Tracking the Origins of Digital Content

Leveraging Natural Language Processing to Identify Information Sources

Jinpeng Zhang, Ahmed Hasan, Zsombor Mátyás Kispéter, Sonaly Akther, Farzana Afrin, Rubens Rissi Onzi

Software Engineering, SW71E25, 2025-12

Master's Project


STUDENT REPORT
AALBORG UNIVERSITY

Electronics and IT
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITET
STUDENTERRAPPORT

Title: Information Origin Tracker (IOT)
Information Origin Tracker

Theme:
Digital Journalism

Project Period:
Fall Semester 2025

Project Group:
SW71E25

Participants:
Jinpeng Zhang
Ahmed Hasan
Zsombor Mátyás Kispéter
Sonaly Akther
Farzana Afrin
Rubens Rissi Onzi

Supervisor:
Suman Sourav

Copies: 1

Page Numbers: 97

Date of Completion:
December 21, 2025

Abstract:

This project addresses the critical issue of information transparency by developing a web application designed to trace the origins of digital content. The system analyzes the piece of information that is submitted by the user that can be either in form of text or URL. In a period where information spread rapidly, this tool helps users to asses the origin of their sources.

The system uses natural language processing to analyze long pieces of information and generate queries to determine the earliest source found. It has been built as a modular, service-oriented system and it consists of a web interface, a middleware server to validate requests, a web module for web scraping and an NLP module for analysis. To ensure scalability, it is deployed in kubernetes.

The outcome of the project is the web application **Information Origin Tracker**

Contents

Preface

The internet is over saturated with information, making it challenging for users to assess the provenance of digital content. This problem motivates our semester project: the "Information Origin Tracker."
We developed a model designed to provide transparency by analyzing user-submitted information and tracking their earliest identifiable source. This report outlines the technical and conceptual frameworks used for this tool, detailing the problems, our proposed solution, the architecture, our development process and the final evaluation of the overall system.

Aalborg University, December 21, 2025

Jinpeng Zhang
jzhang25@student.aau.dk

Ahmed Hasan
ahasa25@student.aau.dk

Zsombor Mátyás Kispéter
zkispe25@student.aau.dk

Sonaly Akther
saktther25@student.aau.dk

Farzana Afrin
mafrin25@student.aau.dk

Rubens Rissi Onzi
rrissi25@student.aau.dk

Part I

Introduction and Definitions

Chapter 1

Introduction

1.1 Project Overview

We live in an age defined by an immense amount of digital information that keeps increasing each day. While this has been a major step forward that allows everyone with an internet connection to access this kind of knowledge, it has concurrently created a critical trust deficit. The rapid spread of misinformation, disinformation and now AI-generated articles makes it even more difficult for the average user to determine whether or not the information they're reading is reliable.
This loss of trust challenges complicates media transparency and undermines the integrity of our shared knowledge base. This challenge motivated our semester project: the "Information Origin Tracker".
We developed a system designed to provide transparency by analyzing digital content from user-submitted web pages or raw text. The project is built to trace the earliest identifiable online source, empowering users with a tool to critically assess the credibility of the information they consume.

1.2 Problem Domain: Information Transparency

This project operates at the intersection of several critical domains:

- **Media Transparency:** Providing users with the tools to see the provenance of information, rather than simply trusting a source's authority.
- **Computational Journalism:** Applying computational methods to analyze and process media, in this case, to perform large-scale source-tracking.

The importance of this project addresses one of the most relevant issues of the modern information ecosystem. By allowing users to verify sources for themselves, our system directly enhances media literacy, and supports the principles of ethical journalism.

1.3 Problem Statement

The specific problem this project addresses is the fundamental lack of accessible and automated transparency in the provenance of online information.
For a typical user, it is often difficult or impossible to determine if a piece of information is reliable, especially when the publication website lacks clear sourcing. When confronted with a claim, users have no simple way to determine if it is an original, verified report or a piece of content that has been copied, altered and fabricated. The people who consume these types of content are our main target and will be offered an effective, indipendent mechanism to verify a claim's origin.
This leads to the following initial problem:

> **How can we design a system that determines the earliest known source of information from a text or webpage and additionally provides the user with a list of all fetched URLs and websites?**

Our project was designed to be a functional, scalable, and user-friendly answer to this question.

1.4 Project Scope & Objectives

The primary objective of this project was to design, build, and deploy a functional tool that provides users with the ability to trace the origin of a specific claim, which we define as the earliest source that our proposed solution found on the internet.
To ensure a deliverable product within the project's timeframe, we defined a clear Minimum Viable Product (MVP):

- A website where a user can input a piece of text or a URL.
- A backend system that processes this text, scrapes the web for its mentions, and analyzes the results.
- Returns a list of all the fetched URLs that mention the text and pinning the oldest one as the first result.

We also defined a **Maximum Ambition** for the project, representing features that were considered but deprioritized for the initial release:

- A browser extension for easier, in-context analysis.
- A further implementation that stores registered users and their queries for faster data retrieval.
- A more powerful webscraping technique that can handle multiple requests.
- Better LLM prompt engineering to prevent hallucinations or losing specific pieces of information.
- A scalable architecture supporting multiple users without performance bottlenecks.
- Give the user the possibility to choose which piece of information from the URL to process.

1.5 Project Outcomes and Deliverables

The successful execution of this project has resulted in a working software solution that addresses the initial problem of information transparency.

1.5.1 The Information Origin Tracker Application

The primary deliverable is the fully functional **Information Origin Tracker (IOT)** web application, a platform that allows the user to enter the desired piece of content and returns the origin source. The application fulfills the requirement defined in the MVP subsection 1.4.

- **User Interface:** A responsive frontend developed in React/Next.js that allows users to submit raw text or URLs for analysis.
- **Result Visualization:** A structured display of search results that pins the earliest source.

1.5.2 Service-Oriented Architecture (SOA) Infrastructure

The project is based on a robust architecture defined in the previous section, moving away from a monolithic structure the team delivered a modular backend deployed within a Kubernetes cluster.

- **API Gateway:** A middleware layer that acts as a coordinator, validatin all requests and managing communication between components.
- **Containerization:** Each module was containerized using docker, ensuring scalability and easier maintenance.
- **Storage:** A configured cloud and local postgreSQL database for smart caching and redundancy, ensuring high availability and fault tolerance.

1.5.3 Advanced NLP Web Scraping Modules

The core of the tracker resides withing the NLP and Web Search module, two specialized backend components:

- **NLP Pipeline:** Python based module leveraging a local LLM and dedicated libraries for natural language processing. This module transforms unstructured text into key word search queries.
- **Web Search Emulation:** Web Scraping technique using stealth sessions to avoid detection in the Bing search engine, parsing HTML results and filtering data.

1.6 Existing Solutions

The current market lacks a direct competitor to the Information Origin Tracker—a tool that returns the origin of a specific piece of digital content; however, there are tools available that partially cover this gap in the market. This section reviews these tools, categorizes their methodologies, and highlights the fundamental gap that our solution is designed to bridge.

1.6.1 Web Archiving and Historical Preservation

The closest product to our proposed solution is the **Wayback Machine**, a web archiving service that stores information:

- **Methodology:** The system periodically savess static snapshots of publicly accessible web pages at specific points in time.
- **Utility:** It is invaluable for citing historical context and preserving digital cultural heritage. Through this tool, users can verify which specific URL contained on a past date.
- **Limitations:** The **Wayback Machine** traces the history of a **URL**, not the content itself. It's not designed to search for the origin of a piece of information, they require the user to already know the URL beforehand and an approximate date.

1.6.2 Content Legitimacy and AI Detection Services

A second category of tools focuses on assessing the legitimacy, originality and authorship of content. Examples include **GPTZero** and sourcing aggregators like **Sourcely**.

- **Methodology: AI Detectors** employ linguistic analysis and probabilistic models trained on large language models (LLMs) to estimate the likelihood of AI generation. **Source Aggregators** cross-reference factual claims within text against established databases of academic papers, reputable news, or scientific journals.
- **Utility:** These tools provide a quality score (e.g., probability of human authorship) or supporting citations for a claim.
- **Limitations:** Their function is to judge the **integrity** of the content, not to trace its path backwards to its initial origin source.

1.6.3 The difference of the Information Origin Tracker

The **Information Origin Tracker** is fundamentally differentiated by its focus on **origin tracking** rather than archiving information and credibility scoring. Instead of simply evaluating the trustworthiness of content, our solution uncovers where the information came from. This capability is essential in the current digital landscape, as it helps user to assess for themselves the origin of a piece of information. Ultimately this solution provides a practical impact by empowering users with transparency and further supports informed decision-making, verification practices and resilience against misinformation.

Chapter 2

Definitions and Theory

Before detailing the system architecture, it is essential to lay down the concepts on which we built the proposed solution. This chapter introduces the key terms and particularly on how this project defines the origin of a piece of information.

2.1 Origin

The definition of the origin of a piece of information is fundamental to this project. It is the main point of information provenance, which represents the chronology of the origin, development, ownership, location, and changes to a system or system component and associated data. [1]

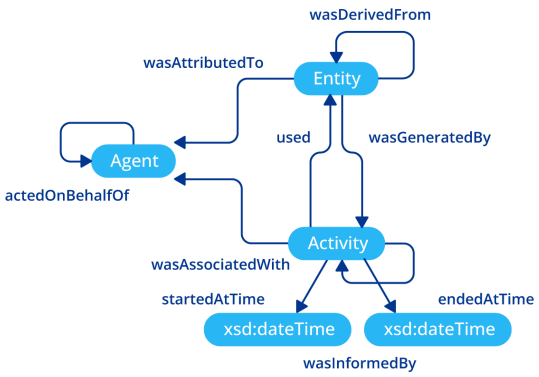

Figure 2.1: Data Provenance Diagram [1]

However, identifying the true point of origin of digital contents present significal challenges. Information is replicated immediately, shared across multiple platforms, modified, rewritten and changed based on the writer's view on the topic making it almost impossible to find the point from which the information originated from. Additionally some

sources could also be deleted and be unreachable adding another layer of difficulty in this challenge.
Given these assumptions, limitations and constraints of the available resources, our project adopts a more operational and achievable definition of origin. Instead of attempting to pinpoint the first occurrence of a piece of information we narrowed it down to the following definition:

- **The earliest identifiable source retrieved by our system in which the information has been mentioned.**

This definition of origin provides a foundation for the project and makes it possible to achieve the goal was initially set.

2.2 News and Website Definitions

During the development of the Web Search Module, the component was designed to scrape and return two distinct sets of result. These results are labeled as "news" and "websites", although they don't align with the traditional definitions, they are redefined within the context of this project to simplify the interpretation:

- **News:** all sources with an identifiable publishing date
- **Website:** all sources without an identifiable publishing date

This distinction allows us to easily showcase different results to the user, for a better assessment of the sources.

2.3 Cosine Similarity

Cosine similarity is a metric system used for measuring the semantic similarity between two vectors. In the context of Natural Language Processing texts are typically represented as numerical vectors through embedding, where semantic meaning is encoded.
Given two vectors $\vec{a}$ and $\vec{b}$, cosine similarity measures the cosine of the angle θ between them. It is defined as:

$$\text{CosineSimilarity}(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \, \|\vec{b}\|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \, \sqrt{\sum_{i=1}^{n} b_i^2}} \tag{2.1}$$

where:

- $\vec{a}$ and $\vec{b}$ are the vector representations of two text inputs.
- $\vec{a} \cdot \vec{b}$ denotes the dot product of the vectors.
- $\|\vec{a}\|$ and $\|\vec{b}\|$ represent the Euclidean norms of the vectors.

The resulting similarity score ranges from -1 to 1, where 1 means identical and -1 opposite.

2.4 Large Language Models

Large language models, also known as LLMs, are very large deep learning models that are pre-trained on vast amounts of data. The underlying transformer is a set of neural networks that consist of an encoder and a decoder with self-attention capabilities. The encoder and decoder extract meanings from a sequence of text and understand the relationships between words and phrases in it. [2]

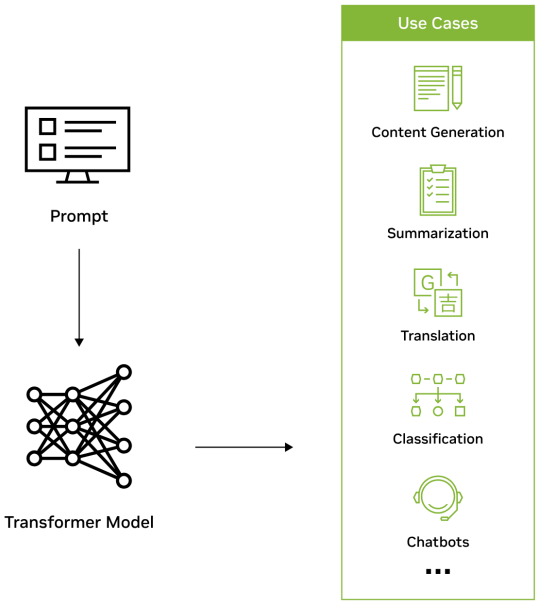

Figure 2.2: LLM Chart [3]

LLMs were initially focused on perception and understanding, however in recent times they've been trained on internet-scale datasets with hundreds of billions of parameters and have now unlocked the ability to generate human-like content. Models can read, write, code, draw, and create in a credible fashion and augment human creativity and improve productivity across industries to solve the world's toughest problems. The applications for these LLMs span across a plethora of use cases. For example, an AI system can learn the language of protein sequences to provide viable compounds that will help scientists develop groundbreaking, life-saving vaccines. Or computers can help humans do what they do best—be creative, communicate, and create. A writer suffering from writer's block can use a large language model to help spark their creativity. Or a software programmer can be more productive, leveraging LLMs to generate code based on natural language descriptions. [3]
Due to the recent advancements in the field, more sophisticated LLMs were developed and made accessible to a broader audience. Given the current landscape we were able to access and use smaller LLMs (such as models with 4B parameters) and utilize them for our project.

Chapter 3

Background

This chapter outlines and describes the technologies that were used to form the foundation of the system.

3.1 Web Data Retrieval Concepts

Retrieving data from the internet generally falls into two categories: utilizing official APIs or emulating web browsing behavior.

3.1.1 Web Search APIs (Free Tier Solutions)

Web Search APIs provide a structured and efficient mechanism for data retrieval, simplifying data extraction process by abstracting the complexities of interacting with web page layouts. However the free-tier accounts impose constraints that result in issues during data collection, the primary restrictions are limited requests and incompatibility with parallel processing. They also lack advanced features such as sorting and filtering, resulting in limited control over acquired data. Additionally queries may also need to be refined through natural language processing.

3.1.2 Web Search Emulation

he alternative approach is to programmatically simulate web search operations. This method involves controlling a web browser or simu- lating HTTP requests to perform searches and extract data directly from the Search Engine Results Pages (SERPs). This technique offers greater scalability and flexibility than the free-tier APIs mentioned earlier

3.2 Container Orchestration

To develop a modular solution that promotes Service Oriente Architecture (SOA), we chose docker and containerization as the primary technology for deployment. This approach provides isolated environments for the services, preventing conflicts across the modules.

To manage this system the two primary solutions for Container Orchestration are Docker Swarm and Kubernetes.

3.2.1 Docker Swarm

Docker Swarm is a container orchestration tool used in a containerization approach for managing and deploying applications. It allows you to create a cluster of Docker nodes and deploy services to the cluster.This is a lightweight and straightforward solution, allowing you to containerize and deploy your software project in minutes.[4]
To manage and deploy applications efficiently this approach offers the following advantages:

- **Cluster Management**: You can create a cluster of nodes to manage the application containers and each node can be a manager or a worker (or both).
- **Service Abstraction**: Docker Swarm introduces the "Service" component which defines network configurations, resource limits and number of replicas.
- **Load Balancing**: Swarm offers built-in load balancing, which allows services inside the cluster to interact automatically without intervention.
- **Rolling Back**: Rolling back to a specific service to its previous version is fully supported in case of a failed deployment.
- **Fault Tolerance**: Docker Swarm automatically checks for failures in nodes and containers to generate new ones to allow users to keep using the application seamlessly.
- **Scalability**: With Docker Swarm, you have the flexibility to adjust the amount of replicas for your containers easily.

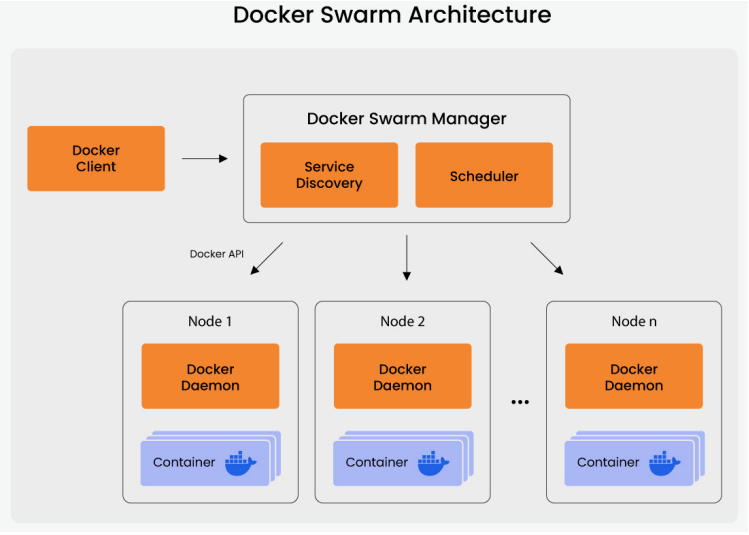

Figure 3.1: Docker Swarm Architecture [5]

In summary, the cluster consists of important components that work together to manage and deploy efficiently. The manager nodes receive commands from the client and assigns a worker node. Inside each manager there are the Service Discovery ad Scheduler component that have the goal to collect information from the worker node. The worker nodes are only responsible for executing the tasks assigned.

3.2.2 Kubernetes

Kubernetes is a container orchestration tool for managing and deploying containerized applications. It supports many container runtimes such as Docker, Containerd, Mirantis and more. This is a feature-rich and highly configurable solution which makes it ideal for deploying complex applications, but requires a steep learning curve.[4]

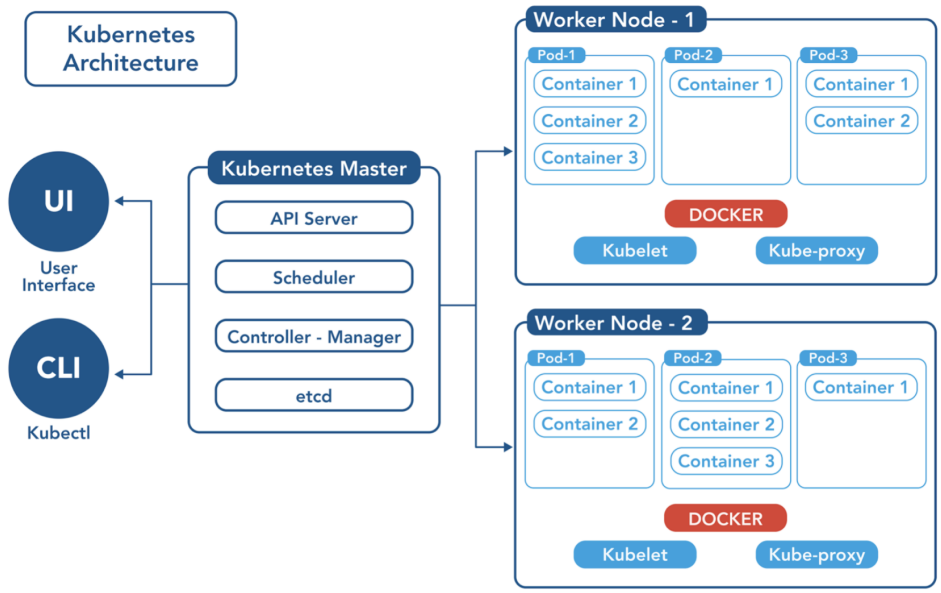

Figure 3.2: Kubernetes Architecture [6]

Some of the key features:

- **Container Orchestration with Pods**: A pod is the smallest and simplest deployment unit. Kubernetes allows you to deploy applications by creating and managing one or more containers in pods.
- **Service Discovery**: Kubernetes allows containers to interact with each other easily within the same cluster.
- **Load Balancing**: Kubernete's load balancer allows access to the group pods via external network, the clients from outside can use the balancer's external IP to to access the pods running inside the cluster. If any pod fails or goes down the client requests will be automatically forwarded to other available pods.

- **Automatic Scaling**: it is possible to scale-in or scale-out by simply adjusting the number of deployed containers.
- **Self-Healing**: If a pod fails or goes down the system will automatically restart and recreate the pod.
- **Persistent Storage**: The persistent storage helps storing information beyond the life-cycle of the pods.
- **Configuration and Secrets Management**: Through configuration files it is possible to manage and store secrets securely.
- **Rolling back**: If one deployment encounters some development issues it is possible to transition to a previous and functioning version of the application.
- **Resource Management**: The resources constraints can be set when deploying.
- **Extensibility**: Kubernetes is not bounded by its API unlike Docker Swarm, making it a highly extensible system that supports additional plugins and tools.

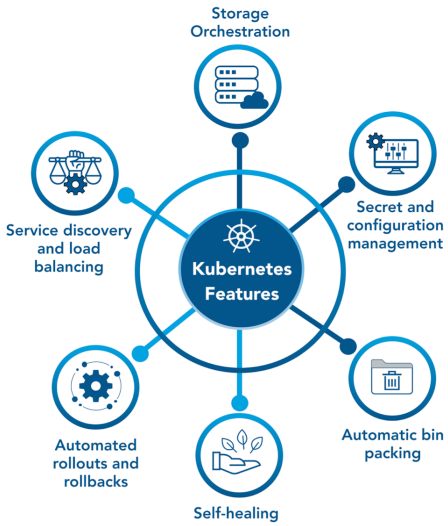

Figure 3.3: Kubernetes Features

Part II

Approach and Architecture

Chapter 4

Approach

This chapter outlines the system architecture designed to ensure scalability, efficiency and reliability. It also describes the technologies used in each module and explains how they interact with each other.
The project's purpose was discussed and we identified the technologies best suited to achieve our goal before starting the development process. Since the project is a system designed to retrieve the earliest source given a piece of information we had to define several key aspects:

- **UI:** We selected React and Next.js for the user interface due to their component-based architecture, fast rendering, and strong ecosystem support.
- **Web Scraping:** For data retrieval, we initially considered using free-tier APIs (such as Brave Search API) but ultimately opted for web emulation with stealth requests.
- **Data Analysis:** The input goes through Natural Language Processing techniques to extract relevant context and identify search parameters.
- **Generated Queries:** To generate efficient queries, we integrated a fast and lightweight LLM (Qwen-3-4b-instruct-2057).
- **Communication:** We introduced middleware to validate and manage requests between the frontend and backend modules.
- **Scalability:** We adopted a service-oriented architecture (SOA) to ensure scalability, modularity, and fault tolerance.
- **Security:** Core security mechanisms include authentication, password encryption and Multi-Factor Authentication(MFA).

This overview defines the key components and guide lines for the project's implementation an development.

4.1 Technology & Stack justification

The selection of the technology stack was defined by the initial goals, system's requirements, performance and scalability needs. Each module was designed to work with a slightly loose coupling between these services, promoting the SOA architecture.

4.1.1 Frontend Module

We chose React and Next.js due to their modular structure, server-side rendering capabilities, and integration with API backends. Another deciding factor were the team members' capabilities and the technologies they were most comfortable working with.

4.1.2 Web Search Module

Retrieving online sources proved to be challenging due to the nature of web scraping and the diversity of online data structures. This issue was addressed in detail in Section 3.1, therefore a test was conducted for both approaches:

- **Free-tier WebSearch API**: The test was conducted using the free tier Brave Search API with the following query and parameters:

```
q = ‘‘Trump sick’’
result_filter = "news’’
freshness = ‘‘1990-01-01to2025-09-22"
```

Results:


```
{
  "query": {
    "original": "Trump sick",
    "show_strict_warning": false,
    "is_navigational": false,
    "is_news_breaking": false,
    "spellcheck_off": false,
    "country": "us",
    "bad_results": false,
    "should_fallback": false,
    "postal_code": "",
    "city": "",
    "header_country": "",
    "more_results_available": true,
    "state": ""
  },
  "news": {
    "type": "news",
    "results": [
      {
        "title": "President Trump warned Charlie Sheen's former in-laws before socialite daughter's ill-fated marriage: doc",
        "url": "https://www.foxnews.com/entertainment/president-trump-warned-charlie-sheens-former-in-laws-before-socialite-daughters-ill-fated-marriage-doc",
        "is_source_local": false,
        "is_source_both": false,
        "description": "Charlie Sheen&#x27;s engagement to Brooke Mueller faced objections from family and friends, including his father Martin Sheen, as revealed in the new Netflix documentary.",
        "page_age": "2025-09-20T13:25:02",
        "fetched_content_timestamp": 1758547614,
        "profile": {
          "name": "Fox News",
          "url": "www.foxnews.com",
          "long_name": "foxnews.com",
          "img": "https://imgs.search.brave.com/8Y6YYaSHf4PweCQwcB2IRvUWPCEsdNoek5FtyWqRRW4/rs:fit:32:32:1:0/g:ce/aHR0cDovL2Zhdmlj/b25zLnNlYXJjaC5i/cmF2ZS5jb20vaWNv/bnMvOGIyN2Q2NTE4/NGRmOTVmZGNmMzEy/OWJjY2MwNzkzYzg0/YWNhOGFmY2UxYTQ1/NmZiM2JkYzcxMjFh/MDFmNDBkYy93d3cu/Zm94bmV3cy5jb20v"
        },
        "family_friendly": true,
        "meta_url": {
          "scheme": "https",
          "netloc": "foxnews.com",
          "hostname": "www.foxnews.com",
          "favicon": "https://imgs.search.brave.com/8Y6YYaSHf4PweCQwcB2IRvUWPCEsdNoek5FtyWqRRW4/rs:fit:32:32:1:0/g:ce/aHR0cDovL2Zhdmlj/b25zLnNlYXJjaC5i/cmF2ZS5jb20vaWNv/bnMvOGIyN2Q2NTE4/NGRmOTVmZGNmMzEy/OWJjY2MwNzkzYzg0/YWNhOGFmY2UxYTQ1/NmZiM2JkYzcxMjFh/MDFmNDBkYy93d3cu/Zm94bmV3cy5jb20v",
          "path": "› entertainment › president-trump-warned-charlie-sheens-former-in-laws-before-socialite-daughters-ill-fated-marriage-doc"
        },
        "breaking": false,
        "is_live": false,
        "thumbnail": {
          "src": "https://imgs.search.brave.com/U1aQPaBqlNyrnW5pSydATk9UcJ_N8zgukFutoWalYTI/rs:fit:200:200:1:0/g:ce/aHR0cHM6Ly9zdGF0/aWMuZm94bmV3cy5j/b20vZm94bmV3cy5j/b20vY29udGVudC91/cGxvYWRzLzIwMjUv/MDkvYnJvb2tlLW11/ZWxsZXItY2hhcmxp/ZS1zaGVlbi1lbW15/cy5qcGc",
          "original": "https://static.foxnews.com/foxnews.com/content/uploads/2025/09/brooke-mueller-charlie-sheen-emmys.jpg"
        },
        "age": "2 days ago"
      }
    ],
    "mutated_by_goggles": false
  },
  "type": "search"
}
```


Figure 4.1: Brave Search API test result

- **Web Search emulation**: The test was conducted through the implementation of web emulation searches on the bing search engine with the same query as the Brave Seach API test:


```
[
  {
    "title": "Watch: Trump to make a televised announcement after deluge of social media rumors",
    "link": "https://www.usatoday.com/story/news/2025/09/02/trump-update/85935964007/",
    "source": "USA TODAY",
    "date": "2025-09-02T16:10:00+00:00"
  },
  {
    "title": "Trump brushes off health speculation: 'It's sort of crazy'",
    "link": "https://www.msn.com/en-us/politics/government/trump-brushes-off-health-speculation-it-s-sort-of-crazy/ar-AA1LJYKn",
    "source": "MSN",
    "date": "2025-09-03T20:22:00+00:00"
  },
  {
    "title": "Trump, who questioned his opponents' health, rebuffs rumors about his own",
    "link": "https://www.nbcnews.com/politics/donald-trump/trump-rebuffs-health-speculation-golf-hand-bruise-biden-clinton-rcna228542",
    "source": "NBC News",
    "date": "2025-09-04T12:48:00+00:00"
  },
  {
    "title": "Trump's Medicaid cuts will hit some children's hospitals",
    "link": "https://www.npr.org/sections/shots-health-news/2025/09/09/nx-s1-5532133/medicaid-children-hospital-closure",
    "source": "NPR",
    "date": "2025-09-09T20:44:00+00:00"
  },
  {
    "title": "'GOP is getting sick of Trump': Republican lawmaker's 'act of war' comment raises alarms",
    "link": "https://www.msn.com/en-us/news/politics/gop-is-getting-sick-of-trump-republican-lawmaker-s-act-of-war-comment-raises-alarms/ar
      -AA1MgBpE",
    "source": "Raw Story",
    "date": "2025-09-10T11:30:28+00:00"
  },
  {
    "title": "'They Were Sick, They Were Stealing': Trump Loses His Cool, Insults VA Employees During Outlandish Remarks At The White House",
    "link": "https://www.msn.com/en-us/news/politics/they-were-sick-they-were-stealing-trump-loses-his-cool-insults-va-employees-during-outland
      -remarks-at-the-white-house/vi-AA1MrF9S",
    "source": "Slingshot News",
    "date": "2025-09-12T18:37:01+00:00"
  },
  {
    "title": "Ron Faucheux: The politics of Donald Trump's health",
    "link": "https://www.nola.com/opinions/ron_faucheux/biden-kennedy-wilson/article_972da492-fc12-4232-9994-9b16ac65a849.html",
    "source": "NOLA.com",
    "date": "2025-09-12T22:45:00+00:00"
  },
  {
    "title": "The Real Reason Trump Lost It After Kirk's Death: Author",
    "link": "https://www.msn.com/en-us/news/world/the-real-reason-trump-lost-it-after-kirk-s-death-author/ar-AA1MwV0r",
    "source": "The Daily Beast",
    "date": "2025-09-14T16:49:33+00:00"
  },
  {
    "title": "'Sick and Disturbing': Nancy Mace Demands Trump Admin Pressure Schools to Fire Teachers for Celebrating Kirk Murder",
    "link": "https://www.msn.com/en-us/news/politics/sick-and-disturbing-nancy-mace-demands-trump-admin-pressure-schools-to-fire-teachers-for
      -celebrating-kirk-murder/ar-AA1MBwJp",
    "source": "Mediaite",
    "date": "2025-09-15T18:39:49+00:00"
  },
```


Figure 4.2: Bing Search emulation test result

Conclusion: Given the results of our evaluation, we decided to pursue the web emulation solution. Although the web search APIs are convenient, they presented too many constraints for our project, such as returning results only from news sites, lack of chronological order, limited amount of requests and also incompatible with a parallel system which prevents scalability. The web search emulation is more complex to implement but offers scalability and parallelism. Python was chosen as the implementation language due to the open-source libraries available.

4.1.3 Natural Language Processing Module

This module was developed in python, as it provides a substantial amount of libraries and tools designed for natural language processing and the integration of LLMs. This module analyzes the contents provided and extract relevant information from the text. For this

project, a small LLM with 4 billion parameter was chosen to balance performance and computational efficiency.

4.1.4 Middleware API Layer

This layer ensures communication between the user and the backend services, by performing request validation, routing and data flow across modules. Developed in Java using Spring Boot, it was chosen for faster development and maintainable code, as the team member responsible had experience and was more comfortable with this technology.

4.1.5 Database

PostgreSQL was implemented for smart caching, as it is a structured relational database it provides faster and reliable querying for the data we store. Since the data is not complex or unstructured, a NoSQL solution was unnecessary, making PostgreSQL the best solution for our project.

4.1.6 Deployment and architecture

Two different architectural approaches were considered to ensure a reliable and scalable system:

- Kubernetes
- Docker Swarm

To determine which solution was better suited for our project we had to evaluate both possibilities.
As detailed in Section 3.2, despite the simple configuration and setup of docker swarm, we decided to adopt the Kubernetes solution as it better aligned with our project's goals.

Chapter 5

Architecture

After the different technologies to implement were defined, the next step was to design the system architecture to better understand how data flows and how the components communicate with each other. As mentioned earlier, the architecture follows a SOA layout, where the user can access the frontend and through that module it sends requests to the API layer.
All requests pass from the frontend to the API layer which act as an entry point. This gateway validates every request and routes them to the appropriate service, managing the communication logic between modules. These backend services will then process and elaborate the required tasks and return the results back to the API Gateway. Finally this gateway will then send the processed data to the frontend, where the results are rendered to the user.
The entire system is deployed inside a Kubernetes cluster, this environment provides container orchestration, scalability and reliability, ensuring that each service can be scaled independently and the architecture stays robust.

5.1 Base Architecture

The diagram in the figure 5.1 below provides an overview of the high-level layout of the system before the implementation. It captures the main components and the communication between the independent modules, serving as a reference for later design decisions and development process.

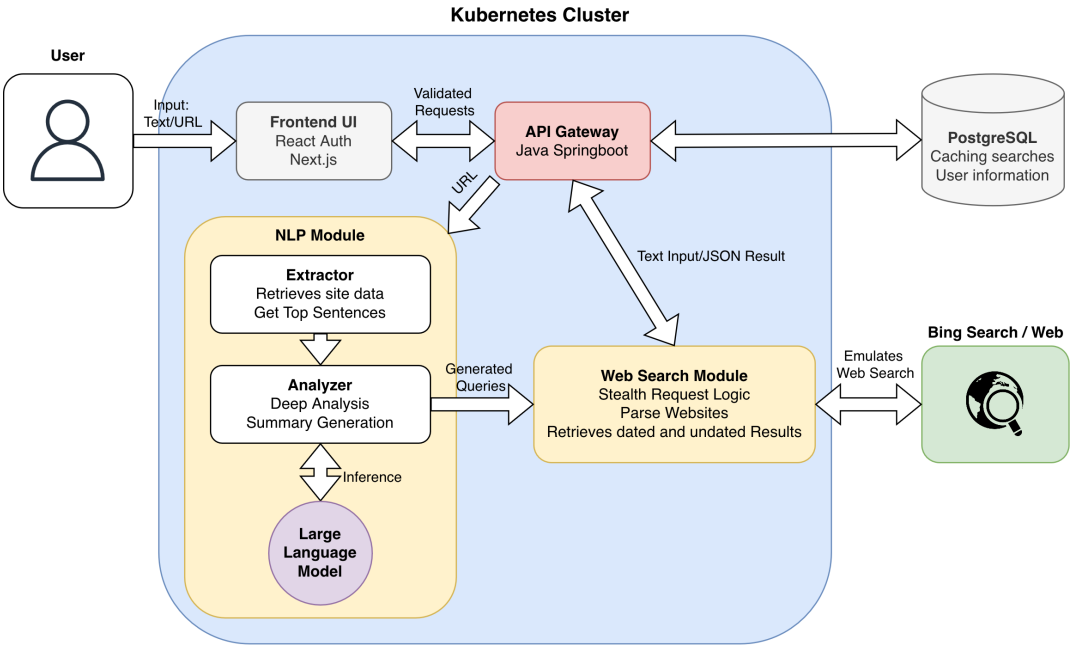

Figure 5.1: Architecture draft

5.2 Service-Oriented Architecture (SOA)

Service-oriented architecture (SOA) is a software design approach that focuses on building functional, scalable software systems from individual components, called services. Services can interact with one another to perform tasks, such as allowing someone to sign in once and access a variety of business applications. In SOA, the emphasis is on modularity, reusability, and interoperability—when businesses break complex applications down into smaller, more manageable building blocks, the result is greater flexibility and scalability. [7]

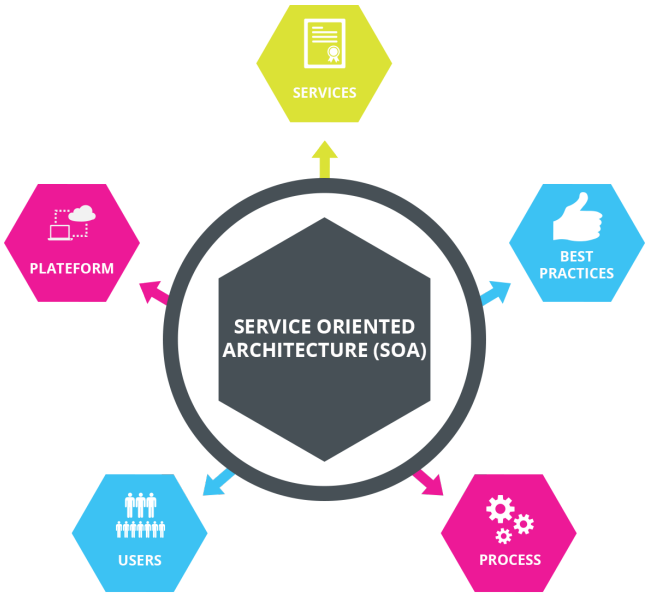

Figure 5.2: Service Oriented Architecture diagram [8]

With SOA, individual services are loosely coupled and can communicate and exchange data as needed. This type of architecture creates benefits, including:

- Faster development cycle: no need to build functions from scratch because of the existing services that can be repurposed
- Easier maintenance: each service can be maintained individually
- Adaptability: it is possible to implement older services in newer systems or replace dated services with newer ones.
- Scalability: in case of increased demand new instances of a service can be created to improve performance.

Our proposed solution is composed of different modules that communicate only by exchanging data and sending requests. For this purpose we adopted a Service-Oriented Architecture.

5.3 Pipeline and Processing Logic

The system processes the information by abstracting the complexity of individual components from the user. The middleware layer acts as a coordinator that manages the exchange of data between the different modules:
The processing logic can be summarized in a three-stage transformation:

1. **Initial Analysis:** The input is processed through the NLP module if necessary and sent to the Web Search module.
2. **Queries:** Based on the results given by the NLP module this component emulates a web search.
3. **Retrieval:** The search queries return a list of results found by the Web Search component and aggregates them into a unified response for the frontend.

This pipeline defines the expected behaviour and communication of each component, living room for improvement and performance enhancements.

5.4 Limitations & Assumptions

The overall architecture introduces demanding requirements that the system would need to meet in order to perform accordingly.

Resource limitations and Hardware Constraints:
The performance of the NLP module, which occasionally relies on the local LLM, is bounded by the computational power of the host machine, specifically GPU memory. This constraint defines the size of the LLM used for processing.

Tier	CPU	GPU (VRAM)	RAM	LLM Size Supported	Expected Performance
Basic	Intel i5 / AMD Ryzen 5	RTX 3060 (12GB), RTX 4060 Ti (16GB), RX 6700 XT	32GB	3B–7B models (4-bit or 8-bit quantized)	Smooth inference for small models and moderate token speed
Intermediate	Intel i7/i9 or AMD Ryzen 7/9	RTX 3080 (10GB), RTX 4080 (16GB), RTX 3090 (24GB)	64GB	13B–30B models (4-bit), 13B (8-bit), limited 30B (8-bit on 24GB VRAM)	Good performance with mid-sized models, minimal quantization, better multi-tasking
Advanced	AMD Threadripper / Intel Xeon	RTX 4090 (24GB), A6000 (48GB), A100 (40–80GB)	128GB–256GB	34B–70B+ models (4-bit or partial 8-bit), full-size 70B on 48GB+ VRAM	High-speed inference, large context windows, fine-tuning

Table 5.1: Example of hardware requirements for running local LLMs. [9]

Large Language Model:
Although modern models have been thoroughly tested and evaluated they still exhibit hallucination issues under certain circumstances.

Web Data:
The web search module is implemented to extract information from the structure of Bing's html. However the layout can be subject to frequent changes and modifications that could make the extraction logic unreliable.

Search Engine:
The search queries generated by the web search module, even if accurate, do not guarantee accurate results or correct information.

Web Scraping:
Potential IP blocking and anti bot detection during web searching emulation and data extraction.

Region Based Search:
Bing search engine might perform research on other regions, returning results in a different structure.

Part III

Project Scope and Management

Chapter 6

Challenges & Project Scope

This chapter outlines the challenges the group encountered throughout the development cycle and describes the overall project's development trajectory, establishing the minimum viable product (MVP) and defining potential improvements for a more ambitious version of the system.

6.1 Challenges

Because the team members had different levels of expertise and experience in distinct areas, the initial implementation phase was characterized by a steep learning curve. Each member had to quickly adapt and acquire new knowledge and skills in fields they were unfamiliar with, this ensured the overall functionality of the project.

The main areas that presented the greatest challenges were:

- **Scalable Architecture:** Kubernetes was chosen as the architectural approach, however in order to adopt this complex but robust and highly available solution, overcoming a steep learning curve was required. The main challenges included:
    1. **Cluster Deployment:** Establishing a multi-node Kubernetes environment using `kubeadm` and `kubectl`, navigating the complexities of container orchestration.
    2. **Network Configuration:** Implemented Calico as the Container Network Interface (CNI) to manage secure inter-pod communication and define traffic policies.
    3. **Resource Optimization:** Scaling up services to improve performance.
- **Natural Language Processing and LLM Optimization:** Researching and integrating a Large Language Model (LLM) given the hardware constraints.
  Difficulties included:
    1. **Research and selection:** The available machines runnin the system have GPUs with 8GB of VRAM, so a lightweight model was optimal for a good performance while also producing high-quality resutls.
    2. **Prompt engineering:** Large Language Models require human input that provides context and clear instructions so they can perform tasks accurately.

- **Web Search Emulation and Data Volatility:** The core objective of data retrieval online was hindered by the unpredictable layout change of the web structure, preventing scraping techniques:
    1. Implementing stealth techniques to avoid triggering anti-bot measures, CAPTCHAs, and IP bans from search engines.
    2. Developing resilience to Search Engine Results Page (SERP) layout changes.

6.2 Minimum Viable Product (MVP)

Despite the technical challenges listed, the project successfully achieved and delivered a Minimum Viable Product (MVP).

Core Deliverables of the MVP The delivered MVP constitutes a fully operational web application with the following guaranteed functionalities:

- A responsive user interface built on React/Next.js.
- A synchronous processing pipeline coordinated by the Spring Boot Middleware.
- Accurate transformation of input into short and optimized search queries using the lightweight LLM.
- Functional web search emulation for data retrieval.
- Smart caching of successful queries and results using PostgreSQL to minimize redundant operations.

In summary, the MVP establishes a functional solution demonstrating the project's core objective.

6.3 Maximum Ambition

The Maximum Ambition defines the roadmap for the system's future development, this vision aims to a highly available and fault-tolerant architecture.

Enhancements for Future Development The key areas for expansion and improvements include:

1. **Additional features:**
    - Implementation of web browser extension with the same functionalities of the web browser.
    - User manual selection of the search queries inside the website.
2. **Advanced Scalability and Redundancy:**
    - Implementing parallel operations across all services to handle high-throughput demands simultaneously.

- Transitioning to a multi-cluster and multi-node deployment model to ensure fault tolerance.
- Developing a redundant system for the API gateway and database to eliminate single points of failure.

3. **Enhanced LLM and Processing Capabilities:**
   - Integration of faster and significantly larger LLMs, potentially utilizing cloud resources or more powerful machines.
4. **Superior Data Retrieval and Security:**
   - Developing better web search capabilities by integrating multiple commercial APIs or advanced distributed scraping frameworks.
5. **Security:**
   - Strict implementation of Kubernetes Network Policies and Zero Trust principles.
   - Integration of Automated Vulnerability Scanning in the deployment pipeline to check all container images for CVEs.
   - Adoption of a SIEM solution for real-time monitoring and alerting on security-related events across all pods.
   - Enforcement of data integrity between services using Cryptographic Signatures (HMAC) on the payload data.

This ambitions seeks to achieve and create a robust, highly available and fast system capable of handling considerable amount of load while also providing reliable results.

6.4 Final Product

As mentioned in the section 1.5, the project reached the MVP goals defined at the beginning of the development cycle. Furthermore, additional features and capabilities were integrated to enhance the system:

- **Email service:** automated communication through a SMTP based service layer.
- **Redundant pv:** ensuring high availability and fault tolerance via replicated local storage.
- **Multi-cluster service:** addresses the single point of failure issue by expanding the system through deployment of multiple Kubernetes clusters.
- **Multi factor authentication (MFA):** Through the email service, users can enable additional security.
- **One-time password (OTP):** provides authentication tokens.
- **User search history:** implements tracking of user actions.
- **Multi-node system:** implementation of additional GPU nodes for parallel processing.

Chapter 7

Project Management

This chapter provides detailed information about roles within the group and how they were divided to ensure balanced and equivalent workloads for all members.

7.1 Team Roles and Responsibilities

The project team consists of six members, organized to cover all the different modules of the system.
The following table highlights the specific responsibilities assigned to each member.

Role	Primary Responsibilities
Team Lead / Tech Lead Jinpeng Zhang	Leading the overall system architecture, conducting code reviews, managing merges, assigning tasks to team members, and addressing critical infrastructure challenges while resolving cross-module communications. Responsible for developing the web-search modules, designing and deploying Kubernetes architecture, and establishing a reliable deployment workflow through tunneling and pooling. Additionally evualuated and benchmarked the final product, assessing LLM accuracy and service response time.
NLP Engineer Zsombor Mátyás Kispéter	Responsible for the research, development and integration of the large language model, including the implementation of the NLP module and data extraction components. Developed the LLM-based summarization, named-entity recognition, and top-sentence extraction and classification. Additionally developed browser extension and OTP system.

Role	Primary Responsibilities
Middleware Developer Ahmed Hasan	Backend API development and message routing, acting as the central layer connecting the different modules to ensure seamless communication and data consistency. Designed the database architecture and implemented smart caching strategy for better performance.
Frontend Developer Sonaly Akther	Design and implementation of user interface, focused on user experience, data visualization and responsiveness of the web application. Integrated the frontend with backend APIs to enable dynamic data flow and added an authentication system to ensure secure access.
Research Lead / Developer Support Rubens Rissi Onzi	Conducted researches and analysis for technology selection. Supported backend development with smart caching, Kubernetes setup and deployment, and overall system architecture design.
Frontend Developer / Support Farzana Afrin	Development and implementation of features in the frontend module, with additional support in security integration, including authentication, multi-factor authentication (MFA), one-times passwords (OTP), password hashing and assisted in security integration such as authentication, mfa, otp, and password hashing

Table 7.1: Team Roles and Responsibilities

7.2 Risk Assessment

Risk management was conducted throughout the development lifecycle of the web application. We identified risks based on technical complexity, hardware dependencies, and team coordination.

Risk	Likelihood	Impact	Mitigation Strategy
NLP Hallucination	Moderate	High	Implementing stricter prompting to avoid unpredictable LLM behaviour.
Hardware Latency	Medium	Medium	Implementing a multicluster system.
Integration Complexity	High	High	Establishing clear API connection between the Frontend, Middleware, and NLP teams early in the development phase.
Disaster recovery	Low	High	Backup servers in each independent system
Corrupted Persistent Volume	Medium	Low	Automated Recovery Scripts that automatically unmount, clean and re-bind the PVCs in case of corruption.
Copyright Infringement	High	Critical	Data Minimization to store only metadata and vector embeddings instead of full article text.
GDPR Non-Comlpiance	High	Medium	Personally Identifiable Information (PII) Redaction Pipeline.
IP Blacklisting	Medium	Critical	Using a pool of rotating proxies and implement stricter rate limiting in the Web Search module.
SQL Injection	Low	High	Rely on Spring Data JPA in the middleware.
SSRF (Server-Side Request Forgery)	Medium	High	Strictly whitelist allowed protocols and block requests on IP ranges.

Table 7.2: Project Risk Assessment Matrix

Part IV

Development

Chapter 8

System Implementation

This chapter outlines the development lifecycle of the web application and describes in details the technologies used, the system's data flow and architecture, and the key design and implementation decisions.

8.1 System Architecture Overview

As previously shown, an initial architecture diagram was presented; however, throughout the development of the project and with a better understanding of the system, the architecture has evolved into a more refined structure. The following diagram illustrates the updated architecture:

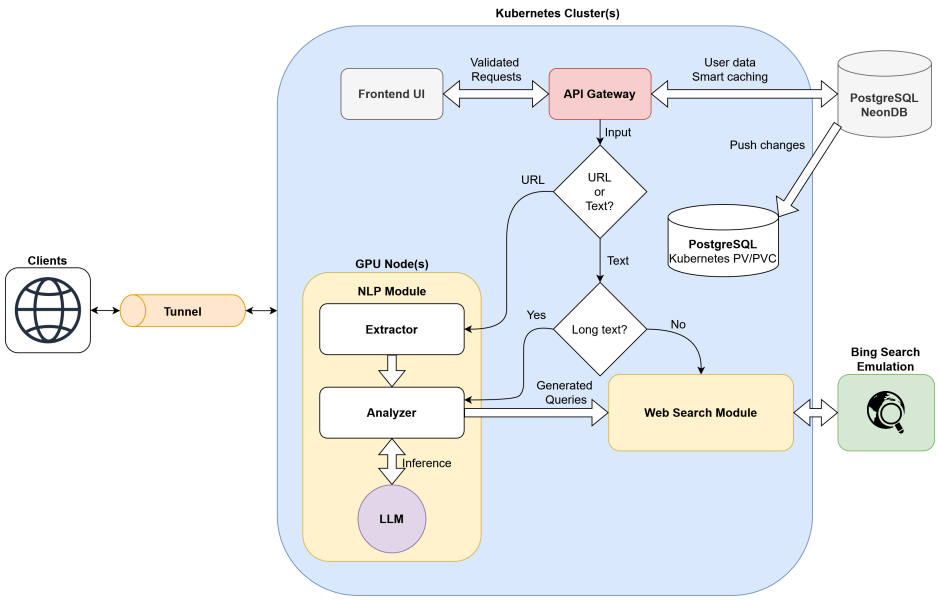

Figure 8.1: Architecture

8.2 Components Descriptions

Each component's technology stack was selected based on the system's requirements and its suitability with the overall architecture. Integration and compatibility with the other modules were also key factors in the process. The prior experience of the team member responsible for each component also influenced the decision and made for smoother and more efficient development process. To ensure a high-performing and reliable system, several factors were evaluated such as performance, scalability and maintainability.

8.2.1 Natural Language Processing (NLP)

Developed in Python, the Natural Language Processing module acts as the analytical core of the system, it is responsible for processing long texts and URLs to retrieve important context for source tracking. This component is the most demanding in terms of hardware capabilities due to the use of LLM, which is responsible for extracting and summarizing information, a crucial step in the generation of reliable search terms.
To setup a local environment to run the LLM at its full capabilities, without relying on cloud services, we retrieved the models from Hugging Face. The platform hosts a substantial amount of open-source language models, and it commonly uses pytorch for its delevoper-friendly debugging features, making it the preferred choice for rapid experimentation and efficient model development.
The processing pipeline is defined by the following components:

1. **Extractor:** Handles URLs, retrieves the site data from the link provided and identifies the top sentences and key entities.
2. **Analyzer:** Through the NLP libraries, such as newspaper3k and Spacy, it dissects the text and extracts the top sentences.
3. **LLM:** The results of the analyzer will be summarized and used to generate search terms.

This module was integrated into the architecture through FastAPI, which is a high performance python web framework, ensuring that it can communicate efficiently with the Java Sprint Boot Middleware layer.

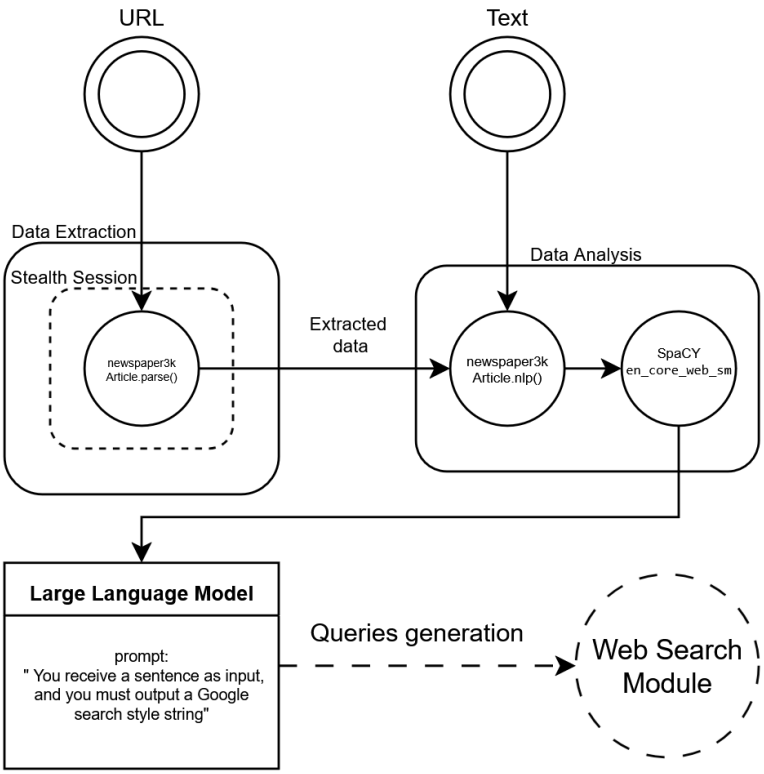

Figure 8.2: NLP Module Pipeline

8.2.1.1 Natural Language Processing in Python

Python was selected as the implementation language due to the amount of available libraries and tools specialized for text analysis. Specifically, the system integrated **Newspaper3k** for article parsin and keyword extraction, and **SpaCY** for linguistic processing. This process provides a robust solution for complex data extraction tasks.

Newspaper3k: Extraction & Analysis
The `Newspaper3k` library serves as the main component for data extraction as it provides a high-lever parsing tool via the `Article.parsing()` method. The analytical section of the process is executed through the `Article.nlp()` method, which relies heavily on the **Natural Language Toolit (NLTK)** library.

The method `Article.nlp()` implements a custom extraction algorithm, it filters content by assigning each sentence a score based on weight:

- **Title Overlap:** Weights sentences that contain words found in the article's title.
- **Positional Bias:** Prioritizes sentences appearing in the first 10% of the article.
- **Keyword Frequency (SBS/DBS):** Calculates a relevance score based on the accumu-

lation and density of keywords with high value.

- **Length Normalization:** Penalizes heavily sentences with a length that diverge heavily from the ideal one (around 20 words).

SBS and DBS Metrics The keyword relevance score is derived from two specific sub algorithms:

- **SBS (Summation Based Selection):** Calculates the sum of the values of keywords in a sentence, ensuring that short sentences with more keywords score higher than long sentences with fewer keywords. Let a sentence S consist of n words, and let each keyword k have a corresponding weight w_k. The SBS score of the sentence is defined as follows:

$$SBS(S) = \frac{1}{10 \cdot |S|} \sum_{w \in S} \mathbb{1}_K(w) \cdot \text{score}(w)$$

where :

  - $|S|$ is the lentgh of the sentence
  - $\mathbb{1}_K(w)$ equals 1 if w is a keyword in the set K, 0 otherwise.
  - $score(w)$ correspon to the value `keywords[word]`.

The corresponding python implementation:

```
    def sbs(words, keywords):
    score = 0.0
    if (len(words) == 0):
        return 0
    for word in words:
        if word in keywords:
            score += keywords[word]
    return (1.0 / math.fabs(len(words)) * score) / 10.0
```

- **DBS (Density Based Selection):** The DBS algorithm calculates and evaluates a sentence by measuring the distance between keywords within the sentence. For every pair of consecutive keywords, with the corresponding positions i and j and their weights, the score is calculated with the following formula:

$$\frac{w_i \cdot w_j}{(i-j)^2}.$$

The algorithm sums these scores over all consecutive pairs of keywords, producing:

$$score_{\text{raw}} = \sum_{\text{pairs}} \frac{w_i w_j}{(i-j)^2}.$$

The score is then normalized using intersections to avoid disproportions:

$$k = |K \cap S| + 1,$$

$$\mathrm{DBS}(S) = \frac{1}{k(k+1)} \cdot \mathrm{score}_{\mathrm{raw}}.$$

where:

- S = sentence being evaluated.
- K = dictionary of keywords with their weights
- i_n, i_{n-1} = positions of the current and previous keywords in the sentence
- w_n, w_{n-1} = weights of the current and previous keywords
- k = number of unique keywords from K that appear in the sentence

The corresponding python implementation:

```
    def dbs(words, keywords):
    if (len(words) == 0):
        return 0
    summ = 0
    first = []
    second = []

    for i, word in enumerate(words):
        if word in keywords:
            score = keywords[word]
            if first == []:
                first = [i, score]
            else:
                second = first
                first = [i, score]
                dif = first[0] - second[0]
                summ += (first[1] * second[1]) / (dif ** 2)
    # Number of intersections
    k = len(set(keywords.keys()).intersection(set(words))) + 1
    return (1 / (k * (k + 1.0)) * summ)
```

The flow of the Article.nlp() method can be interpreted as the following flow chart:

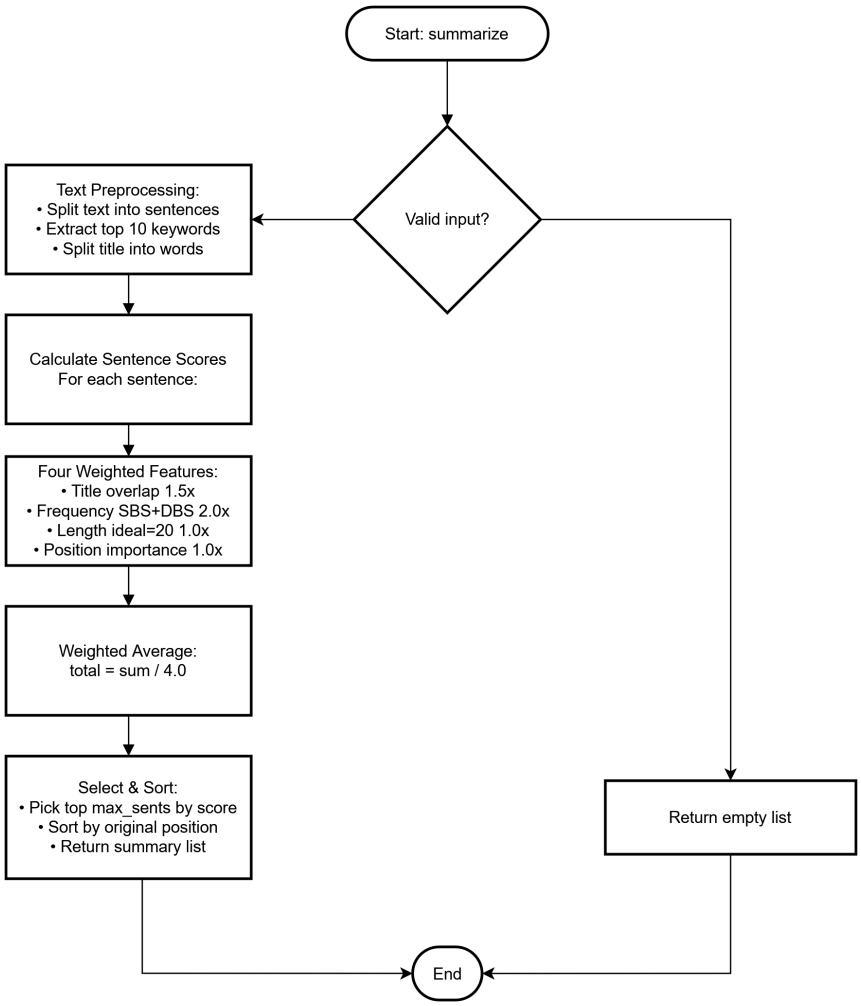

Figure 8.3: Flow of the Article.nlp() scoring algorithm.

SpaCy: Parsing & Segmentation
Although Newspaper3k provides tools for summarization, it lacks the grammatical awareness for proper input formatting. This is where we integrated SpaCY with the `en_core_web_sm` model.

- **Rule Based Tokenization:** `Spacy` utilizes a rule based tokenizer by processing text through steps:

  1. **Split:** The raw text is split on whitespace characters.
  2. **Exception Check:** Each substring is checked against a language specific exception dictionary (e.g. "don't" does not contain whitespace, but should be split into two).

3. **Prefix, Suffix or Infix:** the system checks for punctuations and peels off prefixes, suffixes and infixed.

If there's a match, the rule is applied and the tokenizer continues its loop, starting with the newly split substrings. This way, spaCy can split complex, nested tokens like combinations of abbreviations and multiple punctuation marks.

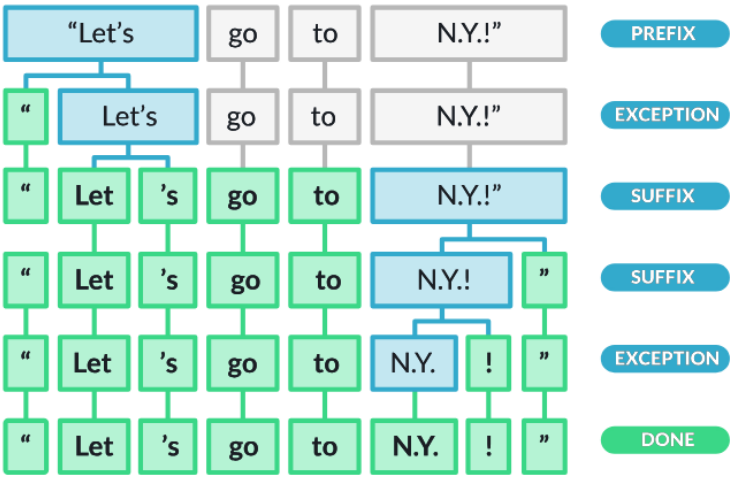

Figure 8.4: Tokenization in SpaCY [10]

- **Dependency Parsing:** Sentence segmentation is performed by the dependency parser, this model utilizes a variant of the non-monotonic arc-eager transition-system described by Honnibal and Johnson (2014), with the addition of a "break" transition to perform the sentence segmentation. This parser jointly learns sentence segmentation and labelled dependency parsing, and can optionally learn to merge tokens that had been over-segmented by the tokenizer. The parser is trained using an imitation learning objective to predict state transitions.

- **Name Entity Recognition (NER):** SpaCY utilizes a transition-based component that identifies non-overlapping labelled spans of tokens. The transition-based algorithm used encodes certain assumptions that are effective for "traditional" named entity recognition tasks, but may not be a good fit for every span identification problem.

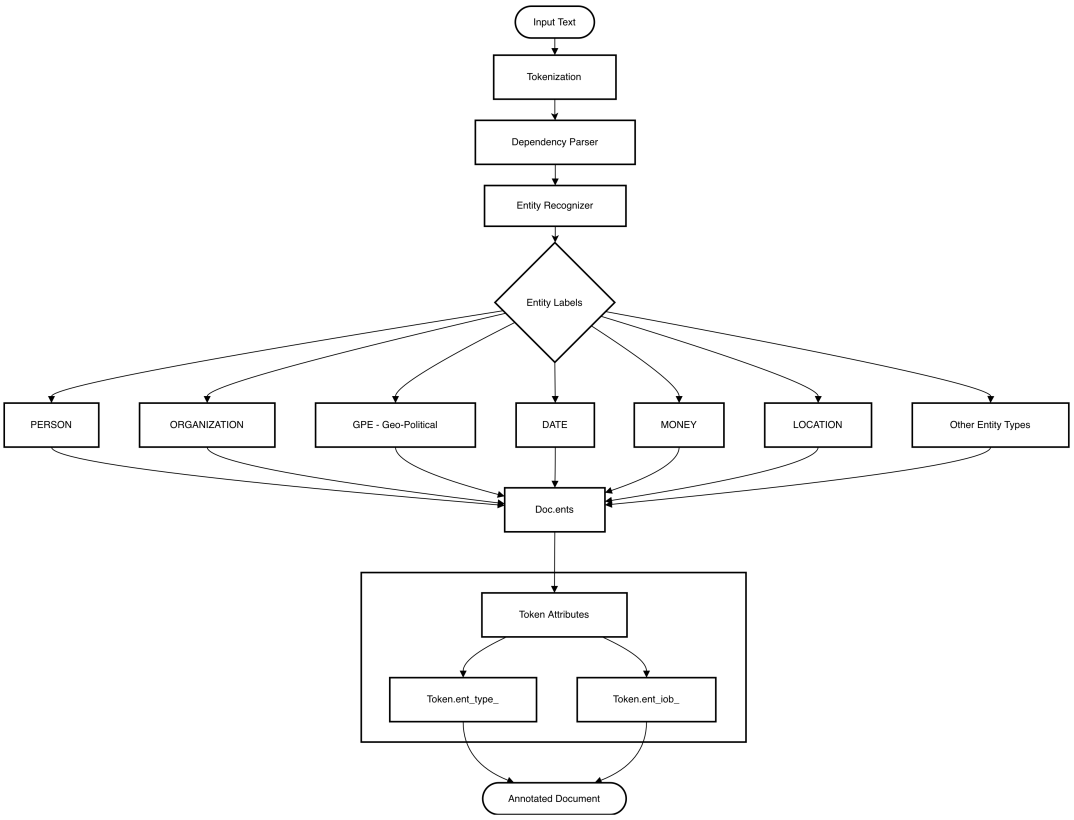

Figure 8.5: Spacy NER Diagram

While SpaCY model detects entities inside a text we implemented a custom selector to retain only relevant types, ensuring that search terms remain focused on the topic. Python implementation of the selector:

```
def find_entities(self, text: str):
entities = []
labels={}
doc = self.nlp(text)
for ent in doc.ents:
    if ent.label_ in {"PERSON", "ORG", "GPE", "EVENT", "
        WORK_OF_ART"}:
        if ent.text not in entities:
          entities.append(ent.text)
          labels[ent.text]=ent.label_
return [{"name":e, "label":labels[e]} for e in entities]
```

SpaCY model
The model used in our NLP module is `en_core_web_sm`, a small english pipeline trained on written web text, optimized for CPU usage. The model acts as the engine for the SpaCY methods, as it understands the grammatical structure of the English language.
The model is based on the following pipeline: Components of the model:

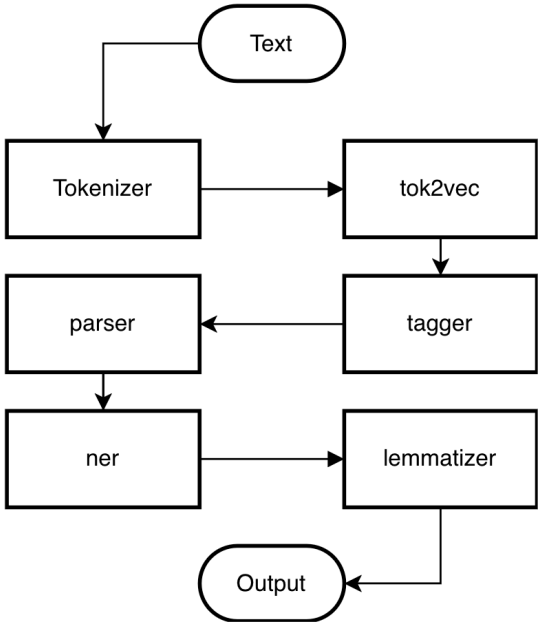

Figure 8.6: Simplified architecture of the en_core_web_sm pipeline.

- **Tokenizer**: Segments raw text into individual tokens.
- **tok2vec**: Converts tokens into numerical vector representations using multi-hash embeddings.
- **tagger**: Assigns part-of-speech (POS) tags to each token.
- **parser**: Analyzes syntactic dependencies between tokens to construct a dependency tree.
- **ner**: Identifies and classifies named entities.
- **lemmatizer**: Reduces tokens to their base or dictionary form.

Example of the model usage:

```
    import spacy
    from spacy.lang.en.examples import sentences

    nlp = spacy.load("en_core_web_sm")
    doc = nlp(sentences[0])
    print(doc.text)
    for token in doc:
        print(token.text, token.pos_, token.dep_)
```

Query Variations and Generation
This component of the project makes the most use of the LLM, as it generates similar search queries for a deeper web scraping to improve the quality of the retrieved information. The motivation for this approach is the risk of missing useful and critical information during the retrieval of the top-k sentences. The additional and conceptually different queries provide a broader coverage and lead to a more precise result.
The LLM received the following prompt:

```
1    "Generate {num_variations} diverse search engine queries about the
         topic '{query}'.
2    The queries should be conceptually different. Do not use a
        numbered list or bullet points. Just list each query on a new
        line."
```

8.2.1.2 Hugging Face

As mentioned earlier, to fully leverage the LLM's capabilities and potential, we chose not to rely on cloud services and instead ran the model locally. The available machines possess decent performing gpus with 8GB of VRAM (respectively RTX2080 and RTX3070 Mobile), for this reason a smaller model with 4B parameters was chosen. The largest and most user-friendly platform that hosts open source LLMs is hugging face.

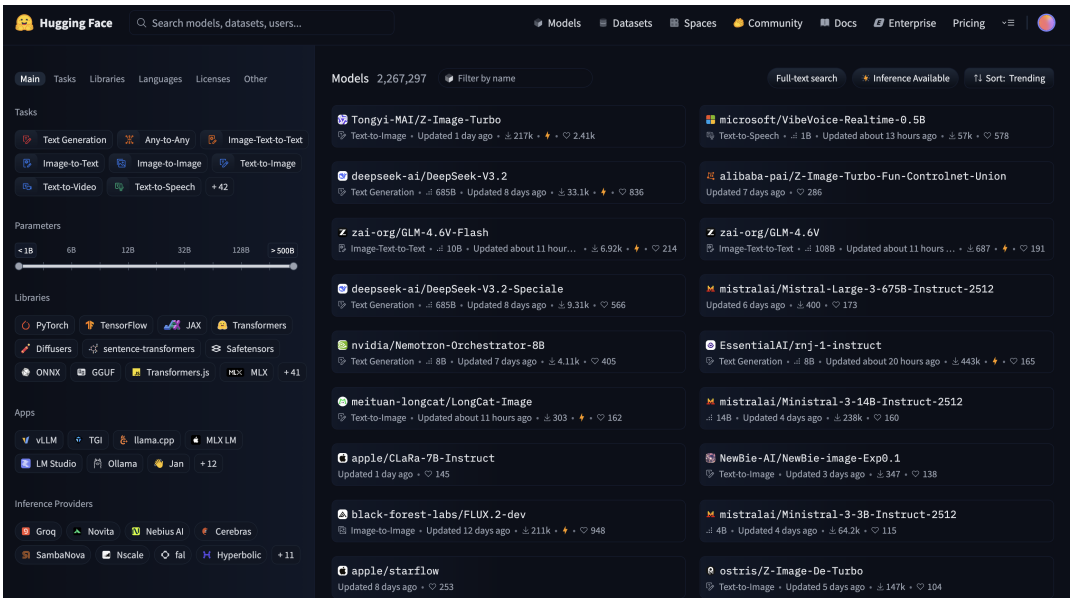

Figure 8.7: Dashboard of available LLMs in the Hugging Face platform.

8.2.1.3 Transformers

Transformers is a library developed by Hugging Face and acts as the model-definition framework for state-of-the-art machine learning in text, computer vision, audio, video, and multimodal model, for both inference and training. [11]
These models in NLP represent a type of deep learning architecture designed to handle operations regarding data elaboration and question answering. Unlike older models, such as RNNs and LSTMs, transformers are able to process entire sentences at once thanks to parallel processing.
The key components of this architecture are:

- Encoder: Reads the input and converts into a numerical representation
- Decoder: Takes the Encoder representation and generates output

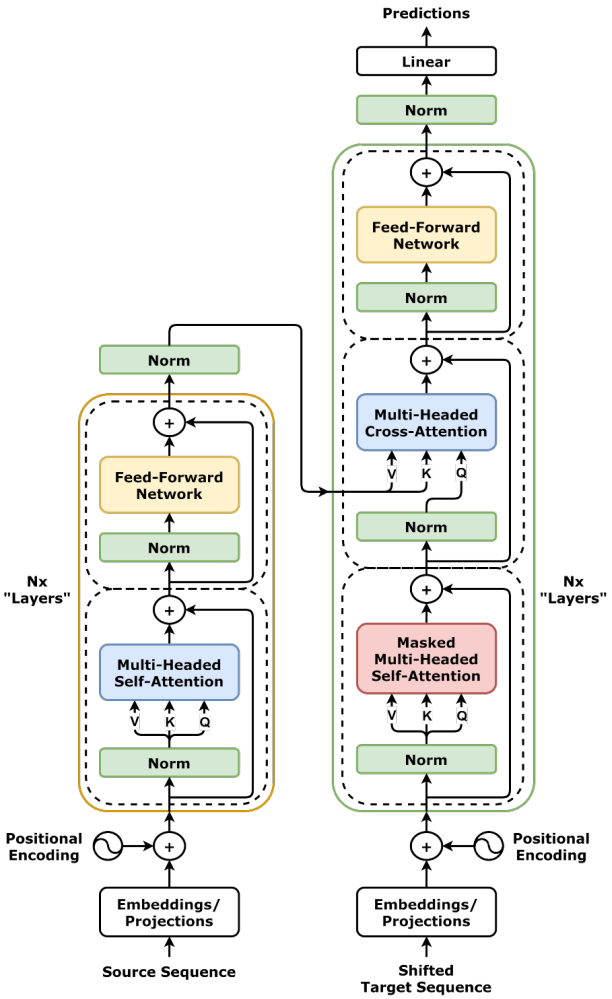

Figure 8.8: Architecture of a standard Transformer [12]

8.2.1.4 Pytorch

The NLP module implements the Hugging Face Transformers which act as a high level interface for models; to maximize performance our system leverages PyTorch as the deep learning framework
PyTorch is an open source deep learning framework built to be flexible and modular for research, with the stability and support needed for production deployment. [13] It is responsible for for executing tensor operations on the hardware, which are mathematical manipulations performed on tensors that act as engine for machine learning models.
Through these optimized engine, PyTorch improves performance significantly:

- **Dynamic Computational Graphs:** PyTorch constructs the computational graph dynamically at runtime, allowing flexible handling of variable-length input sequences.

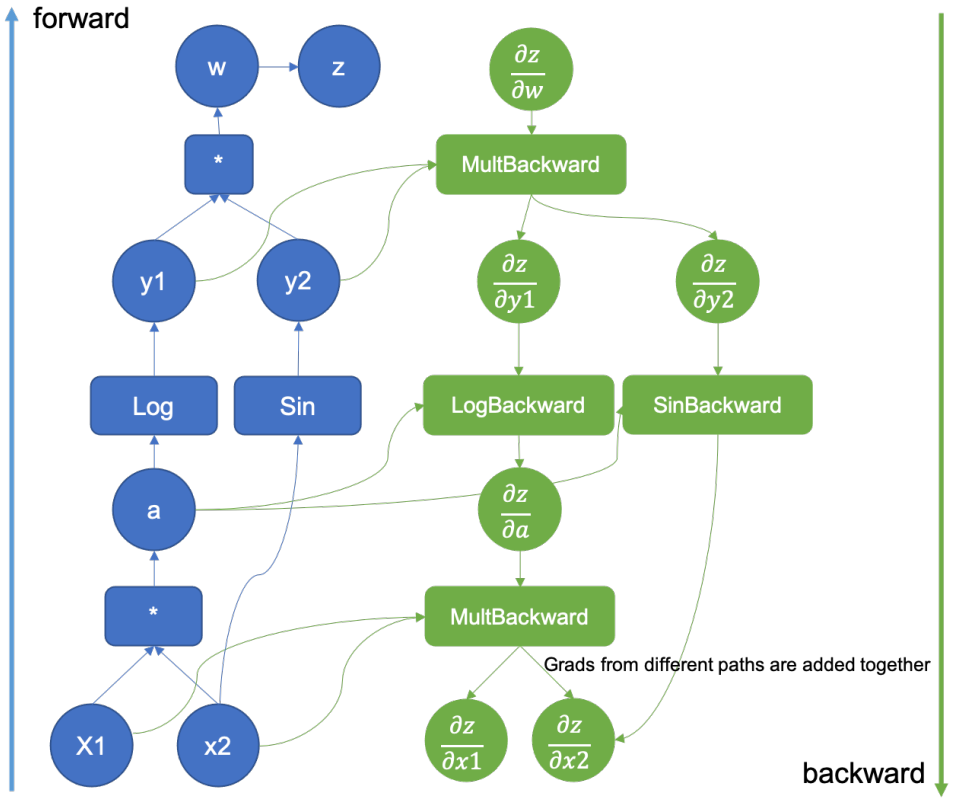

Figure 8.9: Example of an augmented computational graph [14]

Python implementation:

```
        outputs = self.llm.pipeline(
            prompt, # Dynamic length
            max_new_tokens=50,
            eos_token_id=terminators,
            do_sample=False,
        )
```

- **Tensor Computation & CUDA Mapping:** The system uses torch.Tensor data structure to store model weights, these tensors are mapped directly in the GPU's VRAM via the NVIDIA CUDA interface, improving overall performance and avoiding CPU bottleneck. CUDA is a parallel computing platform and programming model developed by NVIDIA that enables dramatic increases in computing performance by harnessing the power of the GPU. [15] The GPU devotes more transistors for data processing, which is beneficial for highly parallel computations. The combination between CPU and GPU maximizes overall performance, applications with a higher degree of parallelism can exploit the GPU for better performance.

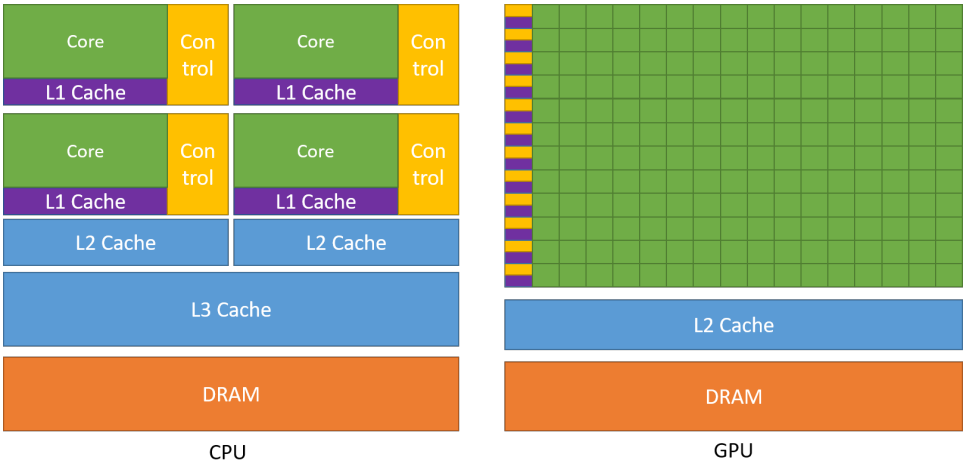

Figure 8.10: The GPU Devotes More Transistors to Data Processing [15]

Python implementation:

```
    def __init__(self, model="llm_model_name", device="cuda", task
        ="text-generation"):
        self.pipeline = pipeline(
            #...
            device_map="auto", # Hardware mapping
            dtype=torch.bfloat16 # Precision
            #...
        )
```

- **Quantization Pipeline:**

  Quantization focuses on representing data with fewer bits while also trying to preserve the precision of the original data. This often means converting a data type to represent the same information with fewer bits. [16] This process maps down a large amount of values into a smaller set of values to improve performance and avoid GPU bottleneck.
  Python implementation:

```
    self.pipeline = pipeline(
        #...
        model_kwargs={"load_in_4bit": True}, # Quantization
        #...
    )
```

The orchestration is handled by the `Accelerate` library, which automatically determines optimal device mapping and available resources to prevent Out of Memory (OOM) errors.

8.2.1.5 LLM

The development machines were equipped with an RTX 2080 and an RTX 3070 Mobile, both limited to 8GB of VRAM. This constraint impacted the choice of the large language model, as larger ones would exceed the memory capacity of the cards and degrade the performance. Given that the role of the LLM in the system was summarisation and query generation, deploying a very large model was unnecessary, while extremely small models were disregarded due to their inaccuracy and inconsistency.

Gemma

The project initially revolved around the Gemma 3 4B model, a lightweight model based on a decoder architecture released in march 2025 developed by Google.
This design makes it a powerful multimodal model that integrates native vision and multilingual capabilities in a 4B parameter architecture, offering the following advantages:

- **Multimodal Native:** Processes both images and text without separate adapters.
- **Edge Optimization:** Compact architecture designed for deployment on consumer devices.
- **Resource Efficiency:** Delivers competitive high-performance inference with minimal memory overhead.

Gemma was promising because it was a significant shift in the "small language model" (SLM) landscape due to it being a multimodal model. At the release, the Gemma models outperformed significantly larger models while still being able to run on a single GPU. This made Gemma an optimal solution for our project scope, enabling low latency, on device inference and fast offline processing.

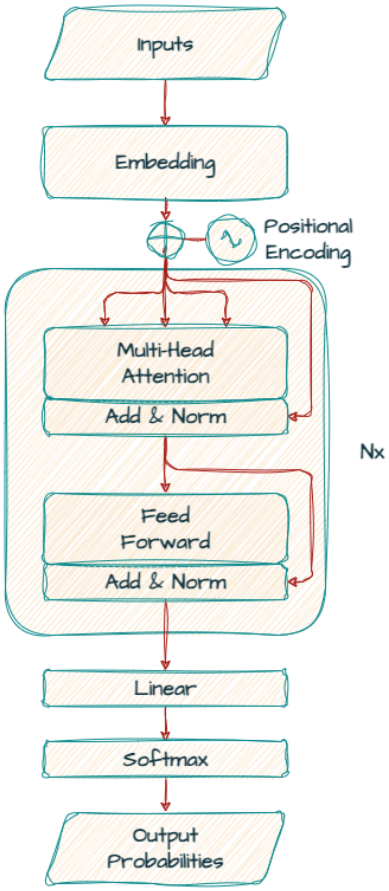

Figure 8.11: Transformer decoder that Gemma models are based on. [17]

Qwen
Although Gemma was promising and demonstrated many advantages, the team decided to ultimately migrate towards the Qwen 3 4B Instruct model, another small language model with 4B parameters released in august 2025 developed by Alibaba.
Qwen is a decoder only model that differs from Gemma in both training strategies and architecture, prioritizing instruction following. The migration choice was driven by the NLP pipeline, where information precision was critical.

- **Word Retention:** Qwen ensures that named entities are preserved during summarization.
- **Reasoning Enhanced:** the model can correct logic flaws during query generation reducing hallucinations.

The issue with LLMs is that, despite thorough benchmarks and research, their behaviour are not always consistent once integrated into a system. Additional evaluation and testing results will be presented in a later chapter.

Complete NLP Pipeline Implementation
These distinct components are orchestrated within the NLP_Pipeline class and the process is executed sequentially:

1. Input processing
2. Summarization and Splitting
3. Entity Enrichment
4. Query Generation

Test Case
To understand what the pipeline returns, a snippet has been used for a test case:
Parameters:

```
top_k = 5
queries = 4
```

Input Text:

> "SpaceX achieved a historic milestone on Sunday by successfully catching the booster of its massive Starship rocket. The 400-foot rocket launched from Texas at sunrise. Minutes later, the booster separated and navigated back to the launch pad, where it was grabbed by mechanical arms known as 'chopsticks'. Elon Musk, CEO of SpaceX, declared the test a massive success for the goal of interplanetary travel."

Pipeline Output:

```
[Source Sentence]: SpaceX achieved a historic milestone on Sunday by
    successfully catching the booster of its massive Starship rocket.
[Entities Detected]: SpaceX (PERSON), Starship (PERSON)
   -> Search Query: SpaceX Starship booster catch milestone
   -> Search Query: What does the SpaceX Starship booster catch milestone mean
       for future space missions?
   -> Search Query: How did SpaceX achieve the first successful booster catch
       during a Starship test flight?
   -> Search Query: What challenges did SpaceX face in achieving the Starship
       booster catch milestone?

[Source Sentence]: The 400-foot rocket launched from Texas at sunrise.
[Entities Detected]: Texas (GPE)
   -> Search Query: 400-foot rocket launch Texas sunrise
   -> Search Query: What does a 400-foot rocket launch look like during a
       sunrise over Texas?
   -> Search Query: Is there a real 400-foot rocket launch scheduled in Texas
       this month with a sunrise view?
   -> Search Query: How would a 400-foot rocket launch in Texas affect the
       local sunrise photography?
....
```

For the implementation of the function please refer to Appendix A, Listing **??** for the pipeline execution and Listing **??**

8.2.2 Web Scraping and Data Collection

The Web Search module was also implemented in Python, primarily due to the extensive ecosystem and open source libraries developed to bypass anti-bot and anti-scraping protections. As discussed in subsection 4.1.2, the use of a free-tier web-search API proved inadequate for our requirements. Given the absence of chronological sorting capabilities in the APIs, we ultimately decided to implement an emulated web-search pipeline by interfacing with the Bing search engine.
This module is defined by the following pipeline process:

- **Market Selection:** search engines tailor results based on the IP address, causing variations. The system explicitly sets the market to the language of the queries.
- **Web Scraping:** Through stealth request and HTML parsing the module retrieves relevant information from Bing's Search Engine Results Page (SERP) elements, bypassing bot detection mechanisms.
- **Post-Processing:** Extracted information are normalized and translated to English.
- **Result Generation:** The system produces two structured result lists: one containing dated sources, and another containing undated materials such as websites, PDFs, or other types of documents.

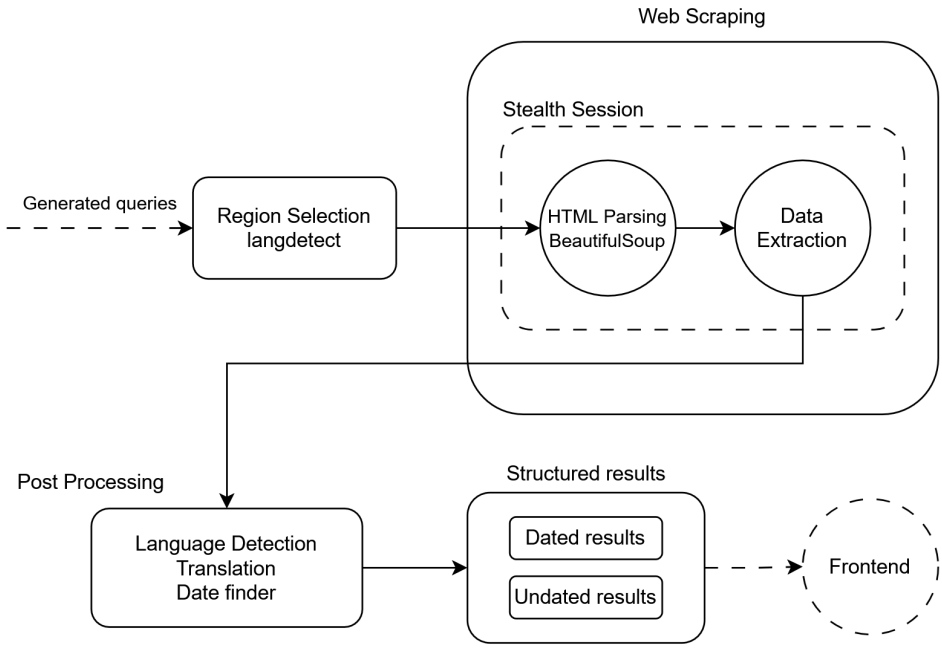

Figure 8.12: Web Search module diagram

8.2.2.1 Web Search Emulation in Python

The module is implemented using a combination of Python libraries, including `stealth_requests`, `BeautifulSoup`, `datefinder`, `deep_translator` and `langdetect`. This toolchain provides the necessary assets for a robust scraping pipeline.

Bing Search
Bing was selected as the search engine for emulation due to its permissive anti-scraping behaviour and structured SERP layout. Google Search, on the other hand, employs an aggressive anti-bot mechanism, making stable scraping approaches almost impossible without headless browser automation. Bing provides a reliable and consistent HTML structure, enabling easier data extraction for the pipeline.
Python implementation of the bing search emulation:

```python
def build_bing_search_url(query: str, first: int = 0, market: Optional[str] = None) -> str:
    # Function from urllib.parse library that ensures the query is URL-safe
    safe_q = quote_plus(query)
    url = f"https://www.bing.com/search?q={safe_q}&first={first}"
    if market:
        url += f"&mkt={market}"
    return url
```

To prevent the IP blocking problem, detailed in Section 5.4, a threshold of requests was set.

Stealth Session
The system used the `stealth_requests` library to enable HTTP requests that mimic real user behaviour by randomizing headers and user-agents; this method reduces the chances of bot detection and ensures uninterrupted access.
For the complete implementation details of the `StealthSession` class and functions, please refer to Appendix A, Listing A.3.

Data Parsing and Extraction
Once the retrieval of the page is successful through the stealth request, the `BeautifulSoup` library parses and processes the HTML, creating a searchable Object.

```python
def parse_bing_results(html, search_type):
    soup = BeautifulSoup(html, "lxml") # Object creation
```

After creating a structured and searchable DOM tree, the parsed element will be forwarded to the pipeline for data extraction:

1. **Title, URL and Snippet retrieval:**

```python
# Parsing title, url, snippet
if a_tag:
    title = a_tag.get_text(strip=True)
    url = self.clean_bing_url(a_tag.get("href", ""))
if snippet_tag:
    snippet = snippet_tag.get_text(strip=True)
```

2. **Date retrieval:** implementation of custom data extraction logic and parsed afterwards.

```
return dateparser.parse(text_lower)
```

   For the complete custom implementation of the date retrieval function please refer to Appendix A, Listing A.4. This snippet illustrates the implementation of a custom data extraction logic.

3. **Language Detection and Translation:** Using `langdetect` to detect the original language of the title and `deep_translator` to translate it.

```
lang = detect(title)
translator = GoogleTranslator(source='auto', target='en')
to_translate = [title or "", snippet or ""]
translations = translator.translate_batch(to_translate)
translated_title = translations[0]
translated_snippet = translations[1]
```

This processing pipeline returns well structured lists that additionally provide the original language and the translated title.

Structured Results
After processing, the module produces two distinct structured result sets:

- **Dated Results:** Sources for which a reference date was detected.
- **Undated Results:** Sources lacking identifiable dates, including static websites, PDF documents, or generic informational pages.

This separations helps the user assess sources that also come from different types of webpages, supporting a more reliable origin tracking.

Oldest Result
The Oldest Result is extracted from the list of Dated Results and set as the **Origin** of the piece of information.

Structure of the results:

```
{
  "result": [
    {
      "sentence": <original_sentence>
      "search_term": <generated_search_term>,
      "news_results": [
        {
          "url":
          "date":
          "title":
          "snippet":
          "original_title":
          "original_snippet":
          "original_language":
        },
        ...
      ],
      "website_results": [
        {
          "url":
          "date": null
          "title":
          "snippet":
          "original_title":
          "original_snippet":
          "original_language":
        }
        ...
      ]
    }
    ...
  ],
  "warning": null,
  "oldest_result": {
    "url":
    "date":
    "title":
    "snippet":
    "original_title":
    "original_snippet":
    "original_language":
  }
}
```

In summary, emulating web search in Python provided a total control over query execution, scraping, metadata extraction and translation capabilities that were not available through the existing free tier APIs. For more detailed information regarding the code implementaion of the results please refer to A, section A.2.3

8.2.3 Middleware Layer

The middleware API was implemented in Java using Spring Boot, it acts as the orchestrator between all the modules. The decision of Spring Boot was driven by the robust ecosystem for microservices, its support for SQL integration, its RESTful API development and the expertise of the members assigned to the production of the module.

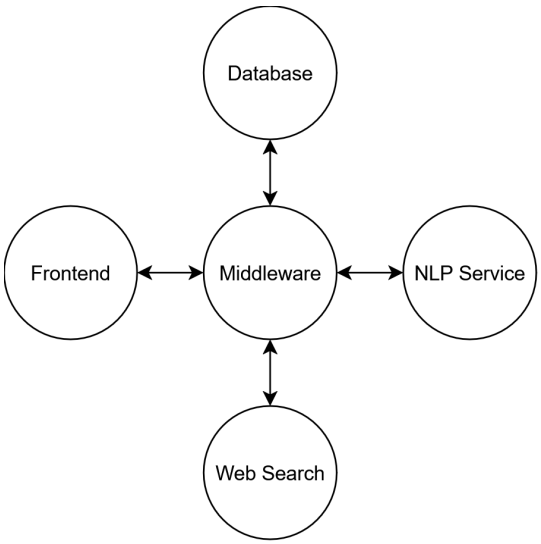

Figure 8.13: Middleware Diagram

This module is defined by the following components:

- **Request Normalization:** Requests from the frontend module are validated and normalized.
- **Cache Management:** Implementation of a caching system that saves data received from the Web Search module into a PostgreSQL database, improving performance for repeated queries.
- **Downstream Service Orchestration:** Routes requests to the appropriate downstream services and handle timeouts.
- **User Management:** Authentication, User history tracking, Multi-Factor Authentication (MFA) via email One-Time Password (OTP).

8.2.3.1 Spring Boot REST API Implementation

This module is built using Spring Boot, leveraging Spring Data JPA for database operations, Spring WebFlux for the reactive client, and Spring Security for password encryption.

Spring Data JPA
Spring Data JPA (Java Persistence API) is a library/framework that adds an extra layer of abstraction on top of the JPA provider line Hibernate. The JPA provides a specification for persisting, reading, and managing data from a java object to a relational table in the database. The API specifies the set of rules and guidelines for developing interfaces that follow standards. Essentially it's part of the Spring framework and its goal is to significantly reduce the amount of boilerplate code required to implement a data access layer for various persistence stores. [18]

Spring WebFlux
Spring WebFlux is a reactive, non-blocking web framework that uses Project Reactor's reactive streams API to enable highly concurrent and asynchronous processing of web requests in a non-blocking and event-driven way. It is fully asynchronous and non-blocking using reactive streams and callbacks.[19]

Spring Security
Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications, it focuses on providing both authentication and authorization to Java applications. [20]

Cache-First Architecture
To avoid redundant processing of the same input, the middleware implements a cache-first strategy. The system queries the database using a composite key and searches for a semantically identical request, if not found it will forward the request to the appropriate module. This mechanism relies on a regex-based key generation, implemented in the `CacheService` class.

Reactive Downstream Communication
As mentioned earlier, the system utilizes Spring WebFlux's WebClient, this framework manages downstream calls asynchronously. This approach allows the middleware to handle high concurrency and enforces strict timeouts to ensure that slow responses from the other modules do not cause failures for the entire API layer.

Database Schema and ORM Mapping
The Spring Boot application maps Java objects into a relational PostgreSQL schema through Spring Data JPA. This step is responsible for handling the results returned by the Web Scraping module, by using a JSONB column type the system stores the analysis payload as a queryable JSON document.

Authentication and Security
Security and access control are enforced by the Spring Security framework, which manages stateless authentication architecture. Private information are protected using the BCrypt hashing algorithm before the storing process. To enhance security, a Multi-Factor Authentication (MFA) mechanism was added with OTPs generation.

Search History Tracking
User activity is tracked via the SearchHistory entity, which links cached results with users.

```
INSERT INTO search_history (user_id, cache_id, view_count, created_at)
SELECT :userId, lc.id, 1, NOW()
FROM link_cache lc
WHERE lc.cache_key = :cacheKey
ON CONFLICT (user_id, cache_id)
DO
UPDATE SET view_count = search_history.view_count + 1,
           created_at = NOW()
```

CORS and Cross-Origin Configuration
Given that the structure of the system is based on SOA, Cross-Origin Resource Sharing (CORS) is strictly configured. The `CorsConfig` class defines a policy that whitelists specific trusted origins used for development and testing, these include Kubernetes NodePort services, localhost, and public tunneling URL.

In summary, the Spring Boot middleware provides a robust and scalable orchestration layer that intelligently manages caching, coordinates services, and implements security features.

8.2.4 Databases

A relational database was selected for this project due to the nature of the data and the limited number of entities involved. Additionally, structured databases perform queries much faster than NoSQL, this approach simplifies data management while still allowing flexible JSON storage when required.

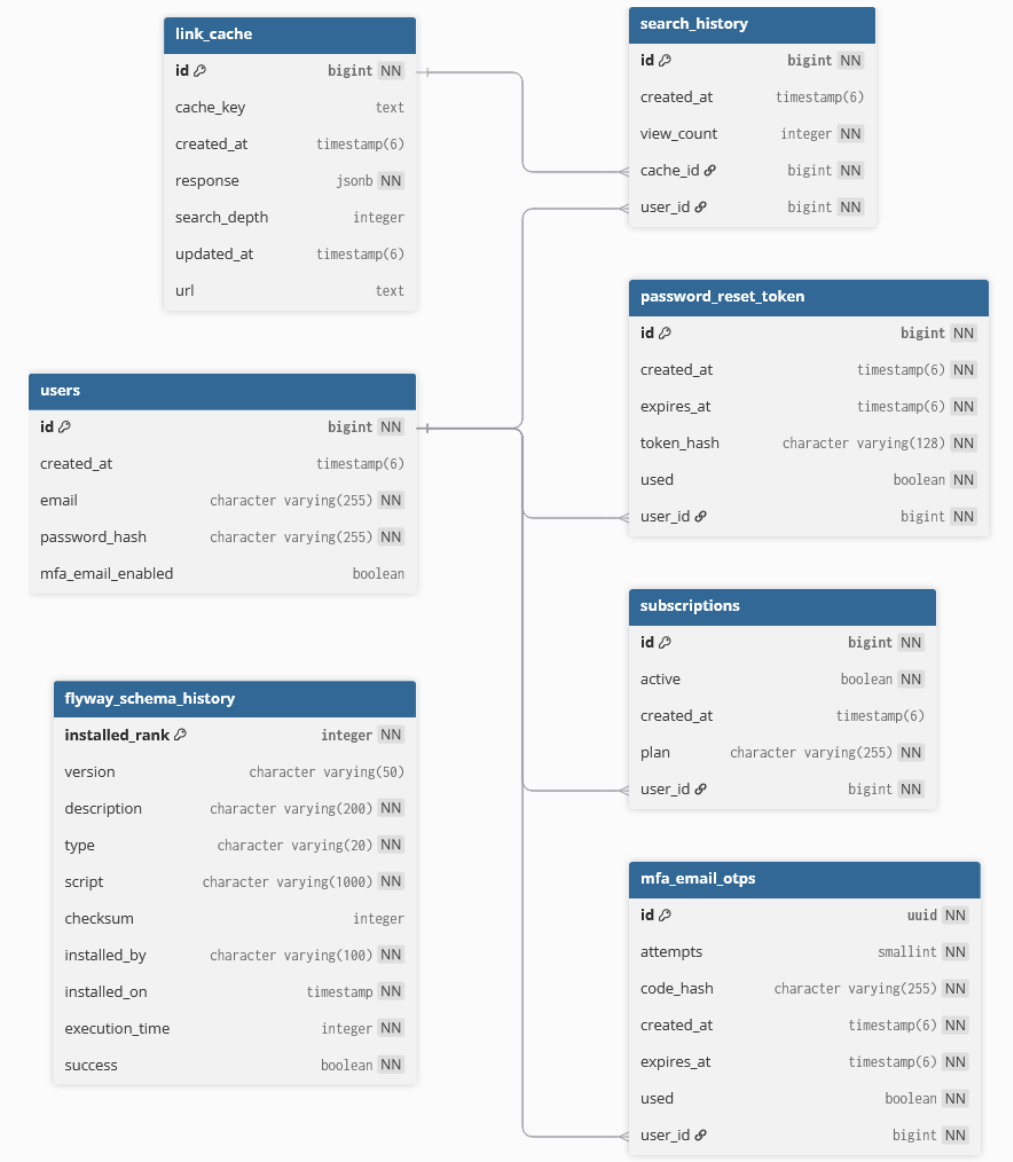

Figure 8.14: Entity Relationship Diagram

The database system is managed via Flyway for version-controlled schema migrations, while the architecture is centered around a hybrid data model that combines relational integrity with document storing flexibility:

- **Core Entities:** The main entities are Users and the schema defines the relationship centered around them. It creates foreign keys to link the different entities to ensure data integrity.
- **Hybrid Storage:** LinkCache table, as mentioned earlier, implements a JSONB column for storing unstructured results. This design allows to indext metadata for high performance queries.
- **Security Tables:** Dedicated tables for User Authentication.

8.2.4.1 Neon Tech

Neon Tech was selected as the database provider for this project due to its ease of use and available storage. Additionally, its serverless architecture separates compute and storage, allowing automatic scaling and isolated environment for development.

8.2.4.2 Persistent Volume and Local Redundancy

PostgreSQL was selected as the primary source of truth for the system. However, relying exclusively on a single external database instance introduces a potential single point of failure. To avoid this risk a local persistent volume was deployed within each Kubernetes cluster. This approach ensures that the database is preserved even if the cloud were to fail, improving fault tolerance and promoting high availability.

Logical Replication
To implement this replication logic, the database would need to support Logical Replication, a publisher subscriber relationship between the cloud service and the local instances. This setup allows the cloud database to stream changes (e.g. inserts, updates, deletes) to the local subscribers. This implementation ensures that each database inside different Kubernetes clusters are synchronized with the cloud source.
Creation of publication in the Publisher

```
1 CREATE PUBLICATION my_neon_publication FOR ALL TABLES;
```

Creation of subscription in the Subscriber

```
1 CREATE SUBSCRIPTION my_backup_sub
2 CONNECTION 'host=<host_name> port=<port> dbname=<db_name> user=<
    db_owner> password=<password> sslmode=require'
3 PUBLICATION my_neon_publication;
```

8.2.5 Email Service

This component was later integrated into the system as an additional security layer for the user. Developed and implemented in Python, this service is desigend to emulate a lightweight SMTP mail server for development and testing purposes. Its main goal is to generate emails such as password reset notification (through OTP) and multi-factor authentication (MFA) activation messages.

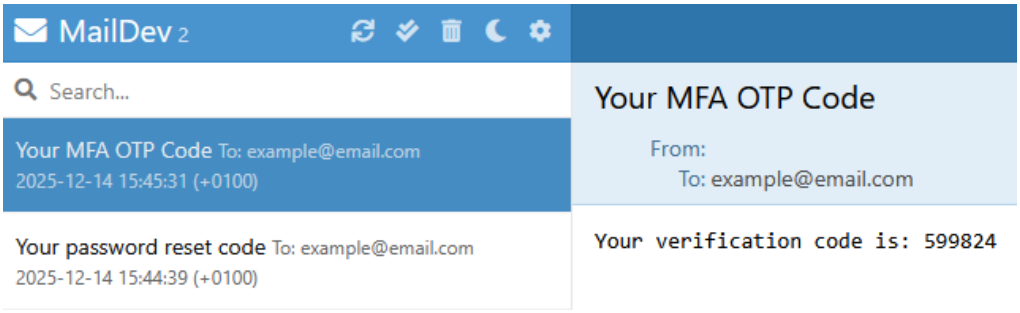

Figure 8.15: Email Service component

8.2.6 Frontend

The project's user interface is currently the only point of interaction for the system, developed using React with Typescript. The reason behind this choice was the framework's support for Server Side Rendering (SSR), its clear division between server-side actions and client-side interactions, and the team members expertise with the language. To ensure a responsive design across all devices the styling was handled with Tailwind CSS.

8.2.6.1 Design and User Experience

The application presents a simple interface with intuitive UI patterns, making the experience more enjoyable and easier to read for the user.

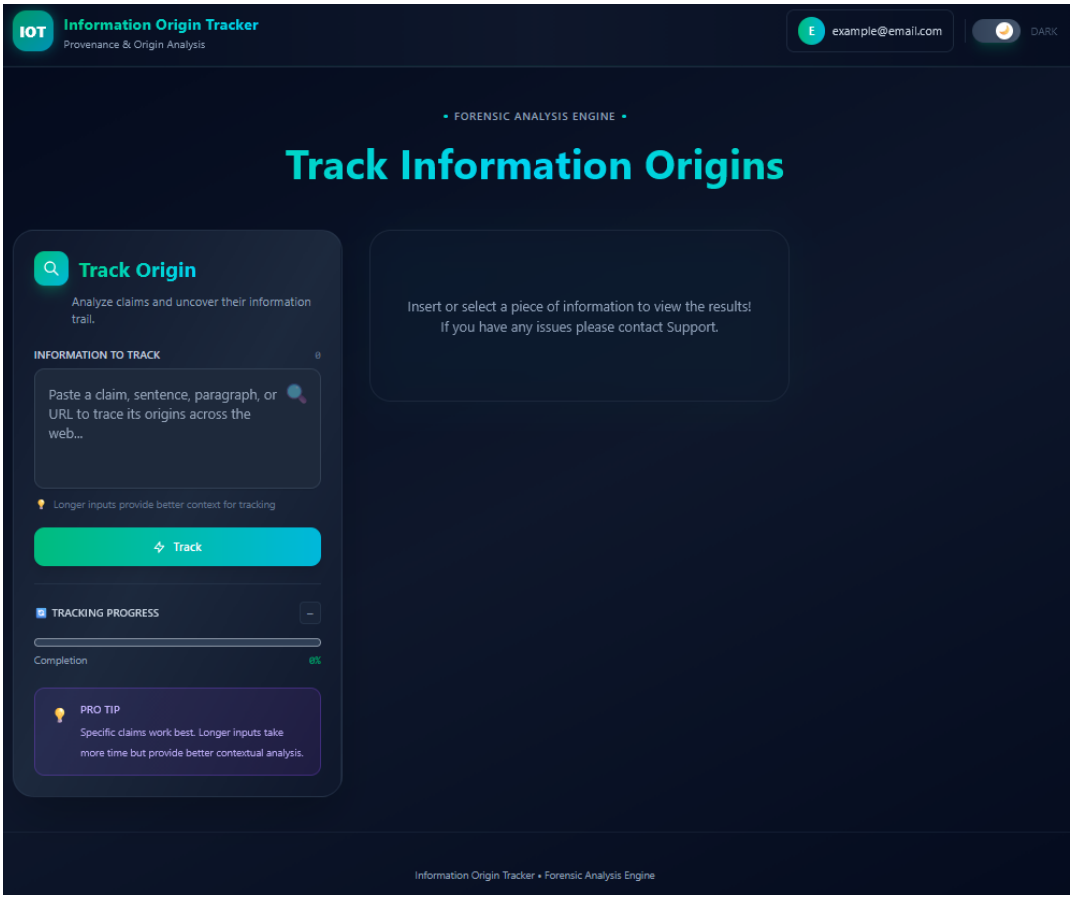

Figure 8.16: Initial page of the Web Application

8.2.6.2 Server Actions and API Integration

The communication happens through Next.js which encapsulates the following operations

1. **Content Analysis:** The `analyzeText` and `analyzeURL` functions send the user input to the middleware and returns an object containing the analyzed section. This action is called once the user clicks the track button in the search panel.
2. **User History:** The `getUserHistory` action retrieves the user's past search activities to populate the history panel, additionally it also memorizes the amount of time a search has been viewed.

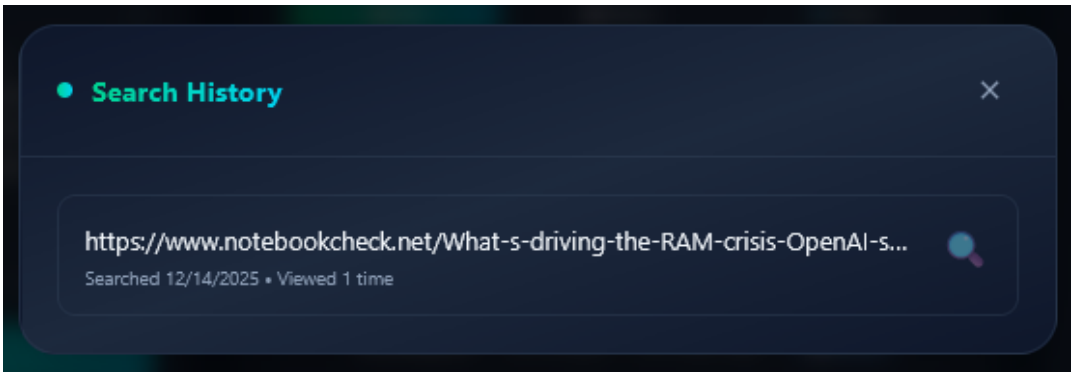

Figure 8.17: IOT History Panel

The full implementations of the features can be viewed in Appendix A, Listing A.5.

8.2.6.3 Client-Side State Management

The main application logic is centralized in the `page.tsx` component, which manages the entire interaction flow. This setup allows the interface to render loading indicators while also waiting for the other modules to process and complete demanding tasks.

8.2.6.4 Visualization Components

The whole interface is displayed through the different components and features highlighted in the image below.

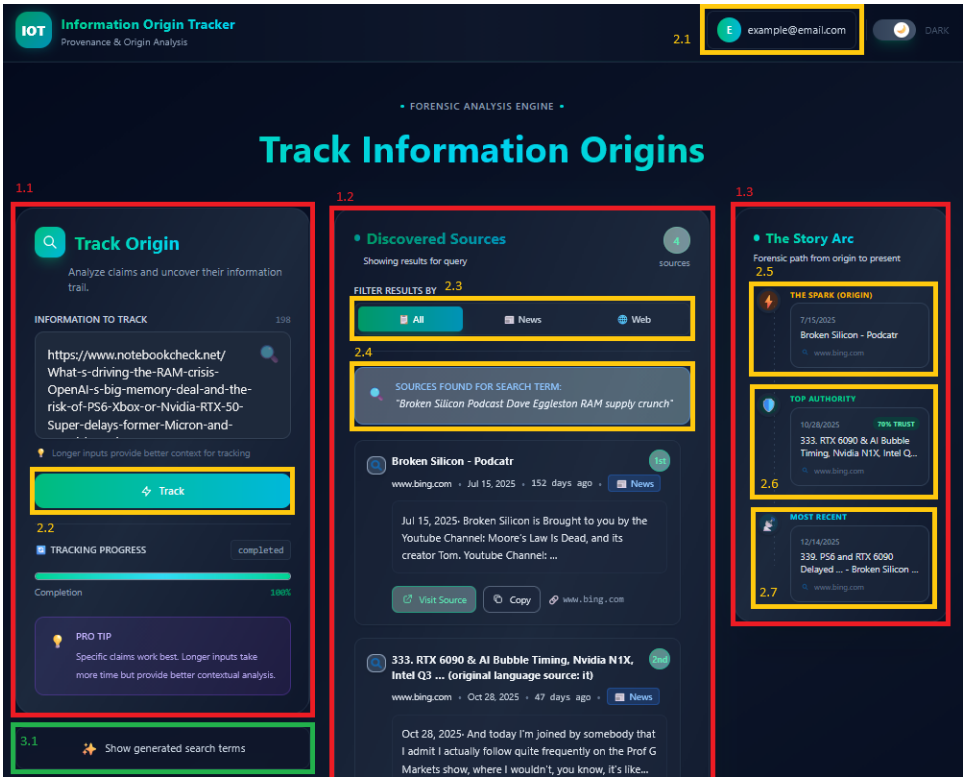

Figure 8.18: IOT Webapp Components and features

Descriptions:

- **1.1:** Search panel that contains the `analyzeTest` and `analyzeURL` functions, it additionally shows the tracking progress and a tip for new users.
- **1.2:** The Result List panel displays the retrieved sourced of a given search term.
- **1.3:** The Timeline panel only highlights 3 elements from the retrieved dated list.
- **2.1:** Button to access to user settings 8.20 and history 8.17.
- **2.2:** Button that initiates the tracking process by sending requests to the backend modules.
- **2.3:** Select the type of results to view.
- **2.4:** The search term that the sources are based on.
- **2.5:** The Origin of the search term.

- **2.6:** Source with top authority.
- **2.7:** Most recent source of the search term.
- **3.1:** New feature which displays the Analyzed Content panel without calling the backend service.

Analyzed Content panel
The following image shows the search terms generated from the top-k sentences extracted from the webpage. Each sentence includes highlighted entities, and users can reveal the type by hovering over them with the cursor.

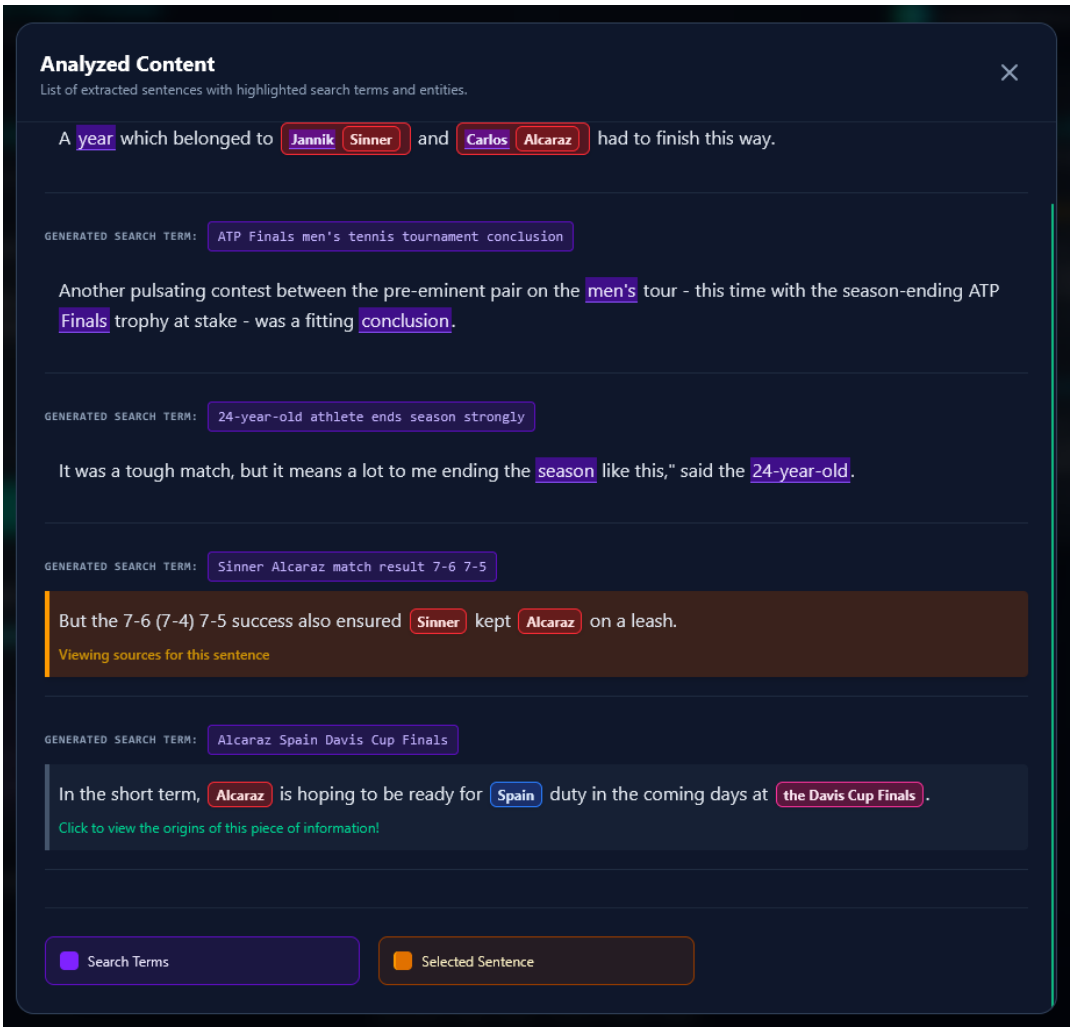

Figure 8.19: IOT Analyzed Content Component

User Settings

Users also have the possibility to reset password and activate MFA (fig.8.15) through the settings.

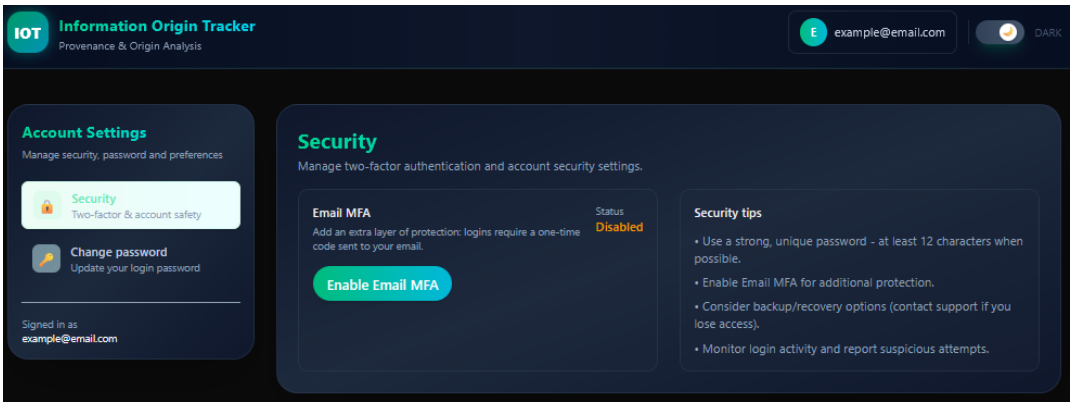

Figure 8.20: IOT User Settings

In summary the UI does not require many elements, as it only retrieves sources of a piece of information, resulting in a simple and straightforward solution accessible to everyone. This approach also promotes the sharing of knowledge and transparent access to information, contributing to several of the United Nations Sustainable Development Goals:

- **Quality Education:** supporting informed learning by leaving the assessment of the sources to the user.
- **Peace, Justice and Strong Institutions:** showcasing verifiable information.
- **Partnerships for the Goals:** sharing knowledge and collaboration across platforms.

8.2.7 System Deployment and Scalability

As discussed in Subsection 4.1.6, Kubernetes was selected for this project due to its architecture, support for SOA, scalability features, and alignment with the project requirements. This sections describes in depth the technical implementation of the deployment environment, hosted within the Windows Subsystem for Linux 2 (WSL2). To run this system on the available machine, a container runtime and Kubernetes components, such as Kubeadm, Kubectl, and Kubelet, were required.

8.2.7.1 Container Runtime

Kubernetes environment utilizes `containerd` as the container runtime interface (CRI). A container runtime is the foundational software that allows containers to operate within a host system. Container runtime is responsible for everything from pulling container images from a container registry and managing their life cycle to running the containers on your system. [21]

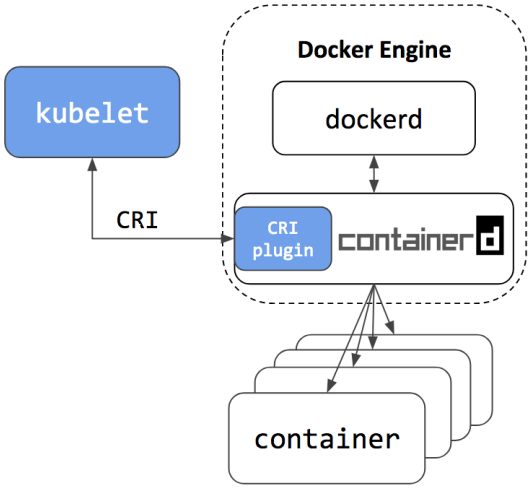

Figure 8.21: Containerd Diagram [22]

Containerd was configured to use `systemd` as the cgroup driver, ensuring resource isolation within the WSL2 environment and preventing conflicts between Kubelet and the container runtime.

Kubelet
The kubelet is the primary "node agent" that runs on each node. It can register the node with the apiserver using one of:

- the hostname
- a flag to override the hostname
- specific logic for a cloud provider.

It works in terms of a PodSpec, a YAML or JSON object that describes a pod. The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the

containers described in those PodSpecs are running and healthy. This node agent doesn't manage containers which were not created by Kubernetes. [23]

8.2.7.2 Cluster Initialization

To initialize the cluster within the WSL2 environment the installation of required tools, such as Kubeadm and Kubectl, were necessary. The API server was advertised on a static IP Tailscale address to ensure consistent reachability after reboots, additionally the Pod network CIDR was set to 10.10.0.0/16 to avoid conflicts with local networks.

Kubeadm
Kubeadm is a tool used to build Kubernetes (K8s) clusters. Kubeadm performs the actions necessary to get a minimum viable cluster up and running quickly. By design, it cares only about bootstrapping, not about provisioning machines (underlying worker and master nodes). Kubeadm also serves as a building block for higher-level and more tailored tooling. [24] *Initialization of the cluster:*

```
sudo kubeadm init \
  --pod-network-cidr=10.10.0.0/16 \
  --apiserver-advertise-address=<tailscale_ip_address> \
  --control-plane-endpoint=<tailscale_ip_address>
```

kubectl
kubectl is a command line tool that enables communications between the Kubernetes API and the control plane. kubectl allows application deployment, cluster resource management, and resource monitoring. Overall, all CRUD operations (Create, Read, Update, and Delete) on Kubernetes resources are carried out with kubectl's help. This tool is one of the most crucial for managing the complete Kubernetes infrastructure, where applications may be distributed across different clusters. It facilitates communication between users, Kubernetes resources, and external components by sending HTTP requests to the Kubernetes API server. [25]
Syntax of a kubectl command:

```
    kubectl [command] [TYPE] [NAME] [flags]
```

Container Network Interface (CNI)
To enable communication between pods, the **Calico** network plugin was used via Tigera Operator. Calico was chosen for its support of network policies and IP address management, the custom resources were configured to match the Pod network CIDR.

```
# 1. Install Operator
kubectl create -f https://.../tigera-operator.yaml
# 2. Download Custom Resources
curl https://.../custom-resources.yaml -O
# 3. Update CIDR to match kubeadm init (10.10.0.0/16)
sed -i 's/cidr: 192\.168\.0\.0\/16/cidr: 10.10.0.0\/16/g' custom-
    resources.yaml
# 4. Apply
kubectl create -f custom-resources.yaml
```

These steps are necessary to transition the node into a Ready state. To verify the successful deployment of Calico, the following command was used:

```
jimpo@Jimpo:~/aau-sw7-iot/k8s$ kubectl get nodes
NAME    STATUS   ROLES           AGE   VERSION
jimpo   Ready    control-plane   11d   v1.30.14
jimpo@Jimpo:~/aau-sw7-iot/k8s$ kubectl get pods -n calico-system
NAME                                       READY   STATUS    RESTARTS   AGE
calico-kube-controllers-7ddbfbc7fc-sqr7r   1/1     Running   0          49s
calico-node-mmd6x                          1/1     Running   0          49s
calico-typha-74547cd877-hld4r              1/1     Running   0          49s
csi-node-driver-bhpdm                      2/2     Running   0          49s
```

Figure 8.22: Calico Pods initialized and Node in ready status

Once the node transitions from a NotReady status to a Ready status, Kubernetes can begin scheduling Pods. This enables container images to be pulled from DockerHub, a cloud-based repository service, and initialized within the cluster.

```
jimpo@Jimpo:~/aau-sw7-iot/k8s$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
email-service-c9654dd6b-h2cqs       1/1     Running   0          2d4h
frontend-588d9fff98-9htxk           1/1     Running   0          3h16m
maildev-cb77d8b7b-lhf7x             1/1     Running   0          2d4h
middleware-api-f64d9cd64-ljfkk      1/1     Running   0          3h53m
nlp-service-695764b877-vwdln        1/1     Running   0          5h21m
postgres-local-0                    1/1     Running   0          8d
web-search-64bd949dfd-7pl8s         1/1     Running   0          4h43m
web-search-64bd949dfd-qh8l6         1/1     Running   0          4h43m
jimpo@Jimpo:~/aau-sw7-iot/k8s$
```

Figure 8.23: Pods within the cluster

8.2.7.3 GPU Enabling in WSL

A necessary requirement for the NLP module (running the Qwen LLM) was to access the hardware. To enable this feature inside WSL2 an additiona configuration was needed:

1. **NVIDIA Container Toolkit:** This toolkit installed in the WSL2 environment enables passing GPU to the containers.
2. **NVIDIA Device Plugin:** A Kubernetes DaemonSet was deployed to advertise GPU resources to the scheduler.
3. **Resource Limits:** The NLP pod requests GPU resources, ensuring that the pod is scheduled on a node with hardware acceleration.

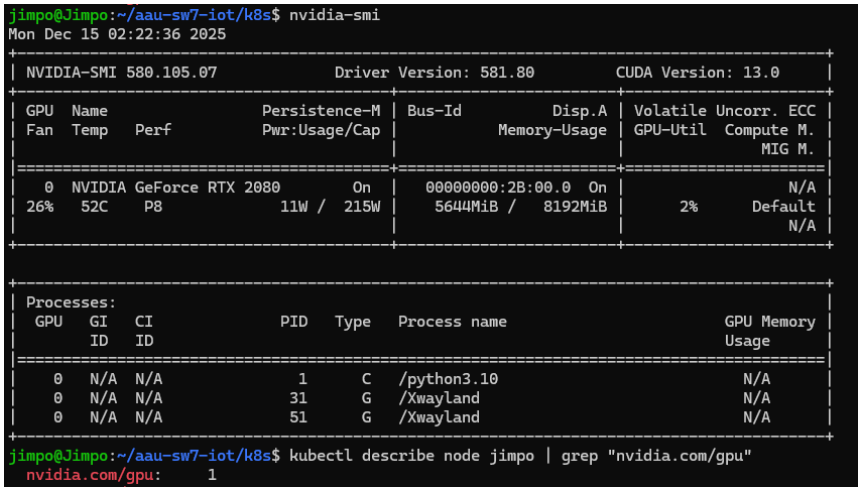

Figure 8.24: Output of `nvidia-smi` running inside a Kubernetes pod verifying GPU access.

8.2.7.4 Services

Each pod hosts a service implemented using different technologies, including FastAPI for the web search and NLP services, a React application for the frontend, and a Spring Boot application for the middleware layer.
Each module run as an independent pod within the cluster, each exposing their endpoints internally via Kubernetes Services (ClusterIP), allowing the Middleware API to communicate with them via DNS names (e.g., `http://<service_name>:<port>`) rather than hard-coded IPs.

8.2.7.5 Connection and Tunneling Layers

Exposing a Kubernetes cluster running inside WSL2 environment on a local Windows machine to the public internet required a layered networking approach to overcome dynamic IP limitations:

- **WSL2 Internal IP Address:** WSL2 uses a dynamic IP address that changes upon every restart.
- **Tailscale Mesh VPN:** Tailscale provides a stable and static IP address that persists. The Kubernetes API server was configured to bind to this interface.
- **Ngrok Tunneling:** To expose the project without using the IP address of the machine, ngrok was chosen as it is a reverse proxy, which masks the IP address. This tunnel connects the NodePort or LoadBalancer to a public URL.

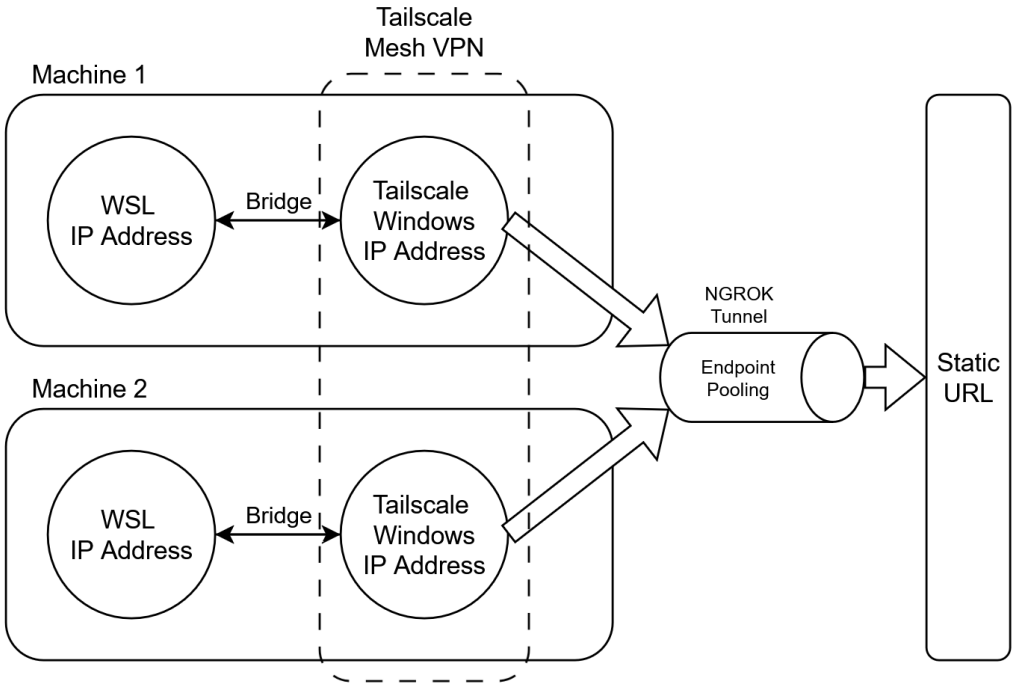

Figure 8.25: Visual representation of the connection layers.

ngrok

ngrok is a globally distributed reverse proxy that secures, protects, and accelerates your applications and network services. It is environment independent because it can deliver traffic to services running anywhere with no changes to the environment's networking. ngrok is a unified platform because it combines all the components to deliver traffic from the services to the internet into one. [26]

Endpoint pooling makes load balancing simple, it helps scale traffic across multiple replicas of our app to increase capacity, tolerate failures and geo-distribute load. When creating two endpoints with the same URL (and binding), those endpoints automatically form a "pool" and share incoming traffic.[27]

```
# Create two endpoints with the same URL forwarding
ngrok http 8080 --url https://your-domain.ngrok.app --pooling-enabled
    true
ngrok http 9090 --url https://your-domain.ngrok.app --pooling-enabled
    true
```

This implementation ensures that the system, although complex, have the proper security measures and a reliable deployment environment. This approach prevents the Single Point of Failure (SPoF) issue and promotes high availability and fault tolerance thanks to ngrok, which through it endpoint pooling will automatically forward the requests to the available system.

8.3 Data Flow and Interactions

The management of the services were managed using the `k8s-config.yaml` configuration file. This file defines all the different services (mentioned in subsection 8.2.7.4) and their connection. Kubernetes ensures that each service runs in its own pod while accessing shared secrets, network configuration and scaling policies.
Key components in the configuration file:

- **Secrets:** Credentials, API tokens and environment variables.
- **Deployments:** Each service is deployed via Deployment resource which manages the amount of replicas. Each service is deployed via a `Deployment` resource, which manages the desired number of replicas, container images and environment variables.
- **Services:** Each service deployment exposes stable endpoints for connection, internally or externally.

A template example for the configuration file is shown below:

```
# Example Secret
apiVersion v1
kind Secret
metadata
  name app-secrets
stringData
  POSTGRES_USER "username"
  POSTGRES_DB "cloud_db"
  POSTGRES_PASSWORD "password"
  API_TOKEN "token_value"

---
apiVersion apps/v1
kind Deployment
metadata
  name example-service
spec
  replicas 1
  selector
    matchLabels
      app example-service
  template
    metadata
      labels
        app example-service
    spec
      containers
      - name example-container
        image example/imagelatest #pulling image
        ports
          - containerPort 8080
        env
```

```
          - name API_TOKEN
            valueFrom
              secretKeyRef
                name app-secrets
                key API_TOKEN

---
# Example Service
apiVersion v1
kind Service
metadata
  name example-service
spec
  selector
    app example-service
  ports
    - port 8080
      targetPort 8080
```

This template illustrates the patterns in which the services were deployed, for a better understanding of the system refer to the image below:

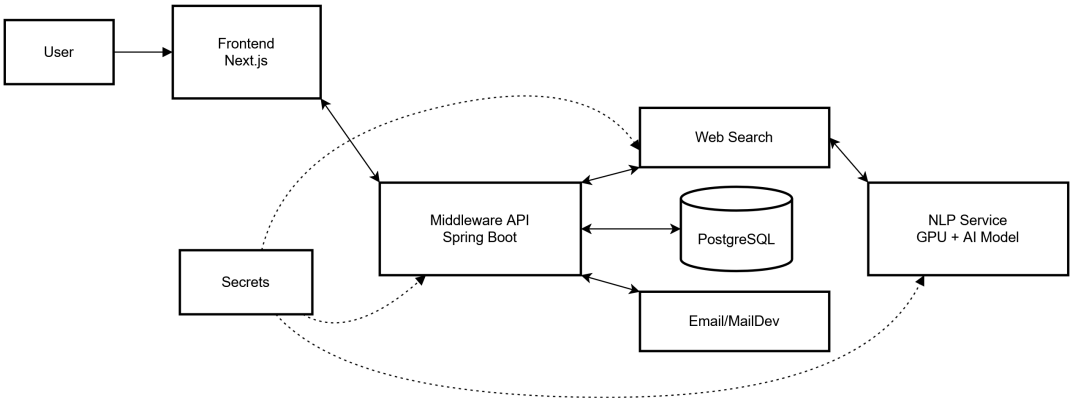

Figure 8.26: Diagram of the communication between deployed services.

Part V

Testing and Evaluation

Chapter 9

Testing

This chapter details the testing strategy applied in order to assess the project's functionality.

9.1 LLM Evaluation Metrics

The selected LLMs were tested and thoroughly compared in order to determine the most fitting for our project.
Since both Gemma 3 4B and Qwen 3 4B models were released earlier this year(2025), a direct comparison between the two can only be found on community benchmarking platforms. An example is Artificial Analysis [28], which indicates that Qwen outperforms Gemma across most categories. Despite these results, these metrics do not always translate into a better choice for our project, for this reason a customized benchmarking approach was adopted.

Speed Comparison
To ensure a better user experience, the models evaluation was also based on the latency of the response and token generation rate:

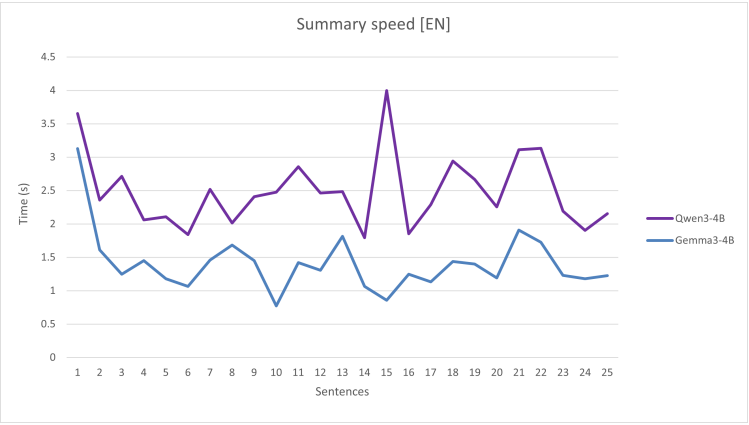

Figure 9.1: Comparison of Response Latency

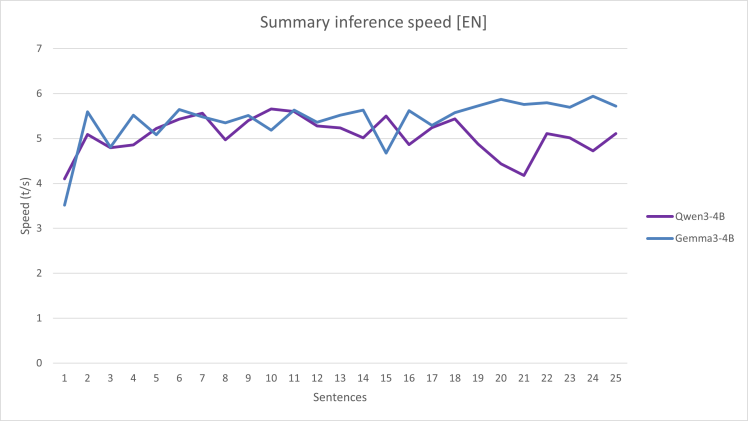

Figure 9.2: Comparison of Token Generation Rate

Search Score
To align the benchmark with the project's scope, a custom "search score" was defined by combining different weighted submetrics:

- **Semantic Similarity**: Semantic closeness with the original input using `cosine_similarity`.
- **Keyword Retention**: How many keywords are preserved.
- **Number Retention**: How many relevant numbers are preserved.

These components are combined into a single score using chosen weights that prioritize keyword preservation:

$$\text{Search Score} = 0.3 \cdot \text{Semantic Similarity} + 0.4 \cdot \text{Keyword Retention} + 0.3 \cdot \text{Number Retention}$$

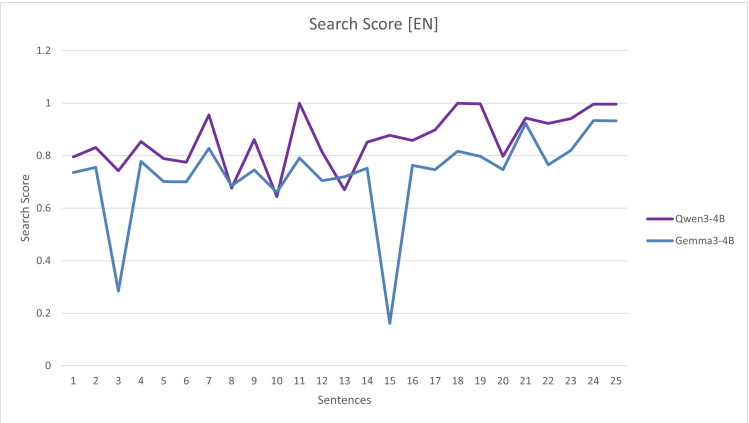

Figure 9.3: Search Score of the 2 models with english sentences.

Additional tests with sentences in different languages were conducted, however they returned similar results.

Although Gemma proved itself to be faster than Qwen in both response latency and token generation rate, Qwen is much more consistent and overall outperforms Gemma in the search score. Given the search score result, the time difference in the speed benchmarks is negligible, as Qwen returns better results in a short period of time.

9.2 Web Search and NLP modules stress test

To test the Web Search and NLP module, these services had to be deployed independently and exercised through direct requests. A high volume of requests were sent to assess the stability and reliability of the services, monitoring whether they would degrade or crash under heavy load.

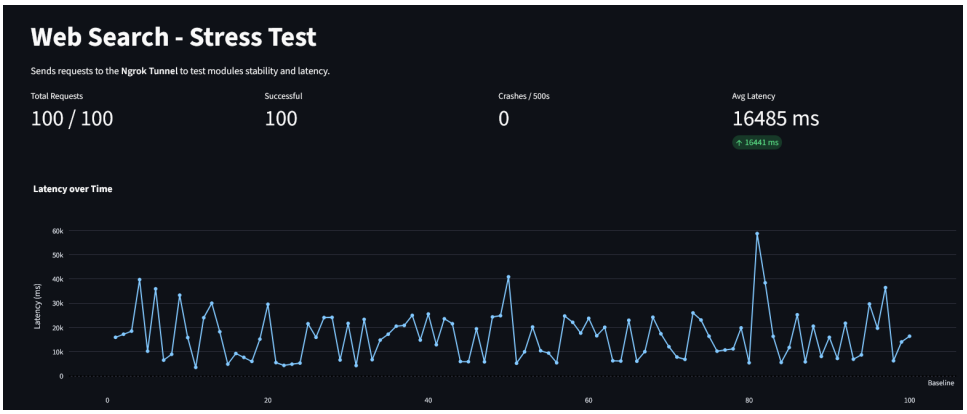

Figure 9.4: Stress test with endpoint /text/all.

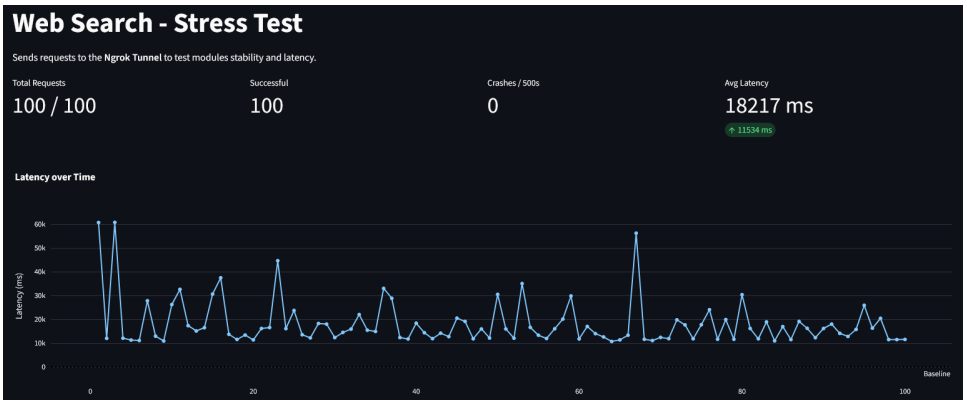

Figure 9.5: Stress test with endpoint /link/all.

The tests were run on the local machine connected to a home wifi, which could affect the results and under specific circumstances cause crashes and lose requests before restarting the pod:

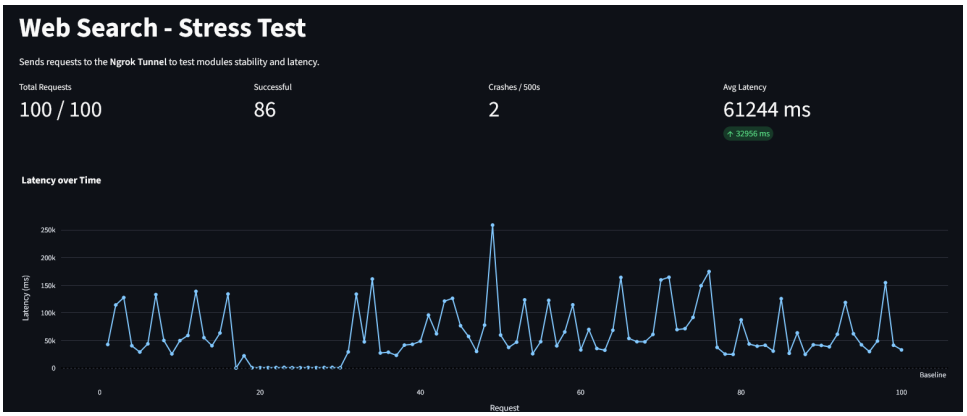

Figure 9.6: Stress test using endpoint /link/all with failures.

9.3 Data Retrieval

As mentioned in Section 8.2.2, the search engine relies on the IP address of the machine to determine the region from which the information is retrieved. Despite forcefully setting the market to en-US, some results may still appear in other languages. Additionally, some specific recent terms used in short sentences may return empty lists.

9.4 Edge Cases

During the stress testing phase, the system processed a wide variety of URLs. While the majority were handled successfully, specific edge cases revealed limitations and errors in the processing pipeline:

- **Unsupported Format:** The input link could contain a format unsupported by our system, causing error.
- **Content Insufficiency:** If the webpage lacks text and context, the `newspaper3k` library wouldn't be able to determine the top-k sentences.
- **Semantic Divergence:** The LLM could strip away important pieces of information and generate unrelated search terms, causing errors or loops.
- **Unrelated Sources:** The region of the search emulation is based on the IP address of the machine, which could return unrelated results in a different language.

Chapter 10

Evaluation

This chapter compares the Information Origin Tracker against its initial objectives, analyzing system performance and architectural robustness.

10.1 Functionality

The final product successfully delivers the MVP defined initially. The system processes both raw text and URL inputs, utilizing the whole processing pipeline of the modules. The frontend correctly showcases the different results and filters them, meeting the requirement of providing a structured result.

10.2 Performance Analysis

The stress tests and model benchmarks highlighted a trade-off between speed and accuracy that dictated the final system configuration. As shown and said in Section 9.1, we chose the more accurate and reliable model due to the negligible time difference.

10.2.1 System Throughput

The results of an average request latency heavily depend on the amount of information to process. For example, the figures 9.6 and 9.4 both leverage the LLM and NLP processing pipeline; however one takes 16 seconds while the other mote than 1 minute. Despite this performance constraint, the system rarely crashed and displayed a success rate of 100% majority of the time.

10.3 Architectural Robustness

The SOA deployed on Kubernetes was a vital step for resource management, allocating the necessary hardware to the different pods for an enhanced performance.

Part VI

Improvements and Conclusion

Chapter 11

Potential Improvements

The final product not only the initial MVP defined at the beginning of the development cycle, but also integrated additional features beyond the original scope. However, despite these achievements, there are multiple areas where further improvement could polish the user experience and enhance the system's overall functionality.

11.1 Frontend

The user interface focuses on simplicity, but this is a double edge sword, as without much interaction it would be difficult to keep users engaged. Additionally, not all users are inclined to manually write down pieces of information or copy and paste URL links. The following improvements help tackle the issues listed:

- **Browser Extension:** Instead of relying on a web page, users can directly install a browser extension and access all the capabilities and features of the product, without the need to access the application.
- **Improved entity recognition feature:** To enhance the user experience a better highlighting of entities inside the content analysis would result in a more readable solution, also providing the user with a filter to decide what type of entity to highlight. Additionally, it should also be possible to click said entity and redirect the user to a specific webpage that describes it (e.g. Wikipedia).

11.2 Middleware

The middleware layer is responsible for handling all the requests and communication between modules, although it works accordingly, there are some missing features that could drastically improve its performance and reliability:

- **Protection:** each request could be handled with temporary tokens and an additional rate limiting would prevent possible threats.

- **Distinct Job IDs:** Currently each request and process will run until it returns the results. However, despite this, in order to promote scalability and high availability each job must have a different ID for better management and provide the user with the possibility to interrupt the process when requested.

Furthermore, this layer could also manage the amount of requests and temporary session for unregistered users, ensuring controlled access to the system.

11.3 Database

The database currently fulfills the role of persistent data storage; however, it could be improved with additional mechanisms such as triggers, constraints, and stored procedures to automate data processing tasks, manage data refresh cycles, and handle inactivity-related logic more efficiently. Additionally, enabling the `pgvector` extension would be a necessary requisite to support the vector storage requirements of the NLP module.

11.4 Web Search

Web Scraping proved to be one of the most challenging aspects of this project due to its ever changing landscape. Modern websites and browsers implement various anti-bot mechanisms that prevent data scraping, which limits the effectiveness of free solutions. These restrictions make it really difficult to achieve an in depth web scraping process. Integrating powerful scraping APIs could substantially improve the current proposed solution, providing a broader coverage, easy access, and support for different types of content formats. Additionally, search engine results are often based on the IP address of the machine performing the request. To fully address this issue, future versions of the system could implement the use of VPNs to control the location; furthermore, it would allow users to explicitly select the target market and retrieve information from a preferred region or in a preferred language.

11.5 Natural Language Processing

Given the complexity of the natural language processing pipeline and its reliance on large language models, several areas could be further improved. While LLMs proved to be somewhat consistent in overall tasks, they may still hallucinate, especially when the input lacks context or ambiguous information. This will lead to generation of search queries that diverge from the initial meaning of the input. To mitigate this issue it is possible to improve the prompt engineering, which could lead to more faithful queries; however due to the probabilistic nature of LLMs, this behaviour is not always guaranteed.

Context Preservation
Another key challenge is context preservation, the parsing and segmentation of sentences after the tokenization process can sometimes lead to the loss of important words related to the context of the sentence. Furthermore, text extraction phase heavily realies on the `newspaper3k` library, which performs well on articles but may file when the source lacks

information and context (e.g. Pages dominated by pages, lists, etc...). This complex processing could be improved through the implementation of a combination of different libraries and merging results, however this would affect the processing time significantly.

Semantic Search
Significant performance improvements can also be achieved through a more advanced and optimized caching mechanism. This approach embeds the information into a vector and performs a semantic similarity search against the stored vectors, providing relevant results directly from the cache instead of performing the full processing pipeline. This reduces latency and redundant operations, optimizing the tracking process by prioritizing contextually relevant results.

Additional resources
Processing time primarily depends on GPU performance and availability, which represent a hardware constraint when performing demanding tasks. To overcome these limitations, horizontal scaling can be implemented by integrating additional GPU resources into the system and distributing workloads across them. This would create a multiple nodes environment withing each cluster, where tasks are partitioned and load balanced across different pods. In case where horizontal scaling is not possible, we can leverage cloud-based solutions, which offer easy access and implementation while also maintaining high performance and low latency under heavy loads.

11.6 Architecture

The final architecture of the product is reliable and fault tolerant, ensuring that the services are fully operational and functional under normal loads. However, to further enhance resilience, availability and fault tolerance a few improvements can be considered:

- **Cloud Services:** Integrate additional cloud services to increase reliability, scalability and high availability.
- **Expansion:** Add more clusters and nodes to improve performance and promote fault tolerance and high availability.
- **Secure Architecture:** Implement a zero-trust architecture between pods and nodes to enhance security and add a layer of protection against both internal and external threats.

11.7 Subscription

The implementation of a subscription-based approach would allow users to access a more powerful web scraping mechanism, which would require longer processing times but return more reliable and comprehensive results. Users without subscription still have access to the basic functionalities of the product, although limited.

11.8 Data Provenance Tree

As defined in Section 2.1, the single point of origin is what our project revolves around. However, to fully understand the evolution and spreading of a single piece of information, users can benefit from a visual component that displays the full lineage of the information. To achieve this result the system can implement an algorithm to build a Data Provenance Tree.

This module integrates a customized Greedy Algorithm combined with semantic vector search and time awareness. Unlike traditional greedy algorithms, this one requires respecting causality, therefore a directed acyclic graph is the optimal solution for this approach.

11.8.1 Algorithmic Approach

Since our project returns the origin of a piece of information, the construction of the Provenance Tree starts from the root downwards. The implementation integrates a k-Nearest Neighbors (k-NN) algorithm that is also aware of time, by finding the most semantically similar descendant d^* for each source s_i:

$$d^* = \underset{h \in H}{\text{argmax}}(\text{CosineSimilarity}(\vec{s_i}, \vec{h})) \tag{11.1}$$

with:

- s_i**:** the current source node
- H**:** all sources with publication date after s_i that are not yet in the tree

This approach connects each source to its most semantically similar successor, building a tree that traces how information spread while also respecting causality.

11.8.2 Complexity Analysis

The algorithm has $O(N^2)$ time complexity, where N is the number of sources. This happens because, in the worst case, each source must be compared against all later sources to find its best child:

$$\text{Total comparisons} = \sum_{i=1}^{N}(N-i) = \frac{N(N-1)}{2} \in O(N^2)$$

This kind of process significantly affects the system's overall performance, by introducing a substantial amount of workload, data processing and additional web scraping.

Algorithm 1 Time Aware Semantic Provenance Construction

Data: Source collection S
Result: Provenance tree T

```
// Sort sources by ascending publication date
```
$S_{sorted} \leftarrow \text{SortByDateAscending}(S)$;
$T \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** $|S_{sorted}|$ **do**
  $s_i \leftarrow S_{sorted}[i]$;
  $best\_child \leftarrow$ null;
  $max\_score \leftarrow 0$;
  `// Search among temporally valid successor sources and compare cosine similarity`
  for $j \leftarrow i + 1$ **to** $|S_{sorted}|$ **do**
    $h \leftarrow S_{sorted}[j]$;
    $score \leftarrow \text{CosineSimilarity}(\vec{s_i}, \vec{h})$;
    if $score > max\_score$ ***and*** $score > THRESHOLD$ **then**
      $max\_score \leftarrow score$; $best\_child \leftarrow h$;
    end
  end
  if $best\_child \neq null$ **then**
    T.addEdge$(s_i \rightarrow best_child)$;
  end
end
return T;

A visual representation of the tree that the user will view, providing them with additional sources to assess rather than the origin only:

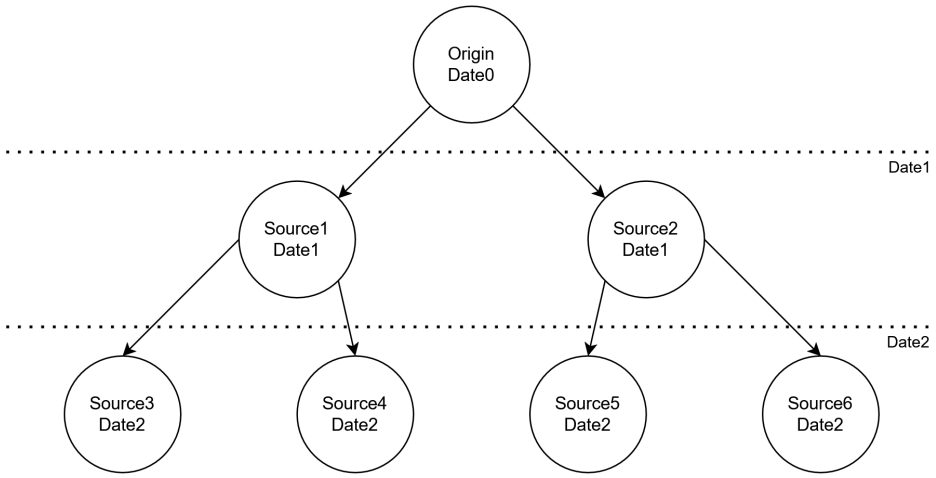

Figure 11.1: Data Provenance Tree from Origin.

11.9 Accessibility Features

To ensure the availability of the product to include a wide range of users with different impairments, accessibility features can be integrated into the system.

- **Visual Impairment:** Improved support for screen reading softwares by structuring and labelling elements properly.
- **Color Blind Friendly:** Integration of a Color Blind Friendly filter.

Implementing these enhancements would contribute to a more inclusive system and improve overall usability for users with diverse needs.

Chapter 12

Conclusion

In conclusion, by reflecting on the initial problem statement in Section 1.3:

> **How can we design a system that determines the earliest known source of information from a text or webpage and additionally provides the user with a list of all fetched URLs and websites?**

We can determine that all the requirements were fulfilled. However, despite reaching the goal we can also conclude that due to the complex processing involved the returned sources could still be unreliable.

The Information Origin Tracker project successfully addresses the challenge of digital information transparency by delivering a functional, scalable and modular solution. Through the integration of advanced Natural Language Processing and a robust architecture, the product provides the user with a tool to track the Origin of a piece of information.

Thus we can conclude that despite the numerous hardware and architectural limitations and constraints, the team successfully delivered a fully functional solution that addresses the initial problem statement, resulting in a reliable, fault tolerant and highly available solution.

Furthermore, as detailed in Chapter 11, additional improvement will drastically improve and enhance the user experience while also strengthening the overall structure of the system.

Bibliography

[1] Usman Hasan Khan Astera Marketing Team. *Exploring Data Provenance: Ensuring Data Integrity and Authenticity*. Tech. rep. Accessed: 2025-12-02. Astera, 2025. URL: https://www.astera.com/type/blog/data-provenance (Cited on page 6).

[2] *What is LLM (Large Language Model)?* Tech. rep. Accessed: 2025-12-02. Amazon Web Services, 2025. URL: https://aws.amazon.com/what-is/llm (Cited on page 8).

[3] *Large Language Models Explained*. Tech. rep. Accessed: 2025-12-02. NVIDIA, 2025. URL: https://www.nvidia.com/en-us/glossary/large-language-models (Cited on page 8).

[4] Donald Le. *Docker Swarm vs Kubernetes: A Practical Comparison*. Tech. rep. Accessed: 2025-12-04. BetterStack, 2025. URL: https://betterstack.com/community/guides/scaling-docker/docker-swarm-kubernetes (Cited on pages 10, 11).

[5] Mehul Budasna Reynal Dsouza. *Kubernetes vs Docker Swarm: A Comprehensive Comparison Between The Tools*. Tech. rep. Accessed: 2025-12-04. Bacancy, 2025. URL: https://www.bacancytechnology.com/blog/kubernetes-vs-docker-swarm (Cited on page 10).

[6] *An Overview of Kubernetes Architecture*. Tech. rep. Accessed: 2025-12-04. OpsRamp, 2025. URL: https://www.opsramp.com/guides/why-kubernetes/kubernetes-architecture (Cited on page 11).

[7] Michael Chen. *What Is SOA (Service-Oriented Architecture)?* Tech. rep. Accessed: 2025-12-02. Oracle, 2025. URL: https://www.oracle.com/nl/service-oriented-architecture-soa/ (Cited on page 21).

[8] Software Development Community. *An Overview of Kubernetes Architecture*. Tech. rep. Accessed: 2025-12-04. Medium, 2025. URL: https://medium.com/@SoftwareDevelopmentCommunity/what-is-service-oriented-architecture-fa894d11a7ec (Cited on page 21).

[9] *Recommended Hardware for Running LLMs Locally*. Tech. rep. Accessed: 2025-12-03. GeeksforGeeks, 2025. URL: https://www.geeksforgeeks.org/deep-learning/recommended-hardware-for-running-llms-locally (Cited on page 22).

[10] Alex Strick van Linschoten Sofie Van Landeghem. *Tokenization*. Tech. rep. Accessed: 2025-12-08. spaCY, 2025. URL: https://spacy.io/usage/linguistic-features#how-tokenizer-works (Cited on page 38).

[11] *Transformers*. Tech. rep. Accessed: 2025-12-09. Hugging Face, 2025. URL: https://huggingface.co/docs/transformers/en/index (Cited on page 43).

[12] Wikipedia. "Transformer (deep learning)". In: (2025). Accessed: 2025-12-09. URL: https://en.wikipedia.org/wiki/Transformer_(deep_learning) (Cited on page 43).

[13] *PyTorch*. Tech. rep. Accessed: 2025-12-10. PyTorch, 2025. URL: https://pytorch.org/projects/pytorch (Cited on page 44).

[14] Preferred Networks. *How Computational Graphs are Constructed in PyTorch*. Tech. rep. Accessed: 2025-12-10. PyTorch, 2025. URL: https://pytorch.org/blog/computational-graphs-constructed-in-pytorch/ (Cited on page 44).

[15] *CUDA C++ Programming Guide (Legacy)*. Tech. rep. Accessed: 2025-12-11. NVIDIA, 2025. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/ (Cited on page 45).

[16] *Quantization - Getting Started*. Tech. rep. Accessed: 2025-12-11. Hugging Face, 2025. URL: https://huggingface.co/docs/diffusers/quantization/overview (Cited on page 46).

[17] *Gemma explained: An overview of Gemma model family architectures*. Tech. rep. Accessed: 2025-12-12. Google, 2025. URL: https://developers.googleblog.com/en/gemma-explained-overview-gemma-model-family-architectures/ (Cited on page 47).

[18] Shubham Kumar. *Spring Boot - Spring Data JPA*. Tech. rep. Accessed: 2025-12-13. GeeksForGeeks, 2025. URL: https://www.geeksforgeeks.org/java/spring-boot-spring-data-jpa/ (Cited on page 54).

[19] Pranay Gaikwad. *Basic Introduction to Spring WebFlux*. Tech. rep. Accessed: 2025-12-13. GeeksForGeeks, 2025. URL: https://www.geeksforgeeks.org/advance-java/basic-introduction-to-spring-webflux/ (Cited on page 54).

[20] VMware Tanzu. *Spring Security*. Tech. rep. Accessed: 2025-12-13. Spring, 2025. URL: https://spring.io/projects/spring-security (Cited on page 54).

[21] Nicolas Ehrman. *Container Runtimes Explained*. Tech. rep. Accessed: 2025-12-14. Wiz, 2025. URL: https://www.wiz.io/academy/container-security/container-runtimes (Cited on page 64).

[22] Mike Brown Lantao Liu. *Container Runtimes Explained*. Tech. rep. Accessed: 2025-12-14. Kubernetes, 2025. URL: https://kubernetes.io/blog/2018/05/24/kubernetes-containerd-integration-goes-ga/ (Cited on page 64).

[23] *kubelet*. Tech. rep. Accessed: 2025-12-15. Kubernetes, 2025. URL: https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/ (Cited on page 65).

[24] *Kubeadm Tutorial*. Tech. rep. Accessed: 2025-12-15. Densify, 2025. URL: https://www.densify.com/kubernetes-tools/kubeadm/ (Cited on page 65).

[25] *kubectl*. Tech. rep. Accessed: 2025-12-15. Armosec, 2025. URL: https://www.armosec.io/glossary/kubectl/ (Cited on page 65).

[26] *What is ngrok*. Tech. rep. Accessed: 2025-12-15. ngrok, 2025. URL: https://ngrok.com/docs/what-is-ngrok (Cited on page 68).

[27] *Endpoint pooling*. Tech. rep. Accessed: 2025-12-15. ngrok, 2025. URL: https://ngrok.com/docs/universal-gateway/endpoint-pooling (Cited on page 68).

[28] *Gemma 3 4B Instruct vs. Gemma 3 4B Instruct*. Tech. rep. Accessed: 2025-12-16. Artificial Analysis, 2025. URL: https://artificialanalysis.ai/models/comparisons/gemma-3-4b-vs-gemma-3-4b?models=gemma-3-4b%2Cqwen3-4b-2507-instruct (Cited on page 72).

Appendix A

Source Code Listings

This appendix contains the complete code implemented in the system's components.

A.1 NLP Module

A.1.1 Pipeline Execution

The following function details the pipeline execution of the NLP module.

Listing A.1: NLP Pipeline Execution

```
def execute_pipeline(self, input, top_x:int=5, query_variations:int=1,
    do_ner=True) -> List[dict]:
        searchterms = None
        entities = None

        if isinstance(input, Article):
            searchterms = self.process_article(input)
        else:
            searchterms = self.process_raw_text(input, top_x)

        if do_ner:
            if isinstance(input, Article):
                entities = self.find_entities(input.text)
            else:
                entities = self.find_entities(input)

            for s in searchterms:
                s["entities"] = []
                for e in entities:
                    if e["name"] in s["sentence"]:
                        s["entities"].append(e)

        if query_variations <= 1:
            return searchterms

```

```
        res = []
        for query in searchterms:
            res.append(query)
            variations = self.query_variations(query["search_term"],
                query_variations)
            for var in variations:
                if var.lower() != query["search_term"].lower():
                    res.append({"sentence": query["sentence"], "
                        search_term": var})

        return res
```

A.1.2 Query generation

The following function details the query generation of the LLM.

Listing A.2: NLP Pipeline Execution

```
def generate_search_term(self, sentence: str) -> str:
        messages = [
            {
                "role": "system",
                "content": "You are an assistant that helps the user
                    search the internet. You receive a sentence as
                    input, and you must output a Google search style
                    string. Respond in the same language as the input."
            },
            {
                "role": "user",
                "content": "Input: President Donald Trump was back in
                    public Tuesday to announce a new location for US
                    Space Command headquarters"
            },
            {
                "role": "assistant",
                "content": "US Space Command new location"
            },
            {
                "role": "user",
                "content": "Input: Nvidia agreed to invest $5 billion
                    in the American chip maker, which will see Intel
                    design custom x86 chips for it."
            },
            {
                "role": "assistant",
                "content": "Nvidia Intel investment"
            },
            {
                "role": "user",
                "content": f"Input: {sentence}"
            }
```

```
        ]

        prompt = self.llm.pipeline.tokenizer.apply_chat_template(
            messages,
            tokenize=False,
            add_generation_prompt=True
        )

        terminators = [
            self.llm.pipeline.tokenizer.eos_token_id
        ]

        possible_stop_tokens = ["<|im_end|>", "<|endoftext|>"]

        for token in possible_stop_tokens:
            token_id = self.llm.pipeline.tokenizer.
                convert_tokens_to_ids(token)
            if token_id is not None:
                terminators.append(token_id)

        outputs = self.llm.pipeline(
            prompt,
            max_new_tokens=50,
            eos_token_id=terminators,
            do_sample=False,
        )

        generated_text = outputs[0]['generated_text']
        answer = generated_text[len(prompt):].strip()

        return answer
```

A.2 Web Search Module

A.2.1 Stealth Session Implementation

The following class showcases the StealthSession logic used to bypass detection mechanisms, it mimics legitimate browser behaviour and implements retry logic with delays to avoid rate limiting.

Listing A.3: StealthSession Class Implementation

```
class BaseStealthSession:
    def __init__(self, **kwargs):
        timeout = kwargs.pop('timeout', 30)
        headers = kwargs.pop('headers', {})
        # Selecting random user-agent to mimic user
        headers.setdefault('User-Agent', random.choice(user_agents))
        # Tracking last url
        self.last_request_url = None
```

```
        super().__init__(impersonate='chrome', timeout=timeout,
            headers=headers, **kwargs)


class StealthSession(BaseStealthSession, Session):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    def request(self, method: HttpMethod, url: str, *args, retry: int
        = 0, **kwargs) -> StealthResponse:
        assert retry >= 0
        # Set the "Referer" header to mimic natural navigation
        referer = {'Referer': self.last_request_url} if self.
            last_request_url else {}
        extra_headers = referer | kwargs.pop('headers', {})

        for attempt in range(retry + 1):
            resp = super().request(method, url, *args, headers=
                extra_headers, **kwargs)
            response = StealthResponse(resp)

            # Break loop if successful or if max retries reached
            if resp.status_code not in RETRYABLE_STATUS_CODES or
                attempt == retry:
                break

            # Retry after delay to prevent rate limiting
            time.sleep(RETRY_DELAY)

        # Mimic human behaviour
        parsed = urlparse(url)
        self.last_request_url = urlunparse(parsed._replace(query='',
            fragment=''))

        return response

    head = partialmethod(request, 'HEAD')
    # Remaining partial methods (get, post, etc.) handled by parent
```

A.2.2 Custom Date Retrieval

The following snippet details the logic used to extract data from Bing SERP elements.

Listing A.4: Custom Date Retrieval

```
container_selectors = ".news-card, li.b_algo"
if search_type == 'web':
    container_selectors = "li.b_algo"

for item in soup.select(container_selectors):
```

```
    title, url, snippet = None, None, ""
    date = None
    date_text = ""
    a_tag = None

    try:
        # news-card structure
        if item.name == 'div' and 'news-card' in item.get('class', []):
            a_tag = item.find("a", class_="title")
            snippet_tag = item.find("div", class_="snippet")
            date_tag = item.find("span", class_="news_dt")
            if date_tag:
                date_text = date_tag.get_text(strip=True)

        # li.b_algo structure
        elif item.name == 'li' and 'b_algo' in item.get('class', []):
            h2 = item.find("h2")
            a_tag = h2.find("a") if h2 else None
            snippet_tag = item.select_one(".b_caption p") or item.find("p")

            # Logic to scrape date text from attribution/metadata elements
            attribution_tag = item.select_one(".b_attribution")
            if attribution_tag:
                all_attr_texts = [text.strip() for text in attribution_tag.find_all(string=True, recursive=False)]
                combined_attr_text = " ".join(filter(None, all_attr_texts))
                spans_inside_attr = attribution_tag.find_all('span')
                span_texts = " ".join([span.get_text(strip=True) for span in spans_inside_attr])

                for text_to_parse in [combined_attr_text, span_texts]:
                    if text_to_parse and any(char.isdigit() for char in text_to_parse):
                        date_text = text_to_parse
                        break

            if not date_text:
                fallback_tags = item.select(".b_meta .b_fact, .b_fact, .news_dt")
                for date_tag in fallback_tags:
                    temp_text = date_tag.get_text(strip=True)
                    if temp_text and any(char.isdigit() for char in temp_text):
                        date_text = temp_text
                        break

```

```
    if date_text:
        date = self.parse_bing_date(date_text)
```

A.2.3 Results

This subsections showcases snippets of how the results are built.

Construction of the lists

```
# Sort into the correct list
if title and url and date:
    result_data = {
        "title": title,
        "url": url,
        "snippet": snippet,
        "date": date.strftime("%Y-%m-%d")
    }

    if original_lang:
        result_data["title"] = f"{title} (original language source: {original_lang})"
        result_data["original_title"] = original_title_text
        result_data["original_snippet"] = original_snippet_text
        result_data["original_language"] = original_lang

    results_with_date.append(result_data)

elif title or url:
    # Website dates
    web_data = {
        "title": title,
        "url": url,
        "snippet": snippet,
    }
    if original_lang:
        web_data["title"] = f"{title} (original language source: {original_lang})"
        web_data["original_title"] = original_title_text
        web_data["original_snippet"] = original_snippet_text
        web_data["original_language"] = original_lang

    websites.append(web_data)
    oldest_result = min(dated_results, key=lambda x: datetime.strptime(x["date"], "%Y-%m-%d"))
```

Population of lists to avoid duplicates

```
for result in page_dated_results:
    if result['url'] not in seen_urls and len(results_with_dates) < num_results:
        results_with_dates.append(result)
```

```
        seen_urls.add(result['url'])

for result in page_undated_websites:
    if result['url'] not in seen_urls and len(websites_without_dates)
        < num_undated_target:
        websites_without_dates.append(result)
        seen_urls.add(result['url'])
```

Retrieval of oldest result

```
    oldest_result = min(dated_results, key=lambda x: datetime.strptime
        (x["date"], "%Y-%m-%d"))
```

A.3 Frontend Module

A.3.1 Server Actions and API Integration

The following snippets show `analyzeURL`, `analyzeText` and `getUserHistory`.

Listing A.5: Frontend server actions

```

export async function analyzeURL(
  url: string,
  searchDepth: number = 2,
  userId?: string | number,
  jobId?: string | null
): Promise<AnalysisSection[]> {
  try {
    console.log('       Analyzing URL via backend:', url);

    const response = await fetch(`${BACKEND_URL}/search/link`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        input: url,
        search_depth: searchDepth,
        userId: userId ? Number(userId) : null,
        job_id: jobId || null
      }),
    });

    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }

    const data: AnalysisResponse = await response.json();
    console.log('    URL analysis data received:', data);
```

```
30
31      return transformBackendResults(data);
32    } catch (error) {
33      console.error("Error in analyzeURL server action:", error);
34      return [];
35    }
36  }
37
38  export async function analyzeText(
39    text: string,
40    searchDepth: number = 2,
41    userId?: string | number,
42    jobId?: string | null
43  ): Promise<AnalysisSection[]> {
44    try {
45      console.log('       Analyzing text via backend:', text.substring
          (0, 50) + '...');
46      const response = await fetch(`${BACKEND_URL}/search/text`, {
47        method: "POST",
48        headers: {
49          "Content-Type": "application/json",
50        },
51        body: JSON.stringify({
52          input: text,
53          search_depth: searchDepth,
54          userId: userId ? Number(userId) : null,
55          job_id: jobId || null
56        }),
57      });
58
59      if (!response.ok) {
60        throw new Error(`HTTP error! status: ${response.status}`);
61      }
62
63      const data: AnalysisResponse = await response.json();
64      console.log('    Text analysis data received:', data);
65
66      return transformBackendResults(data);
67    } catch (error) {
68      console.error("Error in analyzeText server action:", error);
69      return [];
70    }
71  }
72
73  //USER-History
74  export async function getUserHistory(userId: string) {
75    try {
76      console.log("Fetching user history for:", userId);
77
78      const response = await fetch(`${BACKEND_URL}/history/user/${userId
          }`, {
```

```
79        method: "GET",
80        headers: {
81          "Content-Type": "application/json",
82        },
83        cache: "no-store",
84      });
85
86      if (!response.ok) {
87        throw new Error(`Failed to fetch user history: ${response.status
              }`);
88      }
89
90      const history = await response.json();
91      console.log("    User history received:", history);
92
93      return history;
94    } catch (error) {
95      console.error("Error in getUserHistory server action:", error);
96      return [];
97    }
98  }
99
100 export async function addToUserHistory(
101   userId: string,
102   url: string,
103   cacheId?: number
104 ) {
105   try {
106     console.log("Adding to user history:", { userId, url });
107
108     const response = await fetch(`${BACKEND_URL}/history/user/${userId
          }`, {
109       method: "POST",
110       headers: {
111         "Content-Type": "application/json",
112       },
113       body: JSON.stringify({
114         url,
115         cache_id: cacheId,
116       }),
117     });
118
119     if (!response.ok) {
120       throw new Error(`Failed to add to user history: ${response.
              status}`);
121     }
122
123     const result = await response.json();
124     console.log("    Added to user history:", result);
125
126     return result;
```

```
127   } catch (error) {
128     console.error("Error in addToUserHistory server action:", error);
129     return null;
130   }
131 }
```