Crisis Event LLM

CS 4624
Virginia Tech
Blacksburg, VA
11/30/2023

Students:

- Bhargava Elavarthi
- Kunal Nakka
- Srikaran Bachu

Client:

- Dr. Mohamad Farag

Table of Contents:

Table of Figures:

Abstract:

Navigating through the intricate landscape of understanding and classifying crisis events, the "Crisis Events Language Model" project embarked on a comprehensive exploration leveraging Natural Language Processing (NLP) and machine learning. With a primary focus on utilizing BERT, a powerful PreTrained Language Model, our objective was to create an adept classification system for textual data related to crisis events sourced from the web.

Our methodology involved the adept use of the BeautifulSoup library in Python for web scraping, enabling the extraction of textual data from URLs associated with crisis events. This rich dataset served as the backbone for training and evaluating our models. Post-data acquisition, we fine-tuned BERT to align with our specific use case, adapting its output layer to meet our unique classification goals. This strategic modification enhanced BERT's capabilities in recognizing, interpreting, and categorizing crisis event data with precision.

Simultaneously, on the front-end development front, we constructed an intuitive interface using HTML and CSS. This user-friendly interface not only facilitates the visualization of the model's outputs but also simplifies user interaction and data input. The result is a practical tool poised for deployment in real-time crisis management situations.

Anticipating multiple impacts, our project positions itself to simplify the comprehension and categorization of crisis events. This functionality, tailored for decision-makers and crisis management teams, promises to be a valuable asset in the face of urgent situations. Moreover, for the participating students, the project provides a dynamic learning experience, bridging theoretical knowledge with practical applications in NLP, text classification, and transfer learning.

Throughout the project's duration, team members assumed diverse roles, from web scraping and model implementation to front-end development and meticulous documentation. This collaborative effort blended skills in programming, software engineering, Python, and machine learning, ensuring a holistic approach to project development.

In conclusion, our project not only serves as a testament to the technical prowess and collaboration within our team but also makes substantive contributions to the realms of crisis management and NLP. It underscores the potential of integrating machine learning and language models in crisis management, offering valuable insights and avenues for future exploration and development in this critical area.

Introduction:

In the dynamic landscape of data and technology, the analysis of text data has emerged as a pivotal element for a myriad of tasks, notably in text classification and natural language processing (NLP). However, the effective handling of textual data often demands substantial efforts and resources, particularly in the intricate process of preparing features, labels, and datasets. This challenge becomes particularly pronounced when attempting to distill meaningful insights from an expansive sea of unstructured text.

Amidst this, transfer learning emerges as a game-changing machine learning technique. It involves the utilization of pre-trained language models initially designed for specific tasks and adapts them to excel in diverse applications. This approach proves particularly advantageous in numerous NLP applications, streamlining the development of high-performing models with reduced upfront workload.

Our project dives deep into the exploration of the capabilities of Large Language Models (LLMs), with a specific focus on BERT, in the context of analyzing crisis events. Concentrating on English language models that have undergone extensive pre-training on diverse text data, we investigate their efficacy in handling crisis-related textual data extracted from tweets and webpages. The primary objective is to construct classifiers tailored explicitly for crises, facilitating the identification of crucial information and sentiments during challenging situations.

As we navigate through this project, our ultimate goal is to develop a functional application. This application is driven by a user-friendly frontend that allows one to upload a zip file of data to be trained by a model of the user's choice (either Distilbert or BERT. Once the model is trained, the user can type in text on the front end that can be identified by the specific model trained. If the content pertains to a crisis event, the application should further classify the specific nature of the crisis event, assuming it is present in the training data.

The motivation underlying this endeavor stems from the escalating need for effective crisis event analysis. In times of crisis, the ability to swiftly and accurately comprehend information holds significant importance. By harnessing the capabilities of LLMs and transfer learning, we aspire to equip students with the knowledge and tools essential to confront the intricate challenges embedded in NLP. Additionally, we aim to contribute to the ongoing development of innovative solutions in crisis management and related fields. This paper chronicles our journey in harnessing advanced language models for crisis event analysis, detailing the methodologies employed, challenges encountered, and the potential impact of our project on the broader landscape of NLP and crisis management.

Requirements:

The system encompasses a backend, responsible for training the specific models on a specific crisis event as well as the process of identifying whether the specific text is related to that crisis event. This is linked to the HTML & CSS based front end for user engagement and interaction. The functionalities fall broadly into utilizing BERT for crisis event text summarization and classification and providing a user interface to receive zip files of data from the user and display coherent, summarized, or classified outputs about crisis events.

In the zip file module, the objective is to retrieve and process text data uploaded by the user, ensuring organized storage for subsequent model training and testing. Concerning model training and deployment, the system should ingest inputs from the scraping module, and ensure the BERT or DistiBERT model is fine-tuned effectively. It is imperative that this fine-tuning be oriented towards specificity for the English language and the nuances of varied crisis events. The functionalities to be supported by the model include the summarization of multiple documents related to a single crisis event and the classification of specifics related to the crisis events. On the front end, a user-friendly interface, developed with HTML & CSS, must facilitate file upload and present model outputs in a structured and understandable format. The front end should also ensure varied, optimized interfaces for standard and administrative users.

From a non-functional standpoint, the system should exhibit usability through an intuitive interface and provide clear, succinct outputs and instructions. Performance-wise, it must maintain a low latency level from input to output delivery, ensuring timely responses to user interactions. Consistent, reliable, and accurate outputs are crucial for maintaining trust and utility in the system. Furthermore, scalability must be considered, ensuring the system can adapt to future extensions or modifications, such as incorporating additional language models or expanding crisis event categories. The project is very scalable and two models (BERT & DistiBERT) can be used by the user. Future implementations of other language models can be added in as well.

User interfaces are expected to be coherently structured with clear pathways for inputting URLs and receiving model outputs, utilizing HTML for development. In parallel, there must be harmonious software interfacing, ensuring streamlined communication between the front end and the models in the back end, to facilitate a seamless user experience.

While this document outlines the foundational guidelines for project development, it is pivotal that scalability is taken into consideration for future implementations of new models. Periodic review and agreement upon any changes by all stakeholders ensure alignment and minimize scope creep, keeping the project on track to stay practical and useful toward future objectives and applications.

Design

To begin this project, the client provided some insight as to which direction the project should be headed and what tools were needed to accomplish the given task. After careful consideration and research into what “Learning Language Model” should be implemented, we decided upon *BERT* (Bidirectional Encoder Representations from Transformers). BERT, introduced by Google AI researchers in 2018, stands at the forefront of natural language processing (NLP) models, marking a revolutionary milestone in the field. Its distinctive approach to bidirectional language understanding has catapulted it to a position of high influence, significantly advancing the state-of-the-art across various NLP tasks. BERT's bidirectional context modeling allows it to consider the entire context of a word, addressing limitations posed by traditional unidirectional models. This architectural innovation has proven immensely beneficial in tasks such as text classification, sentiment analysis, question answering, and language understanding. The decision to leverage BERT in our project stems from its proven track record in enhancing the efficiency and accuracy of NLP applications, aligning seamlessly with the objectives of our crisis event analysis endeavor.

The ultimate goal of using BERT is to help understand how to classify and analyze text that is being uploaded by the user. An in-depth analysis of how the Crisis Learning Language model works and the flow can be seen below. There are three main phases that we will dive deep into. The first phase (yellow) is for the front-end development. As we mentioned before, it is done in HTML & CSS and has JavaScript for the dynamic pieces of each page. From the front end, the user has the opportunity to upload ZIP files that contain various text files that will be analyzed by the BERT or DistilBERT Model. The second phase (green) is the link to the backend. The ZIP file is then connected to the backend through the use of Flask. With this link, the model can unpack the zip file and use the data to train a model for a specific crisis event. Phase three (purple) is the usage of the model on the front end by the user. Once a model is trained, users can select it from the model list menu. After it is selected, the user can prompt the model with various amounts of text and the model will identify if it relates to that specific crisis event. Below is a diagram highlighting the workflow for the entire project. Each phase is color-coated for easier readability.

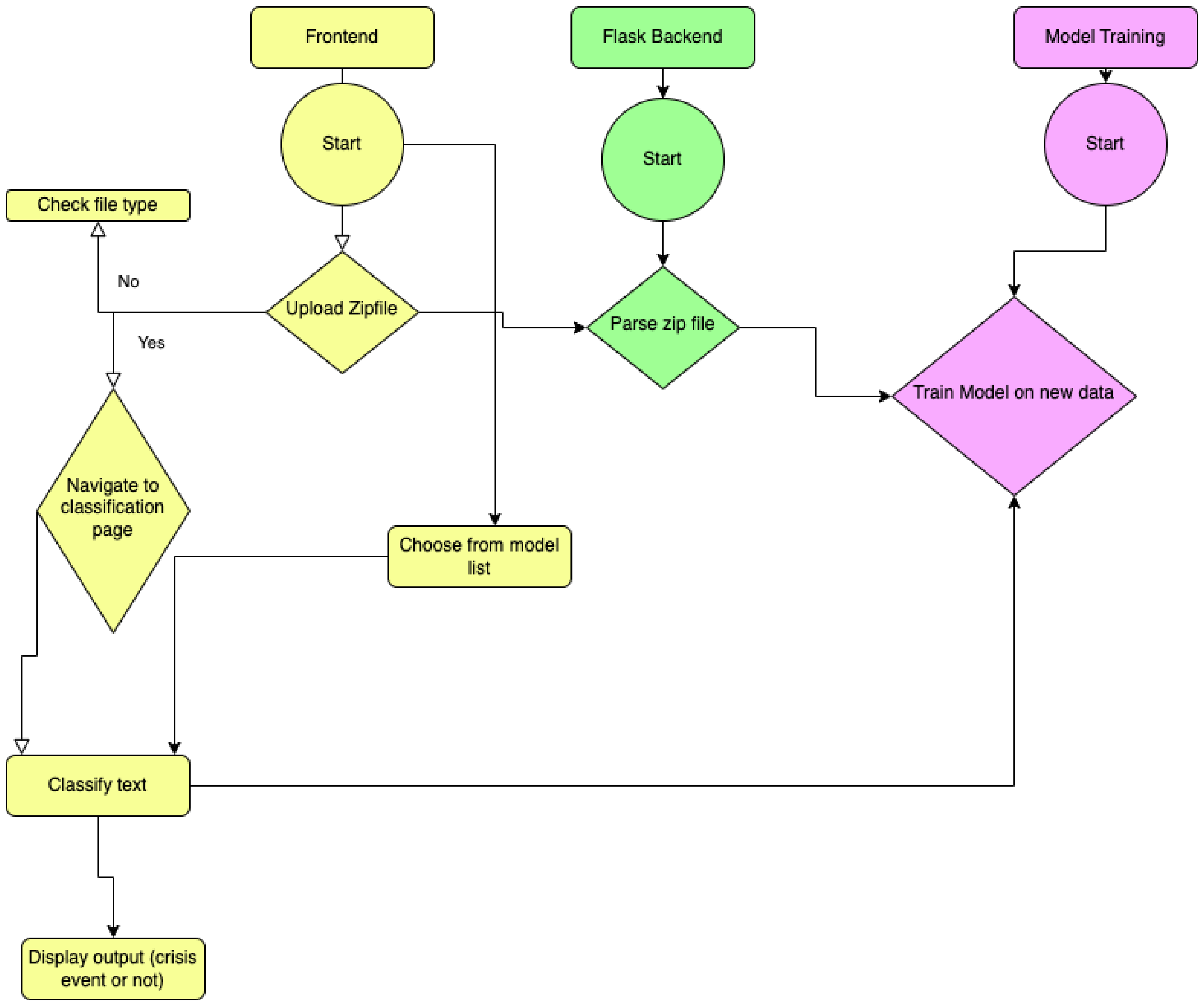


Figure 1: Workflow Flowchart

Implementation:

To begin with, our project's front end is thoughtfully designed to provide users with an intuitive and interactive experience. Comprising five distinct HTML pages, each page serves a specific purpose within the workflow of the Crisis Events Language Model (CrisisLLM). Leveraging a combination of HTML and CSS, we crafted static pages to ensure a clean and aesthetically pleasing layout, enhancing user accessibility and navigation.

For the dynamic aspects of our front end, particularly on pages where user interactions prompt real-time updates, we seamlessly integrated JavaScript functions. One such dynamic feature is evident in the user interface for text classification. When users upload data and input text for classification, the results are dynamically presented on the same HTML page. This responsive behavior not only streamlines the user experience but also provides instant feedback, contributing to the overall usability and effectiveness of the CrisisLLM platform.

Incorporating JavaScript functionalities has created a more dynamic and responsive front end. For instance, the integration of asynchronous requests enhances the speed at which data is processed, ensuring that users receive classification results promptly. Additionally, JavaScript facilitates client-side interactions, reducing the need for constant server requests and enhancing the overall efficiency of the user interface.

The backend of our Crisis Events Language Model (CrisisLLM) is powered by Flask, a lightweight and efficient web framework for Python. Flask seamlessly handles the communication between the user interface and the underlying model, providing a robust and responsive user experience. The backend includes functionalities for model fine-tuning, text classification, and model management.

In the fine-tuning process, users can select a pre-trained language model type, upload a labeled dataset in a zip file, and specify the name of the newly fine-tuned model. Flask manages the file uploads, ensuring that only allowed zip files are processed. The uploaded data is then extracted and prepared for training the specified model type, be it BERT or DistilBERT. The training process is orchestrated through PyTorch, utilizing DataLoader for efficient data handling and optimizing with AdamW. The resulting fine-tuned model is serialized using Pickle and saved for later use.

On the text classification front, users can select the desired pre-trained model, input a block of text, and receive real-time predictions regarding whether the content is related to a crisis event. Flask handles these requests, invoking the appropriate model and tokenizing the input text using the corresponding tokenizer. The prediction is then returned to the user interface for immediate feedback.

Additionally, the backend manages the display of available models, allowing users to explore and select pre-trained models for text classification. The integration of Flask with the model management system ensures a seamless and organized approach to handling different models.

Testing/Evaluation

We approached our testing strategy in two distinct but interconnected phases: frontend and backend testing. Frontend testing primarily focused on ensuring the application's usability and functionality from a user's perspective. This process involved meticulous testing of each element on every HTML page. We rigorously tested the Flask endpoints associated with each page, ensuring smooth and error-free interactions. Furthermore, we evaluated the overall cohesiveness of the Flask application, paying special attention to its integration with the backend components.

In the realm of frontend testing, our primary goal was to refine the user interface (UI) and enhance the user experience (UX). This involved a series of iterative modifications. We consistently evaluated component functionality, removing or tweaking elements that hampered usability or detracted from the visual appeal. We engaged in a dynamic process with our client, showcasing the frontend every Monday, incorporating their feedback to ensure the product aligned with their vision and requirements. Client feedback was invaluable in guiding the UI/UX design, leading to several pivotal changes.

Peer feedback, garnered from demonstration presentations, also played a crucial role in shaping the frontend. A significant recommendation from a peer was the implementation of a zip file upload feature. This replaced the less efficient manual text entry method, significantly enhancing the application's user-friendliness and efficiency. Additionally, aesthetic feedback led to a redesign of the homepage with more vibrant colors, substantially improving the visual appeal and user engagement.

Our backend testing was predominantly focused on evaluating the model accuracies. We employed a comprehensive set of evaluation methods to assess the accuracy metrics of our various models. Given the constraints of our limited dataset, our testing had inherent challenges. We used validation/testing sets comprising 20 elements each, evenly split between positive and negative samples. In some instances, due to dataset limitations, we had to reuse certain negative data points from the training set in the validation for another model. While we endeavored to minimize this overlap, it was occasionally necessary to generate sufficient data for reliable accuracy metrics.

The accuracy metrics employed were loss, total accuracy, and F1-score. These metrics were automatically computed after each training epoch. The epoch yielding the lowest loss score was selected for the final model. This process was uniformly applied across all models, ensuring a consistent and fair evaluation. Furthermore, these metrics were made visible to users during the model training process via the Flask app, adding a layer of transparency and user engagement.

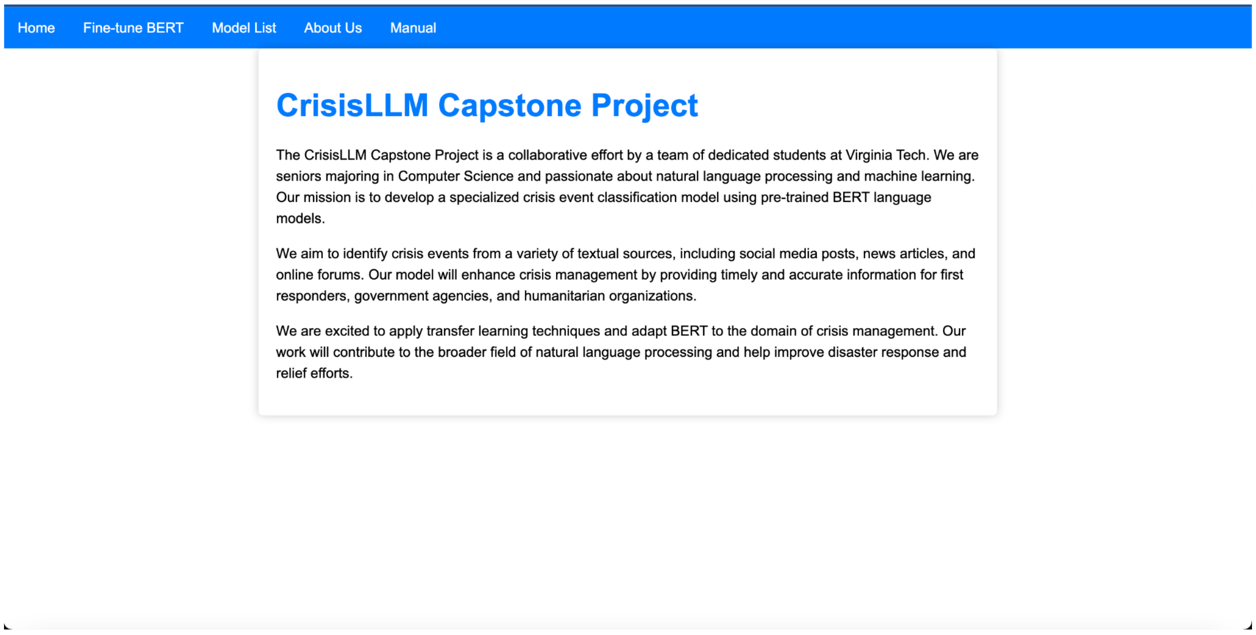
Integrating the frontend and backend was another critical aspect of our testing. We needed to ensure seamless transitions between different functionalities, such as switching between classifying and training models. It was imperative that the correct model was loaded and utilized for each specific user input. This required extensive manual testing, simulating a myriad of potential user interactions to guarantee robustness and reliability.

The integration testing was particularly challenging, as it required a deep understanding of both the frontend and backend workings. Each test case was carefully designed to mimic real-world user scenarios. We systematically tested all possible combinations of user inputs and actions to ensure that the application behaved as expected. This included testing for edge cases and potential error scenarios, ensuring that the application was not only functional but also resilient to unexpected user actions.

Our rigorous testing methodology, encompassing both frontend and backend aspects, was instrumental in developing a robust and user-friendly application. The iterative process of incorporating feedback from various stakeholders, including the client, peers, and our own testing insights, led to a product that not only met but exceeded the initial requirements. This comprehensive approach to testing and evaluation has been pivotal in ensuring the success and reliability of our application.

User Manual

There are several features that have been added or are in the process of being implemented in our project. Users will be greeted with an intuitive and user-friendly home page intended to create a smooth and engaging experience. The main page acts as the main hub, with a clean and structured style that grabs the user's attention right away. Several tabs are carefully positioned at the top of this interface to assist easy navigation. These tabs serve as portals to other areas of the website, each with its own set of features and information about the project. Users may quickly explore and obtain the relevant material by simply clicking on the related tabs, whether they are looking for project details, resources, or specialized functionality.



CrisisLLM Capstone Project

The CrisisLLM Capstone Project is a collaborative effort by a team of dedicated students at Virginia Tech. We are seniors majoring in Computer Science and passionate about natural language processing and machine learning. Our mission is to develop a specialized crisis event classification model using pre-trained BERT language models.

We aim to identify crisis events from a variety of textual sources, including social media posts, news articles, and online forums. Our model will enhance crisis management by providing timely and accurate information for first responders, government agencies, and humanitarian organizations.

We are excited to apply transfer learning techniques and adapt BERT to the domain of crisis management. Our work will contribute to the broader field of natural language processing and help improve disaster response and relief efforts.

Figure 2: Home Page

The carefully developed design guarantees that users, regardless of their knowledge of the project, can easily comprehend the structure of the website and navigate it. Each tab has clear and short names, fostering a user-friendly experience for both novice and expert users. Users may anticipate an interactive and visually appealing home page to serve as the beginning point for a complete investigation of the project's different features as they go across the website.

The 'Fine Tune BERT' option takes visitors to an interactive website where they may enter data and actively participate in the project, tailoring the BERT model to their exact requirements. If visitors are looking for a certain model, the 'Model List' tab takes them to a dedicated page with a variety of alternatives, each with their own set of features. For those curious about the minds behind the project, the 'About Us' tab provides insights into our team's expertise and collaborative efforts. To ensure users make the most of our project, the 'Manual' tab allows for easy access to a downloadable user manual, offering detailed instructions and guidelines. In the following sections, we will go further into each tab, offering a thorough examination of its features.

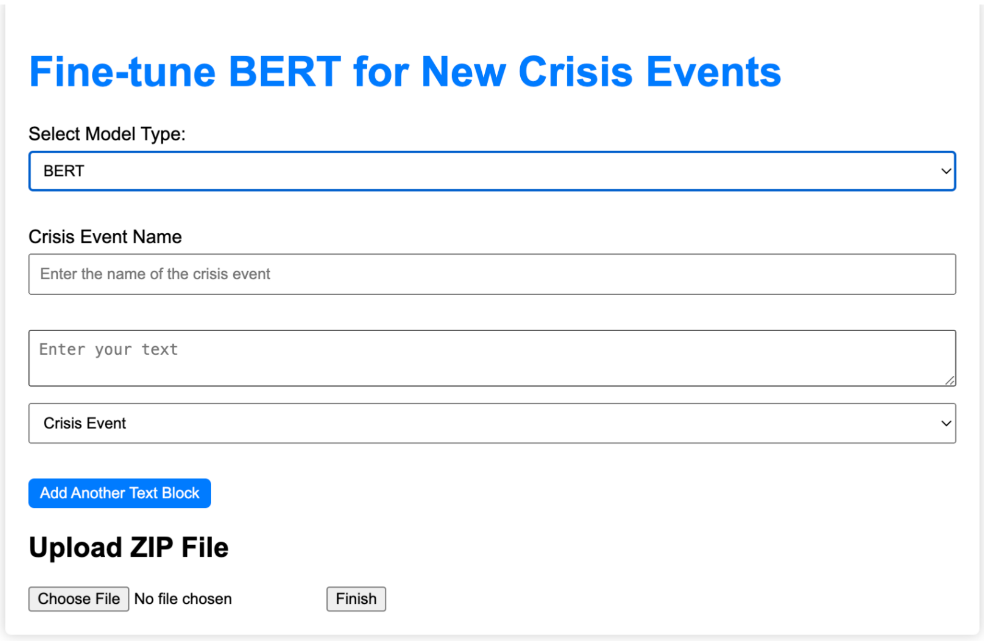


Figure 3: Bert Page

When users first enter the website they will want to input their own data in order to train their own model, they will do such on the *Fine Tune Bert* page, pictured above. There are several ways to do this. One of the first things that the user must do is select whether they want to train the model using DistilBert or Bert.

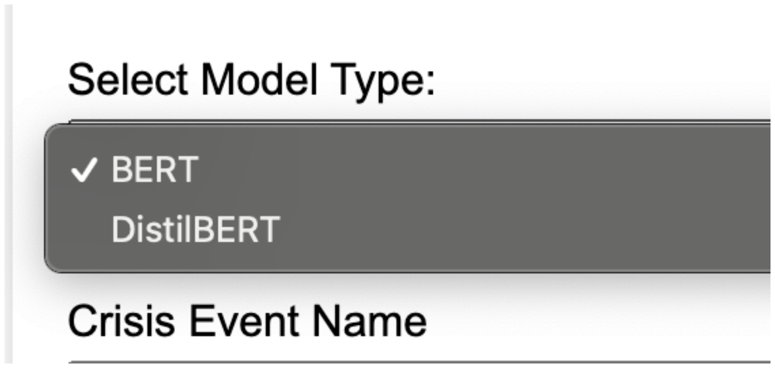


Figure 4: Model Type Selection

There are several differences between the two models that the user should consider before making a choice. The primary difference lies in their scale and computational efficiency. BERT, or Bidirectional Encoder Representations from Transformers, is a large and powerful model that captures intricate contextual information from both the left and right sides of a word. However, its sheer size demands significant computational resources. In contrast, DistilBERT, short for Distill BERT, is a distilled version of BERT, created to maintain similar performance while significantly reducing the model's size and computational requirements. DistilBERT achieves this by removing certain components and reducing the number of parameters. While BERT may outperform DistilBERT on certain complex tasks due to its larger capacity, DistilBERT serves as a more efficient alternative.

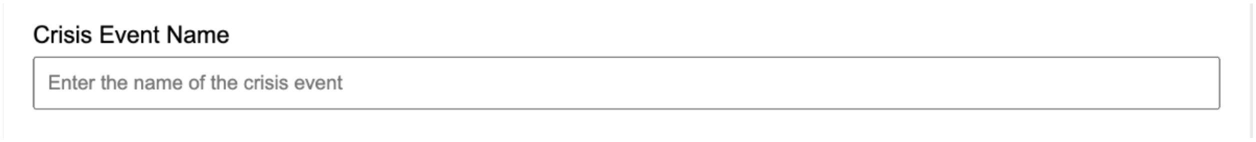


Figure 5: Text block for Crisis Event Name

Next, users have the pivotal task of specifying the crisis event they wish to train the model on. This step is crucial for tailoring the model to address specific nuances and characteristics associated with different crisis events. Upon pressing the submit button at the bottom of the page, a significant operation takes place behind the scenes—the model is fine-tuned with the dataset related to the specified crisis event. Importantly, the resulting pickle file for the trained model is dynamically named as "crisiseventname_modeltype". This nomenclature ensures clear identification and organization, tying the model to its specific crisis event for future reference. Subsequently, this uniquely named pickle file becomes a key component presented in the Model List page on the next tab. This approach enhances user experience and facilitates easy navigation, allowing users to quickly identify and select the trained model associated with their specified crisis event when exploring the Model List.

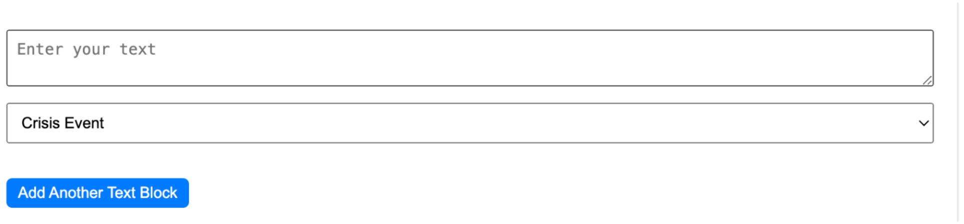
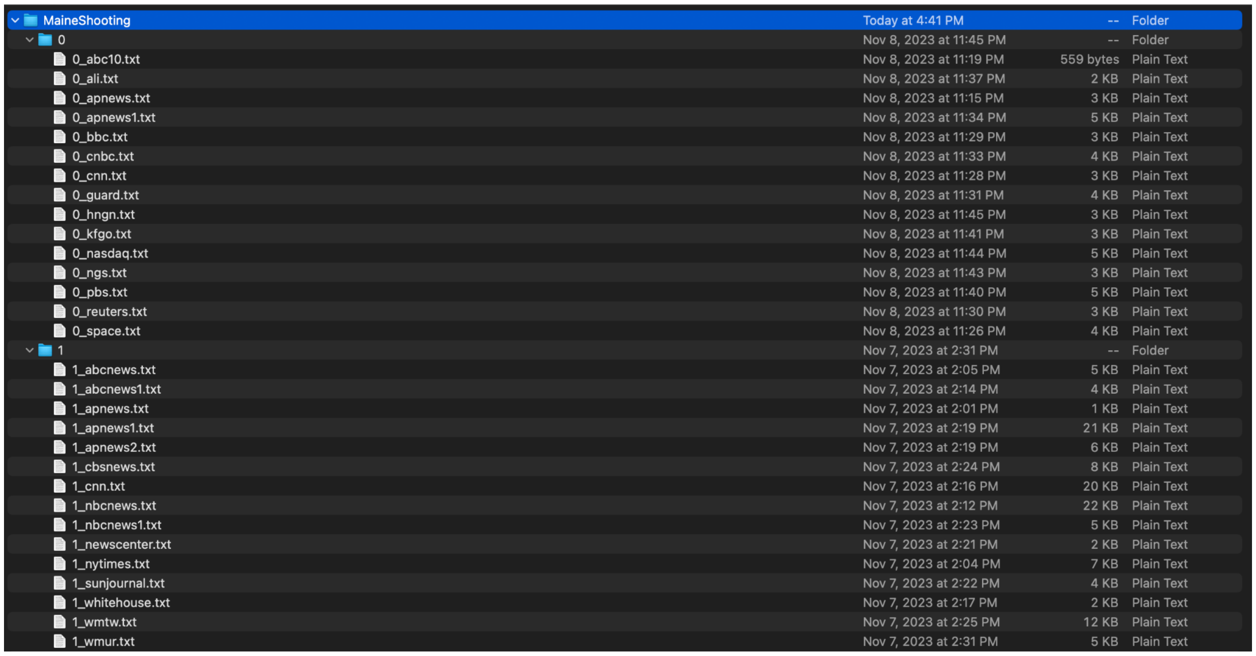


Figure 6: Adding Individual Text Blocks

Users are empowered to refine the model's understanding by inputting individual text blocks and categorizing them as either "Crisis Event" or "Not a Crisis Event." This interactive process allows users to actively shape the model's comprehension of relevant textual data. The presence of an "Add Text Block" button facilitates a dynamic experience, enabling users to input as many text blocks as necessary for robust training. Each added text block serves as a data point, contributing to the model's learning process. Once users have meticulously defined these text blocks, they can seamlessly press the "Finished" button on the page. At this point, the system collects and compiles all the inputted text blocks, utilizing them to fine-tune the model.

This iterative and user-driven approach ensures that the model is trained with a diverse range of examples, enhancing its ability to accurately distinguish between crisis and non-crisis events based on the patterns learned from user-provided text blocks.

The last feature on this page is the main feature that we want users to use. This feature allows users to input their own zip file of documents that are pre-classified as Crisis Events and Not Crisis Events. Below is a picture of what the zip file should look like. In the zip file, there should be two folders, labeled "0" and "1". 0, represents the text documents that are not Crisis Events and 1 represents the documents that are Crisis events.



MaineShooting	Today at 4:41 PM	--	Folder
0	Nov 8, 2023 at 11:45 PM	--	Folder
0_abc10.txt	Nov 8, 2023 at 11:19 PM	559 bytes	Plain Text
0_ali.txt	Nov 8, 2023 at 11:37 PM	2 KB	Plain Text
0_apnews.txt	Nov 8, 2023 at 11:15 PM	3 KB	Plain Text
0_apnews1.txt	Nov 8, 2023 at 11:34 PM	5 KB	Plain Text
0_bbc.txt	Nov 8, 2023 at 11:29 PM	3 KB	Plain Text
0_cnbc.txt	Nov 8, 2023 at 11:33 PM	4 KB	Plain Text
0_cnn.txt	Nov 8, 2023 at 11:28 PM	3 KB	Plain Text
0_guard.txt	Nov 8, 2023 at 11:31 PM	4 KB	Plain Text
0_hngn.txt	Nov 8, 2023 at 11:45 PM	3 KB	Plain Text
0_kfgo.txt	Nov 8, 2023 at 11:41 PM	3 KB	Plain Text
0_nasdaq.txt	Nov 8, 2023 at 11:44 PM	5 KB	Plain Text
0_ngs.txt	Nov 8, 2023 at 11:43 PM	3 KB	Plain Text
0_pbs.txt	Nov 8, 2023 at 11:40 PM	5 KB	Plain Text
0_reuters.txt	Nov 8, 2023 at 11:30 PM	3 KB	Plain Text
0_space.txt	Nov 8, 2023 at 11:26 PM	4 KB	Plain Text
1	Nov 7, 2023 at 2:31 PM	--	Folder
1_abcnews.txt	Nov 7, 2023 at 2:05 PM	5 KB	Plain Text
1_abcnews1.txt	Nov 7, 2023 at 2:14 PM	4 KB	Plain Text
1_apnews.txt	Nov 7, 2023 at 2:01 PM	1 KB	Plain Text
1_apnews1.txt	Nov 7, 2023 at 2:19 PM	21 KB	Plain Text
1_apnews2.txt	Nov 7, 2023 at 2:19 PM	6 KB	Plain Text
1_cbsnews.txt	Nov 7, 2023 at 2:24 PM	8 KB	Plain Text
1_cnn.txt	Nov 7, 2023 at 2:16 PM	20 KB	Plain Text
1_nbcnews.txt	Nov 7, 2023 at 2:12 PM	22 KB	Plain Text
1_nbcnews1.txt	Nov 7, 2023 at 2:23 PM	5 KB	Plain Text
1_newscenter.txt	Nov 7, 2023 at 2:21 PM	2 KB	Plain Text
1_nytimes.txt	Nov 7, 2023 at 2:04 PM	7 KB	Plain Text
1_sunjournal.txt	Nov 7, 2023 at 2:22 PM	4 KB	Plain Text
1_whitehouse.txt	Nov 7, 2023 at 2:17 PM	2 KB	Plain Text
1_wmtw.txt	Nov 7, 2023 at 2:25 PM	12 KB	Plain Text
1_wmur.txt	Nov 7, 2023 at 2:31 PM	5 KB	Plain Text

Figure 7: Zip file Example

Users will then input the zip file and press the finish button. When the finish button is pressed, a Model will be trained on the data set for the user to use on the "Model List" page.

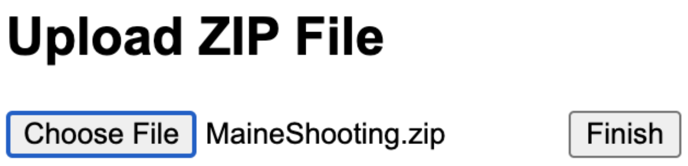
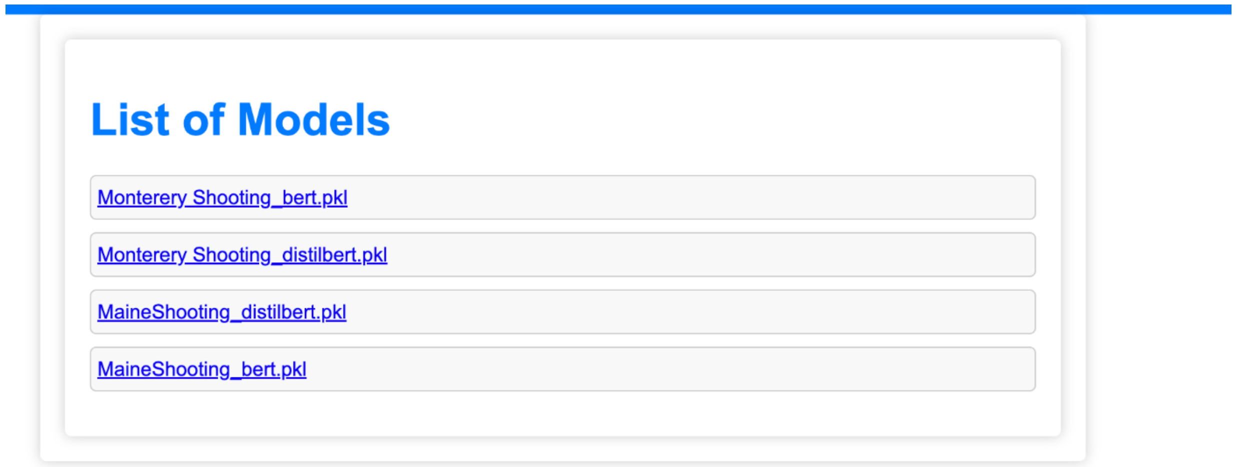


Figure 8: Zip file Upload

As the zip file containing relevant data is submitted, the backend initiates the training procedure. Behind the scenes, the model undergoes a series of epochs, with each epoch representing a complete pass through the training dataset. This iterative training process allows the model to gradually refine its understanding and adapt to the nuances present in the provided data. The uploaded zip file serves as a comprehensive source of information, enabling the model to learn and generalize patterns associated with crisis and non-crisis events. his automated and streamlined approach ensures that the model is continuously fine-tuned based on the user-provided dataset, optimizing its predictive capabilities for accurately discerning crisis-related content. The training of the model will take some time.

After the model has been trained, it will appear on the model list page with the appropriate name as aforementioned. Here there will be a list of models that have been trained and users can click on.



List of Models

Monterery Shooting_bert.pkl

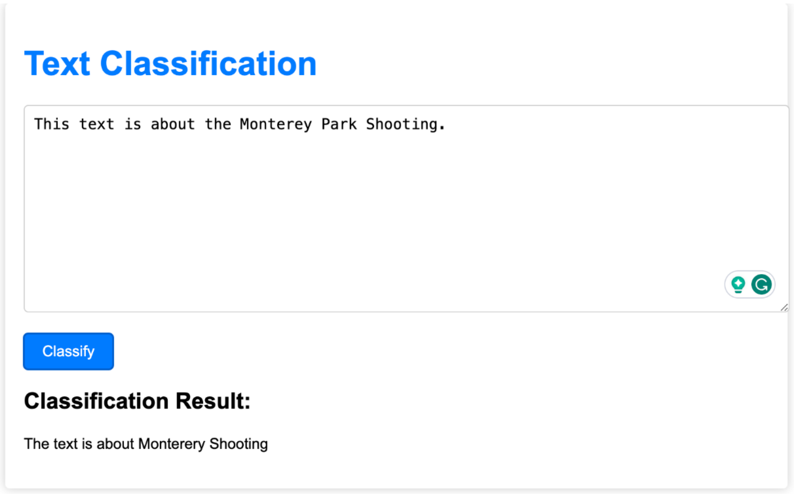
Monterery Shooting_distilbert.pkl

MaineShooting_distilbert.pkl

MaineShooting_bert.pkl

Figure 9: List of Models

When the user clicks one of these models, they can now classify text to see if it relates to the crisis event. In the example below, the model that was the Monterey Shooting that was trained using BERT. Included in the text bar is an example of text that is classified as that relates the the monterey park shooting. When the classify button is pressed it will print the result of the classification.



Text Classification

This text is about the Monterey Park Shooting.

Classification Result:

The text is about Monterery Shooting

Figure 10: Classification page and result

Our application's design philosophy revolves around simplicity and user accessibility. We have adopted a minimalistic interface for the following reasons: Clarity: By reducing clutter and unnecessary elements, we aim to provide a clean and unambiguous user experience. Ease of Use: The minimalistic design ensures that users can quickly locate and access the essential features without distractions. Accessibility: We prioritize accessibility by using clear typography, intuitive navigation, and high contrast elements to accommodate users with diverse needs. Efficiency: The streamlined interface minimizes the learning curve, allowing users to focus on their primary task—identifying and understanding crisis events from URLs.

Developer Manual

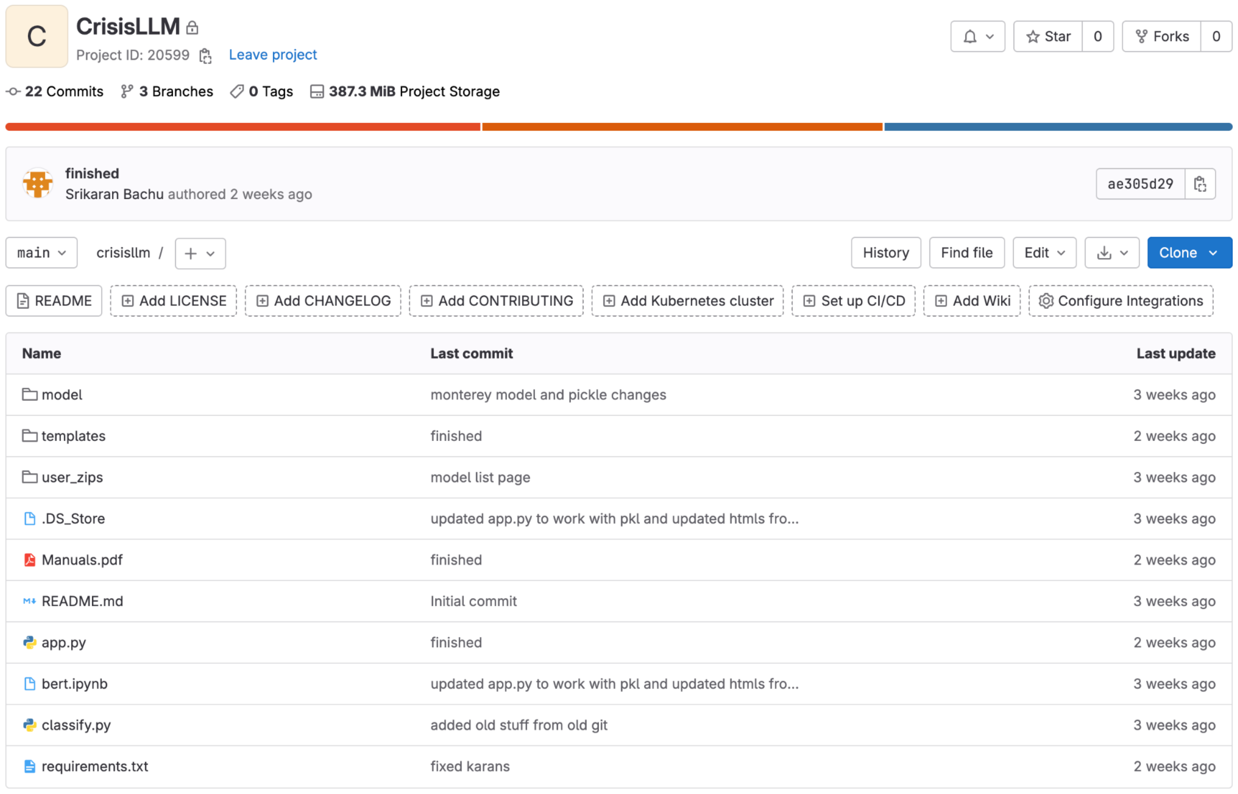
Step 1: Clone the repository

Open a new terminal and cd into an empty directory where you want the project to be cloned into.

Run the following command:

```
git clone https://git.cs.vt.edu/bhargava/crisisllm
```

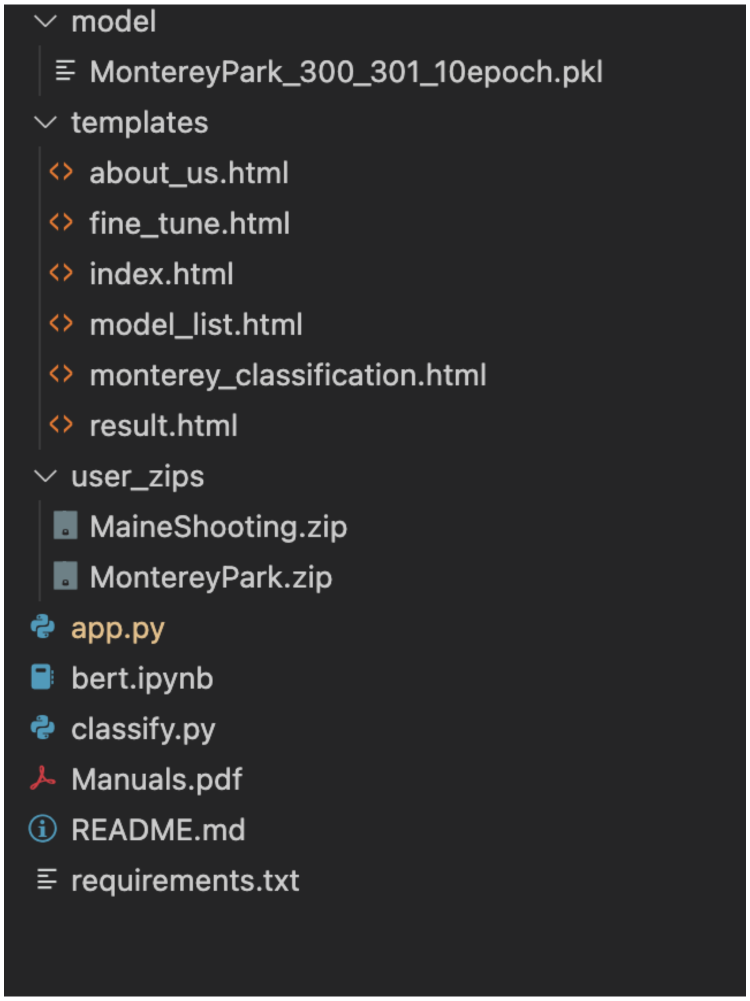
This will clone the git repo of this gitlab page below:



Name	Last commit	Last update
model	monterey model and pickle changes	3 weeks ago
templates	finished	2 weeks ago
user_zips	model list page	3 weeks ago
.DS_Store	updated app.py to work with pkl and updated htmls fro...	3 weeks ago
Manuals.pdf	finished	2 weeks ago
README.md	Initial commit	3 weeks ago
app.py	finished	2 weeks ago
bert.ipynb	updated app.py to work with pkl and updated htmls fro...	3 weeks ago
classify.py	added old stuff from old git	3 weeks ago
requirements.txt	fixed karans	2 weeks ago

Figure 11: Gitlab CrisisLLM page

If everything is properly cloned the directory should look like the figure below:



```
model
    MontereyPark_300_301_10epoch.pkl
templates
    about_us.html
    fine_tune.html
    index.html
    model_list.html
    monterey_classification.html
    result.html
user_zips
    MaineShooting.zip
    MontereyPark.zip
app.py
bert.ipynb
classify.py
Manuals.pdf
README.md
requirements.txt
```

Figure 12: Expected Directory

Step 2: Python+Packages installation

Install a version of Python 3.7 or higher. We used Python 3.9.12 for our version.

Next the necessary python packages must be installed as shown below:

Package	**Version**
Flask	3.00
Werkzeug	3.00
torch	2.1.0
transformers	4.35.0
scikit-learn	1.3.2

Figure 13: Table of Required Python3 Packages

To easily download the packages, you can run the following command:

```
pip3 install -r requirements.txt
```

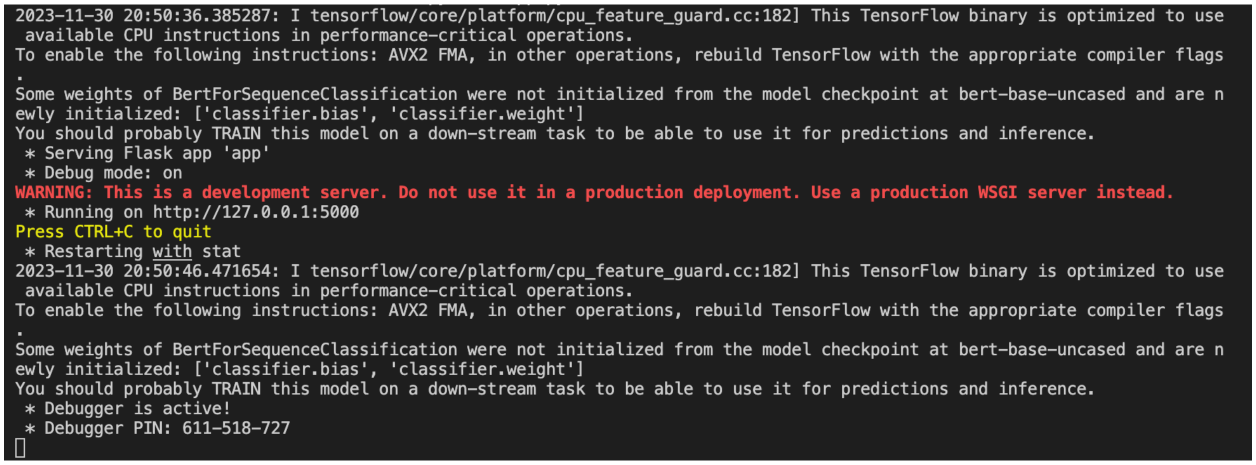
3. Running the flask app

To start the flask application run the app.py file.

This can be accomplished by running the following command:

```
python3 app.py
```

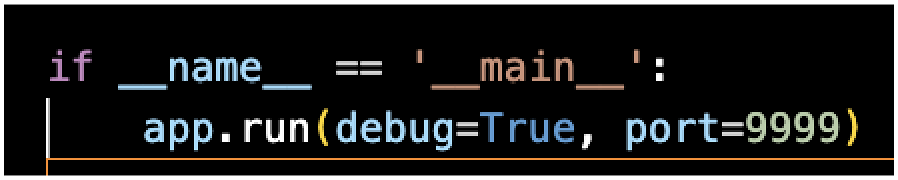
The console output should look like this



```
2023-11-30 20:50:36.385287: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use
 available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags
.
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are n
ewly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
2023-11-30 20:50:46.471654: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use
 available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags
.
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are n
ewly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
 * Debugger is active!
 * Debugger PIN: 611-518-727
```

Figure 14: Expected Terminal Output

You can change the port (optional) by making this change in app.py - line 309:



```
if __name__ == '__main__':
    app.run(debug=True, port=9999)
```

Figure 15: app.py code snippet for port change

To run the flask app command+click(mac) or control+click(windows) on the server link (in our case Running on http://127.0.0.1:5000).

4. System Architecture & File Structure

Backend:

- /app.py: Script for running Flask application
- /bert.ipynb: Script for model training and bert Initialization
- /models: Folder filled with models that have been trained on the FrontEnd
    - These models were created and saved in the folder by app.py
- /user_zips: Folder filled with zips that were input by the users

Frontend:

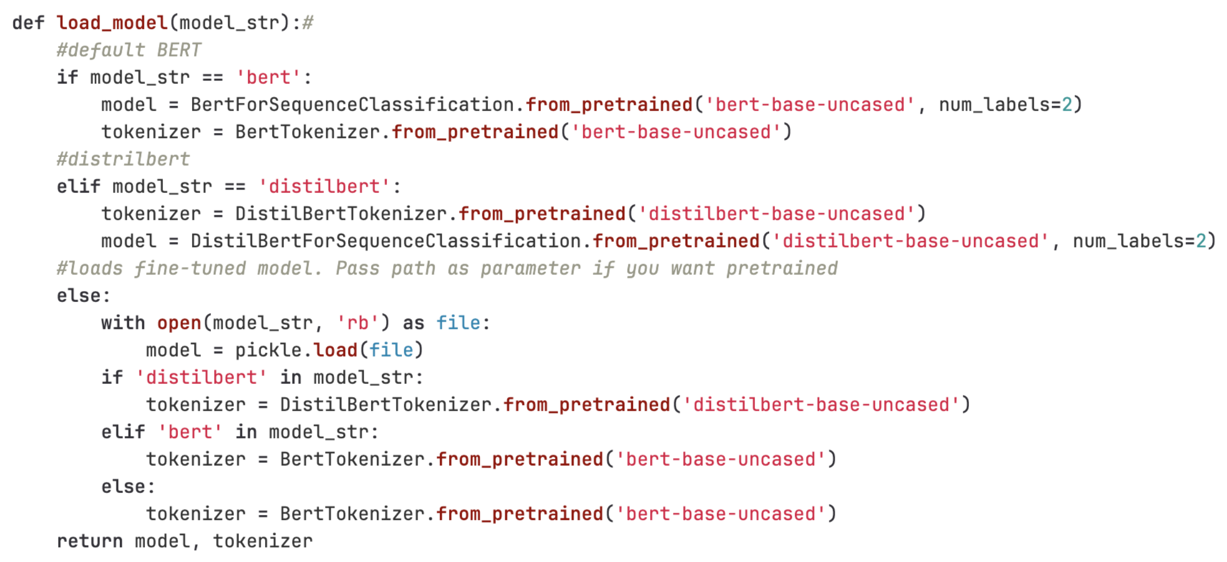
- /templates: Folder filled with html files that are loaded on the front end
    - fine_tune.html
    - about_us.html
    - index.html

- model_list.html
- monterey_classification.html
- result.html

app.py:

App.py is a Python script is a Flask web application designed for fine-tuning and deploying BERT and DistilBERT models for text classification tasks, specifically geared towards crisis event identification.

The load_model function in app.py serves as a crucial component for initializing and configuring language models within the CrisisLLM Flask application. When the function receives a model string as a parameter, it intelligently determines the model type based on the string, allowing users to choose between BERT, DistilBERT, or a custom fine-tuned model. In the case of BERT, the function initializes a BERT-based sequence classification model and its corresponding tokenizer using the popular 'bert-base-uncased' pre-trained weights. Similarly, for DistilBERT, it employs the 'distilbert-base-uncased' pre-trained weights to set up a DistilBERT sequence classification model and its tokenizer. In the event of a custom model specified by a file path, the function loads the pre-trained model using the pickle module, discerns its type (BERT or DistilBERT), and configures the appropriate tokenizer accordingly. This dynamic model loading capability enhances the adaptability of the CrisisLLM application, allowing users to seamlessly switch between different state-of-the-art language models. You can see the function below:



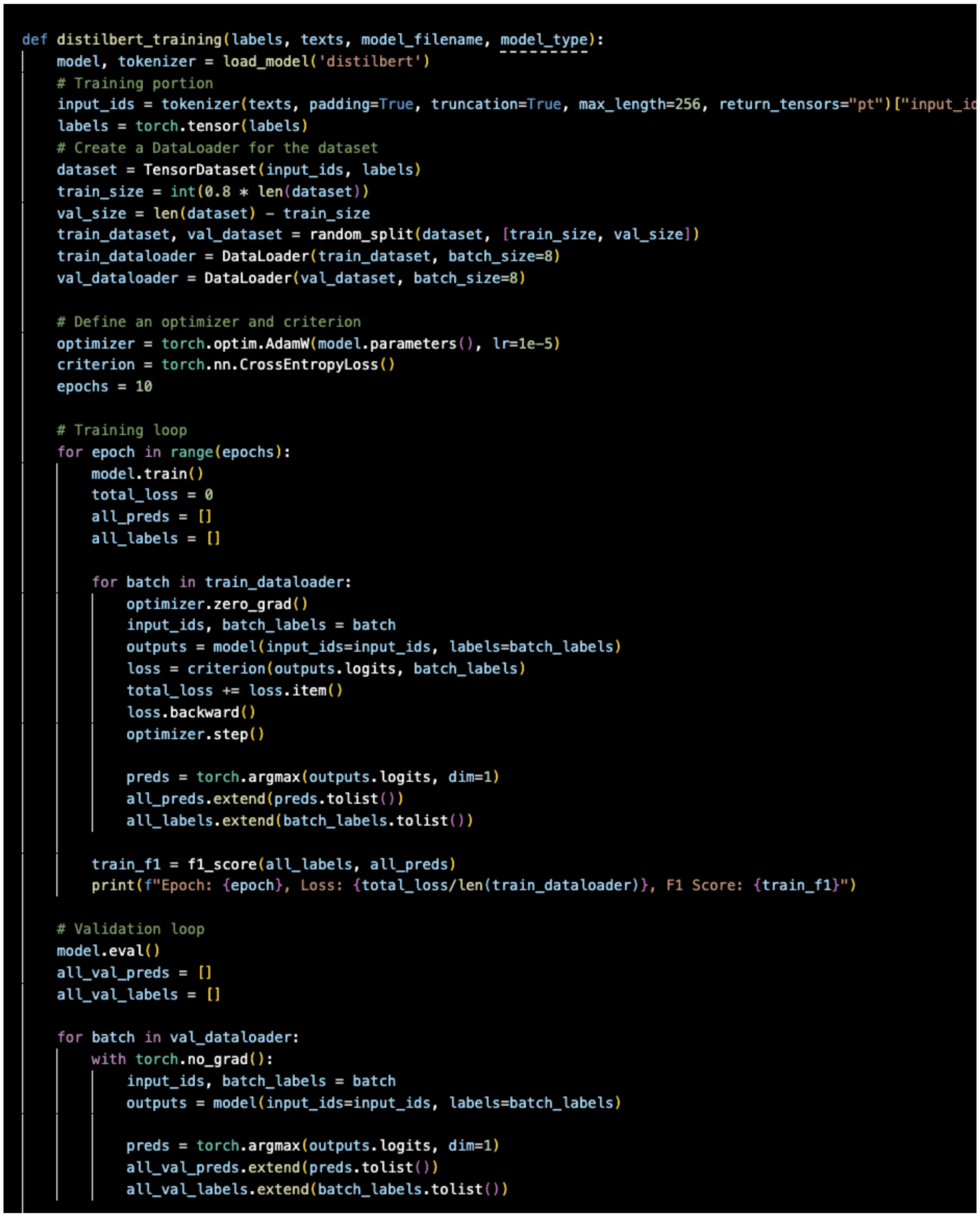
```python
def load_model(model_str):#
    #default BERT
    if model_str == 'bert':
        model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)
        tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
    #distrilbert
    elif model_str == 'distilbert':
        tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
        model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=2)
    #loads fine-tuned model. Pass path as parameter if you want pretrained
    else:
        with open(model_str, 'rb') as file:
            model = pickle.load(file)
        if 'distilbert' in model_str:
            tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
        elif 'bert' in model_str:
            tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
        else:
            tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
    return model, tokenizer
```

Figure 16: Load model function

In the DistilBERT training function, the model is loaded along with its tokenizer. The labeled text data, extracted from the uploaded zip file, undergoes preprocessing and is divided into training and validation sets. The model is then trained over a specified number of epochs, with each epoch comprising a training loop where gradients are computed, backpropagation is performed, and the model parameters are updated. The training progress, including loss and F1 score, is printed to the console. Following training, the model's performance is evaluated on a validation set, and the resulting F1 score is printed. The trained model is serialized using pickle.dump and saved to a specified file path.

Similarly, the BERT training function follows a comparable structure, where the BERT model and tokenizer are loaded, and the dataset is split for training and validation. The model undergoes training over multiple epochs, and training metrics are printed for monitoring. After validation, the trained BERT model is serialized and saved.

Both training procedures include essential components such as data loading, model initialization, optimization using AdamW, loss calculation with cross-entropy, and performance evaluation using F1 score. These functions collectively contribute to the training pipeline, enabling the models to learn and generalize from the provided labeled data for subsequent classification tasks.



```python
def distilbert_training(labels, texts, model_filename, model_type):
    model, tokenizer = load_model('distilbert')
    # Training portion
    input_ids = tokenizer(texts, padding=True, truncation=True, max_length=256, return_tensors="pt")["input_id
    labels = torch.tensor(labels)
    # Create a DataLoader for the dataset
    dataset = TensorDataset(input_ids, labels)
    train_size = int(0.8 * len(dataset))
    val_size = len(dataset) - train_size
    train_dataset, val_dataset = random_split(dataset, [train_size, val_size])
    train_dataloader = DataLoader(train_dataset, batch_size=8)
    val_dataloader = DataLoader(val_dataset, batch_size=8)

    # Define an optimizer and criterion
    optimizer = torch.optim.AdamW(model.parameters(), lr=1e-5)
    criterion = torch.nn.CrossEntropyLoss()
    epochs = 10

    # Training loop
    for epoch in range(epochs):
        model.train()
        total_loss = 0
        all_preds = []
        all_labels = []

        for batch in train_dataloader:
            optimizer.zero_grad()
            input_ids, batch_labels = batch
            outputs = model(input_ids=input_ids, labels=batch_labels)
            loss = criterion(outputs.logits, batch_labels)
            total_loss += loss.item()
            loss.backward()
            optimizer.step()

            preds = torch.argmax(outputs.logits, dim=1)
            all_preds.extend(preds.tolist())
            all_labels.extend(batch_labels.tolist())

        train_f1 = f1_score(all_labels, all_preds)
        print(f"Epoch: {epoch}, Loss: {total_loss/len(train_dataloader)}, F1 Score: {train_f1}")

    # Validation loop
    model.eval()
    all_val_preds = []
    all_val_labels = []

    for batch in val_dataloader:
        with torch.no_grad():
            input_ids, batch_labels = batch
            outputs = model(input_ids=input_ids, labels=batch_labels)

            preds = torch.argmax(outputs.logits, dim=1)
            all_val_preds.extend(preds.tolist())
            all_val_labels.extend(batch_labels.tolist())
```

Figure 17: Training function example

Lessons Learned:

The journey of developing the Crisis Events Language Model (CrisisLLM) has been an enriching experience, marked by the exploration of various technologies, unexpected challenges, and continuous learning. Reflecting on our journey, several key aspects emerge, including the timeline, encountered problems, solutions devised, and the invaluable lessons gained from our endeavors.

Timeline/Schedule

9/14-10/1	Planned workflow and techstack, setup environments, identified techstack, created basic frontend. Began model training
10/1 - 10/15	Backend: BERT tokenizes sentences and returns ID and index of small classifications for testing. Began web scraping from google custom engine using beautifulsoup. Frontend: Finalized all pieces of frontend, needed to link to backend.
10/16-11/1	Frontend: Successfully added in zip file input and linked to backend, submitted interim report. Backend: Continuing to finetune with new data. Lost a team member, scope of project changed.
11/1-11/30	Backend: Linked to frontend with Flask, Incorporated DistilBERT. Frontend: Changed frontend to incorporate all models. Small touches made for presentability.

Figure 18: Semester Schedule

The unexpected departure of a team member presented an intricate challenge, necessitating a reevaluation of our project scope and workflow distribution. This unforeseen setback prompted us to adapt and restructure our approach to accommodate the reduced team size. This led to us changing the scope of the project and working with text file data, instead of web scraped data from different websites. Ultimately, we were able to overcome this milestone with strong communication, and a good plan of attack.

In response to the unexpected challenge, we opted for an agile project management approach, swiftly adjusting the project scope to align with the reduced team size. The decision to temporarily set aside the URL-based text classification and web scraping components allowed us to focus on the core aspects of CrisisLLM, ensuring progress and minimizing disruption.

The development of the backend using Flask introduced our team to the world of web application frameworks. Learning Flask, with its powerful features and flexibility, enabled us to seamlessly integrate our machine learning models into a robust and scalable web application. This experience enhanced our proficiency in web development and equipped us with valuable skills applicable to future projects.

Navigating the complexities of BERT (Bidirectional Encoder Representations from Transformers) and DistilBERT was a significant learning curve for our team. Understanding the intricacies of these state-of-the-art natural language processing models required in-depth exploration of transformer architectures, attention mechanisms, and transfer learning. Another lesson we learned was during the finetuning process of changing our language model, BERT, output layers. This was new to us and required us to get better at using transfer learning. We were able to learn pyTorch TensorDataset library and leverage that to finetune our model. The knowledge gained from working with BERT and DistilBERT has broadened our understanding of advanced NLP techniques, paving the way for future endeavors in language modeling.

Looking ahead, our future work involves delving deeper into the realm of advanced language models. Expanding beyond BERT and DistilBERT, we aspire to explore models such as RoBERTa, GPT-3, and XLNet. Each of these models offers unique architectures and training methodologies, presenting opportunities for enhanced crisis event classification and analysis.

To further augment the CrisisLLMs capabilities, we envision integrating multimodal data sources. This involves combining textual information with visual and auditory cues, fostering a more comprehensive understanding of crisis events. Exploring this avenue will contribute to a more holistic and nuanced analysis of complex situations.

Future iterations of CrisisLLM will prioritize real-time adaptability by exploring the integration of streaming data sources. Incorporating technologies that enable the model to dynamically adjust to events will enhance the system's responsiveness and contribute to timely predictions.

Ensuring the longevity of CrisisLLM's effectiveness requires a commitment to continuous improvement. Implementing mechanisms for ongoing model evaluation, monitoring for potential drift, and regular model updates will be essential for maintaining optimal performance over time.

The journey of developing the Crisis Events Language Model not only expanded our technical skills but also underscored the significance of adaptability, resilience, and collaborative problem-solving. Learning to navigate unexpected challenges, mastering new technologies like Flask and advanced language models, and envisioning future enhancements have collectively shaped a transformative learning experience. The CrisisLLM stands as a testament to the team's dedication, adaptability, and ongoing pursuit of excellence in the dynamic field of machine learning and natural language processing.

Acknowledgements

Our client was our professor, Mohamed Farag. Dr. Farag is a research associate in the Center for Sustainable Mobility (CSM). Our group met with Dr. Farag every week to improve clarity on requirements and display our progress. During every meeting we gained greater clarity on the specifications of the application and specific modifications that were required. Professor Farag can be reached at mfarag@vtti.vt.edu.

References

James. "Classify Text with BERT." TensorFlow, October 2023.
https://www.tensorflow.org/text/tutorials/classify_text_with_bert.

Pham, Khang. "Text Classification with BERT." Medium, May 9th, 2023.
https://medium.com/@khang.pham.exxact/text-classification-with-bert-7afaacc5e49b.

Maximilien Roberti. "Fastai with Transformers (BERT, RoBERTa)." Towards Data Science, Nov 27, 2019.
https://towardsdatascience.com/fastai-with-transformers-bert-roberta-xlnet-xlm-distilbert-4f41ee18ecb2.

Roberti, Marco. "Fastai with Transformers (BERT, RoBERTa)." Kaggle, 2019.
https://www.kaggle.com/code/maroberti/fastai-with-transformers-bert-roberta.

Saini, Amar. "NLP from Scratch with PyTorch, FastAI, and HuggingFace." Epoching Blog, 27 June 2021.
https://amarsaini.github.io/Epoching-Blog/jupyter/nlp/pytorch/fastai/huggingface/2021/06/27/NLP-from-Scratch-with-PyTorch-FastAI-and-HuggingFace.html.