COLORADOSCHOOLOFMINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

CSM Bridgman 2:

Bridg Trolls

Isaac Fry
Katrina Ngo
Jaden Nguyen
Maddi Tajchman

Revised Dec. 5, 2024

CSCI 370 Fall 2024

Prof. Kathleen Kelly

Table 1: Revision history

Revision	Date	Comments
New	Aug. 21, 2024	Completed Sections: I. Introduction II. Functional Requirements III. Non-functional Requirements IV. Risks V. Definition of Done XI. Team Profile References Appendix A – Key Terms
Rev – 2	Sept. 15, 2024	Updated Sections: I. Introduction II. Functional Requirements III. Non-functional Requirements Completed Sections: VI. System Architecture
Rev – 3	Oct. 20, 2024	Updated Sections: I. Functional Requirements Completed Sections: II. Software Test and Quality III. Project Ethical Considerations
Rev – 4	Nov. 9, 2024	Updated Sections: I. Functional Requirements Completed Sections: I. Project Completion Status II. Future Work III. Lessons Learned IV. Acknowledgements V. Appendix

Table of Contents

I. Introduction

This report outlines the development of a cutting-edge system designed to transform the Colorado School of Mines course catalog into an intelligent and interactive flowchart. By leveraging automated web scraping and advanced data preprocessing, the project aims to provide a streamlined, user-friendly visualization of course structures and prerequisites. This tool serves as a valuable resource for students and advisors, simplifying academic planning and enhancing the overall educational experience.
High level scope: Create a highly intelligent and interactive visual layout of the course catalog represented as a flowchart via the automated scraping and preprocessing of the Mines course catalogs.

Client: Terry Bridgman is our official client. However, we see Professor Bridgman as advocating for the entire Mines community, specifically the academic departments.

Currently, there is not a standardized process for departments to automatically create flowcharts. If a department desires a flowchart, that department must create the flowchart ad-hoc without any consistency across departments. There are also departments, like Chemical Engineering, that do not create flowcharts for their students. In order to standardize the communication of department curriculum across the university while minimizing the possibility of

human error, we seek to automate the creation of these flowcharts directly from the Mines course catalog, which is the source of truth.

Previous Software Revisions:

There is no existing software platform. However, Professor Bridgman has prototyped a few versions that emulate the idea. He found these prototypes lacking for a variety of reasons:

- It was not automated
  - Professor Bridgman was copy and pasting the course catalog into ChatGPT to generate a standard
- It was only slightly interactive
  - The incorporation of true post-req and pre-req highlighting was minimal
  - The framework used for network interaction wasn't purposed for the end goal

Source of Data:

We have determined multiple distinct sources of data:

- Mines Course Catalog
  - Accessibility: freely via the web
  - Purpose: accesses the source of truth for the Mines curriculum
- LLM-generated Machine-Readable Flowchart Data
  - Accessibility: freely when pinging an open source LLM
  - Purpose: transforms the course catalog data into a machine-readable format

Stakeholders:

- Departments
  - The administration of individual departments should be able to use our tool to automatically generate and update flowcharts that can be distributed to students
  - These individuals are expected to be non-technical. Thus, this should be a relatively accessible and approachable process.
- Students
  - Students within the university can individually benefit from interacting with their major's flowchart
  - These individuals are expected to be moderately technical, though they would not specifically be tasked with generating the flowcharts individually.

Maintaining the Software:
After the conclusion of field session, Professor Bridgman will assume responsibility of the project, including cloud computing resources, code repositories, and DNS hosting credentials. However, Professor Bridgman has discussed a potential handoff to an administrative department, such as the Registrar or ITS, to control the distribution and maintenance of the tool.

II. Functional Requirements

1. **Automated Flowchart Creation:**
   - Automatically scrape the course catalog to pull information for each department.
   - Generate flowcharts for each major that show pre-requisites & co-requisites.
2. **Interactive Flowchart:**
   - Have an information button on each course to pull up all the relevant information of that course (description, code, pre-requisites, etc).

- Have different modes a course can be for taken, not taken, and currently taking.
- Option to toggle arrows on and off for visualizing pre/co-requisites and post-requisites for each course.
- Implement a user-friendly UI, with front-end enhancements using a JavaScript framework, React JS.
- Have different color schemes and account for color blindness when choosing themes.

3. **Program Usability:**
   - Make the flowcharts distributable to all departments/offices on campus.
4. **Backend Pipeline:**
   - Build a backend pipeline to process catalog data into a machine-readable format (JSON).
   - Ensure that the backend pipeline is separate from flowchart generation to maintain extensibility for future iterations of the project.

Stretch Goals:

5. **Historical Grade Data Integration:**
   - Generate a heat map for courses based on historical grade data to indicate course difficulty.
   - Use this data to suggest optimal semesters for students to take certain classes.
6. **Extended Functionality:**
   - Integrate previous catalog data to allow users to generate flowcharts for past academic years.
   - Explore using additional tools like AWS for scaling and managing large data sets or LLM processing. Toggle for aligning the classes by recommended semester to take it, vs just laying it out in the given course catalog format.
7. **Additional Student Customization**
   - Implement more options so students can fully customize their school flowchart.
   - Possible additions include:
     - Allow students to interact with the flowchart by marking courses they've completed and seeing adjusted pre-requisite chains.
     - Include functionality for students to add minors or master's classes for a fully customized view.
     - Allowing students to select courses to fill elective squares, to visualize the pre-requisites in the flowchart.

III. Non-Functional Requirements

1. **Non-Functional Requirements:**
   - Ensure the system is reliable and scalable.
   - Maintains consistent performance across different departments.
   - Color Mines core, major specific, and focus area courses differently, similar to current department flowcharts.

Stretch Goals:

2. **Additional Visual Enhancements:**
   - Implement a toggle or customization for users to save flowcharts to modify them later.
   - Include a toggle for aligning the classes by recommended semester versus laying them out strictly by pre-requisite format.

- Add drag-and-drop functionality for students to create custom schedules.

IV. Risks

Every project carries inherent risks that can impact timelines, functionality, and overall success, and the Catalog-to-Flowchart system is no exception. Identifying potential challenges early enables the team to develop effective mitigation strategies, reducing the likelihood and severity of adverse outcomes.

In this section, we discuss key risks associated with our project, including time-intensive learning curves, challenges in scraping the course catalog, formatting limitations of visualization libraries, and potential intractability of the flowchart. For each risk, we evaluate its likelihood, potential impact, and propose targeted mitigation strategies to address these challenges and ensure the project remains on track.

1. Time used for learning the chosen language(s) might take too much time.
    - Likelihood: Likely
    - Impact: Major
    - Risk Mitigation: Dedicate time to structured learning using specific online resources such as Codecademy, freeCodeCamp, or Udemy crash courses on the chosen programming language(s). Allocate two weeks for intensive learning sessions to ensure a basic understanding before implementation.

2. We can't scrape the course catalog.
    - Likelihood: Unlikely
    - Impact: Major
    - Risk Mitigation: Consult with our client and domain experts if challenges arise. Additionally, prepare a fallback plan that involves using publicly available datasets or manual data entry for initial testing if scraping becomes unfeasible.

3. Impossible to strictly control format using the given libraries.
    - Likelihood: Unlikely
    - Impact: Moderate
    - Risk Mitigation: Research and prototype other visualization libraries such as D3.js or Cytoscape.js that might provide greater flexibility. Include a comparative analysis of options and allocate time for testing alternate libraries to ensure a smoother transition if needed.

4. Intractability of the flowchart
    - Likelihood: Unlikely
    - Impact: Moderate/Major
    - Risk Mitigation: Maintain a contingency plan to offer a static flowchart version in scenarios where interactivity is too resource-intensive or complex. This fallback ensures the core functionality of the tool is not compromised while allowing room for future enhancements.

V. Definition of Done

Defining "done" is essential for ensuring that our project meets the agreed-upon standards of functionality, usability, and quality. This section outlines the criteria that signify the completion of the Catalog-to-Flowchart system, including the delivery of a minimally viable feature set, automation of key processes, interactivity, and accuracy.

Our project's minimal useful feature set includes the development of an automated, interactive, and accurate system. Automation ensures that the tool can retrieve and process data without manual intervention, interactivity enhances user engagement by allowing real-time updates and dynamic adjustments to the flowchart, and accuracy guarantees that the visualized data aligns with the current course catalog and curriculum information.

To ensure the software meets client expectations, we will conduct rigorous testing, focusing on scenarios such as handling inaccessible catalogs gracefully. For example, the system should fail in a controlled manner, alerting users to the issue without crashing. Additionally, the tool will support the generation of a complete flowchart via a single command, simplifying the user experience and reducing potential errors. Successful testing will be marked when the client, Terry Bridgman, expresses satisfaction—described humorously as "leaping from joy."

The product will be delivered by November 26, before Thanksgiving Break, through the GitHub repository shared with the client. This ensures accessibility and transparency while allowing the client to review and provide feedback as necessary.

VI. System Architecture & Technical Design

System Architecture Overview:

The Catalog-to-Flowchart system architecture, illustrated in Figure 1, is designed to seamlessly integrate data collection and visualization processes. It is divided into two primary segments—**Information Collection** and **Visualization**—comprising three key components: the Web Scraper, LLM, and React program. These components work together to transform raw data from the Colorado School of Mines Catalog into an interactive, user-friendly flowchart tool, ensuring both accurate representation and an engaging user experience.

There are three main channels of communication between components of the system. The first channel concerns communication that uses direct Web URL access of webpages to either pull information or push our final product to the end user. The second channel concerns all external connections to the data storage container of choice. The current iteration makes use of an AWS Simple Storage Service (S3), which will be accessed using a Boto3 API. The last channel concerns internal movement within AWS, which we use for both storage and processing.

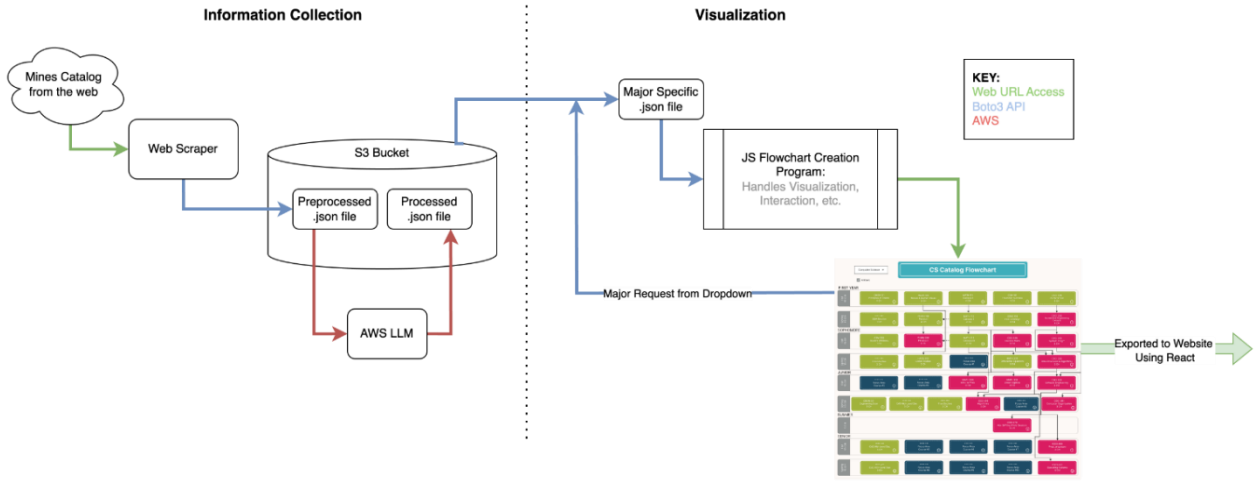

Figure 1: Architecture design of the Catalog-to-Flowchart system

Catalog Web Scraper:

Figure 2 illustrates the foundational assumptions and processes behind the Catalog-to-Flowchart system's data collection. Our project relies on the Colorado School of Mines administration, likely the Registrar, maintaining an updated catalog accessible via the internet. Using web scraping techniques, catalog data is retrieved from the active URL, despite challenges posed by inconsistent formatting across departments and within individual webpages.

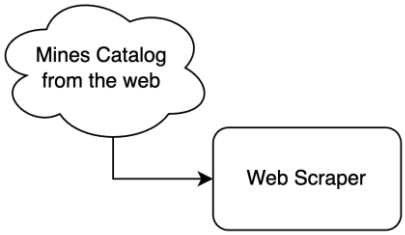

Figure 2: Mines catalog to Scraper Connection.

Two key types of information are critical to this process: the **course catalog**, which includes course descriptions and prerequisites, and the **curriculum**, which outlines required courses and their suggested progression within a student's academic journey. These datasets are scraped separately, reflecting the distinct structure of the information and the need for specialized storage.

Post-scraping, the data is stored as preprocessed .json files, enabling further refinement and consumption by an LLM. While not directly usable for flowchart visualization due to its unstructured nature, this preprocessing stage makes the data manageable and ready for transformation.

The web scraper operates as a serverless function in AWS, scheduled to run as a cron job for regular updates. This ensures the flowchart remains accurate and current without requiring manual intervention from administrative staff. Despite the inherent limitations of web scraping—such as potential errors and reliance on institutional updates—it remains the most reliable and accessible method due to restricted access to clean data sources maintained by ITS and the Registrar.

Large Language Model (LLM) - AWS Bedrock:

After exiting the webscraper, the preprocessed data is stored in an AWS Simple Storage Solution (S3) bucket. Storing data in S3 ensures that the data is accessible by all parts of the model architecture (as opposed to locally storing the data, which may or may not be accessible or distributed).

After the Webscraper runs, it will automatically check to see if the hash of the resulting .json already exists within the S3 bucket. If not, it will trigger the LLM to parse the new version of the .json file. However, if the hash already exists, then we can assume that an LLM has already processed that data, so there will be no need to re-process the same data. This reduces computation overhead and costs.

AWS Bedrock enables streamlined access to foundational LLMs via a simple interface. The LLM can consume the preprocessed data and parse the necessary information into a .json that is consumable by the visualization process. For instance, the LLM can cleanly retrieve the course codes of the pre-requisites, the co-requisites, the credit hours, and the title. While this process is not necessarily deterministic, it is near-deterministic, and the LLM can consistently process the data.

After consumption by the LLM, the data is now considered as “processed” and consumable by the visualization process. The .json output from the LLM is stored in the S3 bucket, as seen in Figure 3, which is accessible by the visualizer. By this point in the system architecture, the messy and text-based data from the publicly accessible webpage has been transformed into a machine-readable format.

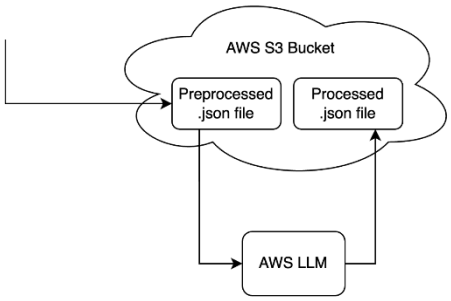

Figure 3: AWS S3 Bucket and LLM Processing connection.

Visualization Program:

The visualization component of the Catalog-to-Flowchart system is a React.js program designed to dynamically render processed data from .json files, offering an interactive and user-friendly experience. This program includes features such as individual course components that display additional details on click, a dropdown menu for selecting majors, and toggleable arrows to visualize prerequisites. These interactive elements enhance usability and ensure proper representation of the processed data.

To support dynamic updates, AWS Boto3 API calls retrieve the appropriate .json files from S3 based on the selected major in the dropdown menu. This enables the tool to update displayed courses and their associated information in real time, as shown in Figure 4.

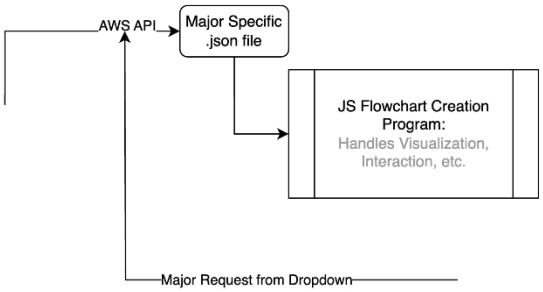

Figure 4: Connection between user interactive dropdown with JavaScript React program.

Final Flowchart Design:

The final flowchart design in Figure 5 combines the features of existing catalog flowcharts across all Mines departments. It displays courses organized by year and semester, taken from the course catalog’s recommended schedule. It also contains a color-coded system to highlight Mines core, major core, and focus area courses. Black arrows are in place to show pre-requisites/co-requisites and can be turned off by the user with a toggle. Additionally, every course has a “more information” button for the full course description as processed from the catalog. All features serve as a centralized place for students to look at curriculum plans and in-depth course information.

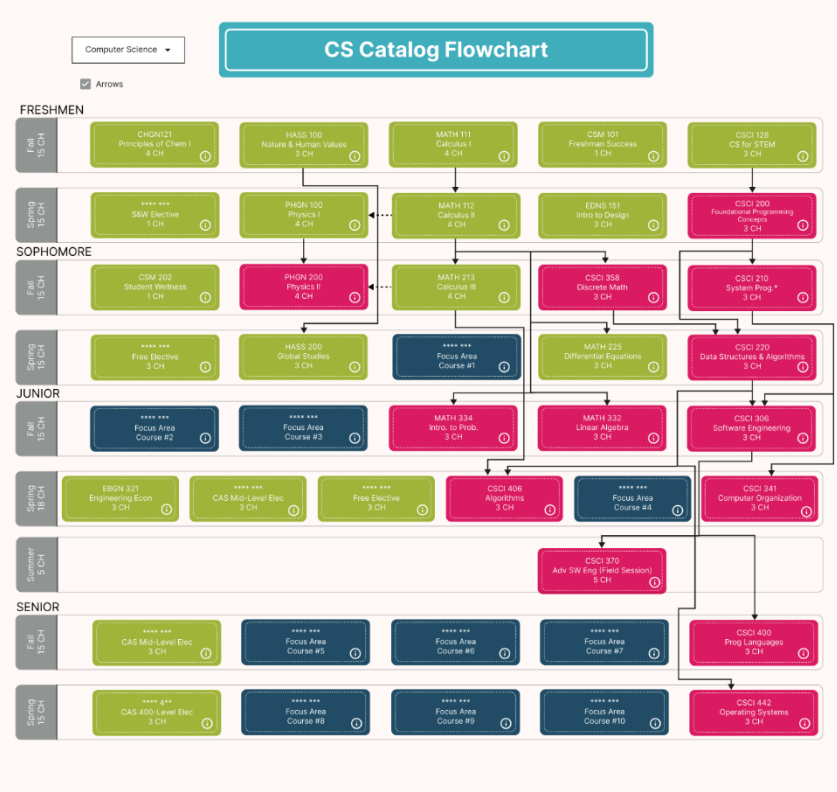

Figure 5: Final Flowchart Design

VII. Software Test and Quality

Functional Requirements: Automated Flowchart Creation, Backend Pipeline

The following unit tests ensure the functionality of the backend scraping and preprocessing objects and methods. This includes, but is not limited to, the webscraping functionality, the S3 interaction and functionality, and the Bedrock LLM determinism.

These tests only focus on functionality rather than interactions. The structure that runs the backend pipeline will be managed differently.

Test Name	Category	Environment	Setup	Action	Expected Result
test_pytest	Unit Testing	Pytest Suite	Catalog metadata, AWS .env	Verifies if pytest is set up and running correctly	The test passes if assert True executes without errors
test_catalog_single_simple	Unit Testing	Pytest Suite	Catalog metadata, AWS .env	Tests the LLM's ability to parse a simple course catalog entry without boolean logic	Each field should match predefined values, and costs calculated for the text output and input should approximately match expected values

test_catalog_single_prompted	Unit Testing	Pytest Suite	Catalog metadata, AWS .env	Tests the LLM's handling of a course catalog entry with unclear boolean logic included in the prompt	Each field should match predefined values, specifically with boolean logic accurately represented, and the LLM's costs calculated for the output and input should align closely with expected value
test_catalog_single_complex	Unit Testing	Pytest Suite	Catalog metadata, AWS .env	Tests the LLM's ability to parse a course catalog entry with unclear boolean logic not included in the prompt	Each field should match predefined values, and costs for text output and input should align closely with the expected values
test_catalog_single_complex_again	Unit Testing	Pytest Suite	Catalog metadata, AWS .env	Repeats the test with another complex course catalog entry to ensure consistency when unclear boolean logic is not in the prompt	Each field should match predefined values, and costs for text output and input should align closely with the expected values
test_catalog_five	Unit Testing	Pytest Suite	Catalog metadata, AWS .env	Tests the LLM's ability to handle and parse multiple (five) classes with varying levels of complexity	Each field should match the predefined values for all classes, and costs for the output and input text should align closely with the expected values
test_multiple_tracks	Unit Testing	Pytest Suite	Catalog metadata, AWS .env	Tests the LLM's ability to extract and correctly identify multiple tracks within the Computer Science (CS) department's curriculum	The response contains 7 tracks, each starting with 'ComputerScience' and including an underscore. Tracks must match predefined categories.
test_multiple_majors	Unit Testing	Pytest Suite	Catalog metadata, AWS .env	Tests the LLM's capability to extract and differentiate between multiple majors (Chemistry and Biochemistry) and their associated tracks within the Chemistry department's curriculum	The response contains 4 entries, with each major track starting with either 'Chemistry' or 'Biochemistry' and separated by an underscore. Tracks should match categories such as 'General', 'Environmental', and 'Biochemistry'
test_multiple_plangrids	Unit Testing	Pytest Suite	Catalog metadata, AWS .env	Tests the LLM's ability to generate multiple curriculum plan grids for different tracks within the Applied	The response contains 3 tracks. The test passes if all 3 grids conform to expected standards

				Mathematics and Statistics (AMS) department	
test_emplace_simple	Unit Testing	Pytest Suite	Catalog metadata, AWS .env	Tests the ability to store and retrieve a simple text file in an S3 bucket	The file is successfully stored and the file's content matches the original string when retrieved.
test_json_formatting	Unit Testing	Pytest Suite	Catalog metadata, AWS .env	Tests the storage and retrieval of a JSON object in an S3 bucket, ensuring correct formatting and content	The JSON object is correctly stored as a string in the S3 bucket and is accurately retrieved. The response should be evaluated as a dictionary when parsed, confirming proper JSON formatting
test_webscraper_init	Unit Testing	Pytest Suite	Catalog metadata, AWS .env	Tests the initialization of the object with catalog metadata	The catalog metadata contains 'CSCI' as a key, confirming successful initialization
test_url_valid	Unit Testing	Pytest Suite	Catalog metadata	Pings a valid URL for the Computer Science (CSCI) department to check connectivity	The HTTP response status code should be 200, indicating the URL is accessible
test_class_code_invalid	Unit Testing	Pytest Suite	Catalog metadata	Tests the behavior of the web scraper when an invalid department code is provided	InvalidDeptCodeError is raised with the message: "NOT_A_DEPT dept. name not found in catalog metadata."
test_class_codes	Unit Testing	Pytest Suite	Catalog metadata	Retrieves class codes for the CSCI department to verify the correctness of the information retrieved	The response includes 'CSCI128' as a key, and the entry for 'CSCI101' contains the correct title and semester hours
test_file_saving	Unit Testing	Pytest Suite	Catalog metadata	Tests the ability of the web scraper to save the retrieved department information as a file	A JSON file for the specified department ('AMS') is successfully created in the predefined directory
test_all_catalog_metadata	Unit Testing	Pytest Suite	Catalog metadata	Verifies metadata .yaml file is correctly formatted and all URLs for departments are accessible.	The number of departments in the catalog metadata matches the number retrieved from the website. Each URL for the departments is accessible, with a status code of 200
test_all_course_catalogs	Unit Testing	Pytest Suite	Catalog metadata	Tests the retrieval and saving of all course catalogs for each department listed in the catalog metadata	A JSON file for each department is created in the directory. Departments that issue courses should have corresponding files.
test_major_requirements_multiple	Unit Testing	Pytest Suite	Catalog metadata	Verifies that the major requirement grids are	All plangrids conform with expected plangrid structure

				correctly formatted for all degree-granting departments	
test_pipeline_init	Unit Testing	Pytest Suite	Catalog metadata, AWS .env	Tests Pipeline class with the specified configuration file and root folder. Checks interactions between S3, Bedrock, and the web scraper components	The Pipeline is consistent with the configuration file

Functional Requirements: Backend Pipeline

The following test plan ensures that the pipeline executes the backend components in appropriate and consistent ways.

Edge cases: failures in the pipeline (at either the Build stage or the Lambda stage), failures in regular execution timing

Threshold for Acceptability: backend is distinct from the frontend and runs automatically

Lambda Connection	Build, Integration, Deployment	AWS CodePipeline	Git repos credentials, AWS credentials to CodePipeline and Lambda	Ensure that the Lambda function can read from the appropriate CodeBuild S3 bucket and update itself with the code.	An update to the production branch (main) of the backend pipeline kicks off the CodePipeline, dumps the build in an S3 bucket, and the Lambda handler updates the Lambda source code.
Pipeline Failures	Build, Integration, Deployment	AWS CodePipeline	Git repos credentials, AWS credentials to CodePipeline and Lambda	Force the pipeline to fail, whether in build or deployment, and ensure that it doesn't affect any production code	A failure in the CodePipeline doesn't change the Lambda code.
Regular Execution Timing	Build, Integration, Deployment	AWS CodePipeline	Git repos credentials, AWS credentials to CodePipeline and Lambda	Create a cron job to periodically run a simple Lambda function to ensure that AWS Lambda can execute on a consistent schedule	A cron job runs periodically to explain sources of failure in the Lambda (i.e. if the simple cron job isn't failing, then the failure in the deployment must be somewhere else besides the Lambda).
Notification of Build Failure	Build, Integration, Deployment	AWS CodePipeline	Git repos credentials, AWS credentials to CodePipeline and Lambda	Force the build to fail and send an update to the AWS account manager with a descrptive failure warning	After the execution the Pipeline Failures test, an email is sent to the account manager (Isaac).

Functional Requirements: Interactive Flowchart and Product Usability

The following test plan ensures the creation, customization, and usability of the final minimum viable product. This includes, but is not limited to instantiation, color selection, base functionality, customization, accessibility, and overall user experience tests.

Edge Cases: Some tests include handling imperfect files pulled from S3, such as non-existent prerequisites.

Test Name	Category	Environment	Setup	Action	Expected Result
Initial Creation Test	Build, Integration, Deployment	AWS Hosted Server	IP and Domain name	WebApp is Launched by a user.	WebApp is launched and instantiated correctly, no errors
Course Status Color Test	Usability Testing	AWS Hosted Server	IP and Domain name	The user cycles through all three course status colors (completed, completing, to be completed), for each class.	All classes cycle in the same way. (To be completed -> Completed -> Completing)
PreReq Highlighting Test	Usability Testing	AWS Hosted Server	IP and Domain name	The user toggles the option to see pre-reqs, selects a class, and then unclicks the class.	Only prereqs of the selected courses get highlighted, other classes stay as they are. Everything returns to normal when a class is unclicked.
PreReq DNE Test	Usability Testing	AWS Hosted Server	IP and Domain name	The user navigates to a major with a known issue in the Mines catalog. They select the classes with a missing prereq	Notice Appears notifying user of a missing prereq
All Major Track Creation Test	Usability Testing	AWS Hosted Server	IP and Domain name	The user cycles through all available majors and checks for issues with any specific class	All majors should be accurately depicted when selecting a new major. All classes needed shown and all prereqs come before postreqs
Colorblind Testing	Usability Testing	AWS Hosted Server	IP and Domain name, Willing Colorblind participant	Testing to make sure our color schemes are understandable for people with colorblindness	Participant can distungish differences between colors on at least 1 palette
Load Testing	Load Testing	AWS Hosted Server	IP and Domain name, Load testing script	1000 instances of our WebApp will be created, performance will be measured by user experience.	Performance will not be altered with multiple users accessing our web app.

Final Usability Test	Usability Testing	AWS Hosted Server	IP and Domain name, Willing participants that haven't interacted with the program at all.	Multiple users will be given access to the full website. They can explore the web app and use it based on their own intuition. Then they will rate the program based on design, usability, convenience, etc., as well as give us any other feedback to incorporate.	The user will rate the program above a 5/10 on every component for a successful test, or average above a 7/10 for a good test. The user will also give tangible feedback that we can implement.

Testing has been carried out throughout the length of the project; it will continue to be iterated on, and our web application will be updated accordingly.

VIII. Project Ethical Considerations

Ethical considerations play a pivotal role in the development of the Catalog-to-Flowchart system. Adhering to the principles outlined by the Association for Computing Machinery (ACM) and the Institute of Electrical and Electronics Engineers (IEEE), this project prioritizes transparency, accessibility, and the responsible use of technology. These guidelines emphasize the importance of creating systems that uphold user trust, maintain data integrity, and promote fairness.

In this section, we explore how the ACM Code of Ethics and IEEE Code of Ethics influence decisions in our project, including ensuring accurate representation of academic information, safeguarding user data, and fostering inclusivity in the tool's design. These principles guide us in delivering a system that not only meets technical objectives but also aligns with broader societal and ethical responsibilities.

Principles Pertinent to the Development of the Product

IEEE 2.01 Provide services in their areas of competence, being transparent about any limitations in their experience and education.

> This principle is essential for ensuring that the flowchart generation system is built by thoroughly understanding the technical requirements and potential constraints. For instance, automating the scraping of course data requires technical expertise in web scraping. Recognizing skill limitations ensures that the product is reliable and works as intended within the scope of the competencies.

IEEE 3.11 Ensure adequate documentation, including significant problems discovered and solutions adopted, for any project on which they work.

> This principle emphasizes the importance of maintaining proper documentation throughout the development process. Given that this project will be handed over to Professor Bridgman, comprehensive documentation will ensure the continuity and maintainability of the system. It will also facilitate the understanding of challenges encountered (such as parsing errors in the course catalog) and how these were addressed in the final solution.

IEEE 7.04 Review the work of others in an objective, candid, and properly documented way.

As the project evolves, code reviews and peer feedback play a critical role in ensuring the quality of the software. This principle promotes a culture of constructive feedback among team members, allowing for improvement of the flowchart automation tool. It also ensures that any design or implementation decisions are properly reviewed and refined to meet the needs of stakeholders.

ACM 1.5 Respect the work required to produce new ideas, inventions, creative works, and computing artifacts.

The project involves automating the process of generating flowcharts based on course data and creating an innovative interface for user interaction. Respecting intellectual property—such as the course catalog data—ensures ethical handling of data and respects the work involved in creating such resources.

ACM 3.5 Create opportunities for members of the organization or group to grow as professionals.

This principle highlights the importance of fostering growth among team members. Opportunities are created for both developers and future users to gain valuable insights and skills. For the development team, it provides hands-on experience in data automation, backend development, and UX design. For users, the tool offers a streamlined process to interact with academic data, improving their understanding of their educational paths.

Principles in Danger of being Violated

IEEE 5.05 Ensure realistic quantitative estimates of cost, scheduling, personnel, quality and outcomes on any project on which they work or propose to work and provide an uncertainty assessment of these estimates.

If realistic estimates for cost, scheduling, and outcomes are not provided, the project could face delays, budget overruns, or fall short of expectations. Misjudging the complexity of automating flowchart generation or the data preprocessing steps could result in the tool being incomplete or unusable, negatively affecting the entire Mines community.

IEEE 5.04 Assign work only after taking into account appropriate contributions of education and experience tempered with a desire to further that education and experience.

Assigning work without considering team members' education and experience could lead to errors in crucial aspects like data scraping or interactive design. This may result in poorly functioning features for the user interface and backend functionality, such as inaccurate prerequisite mapping, ultimately compromising the tool's usability and reliability.

IEEE 7.08 In situations outside of their own areas of competence, call upon the opinions of other professionals who have competence in that area.

Failing to consult experts when outside the team's competence could lead to critical technical mistakes, particularly in areas like data processing, large-scale automation, or user interface design. The project might suffer from incorrect implementations or overlooked technical challenges, reducing the effectiveness and scalability of the system.

ACM 2.9 Design and implement systems that are robust and usably secure.

If the system lacks robust security measures, sensitive data (such as proprietary grade data in a future implementation) could be exposed or manipulated. This would undermine user trust and could result in breaches that harm the institution and violate privacy laws, significantly damaging the tool's credibility and the university's reputation.

Michael Davis Tests

The Michael Davis ethical tests provide a structured framework for evaluating the ethical implications of engineering and computing projects. These tests help assess whether a project aligns with professional and societal ethical standards, such as transparency, fairness, and responsibility. Here, we identify components to be considered when determining whether our product meets these standards:

Stakeholders: The primary stakeholders include Mines students, academic departments, and future maintainers of the tool.

Interests: Students want an accurate and interactive flowchart to guide their academic planning. Departments expect an easy-to-use, reliable tool for automating the generation of standardized flowcharts.

Responsibilities: The team is responsible for ensuring the tool's accuracy, security, and functionality. Professor Bridgman, as a client and advocate, oversees guiding the tool's progress and ensuring it meets Mines' needs.

Alternative Actions and Consequences:

- **Manual flowchart generation**: This option avoids the tool but would continue the current ad-hoc, error-prone process.
- **Outsource development**: Hiring an external developer may result in a more polished tool but could be costly and disconnect the product from the community's specific needs.

Evaluating Alternative Actions through Michael Davis Tests:

- **Harm Test**: The tool aims to reduce errors in academic planning by automating flowchart creation. Developing it in-house avoids the financial burdens of outsourcing and fosters community ownership. If the development team ensures accuracy and security, the harms (like technical errors or data breaches) would likely be minimal compared to the benefits, such as increased efficiency and accuracy in course planning.
- **Common Practice Test:** If every university relied on manual flowchart creation, it would lead to inconsistency, errors, and inefficiency in academic planning. Students could face confusion due to varying formats and outdated information, while departments would struggle to maintain accuracy. Continuing this practice would hinder standardization and improvement, making automated flowchart creation a more effective and beneficial solution.

Ethical Considerations

If the software quality plan for this project is poorly implemented or is insufficient, several ethical concerns arise. Inaccuracies in the flowcharts could lead to students incorrectly planning their academic paths, potentially delaying graduation or causing them to take unnecessary courses. Inconsistent or unreliable software might also undermine trust from both students and departments, reducing the tool's usefulness. Additionally, without a strong focus on security, there could be a risk of exposure to sensitive academic or personal data. Exposure to this kind of data can threaten privacy and confidentiality of student information. Failing to meet these ethical obligations would harm the very stakeholders the tool is designed to serve.

IX. Project Completion Status

This project aimed to create an automated and interactive tool to generate course flowcharts for the Mines community, fulfilling the need for a standardized approach to curriculum mapping across departments. Through automated scraping and processing of the Mines course catalog, the tool builds flowcharts that highlight course pre-requisites, co-requisites,

and post-requisites, helping departments and students better visualize academic pathways. The front-end, built with React, enables users to click on courses to view detailed information, toggle visual elements for course dependencies, and customize their flowcharts.

We successfully met all core functional and non-functional requirements, including automatic flowchart creation, an intuitive interface, and backend data processing in a format easily extendable for future development. Although stretch goals like integrating historical grade data to generate a difficulty heat map were not completed, the tool remains a powerful resource for planning and advising, accessible across departments. The project was delivered through a GitHub repository and AWS hosting credentials, positioning it for ongoing use and maintenance by Professor Bridgman and other Mines stakeholders.

Features Implemented and Summary of Feature Performance

Automated Flowchart Creation

Description: Automatically scrapes the course catalog to gather information for each department, generating flowcharts for each major that accurately display course pre-requisites and co-requisites.

Performance: The scraper successfully extracts data for all departments, allowing flowchart creation to be consistent and reliable. Flowcharts generated are as accurate as expected with inconsistencies in the catalog and cohesive across different departments.

- *Performance caveat:* the scraper depends on an AWS service (Bedrock) that can be curtailed at any point by AWS. For instance, during the middle of the semester, AWS incorrectly flagged the production account as a fraudulent account, thereby significantly reducing the number of calls per minute. While AWS restored the account's standing, it's possible that AWS can, as a service provider, arbitrarily reduce access to Bedrock at any point.

Interactive Flowchart

Description: Users can click on courses within the flowchart to highlight their pre-requisites and co-requisites. An information button on each course displays additional course details, and a toggle on each course allows users to switch highlighting of courses and arrows on and off for clearer visualization of dependencies. There is a settings menu that allows users to choose between modes and color schemes.

Performance: Interactivity is smooth and responsive, enhancing user engagement, according to usability tests with students across campus. The toggle functionality provides flexible visualization options, with minimal loading time when retrieving course information. Color schemes and themes were tested and adjusted amongst individuals with color blindness to best represent flowchart components.

User-Friendly UI (React JS)

Description: The front-end UI, designed in React, aimed to provide an intuitive and accessible experience, suitable for technical and non-technical users.

Performance: User feedback indicates that the interface was clear and straightforward, making flowchart navigation and customization easy. Front-end performance was optimized, ensuring a seamless user experience across different screen sizes.

Program Usability Across Campus

Description: The flowchart tool is distributable to all departments and offices on campus, allowing for easy access and distribution among stakeholders.

Performance: Deployment tests confirm that the tool can be used across various departments, with all necessary permissions and configurations in place to support accessibility campus-wide.

Backend Pipeline

Description: Developed a backend pipeline that processes catalog data into a machine-readable format (JSON), separated from flowchart generation for future scalability and adaptability.

Performance: The backend performs reliably, supported by comprehensive tests for Lambda connection, pipeline integration, and update processes.

- **Lambda Connection Tests:** Verified that Lambda could update from the CodeBuild S3 bucket, ensuring the backend pipeline adapts with every code change.
- **Pipeline Failure Tests:** Safeguards prevented production code from being affected by forced build or deployment failures, ensuring reliability and protection for production.
- **Scheduled Execution and Notification Tests:** A cron job confirmed regular Lambda execution, and notification tests ensured timely alerts on build failures, effectively streamlining backend reliability and maintenance.

Features Not Implemented

- Our team aimed to reimplement the backend by integrating ITS cloud data to replace the original setup with the webscraper, with the goal of achieving centralized data access, improved scalability, and enhanced security. However, dependency on ITS coordination hindered our progress. The integration required close collaboration with ITS to configure access and permissions, which proved challenging within the project's timeline. Despite these setbacks, we see this feature as a high priority for future development, as ITS cloud integration would streamline data handling, improve system reliability, and support secure data management.
- In terms of displaying prerequisites on the flowchart, the initial plan was to implement an arrow toggle button that would show arrows coming from courses with different arrow types to indicate corequisites vs. prerequisites. However, due to the complexity of calculating the directions of the arrow paths to avoid unnecessary overlapping, an alternative feature was implemented to have a prerequisite toggle on each course that will highlight courses and have temporary arrows.

X. Future Work

To enhance the usability and functionality of our project, we recommend several key areas for future work. Each area will require specific resources, skills, and time to implement effectively.

Gear the Platform Towards Students

To tailor the application for students, future work should focus on user experience design and student-focused features such as simplified navigation and academic progress tracking. This would require resources in UI/UX design software (e.g., Figma or Adobe XD) and user testing tools to gather feedback from students. Developers will need skills in user-centered design principles and experience in educational software. We estimate it will take 4–6 weeks to prototype and refine student-centered features.

Increase Customizability

Adding customizability features—such as personalized color schemes, adjustable layouts, and content filters—would empower users to tailor the tool to their preferences. This would require JavaScript and CSS frameworks (e.g., React with Material-UI), which are already in use, but additional frontend libraries may be helpful. Familiarity with advanced front-end design and state management is necessary. Implementation could take approximately 3–5 weeks.

Enable Flowchart Saving (PDF or Browser Cache)

Enabling users to save or export their flowcharts as a PDF or in the browser cache for later retrieval would significantly increase functionality. Resources for this feature include PDF libraries like jsPDF or HTML2Canvas, along with browser storage APIs for caching. Team members will need knowledge of client-side storage and export utilities in JavaScript. This feature would likely take 2–4 weeks to complete, including testing and debugging.

By addressing these areas, the project will better meet student needs, offer enhanced user experience through customizability, and provide practical options for saving work, making the tool more versatile and valuable to its users.

XI. Lessons Learned

- Working with AWS turned out to be more challenging than we initially anticipated. We spent more time than expected learning how to set up services like S3 and IAM roles securely and efficiently to connect to our front-end component. However, referencing resources online and referring to documentation helped navigate the nuances of permission controls and service setup overall. This experience taught us the importance of planning for a learning phase when working with complex technologies, especially when they are central to the project's infrastructure.
- ITS and the Registrar’s office have a database with course catalog information which could potentially replace our entire back-end component to avoid the need for web scraping the Mines course catalog and the inconsistencies that come with this process. Having the connection between the database and the front-end could be faster to pull course information and have the most updated version of the catalog with minimal manual maintenance.
- One of the biggest takeaways from this project was realizing the value of getting to know each other as a team right from the start. Establishing strong communication early on set the foundation for a collaborative and supportive environment where everyone felt comfortable sharing ideas and providing constructive feedback. During our initial meetings, we focused on understanding each other's strengths, work styles, and individual goals for the project. We found that this early bonding provided a framework for open, honest discussions and led to more productive problem-solving and innovative solutions.

XII. Acknowledgments

We would like to express our sincere gratitude to our client, Terry Bridgman, and advisors for their support and guidance throughout this project. Our client’s insights and feedback were invaluable in aligning our efforts with real-world needs, while our advisors provided essential mentorship, constructive feedback, and encouragement, helping us navigate challenges effectively. This project would not have been possible without their contributions, and we are deeply appreciative of their time, knowledge, and commitment.

XIII. Team Profile

Isaac Fry
Year: Senior
Discipline: Computer Science – General, Minor in Public Affairs (McBride)
Hometown: Windsor, CO
Work Experience: Data Engineering Intern (Chevron), Legislative Energy Policy Research Intern (State of Colorado), Quantitative Trading Intern (Solea Energy)
Activities: Music Director (Red Rocks Church)

Katrina Ngo
Year: Junior
Discipline: Computer Science – General
Hometown: Thornton, CO
Work Experience: Software Engineer Intern (Transamerica)
Activities: Teacher Education Alliance, DECtech

Maddi Tajchman
Year: Junior
Discipline: Computer Science – Business, Minor in Applied Mathematics and Statistics
Hometown: Firestone, CO
Work Experience: Software Engineering Intern (Trimble Inc)
Activities: Pi Beta Phi (Vice President of Recruitment), ACM-W member, SWE member

Jaden Nguyen
Year: Junior
Discipline: Computer Science – General
Hometown: Arvada, CO
Work Experience: Engineering Intern for the Department of Defense
Activities: Music Director (Miner Dissonance ACapella@Mines), Lead Peer Mentor (Parents and Families Programs)

Appendix A – Key Terms

Term	Definition
Web scraper	*A tool or script used to extract data from websites, automating the retrieval of specific information.*
Cron job	*A time-based scheduling utility in operating systems that automates tasks, such as running the web scraper periodically.*
Pre-req	*The courses that must be taken before a given course can be taken*
Co-req	*Course that must be taken at the same time as a given course*
Post-req	*The courses for which a given course is a pre-req*
LLM	*Large Language Model, similar to ChatGPT*
IRSA	*Institutional Research & Strategic Analysis, a department within the administration that can issue proprietary reports*
AWS	*Amazon Web Services, a cloud hosting service for storage and computation*
AWS Bedrock	*A functionality from AWS that integrates the storage of foundational LLMs and interaction with those models*
AWS Simple Storage Service (S3)	*A scalable cloud-based storage solution by Amazon Web Services, used to store and retrieve data.*
Near-Deterministic	*A process or operation that produces consistent results in most cases but might vary slightly due to inherently probabilistic structures.*