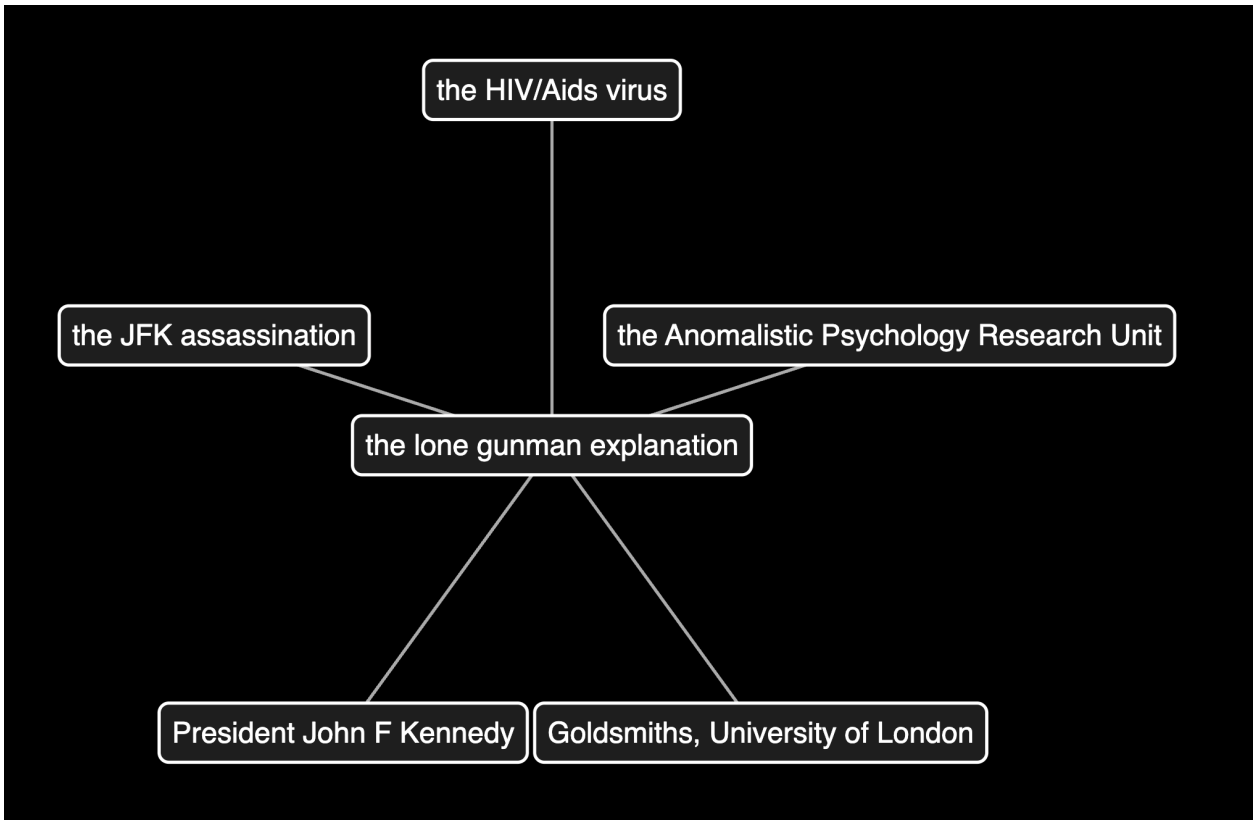
ART AND MACHINE LEARNING
CMU 2025 SPRING

The Belief Graph Explorer



Emma Chen, Joyce Lam, Andy Ouyang

DESCRIPTION

We created the **Belief Graph Explorer**, an interactive tool that allows users to explore how different beliefs are interconnected through a dynamic visual graph interface. Built on a foundation of real-world data extracted from news articles, social media discussions, and Wikipedia entries, the Explorer maps out how various ideas, conspiracies, and topics cluster together based on conceptual similarity and co-occurrence. Users can input a belief, instantly see a web of related ideas, and continue clicking deeper, much like falling down an information "rabbit hole." The tool emphasizes how beliefs are not isolated, but rather exist within complex networks that can lead users from familiar concepts into more extreme or unexpected territories. By making the navigation intuitive and exploratory, the **Belief Graph Explorer** offers a playful yet revealing way to experience the architecture of information ecosystems, reflecting the very real ways that exposure to related content can shape, reinforce, or even radicalize perspectives over time.

Concept

The inspiration for this project came from observing how easily people fall down "rabbit holes" online — starting from one simple belief or idea and quickly finding themselves immersed in an expanding world of related concepts. We wanted to recreate that experience visually and interactively, highlighting the organic, networked nature of knowledge and belief systems. The goal was to provide a tool that feels intuitive, encourages curiosity, and prompts reflection about the structure of information itself.

The goal was to create something that feels both funny and unsettling. Our product reflects how real-world information (and misinformation) often spreads online. It explores how small seeds of belief can quickly snowball into intricate, self-reinforcing narratives when shaped by algorithms. A key influence was the literature on "belief graphs" and knowledge networks, especially how recommender systems can subtly guide users toward specific ideas. We also drew from studies on interactive visualization techniques, emphasizing the importance of user-driven exploration rather than linear presentation.

Technique

Throughout our project, we leveraged a combination of machine learning, natural language processing, and generative AI techniques to explore how algorithmic recommendation systems can amplify misinformation and conspiracy thinking.

First, we constructed a custom dataset by aggregating articles from NewsAPI, The New York Times, The Guardian, Reddit, and Wikipedia. These documents were processed using spaCy to extract conceptual keywords for semantic filtering. To model relationships between concepts, we built a weighted, undirected graph in NetworkX, where edges reflected the frequency of co-occurrence within documents. We then trained a Node2Vec model on this graph to generate 64-dimensional embeddings, using 30-step random walks and transition biases based on edge weights, allowing similar concepts to cluster in vector space. For narrative generation, we first integrated Ollama to deploy lightweight language models capable of producing conspiracy stories based on user-selected concepts. Later, we experimented with fine-tuning LLaMA 2, using quantization via BitsAndBytesConfig to manage hardware limitations and optimize performance.

Then, we created a simple web application where users could type a belief, see results instantly, and interact with them. The backend was developed in Python to handle belief queries and generate related results, while the frontend was built using HTML and JavaScript to create a responsive, interactive experience.

Process

The development of our project involved a lot of iterative experimentation, technical troubleshooting, and conceptual refinement. Below is a summary of the key stages.

Web Scraping and Concept Tagging

Our first step was to construct a comprehensive dataset to support the project's recommendation engine. Initially, we focused on sourcing content from platforms such as Reddit, Wikipedia, Gab, and The Onion. However, despite multiple attempts, we were ultimately unable to incorporate data from Gab and The Onion due to technical difficulties. Gab required authentication protocols that restricted access to posts, while The Onion's content structure made it difficult to extract articles directly from links. We still wanted our dataset to include a wide range of news sources, so we pivoted to using NewsAPI as well as individual APIs from outlets like The New York Times and The Guardian. We also explored using public datasets, such as the FakeNewsCorpus.

We faced a lot of challenges during this process, particularly with API authentication issues and failures in web scraping caused by outdated or broken URLs. However, we were able to collect a smaller but cleaner dataset of news articles, Wikipedia pages, and Reddit posts in the end. Then, all the data was processed using spaCy to extract and assign conceptual keywords, enabling semantic filtering and relevance within the recommendation system.

For consistency and ease of use, the scraped data was formatted into JSON files, structured as lists of dictionaries containing key metadata fields. Our initial dataset was modest, consisting of approximately 20 NewsAPI articles, 80 Wikipedia entries, and 25 Reddit posts, which we used to experiment with tagging concepts and constructing a basic graph structure. Through iterative expansion and refinement, we ultimately grew the dataset to include over 1,000 data points: 100 articles from NewsAPI, 200 from The New York Times, 200 from The Guardian, approximately 180 Wikipedia entries, and around 600 Reddit posts sourced from r/conspiracy. This final dataset provided a solid foundation for both the story generation and the recommendation system of our project.

Figure 1. wiki180.json


```
[
  {
    "title": "Black helicopters",
    "summary": "This conspiracy theory emerged in the US in the 1960s. The John Birch Society
  },
  {
    "title": "Chemtrails",
    "summary": "Also known as SLAP (Secret Large-scale Atmospheric Program), this theory alleg
  },
  {
    "title": "Korean Air Lines Flight 007",
    "summary": "The destruction of Korean Air Lines Flight 007 by Soviet jets in 1983 has long
  },
  {
    "title": "Malaysia Airlines Flight MH370",
    "summary": "The disappearance of Malaysia Airlines Flight 370 in southeast Asia in March 2
  },
  {
    "title": "Malaysia Airlines Flight MH17",
    "summary": "Malaysia Airlines Flight 17 was shot down over Ukraine in July 2014. This even
  },
  {
    "title": "<i>Deepwater Horizon</i>",
    "summary": "Multiple conspiracy theories pertain to a fatal oil-rig industrial accident in
  },
  {
```


Moving on from web scraping, we had to clean the data and get all of the JSONs into a similar format so that we could process them later into concepts. For example, for Reddit posts, we combined the title and body fields into a single title field and renamed comments to summary (list of strings). As such, we standardized everything into a title and summary format within each dictionary.

For SpaCy concept extraction, the first version dumped all word chunks without filtering, resulting in noisy, irrelevant concepts (e.g., single numbers, "the", "this"). After trial and error,

we introduced regex-based filters and POS tagging to greatly improve the quality of extracted spans. We went through several iterations of debugging to refine the spaCy concepts we were extracting, including getting rid of duplicates, only retaining proper nouns and nouns, getting rid of dates and numbers, parts of speech like pronouns, debugging for punctuation inclusions like hyphens and apostrophes, etc.

Figure 2. spaCy concept extraction first iteration based on small dataset

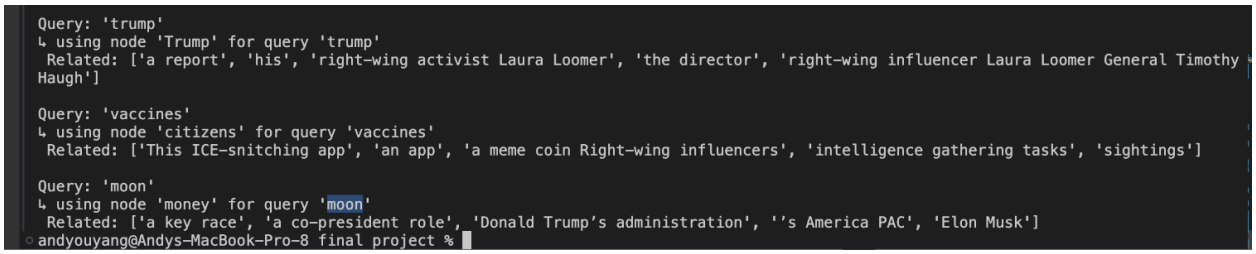
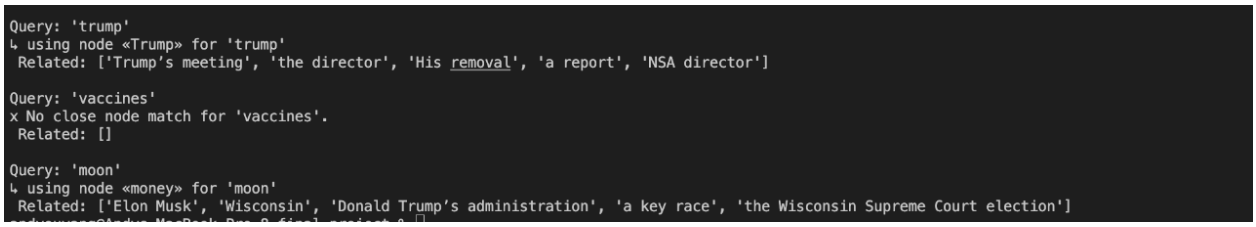


Figure 3. spaCy concept extraction later iteration based on small dataset



Making and Refining the Graph and Node2Vec Model

We start by treating each document's extracted concepts as a tiny "co-occurrence list." For every record, we look at all unordered pairs of concepts that appear together and keep a running count of how many times each pair shows up across the entire corpus. Those counts become the "weights" on the edges of an undirected NetworkX graph, where each unique concept is a node and heavier edges indicate stronger co-occurrence associations. Once the weighted graph is in place, we run Node2Vec. This algorithm performs many short biased random walks over our graph (biased by edge weight) and then feeds those walks into a standard skip-gram (word2vec) model. The result is a 64-dimensional embedding for each concept node that captures its "neighborhood" in the graph—concepts that often co-occur end up with similar vectors. Finally, we serialize that embedding model so we can later load it and query "most similar" concepts without retraining.

For our interface, we would want the user to type in a query, and a map appears with related concepts. To implement this, we wrote _find_best_nodes and similar_to functions, which found the best node for the query and found related nodes to a given node, respectively.

To improve the search and recommendation system for related concepts, we redesigned both the matching and neighbor retrieval functions to prevent repetitive and cyclic results. Initially, using the original similar_to() function based purely on Node2Vec's most_similar output, we encountered a looping problem: for instance, querying "Trump" would suggest "Eric Adams," but querying "Eric Adams" would immediately suggest "Trump" again. This happened because the model prioritized pure similarity without tracking previously seen nodes, resulting in redundancy and circular recommendation paths. To solve this, we updated the _find_best_nodes() function to return up to five candidate nodes based on substring or fuzzy matches rather than selecting only one. In the new similar_to() logic, the first candidate was treated as the "central" node, while the original query and central node were both recorded in a _seen_queries set to prevent them from resurfacing. Then, for each of the five candidates, we pulled the top ten most similar neighbors, pooled up to fifty neighbors, filtered out any already seen concepts, and re-ranked them by similarity score. Finally, we selected the top five freshest, unique neighbors for recommendation. If fewer than five valid neighbors remained after filtering, a warning would be issued, but the function would still return the maximum number of unseen nodes. This multi-step improvement ensured that suggestions remained diverse, relevant, and cycle-free, significantly enhancing the overall user experience of the belief graph exploration.

Figure 4. _find_best_nodes() and similar_to() outputs for query "trump"

```
Query: 'trump'
↳ Mapping 'trump' → node «Mr. Trump» (candidates: ['Mr. Trump', 'Trump DOJ',
'Donald Trump', 'Donald trump', "Trump's pick"])
 Related: ['the charges', 'the Trump Justice Department', 'Eric Adams case',
'the corruption case', 'the New York City mayor']
```

Story and Image Generator

After making the recommendation system, we wanted to take our project a step further by experimenting with AI-driven conspiracy theory and image generation to enhance the immersive quality of the "belief spiral" experience. Initially, we wanted to generate not only personalized conspiracy narratives but also accompanying visual "evidence," such as fake documents, photographs, and/or diagrams, to simulate the types of "evidence" commonly found in online conspiracy communities.

For image generation, we used Stable Diffusion through the diffusers library in Python, configuring the pipeline with the DPMSolverMultistepScheduler to improve generation speed without significantly compromising detail. Our prompts dynamically incorporated user-selected

concepts, aiming to create context-specific visuals such as “leaked government memo about [concept]” or “blurry surveillance photo of [location]”. However, we quickly ran into technical challenges. Despite extensive experimentation with prompt structures, guidance scales, and different models—including both SD 1.5 and SDXL—the outputs consistently failed to meet our expectations.

Stable Diffusion had a lot of difficulties in handling text and human figures. Many images included distorted, unreadable text and uncanny, half-formed faces or bodies. Attempts to generate fake documents or news clippings were unsuccessful, often resulting in layouts filled with nonsensical typography and visually incoherent elements. Examples of these challenges can be seen in the images we generated (Figure 6).

Despite parameter tuning—adjusting inference steps, CFG scales, and swapping between model checkpoints—the results remained very inconsistent and largely unusable. The visuals did not resemble credible conspiracy "evidence" nor did it contribute meaningfully to the user experience. Given these limitations, we ultimately decided to remove the image generation component from the final product, and focused instead on the conspiracy theory generation.

We iterated extensively on the story generation component. Initially, we leveraged Ollama to run lightweight language models capable of generating short-form conspiracy narratives based on user-selected concepts from the graph interface. While this provided a fast and efficient way to prototype, the generated stories often lacked depth, coherence, or the characteristic tone of real-world conspiracies. The stories often felt too formal or neutral, reading more like dry reports or generic AI-generated summaries rather than the more speculative, sensational, and provocative language typical of real-world conspiracies. To address this, we focused on prompt engineering. We designed prompts that encouraged the model to adopt the rhetoric commonly found in misinformation spaces. An example of a prompt we used is as follows:

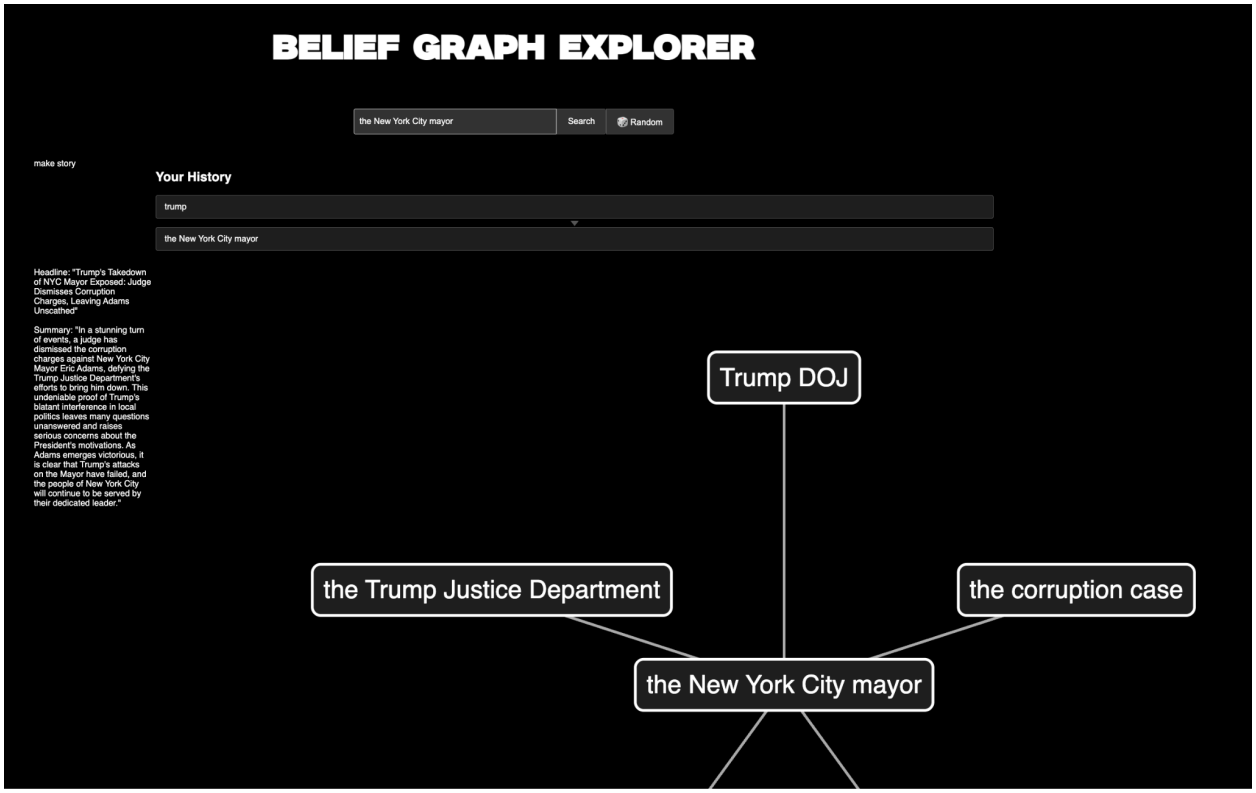
> "You are a world-renowned investigative journalist known for uncovering shocking truths. After reading the context below, craft an attention-grabbing headline followed by a juicy, persuasive summary. The headline should be sensational but believable, using words like 'Revealed', 'Shocking', 'Exposed', or 'Undeniable Proof'. The summary should be 3-4 sentences, dramatic, confident, and designed to captivate readers. Use powerful language, imply urgency, and avoid any qualifiers like 'might' or 'allegedly'."

This structure, combined with the user-selected concepts and summaries corresponding to those concepts, significantly improved the tone of the generated outputs—leading to headlines and summaries that felt more similar to the dramatic, assertive style typical of conspiracy

content. Despite these improvements, however, the base model still occasionally defaulted to safer or more formulaic language.

To further improve the consistency and realism of the narratives, we fine-tuned LLaMA 2 on a dataset of conspiracy-related texts (LIAR and FakeNewsNet). This allowed the model to internalize common linguistic patterns and stylistic features found in misinformation, reducing its reliance on heavy prompt engineering. Post-fine-tuning, the model was able to generate more persuasive and contextually relevant conspiracy stories—better capturing the speculative yet confident tone that makes such narratives compelling in the real-world.

Figure 5. Conspiracy graph working and builds on previous context



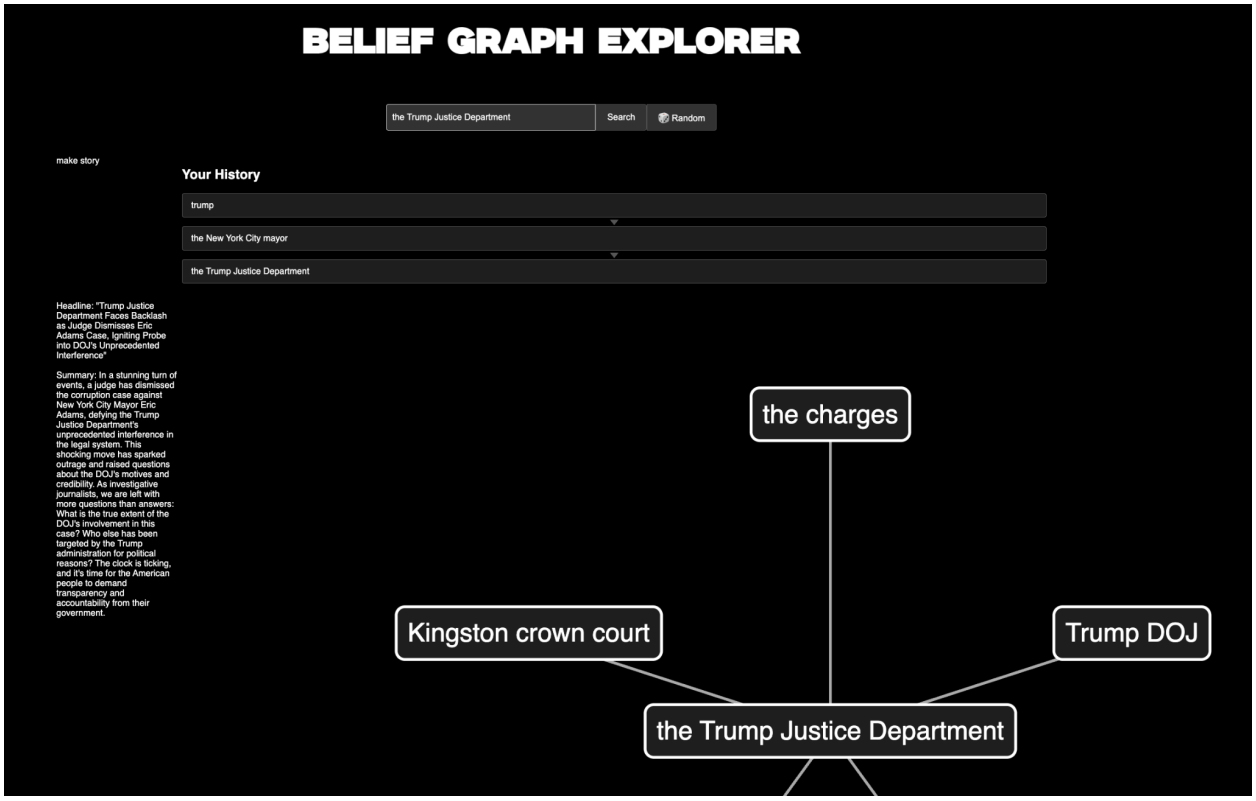
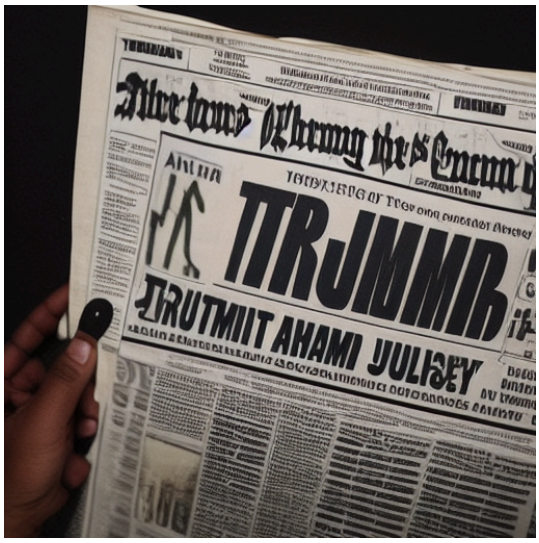
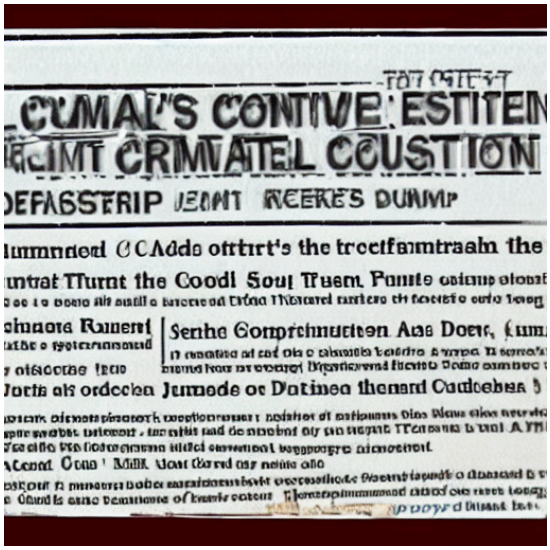


Figure 6. Attempted image generation results





User Interface

Creating the user interface was an interactive process. First make sure the core model worked — users should be able to input a belief, get back related concepts, and see them immediately. We began by building a basic text input form and a results section. This minimal setup allowed me to confirm the backend query flow: belief in → related beliefs out.

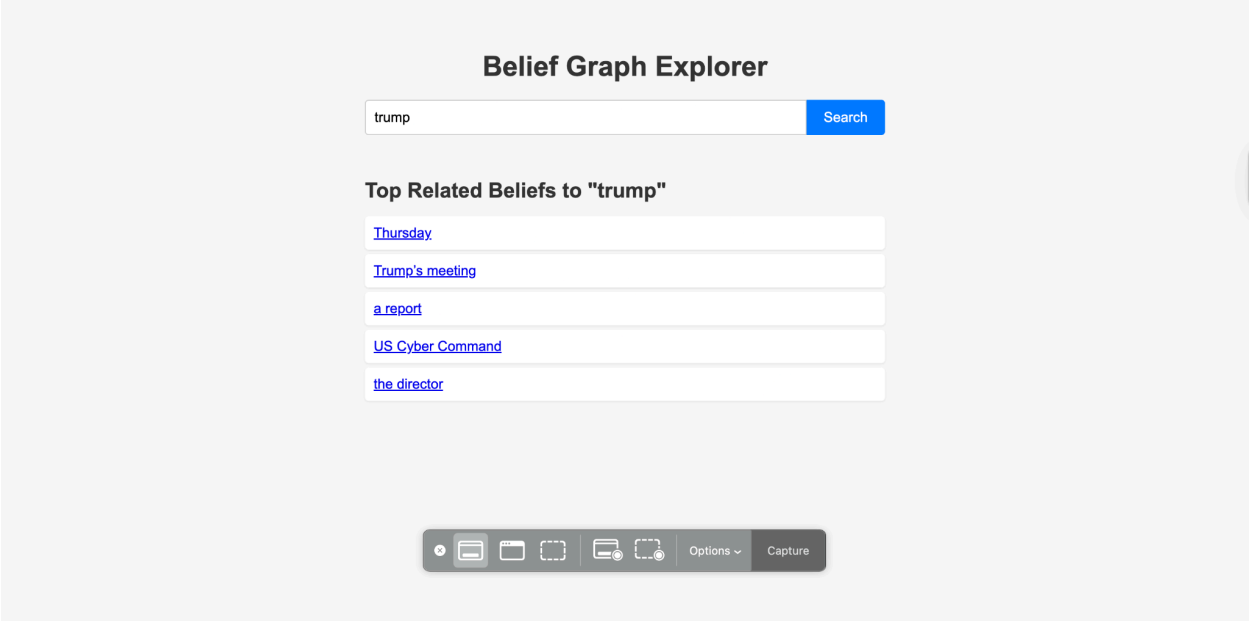


Figure X. First Iteration of the UI

Once the basics were working, we moved on to layering in the graph visualization. We chose to render the graph manually using SVG elements because it gave full control over the layout

without relying on heavy external libraries. we calculated node positions in a circle around the original belief, creating a clear, organized structure that still looked organic. Each belief became a clickable node, drawn with an SVG <circle> and labeled with text.

For interactions, we attached event listeners to each node so that clicking on a related belief would populate the search bar with the new belief and trigger a fresh search — simulating the "falling deeper into the rabbit hole" experience. We also kept track of the user's click history to build longer and longer conspiracy chains, allowing the graph to grow in complexity as users explored.

After that, we polished the UI by:

- Ensuring the graph would reset cleanly when starting a new search.
- Styling the input box, buttons, and graph
- Handling edge cases, like what to show if no results were found.

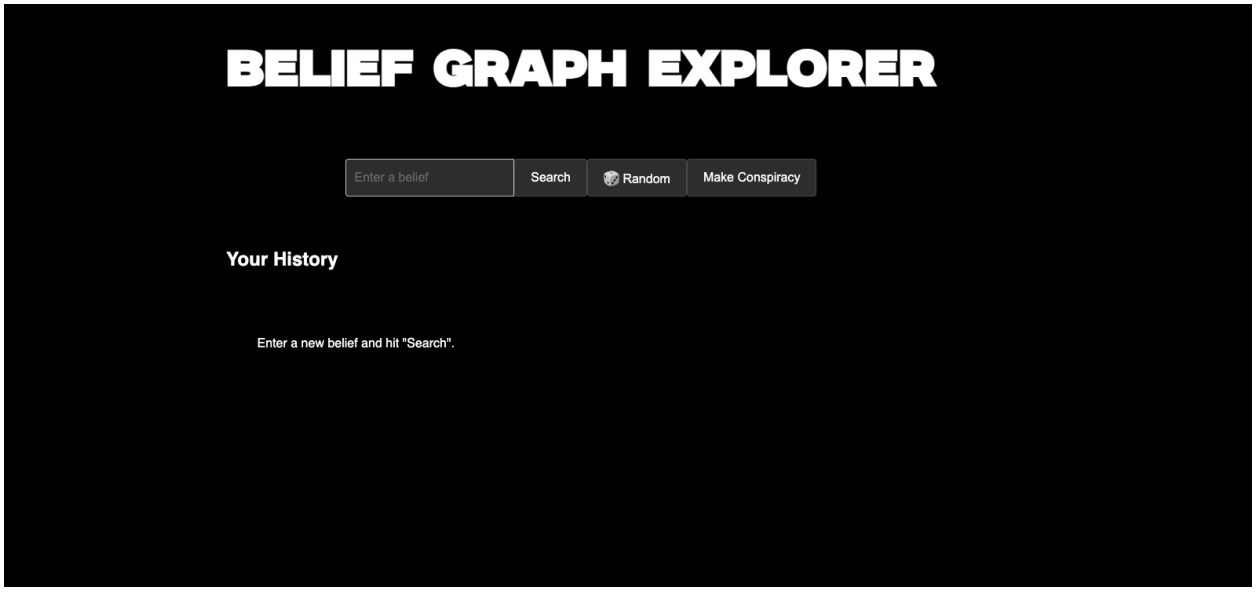


Figure X. Final UI Design

Reflection

In this project, we set out to replicate contemporary information ecosystems by using AI to simulate how disinformation can spread and escalate online. Our goal was to show that through simple techniques like node embeddings and language generation, it is possible to automate the creation and linking of conspiracy narratives. We chose the final version of our belief graph system because it best mirrored the subtle but powerful dynamics that occur in real-world algorithms, particularly how recommendations increasingly pull users toward more extreme or

fringe content. The belief spiral we created is not just a technical achievement but a commentary on the mechanisms that fuel echo chambers and radicalization in modern media environments.

Throughout the project, we were very satisfied with the overall structure of our ideas and techniques. Building the graph through concept co-occurrence, enhancing the search system to prevent looping, and layering a text-generation model on top of it felt both creative and technically robust. At the same time, the trials and errors we faced — especially around repeated recommendations and model loading — taught us a lot about the delicate balance between relevance and novelty in recommendation systems. New ideas also emerged during the process, such as the potential to simulate not just text but also visual "evidence" for conspiracy theories through image generation, reflecting how multimedia is often used to lend false credibility to fake narratives.

Ultimately, this project deepened our awareness of AI's ethical dimensions. While we enjoyed building a technically interesting and playful system, we also saw how easily the same techniques could be used unethically in the real world. The project serves as a small-scale demonstration of how AI systems, if left unchecked or poorly designed, could reinforce misinformation, polarize communities, and distort perceptions of truth. It made us reflect more seriously on the responsibility that comes with designing AI and recommendation systems — not just for accuracy or engagement, but for safeguarding public discourse itself. If we had more time with the project, we would try improving the website layout and UI/UX more.

RESULT

Walk through
https://drive.google.com/file/d/1505B58ueQWcHKpKWn-TnMpSeJUPzCxAz/view?usp=drive_link

Story generation
https://drive.google.com/file/d/1_W8Cqs8EinV9HDLmKbNNkWW5UkQ_A-ZW/view?usp=sharing

Randomized searches
https://drive.google.com/file/d/1WBx6xIG5FEqUTnonQ_HI8fGaA0hKqMB6/view?usp=sharing

CODE and/or TOOL LIST

https://github.com/aouyang2299/BeliefSpiral

Python, Colab

Node2Vec, networkx

ChatGPT, Ollama