for padding.

After padding is all done. Now we are set for splitting the data into train and test. We have used train_test_split to split our data into train and testing sets for training.

After splitting we convert all our data into torch tensors, which is the required datatype for our model.

Step3: Train model:

For this task, we first want to modify our pre-trained model to give outputs for NER, and then we want to continue training the model on our custom dataset until that the entire model. Though there are interfaces which are all built on top of a trained model, each has different top layers and output types designed to accommodate their specific NLP task [3].

For this task we use “XLNetForTokenClassification” class for token-level predictions. XLNetForTokenClassification is a fine-tuning model that wraps XLNetModel and adds token-level classifier on the top of the XLNetModel. The token-classifier is a linear layer that takes as input the last hidden state of the sequence. We load the pre-trained xlnet-base-cased model and provide the number of possible labels. As we feed input data, the entire pre-trained XLNet model and the additional untrained classification layer is trained on our specific task.

Before we go to the next step, we must step up the optimizer and add the parameters it should update. We choose AdamW as our optimizer. We also add some weight_decay which helps as regularization to the main weight matrices.

Step4: Fine-Tuning:

Because the pre-trained model layers already encode a lot of information about the language, then training the classifier is relatively not expensive. Sometimes practitioners will opt to “freeze” certain layers when fine-tuning, or to apply different learning rates, apply diminishing learning rates, etc. all to preserve the good quality weights in the network and speed up training (often considerably). if your task and fine-tuning dataset is very different from the dataset used to train the transfer learning model, freezing the weights may not be a good idea. In this step we load our pre-trained XLNet model. “xlnet-base-cased” means the version that has both upper and lower cases letters [3].

After we loaded our model, we need to grab the training hyperparameters from within the stored model. For fine-tuning, many authors recommend the following hyperparameter.

Hparam	RACE	SQuAD	MNLI	Yelp-5
Dropout		0.1		
Attention dropout		0.1		
Max sequence length	512	512	128	512
Batch size	32	48	128	128
Learning rate	2e-5	3e-5	3e-5	2e-5
Number of steps	12K	8K	10K	10K
Learning rate decay		linear		
Weight decay		0.00		
Adam epsilon	1e-6	1e-6	1e-6	1e-6
Layer-wise lr decay	1.0	0.75	1.0	1.0

Figure 2.2.1 (b): It shows the recommended hyperparameters [3]

After this now we can start our training loop. There will be lot of things going on for each pass in our loop we have a training phase and a validation phase. At each pass we do
Training loop:

- Tell the model to compute gradients by setting the model in train mode
- Unpack our data inputs and labels
- Load data onto the GPU for acceleration
- Clear out the gradients calculated in the previous pass. In pytorch the gradients accumulate by default unless you explicitly clear them out.
- Forward pass (feed input data through the network)
- Backward pass (backpropagation)
- Tell the network to update parameters with optimizer.step()
- Track variables for monitoring progress [3]

Evaluation loop:

- Tell the model not to compute gradients by setting the model in evaluation mode
- Unpack our data inputs and labels
- Load data onto the GPU for acceleration
- Forward pass (feed input data through the network)
- Compute loss on our validation data and track variables for monitoring process.

Results after the training and evaluation we get the metrics of the model:

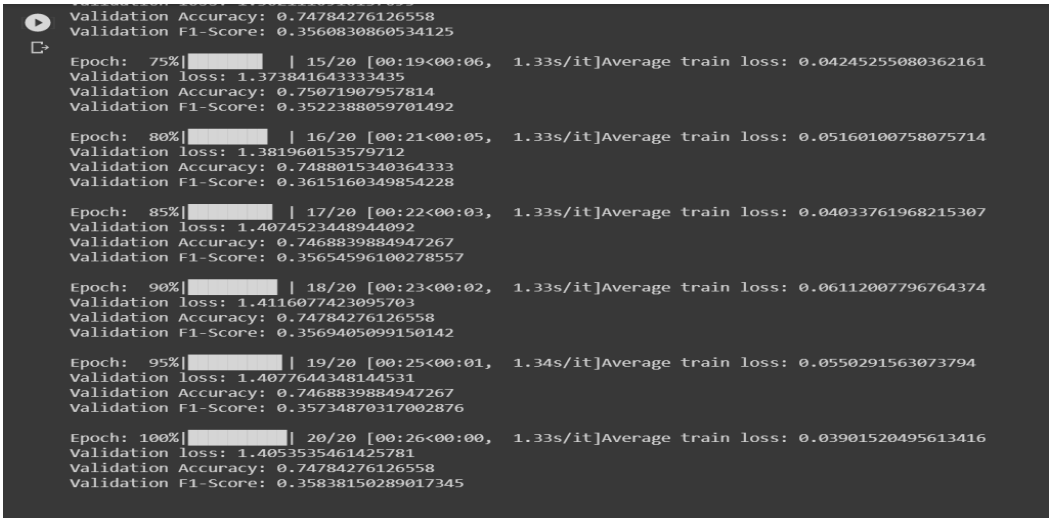


Figure 2.2.1 (c): Results after training and evaluation

By this picture we can see the final Validation loss, Validation accuracy and F1-score of our XLNET NER model.

And here is the visualization.

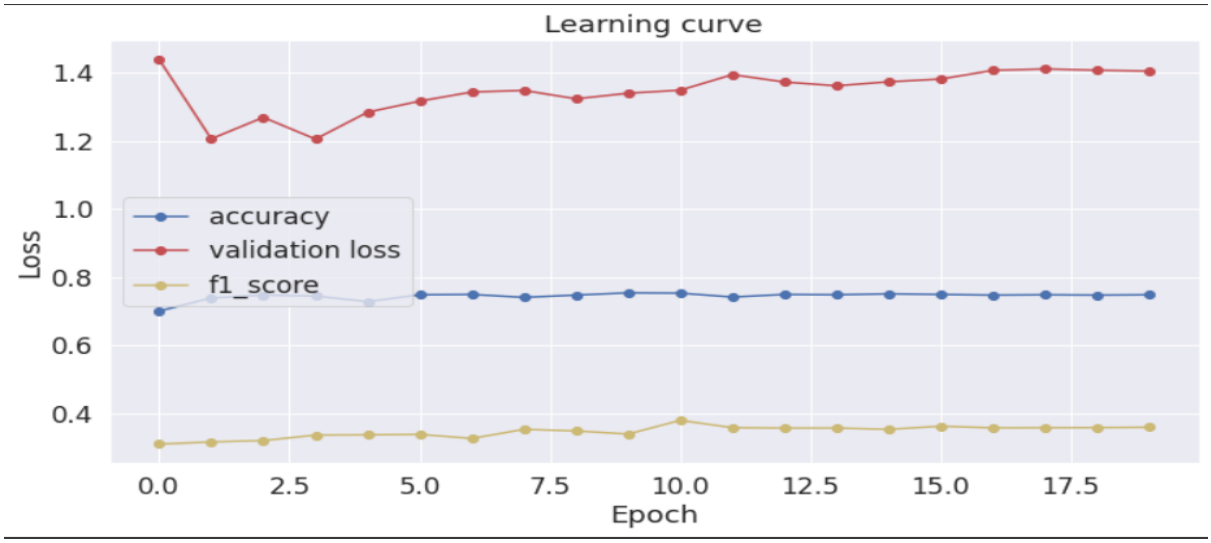


Figure 2.2.1 (d): Graph visualization

Step5: Predictions:

Now after the all the above steps are done now our XLNet model is ready for testing on the new sentences. We can also download our trained model and load it wherever you needed.

2.2.2 BERT

As we discussed in the previous section BERT is a bidirectional model which is a more efficient and powerful general-purpose model that can adapt and work for any text-based task. As we learned that training any model from scratch is time consuming however, now for natural language processing models using BERT, we just have to load a pretrained model and fine tune

the model with the custom data prepared for our specific task. We also learned that the performance of BERT transformer architecture is boosted by combining with pretraining on the Masked Language Modelling task which made BERT a more useful and dominant model in NLP space.

Our main motive is to use our BERT model and fine tune the BERT model for the Named Entity Recognition task to get extraction results for our AI project.

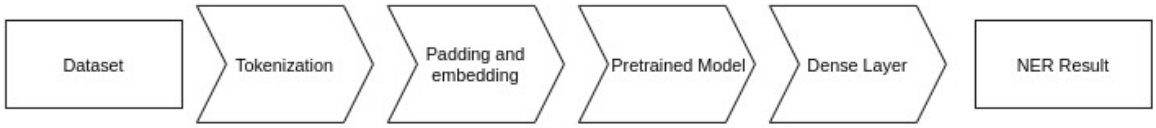


Figure 2.2.2: Basic pipeline for the BERT model used for NER task.

What is Named Entity Recognition (NER):

Named Entity Recognition or NER is a task for entity identification, chunking of entities, extraction and then categorizing the entities (key information) in the text. This means that NER is a subtask that helps to identify the key elements like any name, place and more, extract key information(entities) from the text by locating and helps in determining the category the named entity belongs to.

For such extraction of entities for specific tasks we can use the BERT model. So as the basic pipeline of our BERT model to work is pre-processing of data and preparing a dataset for our pretrained model, load the pretrained BERT model and fine tune it by training the model on a pre-processed custom dataset for a specific task. The steps for the implementation of the BERT model for Extraction of Actor task that is to determine words that can be categorized as actors and such tasks are Named Entity Recognition tasks.

Steps of Implementation

Most important part before implementing an NLP pipeline for BERT NER model, we need to install the transformers package provided by hugging face because of this many transformers libraries can be used including pre-trained BERT models that are in pytorch.

Our next step is to first load the dataset prepared by using Dataset_preprocessor script and adding tags manually for the actor extraction task.

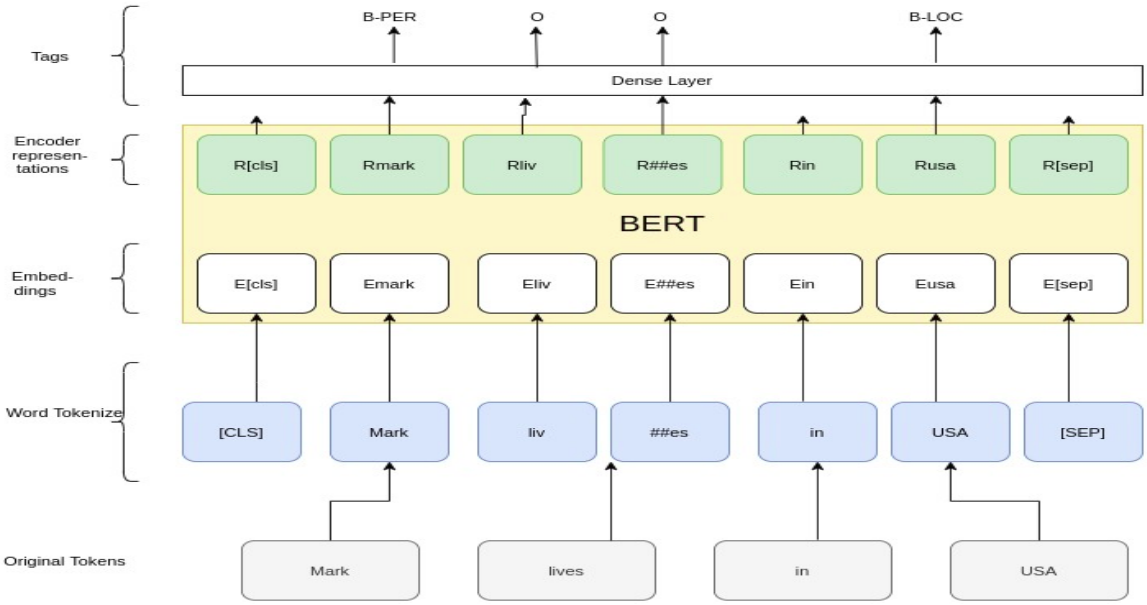


Figure 2.2.2 (a): Architecture for NER task implemented by BERT.

1) Prepare Dataset aligned with BERT For NER

The first step of the NLP pipeline is to load the dataset on which the pre-trained BERT model will be fine-tuned. Therefore, after executing Dataset_preprocessor script that converted the sentences into format that is used for Named Entity Recognition task that is sentence segmentation that break the sentences into separate rows and each word are tagged with the Part of Speech Tags. After this conversion and assigning tags using BIO scheme for each word to perform actor extraction, the following format of dataset is as follows:

	Sentence #	Word	POS	Tag
2390	Sentence 35	as	IN	O
2391	Sentence 35	its	PRP$	O
2392	Sentence 35	relationship	NN	O
2393	Sentence 35	with	IN	O
2394	Sentence 35	hackers	NNS	B-PER
2395	Sentence 35	through	IN	O
2396	Sentence 35	participation	NN	O
2397	Sentence 35	in	IN	O
2398	Sentence 35	hacking	VBG	O
2399	Sentence 35	conferences	NNS	O

Figure 2.2.2 (b): Dataset For training BERT model for NER.

Now after loading the dataset, the next part is to convert the dataset that is acceptable by BERT model for the NER task that is tokenize and embedding. The BERT implementation includes

pretrained tokenize and a defined vocabulary. After tokenizing, padding of tokens and labels to desired length. For this we will limit the sequence length to 75 tokens [6]. After tokenizing, truncating, and converting our dataset to torch tensors and loading the tensors to Data loaders, the next step is to load the pre-trained model for NER.

2) Load Pre-trained Model

Before fine tuning the pre-trained BERT model, we will load our model BERT-based-cased that can handle both upper and lower cases in words. For training such models, it is important to train the top layer classifier with minimum changes to BERT model. So, for this task, the transformers provided the **BERTForTokenClassification class** for token-level predictions. BERT can be used for many tasks by just adding a layer related to task on top of loaded pre-trained model and our task is related to named entity recognition [6]. So, the BERTForTokenClassification is a fine-tuning model that is used to wrap the BERT model and further insert a token-level classifier layer on top of the pre-trained BERT model layer.

3) Fine Tune the model

Before beginning fine tuning, one more step we need to follow is the **setup of AdamW optimizer (**by using this model training with AdamW converges a lot faster and requires less tuning of the learning rate) and adds few parameters for it like learning rate that is used to control the changes in model in response to errors estimated whenever weights are updated [6]. After mentioning all important parameters and values and including metrics to keep track of training processes like using f1 score, the process of fine-tuning by training a pre-trained model on our custom pre-processed dataset can be implemented for 50 epochs. The following are the resultant f1 score:

```
Epoch: 100%|██████████| 48/48 [00:20<00:00,  2.29it/s]Average train loss: 0.02002175711095333
Validation loss: 0.5044547915458679
Validation Accuracy: 0.8821428571428571
Validation F1-Score: 0.5507246376811593
```

Figure 2.2.2 (c): F1 score after training BERT model on custom dataset for 48 epochs.

4) Evaluate Performance

To check how well our model can perform, we tested the model on new sentences to check the prediction result. This was done by downloading the trained model and executing it on a new sentence.

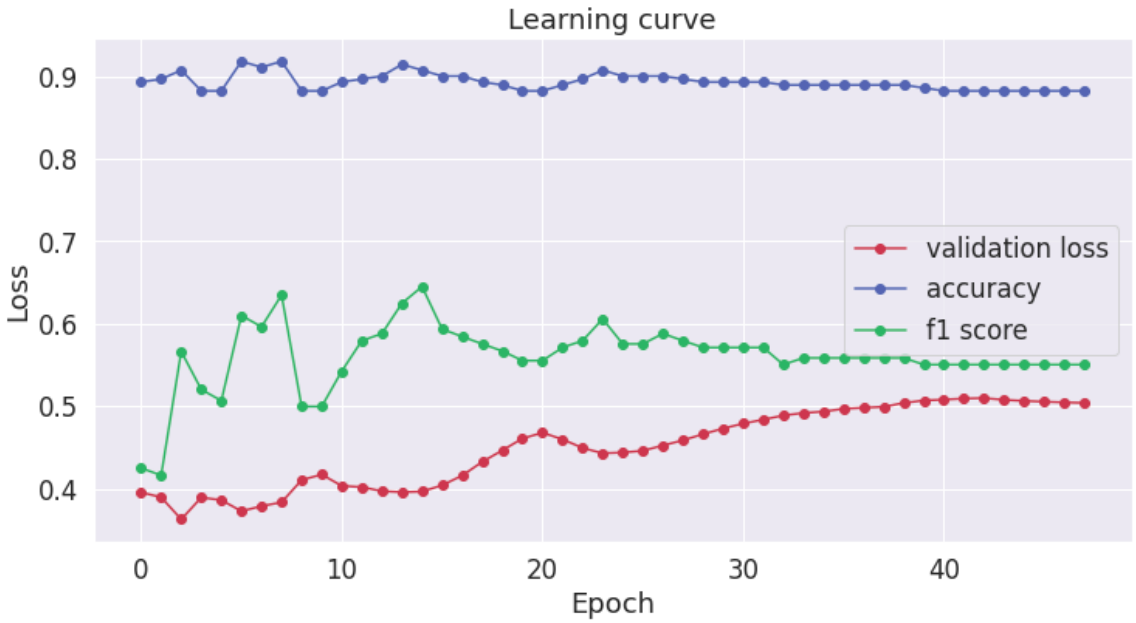


Figure 2.2.2 (d): The graph showing the Trained BERT model performance.

Git Path: In main Branch follow link: [7]

2.2.3 spaCy

spaCy is a free, open-source library for advanced Natural Language Processing (NLP) in Python.

spaCy is designed specifically for production use and helps to build applications that process and "understand" large volumes of text. It can be used to build information extraction or natural language understanding systems, or to pre-process text for deep learning.

Features

spaCy provides many features. A brief list of its provided features is given:

NAME	DESCRIPTION
Tokenization	Segmenting text into words, punctuations marks etc.
Part-of-speech (POS) Tagging	Assigning word types to tokens, like verb or noun.
Dependency Parsing	Assigning syntactic dependency labels, describing the relations between individual tokens, like subject or object.
Lemmatization	Assigning the base forms of words. For example, the lemma of "was" "be", and the lemma of "rats" is "rat".
Sentence Boundary Detection (SBD)	Finding and segmenting individual sentences.
Named Entity Recognition (NER)	Labelling named "real-world" objects, like persons, companies or locations.

NAME	DESCRIPTION
Entity Linking (EL)	Disambiguating textual entities to unique identifiers in a knowledge base.
Similarity	Comparing words, text spans and documents and how similar they are to each other.
Text Classification	Assigning categories or labels to a whole document, or parts of a document.
Rule-based Matching	Finding sequences of tokens based on their texts and linguistic annotations, similar to regular expressions.
Training	Updating and improving a statistical model's predictions.
Serialization	Saving objects to files or byte strings.

Some of its features are explained in detail:

- Tokenization:

During processing, spaCy first tokenizes the text, i.e. segments it into words, punctuation and so on. This is done by applying rules specific to each language. For example, punctuation at the end of a sentence should be split off – whereas "U.K." should remain one token. Each Doc consists of individual tokens, and we can iterate over them:

```
import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("Apple is looking at buying U.K. startup for $1 billion")
for token in doc:
    print(token.text)
```

RUN

Figure:2.2.3 (a)

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Apple	is	looking	at	buying	U.K.	startup	for	$	1	billion

First, the raw text is split on whitespace characters, similar to *text.split(' ')*. Then, the tokenizer processes the text from left to right. On each substring, it performs two checks:

1. Does the substring match a tokenizer exception rule? For example, "don't" does not contain whitespace, but should be split into two tokens, "do" and "n't", while "U.K." should always remain one token.

2. Can a prefix, suffix or infix be split off? For example, punctuation like commas, periods, hyphens, or quotes.

If there's a match, the rule is applied and the tokenizer continues its loop, starting with the newly split substrings. This way, spaCy can split complex, nested tokens like combinations of abbreviations and multiple punctuation marks.

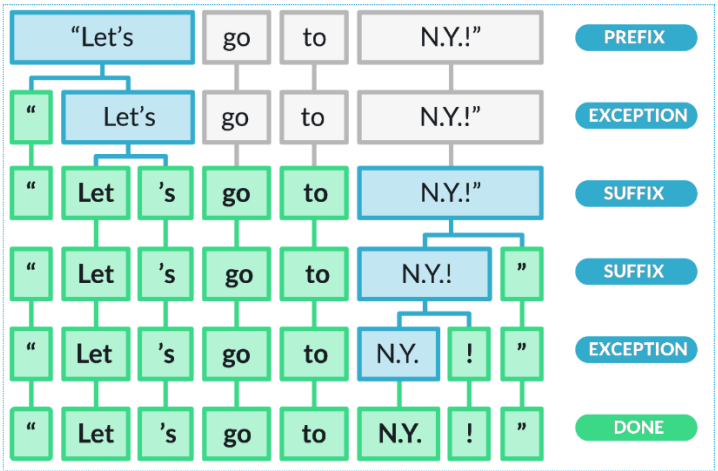


Figure 2.2.3(b)

While punctuation rules are usually pretty general, tokenizer exceptions strongly depend on the specifics of the individual language. Therefore each available language has its own subclass, like English or German, that loads in lists of hard-coded data and exception rules.

- Part-of-speech tags and dependencies:

After tokenization, spaCy can parse and tag a given Doc. This is where the trained pipeline and its statistical models come in, which enable spaCy to make predictions of which tag or label most likely applies in this context. A trained component includes binary data that is produced by showing a system enough examples for it to make predictions that generalize across the language – for example, a word following "the" in English is most likely a noun.

```
import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("Apple is looking at buying U.K. startup for $1 billion")

for token in doc:
    print(token.text, token.lemma_, token.pos_, token.tag_, token.dep_,
            token.shape_, token.is_alpha, token.is_stop)
```

RUN

Figure 2.2.3(c)

TEXT	LEMMA	POS	TAG
Apple	apple	PROPN	NNP
is	be	AUX	VBZ
looking	look	VERB	VBG
at	at	ADP	IN
buying	buy	VERB	VBG
U.K.	u.k.	PROPN	NNP
startup	startup	NOUN	NN
for	for	ADP	IN
$	$	SYM	$
1	1	NUM	CD
billion	billion	NUM	CD

- Named Entities:

This is the main feature for BATMAN Project in extracting the threats. Each threat is dealt as an Entity.
A named entity is a "real-world object" that's assigned a name – for example, a person, a country, a product or a book title. spaCy can recognize various types of named entities in a document, by asking the model for a prediction. Because models are statistical and strongly depend on the examples they were trained on, this doesn't always work *perfectly* and might need some tuning later, depending on required use case.
Named entities are available as its property of a Doc:

```
import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("Apple is looking at buying U.K. startup for $1 billion")

for ent in doc.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

RUN

Figure 2.2.3(d)

TEXT	START	END	LABEL	DESCRIPTION
Apple	0	5	ORG	Companies, agencies, institutions.
U.K.	27	31	GPE	Geopolitical entity, i.e. countries, cities, states.
$1 billion	44	54	MONEY	Monetary values, including unit.

here's what our example sentence and its named entities look like:

Applie **ORG** is looking at buying U.K. **GPE** startup for $1 billion **MONEY**

Pipeline:

When nlp on a text is called, spaCy first tokenizes the text to produce a Doc object. The Doc is then processed in several different steps – this is also referred to as the processing pipeline. The pipeline used by the trained pipelines typically include a tagger, a lemmatizer, a parser and an entity recognizer. Each pipeline component returns the processed Doc, which is then passed on to the next component.

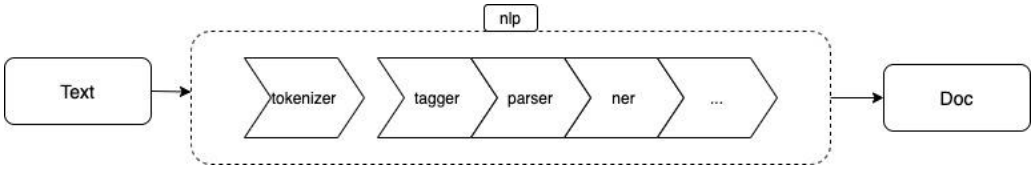


Figure 2.2.3 (e)

The capabilities of a processing pipeline always depend on the components, their models and how they were trained. For example, a pipeline for named entity recognition needs to include a trained named entity recognizer component with a statistical model and weights that enable it to make predictions of entity labels.

- Vocab, hashes and lexemes:

Whenever possible, spaCy tries to store data in a vocabulary, the Vocab, that will be shared by multiple documents. To save memory, spaCy also encodes all strings to hash values – in this case for example, "coffee" has the hash 3197928453018144401. Entity labels like "ORG" and part-of-speech tags like "VERB" are also encoded. Internally, spaCy only "speaks" in hash values.

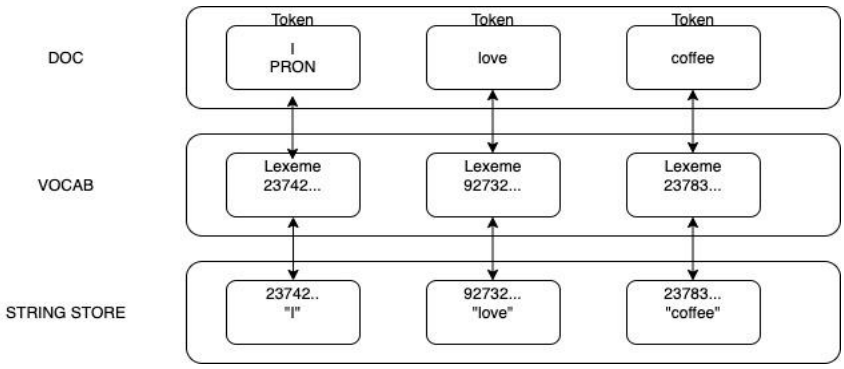


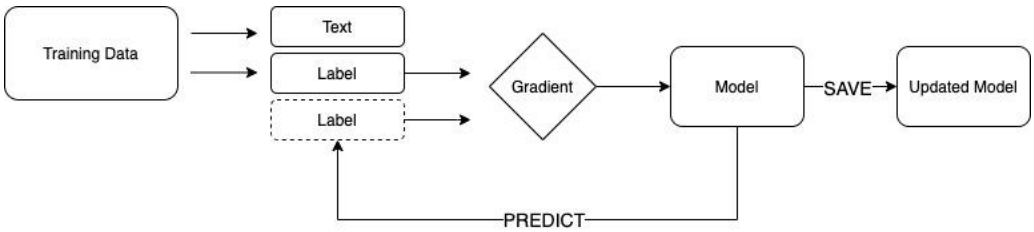
Figure 2.2.3(f)

Training Pipelines & Models:

spaCy's tagger, parser, text categorizer and many other components are powered by statistical models. Every "decision" these components make – for example, which part-of-speech tag to assign, or whether a word is a named entity – is a prediction based on the model's current weight values. The weight values are estimated based on examples the model has seen during training.

To train a model, first training data is needed – examples of text, and the labels we want the model to predict. This could be a part-of-speech tag, a named entity or any other information.

Training is an iterative process in which the model's predictions are compared against the reference annotations in order to estimate the gradient of the loss. The gradient of the loss is then used to calculate the gradient of the weights through backpropagation. The gradients indicate how the weight values should be changed so that the model's predictions become more similar to the reference labels over time.



When training a model, we don't just want it to memorize our examples – we want it to come up with a theory that can be generalized across unseen data. After all, we don't just want the model to learn that this one instance of "Amazon" right here is a company – we want it to learn that "Amazon", in contexts *like this*, is most likely a company. That's why the training data is always representative of the data we want to process. A model trained on Wikipedia, where sentences in the first person are extremely rare, will likely perform badly on Twitter. Similarly, a model trained on romantic novels will likely perform badly on legal text.

This also means that in order to know how the model is performing, and whether it's learning the right things, we don't only need training data – we will also need evaluation data. If we only test the model with the data it was trained on, we will have no idea how well it's generalizing. If we want to train a model from scratch, we usually need at least a few hundred examples for both training and evaluation.

- Language data:

Every language is different – and usually full of exceptions and special cases, especially amongst the most common words. Some of these exceptions are shared across languages, while others are entirely specific – usually so specific that they need to be hard-coded. The lang module contains all language-specific data, organized in simple Python files. This makes the data easy to update and extend.

The shared language data in the directory root includes rules that can be generalized across languages – for example, rules for basic punctuation, emoji, emoticons and single-letter abbreviations. The individual language data in a submodule contains rules that are only relevant to a particular language. It also takes care of putting together all components and creating the Language subclass – for example, English or German.

2.3 Comparison of Models

BERT	XLNET	SPACY
BERT is a bi-directional transformer for pre-training over a lot of unlabeled textual data to learn a language representation that can be used to fine-tune for specific machine learning tasks	XLNet is a large bidirectional transformer that uses improved training methodology, larger data and more computational power to achieve better than BERT prediction metrics on 20 language tasks.	spaCy is an open-source software library for advanced natural language processing, written in the programming languages Python
The performance improvement could be attributed to the bidirectional transformer, novel pre-training tasks of Masked Language Model	To improve the training, XLNet introduces permutation language modeling, where all tokens are predicted but in random order	At runtime spaCy will only use the [nlp] and [components] blocks of the config and load all data, including tokenization rules, model weights and other resources from the pipeline directory
It needs less data for training the model when compared to XLNET	It needs more data for training the model when compared to BERT	It needs a good amount of data for predicting the results accurately
Training time needed is very less than XLNET	Xlnet needs more than 6 times than BERT for training	Very less training time. In very less epochs the model trains and predict very well
F1 score: 0.55	F1 score: 0.35	F1score:0.100

Table 1: Comparison of NLP Models for NER task

3 SPACY IMPLEMENTATION FOR NER IN BATMAN AI SYSTEM

After comparing through all the model spaCy was selected for the NER operations. There are some important changes which were brought in from existing spaCy fundamental code.

3.1 Removed JSON dependencies

Previously spaCy was able to deal with Dataset in JSON format. It was a hazard to create json for each dataset. It was needed to provide exact location of the entities in the sentence to train the model.

Code Snippet:

```
def load_data(file_path):
    with open(file_path, encoding='utf-8') as train_data:
        train = json.load(train_data)

        TRAIN_DATA = []
        for data in train:
            ents = [tuple(entity) for entity in data['entities']]
            TRAIN_DATA.append((data['content'],{'entities':ents}))
    return TRAIN_DATA
```

```
TRAIN_DATA = load_data("data/Dataset.json")
LABELS = ['THREAT']
```

Figure 3.1(a)

In further development, It was changed. In present code, JSON Dataset Format is no longer required. Using Classified Data Conversion Application, a lightweight dataset is generated. This process of dataset creation makes it fast and effective.

```
def load_data_spacy(file_path):
    ''' Converts data from:
    label \t word \n label \t word \n \n label \t word
    to: sentence, {entities : [(start, end, label), (stard, end, label)]}
    '''
    file = open(file_path, 'r')
    training_data, entities, sentence, unique_labels = [], [], [], []
    current_annotation = None
    end = 0 # initialize counter to keep track of start and end characters
    for line in file:
        line = line.strip("\n").split("\t")
        # lines with len > 1 are words
        if len(line) > 1:
            label = line[0][2:]     # the .txt is formatted: label \t word, label[0:2] = label_type
            #print(line)
            #print(label)
            #print(line[0][0])
            label_type = line[0][0] # beginning of annotations - "B", intermediate - "I"
            word = line[1]
            word = word.rstrip()
            sentence.append(word)
            end += (len(word) + 1)  # length of the word + trailing space

            if label_type != 'I' and current_annotation:  # if at the end of an annotation
                entities.append((start, end - 2 - len(word), current_annotation))  # append the annotation
                current_annotation = None                 # reset the annotation
            if label_type == 'B':                         # if beginning new annotation
                start = end - len(word) - 1  # start annotation at beginning of word
                current_annotation = label   # append the word to the current annotation
            if label_type == 'I':            # if the annotation is multi-word
                current_annotation = label   # append the word
```

Figure 3.1(b)

```
            if label != 'O' and label not in unique_labels:
                unique_labels.append(label)

        # lines with len == 1 are breaks between sentences
        if len(line) == 1:
            if current_annotation:
                entities.append((start, end - 1, current_annotation))
            sentence = " ".join(sentence)
            training_data.append([sentence, {'entities' : entities}])
            # reset the counters and temporary lists
            end = 0
            entities, sentence = [], []
            current_annotation = None
    file.close()
    return training_data, unique_labels

TRAIN_DATA, LABELS = load_data_spacy("data/ExtractionDataset.txt")
```

Figure 3.1(c)

3.2 Enhanced Evaluation Process

In existing fundamental code, Evaluation process was not consistent. Each time Precision Recall and F1 Score provide same value. It was an hinderance to calculate accurate measures.

```
def calc_precision(pred, true):
    precision = len([x for x in pred if x in true]) / (len(pred) + 1e-20)
    # true positives / total pred
    return precision

def calc_recall(pred, true):
    recall = len([x for x in true if x in pred]) / (len(true) + 1e-20)
    # true positives / total test
    return recall

def calc_f1(precision, recall):
    f1 = 2 * ((precision * recall) / (precision + recall + 1e-20))
    return f1
from itertools import chain

# run the predictions on each sentence in the test dataset, and return the spacy object
preds = [ner(x[0]) for x in TEST_DATA]

precisions, recalls, f1s = [], [], []

# iterate over predictions and test data and calculate precision, recall, and F1-score
for pred, true in zip(preds, TEST_DATA):
    true = [x[2] for x in list(chain.from_iterable(true[1].values()))]
    # x[2] = annotation, true[1] = (start, end, annot)
    pred = [i.label_ for i in pred.ents] # i.label_ = annotation label, pred.ents = list of annotations
    precision = calc_precision(true, pred)
    precisions.append(precision)
    recall = calc_recall(true, pred)
    recalls.append(recall)
    f1s.append(calc_f1(precision, recall))

print("Precision: {} \nRecall: {} \nF1-score: {}".format(np.around(np.mean(precisions), 3),
                                                         np.around(np.mean(recalls), 3),
                                                         np.around(np.mean(f1s), 3)))
```

```
Precision: 0.588
Recall: 0.588
F1-score: 0.588
```

In further development, Internal library of spaCy was used to calculate its performance measures.

```
from spacy.gold import GoldParse
from spacy.scorer import Scorer

def evaluate(nlp, examples):
    scorer = Scorer()
    for input_, annot in examples:
        #print(str(input_) +" *** "+str(annot))
        text_entities = []
        for entity in annot.get('entities'):
          text_entities.append(entity)
        #print("input_ "+input_)
        #print("text_entities "+str(text_entities))
        doc_gold_text = nlp.make_doc(input_)
        gold = GoldParse(doc_gold_text, entities=text_entities)
        pred_value = nlp(input_)
        scorer.score(pred_value, gold)
    return scorer.scores
```

Figure 3.2(a)

After the new changes these methods were engaged in performance test for each iteration. Following is a snippet for it:

```
Iteration 1 Loss: {'ner': 2041.527153223753}
F Score --> 0.0
Iteration 2 Loss: {'ner': 418.096625018381}
F Score --> 0.0
Iteration 3 Loss: {'ner': 396.8868114460929}
F Score --> 0.0
Iteration 4 Loss: {'ner': 340.31692082314476}
F Score --> 0.0
Iteration 5 Loss: {'ner': 338.14556906719963}
F Score --> 17.391304347826093
Iteration 6 Loss: {'ner': 281.5258992406766}
F Score --> 22.22222222222222
Iteration 7 Loss: {'ner': 240.54164316039612}
F Score --> 29.885057471264364
Iteration 8 Loss: {'ner': 244.9228265869935}
F Score --> 56.41025641025641
Iteration 9 Loss: {'ner': 205.33004131021627}
F Score --> 57.142857142857153
Iteration 10 Loss: {'ner': 193.95396121731926}
F Score --> 65.88235294117646
Iteration 11 Loss: {'ner': 160.33142477199553}
F Score --> 74.07407407407408
Iteration 12 Loss: {'ner': 134.0296838697341}
F Score --> 75.60975609756099
Iteration 13 Loss: {'ner': 109.80150624856381}
F Score --> 80.48780487804878
Iteration 14 Loss: {'ner': 125.14823588663208}
F Score --> 82.05128205128206
Iteration 15 Loss: {'ner': 333.1800763532606}
F Score --> 83.54430379746836
Iteration 16 Loss: {'ner': 118.06040114367921}
F Score --> 81.01265822784811
Iteration 17 Loss: {'ner': 69.78345342998189}
F Score --> 82.92682926829268
Iteration 18 Loss: {'ner': 50.52023840294133}
F Score --> 98.76543209876543
Iteration 19 Loss: {'ner': 72.69601497491573}
F Score --> 100.0
Iteration 20 Loss: {'ner': 44.3214156607178}
F Score --> 100.0
Completed in 16 seconds
```

Figure 3.2(b)

After training the spaCy Model. It's loss and F1 score variants are plotted in a visual format. Following is the image for its visualization.

```
import matplotlib.pyplot as plt
plt.plot(range(IterationCount),lossesArray,linestyle='-',c='g',label='loss')
plt.plot(range(IterationCount),flScoreArray,linestyle='-',c='r',label='flscore')
plt.legend()
plt.show()
```

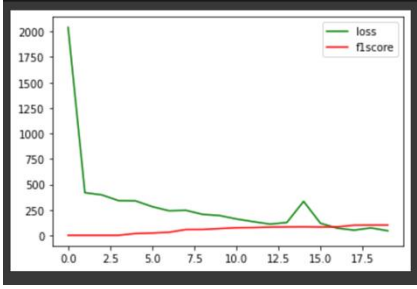


Figure 3.2(c)

After training spaCy. It provided results as following:

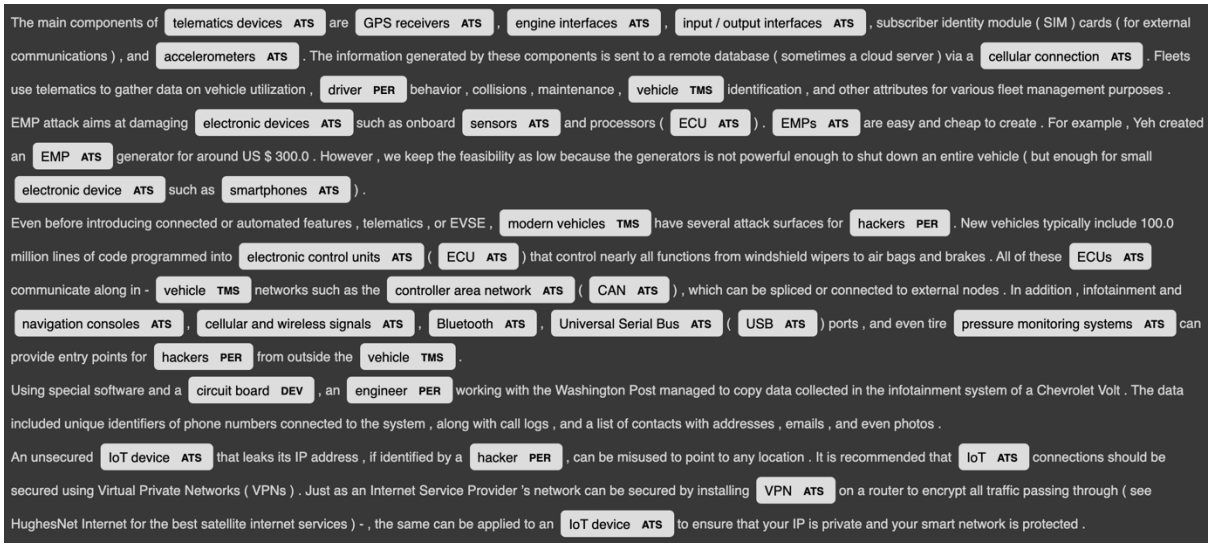


Figure 3.2(d)

GIT Link: [10]

4 CLASSIFICATION USING BERT

4.1 About BERT

BERT (Bidirectional Encode Representation from Transformers) is a new pre-training language representation model which is introduced by Google AI Language researchers. By the word pre-trained representation, it means that we can train the model on a large dataset and then we can use the trained model for any NLP task for which the model is trained [9].

Many NLP tasks require results based on deep understanding of sentences to predict the result. For such tasks, the bi-directionally trained BERT is used which means that the model will look from both sides of the sentence to have a deeper understanding of context in comparison to unidirectional models. The BERT model was trained on an unlabelled plain text dataset that is already available on various public websites in different languages. Such unsupervised, and bidirectional methods helped BERT model outperform other pre-training language representation models [9].

How BERT works?

BERT can be used for various NLP tasks like for predicting answers to questions, sentimental analysis and so on. BERT's bidirectional training relies on transformers (that includes the concept of attention mechanism that helps to learn a contextual relationship between words or sub-words in a sentence [9]. This means that using the attention mechanism, transformers understand the relationship between all words in text input. For example, if a sentence is given as "I reached the bank to withdraw some money", the transformer will quickly pay attention to the bank and understand that the 'bank' refers to financial organization and is not related to the shore of a river [10]).

Such types of characteristics made BERT more effective in learning the contextual relationship. Two important stages of a BERT model are [8]:

1. Pre-training
2. Fine Tuning.

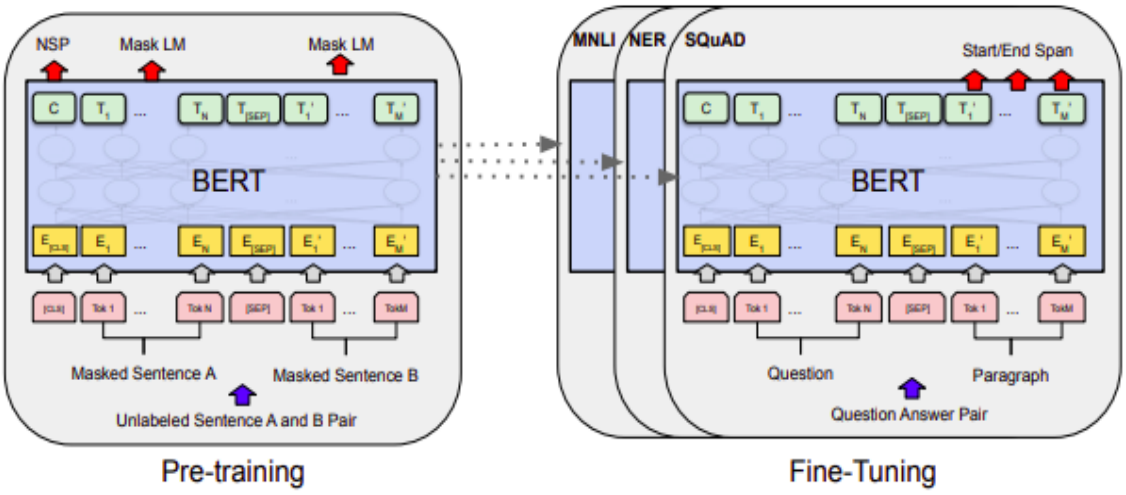


Figure 4.1: Diagram showing the Pre-training and Fine-tuning stages in BERT model [8].

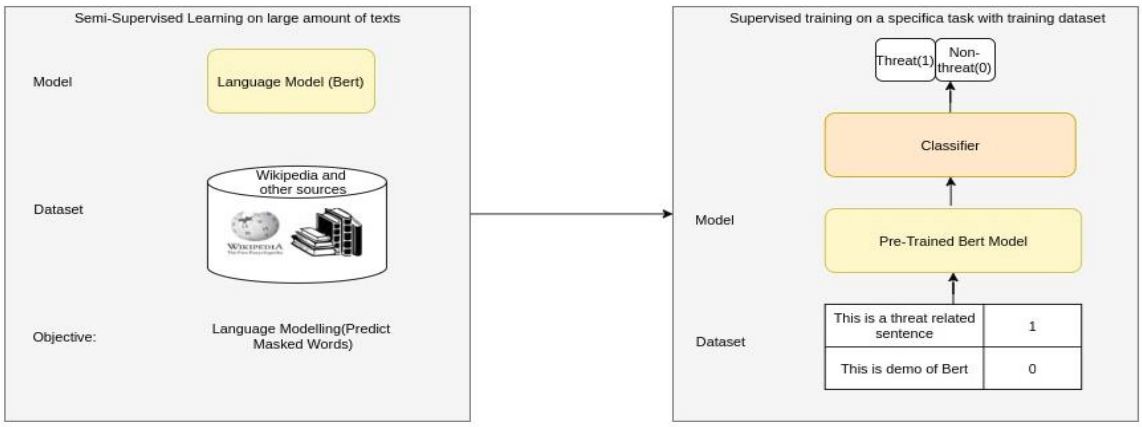


Figure 4.1 (a): Procedure how BERT is trained on large corpus of different language and then the pre-trained model is used for fine-tuning on custom dataset for classification.

Pre-training stage:

Pre-training is the first stage for BERT training. It is basically performed in an unsupervised manner and trained on enormous amounts of unannotated text that is publicly available on the web. Pre-training stage is quite expensive but is a onetime procedure for different languages as shown in figure 4.1 (a). Mostly English language models are used. As the BERT model is already pre trained by Google, a lot of time is saved by most nlp researchers because they don't have to pre train their own model from scratch.

Regarding the pre-training process in BERT, the BERT model consists of two main tasks in pre-training [9]:

1. Masked Language Model (MLM).
2. Next Sequence Prediction (NSP).

Fine-tuning stage: Now after the pre-training stage, after either pre-training BERT model by ourselves or by loading already pretrained BERT model for example BERT-base-case or BERT-base-uncased models, the BERT model can then be fine-tuned or trained on a separately created dataset for a specific task like for sentence classification, question answering and so on to get state-of-the-art predictions. As BERT can be used for various tasks by adding a task-specific layer on top loaded pre-trained model, our main task for this project is text classification, so for such a task a simple SoftMax classifier is added on top of the pre-trained model [11].

Now our task was to develop a News classification model by using the BERT. One subgoal of our AI project is to create a model that takes news content and classifies whether the news is threat related news or non-threat related news. After exploring various techniques, we found that BERT can be used for such text classification tasks.

The implementation of the BERT model for the News classification is described in the following sections and pipeline is shown in figure 4.2 and 4.2.1.

4.2 Implementation of BERT

Now after learning some basic concepts regarding the BERT model, to achieve our subgoal of news classification (to classify whether news content is threat or non-threat), we used the BERT model that is implemented using the pytorch transformers library from the hugging-face repository. We need to perform following steps for classification which are as follows [9,12,14]:

1. First setting up the environment by using the pytorch-transformers library version provided by hugging face.
2. Pre-processing of dataset for BERT model
    a. Convert the news_classification dataset that is in .csv format into .tsv format(table separate value) that is acceptable by BERT
    b. Use the. tsv format that contains text representation and convert it into features(numerical representation) so that pretrained BERT models can be trained on the dataset.
3. Load the Pre-trained BERT model for fine tuning.
4. Fine-Tuning of the pre-trained model for news classification task

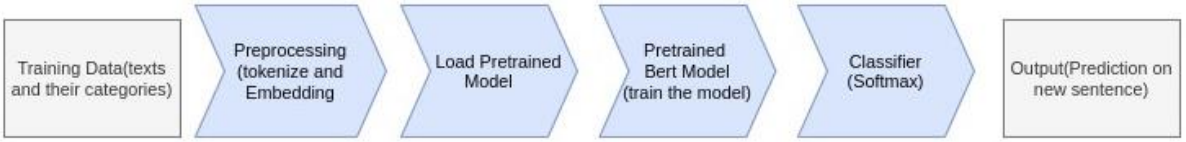


Figure 4.2: Basic pipeline for training and fine-tuning of BERT for classification.

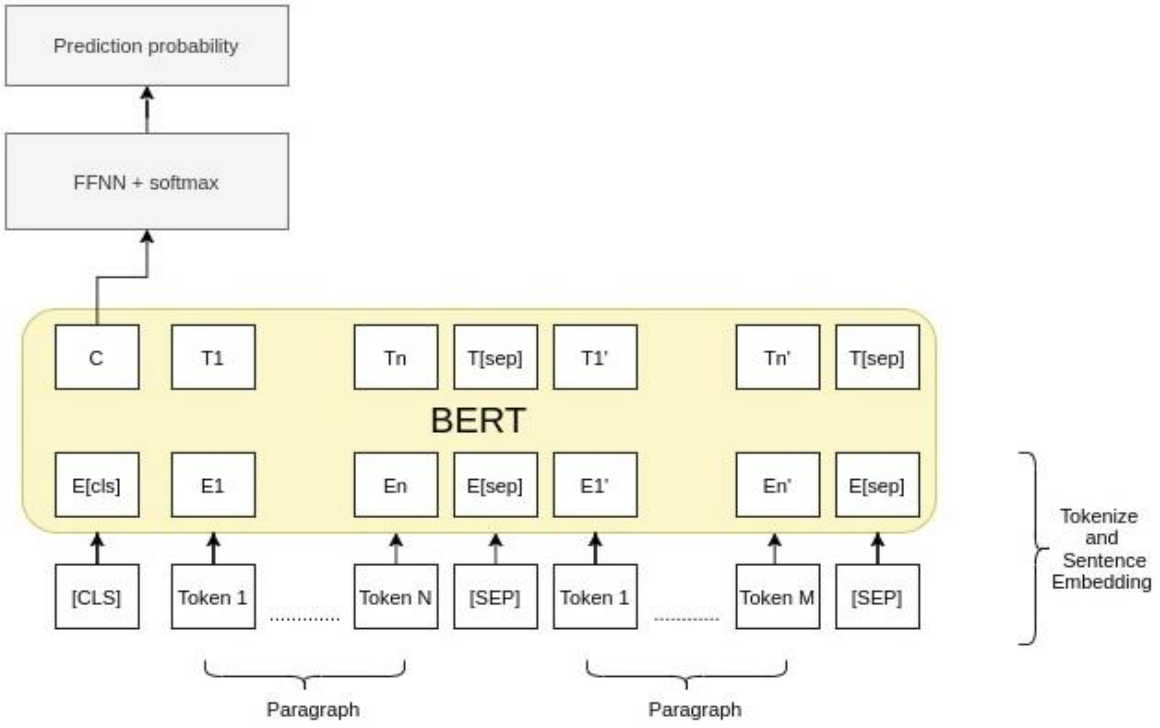


Figure 4.2.1: Inference pipeline for the BERT Model for text classification

Step by step process to understand the classification implementation process which is as follows:

1.) Setting up environment

In this first step we will first install all necessary libraries required for the BERT model to work. We will install pytorch-pretrained-BERT that is a pytorch version of BERT from hugging face. We used Google colab for the implementation of the BERT model for news classification. Google colab allows developers to perform any python code through a browser and is extremely useful for machine learning tasks.

```
[ ] import pandas as pd

[ ] pip install pytorch-pretrained-bert
```

Figure 4.2.1 (a): Setup of libraries for model to work.

2.) Pre-Processing of dataset for BERT model

To train our BERT model for news classification (binary text classification task) we need to first pre-process our custom dataset that is built for the news classification. For this we will first load our news_classification_edited.csv file which is a training dataset and test_dataset_for_classification.csv is a test dataset which will be used for the evaluation of performance of the BERT model.

1. Create a directory in Google Colab using command '!mkdir data'. This used to load training and test dataset. Read the training and test datasets that have names 'news_classification_edited.csv' and 'test_dataset_for_classification'.

In our dataset, there are two columns, one is the labels (1 for threat and 0 for non-threat) and another column contains the news [9,14]. To convert the .csv to .tsv, we needed to structure the dataset by adding four columns: row id, second column for labels, third column with same letter for all text as it is needed by BERT, fourth column contains news. The output after restructuring looks like this:

```
train_df_bert = pd.DataFrame({
    'id':range(len(train_df)),
    'label':train_df[0],
    'alpha':['a']*train_df.shape[0],
    'text': train_df[1].replace(r'\n', ' ', regex=True)
})
train_df_bert.head()
```

	id	label	alpha	text
0	0	1	a	A hacker who swipes a relatively inexpensive P...
1	1	0	a	A self-driving car has learned to make high-sp...
2	2	1	a	Last winter, a hacker who goes by the handle J...
3	3	1	a	In addition, hackers have also been able to lo...
4	4	1	a	A new report from IntSights details the ways t...

```
[ ] dev_df_bert = pd.DataFrame({
    'id':range(len(test_df)),
    'label':test_df[0],
    'alpha':['a']*test_df.shape[0],
    'text': test_df[1].replace(r'\n', ' ', regex=True)
})
dev_df_bert.head()
```

	id	label	alpha	text
0	0	1	a	The industry has been working on improving the...
1	1	1	a	Common ways of stealing cars include carjackin...
2	2	1	a	Bypass kits can be an attack vector to the ant...
3	3	1	a	Hackers can intercept telematics traffic using...
4	4	1	a	Since that first public vehicle hacking demons...

Figure 4.2.1(b): Restructured dataset for train and test dataset for conversion to .tsv.

a. Now after changing dataset format, we converted it into .tsv format as follows:

```
[ ] train_df_bert.to_csv('data/train.tsv', sep='\t', index=False, header=False)

[ ] dev_df_bert.to_csv('data/dev.tsv', sep='\t', index=False, header=False)
```

Figure 4.2.1 (c): Getting .tsv file for further data pre-processing.

In this the train is for training models and dev is for evaluation of model performance.

b. Next step is Conversion of dataset in.tsv to data features (numerical format)

Data Processor and Binary Classification Processor classes are used for reading the content in .tsv format and prepare the .tsv files to be converted into features that is in numerical format as BERT cannot directly work with the text in the dataset because of neural network. As we learnt earlier, a BERT encoder takes input as vectors (numerical value) and give output using a single layer of SoftMax as output layer. To achieve this, data needs to be changed, following are the steps for it.

- First with the help of Binary Classification Processor class, the .tsv files content are converted into lists of Input Example objects. So, to create objects for each feature mentioned for all text in our dataset, the Input Examples class is used. Now, the Input Examples objects still contain the text but in a format that can be easily converted to Input Features (numerical data)[14].

- Now to convert the Input Examples into Input Features further subtasks will be performed which are as follows:

  - **Tokenize, padding and sequential length**:
    - First by using Binary Classification Processor we will load train examples which are Input Examples objects.
    - Then we will load the pretrained tokenizer by BERT. We used a BERT-base-case model for casing the alphabets. Now after normalizing text, tokenization is performed and based on sequence length padding is also performed.
    - In the phase of tokenization, some words will be broken into smaller tokens (or Word pieces) for the transformer to easily recognize all tokens. For example, word calling will be broken into call and ##ing [12, 14].

  - **CLS and SEP:** Another step is to add CLS and SEP tokens to differentiate between starting and ending of sentences as shown in figure 4.2.1(d) and 4.2.1(e).

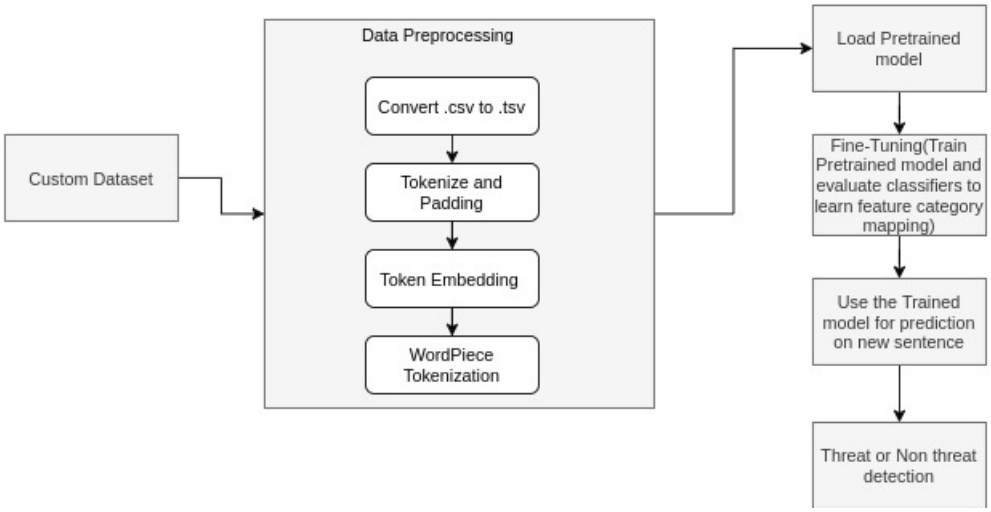


Figure 4.2.1 (d): Flow of Data pre-processing that can be used for training BERT model.

Pre-Processing Conclusion:

Thus, first a list of examples will be prepared and fed into the convert_example_to_feature function for the conversion of examples into features by tokenizing, truncating/padding, and adding a few tokens. As the result of Input Features is returned which matches the same format as accepted by the BERT model for training. Now as we pre-processed our dataset, the last step is fine-tuning the pretrained BERT model.
For this first we load our pre-trained model and set some parameters for fine-tuning.

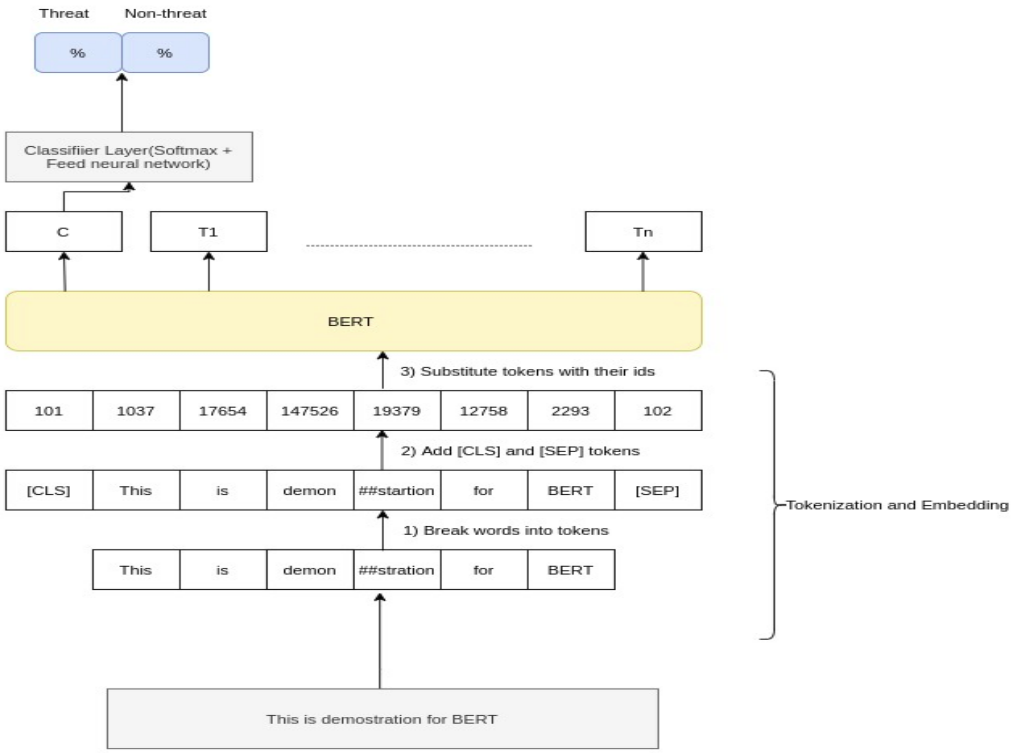


Figure 4.2.1 (e): Complete architecture for data pre-processing, fine-tuning of BERT and testing for classification on trained classifier model.

3.) Fine-Tuning

To train such models we must train the classifier with minimal changes to the BERT model during the training phase. This is called fine tuning [13]. Before the fine-tuning starts, we will load a BERT-base-cased pretrained model. And after installing all important packages required for training, we will mention the directories where our trained model, pre-trained model will be saved and their related output files [14]. After mentioning the directories and loading our model and some weights and setting up Data Loader for training, we will train our model on the pre-processed dataset [14].

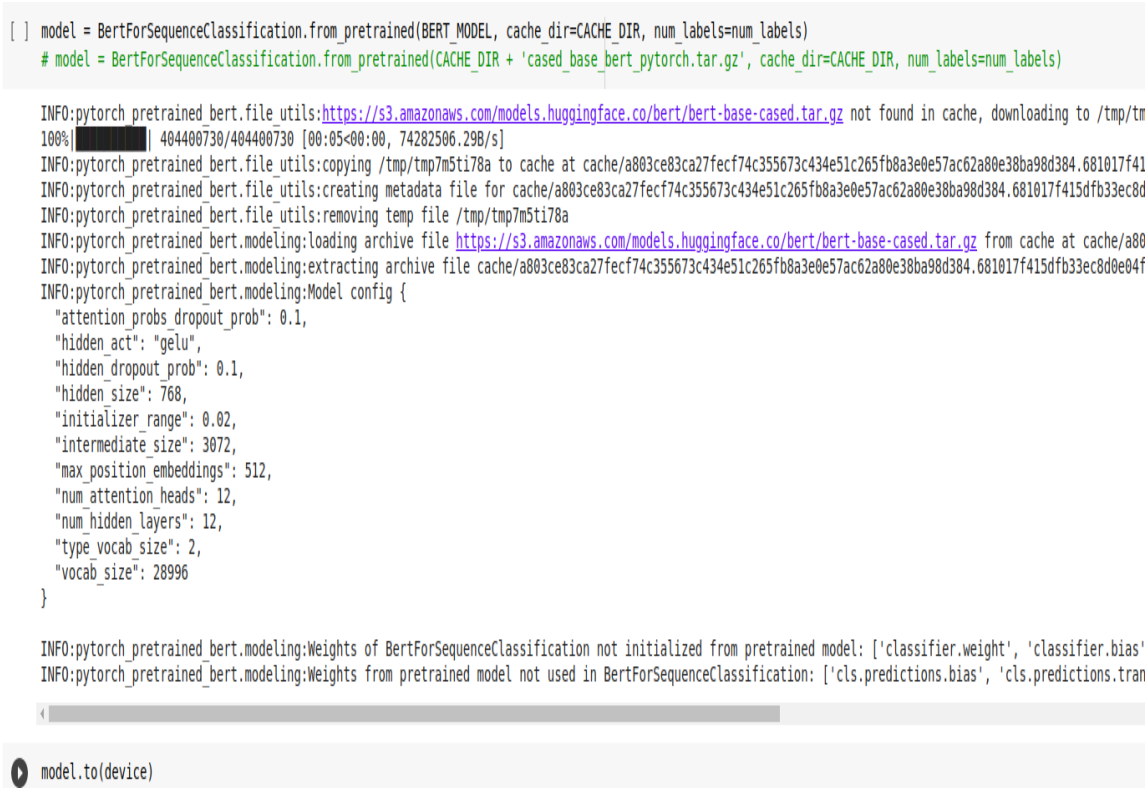


Figure 4.2.1 (f): The loading of Pre-trained BERT model and addition of classifier layer on top of BERT model.

model.train() is used for training and below this command is a loop for training the model for 10 epochs. The training took 36 minutes for 10 epochs. We can train our model for more epochs also, but 10 epochs are sufficient. As we have trained our model, we can evaluate the results. But before testing we will have our trained model in a cache directory created for saving BERT trained models. Complete code implementation flow is shown in figure 4.2.1 (i).

```
model.train()
for _ in trange(int(NUM_TRAIN_EPOCHS), desc="Epoch"):
    tr_loss = 0
    nb_tr_examples, nb_tr_steps = 0, 0
    for step, batch in enumerate(tqdm_notebook(train_dataloader, desc="Iteration")):
        batch = tuple(t.to(device) for t in batch)
        input_ids, input_mask, segment_ids, label_ids = batch

        logits = model(input_ids, segment_ids, input_mask, labels=None)

        if OUTPUT_MODE == "classification":
            loss_fct = CrossEntropyLoss()
            loss = loss_fct(logits.view(-1, num_labels), label_ids.view(-1))
        elif OUTPUT_MODE == "regression":
            loss_fct = MSELoss()
            loss = loss_fct(logits.view(-1), label_ids.view(-1))

        if GRADIENT_ACCUMULATION_STEPS > 1:
            loss = loss / GRADIENT_ACCUMULATION_STEPS

        loss.backward()
        print("\r%f" % loss, end='')

        tr_loss += loss.item()
        nb_tr_examples += input_ids.size(0)
        nb_tr_steps += 1
        if (step + 1) % GRADIENT_ACCUMULATION_STEPS == 0:
            optimizer.step()
            optimizer.zero_grad()
            global_step += 1
```

Figure 4.2.1 (g): The BERT model with classifier layer on top was trained for 10 epochs on a custom dataset.

```
0.003554WARNING:pytorch_pretrained_bert.optimization:Training beyond specified 't_total'. Learning rate multiplier set to 0.0. Please set 't_total' of WarmupLinearSchedule correctly.
0.002883WARNING:pytorch_pretrained_bert.optimization:Training beyond specified 't_total'. Learning rate multiplier set to 0.0. Please set 't_total' of WarmupLinearSchedule correctly.
0.003248WARNING:pytorch_pretrained_bert.optimization:Training beyond specified 't_total'. Learning rate multiplier set to 0.0. Please set 't_total' of WarmupLinearSchedule correctly.
0.002979WARNING:pytorch_pretrained_bert.optimization:Training beyond specified 't_total'. Learning rate multiplier set to 0.0. Please set 't_total' of WarmupLinearSchedule correctly.
0.003308WARNING:pytorch_pretrained_bert.optimization:Training beyond specified 't_total'. Learning rate multiplier set to 0.0. Please set 't_total' of WarmupLinearSchedule correctly.
0.003072WARNING:pytorch_pretrained_bert.optimization:Training beyond specified 't_total'. Learning rate multiplier set to 0.0. Please set 't_total' of WarmupLinearSchedule correctly.
0.003257WARNING:pytorch_pretrained_bert.optimization:Training beyond specified 't_total'. Learning rate multiplier set to 0.0. Please set 't_total' of WarmupLinearSchedule correctly.
0.003274WARNING:pytorch_pretrained_bert.optimization:Training beyond specified 't_total'. Learning rate multiplier set to 0.0. Please set 't_total' of WarmupLinearSchedule correctly.
0.005714WARNING:pytorch_pretrained_bert.optimization:Training beyond specified 't_total'. Learning rate multiplier set to 0.0. Please set 't_total' of WarmupLinearSchedule correctly.
Epoch: 100%|██████████| 10/10 [46:18<00:00, 277.85s/it]
```

Figure 4.2.1 (h): Output of training time of model in 10 epochs

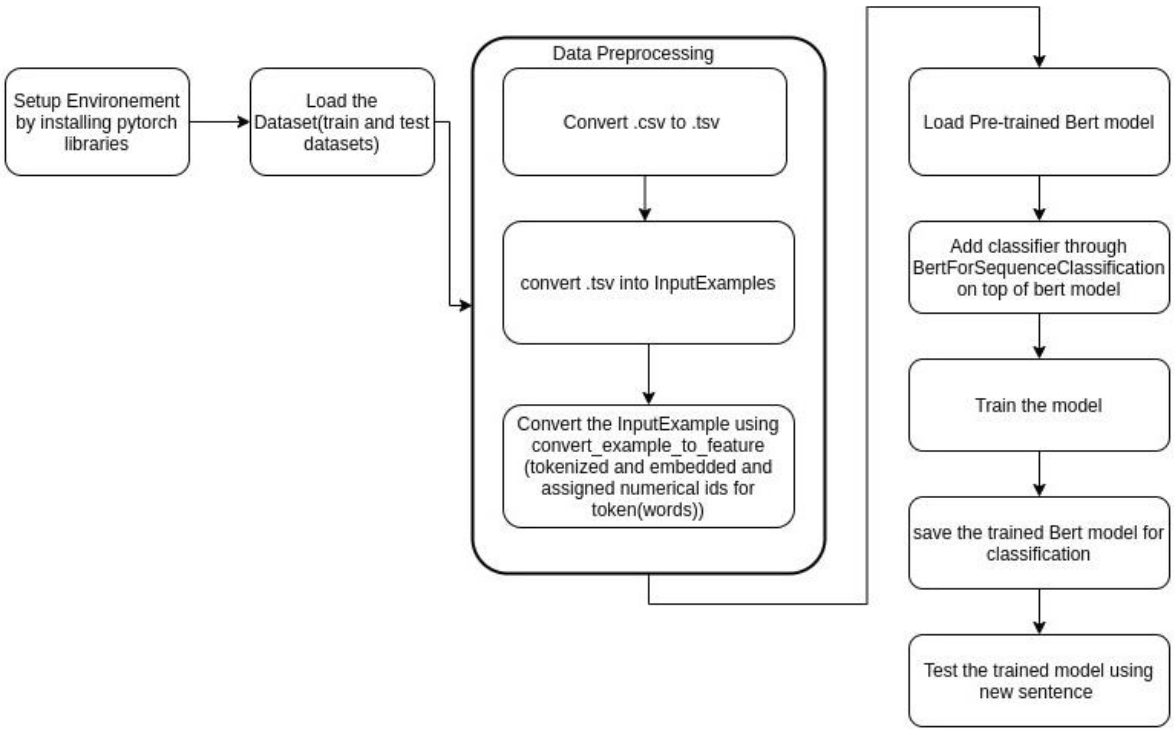


Figure 4.2.1 (i): Complete Flow of the implementation of the BERT for News Classification

To download the trained model first save the trained model in cache directory as follows:

```
[ ] #To save our trained model

[ ] model_to_save = model.module if hasattr(model, 'module') else model  # Only save the model it-self

    # If we save using the predefined names, we can load using `from_pretrained`
    output_model_file = os.path.join(OUTPUT_DIR, WEIGHTS_NAME)
    output_config_file = os.path.join(OUTPUT_DIR, CONFIG_NAME)

    torch.save(model_to_save.state_dict(), output_model_file)
    model_to_save.config.to_json_file(output_config_file)
    tokenizer.save_vocabulary(OUTPUT_DIR)

    'outputs/news-classification/vocab.txt'
```

Figure 4.2.1 (j): Saving of trained model.

Then download the model and all necessary output files as follows:

1. First check the folders what is stored and where.

```
[ ] #to check the folder in which model is saved and specifying path for further testing of our trained model

[ ] cd outputs/news-classification/

    /content/outputs/news-classification

[ ] ls

    config.json  pytorch_model.bin  vocab.txt

[ ] !tar -zcvf news-classification.tar.gz ./config.json ./pytorch_model.bin

    ./config.json
    ./pytorch_model.bin

[ ] cp ./news-classification.tar.gz ../../cache/
```

Figure 4.2.1 (k): For checking folder in which model is saved and specifying path for further testing of our trained model.

2. Download the folders required for working in Backend for Threat Extractor Application.

```
[ ] #to download the trained model in our system

[ ] from google.colab import files
    files.download('news-classification.tar.gz')


[ ] !zip -r /content/outputs.zip /content/outputs

      adding: content/outputs/ (stored 0%)
      adding: content/outputs/news-classification/ (stored 0%)
      adding: content/outputs/news-classification/test_predictions_data.csv (deflated 57%)
      adding: content/outputs/news-classification/vocab.txt (deflated 49%)
      adding: content/outputs/news-classification/config.json (deflated 47%)
      adding: content/outputs/news-classification/pytorch_model.bin (deflated 7%)
      adding: content/outputs/news-classification/test_predictions.csv (deflated 57%)
      adding: content/outputs/news-classification/news-classification.tar.gz (deflated 0%)

[ ] from google.colab import files
    files.download("/content/outputs.zip")


    !zip -r /content/reports.zip /content/reports

    adding: content/reports/ (stored 0%)
    adding: content/reports/news-classification_evaluation_reports/ (stored 0%)
    adding: content/reports/news-classification_evaluation_reports/report_0/ (stored 0%)
    adding: content/reports/news-classification_evaluation_reports/report_1/ (stored 0%)
    adding: content/reports/news-classification_evaluation_reports/report_1/eval_results.txt (deflated 14%)

[ ] from google.colab import files
    files.download("/content/reports.zip")
```

Figure 4.2.1 (l): Commands to download the folders for trained BERT model from Google Colab for the Backend part of Threat Extractor.

4.) Testing of Trained model

For evaluation, we will follow the same procedure as mentioned in the training phase in the same google Colab code after training of pre trained BERT model. The steps are as:

1. The trained BERT model name is based on task specific name. Here the trained model file name is news-classification.tar.gz.
2. For the data tokenization, the tokenizer will be loaded from the vocab file created during the training phase of the BERT model that is from outputs/news-classification/vocab.txt file.
3. Another important part for the pre-processing part is that the Binary Classification Processor will now load the dev.tsv file through the get_dev_examples function and forward it for conversion in input features as done in the pre-processing phase in the training part.
4. And at last load the trained BERT model only.

After all the above steps the model was evaluated on a dev dataset and the results are as:

The trained BERT model achieved: **0.84 Matthews correlation coefficient** (which is a good measure for the evaluation of unbalanced datasets) on a single-layer neural network as the classifier.

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`

Evaluating: 100% ████████████████████ 2/2 [32:15<00:00, 967.98s/it]
INFO:root:***** Eval results *****
INFO:root:  task = news-classification
INFO:root:  mcc = 0.848528137423857
INFO:root:  tp = 9
INFO:root:  tn = 4
INFO:root:  fp = 1
INFO:root:  fn = 0
INFO:root:  eval_loss = 0.44653313304297626
```

Figure 4.2.1 (m): The performance evaluation output for our trained BERT model on a test dataset.

Future work:

We can also improve the model performance with more training and with some hyperparameters tuning also, however the 0.84 score is good enough for our model evaluation for single sentences.

We also tested our model on single news input without any label. As we learned above, the BERT model takes input and converts the text into vectors to feed in the BERT attention encoder. As a single classification layer is added at top of the encoder output, the input after conversion will be sent to a model that further processes the result in a neural network and then forwarded to SoftMax for the calculation of probability of threat and non-threat as shown in figure 4.2.1(m).

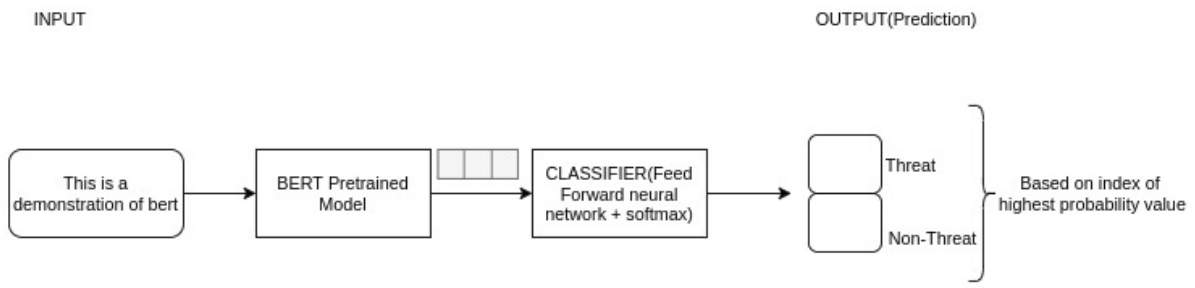


Figure 4.2.1 (n): Flow of testing a single sentence by using trained model for classification.

```
from scipy.special import softmax

pred_probs = softmax(preds, axis=1)
pred_probs
```

```
array([[0.00121751, 0.9987824 ]], dtype=float32)
```

Figure 4.2.1 (o): Output probability of SoftMax layer for single sentence to determine threat or non-threat news.

We leveraged our BERT model for threat type classification also that followed the same procedure.

For News Classification Files

Python File Name: News_Classification_using_BERT_from_Hugging_face.ipynb.

Folder path for code and Dataset in final branch: AI-UC_2021 -> NLP_Fundamentals ->BERT/NEWS_CLASSIFICATION_USING_BERT- >News_Classification_using_BERT_from_Hugging_face.

Final Branch Git Link: [8]

Tool Used for Implementation of BERT: Google Colab

For Threat Type Classification Files

Python File Name: Threat_classification_Using_BERT.ipynb.

Folder contains code and Dataset in final file: AI-UC_2021 -> AI BATMAN THREAT EXTRACTOR -> Backend ->BERT->Threat_Type_Classification in final file.

Final Branch Git Link: [9]

Tool Used for Implementation of BERT: Google Colab

How To Execute the BERT Code (for news classification and threat type classification)

The steps are the same for both News classification using BERT and for threat type classification using BERT. Both the models are trained separately on different collabs with different datasets and the codes can be executed which are as follows

1. Upload the News_Classification_BERT_Using_Pytorch_HuggingFace.ipynb on Google Colab. Similarly for threat type identification model, upload Threat_classification_Using_BERT.ipynb.

2. Also upload the datasets in the data directory. Create a data directory using !mkdir data.

3. Just mention the datasets in read.csv command. After this, execute the commands one by one by checking for errors.

5 ARCHITECTURE AND DESIGN

5.1 AI/UC System Architecture:

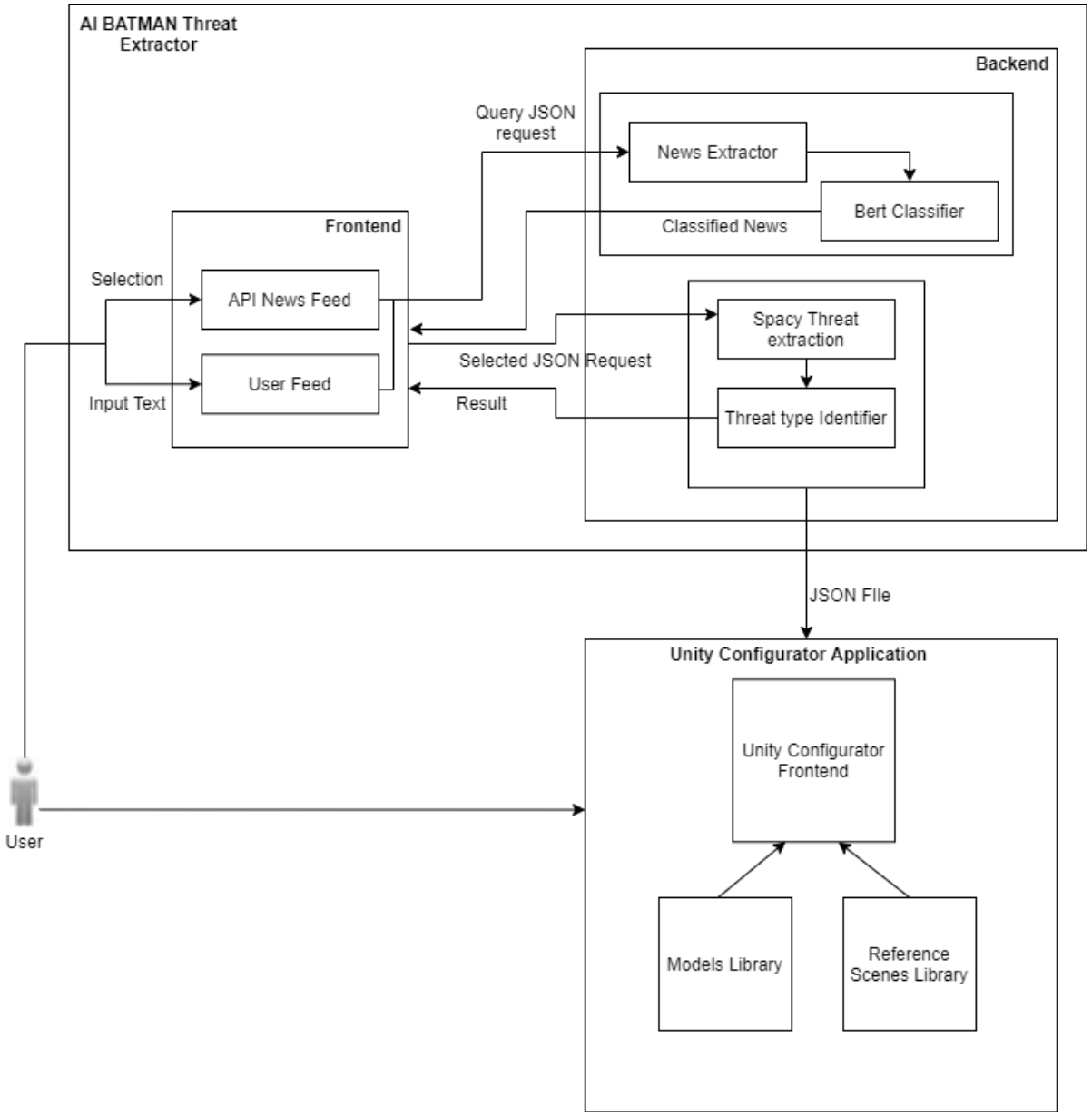


Figure 5.1: Overall AI/UC System Architecture

The overall architecture of AI/UC consists of two major components namely, AI BATMAN Threat Extractor and Unity Configurator application. The news or texts related to threats are fed to AI backend where the threat related information is extracted and forwarded to AI frontend and Unity Configurator application.

5.2 AI System Architecture:

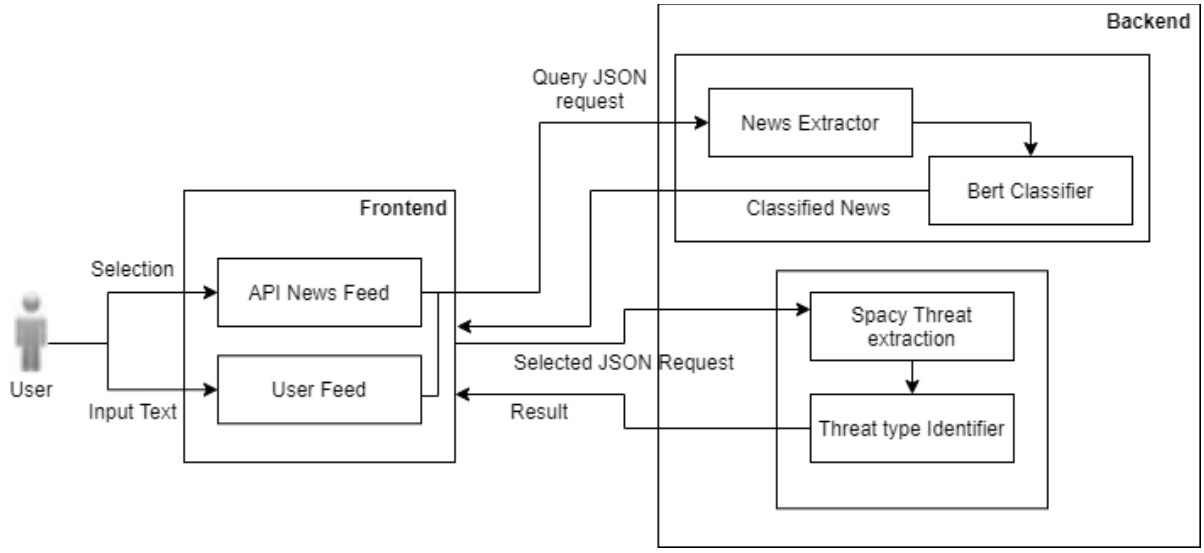


Figure 5.2: Architecture of BATMAN Threat Extractor

The user is provided with two options to give the input ie, the user can select the news from the news API feeds which is a real time news fetched by the API or the user can give input in the form of text in the user feed. The given input goes to the backend in the form of JSON, where the data is stored in a set of key-value pairs. The required data is extracted and then forwarded to the BERT classifier where the extracted data is classified into threat and non-threat type. The data which belongs to a threat type is forwarded to the frontend which we call as Classified news. The user selects a particular news from the displayed news list which will be sent to Spacy Threat Extraction in the form if JSON and to the Threat type Identifier. Later, the result/solution is given back to the user in the form of text.

5.3 Use-Case Diagram

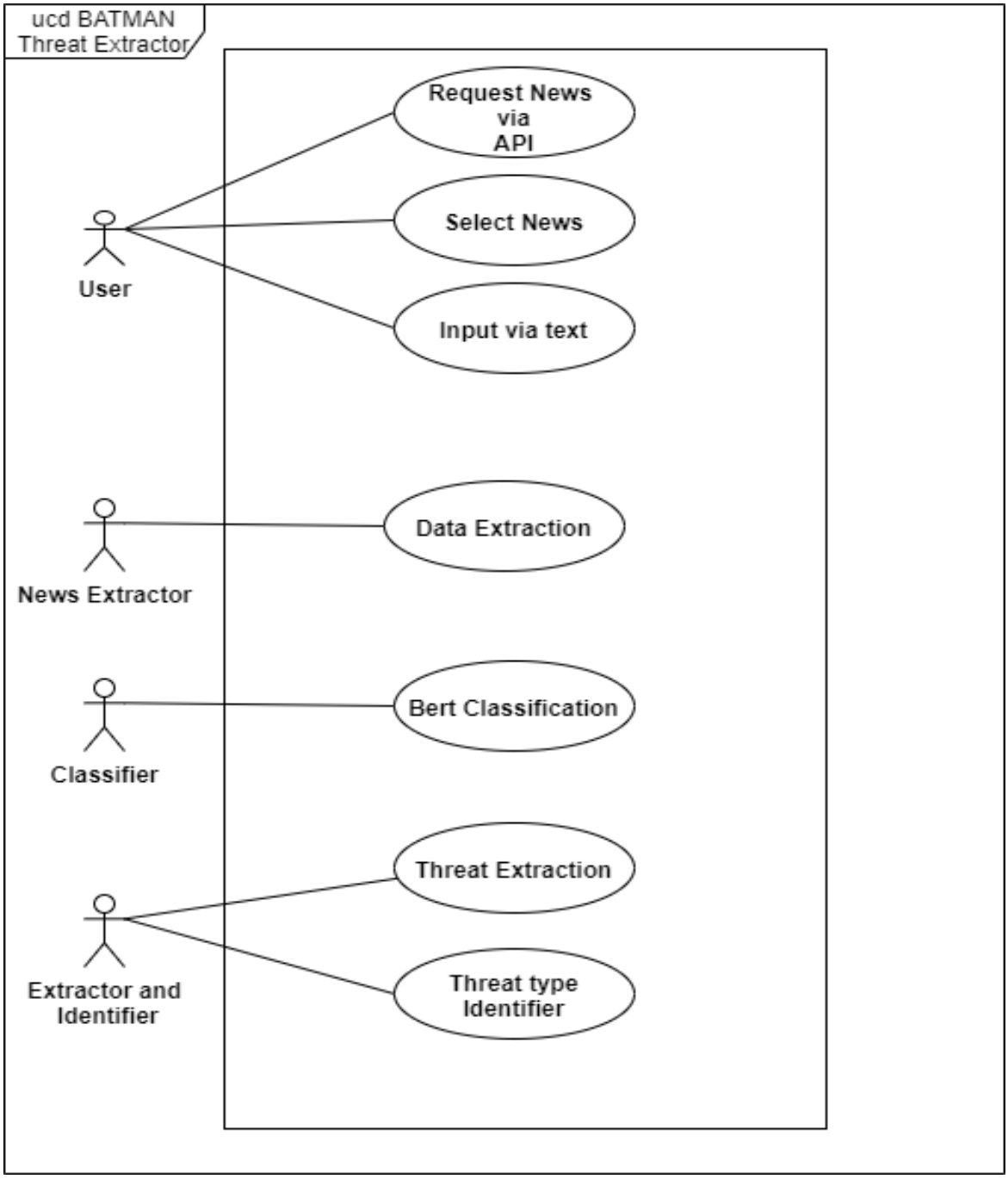


Figure 5.3: Use-case Diagram: BATMAN Threat Extractor

5.4 Block Definition Diagram

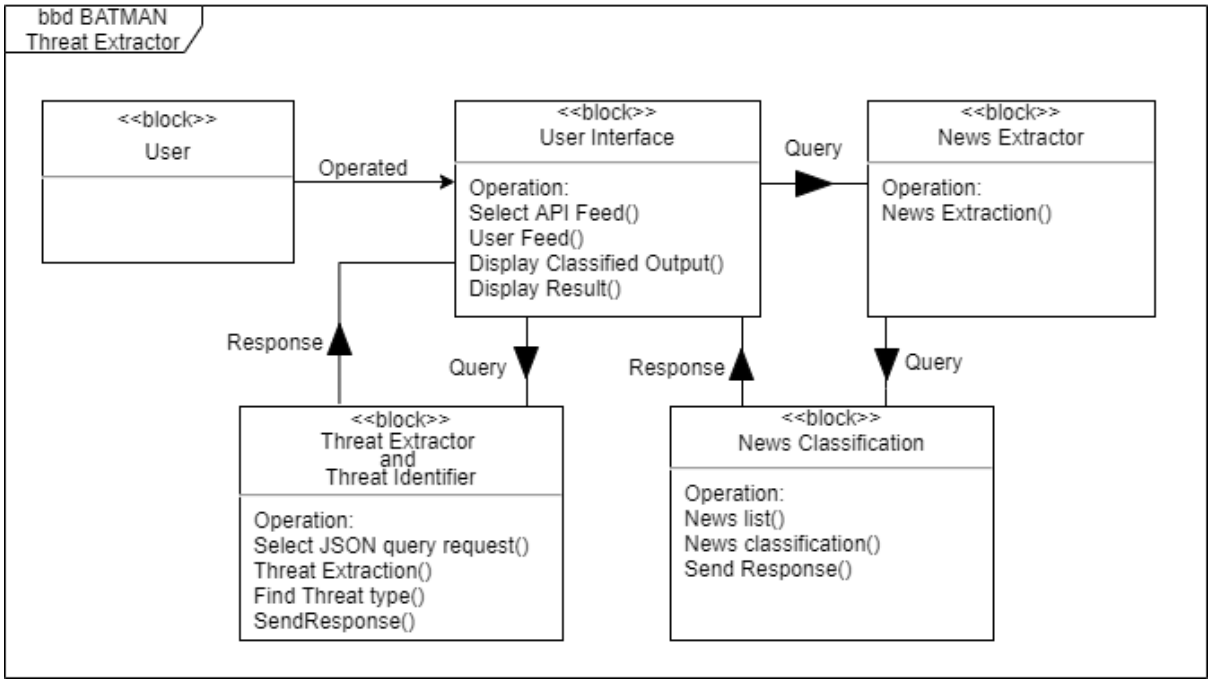


Figure 5.4: Block Definition Diagram of BATMAN Threat Extractor

5.5 Sequence Diagram

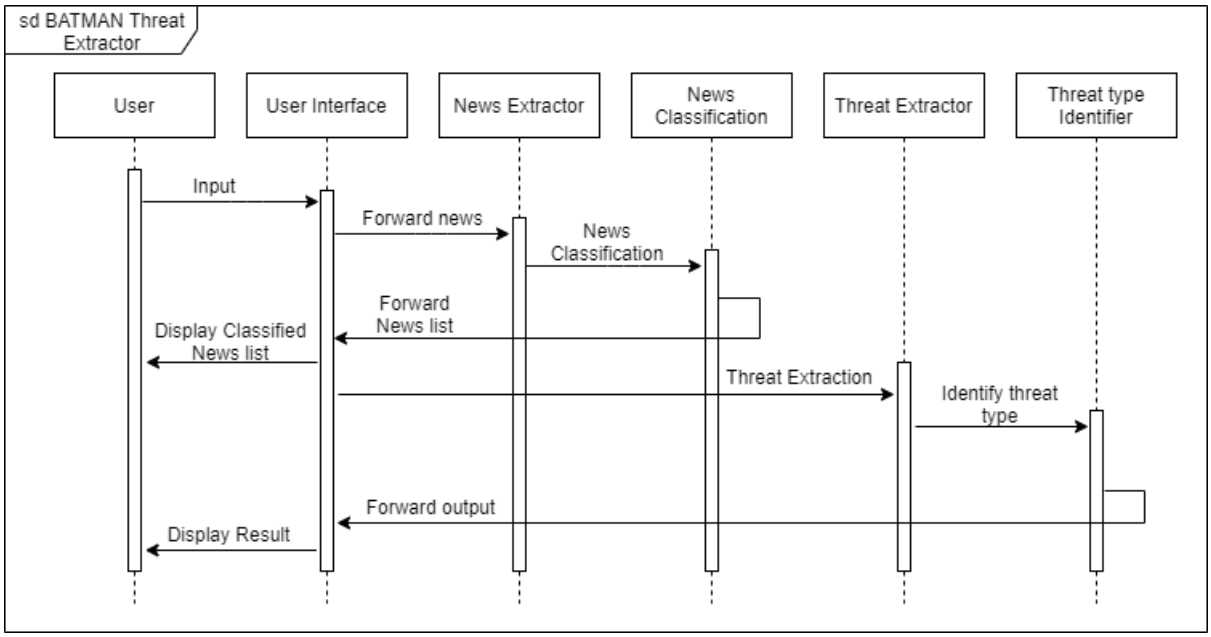


Figure 5.5: Sequence Diagram of BATMAN Threat Extractor

5.6 Activity Diagram

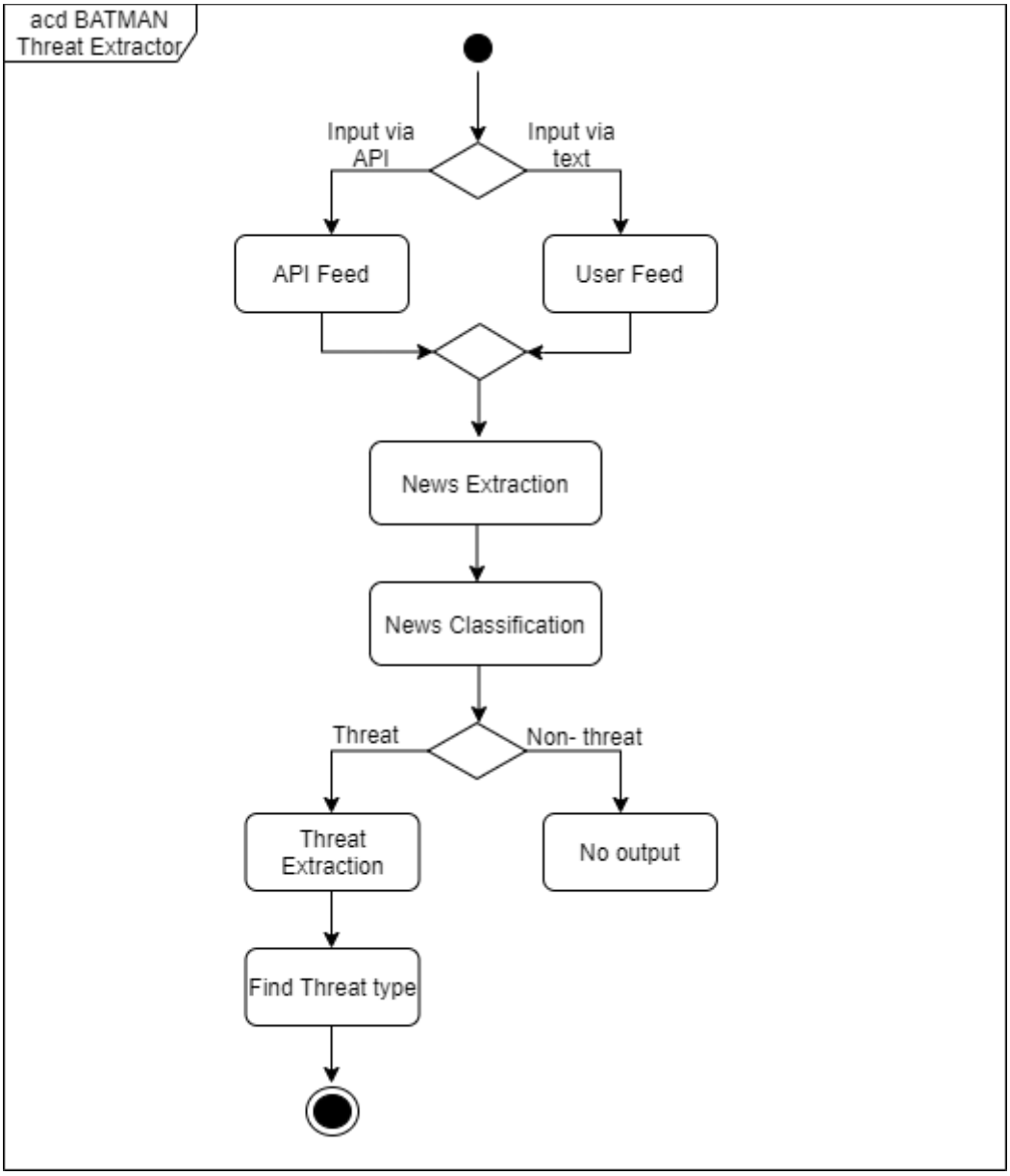


Figure 5.6: Activity Diagram of BATMAN Threat Extractor

5.7 Unity Configurator Architecture:

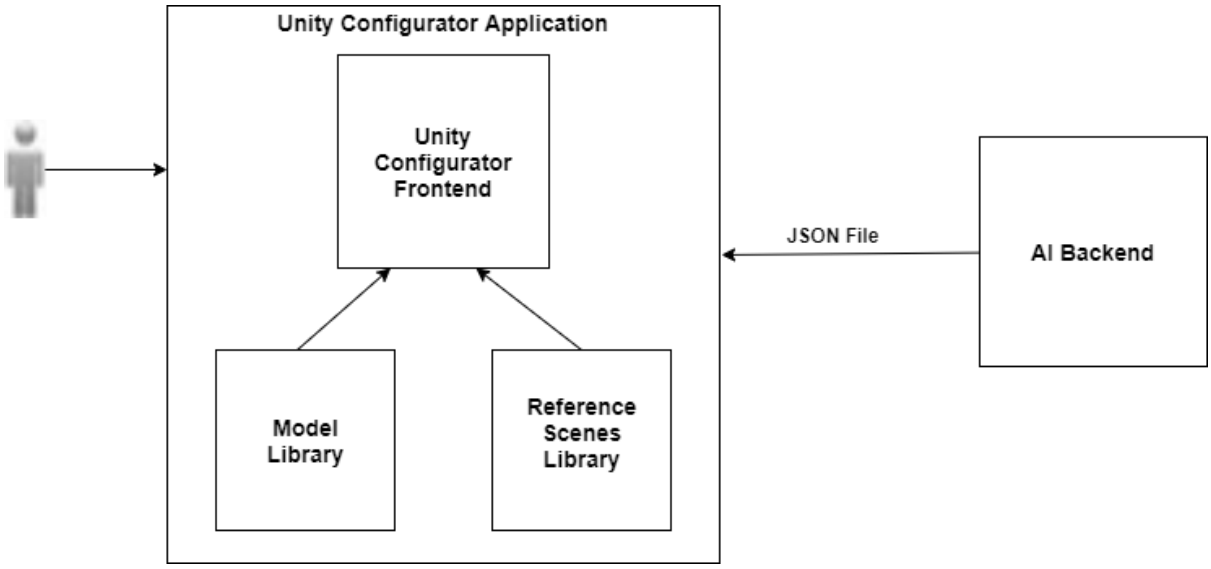


Figure 5.7: Architecture of Unity Configurator Application

Unity Configurator Application consists of the following components.

AI Backend: AI backend has the threat and vulnerabilities extracted data. Which is delivered to the Unity Configurator Application in the form of JSON file.

JSON File: JSON file contains all the information regarding Threat type, Actors, Technical systems, and other details which is parameterized into Unity Configurator Application.

Model Library: All the relevant and necessary models are gathered and stored in a library and Unity Configurator Application load into the respected references scenes.

Reference Scenes Library: A list of eight scenes is gathered and used in this application, based on the threat type one of the reference scene is selected.

Unity Configurator Frontend: This is the visual part where the reference scenes and models are loaded, and newly configured reference scenario available to the user, where user can interact with the application.

User: User can interact with the application, move around the scenes, and can perform some operation to find out the all the possible threats and new threats.

In this way, the threat and vulnerability related scenario data from AI is passed to Unity Configurator in a JSON format and the data is parameterized and used for selecting appropriate reference scenes from reference scene library and configures a new reference scenario by loading the models from the model library into this selected reference scene. Where user can explore the generated reference scenario and can perform some actions such as exploring the generated scenario and find out new threats.

5.8 Unity Configurator Sequence Flow:

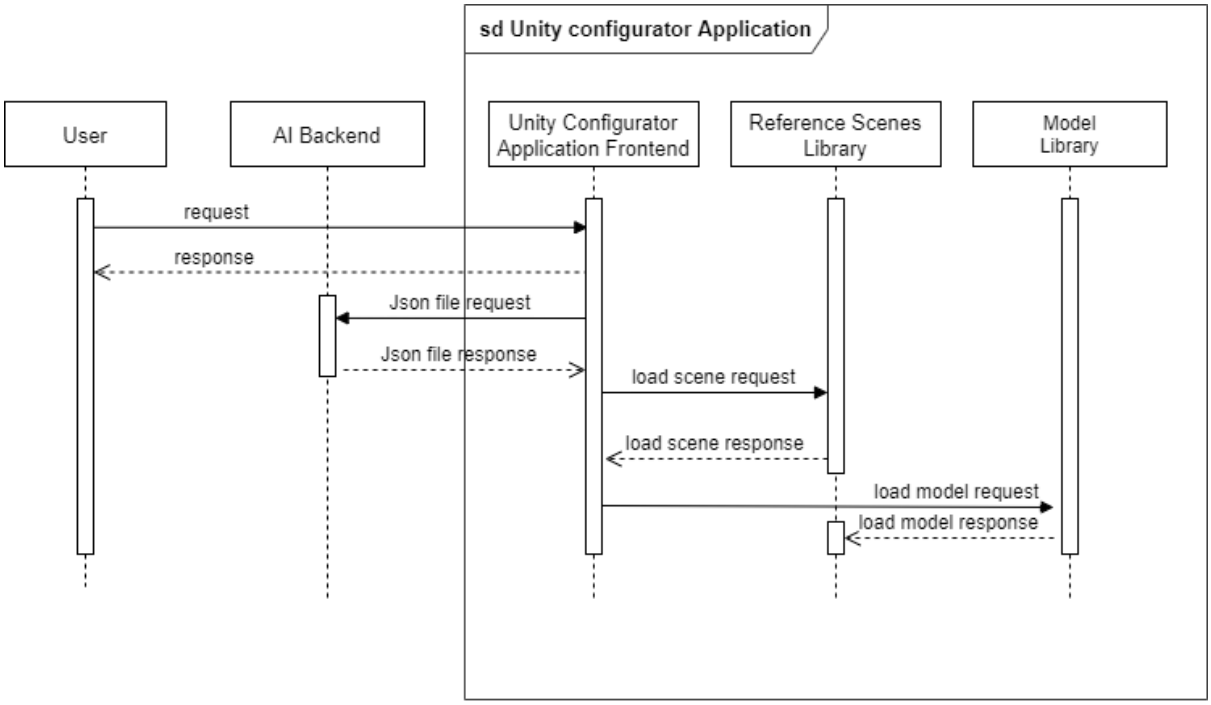


Figure 5.8: Sequence Diagram of Unity Configurator Application

Figure 5.8 shows a sequence diagram, depicting not only the interactions between the user and the Unity Configurator Application but also between the internal components of this application. The major functionalities of the Unity Configurator Application are as follows:

User Request and Response: When the user requests a reference scene, models, a function to be loaded then the response is visible at the front end of the application.

Json File Request and Response: When the request for a json file is made, the json file which is received from AI backend is loaded into the application through frontend of the application.

Load Scene Request and Response: If the user selects a scene through a frontend then in response, the reference scene is loaded.

Load Model Request and Response: If the user selects an option to load the models into the respected scene through front end options, then the models are loaded in response.

6 EXPLAINATION OF SUBPRODUCTS AND FEATURES

6.1 AI Frontend

Front-End Overview

6.1.1 Foundation:

User experience (UX) forms the foundations on which the values of the application are delivered in the digital world and acts as a bedrock before starting with the development of the application. This means that the user interface, ease of use, and speed of a website must all be optimized. User experience (UX) focuses on fully understanding how users interact with an interface. After several research on the technologies to be used to develop our Batman threat extractor web application, we decided to use the combination of ReactJS (JavaScript library) and Bootstrap 4.0 as a CSS designing framework. React is a declarative, efficient, lightweight, and flexible JavaScript library for building Single Page Application (SPA). It is an open-source, component-based, front-end library for the application's view layer. In Model View Controller (MVC) architecture, the view layer is responsible for how the app looks and feels. The advantages such as reusable components, unidirectional data flow, availability of dedicated tools for easy debugging makes it the first and best choice for UI development. Bootstrap is the best HTML, CSS, and JavaScript framework for any web developer. It will quickly produce responsive, clean-looking website applications.

6.1.2 UI pages and functionalities

Our web application is divided into several web pages. The view of our landing page is sectioned into the header, footer, and an anchor tag at the center of the page to navigate to the homepage as in figure 6.1.2 (a)

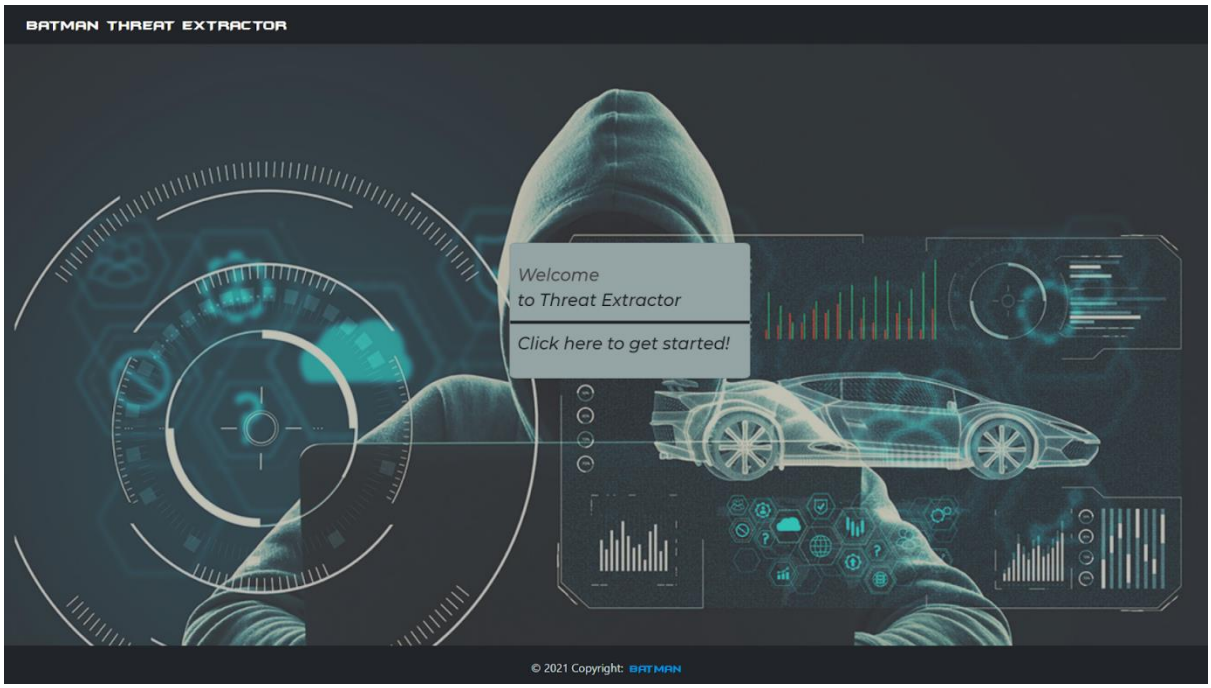


Figure 6.1.2 (a): Landing page

The header and footer are used as the reusable components throughout the application. Once the user clicks on the button, the user will be navigated to the homepage. The homepage is partitioned into two parts. On the left part of the page, the user will be displayed with the live news feed fetched from the open sources such as *NewsAPI*, and *NYTimesAPI* and on the right side, the user is provided with a flexible text field to input the *user feed*. On load, the user will be displayed with the live data fetched from the NewsAPI which is an open-source, simple, easy-to-use REST API which returns the JSON news articles published by over 75,000 worldwide sources. The user will be given an option to switch between NewsAPI and NYTimes API to fetch the news through a dropdown list. The user can also limit the length of list items to be fetched from the API by selecting the *page length*. When the user changes the option to NYTimes API which is another free source to fetch live data as an array of articles, news, and, top stories on the specified sections (automobiles, business, sports, and etc.).

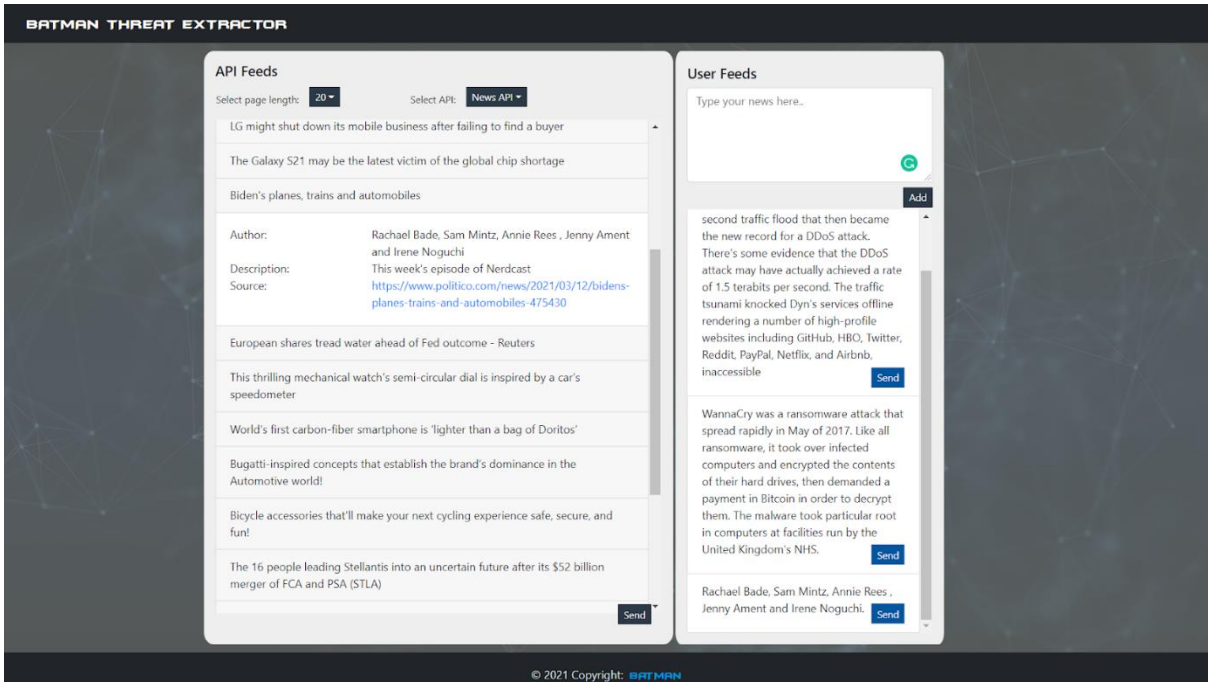


Figure 6.1.2 (b): Home page

In our application, we specifically fetch the data related to automotive, and automobiles. The user will be enabled to add multiple user feeds and the added items will be shown in the list. Figure 6.1.2 (b) is the screenshot of the homepage of our application. The list of items on the left partition will be displayed using collapsible accordion elements.

Thereafter the user will be given an option to send either the fetched items or the user feeds for the threat classification. Once the user clicks on the send button which is present on the left part of the homepage, the user will be redirected to the classified output page wherein a list of classified items is displayed in the accordion format containing the title, content, source of the item, and a NER (Named Entity Recognition) button as shown in Figure 6.1.2 (c).

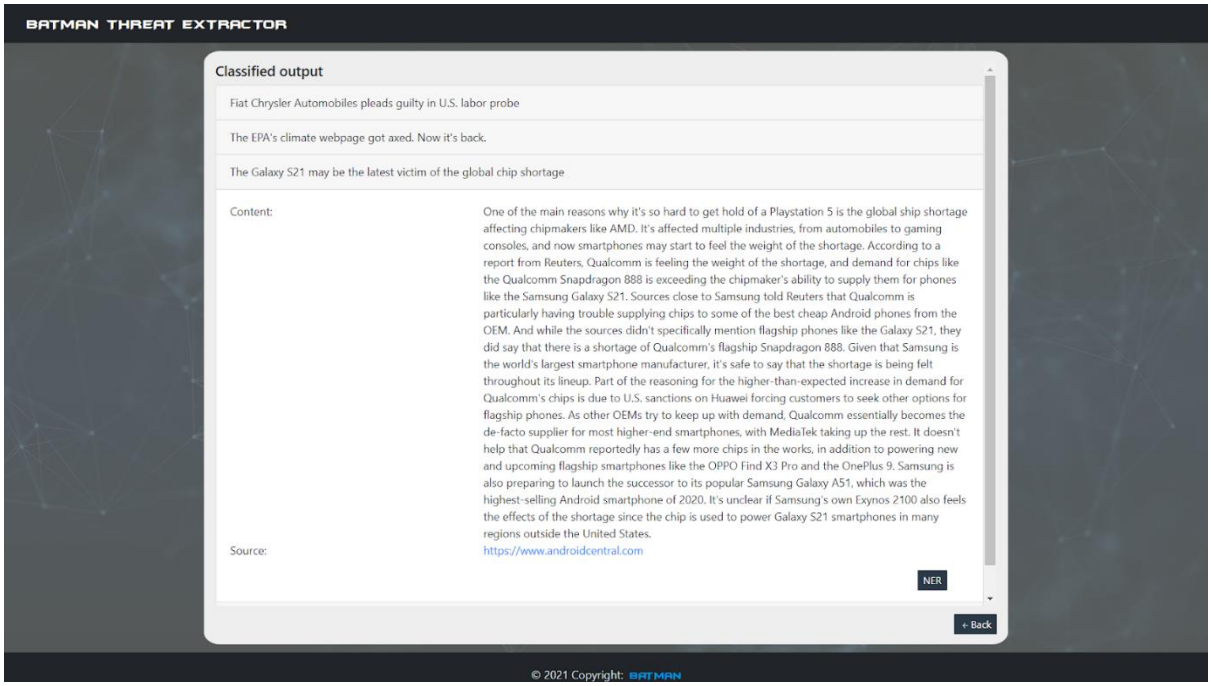


Figure 6.1.2 (c): Classified Output page

Users will also be able to send each of the items loaded by the user as user feed wherein there are two possibilities, when the added item is considered as a threat by the model the user will be navigated to the classified output page displaying the selected content and the *NER* button. If in case it is considered as a non-threat the classified output page will be displayed with an information message *"Not a threat! You can click here to go back to add a new user feed or click on the back button"* with an option to return to the homepage as in Figure 6.1.2 (d).

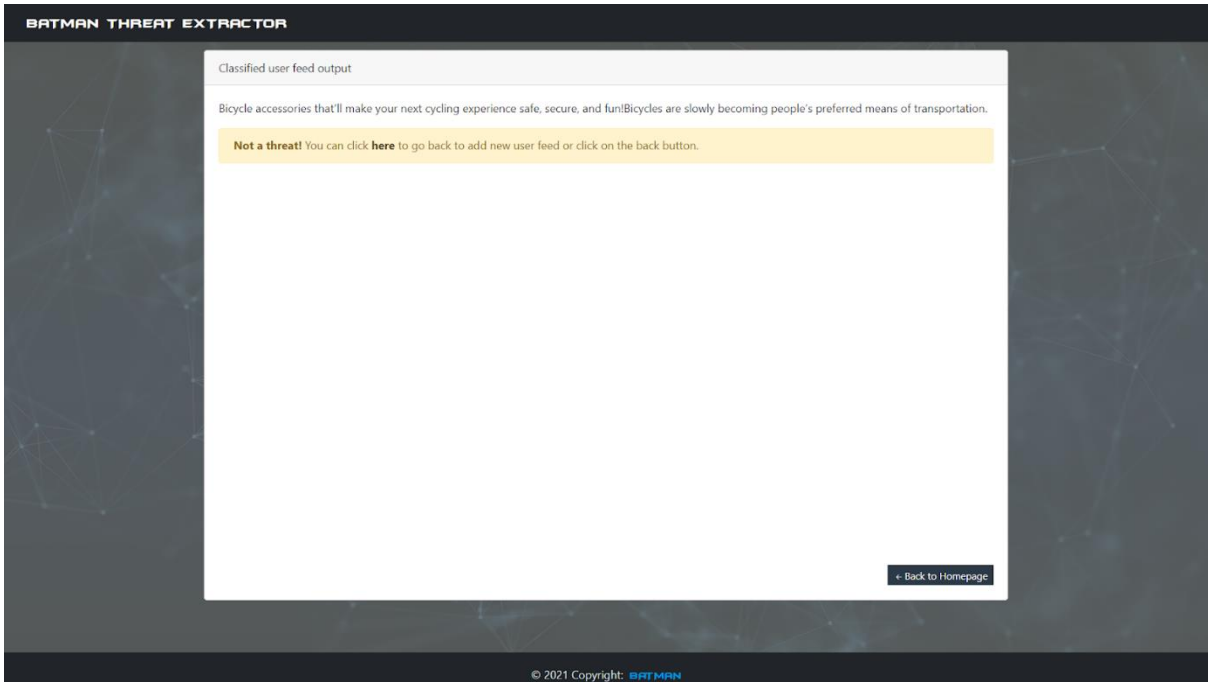


Figure 6.1.2 (d): Classified User Feed Output Page

Once the user clicks on the NER button in the classified output page the user will be redirected to the threat-based extracted output page where the threat specific words (tags) of the content will be highlighted with a distinct set of colours along with the titles for each of the tags, and the threat type will be displayed to the user. There is also a back button option provided to return to the homepage as in figure 6.1.2 (e).

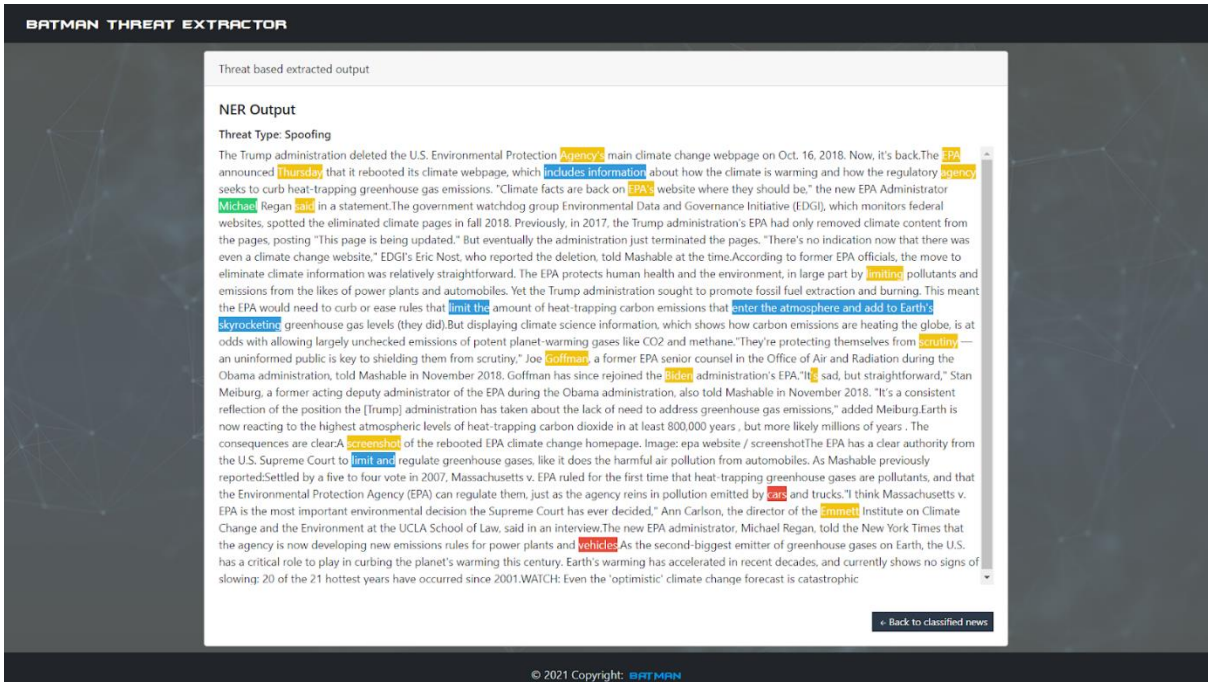


Figure 6.1.2 (e): Threat Based Extracted Output Page

Named-entity recognition (NER) (also known as (named) entity identification, entity chunking, and entity extraction) is a subtask of information extraction that seeks to locate and classify named entities mentioned in unstructured text into predefined categories. In our application altogether our model classifies the named entities from the threat classified content into 8 predefined categories such as *Person* (medium sea green), *Another technical system* (moon yellow), *Technical mobility system* (cinnabar - brick red), *Attack* (orange), *Animal* (light sea green), *Consequences* (summer sky blue), *Target* (deep lilac violet), and *Hazards* (madison blue) are highlighted with above-mentioned colours for each of the tags. And, we have 11 sets of predefined threat types of namely Brute force (SBF), Denial of service (DOS), Spoofing (SPF), Natural Cause (NC), Information disclosure (IFD), Tampering (TMP), Relay Attack (RA), Hijacking (HJK), Phishing (PHG), Physical cause (PC), Cyberattack (CYBB) which are classified by our model and displayed along with the threat extracted output.

There are certain additional functionalities such as a back button in the classified output page and threat extracted output page to make the user comfortable to navigate through the different pages of the application, and a loading screen when there is an *ajax* request made to fetch the threat classified output and threat extracted output from the backend.

6.1.3 Components and Features

Components are rightly defined as the essential building blocks of any application created with React, and a single app most often consists of many components. A component is in essence, a piece of the user interface - splitting the user interface into reusable and independent parts, each of which can be processed separately. Similarly, our application is a combination of several reusable and independent components such as header, footer, pageCount, select API dropdown which are used throughout the application.

We have used the APP component as the parent component containing all the state variable objects and event handlers declared and defined which are used by the child components as the *props* (properties). We have written a promise call which is an asynchronous function that always returns a promise with data to fetch the live news from newsAPI open-source and a promise call to fetch the live news from NYTimes API as shown in figure 6.1.3 (a)

```
import axios from 'axios';
const apiKey = "28121d5045cf4256a730f6ac14df004c";
export const baseURL = `http://newsapi.org/v2/`;
const todayDate = new Date().toJSON().slice(0,10);
export const newStoriesURL = `${baseURL}everything?q=automobiles&apiKey=${apiKey}`;
// async function always returns a promise with data
// Promise call to fetch the data from newsAPI | added by Kruthika
export const getNews = async (pageLength) => {
    // destructuring the the API fetched data by using {data}
    const result = await axios.get(`${newStoriesURL}&pageSize=${pageLength}&from=${todayDate}`).then(({data}) => data && data);
    return result;
}
```

```
// NY times API free
import axios from 'axios';
// import {selectRequiredFields} from "../utils/selectRequiredFields"
const apiKey = "uh9JpCCB4mwSMC1YT6oeYwOQXvkc8JhR";
export const baseURL = `https://api.nytimes.com/svc/news/v3/content/nyt/`;
const todayDate = new Date().toJSON().slice(0,10);
export const newStoriesURL = `${baseURL}automobiles.json,business.json`;

// async function always returns a promise with data
// Promise call to fetch the data from NYTimes | added by Kruthika
export const getNews = async (pageLength) => {
    // destructuring the the API fetched data by using {data}
    const result = await axios.get(`${newStoriesURL}&limit=${pageLength}&published_date=${todayDate}?api-key=${apiKey}`).then(({data}) => data && data);
    return result;
}
```

Figure: 6.1.3 (a)

A list of items fetched from both APIs is rendered to the homepage in the form of the collapsible list using the React-Bootstrap accordion element as shown in figure 6.1.3 (b). The JSON format received from the sources has different key-value pairs which made us use two different components for two of the news sources. Similarly, the list of user-added feeds is displayed in the collapsible form using the accordion element of react-bootstrap.

```
export const NewsApiOutput =    ({responseData,nerOutputClick}) => {
    return(
        // Mapping through the response object to render the items  added by Kruthika
        responseData.map((outputItem,i) =>
        <Card>
            <Accordion.Toggle as={Card.Header} eventKey={i === 0 ? "0" : i} >
                {outputItem.title}
            </Accordion.Toggle>
            <Accordion.Collapse eventKey={i === 0 ? "0" : i}>
                <Card.Body>
                    <div className="row">
                        <div className="col-md-4">
                            Content:
                        </div>
                        <div className="col-md-8" key={i}>
                            {outputItem.content === null ? "-" : outputItem.content}
                        </div>
                    </div>
                    <div className="row">
                        <div className="col-md-4">
                            Source:
                        </div>
                        <div className="col-md-8" key={i}>
                            <a href={outputItem.url} target="_blank" rel="noreferrer">{outputItem.url}</a>
                        </div>
                    </div>
                    <div className="row m-3 float-right">
                        <div className="col-md-4">
                            <button className="btn btn-primary btn-sm btn-general" onClick={() => nerOutputClick(outputItem, true)}>NER</button>
                        </div>
                    </div>
                </Card.Body>
            </Accordion.Collapse>
        </Card>
        )
    )
}
```

Figure: 6.1.3 (b)

The fresh promise call is made each time the user changes the API and page length by using the state variable which keeps track of the page count. We have also created a state variable to keep track of the loader display when the classified or spacy output is being fetched from the backend. We have used Axios library which is a popular promise-based HTTP client that sports an easy-to-use API making HTTP requests to fetch or save data which is one of the most common tasks a client-side JavaScript application will need to do. We have used the post method of Axios to send the complete data set fetched from the free sources into the backend (flask) and receive the classified data as the response from the model as shown in the event handler function of figure 6.1.3 (c). Similarly, the function is written for *NYTimes API* and *User Feed* as well.

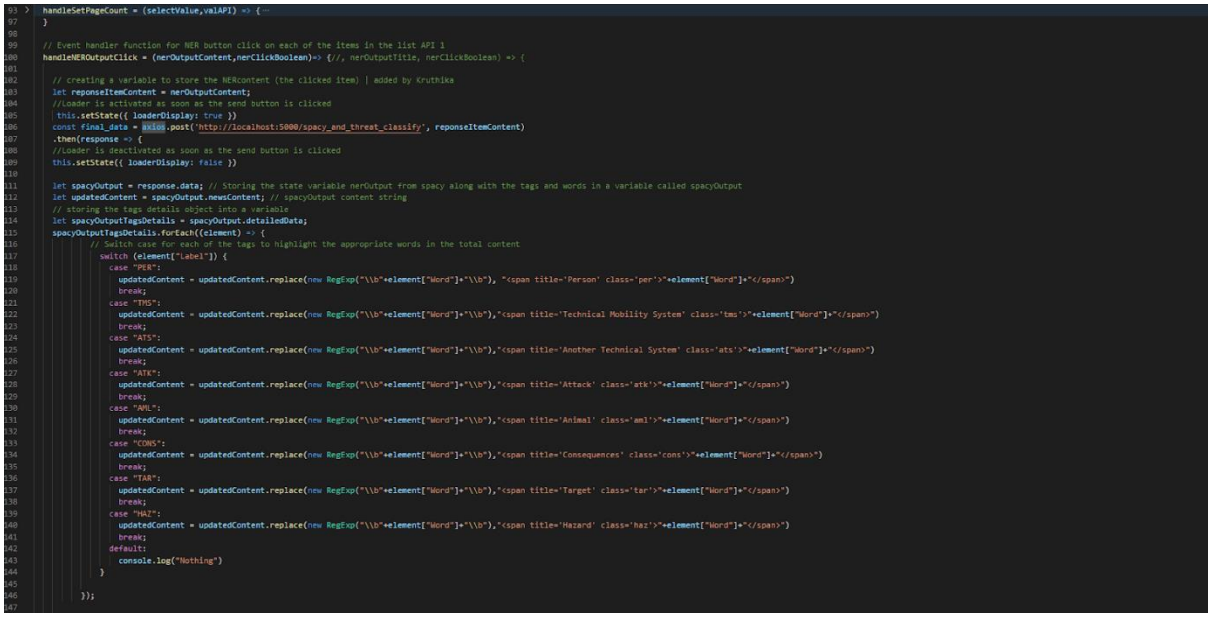
```
handleCompleteNYJSONClick = (completeData) => {
  //Loader is activated as soon as the send button is clicked
 this.setState({ loaderDisplay: true })
 const final_data = axios.post('http://localhost:5000/classify_news_list_nyi', completeData)
 .then(response => {
  //Loader is deactivated as soon as the send button is clicked
  this.setState({ loaderDisplay: false })
  this.setState({newsAPIOutputItems: response.data})
  //Displaying purpose change the boolean values accordingly
  if(!response.data.isEmpty()){
   this.setState({isNewsAPIOutput: true})
   this.setState({isNYAPIOutput: false})
  }
  return final_data
 })
 .catch(error => {
  console.log("Error Message",error)
  });

}

```

Figure: 6.1.3 (c)

The threat classified list of items is sent to the backend model using the post method of the *Axios* which fetch the threat-specific words (tags) of the content and threat type. The threat-specific words will be highlighted with a distinct set of colours along with the titles for each of the tags, and the threat type will be displayed to the user. Figure 6.1.3 (d) shows the event handler function for the above-mentioned event.



```
handleSetPageCount = (selectValue,valAPI) => {…
}

// Event handler function for NER button click on each of the items in the list API 1
handleNEROutputClick = (nerOutputContent,nerClickBoolean)=> {//, nerOutputTitle, nerClickBoolean) => {

  // creating a variable to store the NERcontent (the clicked item) | added by Kruthika
  let reponseItemContent = nerOutputContent;
  //Loader is activated as soon as the send button is clicked
  this.setState({ loaderDisplay: true })
  const final_data = axios.post('http://localhost:5000/spacy_and_threat_classify', reponseItemContent)
  .then(response => {
  //Loader is deactivated as soon as the send button is clicked
  this.setState({ loaderDisplay: false })

  let spacyOutput = response.data; // Storing the state variable nerOutput from spacy along with the tags and words in a variable called spacyOutput
  let updatedContent = spacyOutput.newsContent; // spacyOutput content string
  // storing the tags details object into a variable
  let spacyOutputTagsDetails = spacyOutput.detailedData;
  spacyOutputTagsDetails.forEach((element) => {
        // Switch case for each of the tags to highlight the appropriate words in the total content
        switch (element["Label"]) {
          case "PER":
            updatedContent = updatedContent.replace(new RegExp("\\b"+element["Word"]+"\\b"), "<span title='Person' class='per'>"+element["Word"]+"</span>")
            break;
          case "TMS":
            updatedContent = updatedContent.replace(new RegExp("\\b"+element["Word"]+"\\b"),"<span title='Technical Mobility System' class='tms'>"+element["Word"]+"</span>")
            break;
          case "ATS":
            updatedContent = updatedContent.replace(new RegExp("\\b"+element["Word"]+"\\b"),"<span title='Another Technical System' class='ats'>"+element["Word"]+"</span>")
            break;
          case "ATK":
            updatedContent = updatedContent.replace(new RegExp("\\b"+element["Word"]+"\\b"),"<span title='Attack' class='atk'>"+element["Word"]+"</span>")
            break;
          case "AML":
            updatedContent = updatedContent.replace(new RegExp("\\b"+element["Word"]+"\\b"),"<span title='Animal' class='aml'>"+element["Word"]+"</span>")
            break;
          case "CONS":
            updatedContent = updatedContent.replace(new RegExp("\\b"+element["Word"]+"\\b"),"<span title='Consequences' class='cons'>"+element["Word"]+"</span>")
            break;
          case "TAR":
            updatedContent = updatedContent.replace(new RegExp("\\b"+element["Word"]+"\\b"),"<span title='Target' class='tar'>"+element["Word"]+"</span>")
            break;
          case "HAZ":
            updatedContent = updatedContent.replace(new RegExp("\\b"+element["Word"]+"\\b"),"<span title='Hazard' class='haz'>"+element["Word"]+"</span>")
            break;
          default:
            console.log("Nothing")
        }

      });
```

Figure: 6.1.3 (d)

The response data received from the backend is in the format as shown in figure 6.1.3(e) (console snippet) containing the newsContent and an array *detailedData* having the list of threat-specific words(tags) and their respective labels. The words are highlighted in the content using their labels by adding span tags for each of the words with some predefined class names defined in the *Cascading Style Sheets (CSS)* file for colouring using switch case statements as shown in figure 6.1.3 (e).

```
Response Data                                                                App.js:115
{detailedData: Array(12), newsContent: "One of the main reasons why it's so hard to get ho…phones in many regions outside the United States.", tagsWordMap: {…}, threatClassification: "brute_force"}
  detailedData: Array(12)
    0: {End: 119, Label: "PER", Start: 110, Word: "chipmakers"}
    1: {End: 128, Label: "ATS", Start: 126, Word: "AMD"}
    2: {End: 221, Label: "ATS", Start: 211, Word: "smartphones"}
    3: {End: 372, Label: "ATS", Start: 368, Word: "chips"}
    4: {End: 595, Label: "ATS", Start: 591, Word: "chips"}
    5: {End: 629, Label: "ATS", Start: 623, Word: "Android"}
    6: {End: 867, Label: "ATS", Start: 858, Word: "smartphone"}
    7: {End: 1264, Label: "ATS", Start: 1254, Word: "smartphones"}
    8: {End: 1360, Label: "ATS", Start: 1356, Word: "chips"}
    9: {End: 1436, Label: "ATS", Start: 1426, Word: "smartphones"}
    10: {End: 1719, Label: "ATS", Start: 1716, Word: "chip"}
    11: {End: 1759, Label: "ATS", Start: 1749, Word: "smartphones"}
    length: 12
```

Figure 6.1.3 (e)

Apart from the aforementioned event handlers, we have a set of different handlers such as *handleNewsBackButton, handleNERBackButtonClick, handleDropDownChangeAPI* which will be handed by just changing the state variables containing the boolean values. The application is a single page application meaning that everything happens asynchronously without refreshing the page. We are handling the hiding and displaying of the components based on the conditional or ternary operators.

Git path:

For the Frontend**:** In branch master_AI-UC follow AI-UC_2021 -> AI BATMAN THREAT EXTRACTOR -> BATMAN_Frontend

6.2 AI Backend

6.2.1 Features Implemented in Backend

6.2.1.1 Extraction and Classification

Input: A JSON request is sent from frontend and received by flask via http POST call as http POST carries request parameters in the message body that makes it secure for information transfer from client to server.

Task: The Json data contain information about news APIs that need to be classified as threat and non-threat and only Threat related News needs to be returned to the frontend as a Json structure.

Description: The JSON data contains all the information of selected news APIs and is received from frontend. The main task is to classify the news in threat and non-threat. But before the classification, one important task is to extract relevant information from the Json data of News API. For the extraction of content, author name, title of news for both APIS (News API and NYTimes) different methods of extraction is performed because of different keys in the JSON structures of both the APIS. So, the following details for the extraction from both the APIs is as follows:

Extraction

a. For News API Extraction of Data

After receiving the JSON data through POST call using request.json() and storing it in a variable, we used **newspaper3k library** and next we imported Article class for the extraction. Newspaper library is extremely useful for the extraction, parsing and detection task as if the language is not specified, it can automatically detect the language of the information. As newspapers use some advanced algorithms along with web scraping for the extraction of only useful information, its performance is good with online news sites [15].

Steps for extracting information of News API is as follows:

1. First is to create an instance of News we use URLs that is in Json data and create a list in which the urls are stored as article = Article(urlitem).
2. Second step is to download the news using article.download() and parse using article.parse() to extract main readable information from the news.

3. After parsing we stored the content, author, title, description of news in a list which will be used for the classification by using BERT model.

```
""" This function is used to classify entire news list and return threat news only"""
@app.route('/classify_news_list', methods=['GET','POST'])
def news_list_classification():
    requestData = request.json
    classifiedNewsResult = []
    # File opened to write the Json output of classification
    fp = open('news_list_format.json', 'w')

    urlList = []

    for articleItem in requestData:
        urlList.append(articleItem["url"])
    """For the Extraction of news content and other relevant information"""
    content = []
    for urlItem in urlList:
        article = Article(urlItem)
        article.download()

        if (article.download_state == 2):
            article.parse()
            article_dict = {}
            article_dict = {'Author': article.authors, 'url': article.source_url, 'text': article.text,
                            'title': article.title, 'summary': article.summary, 'description':article.meta_description}
            content.append(article_dict)
```

Figure 6.2.1 (a): First Relevant news information extracted using newspaper3k and then the news classification is performed on extracted news list contents in the above function in Flask**.**

b. For NYTimes API Extraction of Data

For the NYTimes API information extraction we used Beautiful Soup library that is created on top of Html parsing libraries like html.parser and so on [16]. After using this library, it will contain all information of the NYTimes in nested structure that needs to be extracted and further we need to structure the contents by removing irrelevant parts from the content. Steps followed for the extraction of content, author, and title for the NYTimes are as follows:

1. Use Beautiful soup to extract data and parse the data using html.parser.
2. The extracted data needs to be structured and all unnecessary information needs to be removed from the extracted content. For this, we will kill all script and style and extract the content and after that to get only the body of the pge we did this by removing the subheading from the content by using content.body.get_text.
3. Next step is to remove spaces and any trailing spaces on each line of content and drop any blank line.

These steps we followed to get the content from NYTimes and the author and title of the news so that the data can be forwarded for the classification. A separate function is created for the extraction of NYTimes and for the news classification [def news_list_classification_nyi()].

```
""" This function is used to classify entire news list of NYI Times and return threat news only"""
@app.route('/classify_news_list_nyi', methods=['GET','POST'])
def news_list_classification_nyi():
    request_data = request.json
    # File opened to write the Json output of classification
    fp = open('news_list_format_nyi.json', 'w')

    classifiedNewsResult = []
    """to extract content by using BeautifulSoup"""
    for traverse in request_data:
        html = urlopen(traverse["url"]).read()
        content = BeautifulSoup(html, features="html.parser")

        # kill all script and style elements
        for script in content(["script", "style"]):
            script.extract()  # rip it out
        text = content.body.get_text(separator=' ')

        # break into lines and remove leading and trailing space on each
        lines = (line.strip() for line in text.splitlines())
        # break multi-headlines into a line each
        chunks = (phrase.strip() for line in lines for phrase in line.split("  "))
```

Figure 6.2.1 (b): For the extraction of NYTimes API news content extraction and classification implemented in the above function in Flask.

Classification

After selection of one the two APIs, and the json request is sent to backend and after performing the extraction, the next step is to classify the news list as threat and non-threat. For the news classification, we used our trained BERT model. The steps we followed for the implementation of classification is as follows:

1. First step is to download the trained model on our custom dataset and specify the trained model name in the BERT_MODEL variable in the **news_classification.py** file.
2. After specifying all the parameters required to execute the model, we created a function with name prediction_result() in class classify that is used for the calculation of the SoftMax value for news data.
3. In flask file **app.py**, after the extraction of news content, in function (either news_list_classification for News API or news_list_classification_nyi for NYTimes), a for loop is executed that will iterate for each news content in the list created for all news obtained from either of selected APIS.
4. In the loop, a prediction_result function is called that will obtain the probability for each label [non-threat, threat] and then return the index of the maximum probability using argmax. If the value returned is 0 then it is non-threat news and if 1 then it is threat news.
5. Now, in the last step the value returned is 1 for a news then the information of the news is stored in a list. The loop will be executed till the length of the list that contains news.
6. After getting all threat classified news information in the list, a json structure is returned using Jsonify that serializes the JSON and wraps it in a response object as in figure 6.2.1(d).

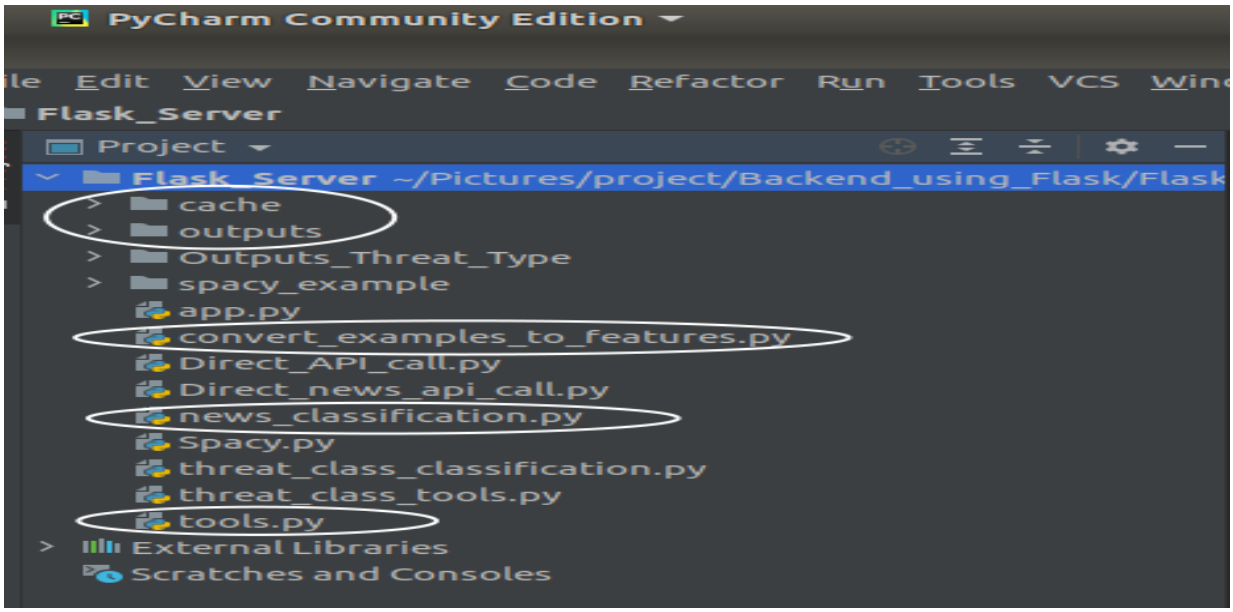


Figure 6.2.1 (c): Files required for News Classification for both APIs.


```
▼0:
   Classify: "Threat"
   Prediction_result: 1
  ▶author: ["David Shepardson"]
   content: "WASHINGTON (Reuters) - Fiat Chrysler Automobiles pleaded guilty on Monday to charges it conspired with company executives to make illeg…
   description: "Fiat Chrysler Automobiles pleaded guilty on Monday to charges it conspired with company executives to make illegal, lavish gifts to…
   title: "Fiat Chrysler Automobiles pleads guilty in U.S. labor probe"
   url: "https://www.reuters.com"
  ▶__proto__: Object
▶1: {Classify: "Threat", Prediction_result: 1, author: Array(0), content: "The Trump administration deleted the U.S. Environm…timistic' climate chan…
▶2: {Classify: "Threat", Prediction_result: 1, author: Array(0), content: "One of the main reasons why it's so hard to get ho…phones in many regions…
▶3: {Classify: "Threat", Prediction_result: 1, author: Array(5), content: "The industry is having to stall more plants to con… and Cadillac's sedans…
▶4: {Classify: "Threat", Prediction_result: 1, author: Array(1), content: "Ellen Lee is a portfolio manager at the $45.5 bill…ring, Broadcom (AVGO),…
▶5: {Classify: "Threat", Prediction_result: 1, author: Array(1), content: "Discussions between big automakers and Apple have …ues to churn, and late…
▶6: {Classify: "Threat", Prediction_result: 1, author: Array(1), content: "There's a solid reason why Autoblog always lists s…lick here to read the …
▶7: {Classify: "Threat", Prediction_result: 1, author: Array(1), content: "ززززShare this Story: The new Canada: How COVID-… issue signing you up.…
▶8: {Classify: "Threat", Prediction_result: 1, author: Array(0), content: "lززI admit to being both amused but also a little …biles will have the wo…
▶9: {Classify: "Threat", Prediction_result: 1, author: Array(0), content: "If these things are not considered in the early st…its original look? You…
▶10: {Classify: "Threat", Prediction_result: 1, author: Array(0), content: "If you can read this, either the style sheet didn'…aring your browser ca…
 length: 11
▶__proto__: Array(0)
```


Figure 6.2.1 (d): Json Response to frontend that containing all news that were classified as Threat news.

Output: A response object that contains JSON structure for the threat classified news will be returned to the frontend for the display of the list.

Python Scripts used: convert_example_to_features.py, tools.py, news_classification.py for the trained BERT model used for classification and for flask is app.py python file.

Files used: cache that contain trained model in tar file, **outputs** file that contain the vocab file and all other important parts.

Git path:

For the **Flask server:** In branch master_AI-UC follow AI-UC_2021 -> AI BATMAN THREAT EXTRACTOR -> BATMAN_Backend -> Flask_Server.

For **BERT classification:** In final file follow: - AI-UC_2021 -> NLP_Fundamentals ->BERT/NEWS_CLASSIFICATION_USING_BERT-
>News_Classification_using_BERT_from_Hugging_face

Final Branch Git Link: [11] [12]

6.2.1.2 Threat Extraction

Task: The main purpose is to extract the different threat information for NER tasks using Spacy for the content of news we get in a json request from frontend. The extracted information needs to be returned in Json structure.

Input: A Json data sent after selection of single news from threat classified News List displayed using frontend through a POST call to the flask api.

Description: In the Json request that is received in the backend through flask contains the information like news content, author name, title, and URL. The following steps were followed for the execution and getting result for threat extraction which are as follows:

1. First, we download the trained spacy model and all related files like vocabulary files and other important files created during training of the spacy model on our custom dataset for the threat extraction that is file **spacy_example** file.
2. A python file is created with the name **Spacy.py** that will load the trained model and create a function test_model_on_single_news(test_sentence) for getting the NER result for threat extraction on single news content.
3. In the next step in flask code **app.py**, a function spacy_and_threat_classify is created that will call the spacy model function(test_model_on_single_news) for the extraction of different tags information related to threat extraction for words that will be signifying labels. The list that contains the news content will be passed to the function for the NER task of threat extraction.
4. After getting all words with their corresponding label and the position of words in the news content, it will be stored in a list and the final Json structure is returned along with arrays for each label that contains the grouped words for the label.

Output: A response object that contains JSON structure for the threat extraction will be returned to the frontend for the display of words in highlighted way with hovering functionality to explore more threat information.

Python Scripts used: Spacy.py for Spacy model, app.py for the flask server.

Files used: spacy_example that contains the trained spacy model and all other important files for the extraction.

Final Branch link: [13]
Link for Spacy: [14]
Git Path:
For the **Flask server:** In branch master_AI-UC follow AI-UC_2021 -> AI_BATMAN_THREAT_EXTRACTOR -> BATMAN_Backend -> Flask_Server.
For **Threat Extraction Using Spacy**: Follow path: AI-UC_2021->NLP_Fundamentals->spaCy->spaCy_Based_NER.ipynb

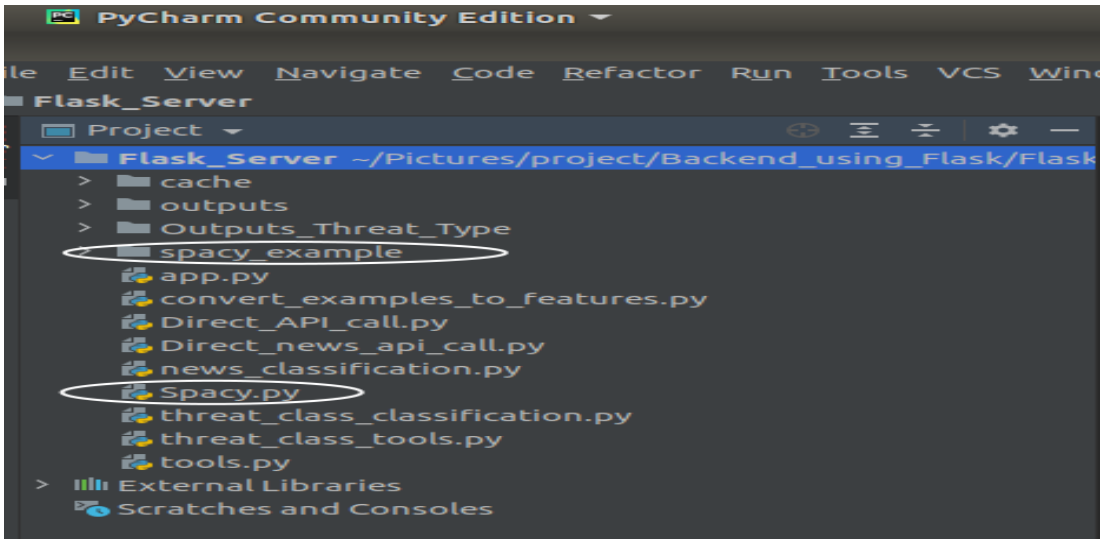


Figure 6.2.1.2 (a): Files important for Threat Extraction using Spacy.

6.2.1.3 Threat Type Identification

Along with the threat extraction information, another result needs to be added that is the identification of threat type.

Task: The main purpose is to identify the threat type for the content of news we get in a json request from frontend. The identified threat type information needs to be returned in Json structure along with the threat extraction result.

Input: A Json data sent from frontend for single news from the threat classified News List through a POST call to the flask api.

Description: The JSON request from frontend contains information of selected news. Along with the threat extraction result, it is also required to identify and return the threat type information for news content. For this task, we used the BERT model and trained the model for identification of threat by using our custom dataset similarly we did for news classification tasks. There are steps that we followed for the identification task implementation which are as follows:

1. Our initial step is to download the trained BERT model for the threat type identification by using a custom dataset. In the **threat_class_classification.py** file before loading the model it was required to mention the model we want to load in the BERT_MODEL variable.
2. After mentioning the values for some parameters and loading our trained model, we created a function threat_prediction_result() in class threat_classify that is calculating the softmax value for the labels for our news content.
3. As we have more than two labels, so to store the probability values for all labels we returned an array containing the values for all labels.

4. Now in flask file **app.py**, after getting the news content for a single news threat_prediction_result() function is called, and the values are stored in an array.
5. Next is to get the index of the maximum probability is done and by using the index, we identified the threat type by getting a label from the label list through index value.
6. After getting the threat type information, the final Json structure contains the threat extraction information along with the identified threat type information and is returned to the frontend using jsonify as shown in figure 6.2.1.3(b).

Output: A response object that contains JSON structure for the threat extraction and threat type information will be returned to the frontend for the display of words in highlighted way along with the identified threat type.

Python Files used: For threat type identification - threat_class_classification.py, threat_class_tools.py, for flask api- app.py.

Files used: cache for the trained model, **Outputs_Threat_Type** for other important files like vocabulary file, weight file and so on.

Git path: Flask server: For Final File in final master_AI-UC and follow path: - AI-UC_2021 -> AI_BATMAN_THREAT_EXTRACTOR -> BATMAN_Backend -> Flask_Server

For **Threat Type Identification:** AI-UC_2021 -> AI BATMAN THREAT EXTRACTOR -> Backend ->BERT->Threat_Type_Classification in final file.

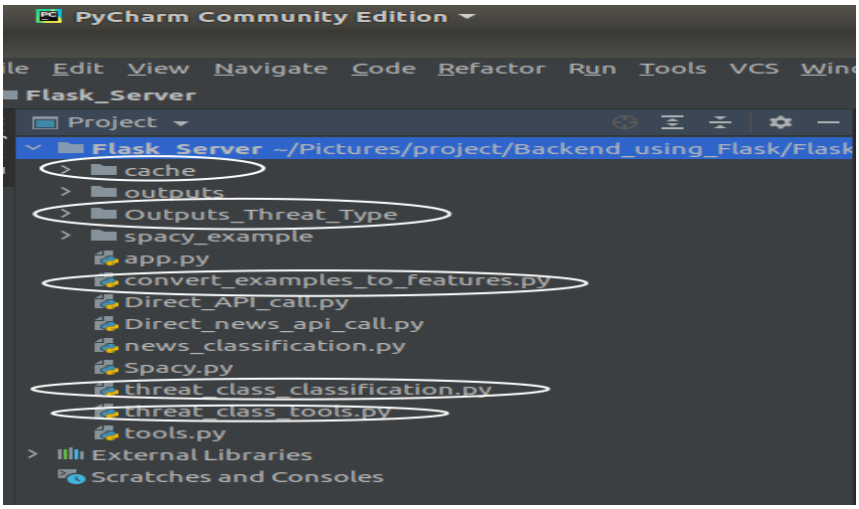


Figure 6.2.1.3.: Files important for the Threat Type identification in Backend.

```python
""" To get Extraction Result and Threat Type """
@app.route('/spacy_and_threat_classify', methods=['GET', 'POST'])
def spacy_and_threat_classify():
    requestData = request.json
    finalResultObj = {}
    """ to write spacy output json in file"""
    fp = open('extraction_and_threat_type_result.json', 'w')

    """Function called for extraction result and threat type information"""
    finalResultObj = extractAndReturnThreatData(requestData)
    j = json.dump(finalResultObj, fp)
    return flask.jsonify(finalResultObj)



'''common function to extract and prepare proper threat output for display using Spacy model for Ner and Threat Type Classification'''
def extractAndReturnThreatData(requestData):

    """ append text in list for spacy"""
    data_list = []
    data_list.append(requestData)

    ### to remove '\n\n' from news
    news_text = []
    for data in data_list:
        """ To get threat type information """
        result_for_threat_type = threat_class_classification.threat_prediction_result(data['content'])
        threatTypeResultArray = result_for_threat_type.ravel()
```

Figure 6.2.1.3 (a): Threat Extraction and Threat Type identification Json response sent using above function in Flask.

```
spacyOutput                                                                    App.js:113
▼ {detailedData: Array(12), newsContent: "WASHINGTON (Reuters) - Fiat Chrysler Automobiles p…engaged in a longstanding pattern of corruption."", tagsWordMap: {…}, threatClassification: "brute_force"}
  ▼ detailedData: Array(12)
    ▶ 0: {End: 178, Label: "ATS", Start: 176, Word: "UAW"}
    ▶ 1: {End: 275, Label: "ATS", Start: 268, Word: "Chrysler"}
    ▶ 2: {End: 292, Label: "ATS", Start: 290, Word: "FCA"}
    ▶ 3: {End: 298, Label: "ATS", Start: 295, Word: "sign"}
    ▶ 4: {End: 441, Label: "PER", Start: 434, Word: "American"}
    ▶ 5: {End: 769, Label: "CONS", Start: 765, Word: "set a"}
    ▶ 6: {End: 1106, Label: "ATS", Start: 1104, Word: "UAW"}
    ▶ 7: {End: 1251, Label: "ATS", Start: 1245, Word: "Detroit"}
    ▶ 8: {End: 1646, Label: "TMS", Start: 1641, Word: "cigars"}
    ▶ 9: {End: 1761, Label: "PER", Start: 1753, Word: "automaker"}
    ▶ 10: {End: 1851, Label: "ATS", Start: 1849, Word: "UAW"}
    ▶ 11: {End: 2008, Label: "ATS", Start: 2001, Word: "Thursday"}
      length: 12
    ▶ __proto__: Array(0)
    newsContent: "WASHINGTON (Reuters) - Fiat <span title='Another Technical System' class='ats'>Chrysler</span> Automobiles ple…
  ▼ tagsWordMap:
    ▶ ATS: (8) ["UAW", "Chrysler", "FCA", "sign", "UAW", "Detroit", "UAW", "Thursday"]
    ▶ CONS: ["set a"]
    ▶ PER: (2) ["American", "automaker"]
    ▶ TMS: ["cigars"]
    ▶ __proto__: Object
    threatClassification: "brute force"
```

Figure 6.2.1.3 (b): Json Response sent to Frontend that containing selected news Threat Extraction information along with the threat type detail.

6.2.1.4 Input field Data

Task: The task is to first classify the received news from the frontend when the user inserted a sentence or some news in the input field to get classification information and if the news is classified as threat, then the threat extraction and threat type identification result is also returned. Python code Text_data_classify_and_ner_result.py.

Input: A News or a sentence is sent as a json data from frontend through POST call for the classification and threat extraction result on the news.

Description: In the Json sent from frontend contains the sentence news provided by the user in the input field. First task is to classify the input as threat or non-threat. We followed the same procedure for the classification as we did for the news list classification obtained from news apis. The steps for the classification and threat extraction result for the input is as follows:

1. First, we will classify the input by using a trained BERT model, by calling the prediction_result function and get the index.
2. If the news is classified as a threat, then the threat extraction and identification of threat type will be obtained by using the function test_model_on_single_news() for the threat extraction and threat_prediction_result() for the identification of the threat.
    a. A response is sent as a Json Structure that contains the classification information that news is threat, threat extraction result and threat type identification result to the frontend for the display.
3. If the news is classified as non-threat, then no threat extraction and threat type identification operations are performed.
    b. A response is sent as a Json structure that contains only classification results that news is non-threat to the frontend.

Output: If news is classified as threat, then, a json response is sent to the frontend containing all classification (news is classified as threat), threat extraction, and threat type identification information. If news is classified as non-threat, then only classification results are sent (news is classified as non-threat) as shown in figure 6.2.1.4(b).

Python Files used: convert_example_to_features.py, tools.py, news_classification.py for the trained BERT model used for classification, Spacy.py for Spacy model, For threat type identification - threat_class_classification.py, threat_class_tools.py, flask api in app.py,

Files used: cache that contains trained model in tar file, **outputs** file that contain the vocab file and all other important parts, **spacy_example** that contains the trained spacy model and all other important files for the extraction, **cache** for the trained model, **Outputs_Threat_Type** for other important files like vocabulary file, weight file and so on.

Git Path: For Final File in final master_AI-UC and follow path: - AI-UC_2021 -> AI_BATMAN_THREAT_EXTRACTOR -> BATMAN_Backend -> Flask_Server

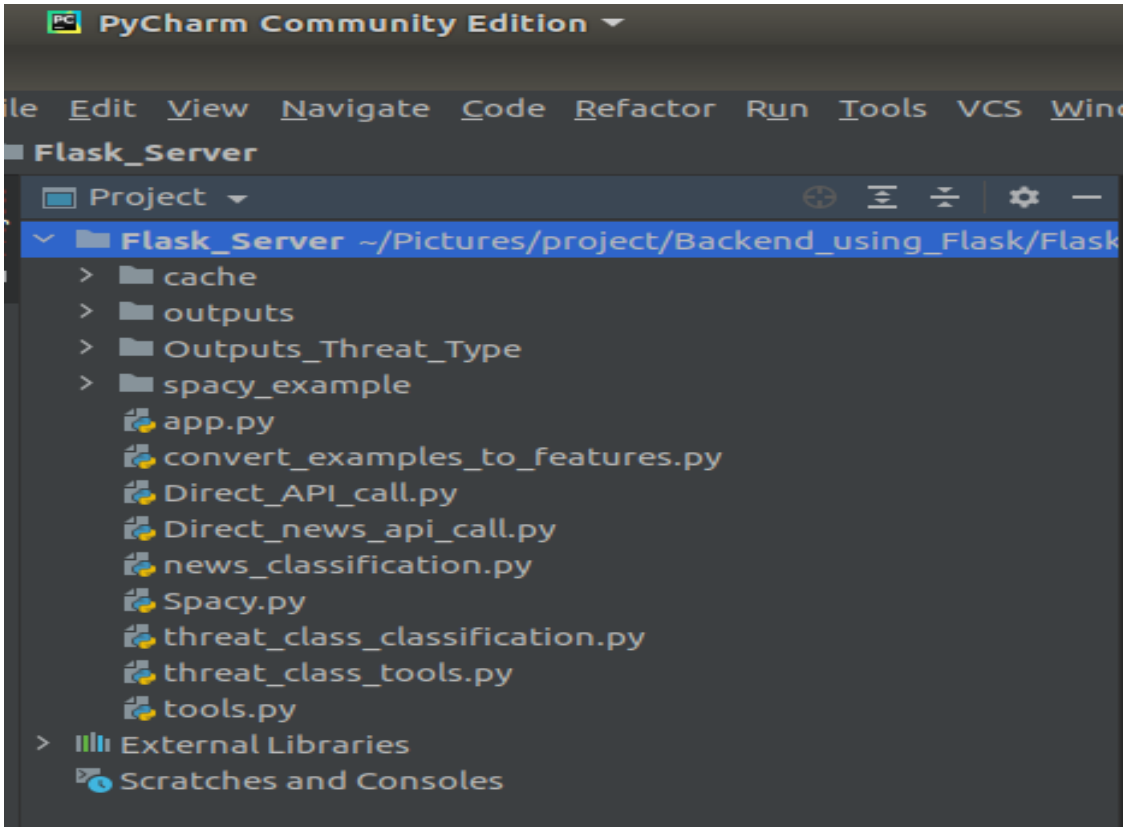


Figure 6.2.1.4: All files important for Input field data

```
""" For CLASSIFICATION OF INPUT FIELD TEXT AND ALSO TO PERFORM NER FUNCTION """

@app.route('/text_data_classify_and_ner_result', methods=['GET', 'POST'])
def text_data_classify_and_ner_result():

    requestData = request.json
    finalJsonToReturn = {} #final json obj to return

    """First to classify news content given in Input Field in Frontend"""

    result_of_classification = news_classification.prediction_result(requestData['content'])
    is_threat = int(result_of_classification)

    #IF THREAT THAN PROCESS THE TEXT FOR SPACY OUTPPUT TO GET EXTRACTION RESULT AND THREAT TYPE INFORMATION
    if is_threat == 1:
        #If want to check the news information after classification
        news = {'news': requestData['content'], 'Classify': 'Threat', 'Predict': is_threat}

        """ Function called for extraction result and threat type information """
        finalJsonToReturn = dict(extractAndReturnThreatData(requestData), **{'Predict': is_threat, 'Classify': 'Threat'})

    #For Non-Threat News
    elif is_threat == 0:
        finalJsonToReturn = {'news': requestData['content'], 'Classify': 'Non Threat', 'Predict': is_threat}

    return flask.jsonify(finalJsonToReturn)
```

Figure 6.2.1.4 (a): Handles Post call for the input field news data sent from frontend and perform classification and threat extraction and threat type identification.

```
▼data:
    Classify: "Threat"
    Predict: 1
  ▼detailedData: Array(1)
    ▶0: {End: 28, Label: "PER", Start: 22, Word: "hackers"}
      length: 1
    ▶__proto__: Array(0)
    newsContent: "In 2015, unidentified hackers have used DNS spoofing techniques to redirect traffic from the offi…
  ▼tagsWordMap:
    ▶PER: ["hackers"]
    ▶__proto__: Object
    threatClassification: "spoofing"
```


Figure 6.2.1.4 (b): Json response sent to frontend for the input provided by user on frontend.

7 INSTALLATION STEPS TO SETUP THE PROJECT

7.1 Backend:

GIT PATH: In branch *master_AI-UC*, the Final folder path is *AI-UC_2021 -> AI_BATMAN_THREAT_EXTRACTOR -> BATMAN_Backend -> Flask_Server.*
Folder name: *BATMAN_Backend* contains ***Flask_Server*** for complete backend and *Packages_required_for_backend.txt* file for packages list required to be installed for Flask_Server to work.
Final Branch link: [15] [16]
For the setup of complete backend in the system follow the steps (1-8) which are as follows:

Essential Tools and Technologies required:

1. Python 3.8 installed.
    a. https://www.python.org/downloads/release/python-380/
2. PyCharm IDE:
    a. https://www.jetbrains.com/pycharm/download/#section=windows
3. Visual Studio code (VScode) IDE installed:
    a. https://code.visualstudio.com/download

Step 1: Before beginning with the final files, it is important to install the technologies used for the implementation and execution of the backend services.
    a. Install python 3.8 version and install Pycharm IDE.

Step 2: Download the folder 'Flask_Server' from the Backend folder from git [15]. That is as follows on git repository:

a. Download the Folder AI_BATMAN_THREAT_EXTRACTOR from AI-UC_2021 and then follow the path: AI_BATMAN_THREAT_EXTRACTOR -> BATMAN_Backend -> Flask_Server as shown in figure 7.1 and 7.1(a).

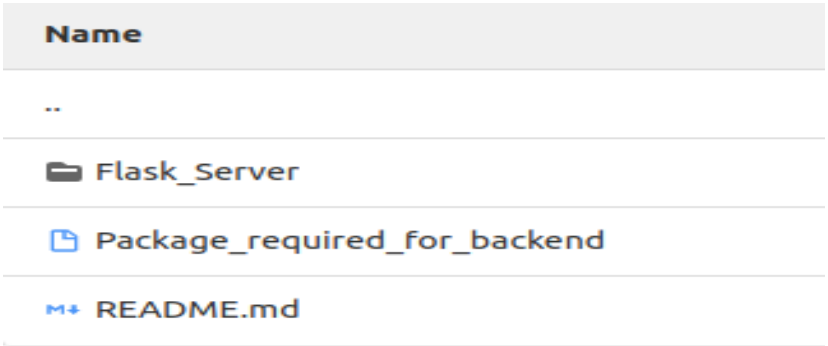


Figure 7.1: The Flask_Server and Package_required_for_backend on git required for Backend setup.

Developers can directly download the entire repository and can find the backend by following the above path.

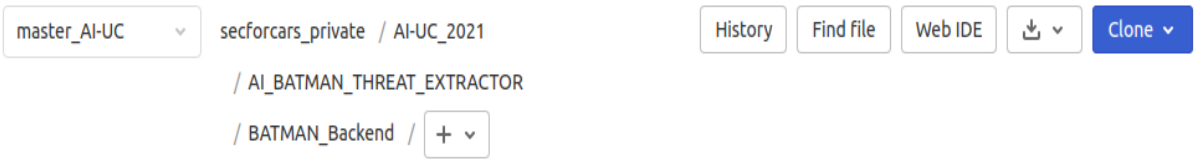


Figure 7.1 (a): Path on Git where the complete Backend files are pushed, the developer can follow this path to get the Flask_Server and package required list.

Step 3: After downloading the folder, put the entire Flask_server folder in pycharmProjects folder in the system. When the Pycharm IDE is installed, a PyCharm project will be created in the C drive of the system. Or developer can move the file anywhere in the system.

Step 4: After moving the entire Flask_Server folder in the pycharmProjects folder or in any other folder in the system, open Pycharm IDE.

a. First open a new project, then go to files and open the Flask_Server by selecting the path where the Flask_server is moved.

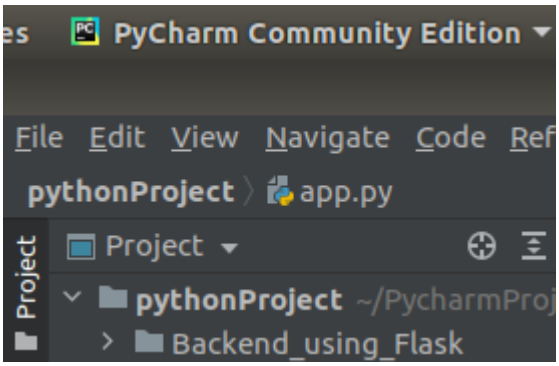


Figure 7.1 (b): Go to File -> Open.

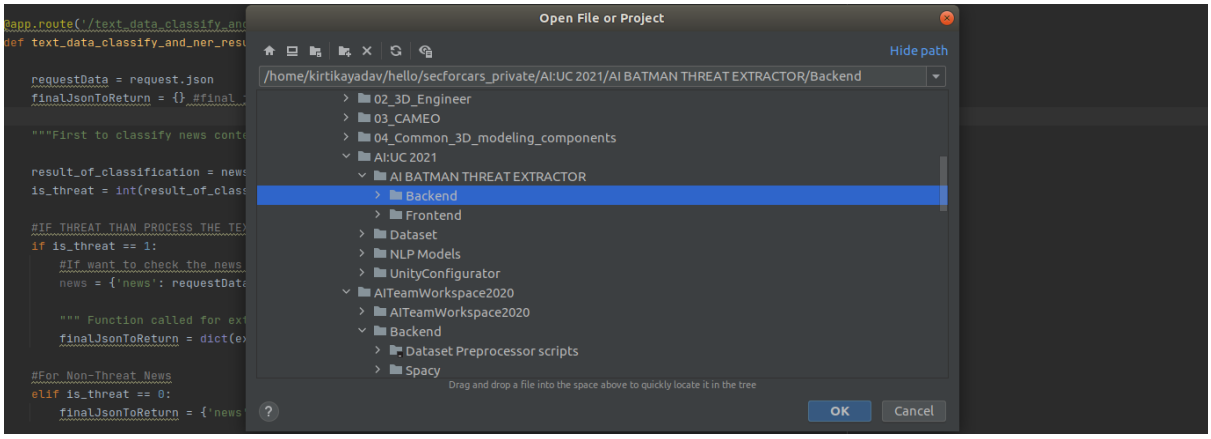


Figure 7.1 (c): Select the folder where Flask_Server folder is downloaded or moved.

b. Developer will get the project structure as shown in the image below:

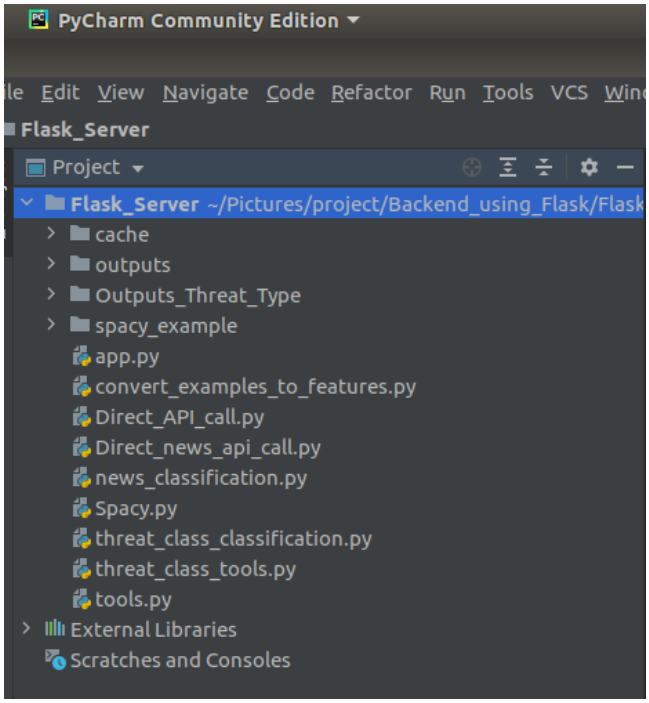


Figure 7.1 (d): Folders in the Flask_Server.

Step 5: After getting this, it is important to install all necessary packages that were required for implementation and execution of Backend Codes.

a. Also Install pip in the system.
b. All the packages and the version are specified in the **Packages_requires_for_backend.txt** file that is in AI-UC 2021 -> AI_BATMAN_THREAT_EXTRACTOR -> BATMAN_Backend.
c. To add packages, follow the below steps:
    1) Go to File-> Settings in PyCharm.
    2) Select Python Interpreter under the python Project: Flask_Server name.

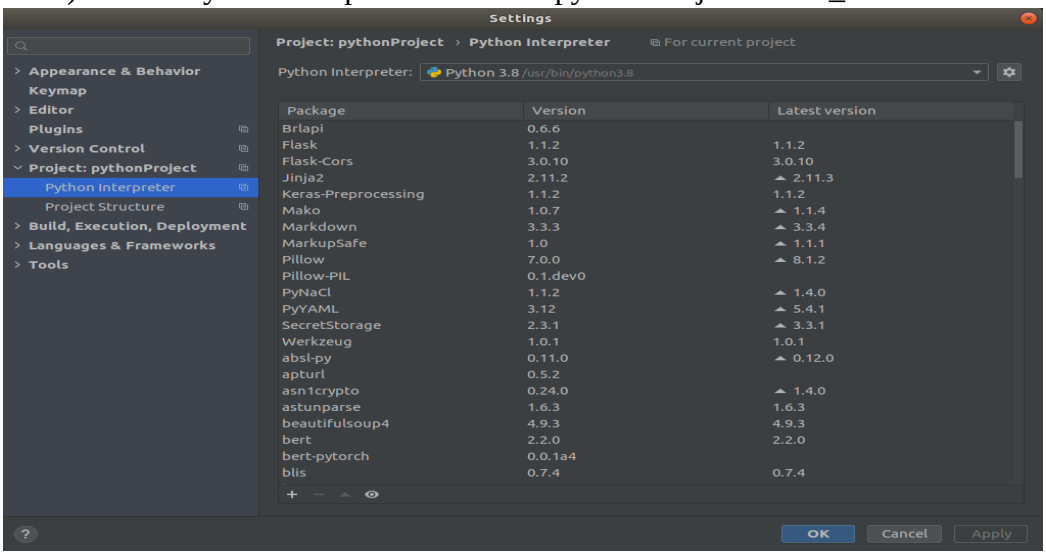


Figure 7.1 (e): After selecting Files-> Settings, go to Python Interpreter.

3) Click '+' below to install packages.

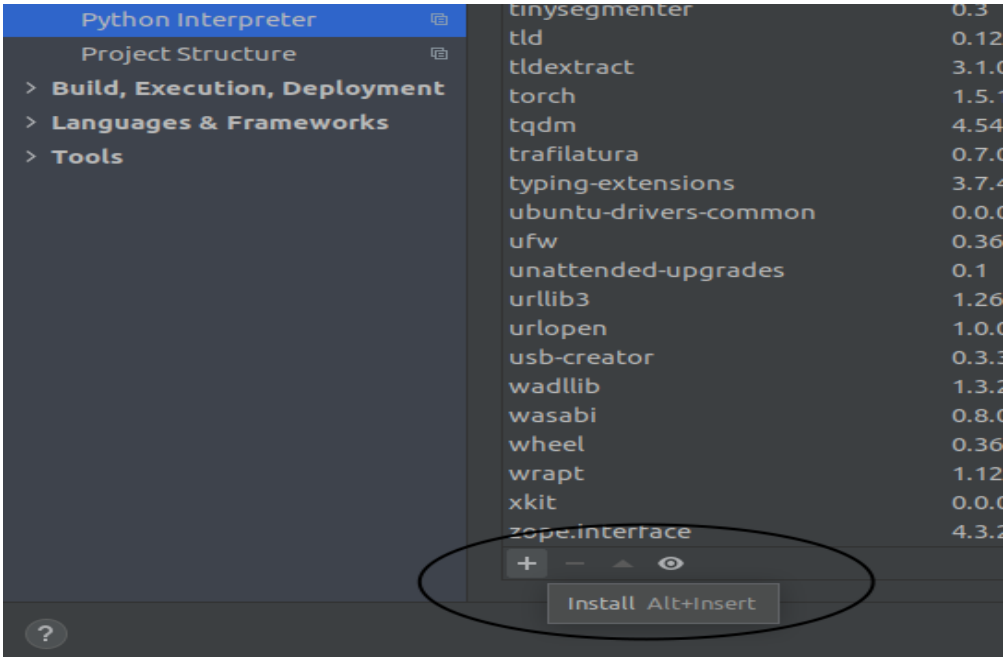


Figure 7.1 (f): Select '+' sign to find packages and install them for the project virtual environment.

4) On the new window of Available Packages, write the package name then on right side specify the version and after this click on Install Package.

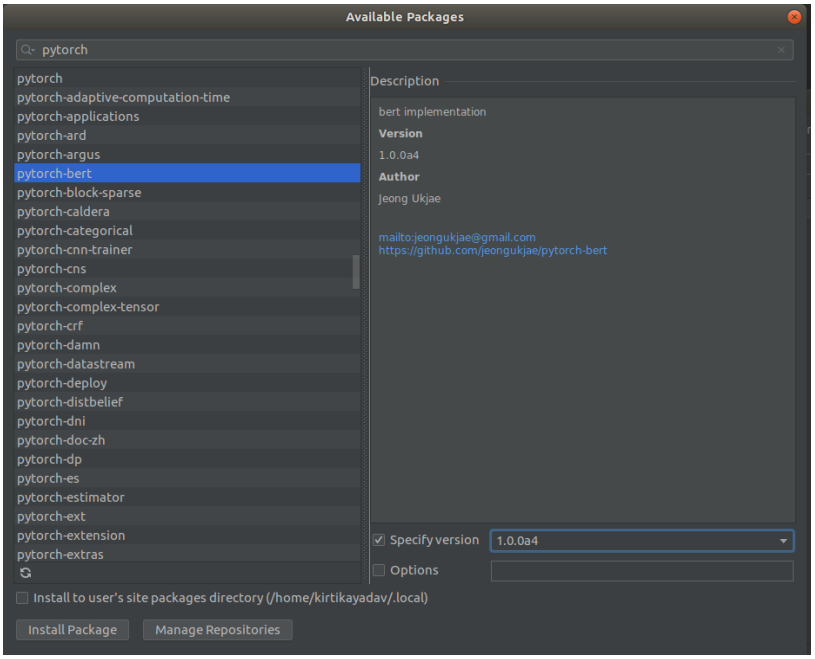


Figure 7.1 (g): Write package name above and specify version also and then install packages.

Step 6: Now, First Execute some .py codes to check if any package is missing. Start in the following order:

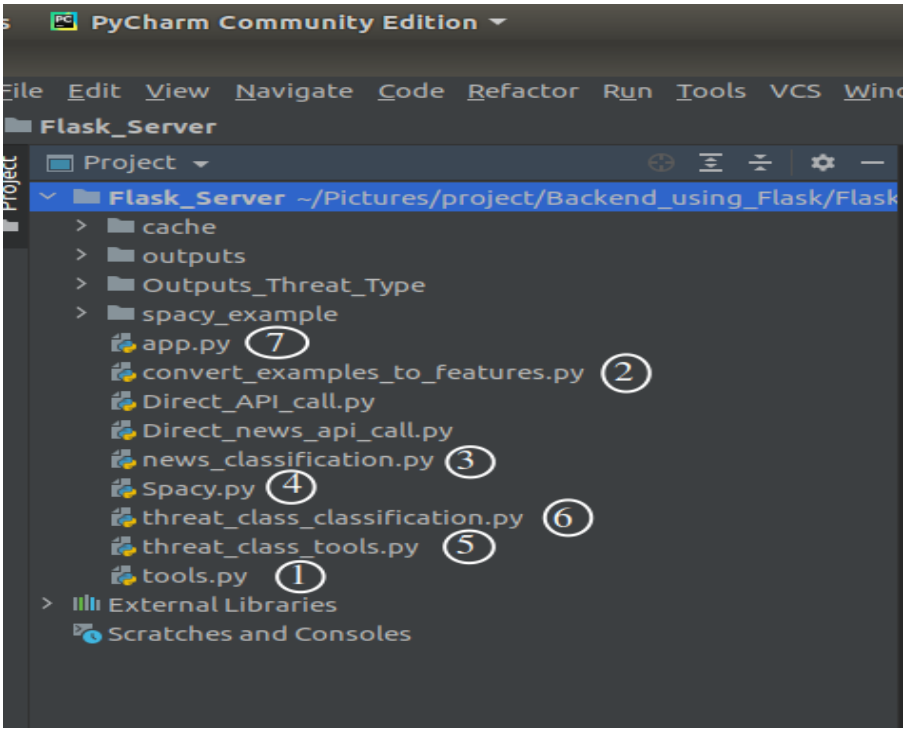


Figure 7.1 (h): Execute or Run the codes in above order (mind the serial numbers in circles) to check for any missing packages to avoid any kind or error.

Step 7: After adding all the packages, execute the final Flask Server code which is in app.py file. To run the app.py file follow either of two steps

a. Either open app.py code in PyCharm right click on the code and select "Run 'app'
b. Or go to the top right of PyCharm, select edit configuration and add app.py name.

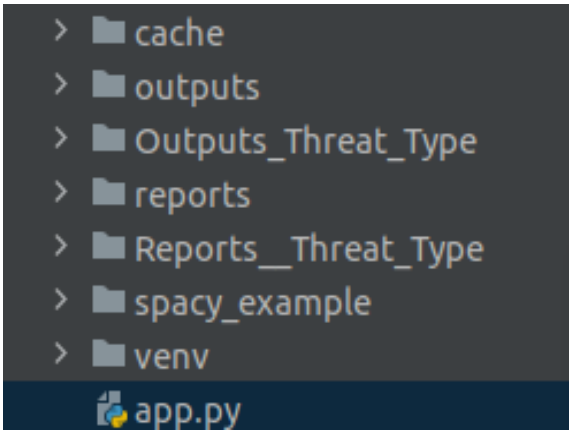
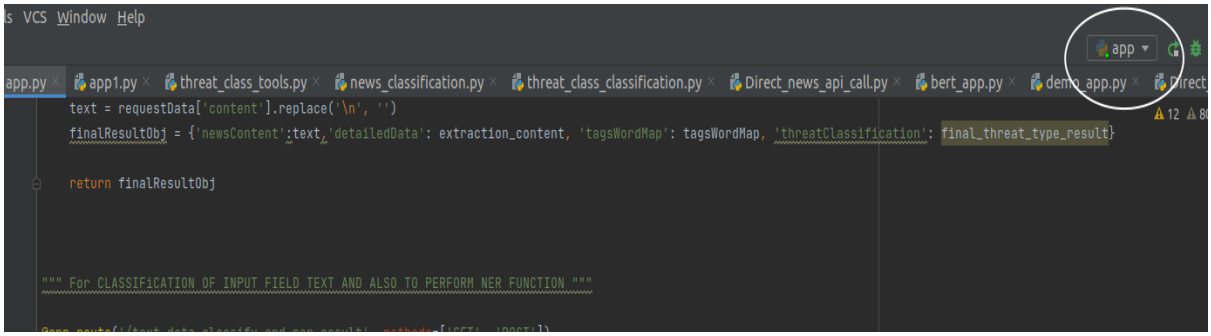
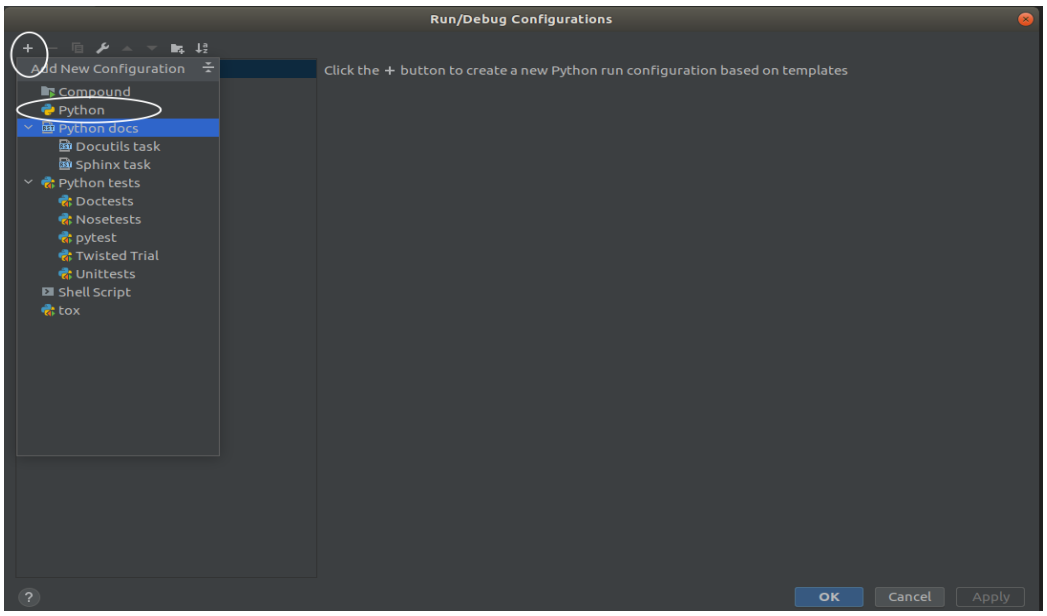


Figure 7.1 (i): Execute app.py file.



(a)



(b)

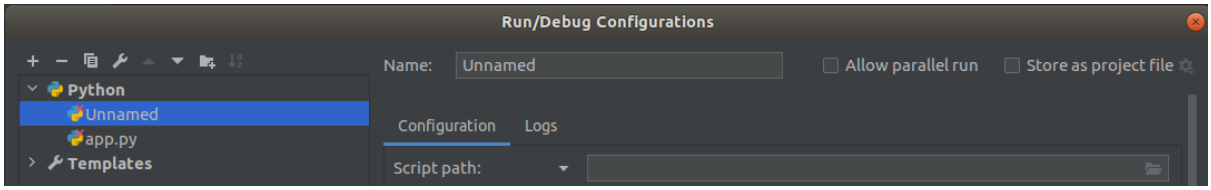


Figure 7.1 (j): Either go to edit configuration and specify app.py name to run by following sequence (a), then (b) and then (c). Specify the app.py name in field of name as shown in c part image.

Step 8: Last step is to wait to get the output with the server url which is INFO:werkzeug: * Running on **http://127.0.0.1:5000/** (Press CTRL+C to quit) and the output will be as follows:

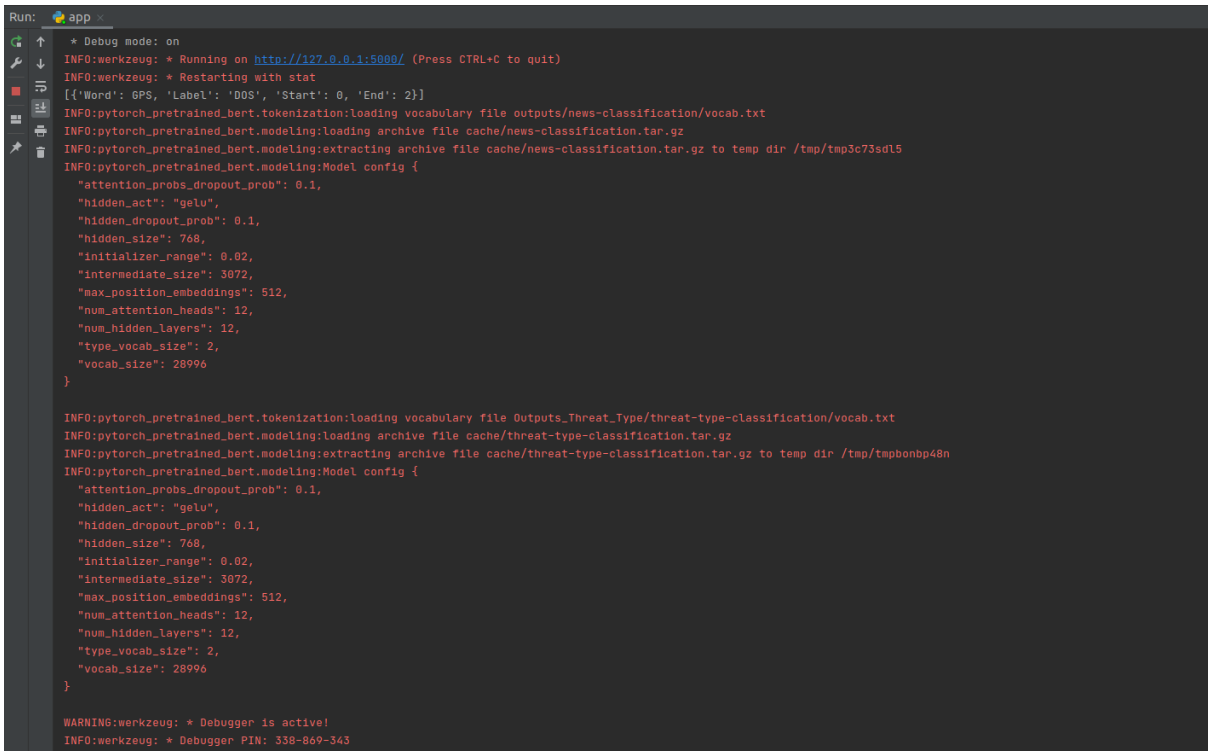


Figure 7.1 (k): Running of Backend and getting url for server.

After the complete setup of Backend and Frontend, First run the app.py file in PyCharm. Then run the Frontend code by using npm start command in the VScode IDE which was used for the Frontend implementation as mentioned in Installation Steps for Frontend.

7.2 Frontend:

To setup the frontend in your system these are the necessary steps to be done once the backend is setup in the system.

1. Download the complete folder named “BATMAN THREAT EXTRACTOR” which is in AI-UC 2021 folder under Master_AI-UC branch. Git Link [20]
2. Unzip the folder in your local system and you can see a folder called “BATMAN_Frontend”
3. Open the folder in Visual Studio and follow the respective commands: This is how the file structure looks in VS Code (you would not see node_modules if the code has been cloned and ran on your system for the very first time)

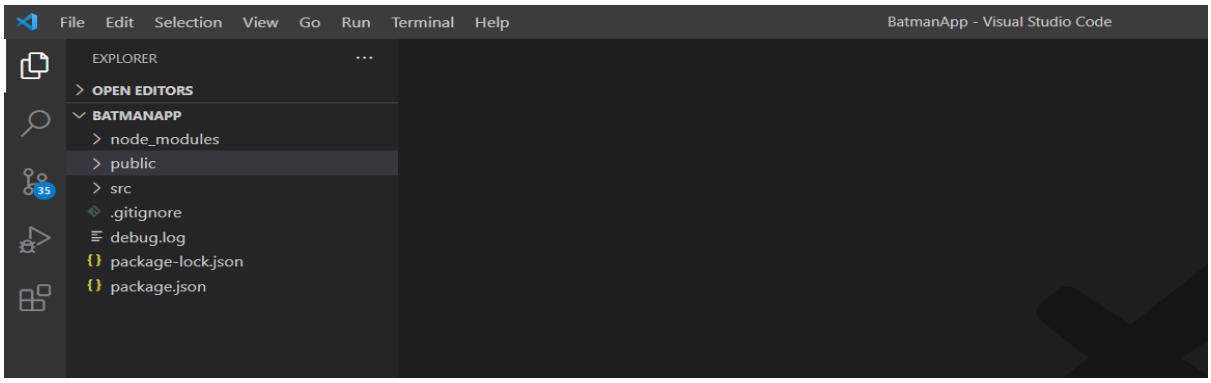


Figure:7.2 (a)

1. In the terminal give " npm audit fix " to run all the files inside the folder so that all the necessary packages which needs to be installed will be completed.

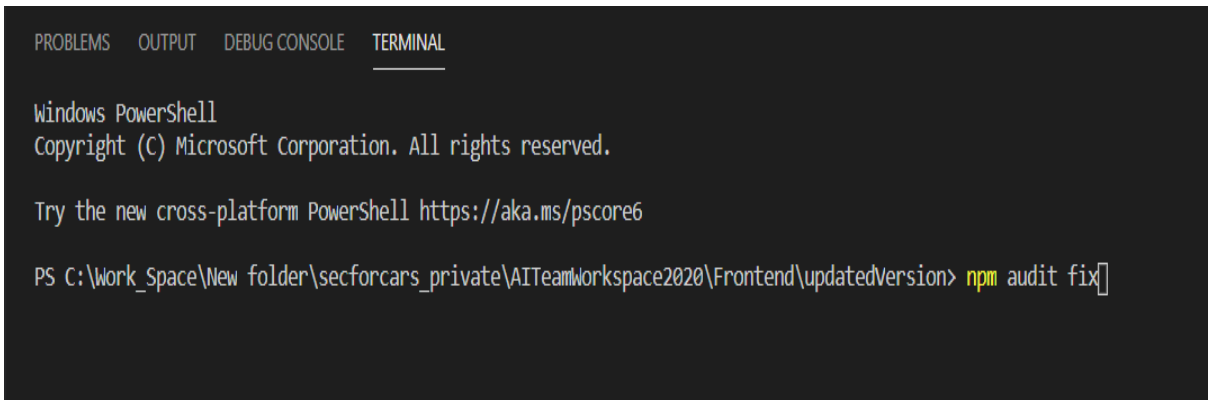


Figure: 7.2 (b)

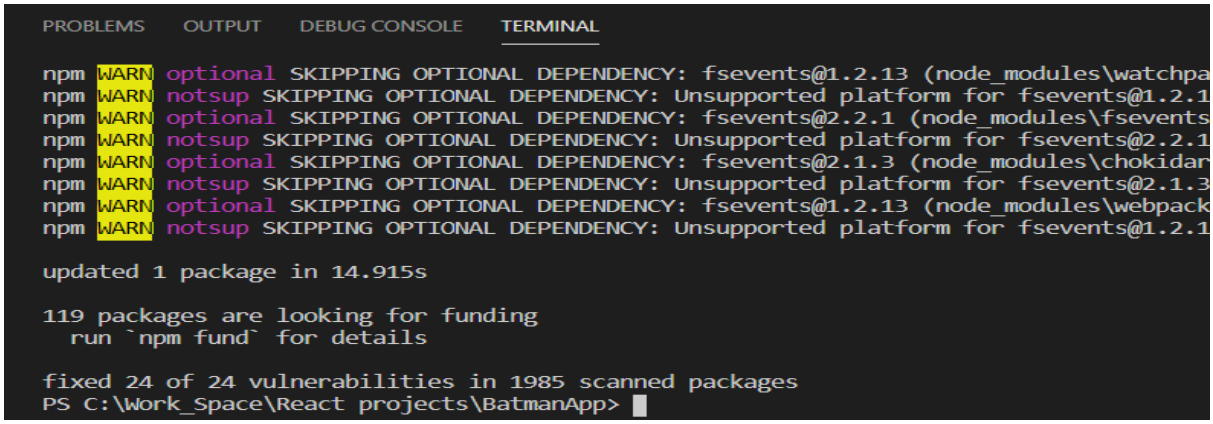


Figure: 7.2 (c)

2. After this step give “ npm start ” by doing this node modules will be created and hence after this the project will be hosted in your browser.

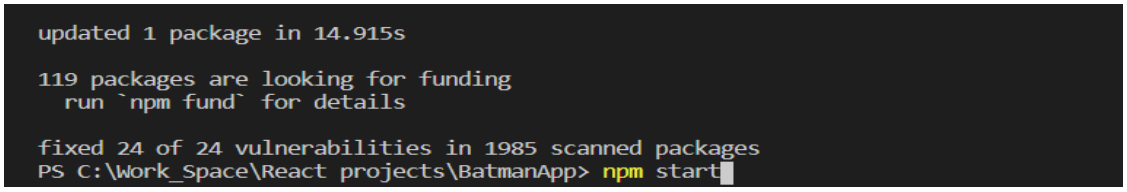


Figure: 7.2 (d)

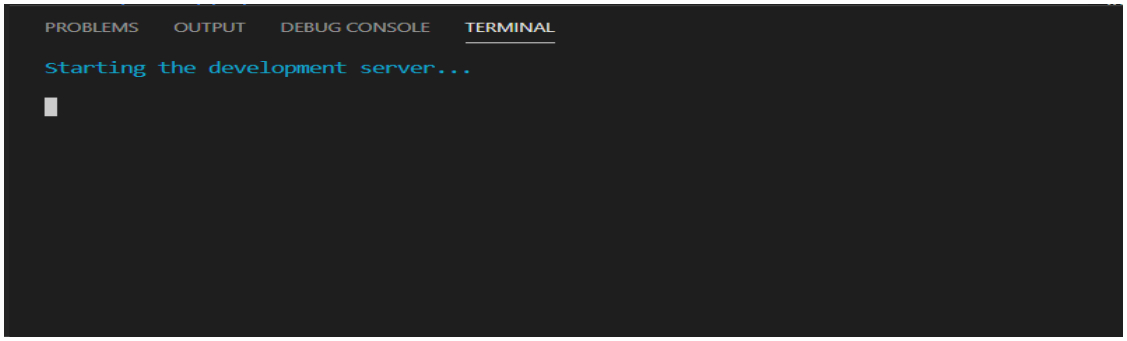


Figure: 7.2 (e)

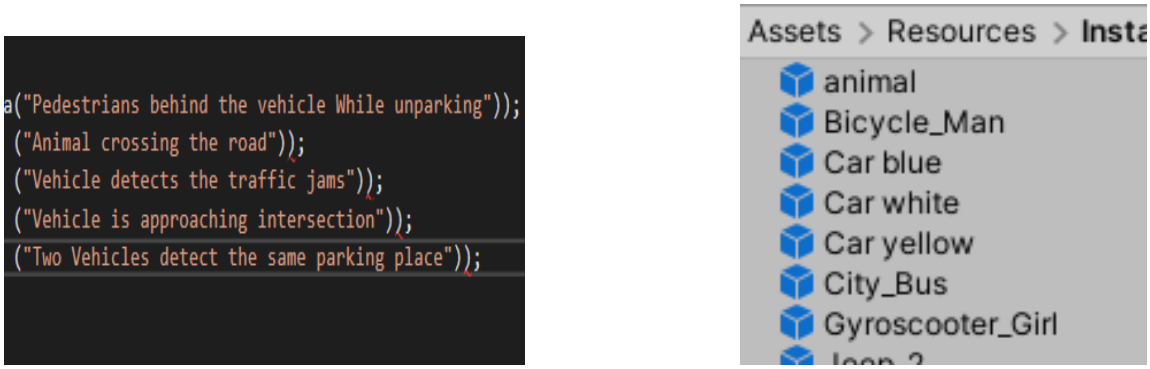
8 UNITY CONFIGURATOR DESIGN AND IMPLEMENTATION

In this section we will discuss about the initial design phase and how all the front end and backend features of Unity Configurator Application are implemented.

8.1 Unity Configurator Design Concept:

Unity Configurator Application is developed on Unity 3d game engine, and we used C# for backend programming, and Windridge City asset for setting up the environment in the Unity Configurator Application, UTS_PRO2020 asset for 3d models such as vehicles, persons etc., and Enviro - Sky and Weather Asset for weather and time simulation.

As an initial step we made a research [17] to find out the commonly occurred reference scenarios. We have selected eight of these reference scenarios and framed them into a reference scene library and identified required models and selected these models from UTS_PRO2020 asset and stored into model library.



Reference Scene Library **Model Library**

Figure 8.1(a): Screenshots of Reference scenes Library and Model Library

In the Windridge City environment, we identified 4 locations namely Intersection, Highway, Countryside, Parking area where we configure the eight reference scenes by loading the model and generate new reference scenario.

We also identified few models positions in the environment where we load the models into the scenes based on the information that we are getting from AI.

There are also different sensor defects identified and tried to make it as real as possible to the original defects on sensors such as Lens flare, Moisture droplets, and 3 types of dirt effects.

Enviro - Sky and Weather Asset provided various weather simulation options and time simulation option.

Now comes backend design, we extract the threat type value (Variable name in json file: ThreatClassification) from the Json file and compare it with our predefined and fixed set of 10 threat types and select the threat types from predefined threat types which are having 50% match with the json's threat types. Based on the selected threat types, reference scenes are loaded. Note: We have hard coded these 8 reference scenes to 10 predefined threat types.

```
"ThreatClassification": "DOS" , "Word": "Dinel of service"
"ThreatClassification": "SPF" , "Word": "Spoofing "
"ThreatClassification": "SBF" , "Word": "-Brute Force"
"ThreatClassification": " NC" , "Word": "Natural Cause -"
"ThreatClassification": "TMP" , "Word": "Tampering"
```

Figure 8.1(b): Predefined Threat types

8.1.1 Identified Locations with Camera Coordinates:

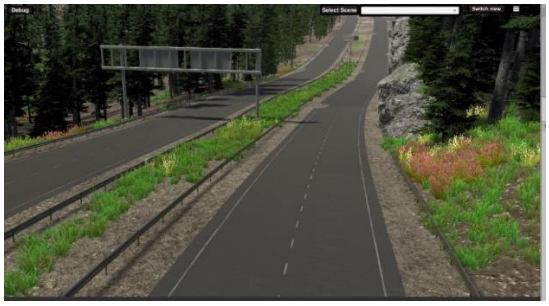
Below are the Idetified Locations with Camera Location Coordinates in Windridge City Environment. 8 reference scenes are configured in these 4 locations based on threat type.

For example, reference scenes "Vehicle detects the traffic jams" and "Vehicle is approaching intersection" are loaded at Intesection location when threat type is "SBF" and "SPF" respectively.

Intersection

Position (236, 354, -220)

Rotation (30, 255, 0)



Highway

Position (494, 302, 167)

Rotation (38, 267, 0)

Countryside

Position (263, 312, 384)

Rotation (29, 262, 0)

Parking Area

Position (305, 350, -378)

Rotation (30, 15, 0)

Figure 8.1.1(a): Screenshots of the four Idetified Locations and their Location Coordinates in the Windridge City Environment

8.1.2 Identified Models Positions with Coordinates:

Below figure is the screenshot of identified models position and orientation for Intersection location, these models are loaded into the reference scenes based on their position and orientation.

Models Library	Position			Orientation		
Intersection	X	Y	Z	X	Y	Z
Car blue	210	342	-221	0	-103	0
Gyroscooter_Girl	196	342	-227	0	-190	0
Taxi(yellow)	215	342	-223	0	-104	0
Car white	168	342	-231	0	-102	0
Car yellow	178	342	-243	0	75	0
SportBike	224	342	-220	0	-99	0
Car white	218	342	-219	0	-103	0
Car yellow	226	342	-228	0	79	0
car yellow	189	342	-225	0	-18	0
Car white	174	342	-240	0	83	0
Police vehicle	213	342	-231	-2	78	0
Bicycle_Man	195	342	-223	0	-190	0
City_bus	192	342	-238	0	-16	0
Bicycle_Man	195	342	-218	0	-193	0
person	198	342	-220	0	-200	0
person	195	342	-224	0	-193	0
Car blue						

Figure 8.1.2: Idetified Models Position and Orientation for Intersection

In the upcoming sections we see how the particular location is selected based on json file data and how refernce scene is selected and how models are loaded into the scene to generate new reference scenario and also the various features implemeted.

8.2 Unity Configurator Frontend Features Implementation

In this section we discuss about all the Front-end features implemented in the application. User can interact with the application using these Front-end features.

8.2.1 Starting Page

A starting point of the Unity Configurator Application, which includes various options such as Load File button, Exit button, Dropdown option, and File Explorer to search and load json file.

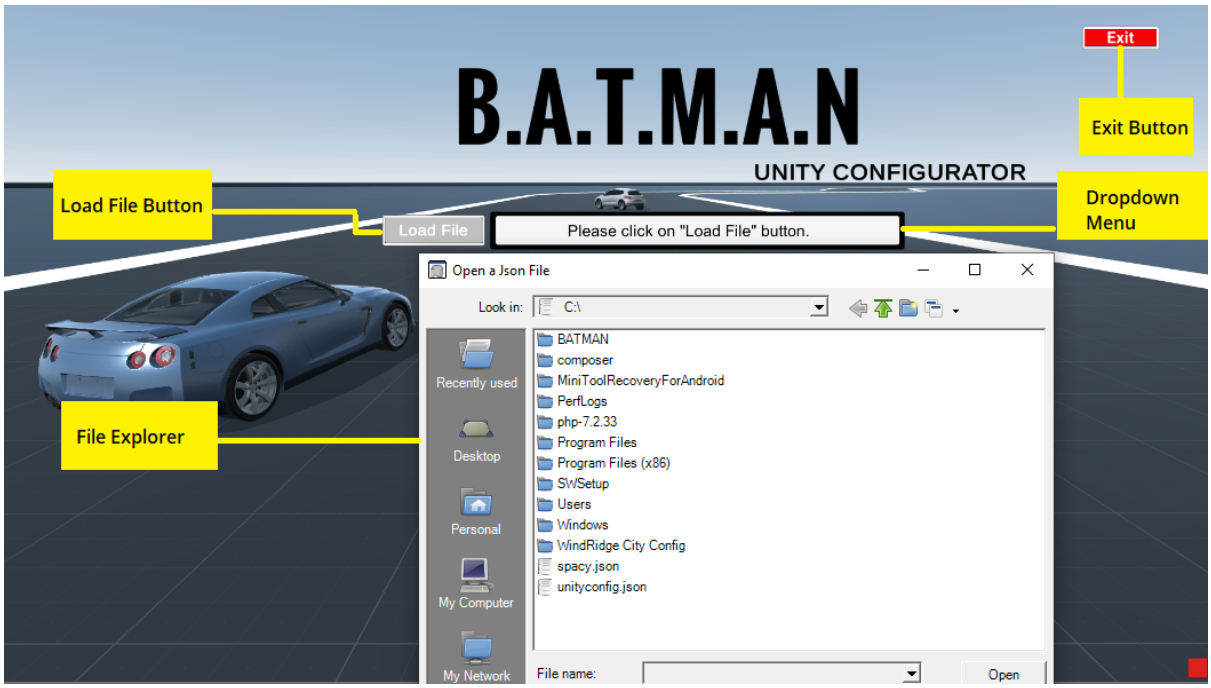


Figure 8.2.1: Starting Page

Load File Button: This button loads a file explorer pointing to the local machine.

File Explorer: Once the Load File button is clicked then file explorer will pop up it will be used to locate and load the json file which we received from AI backend and stored it in our local machine.

Exit Button: This button terminates the process and exits the application.

Dropdown Menu: Once the file is selected and opened using the file explorer then the file is processed at backend and list of reference scenes are loaded in the dropdown menu based on the processed file data.

8.2.2 Camera Operations and Mouse Lock

Camera operations allows to move and look the scene from various angles and from various positions, there are 4 camera operations are implemented in this application namely Fixed View, Switch View, Superman View, UsainBolt View.

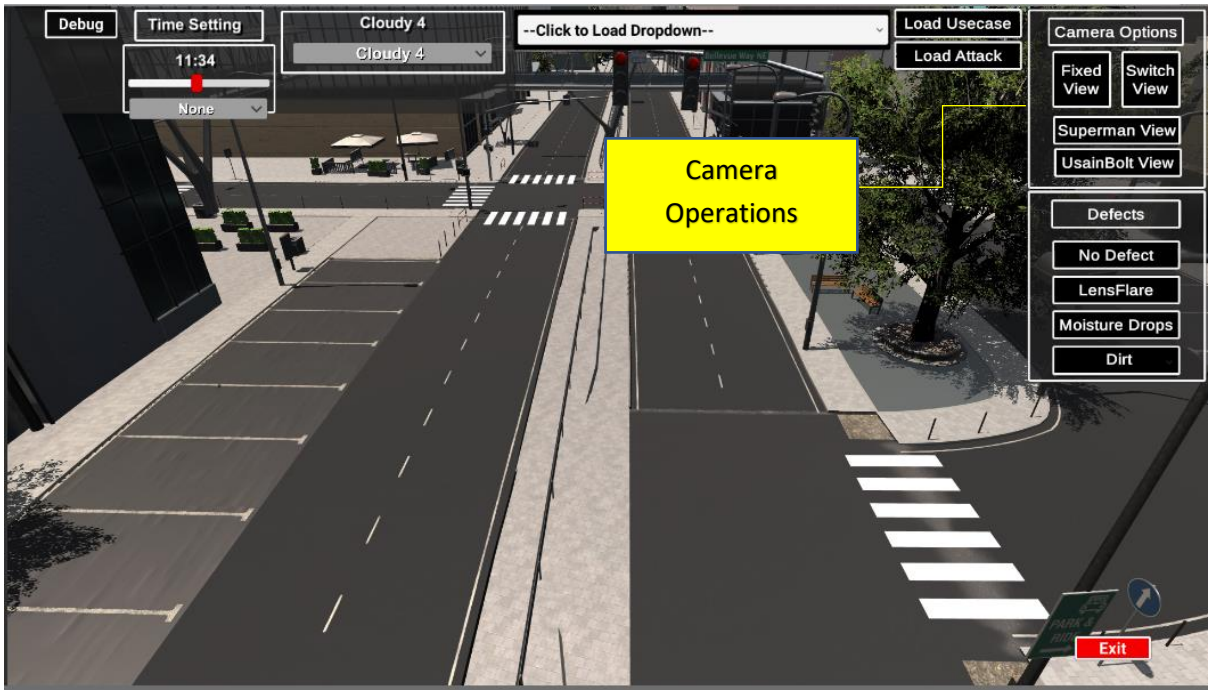


Figure 8.2.2: Various Camera Operations

Fixed view: This button moves the camera coordinates to fixed position so that an overall view of the scene is visible clearly.

Switch View: This button moves the camera coordinates to top of the scene where the scene can be seen from a particular angle from the top.

Superman View: This button allows to fly around the scene and clicking “shift+ forward” on keyboard makes it fly faster.

UsainBolt View: This button allows drop the camera coordinates to ground and helps to walk around and scene and “shift+forward” on keyboard makes it run.

Mouse Lock: Click “L” on keyboard locks the mouse and another click on “L” unlock the mouse.

8.2.3 Weather, Time Simulation and Various Sensor Defects

In this application, there are various weather types and time modes to select, and there are also three sensor defects. These features help to visualize the selected scene as close as to the real-world scene.

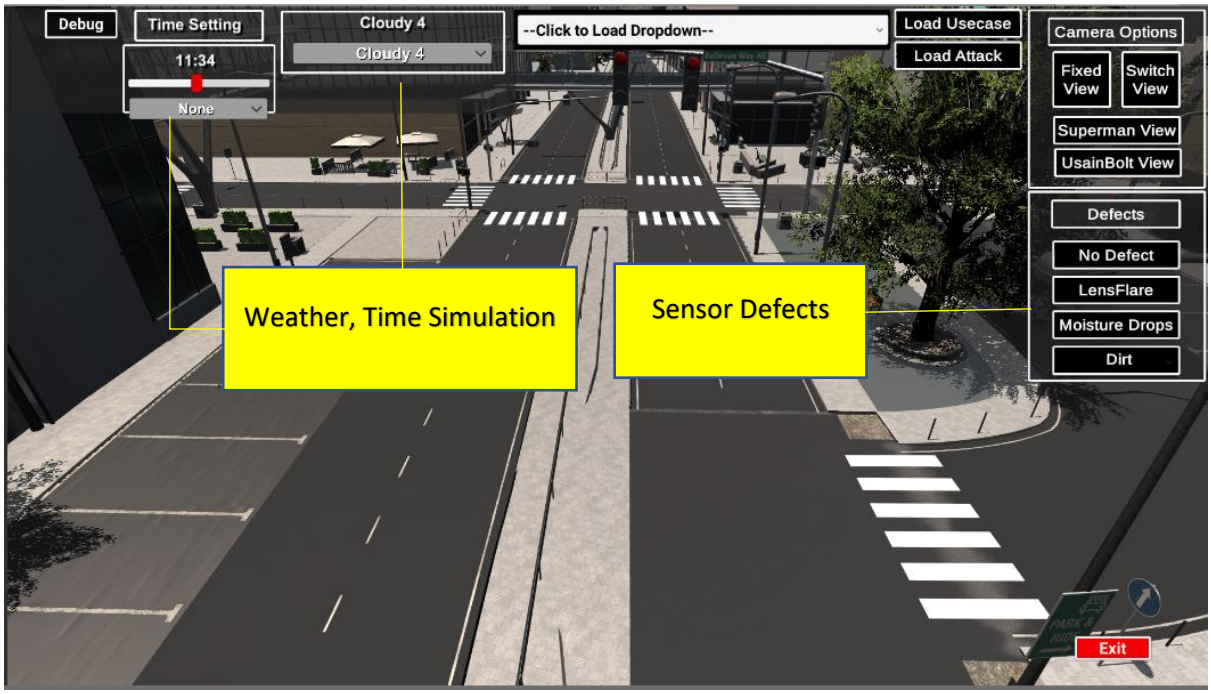


Figure 8.2.3: Weather, Time Simulation and Sensor Defects

Weather Simulation: Weather simulation allows to add various weather types into the scene such as Foggy, Light Rain, Heavy Rain, Light Snow, Heavy Snow, Clear Sky, Cloudy1, Cloudy2 etc.

Time Modes: We can switch between the day and night time, we can select a particular time of the day, and we can set the time of the system in the application using this feature.

Sensor Defects:

Sensor defects reflects the various kinds of camera sensor defects of the vehicles which hinders the vehicles capturing ability, various defects are implemented in this application such as

a) Lens Flare: It creates an effect of light scattering on the lens of the vehicle camera due to bright sun light or other light source.

b) Moisture Droplets: Creates a droplet effect on the sensors.

c) Dirt Effect: This feature adds a dirt effect on the sensor, in this application 3 levels of dirt effects are added such as light dirt, Average dirt, and Heavy dirt.

8.2.4 Load Models, Debug and Other Features

Load models feature has two types of load options one is Load Usecase and another one is Load Attack, and Other features such as Debug, Exit Button, Hacker are also discussed below.

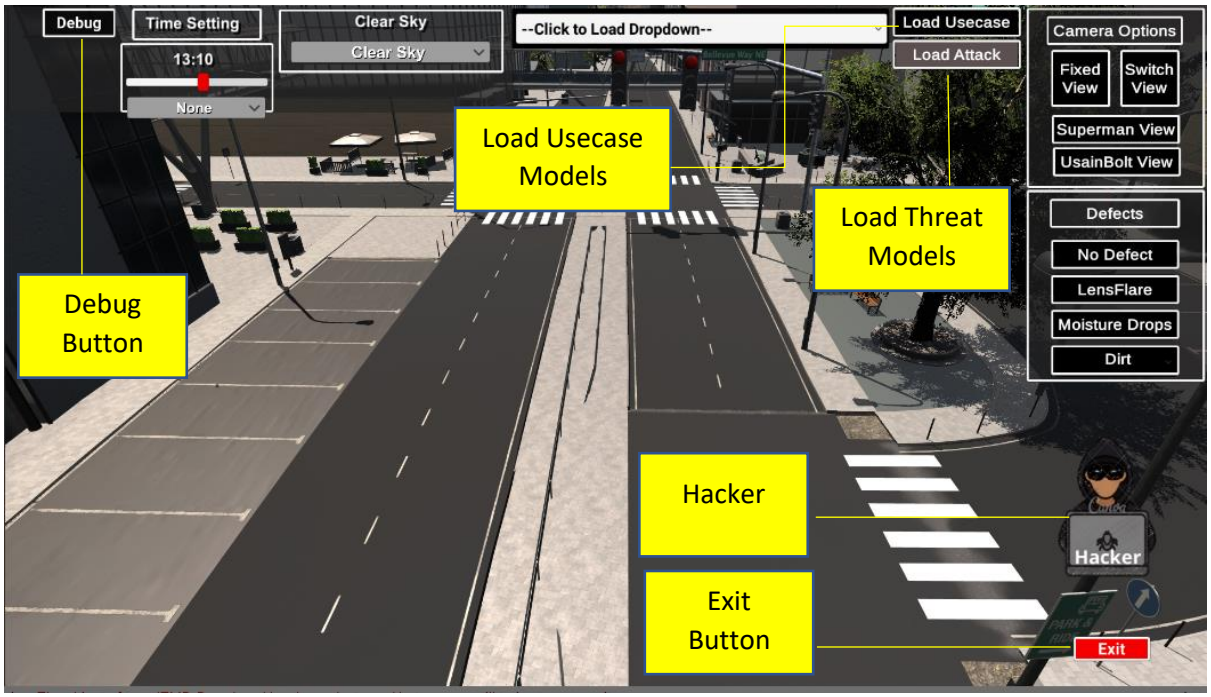


Figure 8.2.4: Load Models, Debug and Other Features

Load Models:

a) Load Use case: Once the scene is selected from dropdown menu then using Load Usecase we can load the models into the scene. Figure 6.2.4(a)

b) Load Attack: Using Load threat we can move the few models in a way that they will create a new threat or security related scenario. Figure 6.2.4(b)

Debug Button: Debug button pop up a text field which contains all the information related to the loaded scenario.

Exit Button: This exit button helps to get out of the current scene and loads the starting page.

Hacker: Whenever Load attack is clicked this hacker image gets activated to indicate that the hacker has manipulated the sensors of the vehicles.

Figure 8.2.4(a): Screenshot shows on "Load Usecase" button click, models loaded

Figure 8.2.4(b): Screenshot shows on "Load Attack" button click; models changed position

8.3 Unity Configurator Backend

This section contains the logic and functionality behind the application. We briefly discuss each source code file and its implementation. Below are the list of source code files and their explanations.

a) ParseHandler.cs
b) LevenshteinAL.cs
c) Scene_Dropdown.cs
d) SceneSelector.cs
e) LoadModels.cs
f) SceneInfo.cs
g) CameraOPS.cs

a) ParseHandler.cs

This script contains the logic to open file explorer and load the json file and parameterize them into local variables. The threat type (ThreatClassification) from json is loaded into a list (jsonData) and it is matched with our predefined threat type list (threatLib). Next, we find the percentage matching between these two lists using levenshtein distance logic. Any of the threat from predefined threat list has more than 50% percentage match with threats from json file (jsonData) then we select those threats of predefined list and stores into new list (threat_select_new).

b) LevenshteinAL.cs

This script finds the similarity between two words using LevenshteinDistance. In this the length of both the words are computed and the similarity of each character of the word is compared.

d) SceneSelector.cs

Based on the selected threats from predefined list, the reference scenes names are loaded into the dropdown menu of the start page and a scene loaded when it is selected through dropdown menu. Note: We have hard coded these 8 reference scenes to 10 predefined threat types.

```
if (threat == "DOS")
{
    m_Dropdown.options.Add(new Dropdown.OptionData("Vehicle on highway met with an accident"));
    drop_opt.Add("Vehicle on highway met with an accident");
}
if (threat == "HJK")
{
    m_Dropdown.options.Add(new Dropdown.OptionData("Pedestrians behind the vehicle While unparking"));
    drop_opt.Add("Pedestrians behind the vehicle While unparking");
}
if (threat == "CYB")
{
    m_Dropdown.options.Add(new Dropdown.OptionData("Animal crossing the road"));
    drop_opt.Add("Animal crossing the road");
}
```

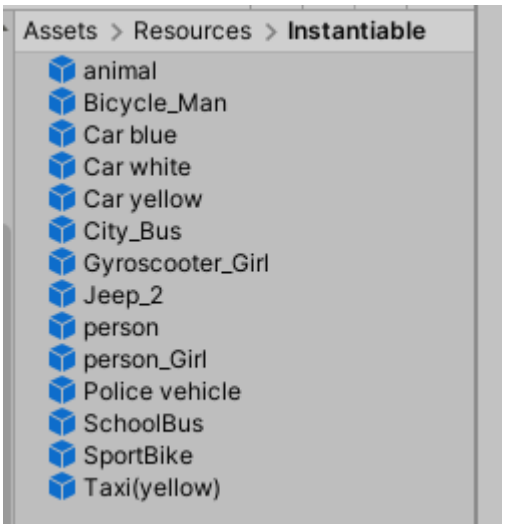
Figure 8.3: Screenshot shows how reference scenes are assigned to threat types.

c) Scene_Dropdown.cs

The same scenes which are loaded into the dropdown menu of start page is loaded into the dropdown menu of other scenes and toggle between the various scenes can be done by this dropdown menu.

e) LoadModels.cs

once the Load Usecase button is clicked then based on the reference scene type all the models are loaded into the scene and after that clicking on Load Attack will moves the certain models i.e., models of interest, in a way that it will create a new threat reference scenario. (The models loaded are extracted from the folder path: Assets/Resources/Instantiable of the application)



7.3 Models Library and its Location

f) SceneInfo.cs

This script contains all the information about configured scene such as scene name, actors, threat, environment, location etc. This information is visualized on a text field at frontend on Debug button click.

g) CameraOPS.cs

The logic for various camera operations and defects are implemented in this script using some of the unity game objects.

9 UNITY CONFIGURATOR INSTALLATION AND USER INTERACTION PROCESS

9.1 Installation and Setup

User can install and setup this unity configurator application by downloading the complete Unity Configutaor application project, which is uploaded on GitLab [18].

9.1.1 Install Unity

First user has to install unity version 2019.313f1, Unity hub version 2.3.2 and click on add new project and select the project “UnityConfig_Complete_project” from Gitlab.

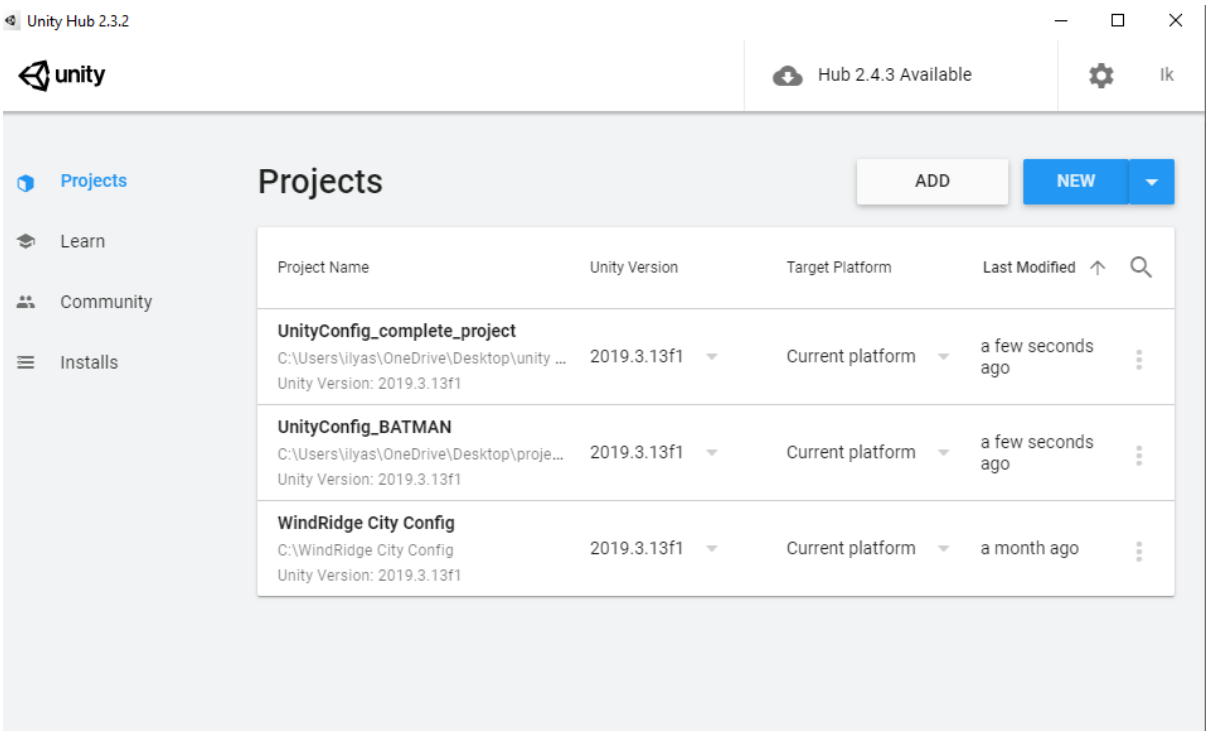


Figure 9.1.1 Screenshot shows how to install the project

9.1.2 Common Errors

After adding the project click on the project name to load unity application. Once the project is loaded it might give postprocessing errors so downgrade the postprocessing version to 2.3.0 and for Text mesh pro errors reimport TMP essential resources and TMP Examples and Extras, for Error: 'TMP_Dropdown' has been destroyed, add “base.OnDisable();” at

Library\PackageCache\com.unity.textmeshpro@2.0.1\Scripts\Runtime\TMP_Dropdown.cs:460

9.2 Details of Unity Configurator Files Added in GitLab

a) Script folder: Contains all the scripts of the project.

b) Unity config application: Contains application files and executable file for the application.

c) UnityConfig_complete_project: This is the complete project folder, which can be added in Unity hub as explained in above section.

d) Scene & Model library: This is an excel file which contains all the information about reference scenes and models.

e) Unityconfig_current: This is the format of the current json file which we are using in this project.

f) Unityconfig_future_format: This is a suggested json file format for future developments.

9.3 User Manual

In this section we will see how the user can interact with the Unity Configurator Application.

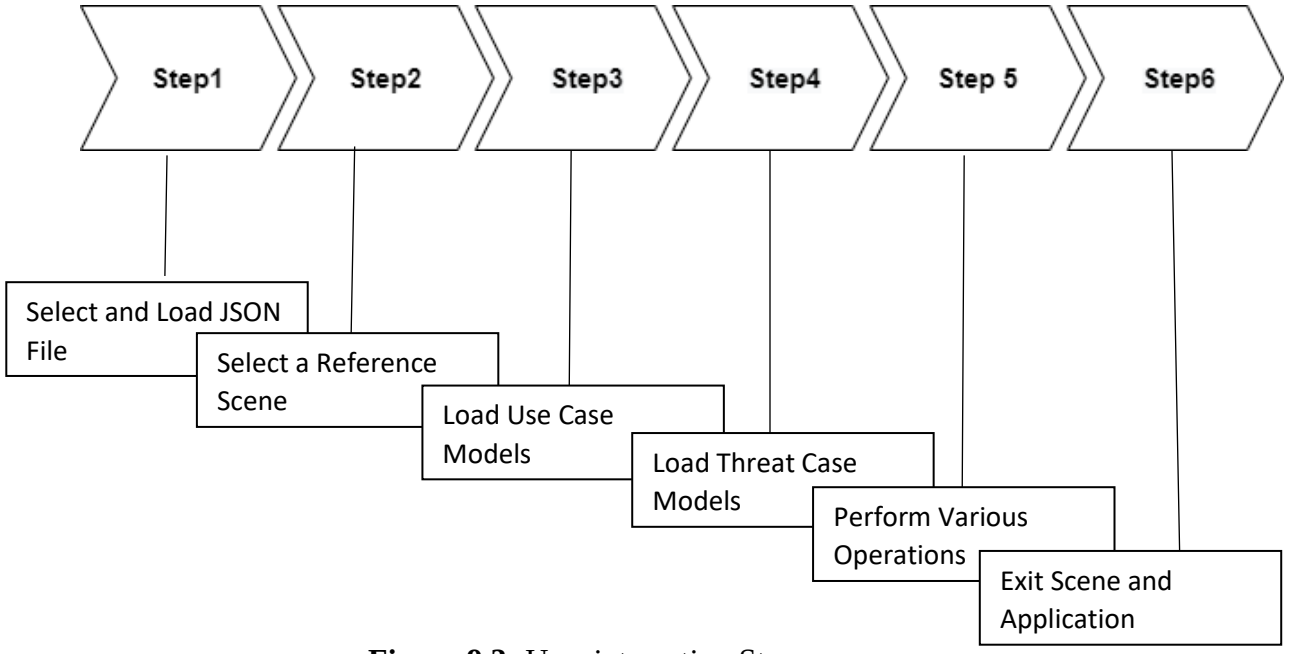


Figure 9.3: User interaction Steps

Step1: Select and Load JSON File

At the starting page, user clicks on “Load File” and selects the json file which is received from AI backend.

Step2: Select a Reference Scene

Once the file is loaded the dropdown menu is filled with reference scenes, now user can select one of the reference scenes to load.

Step3: Load a Use Case Model

Once the Reference scene is created user can load the models by clicking on “Load Usecase“ button which will load models based on the reference scene selected.

Step4: Load Threat Case Model

Now, upon clicking Load Attack button models are rearranged in a way which generates a threat reference scenario.

Step5: Perform Various Operations

Once the reference scene is generated user can perform various operations such as defects selection, weather simulation and camera operations and debug information for closely observing and extracting more information regarding the threats and finding out new possible threats.

Step6: Exit Scene and Application

User can come out of reference scene and travel back to start page by clicking on exit button in reference scene but when the user clicks the exit button of the start page then the application will be closed.

10 EVALUATION OF FEATURES

AI

Feature	Brief Explanation	Implemented By
Extraction of News information from NewsAPI in Backend Flask	Extracted relevant news information from the News API Json request from frontend received by POST call.	Kirtika Yadav
News Classification using BERT.	Implemented BERT for the news classification.	Kirtika Yadav
Classification of Extracted news List using BERT in Flask server and return JSON response from Backend Flask.	Used the trained BERT model for classifying extracted news list of News API and for NYTimes API and sending Json response to frontend containing threat classified news only.	Kirtika Yadav
Extraction of news information from NYTimes API in Backend Flask	Created separate function for the extraction of relevant information from NYTimes API Json request from frontend and classified and send the Json response to frontend.	Kirtika Yadav
Threat Extraction function in Backend Flask and structured a JSON response required for frontend	Implemented a function that perform threat extraction using Spacy for ner for the Json data sent from frontend and send the Json request containing extracted threat information to frontend.	Kirtika Yadav
Threat Type Identification Implementation	Implemented threat type Identification using BERT model. Trained the model by using created dataset.	Kirtika Yadav
Identifying threat type for single news in Backend Flask and created and send JSON response to frontend	Implemented method that identify threat type for the news sent as Json request to backend and send the Json response containing threat extraction and threat type data to frontend.	Kirtika Yadav
Classification and Threat extraction for Input Field Data in Backend Flask and structured JSON response and send to frontend	Implemented function in Backend in Flask that get Json data from frontend through POST call that contain user input and classified the news first and if threat then send JSON response with threat extraction and threat type details and is classified as non-threat then send only classified information as a JSON response to frontend.	Kirtika Yadav
Dataset Pre-processor Script in Python	Following the coding standards of project, implemented a script to prepare dataset for Ner from raw dataset by breaking sentences into words and assigning POS tags for each word. This	Kirtika Yadav

	prepared dataset used in BERT, XLNET and Spacy	
Dataset Preparation Script	Implemented to convert the csv file for Ner after tagging into txt for specific models that work with txt format of dataset for NER task written in python.	Kirtika Yadav
News classification Dataset	Added sentences related to threat and non-threat news by searching various websites and papers for the news classification dataset.	Kirtika Yadav
Threat Type Identification Dataset	Added sentences related to threat types by searching various websites and papers for the news classification dataset.	Kirtika Yadav
Threat extraction Dataset	Added sentences that contained various threat extraction techniques so that model can be trained better.	Kirtika Yadav
Actor extraction and threat extraction using BERT (Ner task)	Implemented BERT for Actor extraction for NER task and compared the performance with Xlnet and Spacy	Kirtika Yadav
Synonym Detection	Created two POC using Python on synonym detection using free apis.[19]	Kirtika Yadav

Table 2

User friendly UI design	Building the Single Page Application containing multiple components using ReactJS and CSS designing framework Bootstrap.	Kruthika Hosaballi YajnanarayanaBhat
Fetching the news from the free-source NewsAPI.	Fetching the real-time live data from the free-source NewsAPI by making a promise asynchronous call which returns a promise data using Axios GET function.	Kruthika Hosaballi YajnanarayanaBhat
Fetching the news from the free-source NYTimesAPI.	Fetching the real-time live news data from the free-source NYTimesAPI by making a promise asynchronous call which returns a promise data using Axios GET function.	Kruthika Hosaballi YajnanarayanaBhat
Placeholder for the fetched NewsAPI data.	Components for the NewsAPI is built containing the accordion element to render the data fetched from the NewsAPI in the collapsible list format.	Kruthika Hosaballi YajnanarayanaBhat
Display of the news items fetched from NYTimesAPI.	News components for the NYTimes API having the different JSON format containing the accordion element in the form of card items to render the news data fetched from the NYTimesAPI in the collapsible format.	Kruthika Hosaballi YajnanarayanaBhat

Component creation for the pagination dropdown.	Component creation and state variable declaration which holds a predefined set of values which should be displayed in the UI for pagination.	Kruthika Hosaballi YajnanarayanaBhat
Pagination function	Number of items fetched from the source can be modified by the user by selecting a number (e.g., 20) set of predefined numbers from the dropdown present in the homepage.	Kruthika Hosaballi YajnanarayanaBhat
Component creation for the API dropdown.	Component creation and state variable declaration which holds a set of predefined set of values for API selection.	Kruthika Hosaballi YajnanarayanaBhat
Header component.	Common navigation bar creation as a reusable component which will be displayed throughout the application.	Kruthika Hosaballi YajnanarayanaBhat
Footer component.	Common footer component creation as a reusable component containing copyrights and other useful information which will be displayed throughout the application.	Kruthika Hosaballi YajnanarayanaBhat
Landing page.	Landing page creation as separate component containing the background image, and an anchor tag which can be clicked to navigates the user to the homepage.	Kruthika Hosaballi YajnanarayanaBhat
Homepage component.	Homepage component which acts as a parent component for a couple more components which adds up together to form a homepage.	Kruthika Hosaballi YajnanarayanaBhat
User feeds component.	User feeds component having a text field wherein the user inputs the content and submits which will be added at the bottom as a list of items.	Kruthika Hosaballi YajnanarayanaBhat
Classification output page component.	Classification output page having a accordion element to display the list of threat classified items received from the backend.	Kruthika Hosaballi YajnanarayanaBhat
Request from the frontend to fetch the threat classified output.	A HTTP post request is made from the frontend to the backend to fetch the classified the threat classified items from the backend.	Kruthika Hosaballi YajnanarayanaBhat
NER spacy output placeholder.	Component to hold the data received from the spacy model output containing the threat classified tags (words), and threat type.	Kruthika Hosaballi YajnanarayanaBhat
CSS file.	A Cascading Style Sheet (CSS) creation containing the common styles for the complete application.	Kruthika Hosaballi YajnanarayanaBhat

Back button functionality.	A back button designing and functionality to navigate back to the homepage from the classification output page or threat based extracted output page.	Kruthika Hosaballi YajnanarayanaBhat
Loading functionality.	A loader design and display while fetching the classified output and threat-based extraction output from the backend and implementation.	Kruthika Hosaballi YajnanarayanaBhat

Table 3

Implemented NER in XLNET	Used a pretrained model XLNet model for Named Entity Recognition task. Trained the model using custom data	Yashwanth Tadakamalla
Dataset preparation	In dataset preparation we need to allot Parts of speech (POS) for each word of the sentence. Wrote a script where we provide a sentence, and the result will be displayed with breaking the sentences to words and with the respective POS	Yashwanth Tadakamalla
Json structure	We need the Json format in the backend to send the results of spacy to the frontend. Wrote two scripts for this task and finalized one of them	Yashwanth Tadakamalla
Dataset Tagging	In dataset preparation we also need to add the respective tags to the words in each sentence. I manually did the tagging for whole dataset by assigning respective tags	Yashwanth Tadakamalla
Test Sentences	While comparing all the three models we need to use the same test data. So, I digged out the sentences which are being used by the XLNET model for evaluation and provided the sentences to other models for testing	Yashwanth Tadakamalla
Dataset for threat type and classification	While preparing the dataset we needed to search for content. Added many sentences to our dataset for threat type and for classification	Yashwanth Tadakamalla
Dataset for final Threat type	Here I added many different threat related sentences to the dataset	Yashwanth Tadakamalla
Demo web application	Implemented the demo web application using flask creating a demo frontend and backend with a connection	Yashwanth Tadakamalla
Visualization of metrics	While comparing the models we needed to visualize the metrics like accuracy, f1score and loss	Yashwanth Tadakamalla

Table 4

Feature	Brief Explanation	Implemented by
spaCy Implementation	Implemented SpaCy for Named Entity Recognition (NER)	Chandra Prakash
spaCy Evaluation Process	Improved SpaCy Model and Its Evaluation Process	Chandra Prakash
Remove JSON Dependency for spaCy	Implemented idea of Removing Complex JSON Format Dependencies for Spacy NER Model	Chandra Prakash
Threat Type and Extraction Dataset	Dataset Preparation & Training for Threat Type and Treat Extractions	Chandra Prakash
Comparison of NER Models	Compared spaCy with all the NER Models	Chandra Prakash
Classified Data Conversion Application	This App takes Classified or Raw sentences as Input to Intermediated Dataset	Chandra Prakash
Dataset Construction Application	This App takes Intermediate Datasetas Input, which is preprocessed for Extraction Type and Tag Assignments. After that it produces JSON free and efficient Dataset for NER	Chandra Prakash
spaCy Backend	Backend Service was developed for Spacy	Chandra Prakash
Entities Extraction for Backend	Extraction of Entities for Backend Service for spaCy and generate a response JSON	Chandra Prakash
Frontend & Custom Input Widget	Frontend Service and Custom Input Widget for Manual NER Process	Chandra Prakash
Highlight Feature for Frontend	Highlight Component to represent various types of NER Entities from Backend on UI	Chandra Prakash
Connection Services	Implemented Connection services for both Frontend & Backend	Chandra Prakash
JavaFX, Dataset Pre-Processing & Construction Apps	Development of Dataset utilities with JavaFX Technology to provide Backend and Frontend Both, Functional Programming, Chained Exceptions, Strong Type Checking for Dataset Creation	Chandra Prakash
Index Feature, Instruction Popups and Dialogs	Index feature for Dataset Creation to enhance “start from anywhere” functionality Guided Instructions with various Instruction Popups and Dialogs for User View to represent Internal States	Chandra Prakash
UI Features for Dataset Utilities i.e. Upload, Start etc.	UI Features for Dataset Applications, Upload Functionality from any path, Start button etc.	Chandra Prakash

Cascading Style Sheets & Background Cover	CSS for Highlight Feature and CSS for Classified Data Conversion & Dataset Construction Application's features, Background Cover Manual Creation	Chandra Prakash
Formatted JSON for Tags	Formated JSON to Add Tag in Response Data for distinctly Identification of various Entities	Chandra Prakash
Integrated Back End & Front End	Integrated Back End & Front End Components, Connection Services. Resolved bugs of Merging Issues after Integrating. Added Cross Platform functioning Independency	Chandra Prakash
Build & Deployment	Build and Deployment for Dataset Applications	Chandra Prakash
GIT	GIT Process & Codebase Management	Chandra Prakash

Table 5

UNITY

Feature	**Brief Explanation**	**Implemented by**
Scene Selector Dropdown	Allows user to select a scene from a list of dropdown scenes.	Ilyaz khan Mohammed
Camera Fixed View	Facilitates to see the scene from a fixed camera view.	Ilyaz khan Mohammed
Camera switch View	Allows to see the scene from the top.	Ilyaz khan Mohammed
Superman View	Allows to fly around the scene.	Ilyaz khan Mohammed
UsainBolt View	Allows to walk and run around the scene.	Ilyaz khan Mohammed
Load Usecase	Loads all the related models of the scene	Ilyaz khan Mohammed
Load Threatcase	Loads all the related models of the scene based on threat type.	Ilyaz khan Mohammed
Lensflare Defect	Creates lens flare effect on camera sensors.	Ilyaz khan Mohammed
Moisture Droplets	Creates droplet effect on camera sensors.	Ilyaz khan Mohammed
Dirt Defect	Creates 3 types of dirt effects on camera.	Ilyaz khan Mohammed
Weather Simulation	Allows user to select various weather conditions.	Ilyaz khan Mohammed

Day Night Mode	Allows user to change between day and night-time.	Ilyaz khan Mohammed
Debug Panel	Shows complete scene details.	Ilyaz khan Mohammed
Scene Exit Button	This will exit the current scene and loads into starting scene.	Ilyaz khan Mohammed
Application Exit Button	This will help to exit the application.	Ilyaz khan Mohammed
Load Json File	This button helps to load json file.	Ilyaz khan Mohammed
Folder for Models	A resource folder where all the models are stored.	Ilyaz khan Mohammed
Starting Scene	Starting scene of the Application.	Ilyaz khan Mohammed
Json Parameterization	Implemented Json parameterization to work in unity at the backend.	Ilyaz khan Mohammed
Percentage Matching of Threat Type	Identifying the threat type based on percentage match between predefined threat types and threat types in Json file.	Ilyaz khan Mohammed
Screen Lock	On pressing “L” on keyboard, screen locks.	Ilyaz khan Mohammed
Hacker Logo Activator	On click threat load button, hacker logo is activated to indicate that the sensors are hacked by hacker.	Ilyaz khan Mohammed

Table 6

11 GIT REPOSITORY PROCESS AND MANAGEMENT

It was needed to select one freeware Git Client Desktop Application for our production. SourceTree is the one which provided a lot of features in one hand.
SourceTree is Mercurial and Git Client. It is available for both platforms powered by Windows and iOS. The software is free. It features an intuitive graphical interface for repositories bridging the gap between a user and a Git. As to mature developers, they will also benefit from using the sourcetree as it allows them to focus on writing code and being more productive.

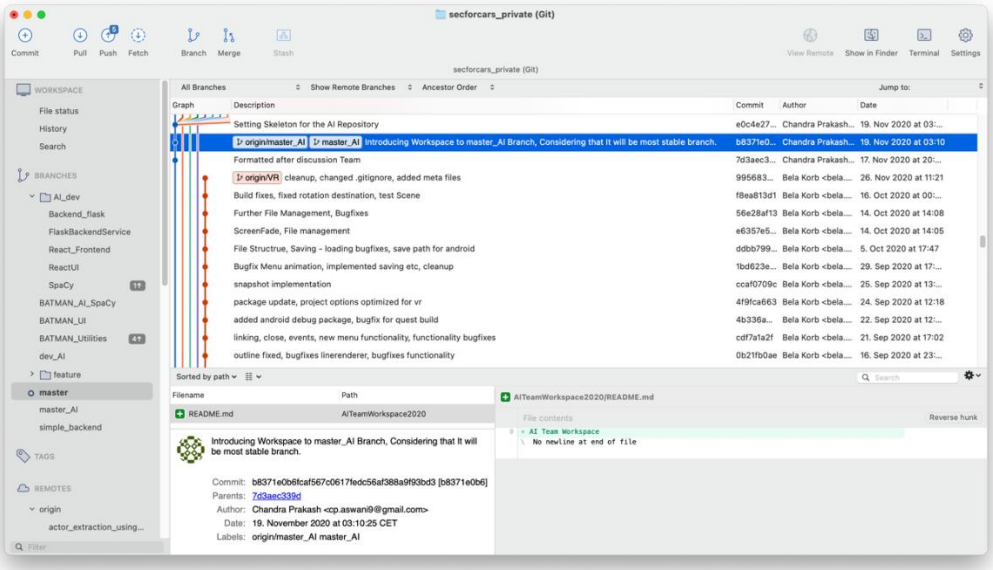


Figure 11

A pattern was chosen to deal with code commits. Two branches were created:

1. dev_AI-UC: Sub-branches were created from this branch bases on individual features. After each development, these sub-branches are merged in dev_AI-UC branch.

2. master_AI-UC: It is main branch for all the code. When all the development was over. Dev_AI-UC is merged in this branch.

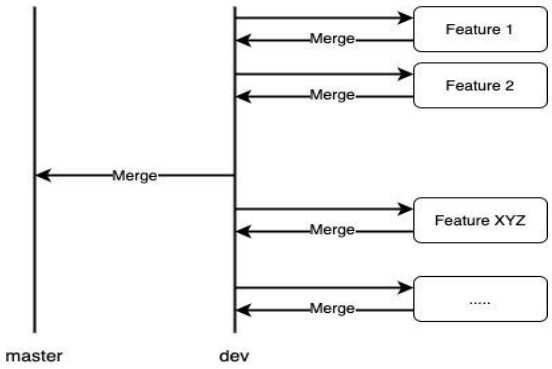


Figure 11(a)

Repository URL:

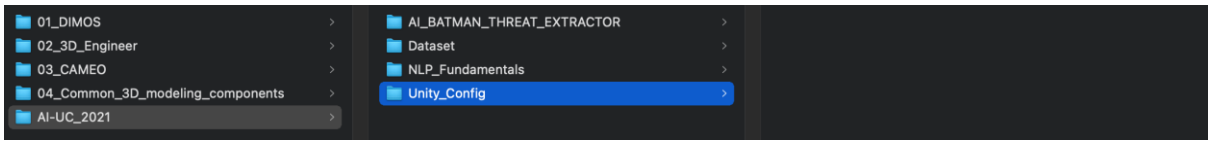
Root Link:
https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC

BATMAN AI-UC 2021 Code Drop Link:
https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC/AI-UC_2021

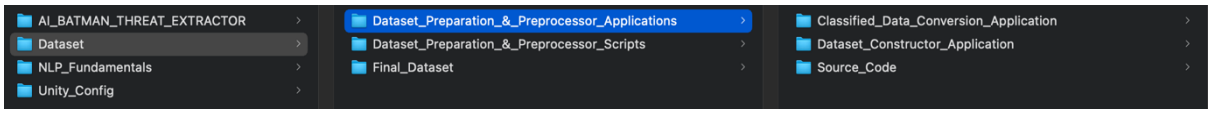
Snippet of the folder structure of BATMAN AI/UC codebase:

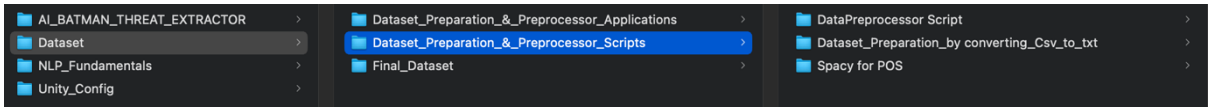
1. AI BATMAN THREAT EXTRACTOR Frotnend and Backend

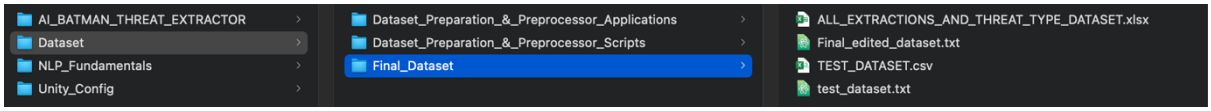
2. UNITY CONFIG



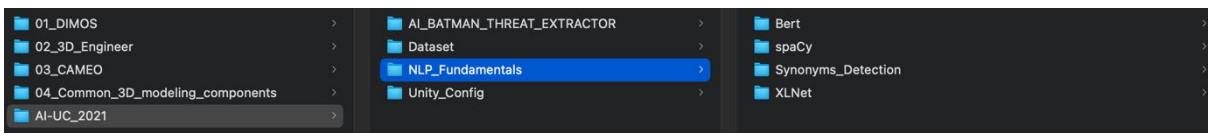
3. Dataset Preparations & Pre-processing







4. NLP Fundamentals for BATMAN AI



- **GIT AND SOURCE CODE LINKS:**

1 https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC/AI-UC_2021/Dataset/Dataset_Preparation_And_Preprocessor_Applications/Classified_Data_Conversion_Application

2 https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC/AI-UC_2021/Dataset/Dataset_Preparation_And_Preprocessor_Applications/Source_Code/classified_data_conversion_SourceCode

3 https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC/AI-UC_2021/Dataset/Dataset_Preparation_And_Preprocessor_Applications/Dataset_Constructor_Application

4 https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC/AI-UC_2021/Dataset/Dataset_Preparation_And_Preprocessor_Applications/Source_Code/dataset_constructor_SourceCode

5 https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC/AI-UC_2021/Dataset/Dataset_Preparation_And_Preprocessor_Scripts/DataPreprocessor%20Script

6. https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC/AI-UC_2021/Dataset/Dataset_Preparation_And_Preprocessor_Scripts/Dataset_Preparation_by%20converting_Csv_to_txt

7. https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC/AI-UC_2021/NLP_Fundamentals/Bert/Actor_Extraction_using_BERT

8 https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC/AI-UC_2021/NLP_Fundamentals/Bert/NEWS_CLASSIFICATION_USING_BERT/News_Classification_using_BERT_from_Huggging_face

9 https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/blob/master_AI-UC/AI-UC_2021/NLP_Fundamentals/Bert/Threat_Type_Classification/Threat_classification_Using_Bert.ipynb

10 https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC/AI-UC_2021/NLP_Fundamentals/spaCy

11 https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC/AI-UC_2021/AI_BATMAN_THREAT_EXTRACTOR/BATMAN_Backend/Flask_Server

12 https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC/AI-

UC_2021/NLP_Fundamentals/Bert/NEWS_CLASSIFICATION_USING_BERT/News_Classification_using_BERT_from_Huggging_face

13 https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC/AI-UC_2021/AI_BATMAN_THREAT_EXTRACTOR/BATMAN_Backend/Flask_Server

14 https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC/AI-UC_2021/NLP_Fundamentals/spaCy

18 https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC/AI-UC_2021/Unity_Config

19 https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC/AI-UC_2021/NLP_Fundamentals/Synonyms_Detection

20 https://gitlab.cc-asp.fraunhofer.de/mbseguy/secforcars_private/-/tree/master_AI-UC/AI-UC_2021/AI_BATMAN_THREAT_EXTRACTOR/BATMAN_Frontend

12 PROJECT MANAGEMENT

Name	Organizational Task	Responsibilities
Ilyaz Khan Mohammed	Project Manager	Planning the project and the weekly meetings. Monitoring Progress Organizing presentations and managing issues.
Kirtika Yadav	Scrum Master	Creation and maintenance of user stories. Updating Weekly meeting agenda Sprint planning and presented the plan in meetings
Kruthika Hosaballi YajnanarayanaBhat	Head of Documentation and Presentation	Creating templates for the presentation Giving ideas on how to present the concepts in non-textual way. Created Documentation template Edition of Document and Presentation slides.
Chandra Prakash	Head of Software Development	Determining operational practicality Maintaining Git Repository Design and implementation of the Integration
Yashwanth Tadakamalla	Head of Software Testing	Tested the app-lication thoroughly by providing related and unrelated data. Creation of test suites and test cases

Table 7: Organisational Roles

13 SUMMARY

Batman Threat extractor is a web application which works as an assistant developed to help system engineer in the design phase of building the autonomous systems. Batman threat extractor helps the system engineer by providing the knowledge of the threats that might occur while building a autonomous system. System engineer can provide any custom unstructured data and can know which it is threat or non-threat data. If it is a threat data, then the engineer will be able to generate the threat extraction report. By this report the system engineer can get to know how a threat is implemented, tools that are used to compromise our system and what the consequences that occurred by that threat. System engineer will also get the information related to the threat type.

As part of future work. Our underlying models which we are using in the backend can be trained with increasing the dataset to get better results and to increase the accuracy. We can also extend our model to predict if there is more than one threat type in the given unstructured data. We can add more customized tags to the data, or we can decrease the tags and make them more generalize so that model needs to predict less tags by that the performance of the model also increases. But when we want to increase the tags, we need to increase the data so that model can predict accurately. When it comes to web application, we can implement a feature the user is able to filter out the tags so that we want to see particularly at that moment. We can also integrate few more NEWS APIs to get different news. We can also update our web application where user can learn about few threat types and how they can be implemented this feature will be useful for every individual. In the future we might have to handle huge amount of data, so it is innovative idea to use a database to store the data which improves the performance of our application.

The future work related to Unity Configurator Application includes automatic detection of reference scenes based on threat data from AI backend. As of now, we are using a predefined list of threat types and configuring the predefined set of scenarios using threat type. To add more intelligence to the application, the automatic detection of reference scenes should happen, and if we can extract more information through json file then it will become more easier to detect the reference scene and model locations. Both current json file format and future json file format are uploaded on Gitlab.

14 BIBLOGRAPHY

[1] https://medium.com/mysuperai/what-is-named-entity-recognition-ner-and-how-can-i-use-it-2b68cf6f545d
[2] https://towardsdatascience.com/xlnet-explained-in-simple-terms-255b9fb2c97c
[3] https://mccormickml.com/2019/09/19/XLNet-fine-tuning/
[4] https://newsapi.org/
[5] https://developer.nytimes.com/docs/top-stories-product/1/overview
[6] https://www.depends-on-the-definition.com/named-entity-recognition-with-BERT/
[7] https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/
[8] https://arxiv.org/pdf/1810.04805v2.pdf
[9] https://towardsml.com/2019/09/17/BERT-explained-a-complete-guide-with-theory-and-tutorial/
[10] http://jalammar.github.io/illustrated-transformer/
[11] http://jalammar.github.io/illustrated-BERT/
[12] https://analyticsindiamag.com/step-by-step-guide-to-implement-multi-class-classification-with-BERT-tensorflow/
[13] https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-BERT-for-text-classification/
[14] https://medium.com/swlh/a-simple-guide-on-using-BERT-for-text-classification-bbf041ac8d04
[15] https://www.geeksforgeeks.org/newspaper-article-scraping-curation-python/
[16] https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/
[17] https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/documents/13882-automateddrivingsystems_092618_v1a_tag.pdf
[18] https://spacy.io/usage/spacy-101