Department of Computer Science
Faculty of Computing and Information Technology
(FCIT) University of the Punjab

TruthLens- An Explainable AI System for Fake News Detection

Final Year Project Proposal
Session 2025-2026

A project submitted in partial
fulfilment of the BS in Computer
Science
Fall 2025

15 December 2025

FYP-PHASE1 REPORT

Group Members:

Abdullah Liaqat- BCSF22M520- Project Leader

Muhammad Sajjad- BCSF22M540

Table of Contents

Chapter 1: Introduction 8

1.1 Background 8

1.2 Problem Statement 8

1.3 Proposed Solution 10

1.4 Main Objectives 10

1.5 Assumptions & Constraints 10

1.5.1 Assumptions 10

1.5.2 Constraints 11

1.6 Project Scope 11

1.7 Software Development Life Cycle Model 11

Chapter 2: Requirement Analysis 13

2.1 Literature Review 13

2.1.1 Industrial Efforts in Misinformation Detection 13

2.1.2 Evolution of Academic Approaches 13

2.1.3 Early Machine Learning Techniques 13

2.1.4 Deep Learning-Based Models 14

2.1.5 The Transformer Revolution 14

2.1.6 Explainable Artificial Intelligence (XAI) in NLP 15

2.1.7 Hybrid and Integrated Approaches 15

2.1.8 Comparative Analysis 17

2.2 Research Gap and Motivation 17

2.3 Requirements Elicitation 18

2.3.1 Functional Requirements 18

2.3.2 Non-Functional Requirements 18

2.3.3 Requirements Traceability Matrix 19

2.4 Use Case Descriptions 20
2.4.1 Use Case 1: Analyze News 20
2.4.2 Use Case 2: Evaluate Model Performance 20
2.4.3 Use Case 3: Visualize Explanations 20
Chapter 3: Proposed Methodology and System design 21
3.1 Overview 21
3.2 Datasets 23
3.3 Data Preprocessing 23
3.3.1 Steps and tools 23
3.3.2 Token length normalization (example) 24
3.4 Model Architecture 25
3.4.1 Self-attention mechanism 25
3.4.2 Encoder and classification 26
3.4.3 Loss function and optimization 26
3.4.4 Backend Class Diagram 29
3.5 Explainability Techniques 29
3.5.1 LIME-local linear surrogate 29
3.5.2 SHAP- Shapley value explanations 30
3.5.3 Integration rules and caching 30
3.6 System Architecture and Component Interaction 33
3.6.1 Data flows 33
3.7 Flow of Methodology (textual synthesis) 36
3.8 Algorithm / Model Selection 39
3.8.1 Problem Nature 39
3.8.2 Why Transformer-Based Models? 40

3.8.3 Why BERT Specifically? 40
3.8.4 Fine-Tuning Strategy 40
3.8.5 Explainability Compatibility 41
3.8.6 Why Not Real-Time Fact Checking? 41
3.8.7 Summary of Model Selection Justification 42
Chapter 4: Implementation 43
4.1 Work Breakdown Structure (WBS) 43
4.2 Team Roles and Responsibilities 45
4.3 Tools and Technologies 46
4.3.1 Programming Language 46
4.3.2 Development Platforms 46
4.3.3 Libraries and Frameworks 46
4.3.4 Web and API Development 47
4.3.5 Database Management 47
4.3.6 Development Tools and Version Control 47
4.3.7 Justification 48
4.4 Implementation details 48
4.5 Screenshots of Prototype 51
4.6 Challenges During Implementation 57
4.6.1 Limited Computational Resources 57
4.6.2 High Computational Cost of Explainability (SHAP) 58
4.6.3 Dataset Heterogeneity and Input Length Variability 58
4.6.4 Integration Complexity Across System Components 59
4.6.5 Latency and Interpretability Trade-off 59
4.6.6 Time Constraints and Academic Deadlines 60

4.6.7 Development and Debugging Challenges 60
4.6.8 Engineering Insight Gained 60
References 61

List of Figures

Figure 3.1: System Architecture 16
Figure 3.2: Pre-processing Module 23
Figure 3.3: Model Training 26
Figure 3.4: Back-end class Diagram 27
Figure 3.5: Shap Generation 29
Figure 3.6: DFD level 1 30
Figure 3.7: Component Diagram 32
Figure 3.8: Front-end class diagram 33
Figure 3.9: Fake news prediction 35
Figure 3.10: Metrics Evaluation and result display 36
Figure 3.11: Sequence Diagram 37
Figure 4.1: Home Screen 49
Figure 4.2: Analyze Screen 50
Figure 4.3: Lime Screen 51
Figure 4.4: Shap Screen 52
Figure 4.5: Evaluation Dashboard 53
Figure 4.6: ABout 54

List of Tables

Table 2.1: Literature Comparison 16
Table 2.2 : Traceability Matrix 19
Table3.1:Dataset 24
Table 3.2 : Preprocessing Operations 25
Table4.1: Work Breakdown Structure 44
Table 4.2:Roles and responsibilities of team members 46
Table 4.3: Libraries and Framework 47
Table 4.4: Implementation 49
Table 4.5: Model Configuration Parameters 50

Chapter 1: Introduction

1.1 Background

These days, digital media increases the spread of false news through the release of information by social media and online news platforms without verification. Although this connectivity has indeed promoted the sharing of information in real time, it has also provided a fertile ground for the dissemination of misinformation and fake news. The increasing availability of content-generation tools has blurred the line between legitimate journalism and fake stories. This has led to the development and emergence of misinformation as a multifaceted sociotechnical issue that is profoundly affecting society, politics, and the economy [1].

In recent years, the literature on fake news detection and explainable artificial intelligence (XAI) has expanded at a high rate. Early systems were rule-based and statistical; however, they were not flexible enough to accommodate language variations. Subsequently, classical machine learning algorithms like Naive Bayes, Support Vector Machines (SVM), and Decision Trees took over, later being outperformed by neural networks and then, most recently, transformer-based neural networks like BERT [6]. Although these are made, one persistent weakness remains: interpretability. Users and policymakers require models that cannot only categorise news as fake or real but also provide transparent and verifiable explanations for their reasoning [2], [9].

The chapter provides a comprehensive overview of current industrial and academic strategies for misinformation detection, including the evolution from classic models to deep learning models and the incorporation of XAI frameworks such as LIME and SHAP. The discussion ultimately identifies a critical research gap driving the proposed system: a BERT-based explainable fake news detection framework capable of making reliable and interpretable predictions

1.2 Problem Statement

The blistering development of the online information resulted in the increasing number of individuals who do not know how to draw the line between genuine journalism and a fake

narrative. Even though deep learning has contributed greatly to the detection of fake-news, most of the models are black-box machines; they can make accurate predictions but fail to give a reason why a given article is either fake or real [5][6].

This interpretability results in a high level of trust deficit. Stakeholders like journalists, policymakers and researchers might be reluctant to use automated systems that are unable to explain their decisions. An explanatory predictive model would pose a danger of creating a bias that reinforces, spreads doubt, and enhances a lack of credibility in sensitive socio-political situations. In this manner, achieving transparency is as crucial as improving accuracy in the detection of misinformation.

Naive Bayes or Support Vector Machines (SVM) are traditional models that have been moderately successful but have poor contextual comprehension [9]. Transformer-based models, such as BERT [6], incorporate contextual embeddings that capture in-depth semantic meaning, thereby significantly enhancing classification accuracy. However, these models are yet to be interpretable-people are unable to understand how a prediction came about.

In the meantime, Explainable AI (XAI) procedures like LIME [7] and SHAP [8] have been developed to explain intricate machine-learning models by determining which words or characteristics affect predictions most significantly. Although these methods are promising, they were mostly presented in isolation in academic literature, but were not combined into fake-news-detection systems that non-technical users can use.

Therefore, the core research question to be pursued in this project is whether a unified, explicable, and high-precision fake-news-detection system can unify state-of-the-art transformer models with interpretable AI engines in a manner that is computationally efficient and easy to use.

Problem Definition

Which algorithm can be designed to detect fake news and combine the precision of the transformer-based algorithms (like BERT) with the clarity of the XAI techniques (LIME and SHAP) to offer real-time, transparent, and credible classification outputs using a web-based interface?

1.3 Proposed Solution

The proposed system processes a text-based news input, applies a trained machine-learning model, and produces a classification of *real* or *fake*. To increase transparency, the system highlights contributing words using explainability techniques such as LIME and SHAP.

The solution is designed as:

- A text-analysis tool
- Built upon established, labeled datasets
- Capable of showing why certain patterns may indicate misinformation

This allows users to examine the linguistic structure of news rather than relying solely on manual judgment.

1.4 Main Objectives

The primary objectives are:

1. To develop a machine-learning model capable of analyzing textual news data using modern transformer-based architecture.
2. To emphasize the most significant textual elements that affect the classification, enhance user knowledge and credibility.
3. To develop a prototype interface with a simple interface, where users can fill in a news statement and see the results of interpretability.
4. To develop a workflow, research-grade and structured, to include the analysis of the dataset, its preprocessing, training a model, and integration of the system.

Secondary goals are exploration of data sets, comparative analysis and creating reusable elements to be extended in the future.

1.5 Assumptions & Constraints

1.5.1 Assumptions

- Textual news items contain identifiable linguistic patterns that can be learned from structured datasets.
- Users will provide the news text directly (no external scraping or multimedia content).

- The datasets used for training represent a sufficient variety of writing styles and misinformation patterns for a prototype system.

1.5.2 Constraints

- The system operates purely on **text analysis** and does not verify external facts or real-time events.
- The model is trained only on **available labeled datasets**, and therefore depends on the quality, variety, and labeling consistency of those datasets.
- No external databases or verification sources are connected; results are based solely on learned patterns.
- The system does not incorporate multimedia analysis or cross-checking sources, focusing strictly on the textual dimension.

1.6 Project Scope

The scope of TruthLens is intentionally defined to focus on textual features within labeled datasets. The project includes:

- Dataset analysis and preprocessing
- Model training using transformer-based architectures
- Local evaluation on structured datasets
- An explainable output interface (highlighting influential words)
- A prototype-level frontend to demonstrate functionality

The project **does not** include:

- Real-time web scraping
- Multimedia or image-based misinformation detection
- APIs External fact-verification.
- Monitoring of changing news reports
- Processing of non-textual or mixed-media content.

This breadth makes sure that the findings are consistent, explicable as well as that they can be measured academically.

1.7 Software Development Life Cycle Model

The SDLC Model of the project is an Incremental one and each component of the project is developed and tested in stages. This approach supports:

- Early model experimentation (training and baseline evaluation)
- Gradual integration of explainability methods
- Progressive development of the frontend prototype
- Step-by-step validation of each module

Chapter 2: Requirement Analysis

2.1 Literature Review

2.1.1 Industrial Efforts in Misinformation Detection

The disruptive technology giants, including Facebook, Twitter (now X), and Google, have installed large-scale automated systems that fight against misinformation. The platforms combine both a combination of keyword-based filtration, crowdsourced flagging, and machine learning algorithms to identify potentially misleading content [1].

Facebook launched its Third-Party Fact-Checking Program that is based on automated classifiers, as well as human reviewers. Twitter used community-based labelling through Birdwatch and introduced automated tagging of posts that had breached misinformation policies. Although these systems have been effective at creating limits on the spread of fake claims by the virus, researchers have criticized them for having a low degree of transparency and interpretability - users often get flags without knowing the reasoning behind them.

NewsGuard and Media Bias/Fact Check (MBFC) are independent tools to determine the credibility of news using semi-automated processes. These tools use a combination of rule-based indicators (e.g., domain reliability, ownership bias) and expert rating, in the form of browser extensions that show credibility scores. Nevertheless, their scalability is low, and they do not cover languages other than English, as well as the expert-driven approaches, as argued by Alalawi et al. [1].

Accordingly, as industrial systems are scalable and reachable, they lose explainability and accountability, which are the core of user trust. These deficiencies constitute the basic impetus to scholarly research on interpretable fake news detection.

2.1.2 Evolution of Academic Approaches

2.1.3 Early Machine Learning Techniques

The earliest academic works designed fake news detection as a supervised classification task with shallow machine learning algorithms. The major models were Naïve Bayes, SVMs, and Logistic

Regression [9]. These systems were heavily reliant on manual textual characteristics, including term frequency-inverse document frequency (TF-IDF), sentiment polarity, and word variety.

Indicatively, Shu et al. [9] point out that initially systems performed moderately clearly on curated datasets but failed to do so across domains - models trained on a specific dataset (e.g., political news) underperformed when applied to another (e.g., health misinformation). These strategies were fragile and failed to scale because they relied on the feature-engineering dependency.

Moreover, these models lacked the ability to model rich semantic and syntactic relations among words, a vital component in identifying advanced instances of misinformation that employ linguistic framing techniques to convince people, as opposed to a blunt falsehood.

2.1.4 Deep Learning-Based Models

To address such shortcomings, researchers have resorted to Deep Neural Networks (DNNs), specifically Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). CNNs learned local semantics of phrases, whereas RNNs, such as LSTM and GRU, learned the sequential text dependencies.

Nevertheless, according to Berrondo-Otermin and Sarasa-Cabezuelo [4], deep architectures also faced challenges with long-range dependencies, which caused the loss of information in long articles. RNNs were more context-sensitive than classical ML but had a weakness of vanishing gradients and expensive training. In addition, their interpretability remained poor, and the generated features were not easily understandable by humans.

2.1.5 The Transformer Revolution

The aim of the paper by Vaswani et al. [5] was to introduce the Transformer architecture, which became a game-changer in natural language processing (NLP). The Attention Is All You Need paper substituted the sequential processing method with an attention mechanism, enabling models to consider all word dependencies simultaneously. This architectural advancement allowed the construction of large pre-trained language models like BERT, RoBERTa, and DistilBERT, which are more effective in contextual understanding and generalisation as compared to earlier architectures.

BERT (Bidirectional Encoder Representations with Transformers) is an objective-based pre-training of bidirectional context, introduced by Devlin et al. [6], based on a masked-language modelling goal. It can also be fine-tuned on downstream tasks, such as sentiment analysis, question answering, and fake news detection, with little data due to its transfer learning ability. Transformer models are now able to reach state-of-the-art accuracy in misinformation research because they can learn both contextual information, sarcasm, and discourse-level relations.

Nevertheless, a new problem arises with these models: despite making perfect predictions, they fail to provide insight into why certain content is deemed fake. This difficulty led to the development of explainable AI studies in the field of NLP.

2.1.6 Explainable Artificial Intelligence (XAI) in NLP

Interpretability became one of the key research frontiers as the complexity of the models increased. In 2016, Ribeiro et al. proposed LIME (Local Interpretable Model-agnostic Explanations) [7], which can be used to explain predictions of each instance by fitting simple surrogate models about the decision boundary. LIME introduces perturbation to input features and quantifies their effect on prediction, which gives intuitive visualisations of word-importance.

Shortly, Lundberg and Lee suggested SHAP (SHapley Additive exPlanations) [8], which incorporates the merits of game theory and the ability to interpret the model. SHAP calculates Shapley values, which represent the average contribution of each feature to any model prediction. In contrast to LIME, which is locally interpretable, SHAP is globally consistent, meaning that the importance of features would be additive and comparable across cases.

The two approaches have been applied to areas such as healthcare, finance, and sentiment analysis, although integrating them with transformer-based fake news detectors is relatively unexplored. The recent surveys [4], [9] emphasise that using SHAP and LIME on text-based, deep models requires optimisation to control computational costs, yet they are indispensable for credible AI and model accountability.

2.1.7 Hybrid and Integrated Approaches

Recent studies have attempted to integrate transformer-based NLP models with explainability systems to develop end-to-end misinformation detection systems.

It is also important to note that Rodrigues [3] emphasises the inseparability of media literacy and automated recognition, suggesting that effective systems should not only categorise misinformation but also provide users with evidence-based statements. The combination of SHAP and LIME with transformers offers a two-fold benefit: the accuracy of deep learning and the transparency of XAI.

Alalawi et al. [1] propose a semi-supervised misinformation detection pipeline that integrates machine learning and user feedback loops, although it does not provide real-time explanations. On the same note, a detailed study of the tendency of fake news to spread was performed by Khan et al. [2], who also elaborated on the necessity of a context-dependent explainable system that will lead to user trust.

Although interest has increased, current hybrid implementations remain experimental, lacking the user-focused interface and optimisation required for deployment. SHAP explanations have a comparatively high computational intensity, and the transient nature of LIME means that it cannot scale to a real-world problem in the short term, an issue the current project tries to mitigate by optimising both in a Google Colab/Kaggle-based environment and displaying the output in a simple React web app.

Table 2.1: Literature of comparison

Author(s) & Year	Methodology	Dataset Used	Strengths	Limitations / Gap
Devlin et al., 2019 [6]	BERT (Transformer NLP)	GLUE, Wikipedia	State-of-the-art NLP accuracy	Black-box, lacks interpretability
Ribeiro et al., 2016[7]	LIME	Multiple classifiers	Provides local interpretability	Not applied directly to fake news
Lundberg & Lee, 2017 [8]	SHAP	Tree & Neural Models	Theoretically sound explanations	Computationally expensive

Shu et al., 2017 [9]	Survey of Fake News Detection	Social Media	Comprehensive overview	No practical deployment with XAI

2.1.8 Comparative Analysis

According to the literature reviewed, it is possible to make several observations:

1. Industrial initiatives are closed source and lack transparency, focusing on scaling but undermining user confidence.
2. Traditional ML models rely on high-level lexical representations and do not perform well in cross-domain settings.
3. Deep learning models enhance the contextual knowledge yet pose a problem of interpretability.
4. Transformer-based models (BERT) offer unsurpassed performance but act as a black box.
5. XAI techniques (LIME and SHAP) are interpretable and have little to no presence in transformer-based systems at scale.

The present comparative analysis highlights the trend in the research, which is shifting towards transparency-based, fairness-based, and accountability-focused trust-centric AI.

2.2 Research Gap and Motivation

Even though the fake news detection algorithms are mature, there is an apparent technological and functional disparity between a model and its interpretability.

1. Industrial systems consider real-time scalability and moderation control priorities without elaborating on the reason as to why content is flagged.
2. Academic solutions are concerned with interpretability, but tend to overlook usability, latency, and deployability.
3. Transformer-based models are highly accurate but opaque, raising ethical issues of bias, fairness, and accountability to the user.

Thus, there is an outstanding research gap in an integrated framework that:

- Classify text with the contextual knowledge of BERT,
- Improves LIME to explain things in real time, and
- Visually evaluates the global feature importance using SHAP,
- In a lean, usable web-based interface that is a compromise between research prototypes and deployable tools.

The given system will solve this unmet need by integrating the benefits of deep contextual learning and explainable model analysis into a transparent and unified fake news detection pipeline. This strategy supports the recent calls for responsible and interpretable AI in open information ecosystems [3], [4], [8].

2.3 Requirements Elicitation

2.3.1 Functional Requirements

1. **News Classification:**
    - Input: News text
    - Output: Prediction (Fake or Real) with confidence score
2. **Explainability:**
    - Interactive visualization of LIME highlights (words contributing to prediction)
    - Optional SHAP explanation generation
3. **Hybrid Dataset Training:**
    - Ability to train model on multiple datasets
    - Store trained model for evaluation
4. **Evaluation Metrics:**
    - Accuracy, Precision, Recall, F1-score on test datasets
    - Display graphs for better understanding
5. **Dashboard & Visualization:**
    - Home screen, Analyze screen, Results screen
    - Dynamic charts for performance and confidence
    - Buttons for generating explanations

2.3.2 Non-Functional Requirements

1. **Performance:**
    - Fast model inference (<2 seconds per news article)
    - Efficient SHAP/LIME generation with user-triggered control
2. **Usability:**

- Intuitive UI/UX
- Clear highlighting of words/features

3. **Reliability:**
   - Model predictions consistent across similar inputs
   - Handle invalid or incomplete inputs gracefully
4. **Scalability:**
   - Can add new datasets without major system redesign
   - Can integrate more explainability methods in the future

2.3.3 Requirements Traceability Matrix

Table 2.2: Traceability Matrix

Requirement ID	Description	Source	Test Case
FR1	Classify news as Fake or Real	Literature & Scope	Provide sample text and verify prediction
FR2	Show LIME highlights	Explainability requirement	Click "Generate LIME" and check highlighted words
FR3	Generate SHAP values	Optional XAI	Click "Generate SHAP" and confirm chart displays
FR4	Train on hybrid datasets	Methodology	Train on LIAR + Kaggle and verify test accuracy
NFR1	Fast response	Usability & Performance	Measure inference time <2s
NFR2	Clear UI	Usability	User feedback on interface clarity

2.4 Use Case Descriptions

2.4.1 Use Case 1: Analyze News

Actor: User
Description: User inputs news text, system returns prediction with confidence and LIME highlights.
Steps:

1. User enters or pastes news text in Analyze screen
2. User clicks **Analyze**
3. System shows prediction, confidence score, and LIME highlights
4. Optional: User clicks **Generate SHAP** to see feature contribution chart

Alternate Flow:

- Invalid input → System prompts user to enter valid news text

2.4.2 Use Case 2: Evaluate Model Performance

Actor: Developer/Researcher
Description: Evaluate trained model using test datasets.
Steps:

1. Select dataset for evaluation
2. Run evaluation
3. Display metrics (accuracy, F1-score) with graphs and explanations

2.4.3 Use Case 3: Visualize Explanations

Actor: User
Description: Generate visual explanations for model predictions.
Steps:

1. Click **Generate LIME** → Words highlighted in text
2. Optional: Click **Generate SHAP** → Feature contribution chart shown
3. Return to previous explanation view via **LIME** button

Chapter 3: Proposed Methodology and System design

3.1 Overview

The chapter explains the methodology framework and the actual techniques that were used in the design of the proposed fake-news detector. The goal is to generate a usable and repeatable solution that identifies deceptive news materials with a high degree of accuracy with a finely-tuned transformer and render human-understandable explanations of each prediction to incorporate builtin XAI solutions (SHAP and LIME).

The system is developed as a web application in the form of three modules:

- Frontend (React.js) – User interface for text submission, prediction, and explanation visualization.
- Backend (Flask + Python) – Implements `/api/v1/predict` and `/api/v1/explain/shap` endpoints for model inference and explainability.
- Database (PostgreSQL) – Stores datasets, predictions, explanations, and metrics for persistence.

The whole process is based on an iterative process inspired by agile and has six sprints:

dataset preparation → model fine-tuning → LIME integration → SHAP module → frontend interface → testing and deployment.

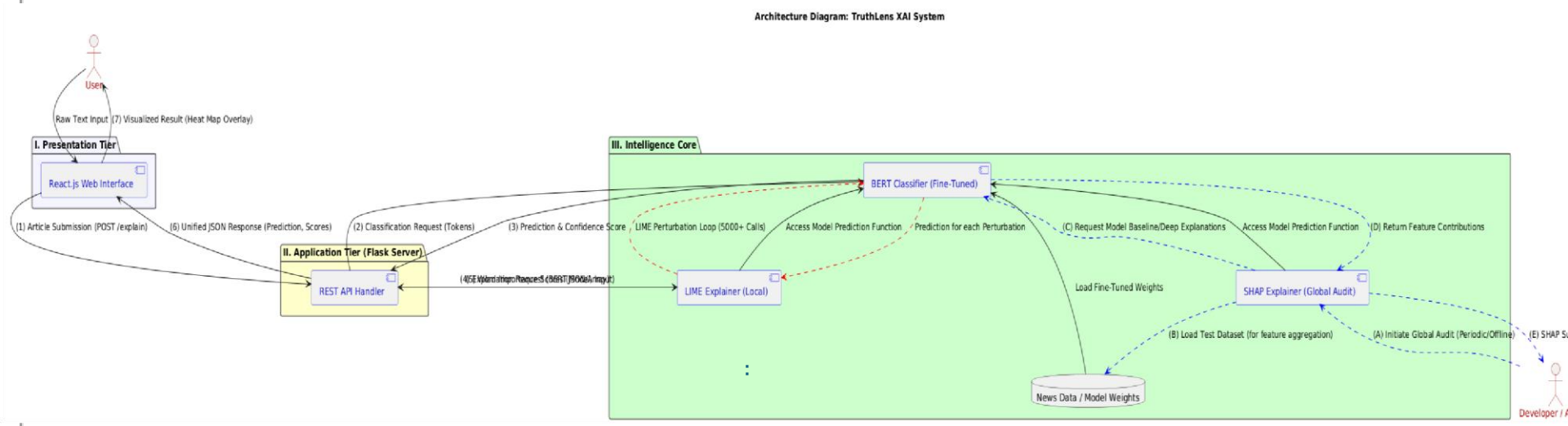


Figure 3.1: ***System Architecture***

3.2 Datasets

Three benchmark datasets were used so that to guarantee that the model can be generalized to different writing styles and contexts:

Table 3.1: Dataset

Dataset	Description	Size	Type	Label Scheme
LIAR	Short political statements with factchecked labels.	12.8k	Text	Fake = 0, Real = 1
Kaggle Fake News	Full news articles with binary labels.	40k	Text	Fake = 0, Real = 1
FakeNewsNet	Verified web and social media news samples.	22k	Text	Fake = 0, Real = 1

After the merging was done, the duplicates were eliminated and the data were divided into training (70%), validation (15%) and testing (15%).

Both datasets provide novel linguistic diversity, which enables the model to reflect the patterns in both the factual claims and long-form journalism.

The combination of these three datasets gives both short and long text distributions and decreases overfitting to any one source style.

3.3 Data Preprocessing

Preprocessing is intended to convert raw texts to normalized sequences of tokens that can be input to BERT without loss of semantic information for classification and description. Preprocessing operations are deterministic and are used in the same way during training and inference.

3.3.1 Steps and tools

The following pipeline is applied to every input text:

1. **Unicode normalization and HTML stripping**: remove HTML tags and non-printable characters.

2. **URL and email removal**: replace URLs and email addresses with a single token <URL> or remove, depending on policy.
3. **Punctuation and numeric handling**: remove extraneous punctuation while preserving intra-word hyphens; numbers are optionally normalized.
4. **Lowercasing**: convert all characters to lower case for bert-base-uncased consistency.
5. **Tokenization**: using BertTokenizer from Hugging Face; retains subword tokens (WordPiece).
6. **Stopword removal / selective filtering**: optional for SHAP/LIME clarity; core model training uses raw tokenization rather than aggressive stopword removal to preserve context in BERT.
7. **Lemmatization (analysis step)**: done for diagnostic datasets and for constructing SHAP background samples; not applied to final sequence tokens because subword tokenization and contextual embeddings reduce the need for explicit lemmatization.
8. **Padding/Truncation**: sequences padded/truncated to fixed length $L = 256$ tokens.

*Table 3.2: **Preprocessing operations***

Operation	**Implementation detail**	**Purpose**
HTML strip	regex re.sub(r'<[^>]+>', '', text)	Remove tags
URL removal	`re.sub(r'https?://\S+	www.\S+', '', text)`
Email removal	re.sub(r'\S+@\S+', '', text)	Remove personal info
Punctuation	re.sub(r'[^\w\s\-]', '', text)	Keep hyphenated words
Lowercase	text.lower()	Tokenizer consistency
Tokenize	BertTokenizer.from_pretrained('bert-baseuncased')	Subword tokens
Pad/Truncate	max_len = 256	Fixed input length

3.3.2 Token length normalization (example)

To standardize sequence length, each tokenized sequence s is either truncated or padded to length L.

Padding uses the special [PAD] token to extend sequences shorter than L. This normalization ensures batch tensors of shape (B,L) where B is batch size.

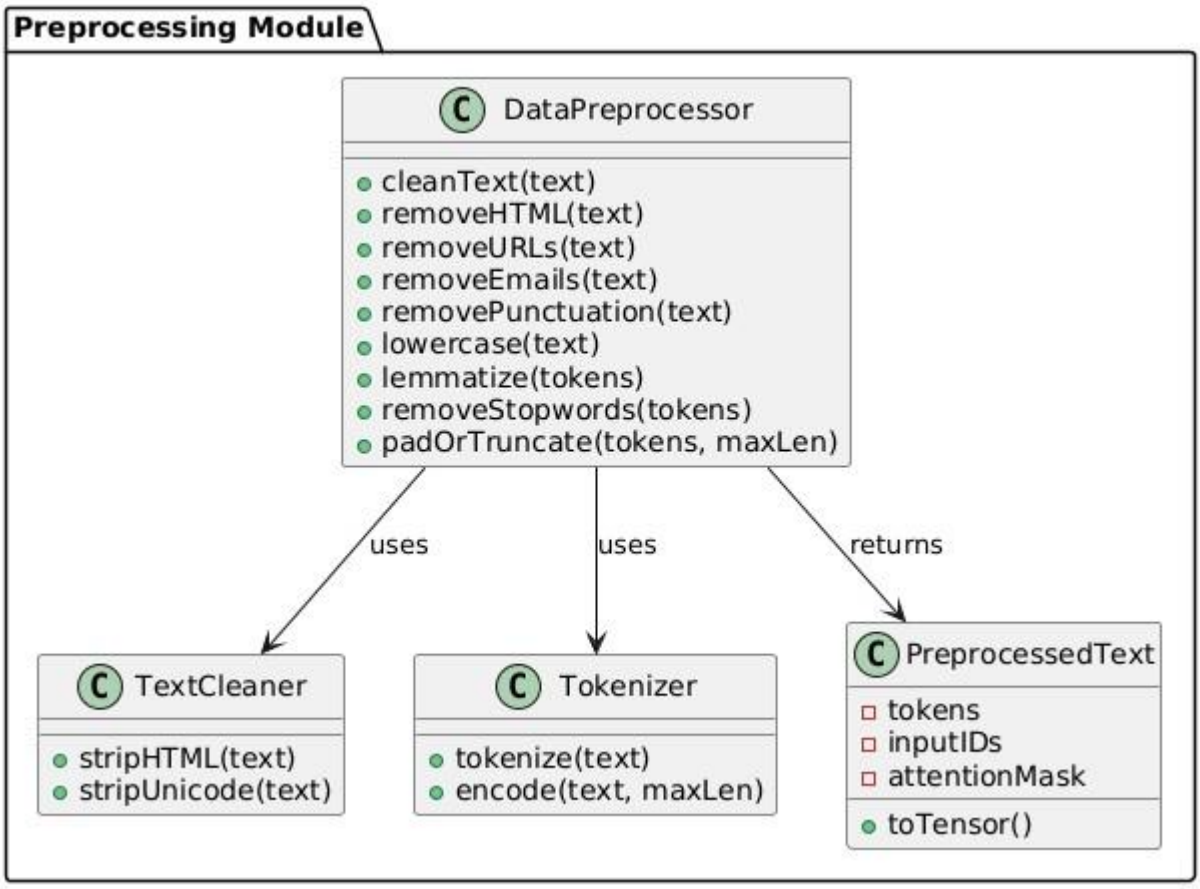


Figure 3.2: Preprocessing module52740

3.4 Model Architecture

The classification backbone is the BERT transformer. The following subsections present the architectural components and the training configuration.

3.4.1 Self-attention mechanism

The core of the transformer encoder is the scaled dot-product attention. For queries QQQ, keys KKK, and values VVV, the attention is computed as:

$$\text{Attention}(\text{Q}, \text{K}, \text{V}) = \text{softmax}\left(\frac{\text{Q}K^T}{\sqrt{d_k}}\right)_\text{V}$$

where dkd_kdk is the dimensionality of the key vectors. Multi-head attention concatenates several attention outputs and applies a learned projection.

3.4.2 Encoder and classification

BERT is a stack of N=12N=12N=12 encoder layers (for bert-base). Each encoder layer applies multi-head self-attention followed by position-wise feed-forward sublayers. The output corresponding to the classification token [CLS] is used for downstream binary classification. The classification head consists of a dropout followed by a fully connected layer and a sigmoid activation to produce y^∈(0,1)\hat{y} \in (0,1)y^∈(0,1).

3.4.3 Loss function and optimization

Training minimizes binary cross-entropy loss with logits (numerically stable form). For a single example with true label y∈{0,1}y \in \{0,1\}y∈{0,1} and model logit zzz, the binary cross-entropy loss is:

$$L(y,z) = -(y \cdot \log(\sigma(z)) + (1-y) \cdot \log(1-\sigma(z)))$$

The optimizer used is AdamW (weight-decay variant) with learning rate α=2×10−5\alpha = 2 \times 10^{-5}α=2×10−5. Training hyperparameters are fixed for reproducibility:

*Table 3.3: **Hyperparameters***

Parameter	**Value**
Batch size	16
Epochs	3
Learning rate	2×10−52\times 10^{-5}2×10−5
Optimizer	AdamW
Max sequence length	256
Dropout	0.1

Random seed	42

Rationale: BERT fine-tuning typically converges in a small number of epochs (1–5); selecting 3 epochs balances convergence and compute cost on free GPU resources.

Model Training Activity:

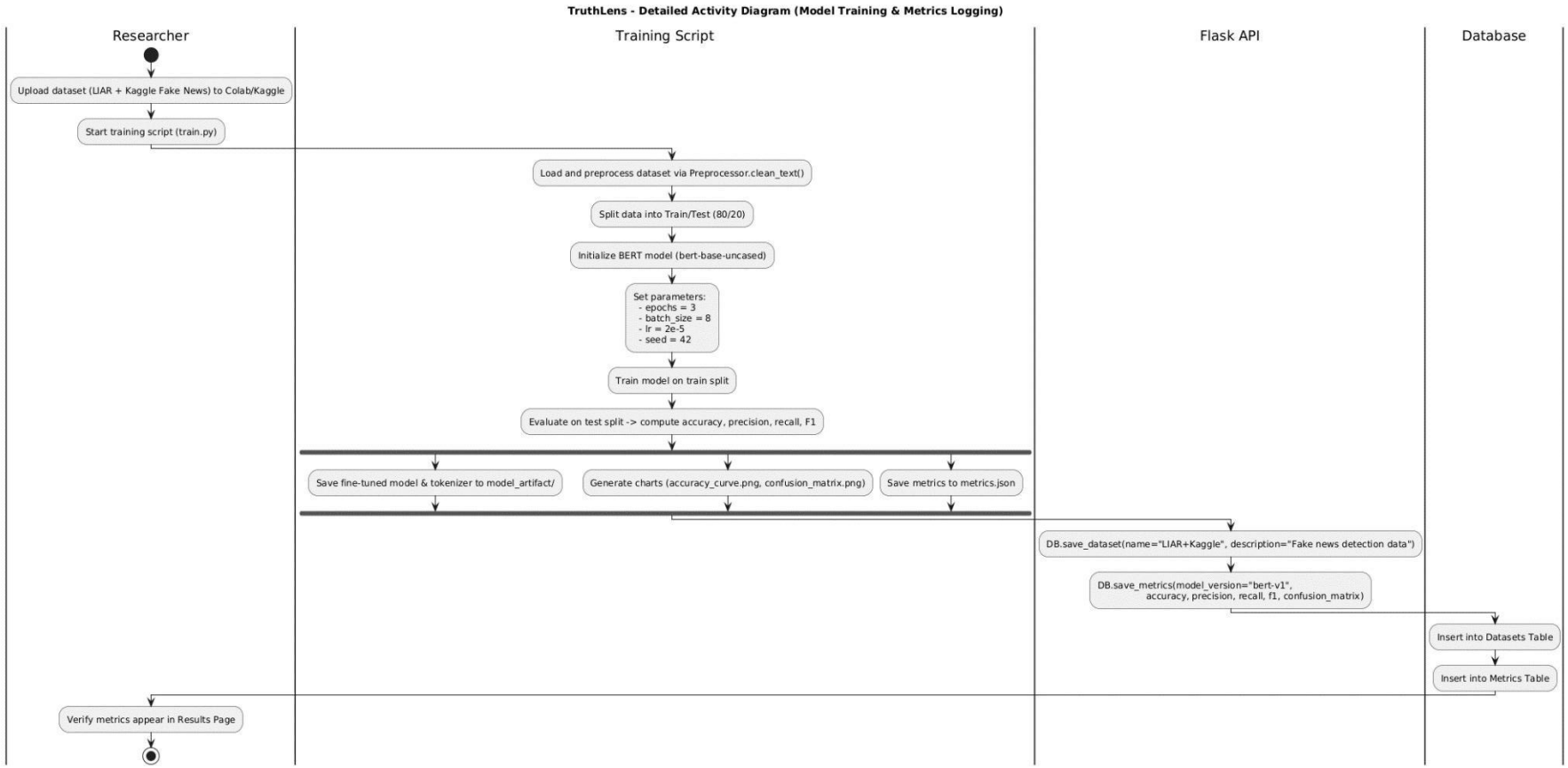


Figure 3.3: Model Training

3.4.4 Backend Class Diagram

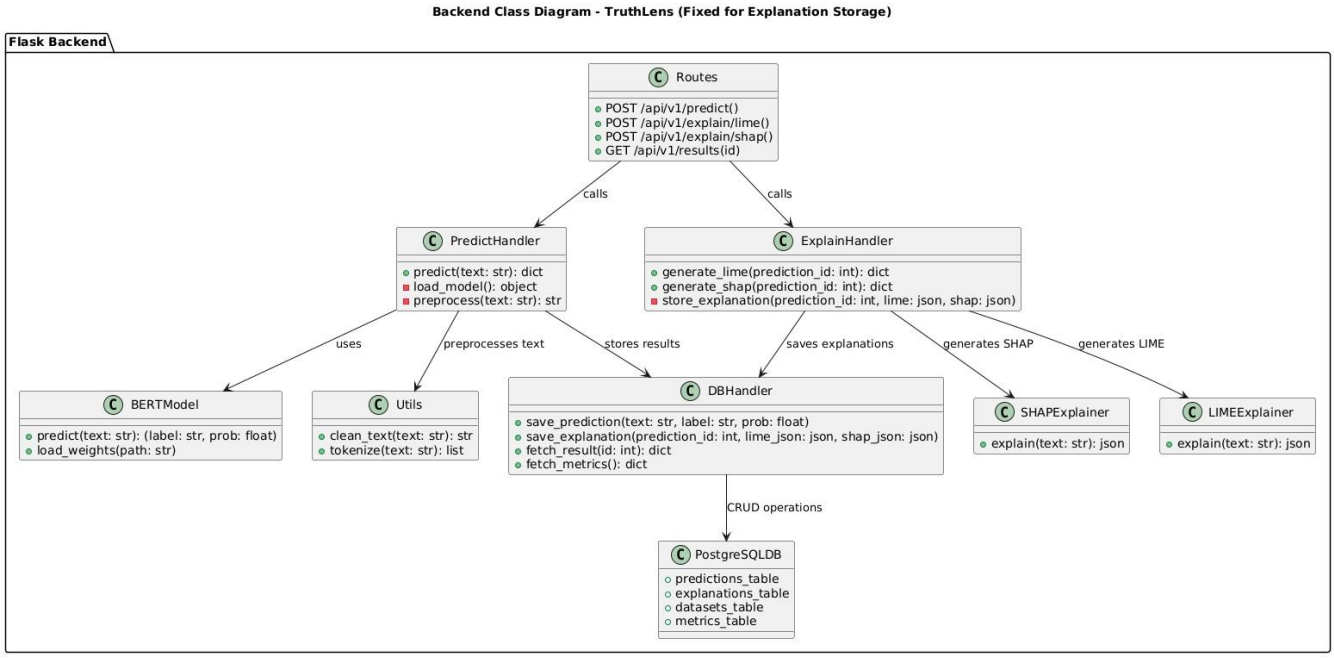


Figure 3.4: Backend Class Diagram

3.5 Explainability Techniques

Explainability is provided by the combined use of LIME and SHAP. Their roles are explicit and separated in the system: **LIME** provides immediate local explanations per prediction and is computed synchronously within the prediction API; **SHAP** is computed on demand (user clicks the Explain button) and uses a background dataset for baseline estimation.

3.5.1 LIME-local linear surrogate

LIME approximates the complex model f locally by fitting a weighted linear model g over perturbed samples. For an input instance x, LIME minimizes:

$$\{\boldsymbol{L}\}(\boldsymbol{f},\boldsymbol{g},\boldsymbol{\pi_x}) + \boldsymbol{\Omega}(\boldsymbol{g})$$

where π_x weights perturbations by similarity to x, $\{L\}((\boldsymbol{f}, \boldsymbol{g}, \boldsymbol{\pi_x})$)L is loss (e.g., squared loss between f and g outputs), and Ω(g))is a complexity penalty on the surrogate model g . In practice, LIME perturbs text by removing or masking tokens and fits a sparse linear model whose coefficients indicate token importance.

3.5.2 SHAP- Shapley value explanations

SHAP attributes to each feature (token) is a Shapley value ϕ_i, reflecting the average marginal contribution of that feature across all coalitions:

$$\phi_i = \sum_{(S \subseteq N) \setminus \{i\}} \frac{\{|S|!\,(|N| - |S| - 1)!\}}{\{|N|!\}} \left(f_{\{S \cup \{i\}\}}(x_{\{S \cup \{i\}\}}) - f_{S(x_S)} \right)$$

where N is the set of features (tokens), S is a subset not containing i, and$(f_S)^3$denotes model output when features in S are present and others are set to a baseline. Exact computation is intractable for text; therefore, we approximate using model-specific SHAP algorithms for deep models (e.g., DeepExplainer or sampling-based estimators).

3.5.3 Integration rules and caching

- **LIME**: use num_features=10 and num_samples=500 as defaults for speed/quality tradeoff. Output is top-k token importance list (token, weight). LIME runs synchronously and is returned with /api/v1/predict.
- **SHAP**: background set precomputed (100 cleaned texts), max_evals limited (100–200) to control runtime, and results cached keyed by SHA256 of cleaned text to avoid repeated expensive computations.
- **Visualization**: both LIME and SHAP outputs are converted to token-level color overlays (RGB mapped to weight sign and magnitude) and rendered by HighlightedText in React.

SHAP Explanation Activity:

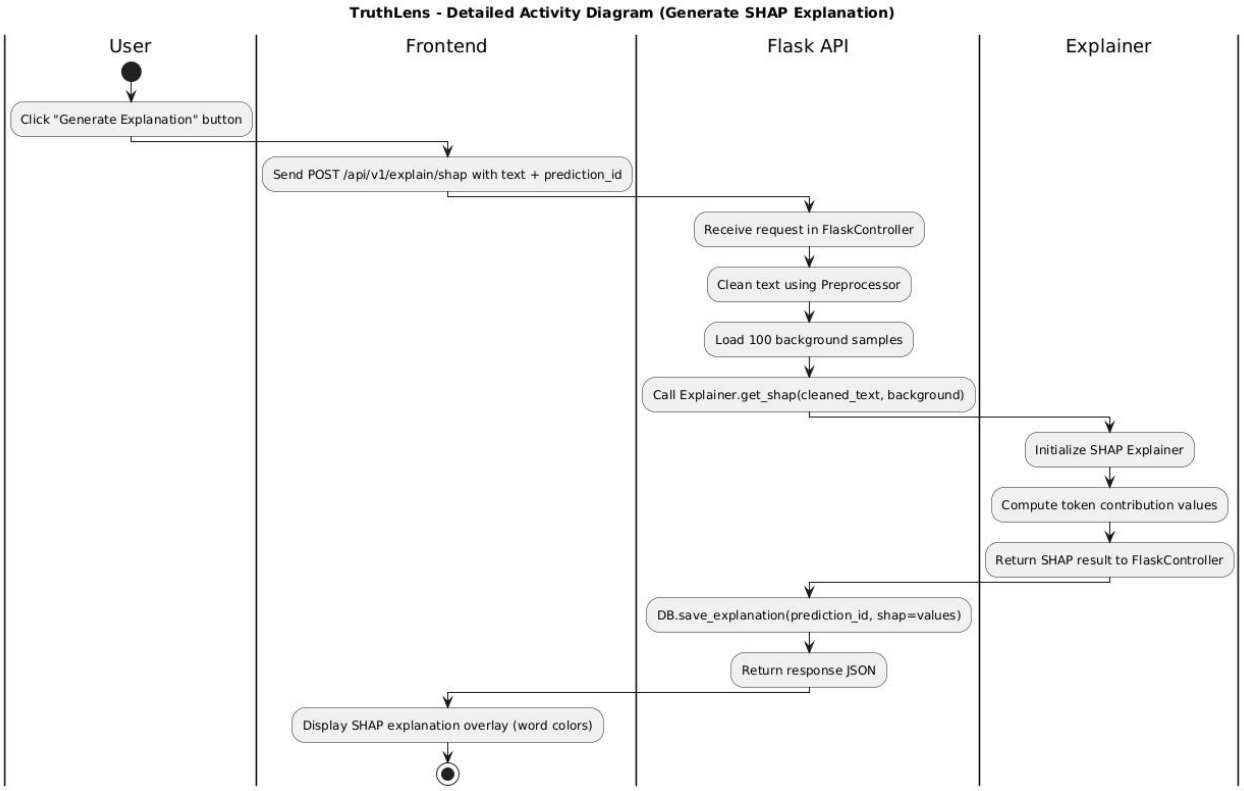


Figure 3.5: Shap Explanation Generation

DFD Level-1:

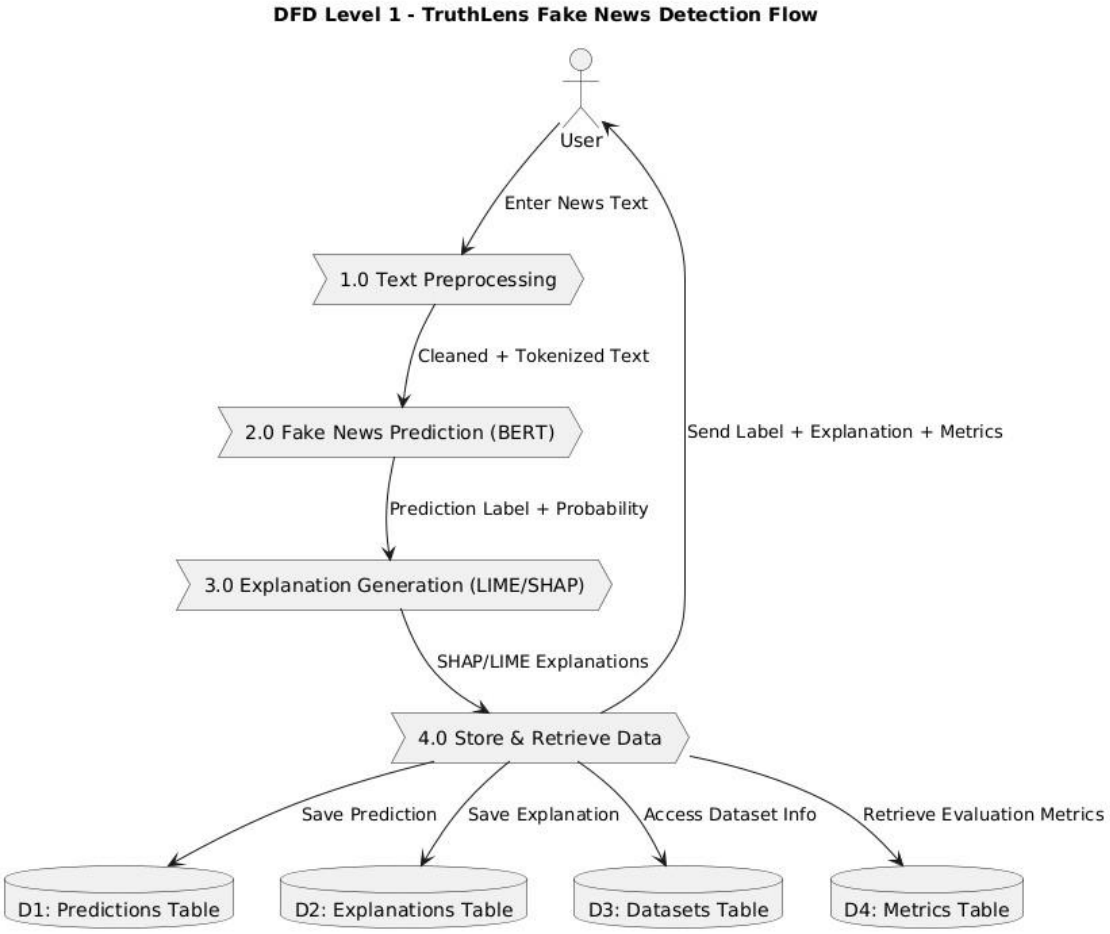


Figure 3.6: DFD level-1

3.6 System Architecture and Component Interaction

The system is realized as a modular, service-oriented web application. Components and their responsibilities are:

- **React Frontend**: user interface (Detect, Explain, Results, Dataset pages); posts input to backend; consumes JSON responses; renders LIME/SHAP overlays and static evaluation charts.
- **Flask Backend**: REST API endpoints:
  - POST /api/v1/predict — accepts raw text, runs Preprocessor.clean_text, tokenizes, calls ModelServer.predict_proba, computes LIME, persists prediction and LIME in DB, returns label, probabilities, LIME. o POST /api/v1/explain/shap — accepts text or prediction_id, runs cleaning, loads background, runs Explainer.get_shap, persists SHAP JSON, returns note or full SHAP JSON.
  - GET /api/v1/results/{id} — returns stored prediction + explanation.
- **Model Server**: loads BERT artifacts lazily; predict_proba returns class probabilities in canonical order [p_real, p_fake].
- **Postgre Database**: tables datasets, metrics, predictions, explanations (ERD is consistent with earlier design).
- **Training Environment**: Google Colab / Kaggle for fine-tuning, producing model_artifact/ and metrics.json.

3.6.1 Data flows

1. User submits text via React → frontend posts to /api/v1/predict.
2. Flask receives request → Preprocessor.clean_text → tokenization.
3. ModelServer.predict_proba returns probabilities; label decided by argmax.
4. Explainer.get_lime computed and saved along with prediction via DB.save_prediction and DB.save_explanation.
5. Frontend displays label, probability gauge, and LIME overlay.
6. When user requests deeper explanation, frontend calls /api/v1/explain/shap → Flask coordinates SHAP computation and DB update → frontend fetches and displays SHAP overlay.

This pipeline is designed to maintain interactive responsiveness while providing a heavycomputation path for deep explanations.

Component Diagram:

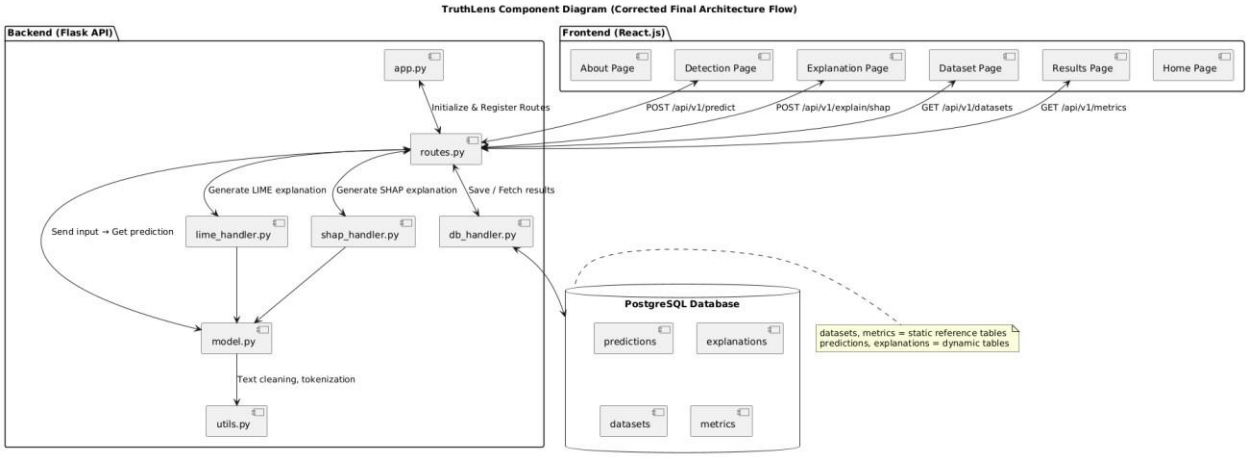


Figure 3.7: Component Diagram

Frontend_Class:

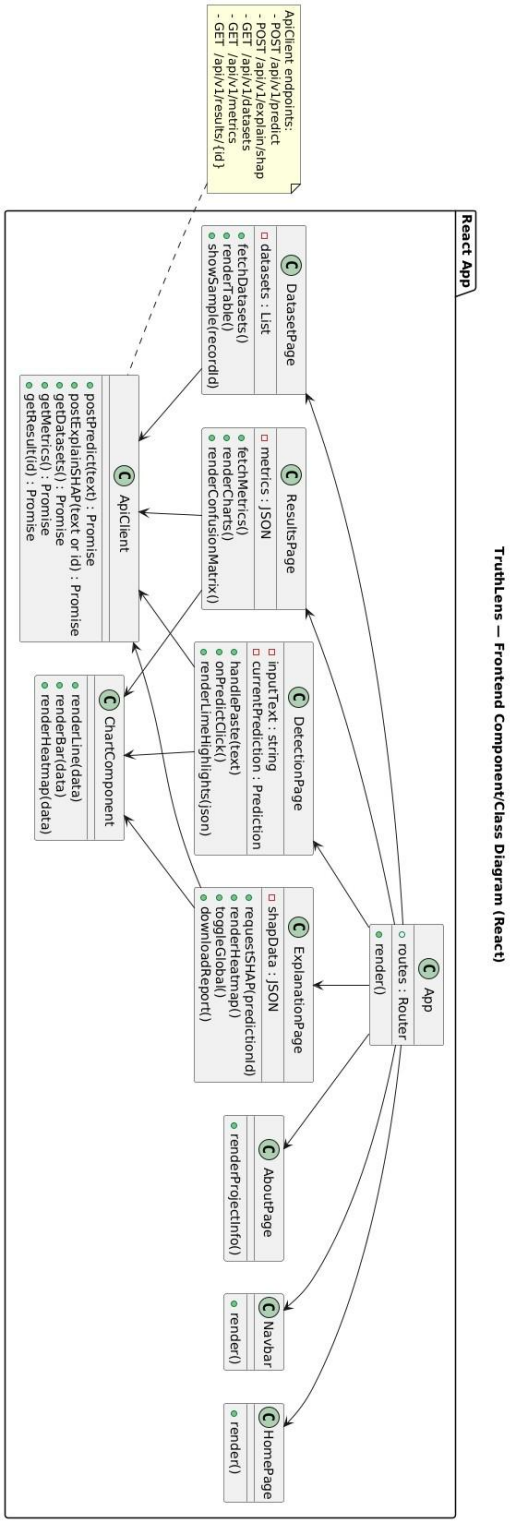


Figure 3.8: Frontend Class diagram

3.7 Flow of Methodology (textual synthesis)

The operational methodology can be summarized as:

1. **Data consolidation**: collect LIAR + Kaggle + FakeNewsNet; merge, deduplicate, and harmonize labels.
2. **Preprocessing**: deterministic cleaning and tokenization with `BertTokenizer`, standardizing sequence length to 256 tokens.
3. **Model training**: fine-tune `bert-base-uncased` (batch size 16, epochs 3, lr 2e-5, AdamW) on the combined dataset with stratified splits.
4. **Explainability**: LIME for instant local explanations; SHAP for rigorous on-demand global instance explanations using a 100-sample background set.
5. **Deployment**: React + Flask + Postgre deployment; training artifacts stored and deployed to model server.
6. **Iteration**: agile sprints refine preprocessing, hyperparameters, and UI based on supervisor reviews.

Fake News Prediction:

et

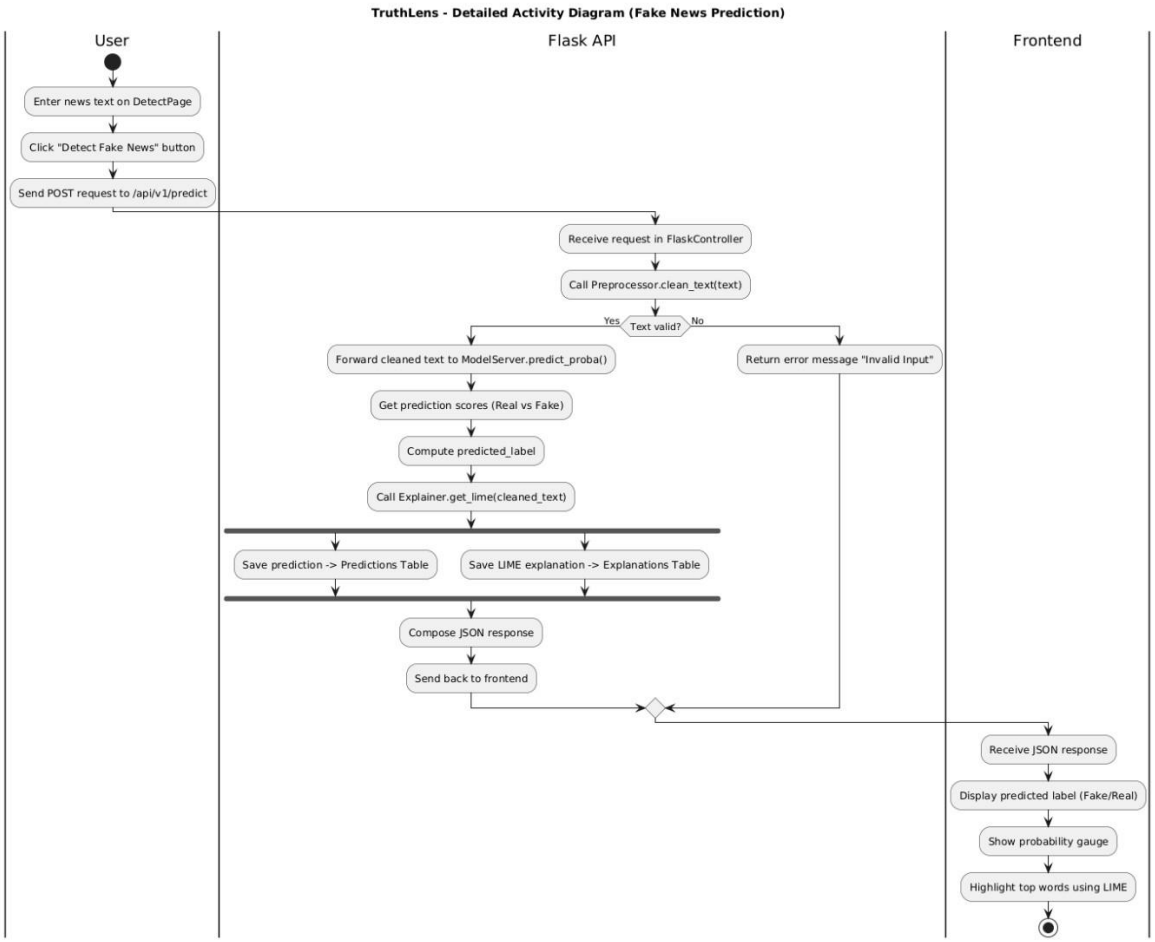


Figure 3.9: Fake news prediction

Metrics Evaluation and Result Display:

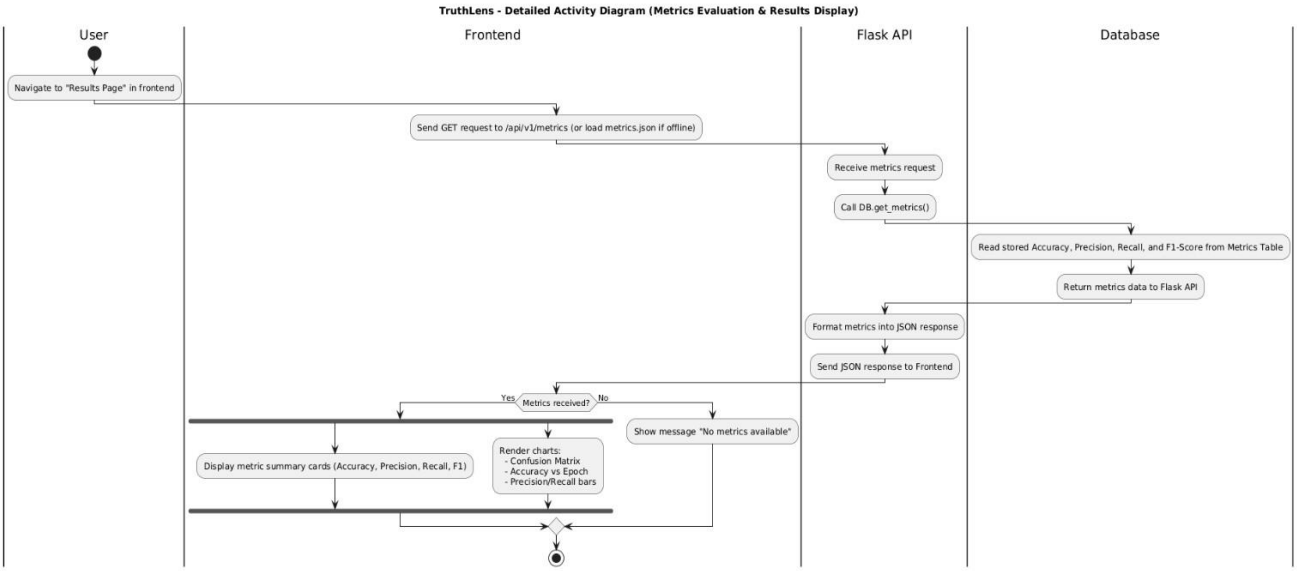


Figure 3.10: Metrics Evaluation and Result Display

Sequence Diagram:

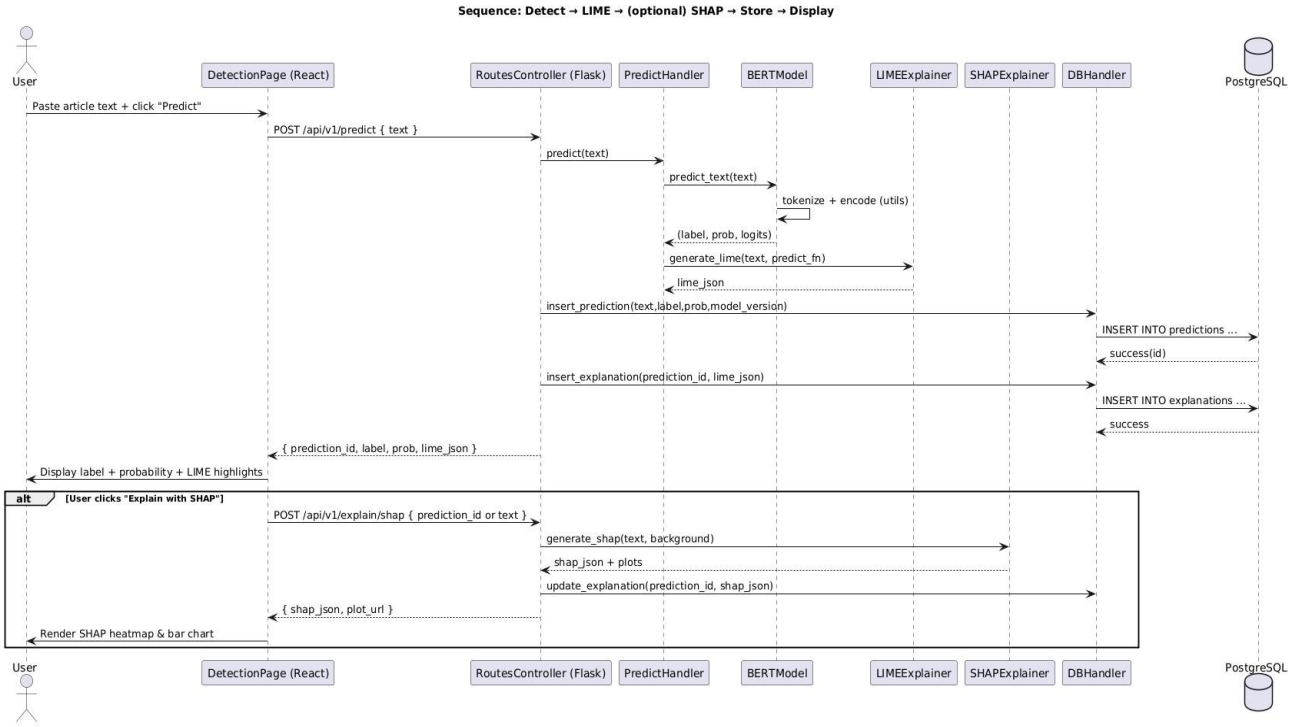


Figure 3.11: Sequence Diagram：

3.8 Algorithm / Model Selection

3.8.1 Problem Nature

Fake news detection is a **binary text classification problem** that requires:

- Context understanding
- Semantic relationships
- Long-range dependencies

Traditional ML algorithms (Naive Bayes, SVM) struggle with:

- Sarcasm
- Context shift
- Complex sentence structures

3.8.2 Why Transformer-Based Models?

Transformer models (BERT) were selected because they:

- Capture **context bidirectionally**
- Understand word meaning based on surrounding words
- Handle long and short text effectively
- Are state-of-the-art for NLP classification

This directly aligns with research trends.

3.8.3 Why BERT Specifically?

BERT (Bidirectional Encoder Representations from Transformers):

- Uses **self-attention**
- Learns deep contextual representations
- Performs well even with limited labeled data
- Can be fine-tuned efficiently

Compared to Alternatives

Model	Limitation
LSTM	Sequential, slow, context loss
CNN	Limited semantic depth
GPT-style	Overkill, harder to control
BERT	Balanced, explainable, efficient

3.8.4 Fine-Tuning Strategy

Instead of training from scratch:

- A **pretrained BERT** model is fine-tuned
- Classification head is added
- Trained on combined datasets

Benefits:

- Faster convergence
- Better accuracy
- Lower data requirement

3.8.5 Explainability Compatibility

BERT is chosen **not just for accuracy**, but because:

- It is compatible with **LIME**
- It supports **SHAP**
- It allows **token-level contribution analysis**

This aligns with the project's core goal:

Transparency over blind prediction

3.8.6 Why Not Real-Time Fact Checking?

This is **intentional**, not a weakness.

The model:

- Detects **linguistic deception patterns**
- Does not verify factual correctness
- Works without external databases

This ensures:

- Offline usability
- Research control
- Explainability clarity

3.8.7 Summary of Model Selection Justification

The selected model:

- Solves the defined problem
- Matches dataset characteristics
- Supports explainability
- Aligns with project scope
- Avoids unnecessary complexity

Chapter 4: Implementation

4.1 Work Breakdown Structure (WBS)

Table 4.1: Work Breakdown Structure

WBS ID	Phase	Task / Activity Description
1.0	Project Initiation	Requirement analysis and problem definition
1.1	Data Collection	Collection of LIAR and Kaggle Fake News datasets
1.2	Data Understanding	Dataset exploration and class distribution analysis
2.0	Environment Setup	Configuration of Python environment and libraries
2.1	Development Tools	Setup of Google Colab / Kaggle GPU runtime
2.2	Version Control	Project structuring and dependency management
3.0	Data Preprocessing	Text cleaning and normalization
3.1	Text Normalization	Lowercasing, stopword removal, lemmatization
3.2	Tokenization	BERT tokenizer encoding
4.0	Frontend Prototype	UI wireframe and layout design
4.1	Frontend Development	Implementation of news input and result display
5.0	Model Training	Fine-tuning BERT base uncased model
5.1	Training Pipeline	Batch training, validation, early stopping
5.2	Model Evaluation	Initial accuracy and loss verification

6.0	Backend API	Development of Flask RESTful APIs
6.1	Prediction Endpoint	API for text submission and classification
6.2	Explainability Trigger	API endpoints for SHAP and LIME
7.0	Model Optimization	Hyperparameter tuning
7.1	Inference Optimization	Latency reduction and batch inference
8.0	Database Integration	Design of prediction storage schema
8.1	Data Persistence	Storing results and explanation metadata
9.0	Explainability (SHAP)	Integration of SHAP explainer
9.1	SHAP Visualization	Token-level contribution heatmaps
9.2	UI for SHAP	Frontend visualization of SHAP outputs
10.0	Explainability (LIME)	Integration of LIME text explainer
10.1	LIME Perturbation	Local explanation generation
10.2	UI for LIME	Frontend display of LIME explanations
11.0	System Integration	Integration of frontend, backend, model, and database
11.1	End-to-End Flow	Verification of complete system pipeline
12.0	Testing & Debugging	Functional and integration testing
12.1	Error Handling	Bug fixing and exception management
13.0	Final Documentation	Report writing and final review

4.2 Team Roles and Responsibilities

Table 4.2: Roles and responsibilities of Team members

Team Member	Role	Responsibilities	Duration
Abdullah Liaqat	AI / ML Developer	Dataset collection, data preprocessing, BERT model training, model optimization, SHAP explainability integration, LIME explainability integration, system integration, testing and debugging, final documentation	20 September 2025 – 28 February 2026
Muhammad Sajjad	Web & Backend Developer	Environment setup, frontend prototype development, backend API development, database integration, SHAP UI implementation, LIME UI implementation, system integration, testing and debugging, final documentation	20 September 2025 – 28 February 2026

4.3 Tools and Technologies

The TruthLens system was developed based on the use of an open-source code, cloud computing, and the latest machine learning libraries in order to guarantee reproducibility, scalability, and performance when operated under limited computational resources.

4.3.1 Programming Language

- The main programming environment was Python 3.10, which has a vast range of support features concerning data processing, natural language processing, machine learning, and explainable AI frameworks.
- Python (with integration with libraries like PyTorch, Transformers, SHAP, and LIME) is compatible with other libraries, which is why it was applicable both to training models and to integration.

4.3.2 Development Platforms

- **Google Colab** and **Kaggle Notebooks** were employed as cloud-based GPU-enabled platforms.
- Benefits included:
    - Free access to NVIDIA Tesla T4 GPUs
    - Preinstalled ML libraries
    - Easy sharing and reproducibility
- Limitations, such as session timeouts and memory constraints, were mitigated through careful batch sizing, reduced epochs, and modular implementation.

4.3.3 Libraries and Frameworks

Table 4.3: Libraries and Frameworks

Component	Library / Framework	Purpose
Data Processing	NumPy, Pandas	Numerical computations, dataset manipulation
Text Preprocessing	NLTK, spaCy	Tokenization, lemmatization, stopword removal, text normalization

Component	Library / Framework	Purpose
Machine Learning	PyTorch, Hugging Face Transformers	Fine-tuning BERT base uncased model
Explainability	SHAP, LIME	Global and local model interpretability
Model Optimization	PyTorch utilities	Early stopping, learning rate scheduling, weight decay

4.3.4 Web and API Development

- **Backend Framework:** Flask

  Supported RESTful APIs where prediction requests are provided, where explanation triggers are provided and where data storage operations are provided.

- **Frontend Technologies:** React.js, HTML5,Talwind CSS , JavaScript

Facilitated an interactive and responsive user interface to input data and display the results and visualize the explanations.

4.3.5 Database Management

- The data to be stored in Postgre was chosen as a lightweight, file based relational database (Postgre):
  - Predicted labels
  - Explanation outputs
  - Metadata for further analysis
- Chosen for simplicity, portability, and integration with Flask.

4.3.6 Development Tools and Version Control

- **VS Code** was used as the main code editor for script development, debugging, and code navigation.

- **Git and jira** were employed for version control, collaborative development, and backup management.

4.3.7 Justification

All the tools and technologies have been chosen in accordance with:

- Open-source to save on cost.
- API The ability to integrate AI models, explainability models, and web applications easily.
- Scalability and reproducibility of the ultimate system.
- Resource limitations, where the system would be able to work efficiently using free GPU platform.

4.4 Implementation details

The chapter demonstrates the full application of the fake news detection system suggested in the chapter based on BERT and Explainable AI models (SHAP and LIME). The system is built upon natural language processing, machine learning and explainable modules to deliver a transparent and trustworthy prediction mechanism. The entire implementation was done in phases based on agile-inspired iterative approach as mentioned above.

4.4.1 Development Environment and Tools

The system was developed based on the freely available cloud resources, which makes it accessible and reproducible. The training and testing of the models were done in Google Colab and Kaggle notebooks, which have a GPU enabled environment.

Table 4.1: Implementation Environment and Tools

Table 4.4: Implementation

Component	Description	Tool / Version
Programming Language	Python 3.x	Python 3.10

Component	Description	Tool / Version
Development Platform	GPU-based cloud notebook	Google Colab / Kaggle
Libraries and Frameworks	Data preprocessing, NLP, and model handling	NumPy, Pandas, NLTK, spaCy, Transformers (Hugging Face)
Explainability Libraries	Explainable AI modules	SHAP 0.44, LIME 0.2.0.1
Model Architecture	Pre-trained transformer	BERT (bert-base-uncased)
Web Framework	Backend service layer	Flask
Database	Storage for predictions and explanations	SQLite (lightweight, file-based)
Frontend Technologies	Graphical user interface	HTML5, CSS3, React, JavaScript

4.4.2 System Modules Implementation

It is implemented in modules, and each component can be developed and tested separately. The system comprised four primary modules, which include Data Preprocessing, Model Training, Explainability Integration, and Frontend-Backend Communication.

A. Data Preprocessing Module

Data cleaning and normalization is done in the preprocessing module with the help of regular expressions, NLP, and spaCy in Python. It copy pastes URLs, punctuation, numbers, emojis and HTML tags off to guarantee clean and standardized input. Linguistically normalized tokens are obtained by tokenization and lemmatization of raw text.

Tclean=lowercase(remove_url(remove_punctuation(remove_html(T))))T_{clean} = \text{lowercase(remove_url(remove_punctuation(remove_html(T))))}Tclean =lowercase(remove_url(remove_punctuation(remove_html(T))))

Noise is removed by removing stopwords and the data is divided into training (70%), validation (15%), and testing (15%) parts. The processed tokens are coded to the BERT tokenizer that transforms words into the input IDs and attention masks needed by the transformer model.

B. Model Training Module

The model is trained on the combination of the Kaggle and LIAR as fine-tuned on the BERT-base-uncased. All the training instances are sent through the transformer model in tokenized form. The token output [CLS] is the semantic summary of the input sequence and is the input of the classification layer to make binary predictions.

Adam optimizer and binary cross-entropy loss are used in the fine-tuning process. Early termination is implemented about the accuracy of validation to avoid overfitting.

Table 4.5: ***Model Configuration Parameters***

Parameter	Value	Description
Learning Rate	2e-5	Optimized for BERT fine-tuning
Batch Size	16	Balance between performance and resource limits
Epochs	4	Optimal convergence point
Optimizer	AdamW	Weight-decay variant for transformers
Max Sequence Length	256	Suitable for mixed-length news articles
Dropout Rate	0.1	Prevents overfitting
Random Seed	42	Ensures reproducibility

The model is then trained and stored as a .pt or.bin file to be deployed. This is because batch prediction will support a trained model to achieve a stable inference time with low latency.

C. Explainability Integration Module

The explainability layer consists of two sub modules SHAP Explainer and LIME Explainer which perform global and local interpretability respectively.

- **SHAP Integration:**

  SHAP explainer calculates contributions of tokens to the final decision of the model. It uses the output embeddings of the transformer and displays key words as heat maps. Green color is the color of the words that positively affect the classification of Real and red is the color of the words that affect Fake. The SHAP is activated on-demand, upon clicking the Explain Prediction button by the user.

- **LIME Integration:**

  LIME comes up with explanations of an individual prediction by perturbing words within the input sentence and training a local linear model to approximate the decision boundary BERT. It generates a weighted list of words adding up to positive or negative contributions to the classification. This module is run immediately after prediction, to give instant interpretability feedback.

D. Frontend and Backend Implementation

The back-end is written in Flask, which supports the RESTful APIs of submitting the data and inference of the model. Upon the submission of the news article by the user, the API will invoke preprocessing, prediction and explanation pipelines in that order. The results of prediction and explanation are saved in the SQLite database to be retrieved and analyzed in the future.

The frontend is implemented with the help of HTML5, CSS3, React, to ensure the responsive and professional look. It contains the following interfaces:

- **Home Page:** A description of the purpose and functionality of the system.
- **Analyze screen:** shows the summary statistics of the Kaggle and LIAR datasets.
- **Lime explanation page:** Displays classification results and evaluation measures.
- **SHAP explanation Page**: Interactsively with SHAP and LIME.

4.5 Screenshots of Prototype

Truthlens is primarily a research driven and AI based system therefore frontend is kept limited as a functional layer to demonstrate model interaction, explanation generation , describing model evaluation metrics and giving insight about project, following screens present end to end prototype

1: **Home Page**

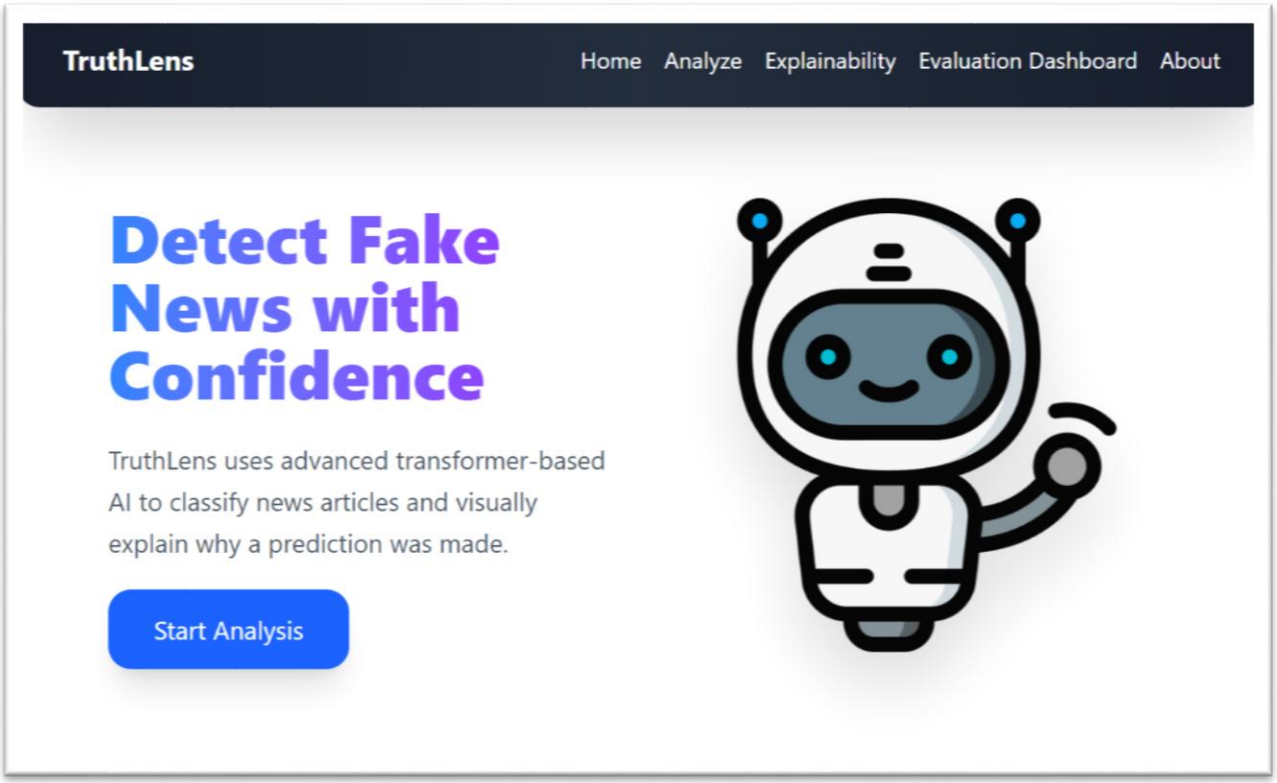


Figure 4.1: Home Screen

2: **Analyse Screen:**

User would paste the subjective article and get the result on same page with the confidence level description

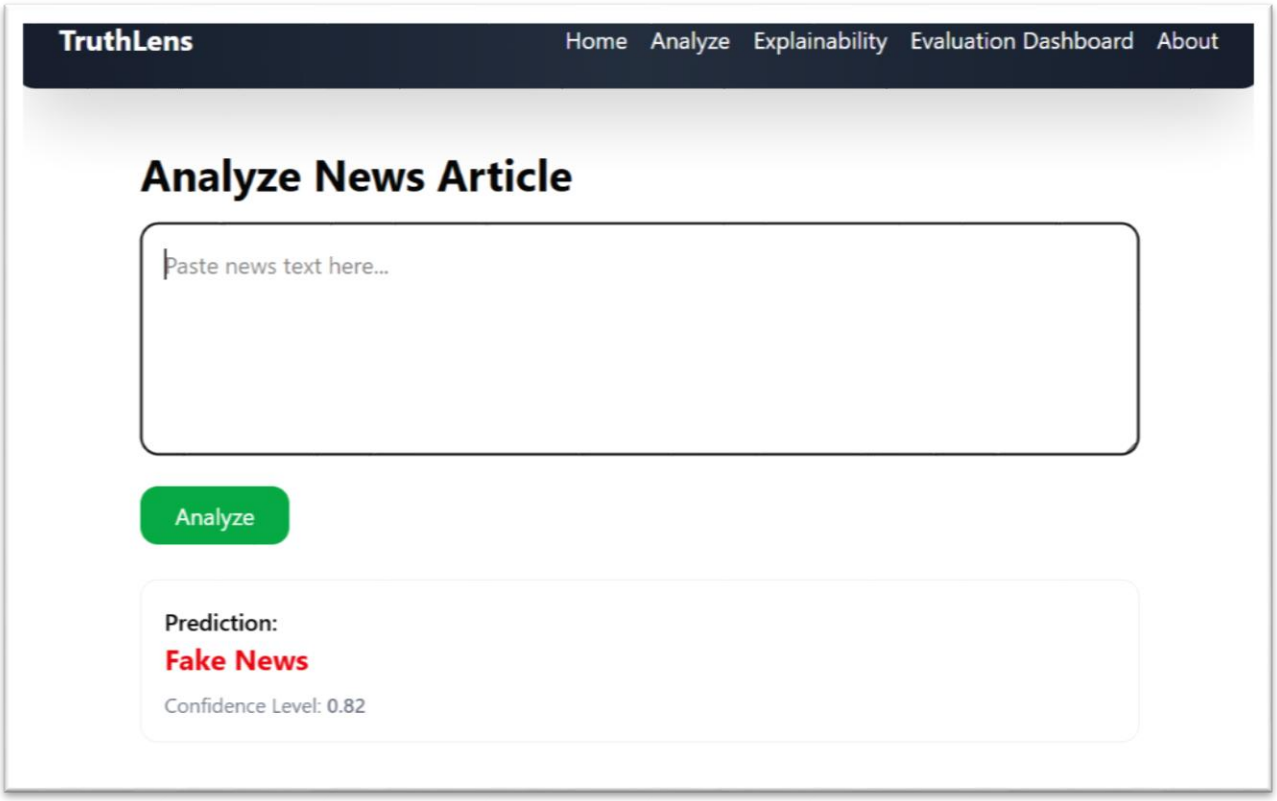


Figure 4.2: Analyze Screen

3: Lime Screen:

When the user will navigate to explainability in nav bar of home page lime explanation would come in front with the option of shap generation in the bottom

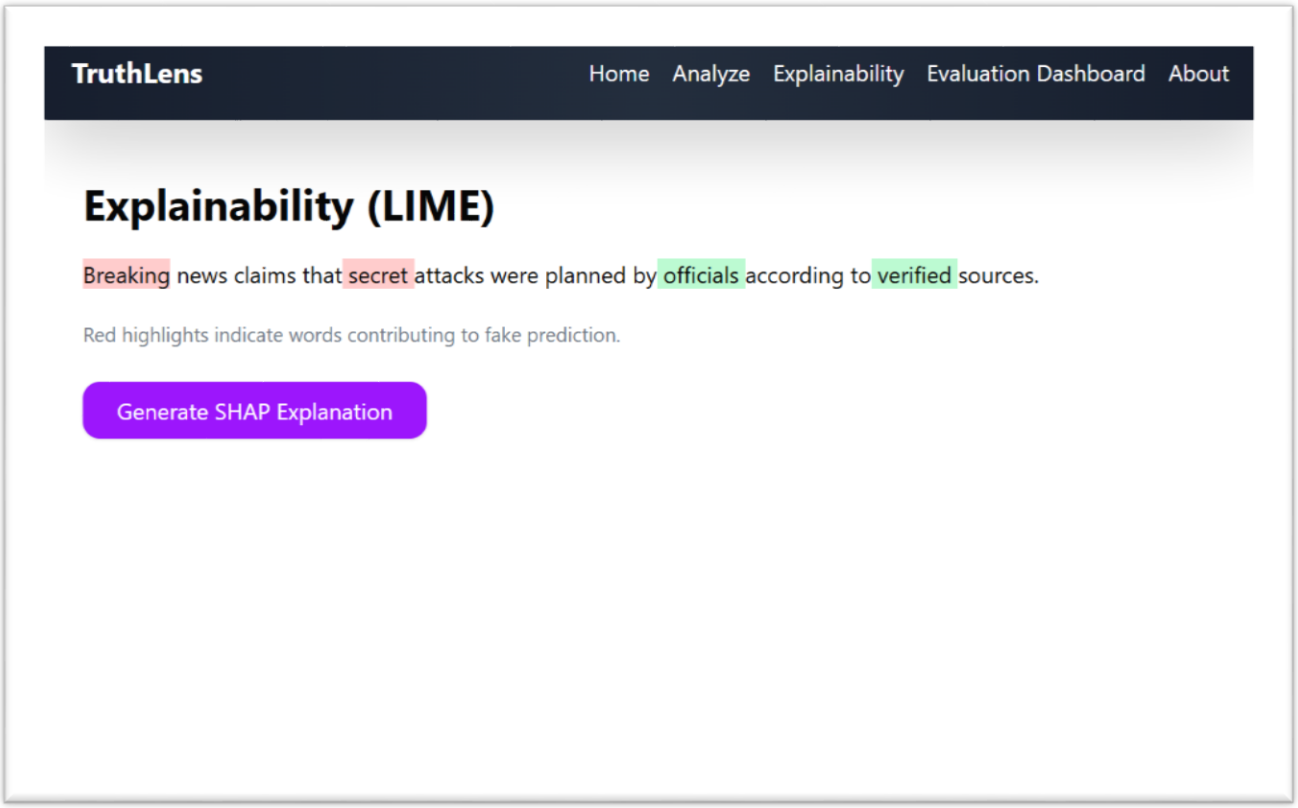


Figure 4.3: Lime Screen

4: Shap Screen:

This screen would show detailed SHAP value and explanation and will consist of a button to return to lime explanation

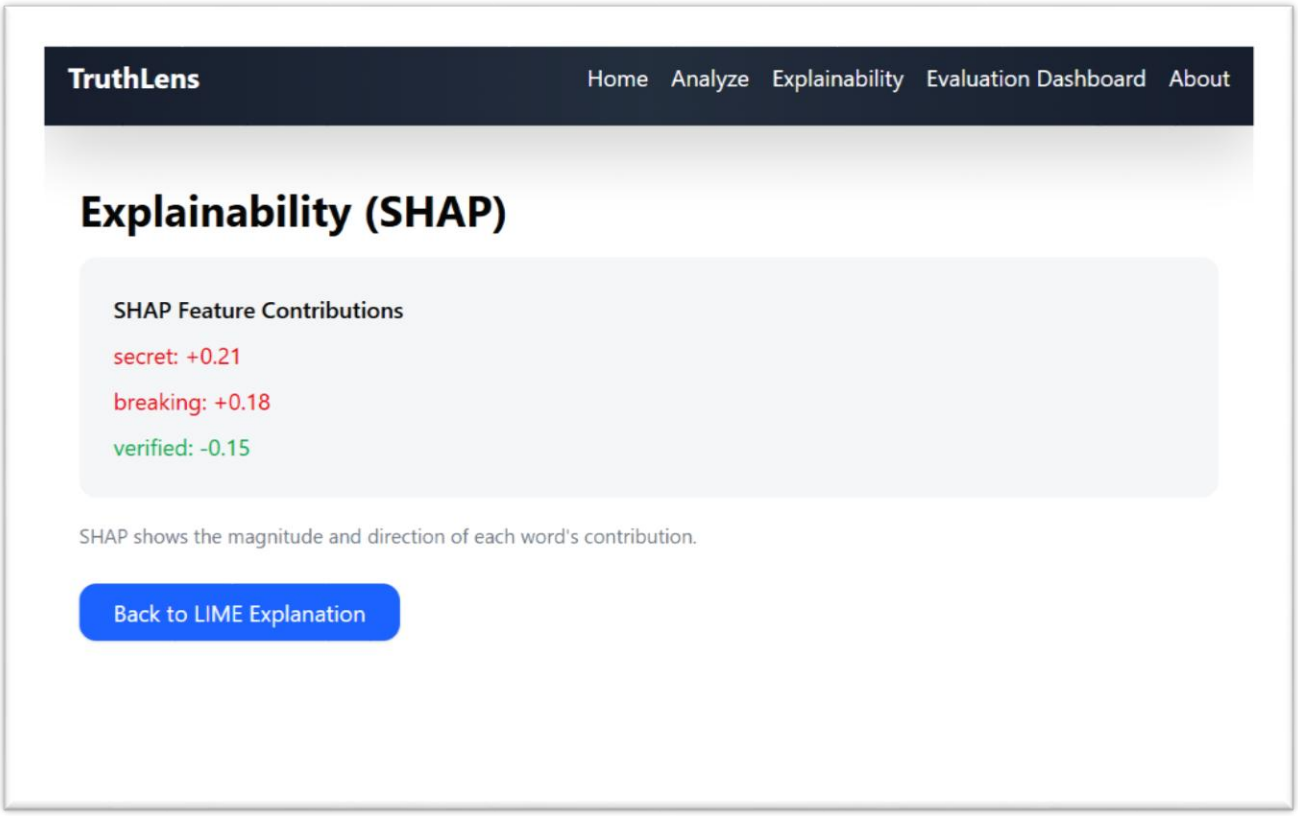


Figure 4.4: Shap Screen

5: Evalaution Dashboard:

This screen is static and would provide info about the evaluation metrics of the model its accuracy precision f1 score other metrics with graph but for now graph place is left intentionally and will be populated after complete model training

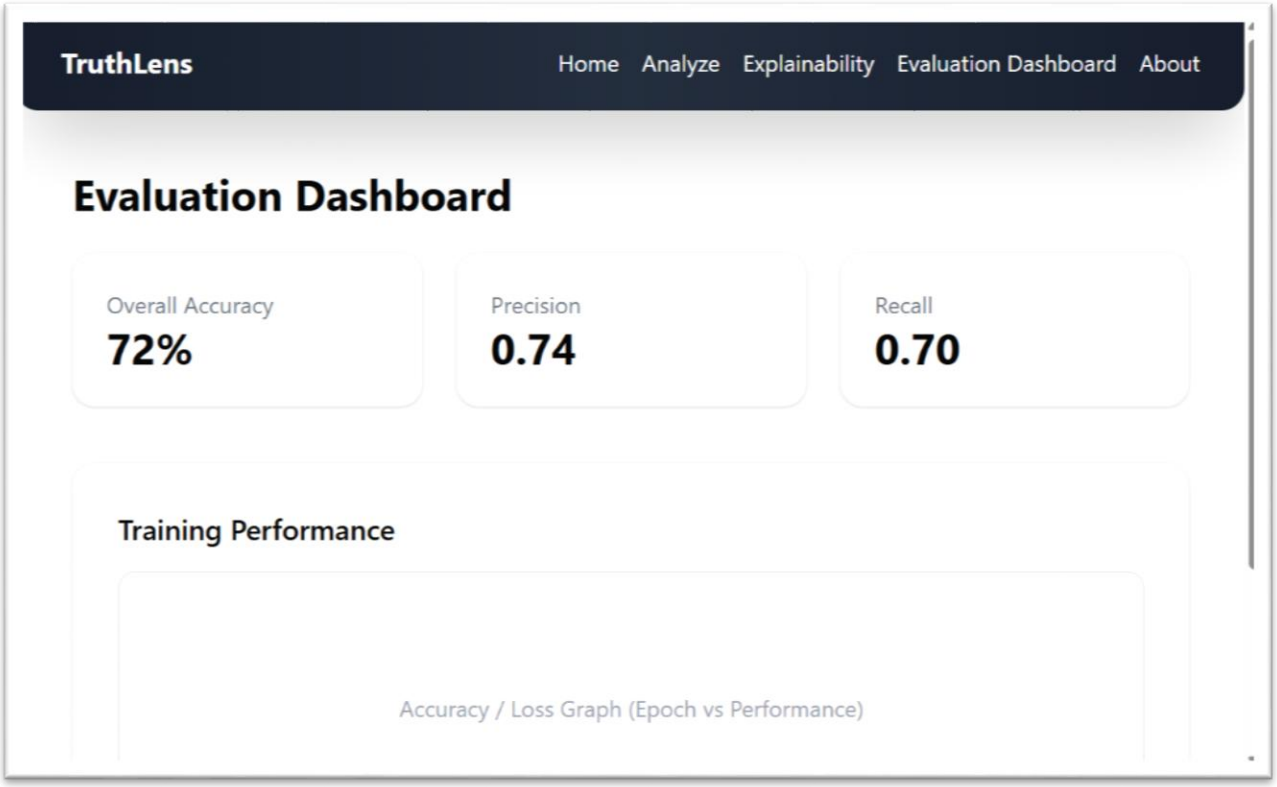


Figure 4.5: Evaluation Dashboard

6: About

This is also a static page which will give insight about the project and how team members and Supervisor contributed in the journey .this will be populated in last.

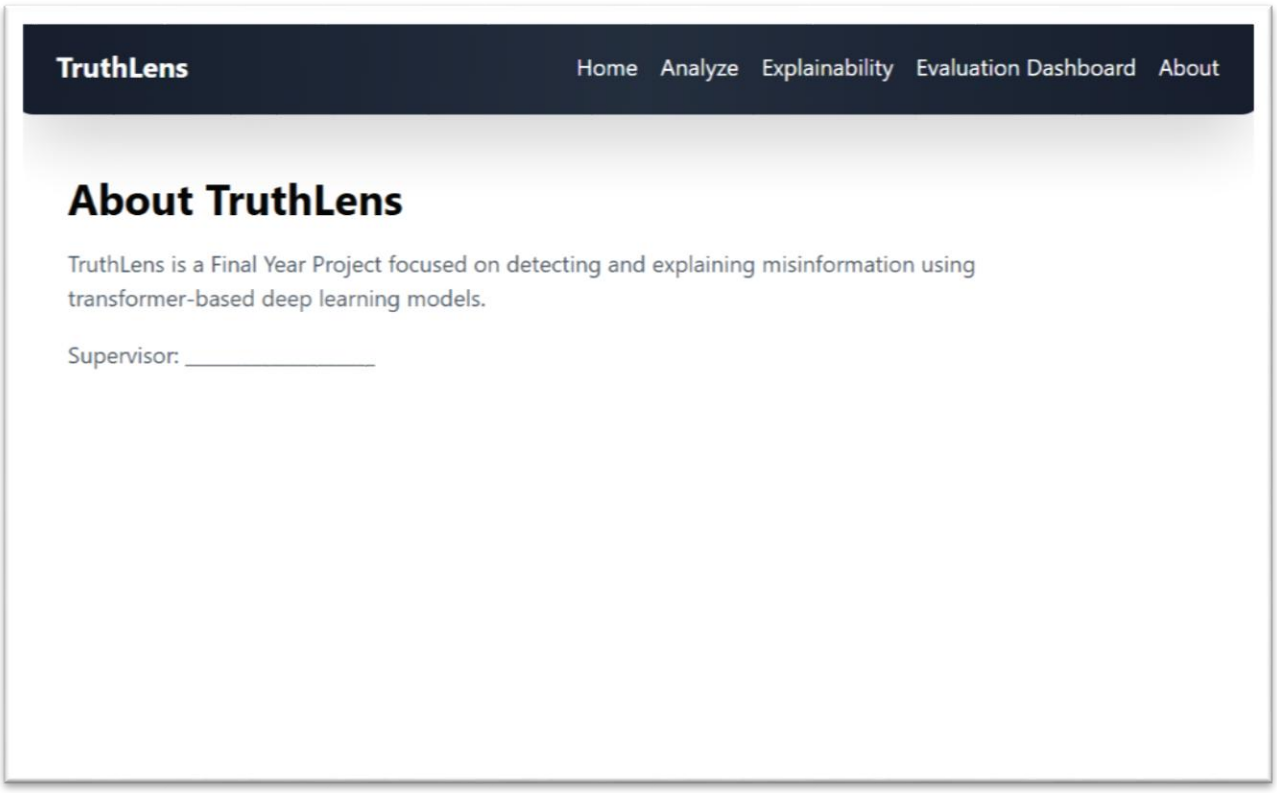


Figure 4.6: About

4.6 Challenges During Implementation

The deploying of the TruthLens system was done under the extreme academic, computational, and infrastructural limitations. The following were development challenges that were met and solved using practical engineering decisions.

4.6.1 Limited Computational Resources

The greatest obstacle was the use of free cloud-based GPU environments (Google Colab and Kaggle). The boundaries that are enforced by these platforms are limited to:

- GPU availability and session timeouts
- Restricted RAM and VRAM
- Sudden runtime disconnections

As a result:

- Long training runs were frequently interrupted
- Large batch sizes caused memory overflow
- Extended SHAP computations exceeded session limits

Mitigation:
The optimization of the model training was performed through the small batch size, limited epochs, and a proper choice of the maximum sequence length. The idea of explainability was to be performed on-demand as opposed to being continuous.

4.6.2 High Computational Cost of Explainability (SHAP)

Transformer-based models have computationally expensive SHAP explanations that require the estimation of Shapley values, which do not scale well both with input length and model complexity.

Practical issues faced:

- SHAP explanation time increased significantly for long news articles
- GPU memory exhaustion during token-level explanation
- Delays in interactive response

Mitigation:

- SHAP was restricted to user-triggered explanations only
- LIME was used for faster local explanations
- Explanation workflows were separated from prediction pipelines

This architecture is compliant with best practices that can be found in the literature of explainable AI on resource-constrained environments.

4.6.3 Dataset Heterogeneity and Input Length Variability

The system integrates two fundamentally different datasets:

- **LIAR dataset:** Short political statements
- **Kaggle dataset:** Long-form news articles

This caused challenges in:

- Selecting a fixed sequence length for BERT
- Preventing information loss during truncation
- Maintaining consistency across training samples

Mitigation:
There was a chosen balanced maximum length of token and the truncation and padding were done equally.

4.6.4 Integration Complexity Across System Components

The system combines:

- A transformer-based ML model
- Explainability frameworks (LIME and SHAP)
- Flask backend APIs
- React-based frontend UI
- Database storage

Challenges included:

- Synchronizing prediction and explanation outputs
- Managing asynchronous frontend–backend communication
- Ensuring consistent data formatting between modules

Mitigation:
Modular architecture was used and the execution occurred sequentially, with the preprocessing, prediction and explainability components being clearly defined through a clear REST API.

4.6.5 Latency and Interpretability Trade-off

The ability to balance between the degree of responsiveness and explainability of the system was one of the most urgent problems. Observed trade-offs:

- Fast predictions but slow explanations
- SHAP provided richer insight but higher latency
- LIME was faster but less globally consistent

Mitigation:
The system offers both explainability approaches, enabling the users to decide which option between speed (LIME) and depth (SHAP) one needs.

4.6.6 Time Constraints and Academic Deadlines

Due to strict academic schedules:

- Reasoning the exhaustive hyperparameter tuning was not feasible.
- It was not experimented on a large scale.
- Stability and reproducibility continued to be an issue that it had to listen to.

Mitigation:
Effective system design, the readability and repeatability of performance and less about a few points of marginal accuracy were all put in place.

4.6.7 Development and Debugging Challenges

During development, there were several low-level problems that were encountered and they included:

- Version conflicts among the libraries.
- Crash on serializing of models at runtime.
- UI - backend discrepancies on explanation rendering.

Mitigation:
The system was stabilized by employing the incremental testing, version pinning and repeated validation.

4.6.8 Engineering Insight Gained

These problems directly touched the system design. They have not pointed to the limitations as weaknesses, but they have turned the emergence of:

- A scaled and scalable architecture.
- Deployment logic that is aware of explainability.
- An actual and repeatable research implementation.

The last system proves that explainable transformer-based models can be successfully implemented despite limited computational resources, which aligns with the current trends in applied explainable AI.

References

[1] S. Alalawi, S. Baalfaqih, M. Almeqbaali, and M. M. Masud, "Social Media Misinformation Propagation and Detection," in *2023 15th International Conference on Innovations in Information Technology (IIT)*, 14-15 Nov. 2023 2023, pp. 240-245, doi: 10.1109/IIT59782.2023.10366483. [Online]. Available: https://doi.org/10.1109/IIT59782.2023.10366483

[2] T. Khan, A. Michalas, and A. Akhunzada, "SOK: Fake News Outbreak 2021: Can We Stop the Viral Spread?," *ArXiv,* vol. abs/2105.10671, 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2105.10671.

[3] T. Vieira Rodrigues, "Navigating the Fake News Environment: Enhancing Media Literacy in the Digital Age," *International Journal for Innovation Education and Research,* vol. 13, no. 1, 2025. [Online]. Available: https://doi.org/10.31686/ijier.vol13.iss1.4249.

[4] M. Berrondo-Otermin and A. Sarasa-Cabezuelo, "Application of Artificial Intelligence Techniques to Detect Fake News: A Review," *Electronics*, vol. 12, no. 24, doi: https://doi.org/10.3390/electronics12245041.

[5] A. Vaswani *et al.*, "Attention Is All You Need," *arXiv,* vol. 7, 06/12 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1706.03762.

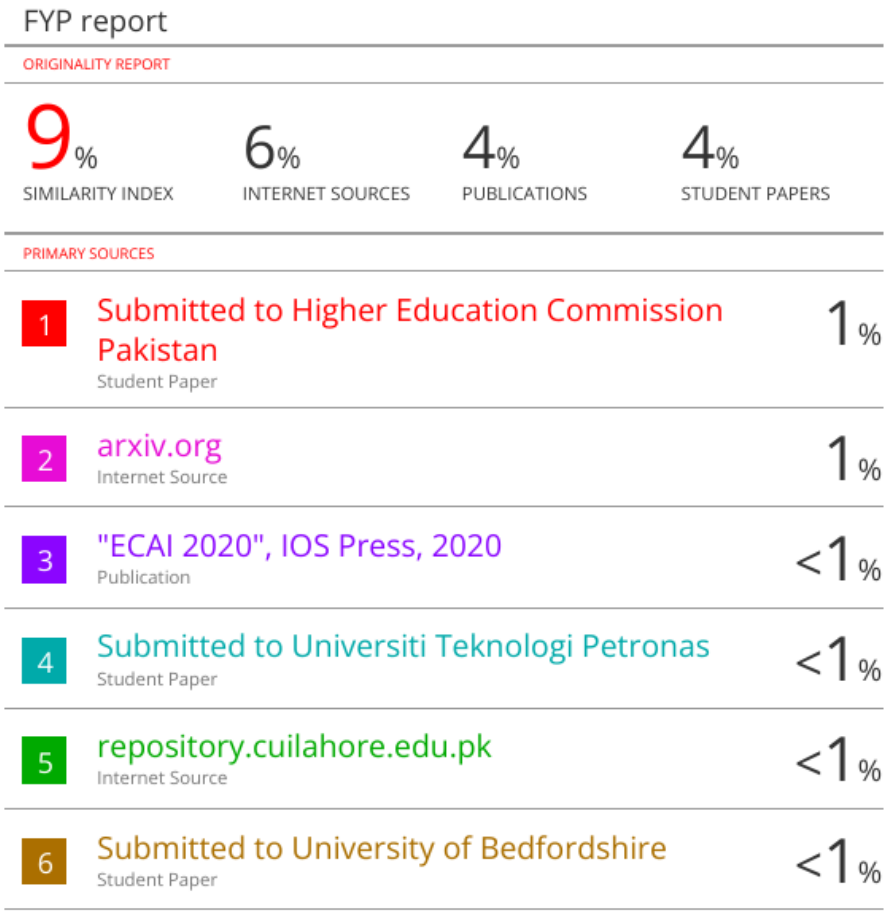
[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2018, vol. 1: Association for Computational Linguistics, doi: 10.48550/arXiv.1810.04805. [Online]. Available: https://aclanthology.org/N19-1423/

[7] M. Ribeiro, S. Singh, and C. Guestrin, ""Why Should I Trust You?": Explaining the Predictions of Any Classifier," presented at the The 22nd ACM SIGKDD International Conference, 2016. [Online]. Available: http://dx.doi.org/10.1145/2939672.2939778.

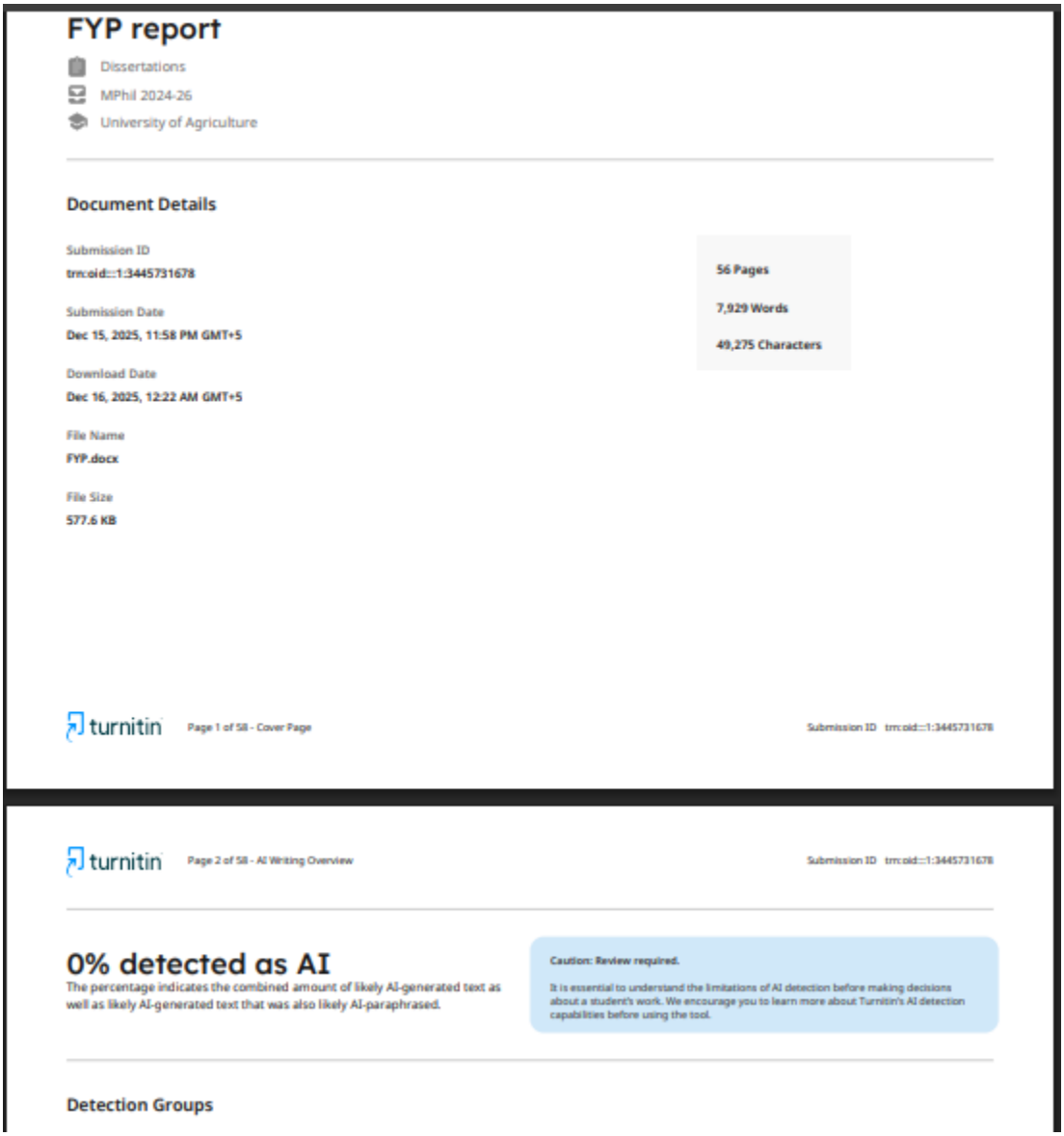
[8] S. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," presented at the NIPS, 2017. [Online]. Available: http://dx.doi.org/10.48550/arXiv.1705.07874.

[9] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu, "Fake News Detection on Social Media: A Data Mining Perspective," *ACM SIGKDD Explorations Newsletter,* vol. 19, no. 1, 08/06 2017. [Online]. Available: http://dx.doi.org/10.1145/3137597.3137600.

Similarity Report



AI Report



FYP report

Dissertations
MPhil 2024-26
University of Agriculture

Document Details

Submission ID
trn:oid:::1:3445731678

Submission Date
Dec 15, 2025, 11:58 PM GMT+5

Download Date
Dec 16, 2025, 12:22 AM GMT+5

File Name
FYP.docx

File Size
577.6 KB

56 Pages
7,929 Words
49,275 Characters

turnitin Page 1 of 58 - Cover Page Submission ID trn:oid:::1:3445731678

turnitin Page 2 of 58 - AI Writing Overview Submission ID trn:oid:::1:3445731678

0% detected as AI

The percentage indicates the combined amount of likely AI-generated text as well as likely AI-generated text that was also likely AI-paraphrased.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Detection Groups