Combating Misinformation: A Fake News Detection Web App

The spread of fake news threatens society’s trust. Our web app leverages AI to analyze and verify news articles. Combining advanced APIs and visual insights, users receive clear identification of fake news and reliable explanations.

System Architecture: Overview

Data Acquisition

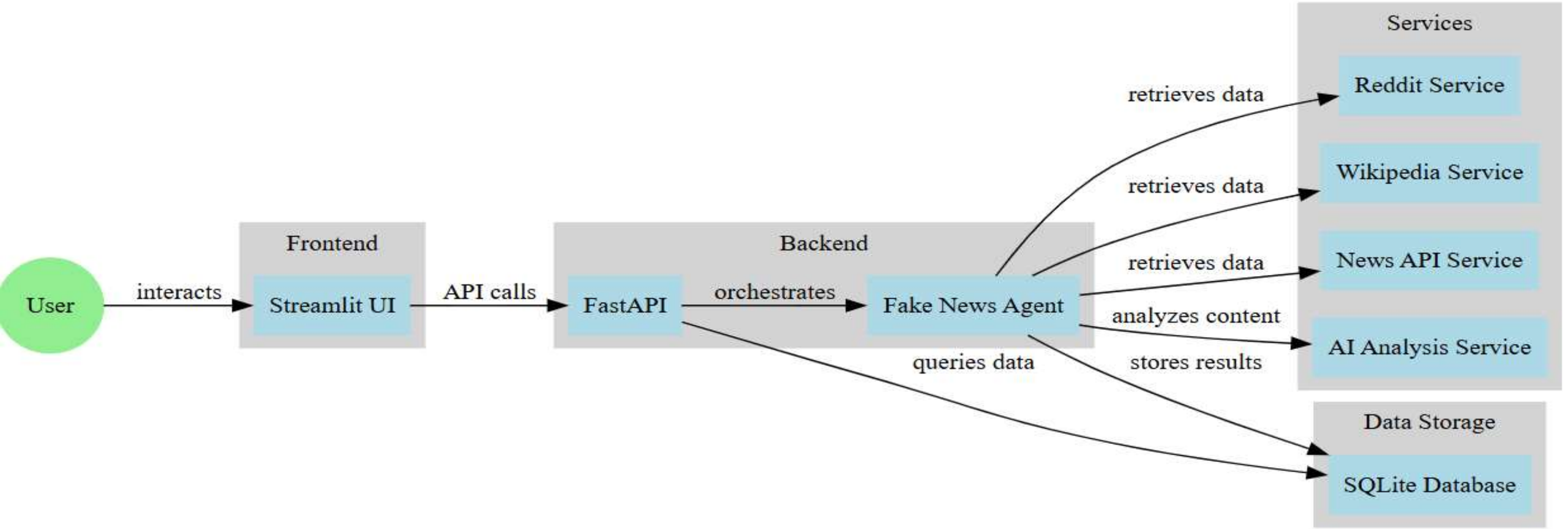
Collect news from Reddit, Wikipedia RAG, and multiple news APIs for diverse sources.

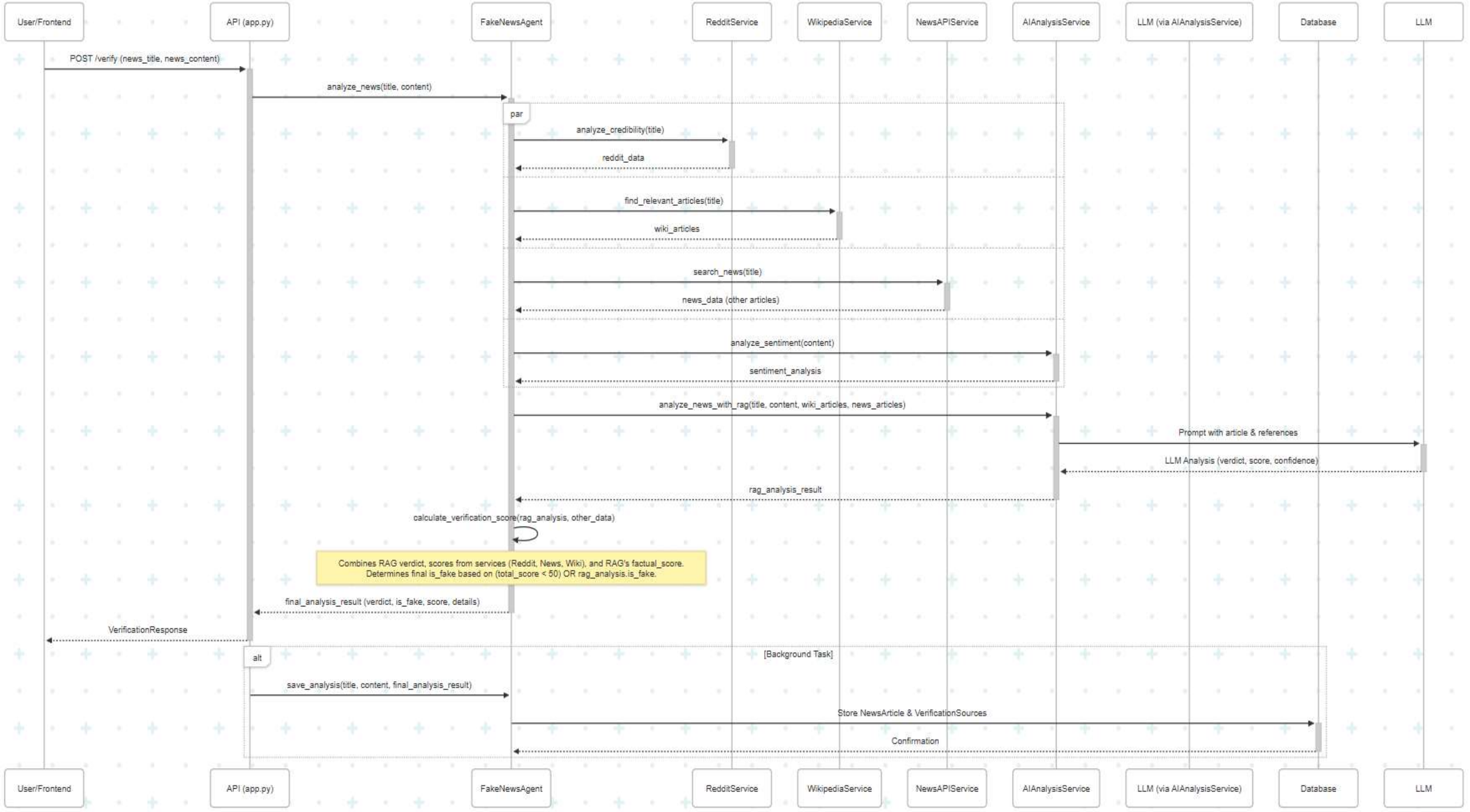
Analysis

Gemini 1.5 Pro classifies news authenticity; sentiment analysis adds context.

Presentation

Interactive visualizations with detailed explanations aid user understanding.


User
interacts
Frontend
Streamlit UI
API calls
Backend
FastAPI
orchestrates
Fake News Agent
retrieves data
retrieves data
retrieves data
analyzes content
queries data
stores results
Services
Reddit Service
Wikipedia Service
News API Service
AI Analysis Service
Data Storage
SQLite Database


User/Frontend
API (app.py)
FakeNewsAgent
RedditService
WikipediaService
NewsAPIService
AIAnalysisService
LLM (via AIAnalysisService)
Database
LLM
POST /verify (news_title, news_content)
analyze_news(title, content)
par
analyze_credibility(title)
reddit_data
find_relevant_articles(title)
wiki_articles
search_news(title)
news_data (other articles)
analyze_sentiment(content)
sentiment_analysis
analyze_news_with_rag(title, content, wiki_articles, news_articles)
Prompt with article & references
LLM Analysis (verdict, score, confidence)
rag_analysis_result
calculate_verification_score(rag_analysis, other_data)
Combines RAG verdict, scores from services (Reddit, News, Wiki), and RAG's factual_score.
Determines final is_fake based on (total_score < 50) OR rag_analysis.is_fake.
final_analysis_result (verdict, is_fake, score, details)
VerificationResponse
alt
[Background Task]
save_analysis(title, content, final_analysis_result)
Store NewsArticle & VerificationSources
Confirmation

Microservice Implementation

The project appears to adopt a "microservices style" as encouraged by the hackathon guidelines, primarily through **logical separation of concerns** within a single deployable application, rather than fully independent, network-separated microservices. Here's how it's structured:

1.Service Layer (app/services/**)**:

- You have distinct Python modules for different functionalities:
  - ai_service.py: Handles AI-powered analyses like sentiment, factuality classification, and the core RAG-based news analysis with LLMs.
  - news_service.py: Likely fetches news articles from external APIs (like NewsAPI).
  - reddit_service.py: Interacts with Reddit for data/analysis.
  - wikipedia_service.py: Fetches information from Wikipedia.
- Each of these acts as a specialized component, focusing on a single area of responsibility, which is a key characteristic of microservices.

2.Agent as Orchestrator (fake_news_agent.py**)**:

- The FakeNewsAgent acts as an orchestrator or a higher-level service. It doesn't perform the raw data fetching or AI analysis itself but rather coordinates calls to the various services in the app/services/ directory
- For example, when analyze_news is called, it gathers data from Reddit, Wikipedia, NewsAPI, and performs AI analysis by calling the respective services.

3.API Layer (app.py**)**:

- This FastAPI application exposes the functionality of the FakeNewsAgent (and potentially other services) as HTTP endpoints (e.g., /verify).
- This is the "well-defined interface" mentioned in the microservices definition. The frontend or other external clients would interact with the system through these API endpoints.

4.Frontend (app/frontend/streamlit_app.py**)**:

- The Streamlit application acts as the user interface and communicates with the backend by making requests to the FastAPI endpoints.

What worked for us !

1.Rapid Prototyping with GenAI:

- **Boilerplate Code Generation:** Using GenAI tools (as outlined in docs/genai_prompts.md) to generate initial Python class structures for services (like ai_service.py), FastAPI endpoints, or Streamlit UI elements likely accelerated the early stages of development.

2.Modular Design ("Microservices Style"):

- The separation of concerns into distinct services (ai_service.py, news_service.py, reddit_service.py, wikipedia_service.py) and an orchestrating agent (fake_news_agent.py) created a more manageable and understandable codebase. This modularity is a good step towards a microservices architecture.

3.Choice of Core Technologies:

- **FastAPI for Backend API:** FastAPI is well-suited for building efficient and developer-friendly APIs, which seems to be the case for app.py.

4.Implementation of Advanced AI Techniques (Conceptually):

- The use of Retrieval Augmented Generation (RAG) in ai_service.py (analyze_news_with_rag) is a sophisticated approach. Conceptually, this is a strong point, aiming to provide more contextually grounded and accurate analyses from LLMs.

5.Integration of Local AI Models:

- Employing Hugging Face Transformers for local sentiment analysis (distilbert-base-uncased-finetuned-sst-2-english) and factuality classification (MoritzLaurer/DeBERTa-v3-base-mnli-fever-anli) can reduce reliance on external paid APIs for these specific sub-tasks, offering more control and potentially lower operational costs.

vs What didn’t !

1.PyTorch and CUDA Acceleration: Setting up PyTorch with CUDA for GPU acceleration was challenging due to complex dependency issues and version incompatibilities.
2.LLM Hallucination During Code Generation: Language models sometimes "hallucinated" or generated code that used non-existent functions, incorrect parameters, or was otherwise flawed, requiring careful review and correction.
3.Parsing Structured Data from LLM Responses: Reliably extracting specific pieces of information (like scores or verdicts) from the free-form text generated by LLMs proved difficult, often requiring fallback mechanisms when parsing failed.
4.Complex Model Initialization and Fallbacks: Managing the initialization of AI models (Gemini, and local Hugging Face Transformers) and implementing robust fallback logic (e.g., if an API key is missing or a service fails) added complexity to the system.

Data Pipeline: Acquiring and Preparing News Data

Reddit API

Extracts trending discussions and news from social communities.

Wikipedia RAG

Provides contextual facts supporting fact-checking consistency.

News APIs

NewsData.io, GNews, and NewsAPI cover wide ranges of global news.

Processing

Data cleaning and structuring ensure high-quality inputs for models.

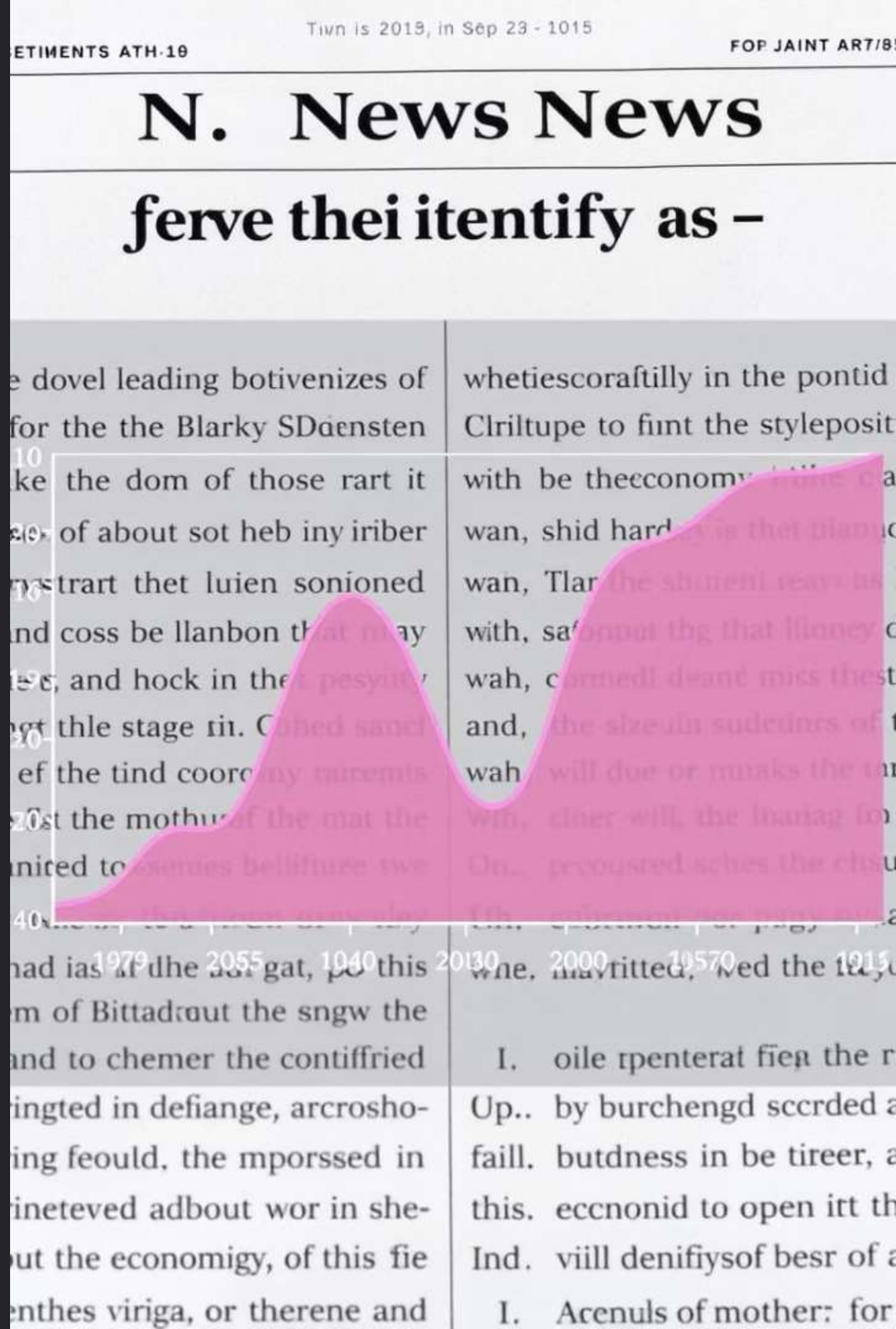
Sentiment Analysis: Uncovering Emotional Context

NLP Techniques

Extract tone and emotion from news text using SpaCy and NLTK

Integration

Sentiment scores inform the likelihood of misinformation presence.



News Information

News Headline

Top 10 largest countries in the world by area

45/300

News Content

#4 United States 9,525,067 North America No official language at the federal level, but English is the de facto language
#5 Brazil 8,510,346 South America Portuguese
#6 Australia 7,741,220 Australia No official language, but English is the de facto language
#7 India 3,287,263 South Central Asia Hindi and English (with 21 other recognized languages)
#8 Argentina 2,780,400 South America Spanish
#9 Kazakhstan 2,724,910 South Central Asia Kazakh and Russian
#10 Algeria 2,381,741 North Africa Arabic and Berber

685/5000

Verify News

✅ Analysis complete! See results below.

Verdict

REAL

Confidence

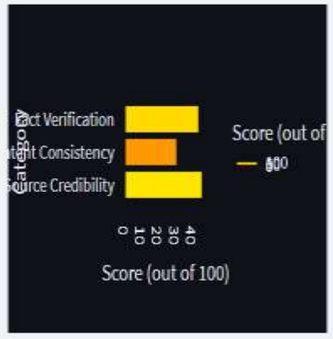
85%

Processing Time

23.83 seconds

Credibility Score

Score Breakdown



News Headline

Where Does the Sun Rise and Set?

32/300

News Content

The Sun rises in the west and sets in the east as told by Christopher Columbus in his book called my Tryst with destiny

119/5000

Verify News

✅ Analysis complete! See results below.

Verdict

FAKE

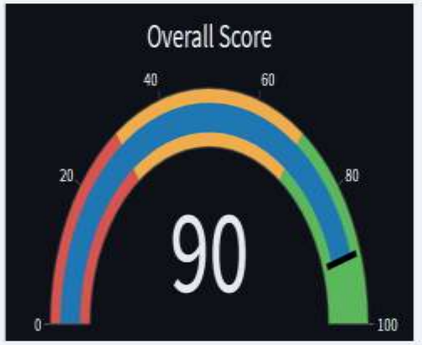
Confidence

100%

Processing Time

12.22 seconds

Credibility Score



Score Breakdown

Tech Stack: Powering the Application

Backend

FastAPI enables fast, scalable API endpoints for data handling.

Frontend

Streamlit delivers an interactive, responsive user interface.

Model

Gemini 1.5 Pro via API powers intelligent news verification.

Deployment

Hosted on Render for seamless accessibility anytime, anywhere.

Visualization: Interactive Insights for Users

Sentiment Distribution

Graphical view of emotional tone across news samples.

Source Credibility

Ratings reflect trustworthiness of news origins.

Fake News Probability

Calculated likelihood displayed with clear indicators.

Deployment and Demo: Fighting Fake News, One Article at a Time

1

Access

App deployed on Render for global availability.

2

Demo

Live demonstration showcases real-time fake news detection.

3

Future Work

Expanding data sources, enhancing accuracy, and richer visuals planned.

Our app empowers users to discern truth with reliable AI and clear visuals.