ELMO

Team 1 - CS 4485.0W1 Group Project

Shaz Kumar

Avanthi Reddy

Kshitij Kulshrestha

Abel Thomas

Isaac Hasan

Supervisor: Professor Sridhar Alagar

Course Coordinator: Thennannamalai Malligarjunan

Erik Jonsson School of Engineering and Computer Science

University of Texas at Dallas

May 11th 2025

Table of Contents

Chapter 1: Introduction

1.1 Background

In today's information-rich world, news consumption has become increasingly challenging. The average person must navigate through numerous sources, often encountering repetitive information across multiple articles to gain a comprehensive understanding of current events. This fragmentation creates a significant burden, as users must manually filter, cross-reference, and synthesize information from disparate sources. Furthermore, the rise of personalized news algorithms has led to concerns about filter bubbles and echo chambers, where users only see content that aligns with their existing beliefs.

ELMO is an AI-powered platform designed to streamline how users consume news. Using AI, ELMO scans multiple sources, filters out redundancy, and delivers a concise, structured, and customizable news summary based on user preferences. Whether you want quick updates, detailed insights, or something in between, ELMO adapts to your needs.

1.2 Objectives, Scope, and Goals Achieved

Objectives & Scope:

- Develop a web-based platform that aggregates news from multiple trusted sources
- Implement AI-powered content summarization that eliminates redundancy
- Create a personalized user experience with customizable detail levels
- Enable topic tracking and exploration through a user-friendly interface
- Implement secure user authentication and account management
- Design an intuitive content organization system with categories and subcategories
- Establish a question-answering feature for topic exploration

Goals Achieved:

- Developed a secure user authentication system using AWS Cognito
- Designed and built a scalable platform using exclusively AWS services
- Created a Home Page for personalized content delivery
- Implemented a content aggregation system with source filtering capabilities
- Developed customizable detail levels for news delivery, allowing users to choose between summaries or full articles
- Built a bookmark system for tracking topics of interest
- Established a foundation for query-based article generation

- Created a reliable news API for content delivery
- Delivered a responsive, intuitive web application accessible across devices

1.3 Key Highlights

- **Smart News Aggregation:** ELMO combines information from multiple sources without repeating content, providing comprehensive coverage while eliminating redundancy.
- **Customizable Detail Levels:** Users can choose between concise summaries or detailed full articles based on their preferences and time constraints.
- **Source Transparency and Control:** The platform maintains transparency by allowing users to view and manage content sources, ensuring they remain informed about information origins.
- **Onboarding System:** Users can follow specific topics of interest, creating a personalized news experience.
- **Retrieval Augmented Generation (RAG):** ELMO's AI system pulls in the latest news beyond the model's original training data, ensuring content remains current and relevant.

1.4 Significance of the Project

In an era of information overload and concerns about misinformation, ELMO addresses critical needs in how users consume news. ELMO tackles the challenge by bringing together news from a wide range of sources and presenting it in clear, concise summaries. This helps reduce the time and effort needed to make sense of the news, without having the user sacrifice depth or accuracy.

ELMO gives users more control and promotes source transparency and diversity, helping readers break out and engage with broader perspectives.

The project demonstrates how AI can be leveraged to improve information literacy and accessibility in the digital age. By streamlining how news is accessed and understood, ELMO supports more thoughtful and informed consumption, making it easier for users to keep up with the world without feeling overwhelmed.

Chapter 2: High-Level Design

2.1 System Overview & Proposed Solution

ELMO is an AI-powered news aggregation and content synthesis platform designed to provide users with personalized, comprehensive articles on topics of interest. The system collects, processes, and presents news content from multiple sources, offering varying levels of detail based on user preferences.

The application follows a modern web architecture with dedicated frontend and backend components, leveraging cloud services for scalability and reliability. ELMO's content pipeline combines traditional news sources with AI-enhanced generation capabilities to deliver unique, valuable insights to users.

Key features include personalized content recommendations, variable article detail levels, user preference management, and efficient content navigation. The system integrates multiple data sources and processing capabilities to create a seamless news consumption experience.

Key API Endpoints:

NewsAPI

- Endpoint: https://newsapi.org/v2/everything
- Purpose: Fetches news articles by query

Custom Article Generation

- Endpoint: /api/generate
- Purpose: AI content generation with DeepSeek

External API for Curated Articles

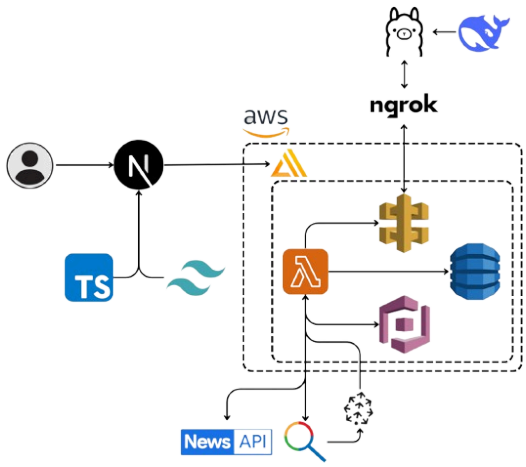
- Endpoint: https://e5e6ofaqlj.execute-api.us-east-1.amazonaws.com/prod
- Purpose: Fetches curated articles

Ollama API Endpoint

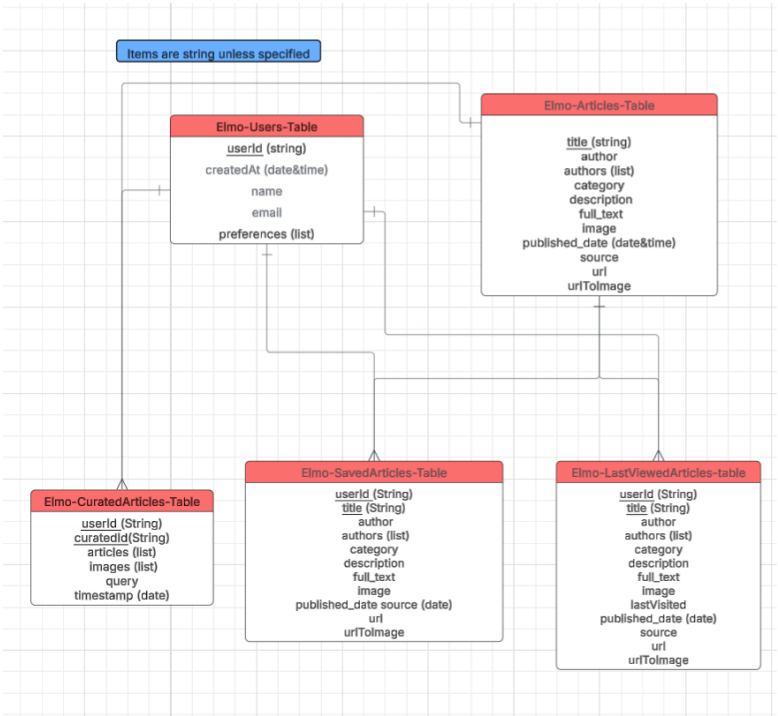
- Endpoint: https://workable-lemur-primary.ngrok-free.app/api/generate
- Purpose: AI-based content generation with DeepSeek R1:14b

2.2 Design and Architecture Diagrams

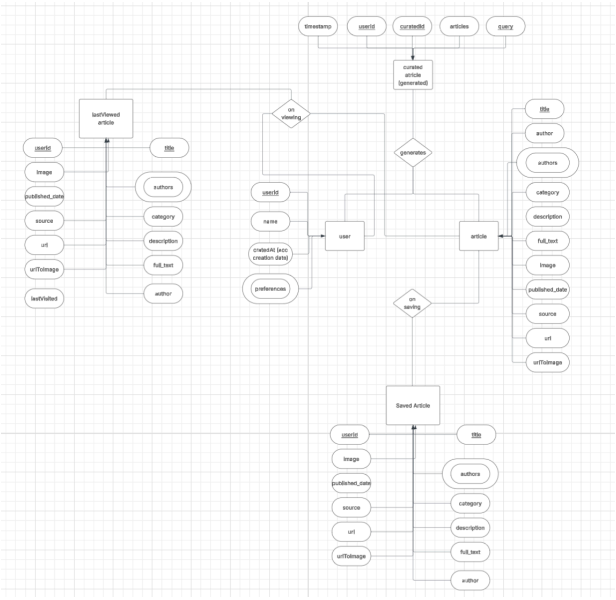
System Architecture:



DB Schema:



Database ER Diagram:



2.3 Overview of Tools Used

Open-Source Technologies:

- **TypeScript:** Static typing for enhanced development productivity
- **Tailwind CSS:** Utility-first CSS framework for responsive design
- **Ollama:** Open-source AI model serving platform for DeepSeek
- **JSDOM & Readability:** HTML content extraction for news articles

Design Tools:

- **Figma:** UI/UX design and prototyping
- **Draw.io:** Architecture and workflow diagrams

UI Component Libraries:

- **shadcn/ui:** Accessible UI components with Tailwind integration
- **Aceternity UI:** Advanced animations and interactive elements
- **Material UI:** Complex data components and form controls

Third-Party Tools:

- **DeepSeek:** LLM for content generation and summarization
- **Ngrok:** Secure tunneling for development environment
- **Pinecone:** Vector database for semantic search
- **Axios:** HTTP client for API requests

Cloud Services (AWS):

- **Amplify:** Frontend-backend integration
- **Lambda:** Serverless computing and database query refinement
- **Cognito:** User authentication
- **DynamoDB:** NoSQL database for user and article data
- **API Gateway:** Secure link between frontend and backend functions
- **EventBridge:** Triggers scheduled/event-based backend actions

External APIs:

- **NewsAPI:** Primary news content source
- **Google Search API:** Supplementary content discovery

Chapter 3: Implementation Details

3.1 Technologies Used

Frontend Stack:

- **Next.js with TypeScript:** Provides a framework for building the user interface with type safety and server-side rendering capabilities.
- **Tailwind CSS:** Enables rapid UI development with utility classes for consistent styling across the application.
- **React Hooks:** Used extensively for state management and component logic.

Backend Stack:

- **AWS:**
    - **Amplify**: Acts as the interface connecting the frontend and backend, simplifying deployment and resource management.
    - **Lambda**: Executes serverless backend logic, including handling API requests, processing article data, and triggering AI services.
    - **Cognito**: Provides secure user authentication, account creation, and session management.
    - **DynamoDB**: Stores structured user data such as preferences, saved articles, curated content, and reading history.
    - **API Gateway**: Exposes Lambda endpoints as secure, authenticated APIs with rate limiting and logging.
    - **EventBridge**: Enables scheduled and event-driven Lambda invocations for tasks like daily article curation or user engagement routines.

- **AI Components:**
    - **DeepSeek:** Serves as the primary language model for text generation and summarization tasks.
    - **Ollama:** Facilitates local hosting of the AI model with efficient inference.
    - **Pinecone:** Enables semantic search functionality through vector embeddings of article content.
    - **RAG Framework:** Custom implementation combining retrieval and generation for contextually relevant content.

Development Tools:

- **Git/GitHub:** Version control and collaboration platform.
- **AWS CloudFormation:** Infrastructure as code for consistent deployment.
- **Ngrok:** Secure tunneling to expose local services during development.

3.2 Code Documentation

The ELMO codebase is structured with modularity and readability in mind, following a page-based layout within the app/ directory. Core features include the home page (app/explore/), which serves as a place for users to browse through a curated set of 200 articles fetched from NewsAPI; the saved page (app/saved/), where users can revisit previously stored articles; and the preferences page (app/preferences/), where users customize the types of content they want to see. Each article is accessible via a unique URL at app/article/[id], which opens a dedicated reading view. This view also offers functionality for users to simplify or expand articles by referencing additional related content for enhanced clarity. User authentication is handled through the app/authentication/ directory, which displays either a login or signup form using reusable components from components/login-form.tsx and components/signup-form.tsx.

Authentication is a prominent feature in ELMO and is implemented using the files in the lib directory: AuthContext.tsx, useAuth.ts, and withAuth.tsx. The useAuth hook is consistently referenced across protected routes to ensure that users are logged in before navigating through the application. This setup integrates with AWS Cognito via amplifyConfig.ts, which also manages all interactions with the AWS backend, including reading and writing user preferences and saved articles to DynamoDB. The routing system is handled through Next.js, using files like layout.tsx and page.tsx to define the global layout and root-level rendering logic.

Backend logic in ELMO is primarily handled through the amplifyConfig.ts file, which acts as the central connector between the frontend and AWS services, enabling seamless CRUD operations on user and article data stored in DynamoDB. Additionally, AI-powered article generation is handled in app/api/generate/route.tsx. This function processes user-submitted queries and calls the Deepseek API to generate original articles using recent and relevant news data. These AI-generated articles are then returned to the frontend for display and can be saved by users. Together, this architecture enables a secure, personalized, informative, and scalable experience.

3.3 Development Process

The ELMO team adopted an Agile development process focused on adaptability and continuous collaboration. Our workflow was organized into weekly sprints, with clearly defined goals, ownership, and regular sync-ups. The entire project used Git for version control and tracked via GitHub Projects using a Kanban board setup.

Planning and Design

- **Requirement Gathering**
    - Early-stage research on content aggregation workflows, user needs, and technical feasibility.
    - Brainstorming sessions to scope features like article generation, user preferences, and saved content.

- **System Design**
    - Created rough wireframes and user flows to guide interface layout and navigation.
    - Outlined backend architecture involving AWS services, 3rd party APIs and open source libraries.

Development Workflow

- **Sprint-Based Iteration**
    - Weekly sprints focused on implementing, refining, and testing key features such as authentication, article fetch, and AI integration.
    - Sprint reviews included testing on local environments and refining based on feedback.

- **Version Control & Task Management**
    - GitHub used for all codebase operations with separate branches for each collaborator.
    - Pull requests were submitted and reviewed before merging into main to ensure code quality and prevent any merge conflicts.
    - GitHub Projects' Kanban board helped track progress across multiple To-do List stages such as "General" "Frontend" and "Backend" as well "In Progress", "Scrapped", and "Done" stages.

- **Testing & Debugging**
    - Local testing was conducted regularly for authentication, API integration, and UI behavior.
    - Manual testing covered edge cases like broken logins, empty states, and article loading failures.
    - Console logging and incremental updates were used for debugging.

Collaboration Tools

- **GitHub** – Source control, issue tracking, and Kanban-style project board.
- **Discord** – Daily communication and debugging discussions.
- **Figma** – UI mockups and design references for visual consistency.

Chapter 4: Performance & Validation

4.1 Testing and Validation

Testing for ELMO focused primarily on manual validation, API testing, and local environment checks to ensure reliability, accurate data handling, and seamless user interaction.

Local Testing & Debugging

- Most testing was conducted directly on local development environments during implementation.
- Components and flows such as authentication, article generation, and data saving, were manually verified through trial runs and iterative debugging.
- Console logs and browser developer tools were frequently used to identify and resolve rendering issues or broken logic.

API and Backend Validation

- Postman was used extensively to test API and backend endpoints, particularly the AI article generation route and data fetch operations.
- AWS's built-in test tools (e.g., Lambda test events and DynamoDB data inspection) helped validate serverless function performance and database interactions.
- Backend responses were tested for both correctness and speed, ensuring that article fetching and AI content was accurate and delivered without excessive delay.

User-Centric Validation

- Informal usability testing was done by team members to ensure intuitive navigation and readable interfaces.
- Functionality like saving preferences, accessing personalized articles, and navigating through pages was tested through real-world use cases.

4.2 Metrics Collected and Analysis

Backend Lambda Function Metrics

- AI Article Generation Lambda (Deepseek + Google Search API):
    - Average Duration: ~14.3 seconds
    - Memory Utilization: 82–83 MB (out of 128 MB)

- Article Population Lambda (daily bulk insert of ~90 articles):
    - Average Duration: ~340 seconds → ~5.6 minutes
    - Memory Utilization: 124 MB (out of 128 MB)

Page Performance Metrics:

- User Auth Page (Login/signup via Cognito):

    - Avg Load Time: ~200 ms
    - Payload Size (Request/Response): ~50–200 KB

- Home Page (Loads ~200 articles daily):
    - Avg Load Time: ~200 ms
    - Payload Size (Request/Response): ~1 KB (request), ~2-3 KB (response)

- Article View (Opens full article details):
    - Avg Load Time: ~60ms
    - Payload Size (Request/Response): ~50-150 KB (article details)

- Preferences Page (Shows preferences to the user):
    - Avg Load Time: ~100 ms
    - Payload Size (Request/Response): ~10–50 KB (user preferences)

App Feature Performance Metrics:

- Article Summarization (Calls AI to shorten content):
    - Avg Load Time: ~3-4 seconds
    - Payload Size (Request/Response): ~10–50 KB (summarized content)

- Article Expansion (Adds context from other sources):
    - Avg Load Time: ~3-4 seconds
    - Payload Size (Request/Response): ~50–150 KB (expanded article content)

- AI Article Generation (Uses Deepseek API + Google Search API):
    - Avg Load Time: ~3-4 seconds
    - Payload Size (Request/Response): ~50–200 KB (articles fetched)

Chapter 5: Lessons Learned

5.1 Insights Gained

Throughout the development of ELMO, our team gained valuable insights into working with AI, external APIs, cloud infrastructure, and user-centered design. Each aspect of the project offered its own set of lessons that shaped the final outcome.

- **LLM Integration and Enhancement**
    - One of the biggest takeaways came from building our AI article generation pipeline using the Deepseek language model. Rather than relying solely on the model's built-in capabilities, we improved its responses by incorporating articles fetched in real time through the Google search API. This allowed us to generate more relevant content efficiently. It also helped us better understand how to guide AI models using outside data and how important context is when trying to produce accurate and meaningful results.

- **Researching and Working with APIs**
    - Integrating third-party APIs played a major role in how we built our system. From testing different news APIs to figuring out how to filter and clean up results, we learned how to evaluate tools quickly and adjust based on what worked best. Handling things like inconsistent response formats, rate limits, and data quality pushed us to be more adaptable. It also taught us how much behind-the-scenes work is required just to get clean, usable information for our app.

- **Backend Setup with AWS**
    - Working with AWS services gave us hands-on experience with deploying a semi-serverless backend. Setting up services like Amplify, Lambda, Cognito, and DynamoDB taught us how these pieces connect to form a scalable and secure architecture. While there was a learning curve in managing configurations and debugging across services, it gave us a much clearer understanding of how cloud-based applications are structured in real-world scenarios.

- **User Interface and Design Thinking**
    - Even though we didn't conduct user surveys or live testing, we consistently approached the project with a user-first mindset. We asked ourselves how we would want to use the app and tried to make navigation as simple and intuitive as possible. This led to small but meaningful design choices, such as allowing users to toggle between simplified and expanded article views, or easily managing their topic preferences. Thinking about design in this way reminded us that the best interfaces often come from anticipating user needs rather than overloading with features.

5.2 Lessons from Challenges

Building ELMO came with its fair share of unexpected challenges. From technical hurdles to vague documentation, each issue required some form of workaround to move forward. These obstacles helped us become more resourceful and taught us to better manage complexity across different parts of our tech stack.

- **Navigating AWS Authentication and Services**
    - One of the earliest challenges we faced was setting up user authentication with AWS Cognito. The documentation was often inconsistent or out-of-date due to changes in the Amplify ecosystem, which made it confusing to figure out the correct configuration and steps. Troubleshooting required a lot of trial and error, as well as digging through forum posts, GitHub issues, and scattered AWS resources. We eventually got things working, but it taught us how important it is to be patient when working with evolving platforms.

- **AI Hallucination and Prompt Engineering**
    - Another significant challenge came from using Deepseek for content generation. Despite setting clear instructions in our prompts, the model sometimes hallucinated facts or even responded in Chinese instead of English. This was especially difficult because we wanted our AI-generated content to appear trustworthy. We spent time refining our prompts, testing different phrasing, and experimenting with inputs to reduce hallucinations and ensure more consistent outputs. This process taught us how sensitive LLMs can be to prompt structure and how to think critically about model behavior.

- **Local Hosting for AI Accessibility**
    - Running the AI model locally introduced some additional challenges. Because the model required a decent amount of memory and processing power, not everyone on the team could run it efficiently. To solve this, one team member with a more powerful machine hosted the model and made it accessible via a shared local server. This allowed everyone to interact with it during development, but also highlighted the limitations of relying on local resources in a collaborative project.

- **Limitations with News APIs**
    - When it came to fetching article data, NewsAPI had several restrictions. First, it didn’t provide real-time content, with most articles being at least 24 hours old. To work around this, we combined recent articles with slightly older ones to fill in the gaps. Additionally, NewsAPI had a limited set of usable categories, which forced us to reduce the number of preference options available to users. We adjusted our system to only include the most reliable and consistently populated categories.

- **Enhancing Article Data with Web Scraping**
    - Another issue was the lack of detailed content in many articles returned by NewsAPI. Often, only the title, URL, and short description were provided. To solve this, we used an open-source web scraping library to extract full article text from the source URLs. This allowed us to enrich the content and give users a more complete reading experience. It also required us to merge two data sources, which were the scraped content and metadata from the API, into one clean dataset for each article.

5.3 Skills Developed and Improved

Throughout the course of the ELMO project, we developed and strengthened a wide range of technical and teamwork skills. On the technical side, we gained hands-on experience with building user interfaces that balance functionality and simplicity. Working with AWS services like Amplify, Lambda, Cognito, and DynamoDB helped us better understand cloud infrastructure and the workflow required to deploy secure, scalable applications. We also became much more comfortable researching and integrating third-party APIs, troubleshooting incomplete documentation, and navigating inconsistent or unfamiliar data sources. Additionally, working with an AI model like Deepseek challenged us to think critically about prompt design, data input structure, and how to improve generative outputs through layering and refinement.

From a collaboration standpoint, we improved our ability to plan and execute using Agile methodologies. Weekly sprints helped us break down the work into manageable goals and allowed us to track progress in a more organized way. We also sharpened our use of Git and GitHub for version control, issue tracking, and code reviews, which became essential for maintaining a stable and collaborative development environment. These skills, both technical and interpersonal, were developed not just by completing tasks but by learning how to adapt, troubleshoot, and work effectively as a team.

Chapter 6: Future Work

6.1 Proposed Enhancements

Our future plans focus on enhancing content quality and accessibility through multi-language support, bias detection, multimedia summaries, and fact-checking capabilities. The user experience will be expanded with mobile apps, social features, and audio article options, while technical improvements will boost speed, personalization, and real-time updates. Advanced analytics will provide insights into user habits, topic trends, and content diversity.

Content & AI

- Multi-language support for global news coverage
- Opinion/bias detection in articles
- Multimedia integration (images, videos) in summaries
- Fact-checking for controversial topics

User Experience

- Mobile apps (iOS/Android)
- Social sharing & collaborative reading lists
- Audio articles (Text-to-Speech)

Technical Optimizations

- Edge caching for faster global delivery
- Personalized recommendations based on reading habits
- Real-time breaking news alerts
- AI model optimization for lower latency and cost

Data Analytics

- User reading habit dashboards
- Trend analysis across topics and sources
- Insights on information diversity

6.2 Development Recommendations

Our recommendations are centered around building a scalable and efficient architecture through services within AWS and enhanced monitoring. The AI strategy includes

fine-tuning DeepSeek models using user feedback, integrating multi-modal capabilities, and improving content quality metrics. To support continuous improvement, we suggest conducting in-depth user research and implementing process enhancements like feature flags, automated testing, and thorough AI documentation.

Architecture

- Microservices transition as complexity grows
- Containerization (AWS Fargate) for AI workloads
- Enhanced monitoring and alerting

AI Strategy

- Fine-tune DeepSeek models with user feedback
- Multi-modal AI (images/videos)
- Better content quality metrics

User Research

- Long-term studies on ELMO's impact
- Bias and trust perception analysis
- User segmentation studies

Process Improvements

- Feature flags for controlled rollouts
- Automated performance testing
- AI component documentation

Chapter 7: Conclusion

7.1 Summary of Key Accomplishments

ELMO has successfully delivered an innovative AI-powered news platform that addresses key challenges in modern information consumption:

1. **Efficient Information Access:** ELMO provides users with concise and synthesized news content that eliminates redundancy while preserving comprehensive coverage.
2. **Personalized Experience:** Through customizable detail levels, topic following, and preference management, ELMO adapts to a user's individual needs and interests.
3. **Source Transparency:** The platform maintains transparency about information sources, allowing users to make informed decisions about content reliability.
4. **Technical Innovation:** The integration of AI, architecture and vector search demonstrates effective application of cutting-edge technologies to solve real-world problems faced in media.
5. **User-Centered Design:** Throughout development, ELMO maintained a focus on user needs, resulting in a modern and intuitive interface with intentional features.

7.2 Acknowledgements

The ELMO team would like to express sincere gratitude to:

- Professor Sridhar Alagar for supervision and guidance throughout the project.
- Course Coordinator Thennannamalai Malligarjunan for administrative support and guidance throughout the project.
- The Erik Jonsson School of Engineering and Computer Science at the University of Texas at Dallas provided the educational foundation and resources necessary for this project.
- Early users who provided valuable feedback during the testing phase.
- Open-source communities behind the tools and libraries that made this project possible.

Their support and contributions were instrumental in bringing ELMO from concept to reality.

References

- AWS Documentation. (2024). AWS Amplify Developer Guide. https://docs.aws.amazon.com/amplify/
- AWS Documentation. (2024). Amazon API Gateway Developer Guide. https://docs.aws.amazon.com/apigateway/
- AWS Documentation. (2024). Amazon Cognito Developer Guide. https://docs.aws.amazon.com/cognito/
- AWS Documentation. (2024). Amazon DynamoDB Developer Guide. https://docs.aws.amazon.com/dynamodb/
- AWS Documentation. (2024). Amazon EventBridge User Guide. https://docs.aws.amazon.com/eventbridge/
- AWS Documentation. (2024). AWS Lambda Developer Guide. https://docs.aws.amazon.com/lambda/
- DeepSeek AI. (2024). DeepSeek Language Model Documentation. https://api-docs.deepseek.com/
- Google. (2024). Google Programmable Search Engine API Documentation. https://developers.google.com/custom-search/v1/overview
- Hugging Face. (2024). Hugging Face Documentation. https://huggingface.co/docs
- Lewis, P., et al. (2023). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Proceedings of Neural Information Processing Systems.
- NewsAPI. (2024). NewsAPI Documentation. https://newsapi.org/docs
- Ngrok. (2024). Ngrok Documentation. https://ngrok.com/docs
- Nielsen, J. (2023). User Experience Factors in News Consumption Interfaces. Journal of Human-Computer Interaction, 39(2), 145–162.
- Ollama. (2024). Ollama Documentation. https://ollama.com/library
- Pinecone. (2024). Vector Database Documentation. https://www.pinecone.io/docs/
- Tailwind Labs. (2024). Tailwind CSS Documentation. https://tailwindcss.com/docs
- Unsloth. (2024). Unsloth Documentation. https://unsloth.ai/blog/deepseek-r1
- Vercel. (2024). Next.js Documentation. https://nextjs.org/docs

9. Appendices

Appendix A: Team Member Contributions

- **Isaac Hasan (Backend Developer):**
  - Created and leveraged lambda functions to utilize APIs for project goals
  - Led the onboarding of the AI model and its integration with frontend and backend components
- **Abel Thomas (Backend Developer):**
  - Set up most backend operations and APIs within AWS to handle authentication, user data storage, and other key-driven functions. Integrated the backend with the frontend components
- **Avanthi Reddy (Frontend Developer):**
  - Developed the user flow and design of ELMO to ensure an intuitive, modern, and aesthetically pleasing experience
  - Implemented key frontend pages and components
- **Kshitij Kulshrestha (Frontend Developer):**
  - Established the design system for consistent branding and implemented real-time design updates
  - Refined article rendering and sidebar, developed the landing page, and designed a streamlined user flow
- **Shaz Kumar (AI Engineer):**
  - Created ELMO AI by integrating a fine-tuned DeepSeek model for text generation, implementing the Ollama API for model deployment, and developing a custom RAG Framework to ensure contextually relevant content generation

Appendix B: User Interface Mockups

[Initial Designs]