Project Proposal

Connor Denney

The "Fake News" Multi-Model Disinformation Dashboard

1. Problem Overview

- **Target Audience:** The primary target audience is the **casual scroller, online news consumer**. This demographic includes university students, busy professionals, and any non-technical individual who gets a significant portion of their news from social media feeds, aggregators, and forwarded links.
    - **Needs:** They need a way to quickly and easily verify the trustworthiness of a piece of content (an article, a post) without being an expert in digital forensics.
    - **Pain Points:** The current process of verification is high-friction. A skeptical user has to (1) read the article, (2) open a new tab, (3) search for a fact-checking site, (4) search for the claims, and (5) separately assess any media. This process is too slow for the speed of modern information, and most users simply won't do it.
- **Context:** We are in a "disinformation crisis." The barrier to creating sophisticated, realistic-looking misinformation has collapsed due to generative AI. Malicious actors can now mass-produce fluent text, photorealistic images, and convincing (if short) deepfake videos for pennies. This content spreads rapidly on platforms designed for engagement, not truth.
- **Significance:** The consequences of unaddressed misinformation are severe, including the erosion of public trust in media and institutions, increased political polarization, public health risks (e.g., vaccine misinformation), and financial scams/fraud. The problem is no longer just "fake news"; it's an attack on information integrity.
- **Analyze Existing Solutions:**
    - **Manual Fact-Checking Sites (e.g., Snopes, PolitiFact):** These services are invaluable but inherently *reactive*. By the time an article is manually debunked, it has often already reached millions of people.
    - **Source Reputation Tools (e.g., NewsGuard):** These are excellent for rating the *source* (e.g., nytimes.com) but cannot analyze a *specific piece of content*. A "good" source can still publish a bad article.
    - **Siloed AI Detectors (e.g., GPTZero, Hive AI):** These tools are powerful but specialized and disconnected. A user must copy-paste text into one tool, upload images to another, and still have no solution for fact-checking the claims. This "silo" problem is the key gap my project will fill.

2. Proposed Solution

- **Core Concept:** I propose **Aegis**, a web-based "misinformation dashboard." Aegis is a service-oriented application that aggregates the power of multiple, best-in-class AI verification APIs into a single, unified "Trust Report." A user will provide a single URL, and Aegis will orchestrate a multi-model analysis—checking the text for AI generation, identifying its core claims, and (as a stretch goal) analyzing its media for deepfakes.

- **Minimum Viable Product (MVP) Features:**
    1. **Unified Content Ingest:** A clean UI where a user can paste a URL. The system will scrape the article's main text.
    2. **Back-End Orchestrator API:** A central Python (FastAPI) back-end that manages the analysis pipeline. It will receive the scraped text and make asynchronous calls to all integrated external AI services.
    3. **Text Generation Analysis:** Integrate with an AI text-detection API (e.g., **Copyleaks API**). The orchestrator will send the article text and get back a score on the likelihood of it being AI-generated.
    4. **Claim Identification & Fact-Check:** Integrate with the **ClaimBuster API**. The orchestrator will send the text to identify "check-worthy" factual claims. The results will be displayed to the user.
    5. **Unified Trust Report:** A single, clean dashboard that presents the results from all modules in an easy-to-understand, non-technical format (e.g., "Text is 85% likely to be AI-generated," "2 Factual Claims Identified," etc.).
- **Post-MVP Roadmap:** This roadmap provides clear "stretch goals" to fill the 15-week semester and demonstrate foresight.
    1. **Image/Video Analysis Module:** Integrate a deepfake detection API (e.g., **Hive AI**). This is a complex but high-value addition. The back-end orchestrator would be updated to scrape key images/videos from the URL and send them for analysis.
    2. **Google Fact-Check Integration:** As a secondary source to ClaimBuster, query the **Google Fact Check Tools API** to see if the claims identified have already been debunked by a known publisher.
    3. **Browser Extension:** A Chrome/Firefox extension that allows a user to launch an analysis of their current page with a single click.
    4. **Source Reputation Score:** Add a module that also provides a score for the source domain (e.g., by integrating a service like NewsGuard) to complement the content analysis.
- **Connecting Solution to Problem:** Aegis directly solves the "silo" and "high-friction" pain points. It moves verification from a multi-step, expert-level research task into a simple, one-click action. It empowers the "casually scrolling" user to perform a sophisticated, multi-model analysis in seconds, right when they first encounter the content.

3. Validation Plan

- **User Testing & Research:**
    - **Target:** Fellow university students (a diverse and highly accessible pool representing the target audience).
    - **Phase 1: Prototype Validation (Week 4):** Create Figma mockups of the "Trust Report" UI. Conduct interviews, asking "Does this report make sense to you? What information is most/least useful? What's missing?"
    - **Phase 2: MVP Usability Testing (Week 10):** Conduct task-based usability tests with 5 users. Give them 3 URLs (one known-good, one known-fake, one AI-generated) and ask them to "determine if you should trust this article" using the tool. I will observe for confusion, hesitation, and "aha!" moments.
- **Feedback Integration:**

    - **Collection:** All feedback from interviews and tests will be collected into a simple Google Sheet.
    - **Prioritization:** I will use a simple **Trello board** as a master backlog. Feedback will be translated into "User Stories" (e.g., "As a user, I want to see *why* a claim is 'False'"). I will prioritize this backlog using the **MoSCoW** method (Must have, Should have, Could have, Won't have) to decide what to build next.
- **Software Testing (Automated & Manual):**
    - **Unit Tests (Pytest):** This is critical for an integration-heavy project. I will write unit tests for all "adapter" functions—the code responsible for translating the JSON responses from external APIs into my app's internal, normalized data model. This ensures my *translation* is correct.
    - **Integration Tests:** I will create a test suite that makes *live* calls to my back-end API with a set of test URLs. These tests will verify that the full pipeline (Scraping -> Orchestration -> API Calls -> Final Report) is functioning correctly.
    - **Manual QA:** Before each user-testing session and the final submission, I will perform a 10-point manual check to test for edge cases (e.g., broken URLs, URLs with no text, URLs with no images, API timeouts).

4. Iteration Plan (15-Week Outline)

Week	Focus / Goal	Key Activities	Potential Deliverables
1	**Project Kick-off**	Finalize project scope. Set up Git repo. Sign up for all API developer accounts.	Project Proposal (this doc). GitHub repository.
2	**Research & API Validation**	Research & select final APIs. Get "Hello World" curl requests working for all APIs.	API keys. Postman collection of working API calls.
3	**System Architecture**	Design the back-end API (endpoints, data models). Design the front-end/back-end contract (the JSON structure).	System architecture diagram. OpenAPI (Swagger) spec.
4	**Prototyping & UX Design**	Design Figma mockups of the "Trust Report" dashboard.	Figma prototype. **Validation:** Conduct Phase 1 user interviews.
5	**Back-End: Scaffolding**	Build the core FastAPI server. Implement the URL scraping/text extraction module.	A working /scrape endpoint.
6	**Back-End: Text Module**	Integrate the Copyleaks (or other) API. Write the adapter function and unit tests.	/analyze endpoint now returns AI text score.

7	**Back-End: Fact-Check Module**	Integrate the ClaimBuster API. Write the adapter function and unit tests.	/analyze endpoint now also returns identified claims.
8	**Back-End: Orchestration**	Refactor all API calls to run asynchronously (in parallel) to ensure a fast response.	A fully-featured, unified /analyze endpoint.
9	**Front-End: UI Build**	Build the React UI (input box, button, loading state) based on Figma mockups.	A static but functional React application.
10	**Full-Stack Integration**	Connect the React front-end to the FastAPI back-end. Build the "Trust Report" dashboard to display the live JSON.	**Deliverable: MVP. Validation:** Conduct Phase 2 usability tests.
11	**Feedback & Refinement**	Analyze feedback from usability tests. Triage bug reports and UI/UX suggestions into the Trello backlog.	Prioritized backlog of issues and features.
12	**Iteration 1 (Post-MVP)**	Implement the highest-priority feedback from testing (e.g., UI polishes, error handling).	An improved, more robust MVP.
13	**Iteration 2: Stretch Goal 1**	Begin work on the Post-MVP Roadmap. **Goal:** Integrate the Hive AI API for image analysis.	/analyze endpoint now includes basic image analysis.
14	**Iteration 3: Stretch Goal 2 Prep final**	Update the React front-end to gracefully display the new image analysis data. Start on final paper/presentation	Final feature-complete application. Code freeze.
15	**Finals Week**	Prepare the final presentation. Write final project paper. Deploy the application to a public URL.	Live demo. Final presentation slides. Final paper.

5. Technical Stack

- **Programming Languages:**
  - **Python:** For the back-end. Its robust data handling, asyncio support, and dominance in the AI/data space make it the perfect choice.
  - **JavaScript (TypeScript):** For the front-end, to build a modern, responsive user interface.
- **Frameworks & Libraries:**

    - **FastAPI (Python):** A high-performance, modern Python framework for building the back-end API. Its native async support is essential for orchestrating multiple API calls in parallel.
    - **React (JavaScript):** A leading front-end library ideal for building the dynamic, component-based "Trust Report" dashboard.
    - **Pytest (Python):** For writing and running all unit and integration tests.
- **Database:**
    - **None (Stateless by Design):** For the MVP, the application will be stateless (it receives a request, processes it, and returns a response without storing any data). This strategically reduces complexity, eliminating the need for database management, user accounts, or auth.
- **AI Models or Integrations (The "Borrowed Systems"):**
    - **AI Text Detection: Copyleaks API** or **GPTZero API** (commercial, high-accuracy).
    - **Fact-Checking: ClaimBuster API** (academic, excellent for identifying claims).
    - **(Post-MVP) Media Detection: Hive AI API** (commercial, robust deepfake and AI-generated image detection).
    - **(Post-MVP) Fact-Checking: Google Fact Check Tools API** (authoritative, to find existing debunks).
- **Deployment & Hosting:**
    - **Back-End (FastAPI): Heroku** or **AWS Elastic Beanstalk**. These PaaS (Platform-as-a-Service) solutions are well-suited for deploying Python web servers.
    - **Front-End (React): Vercel** or **Netlify**. These platforms are optimized for hosting static/JAMstack front-ends and offer a seamless, free-tier CI/CD pipeline.