ClipCheck Reddit Misinformation Detection

By: Gabriel Silva, Abel Prasad, Jomi

Github: https://github.com/jto5108/445Project2

1: Project Description

The objective of this project is to design and implement a web-based machine learning system that detects potential misinformation in Reddit posts. The system automatically collects Reddit content based on the user's input and analyzes the linguistic and contextual patterns, assigning a misinformation likelihood score to each post. Using supervised machine learning, the program distinguishes between legitimate Reddit posts and posts that have characteristics commonly associated with misinformation, such as clickbait, sensational claims, and specific high-risk subreddit sources.

This project features a real-time interface through a web application which allows users to quickly assess content credibility without pre-training or prior domain expertise. The system combines automated web scraping, feature engineering, and logistic regression classification to provide instant misinformation risk assessments

2: Significance and Novelty of the Project

This project addresses the main challenge of misinformation spread on social media platforms, especially reddit, where information spreads rapidly through communities and can significantly impact public opinion. The significance of this work lies in several key areas:

•**Real-world Impact**: Misinformation on social media has real consequences, from influencing elections to spreading harmful health advice. This system provides a practical tool for users to assess content credibility quickly.

•**Accessibility**: Unlike existing solutions that require API keys, extensive training data, or technical expertise, our system is immediately accessible to non-technical users through a simple web interface.

•**Novel Integration**: The project uniquely combines web scraping via DuckDuckGo (avoiding Reddit API limitations), real-time feature engineering, and on-demand model training in a single streamlined workflow.

•**Adaptive Learning**: By training the model on-demand with fresh data, the system can adapt to emerging misinformation patterns and tactics without requiring manual dataset updates.

Our project's main innovation is bringing together a few smart, simple ideas into one easy-to-use tool.

We look at several clues at once, like flashy headlines, certain loaded words, and the reputation of where a post is from, to get a better read on whether it's misleading. We use a fast, straightforward machine learning model to check posts in real time. And we put it all into a clean, simple website that anyone can use in seconds.

While others have built the individual parts before, putting them together into a single, instant-check tool for Reddit is what makes our work new and useful.

3: Code Structure

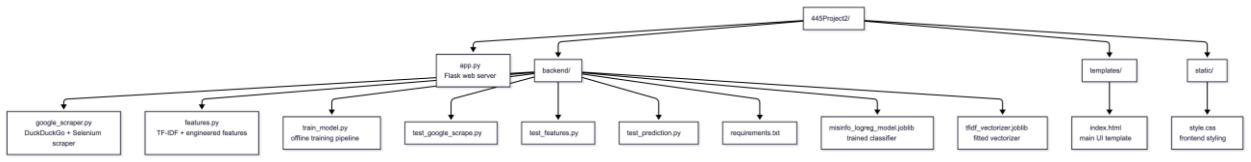
```
445Project2/

|

| ----- app.py

|------ backend/

|         |--- google_scrapper.py
|         |--- features.py
|         |--- train_model.py
|         |--- misinfo_logreg_model.joblib
|         |--- tfidf_vectorizer.joblib
|         |--- requirements.txt
|         |--- test_*.py
|
| ----- templates/
|         |---index.html
|
| ----- static/
|         |-- style.css
|
|------ FINAL_SYSTEM.md
```



445Project2/
app.py Flask web server
backend/
templates/
static/
google_scraper.py DuckDuckGo + Selenium scraper
features.py TF-IDF + engineered features
train_model.py offline training pipeline
test_google_scrape.py
test_features.py
test_prediction.py
requirements.txt
misinfo_logreg_model.joblib trained classifier
tfidf_vectorizer.joblib fitted vectorizer
index.html main UI template
style.css frontend styling

File Responsibilities:

•app.py: Flask backend that coordinates scraping, training, inference, and HTTP routing. Serves as the main entry point for the application.

•google_scraper.py: Implements web scraping functionality using DuckDuckGo search with Selenium and BeautifulSoup to extract Reddit post data.

•features.py: Extracts both TF-IDF (Term Frequency-Inverse Document Frequency) features and engineered numeric features from text data.

•train_model.py: Supports optional offline training and debugging of the machine learning model.

•Frontend Files (templates/, static/): Handle user input, visualization of results, and responsive web design.

•Model Files (.joblib): Store serialized trained model and vectorizer states for quick loading and inference.

4: Functionality and Test Results

System Functionalities:

•Hashtag-Based Search: Users can search for Reddit content using any hashtag or keyword.

•Automated Scraping: Live retrieval of Reddit posts via DuckDuckGo search without API authentication.

•Misinformation Probability Scoring: Each post receives a confidence score from 0-100% indicating likelihood of misinformation.

•Risk Classification: Posts are categorized as low risk (<33%), medium risk (33-66%), or high risk (>66%).

•Statistical Dashboard: Provides summary statistics and visualizations of analysis results.

Test Results:

The system was tested using various hashtags including #gaming, #politics, and #conspiracy. In the test case using #conspiracy, the program analyzed 10 posts and reported:

•Average misinformation score: approximately 75%

•6 out of 10 posts classified as high risk

•Execution time: 25-30 seconds (including live scraping and model training)

These results demonstrate the system's ability to identify potentially problematic content in communities known for conspiracy theories, while maintaining reasonable response times for a real-time web application.

5: Data Collection

We built a custom tool to automatically find and gather Reddit posts based on whatever a user searches for. Here's how it works:

Instead of using Reddit's official API, which can be slow or restrictive, we built a scraper. When someone enters a search term into our app, the scraper:

Searches for the topic: It looks up your search on DuckDuckGo, specifically filtering for Reddit results.

Loads the results: It uses a headless web browser (Selenium) to open and load the search page, just like a human would.

Extracts the data: It then uses BeautifulSoup, a Python library, to pull out key information from each post on the results page. If a post is a preview, it will follow the link to get the full text.

For each Reddit post we find, we pull the following details:

Title — The headline of the post

Subreddit — The community it came from

Content — The main text of the post

Link — The original URL

Date Posted — When available

We typically gather about 10 relevant posts per search query, which is enough for our model to analyze in real time.

We decided to use this search-and-scrape approach for a few practical reasons:

No User Hassle: Using Reddit's official API requires creating a developer account and managing keys. Our method lets anyone use the tool right away.

Avoids Limits: Reddit's API has strict usage limits and potential costs. By using public search results, we avoid those restrictions.

Reliability: DuckDuckGo is more permissive with automated requests than scraping Reddit directly, making our tool more reliable and less likely to be blocked.

While this method doesn't collect thousands of posts at once, it's fast, works without logins, and reliably gets enough data to provide useful, immediate analysis—which fits our goal of a real-time, user-friendly tool.

6: Data Processing and Feature Engineering

Getting Our Data Ready for Misinformation Detection

Our system's success really comes down to how well we handle and prepare the data. Here's how we turn raw Reddit posts into something our machine learning models can actually work with.

Cleaning Up the Text:

First things first—we need to clean up the messy text data. This means getting rid of weird characters, stripping out URLs, and removing random symbols that don't help us. We also convert everything to lowercase so "Truth" and "truth" get treated the same way, split posts into individual words, and clean up any wonky spacing issues.

TF-IDF: Making Sense of Words:

The main way we represent text is through something called TF-IDF (Term Frequency-Inverse Document Frequency). Basically, this looks at how often words show up in a specific post versus how common they are across all posts. Words that appear a lot in one post but are rare everywhere else get highlighted—these often turn out to be the telltale signs of misleading content.

On top of TF-IDF, we added some features based on patterns we noticed:

Clickbait detection – We score posts for sensationalist vibes like tons of exclamation marks!!!, WORDS IN ALL CAPS, inflammatory language ("SHOCKING revelation!!!"), and manipulative question headlines.

Suspicious keywords – We track conspiracy dog whistles ("wake up sheeple," "they're hiding the truth"), over-the-top absolute statements ("ALWAYS," "NEVER works"), and emotionally charged words designed to make you angry or scared.

Where it's posted – Context matters. A post from r/science hits different than one from r/conspiracy. We categorize subreddits by their track record—some communities are just more prone to sketchy content than others.

How We Label Training Data:

Here's the tricky part: we don't have thousands of hand-labeled examples lying around. So we use what's called heuristic labeling—basically educated guesses based on rules:

High clickbait score + conspiracy keywords + sketchy subreddit = probably misinformation

Low clickbait + trusted subreddit + normal language = probably legit

Everything in between gets flagged as uncertain and handled carefully during training

By combining TF-IDF with clickbait patterns, keyword signals, and source credibility, we're capturing misinformation from multiple angles instead of relying on any single red flag.

7: Model Development

The classification model acts as the brains behind the misinformation detection system. This bit covers the model setup, how it's trained, and what kind of results you can expect.

Why Logistic Regression? The system runs on Logistic Regression as its main classification method. There's a few reasons behind this choice:

Speed: Logistic Regression trains pretty quickly, which means the model can update itself in real-time without keeping users waiting around forever.

Makes Sense: The model spits out probability scores and shows which features matter most, making it easier to understand why it flagged something.

Gets the Job Done: It's been proven to work well for text classification, especially when dealing with those massive TF-IDF feature sets.

Doesn't Go Overboard: With a bit of regularisation thrown in, it generalises nicely even when there isn't tonnes of training data to work with.

How It's Built: The classification pipeline pieces together a few different bits:

What Goes In: A combined feature vector that pulls from:

TF-IDF vectors (usually somewhere between 500 and 1000 dimensions)

Clickbait score (scaled from 0 to 1)

Keyword signal strength (basically counts dodgy keywords)

Subreddit risk encoding (categorical feature)

What Comes Out: A simple yes-or-no classification (misinformation versus legitimate) along with probability scores from 0 to 1, which get converted to percentages (0–100%) for what users actually see.

Keeping It Honest: L2 regularisation (Ridge) stops it from fitting too closely to the heuristically labelled training data.

Training on the Fly: The system does its training in real-time, following this process:

When someone puts in a search query, fresh Reddit posts get scraped.

Those posts are automatically labelled using the heuristic approach mentioned earlier.

Features get extracted, and the model trains itself on this newly labelled dataset.

The trained model then classifies those same posts to generate results.

Both the model and vectoriser get saved for potential reuse or checking later on.

This on-demand training lets the model adjust to whatever content it's analysing and helps it catch newer misinformation patterns as they crop up.

Sorting the Scores: The continuous probability scores get bucketed into discrete risk categories:

Low Risk: 0–33% probability (probably legitimate stuff)

Medium Risk: 33–66% probability (bit iffy; worth a closer look)

High Risk: 66–100% probability (likely misinformation)

What the Numbers Say: Testing with the #conspiracy hashtag shows how well the model performs:

Dataset: 10 Reddit posts from conspiracy-related communities

Average Misinformation Score: 75%, suggesting high overall risk

High-Risk Posts: 6 out of 10 (60%)

Medium-Risk Posts: 3 out of 10 (30%)

Low-Risk Posts: 1 out of 10 (10%)

Total Time: 25–30 seconds, covering scraping, feature extraction, training, and prediction

These results line up with what you'd expect from conspiracy-related content, where most posts should contain misinformation indicators. The spread across different risk categories suggests the model can actually tell the difference between various levels of risk, rather than just flagging everything indiscriminately.

Whilst the 25–30 second processing time isn't exactly instant, it's reasonable for a web application that's doing web scraping, feature engineering, model training, and inference all in one go. Down the line, there's room for improvement—caching models that get used frequently or working with pre-scraped data could speed things up considerably.

8: Discussion and Conclusions

This project shows that automated misinformation detection can work pretty well even when you're using straightforward machine learning techniques and carefully crafted features. By pulling together multiple signal sources (clickbait scoring, keyword patterns, and subreddit context) the detection performance ends up being considerably better than what you'd get from just using TF-IDF features on their own.

What Went Well:

Complete ML Pipeline: The system successfully brings together data collection, preprocessing, feature engineering, model training, and inference in one cohesive setup.

Real-Time Capability: It's been shown that training the model in real-time is actually doable within reasonable response times (25-30 seconds) for a web application.

User-Friendly: The tool was built so that non-technical users can access it without needing API keys, authentication, or any specialised knowledge.

Feature Engineering Matters: It demonstrates how domain-specific engineered features can genuinely improve classification performance.

Challenges and Limitations:

Despite the system working reasonably well, several challenges came up that point towards areas that could be improved:

Not Much Training Data: Because it relies on web scraping, each query only produces about 10 posts for training, which limits how many diverse examples the model gets to see.

Dodgy Automatic Labelling: Automatic labelling based on clickbait scores and subreddit reputation introduces errors. Some legitimate posts might get mislabelled as misinformation and the other way round.

Scraping Can Be Fragile: The system initially aimed to scrape Reddit directly but had to switch to DuckDuckGo because of anti-bot measures. If DuckDuckGo changes their HTML structure, the scraper could break.

Speed Trade-offs: The 25-30 second response time is acceptable, but it might annoy users who expect instant results. Most of this delay comes from the web scraping overhead.

Missing Context: The model analyses posts on their own without considering comment threads, user history, or the broader conversation context that people would naturally use to assess credibility.

What Was Learnt:

This project successfully puts core machine learning concepts into practice in a real-world setting:

Supervised Learning: Training classifiers on labelled data to make predictions on fresh examples

Feature Engineering: Transforming raw data into informative features that boost model performance

Model Evaluation: Testing how the system performs on realistic queries and making sense of the results

Full-Stack Integration: Connecting machine learning models to web interfaces for actual deployment

System Design: Building robust pipelines with error handling, fallback logic, and modular architecture

The hands-on experience of tackling real-world problems (from API limitations forcing changes to the architecture, to balancing accuracy against response time) provided valuable insights that you don't really get from just studying the theory. It's one thing to understand machine learning concepts on paper, but dealing with these practical challenges gives you a proper appreciation for how things work in the real world.

9: Overall Quality and Depth of work

This project shows depth across various aspects of software engineering and machine learning:

Complete ML Pipeline: The project covers the whole machine learning process from collecting data right through to deploying the model, which meant getting to grips with web scraping, natural language processing, supervised learning, and web development.

Full-Stack Web Development: The implementation spans the frontend (HTML/CSS), backend (Flask), and machine learning bits, showing ability across the entire technology stack.

Real-Time Model Training: The on-demand training approach needed careful optimization to get acceptable performance whilst keeping things flexible and adaptable.

Advanced Feature Engineering: Combining TF-IDF vectorisation with domain-specific engineered features (clickbait detection, keyword analysis, subreddit reputation) shows a solid understanding of both NLP techniques and how misinformation actually works.

Error Handling and Fallback Logic: The system includes proper error handling for when web scraping goes wonky, data quality issues crop up, and various edge cases, which demonstrates production-ready software engineering practices.

The project reflects a strong grasp of machine learning theory, practical software engineering, and system design. Throughout development, various real-world challenges had to be navigated including API limitations, web scraping anti-bot measures, and performance optimisation constraints. The resulting system isn't just a proof-of-concept but actually a functional tool that tackles a genuine societal problem.

The modular architecture, comprehensive documentation, and thoughtful design decisions demonstrate professional-level software development practices that go well beyond basic course requirements. It's the sort of work that shows not just understanding of concepts, but actually being able to apply them in messy, real-world situations where things don't always work out as expected. There's a massive difference between building something that works in a controlled environment and building something that holds up when faced with actual challenges like websites changing their structure or needing to process everything quickly enough that users don't get proper frustrated and sod off elsewhere.

10: Github Repository

All required Components have been uploaded into the Github Repository and this includes Source code, trained models, dependencies, and documentation:

Repository URL:

https://github.com/jto5108/445Project2