COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

Team Infinity 2

Summer Brown
Evan Harbach
Chiahsien Ko
Joseph Racicot

Revised June 10, 2025

CSCI 370 Summer 2025

Dr. Thompson

Table 1: Revision history

Revision	Date	Comments
New	05/16/2025	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	05/22/2025	Updated Sections I. Introduction II. Functional Requirements III. Non-functional Requirements IV. Risks V. Definition of Done Completed Sections: VI. System Architecture
Rev – 3	05/30/2025	Updated Sections I. Introduction II. Functional Requirements III. Non-functional Requirements IV. Risks V. Definition of Done VI. System Architecture Completed Sections: VII. Software Test and Quality VIII. Project Ethical Considerations
Rev – 4	06/07/2025	Updated Sections I. Introduction II. Functional Requirements III. Non-functional Requirements IV. Risks V. Definition of Done VI. System Architecture VII. Software Test and Quality VIII. Project Ethical Considerations Completed Sections: IX. Project Completion Status X. Future Work XI. Lessons Learned

		XII. Acknowledgments
Rev – 4	06/09/2025	Updated Sections I. Introduction II. Functional Requirements III. Non-functional Requirements IV. Risks V. Definition of Done VI. System Architecture VII. Software Test and Quality VIII. Project Ethical Considerations IX. Project Completion Status X. Future Work XI. Lessons Learned XII. Acknowledgments
Rev – 5	06/10/2025	Updated Sections I. Introduction II. Functional Requirements III. Non-functional Requirements IV. Risks V. Definition of Done VI. System Architecture VII. Software Test and Quality VIII. Project Ethical Considerations IX. Project Completion Status X. Future Work XI. Lessons Learned XII. Acknowledgments Appendix A – Key Terms

Table of Contents

I. Introduction

High level scope/description of the project

- Infinity Concrete wanted to improve the efficiency of its bid evaluation process. The manual process of reviewing incoming bid packages is time-consuming and inconsistent. Our goal was to build a system that can automatically extract key information from bid invitations and assist in the bid/no-bid decision process.
- We developed a system that processes incoming bid packages, extracts relevant details using local AI tools, summarizes the content, and provides a recommendation on whether to bid based on Infinity's internal decision-making criteria. This will reduce time spent on analysis and ensure more consistent evaluations.

Who is the Client?

- Infinity Concrete is a construction contractor that receives project bid invitations via email. These invitations often contain hundreds of pages of documents that need to be reviewed before deciding whether to submit a bid.

Why does Infinity want this project?

- Employees spend hours manually reading and evaluating each project invitation. Infinity wanted a system that automates the extraction and summarization of bid details, saving time and improving consistency.

Source of data

- While we initially considered using old data to train a model, we instead focused on a retrieval-based system. The client provided a sample set of past projects for development and testing purposes. The system will need to process new bid data as it arrives via email.

Description of definitions, acronyms and abbreviations

- **RAG**: Retrieval-Augmented Generation
- **LLM**: Large Language Model
- **PDF Parser**: Tool for extracting text from PDF documents
- **Vector DB**: Database for similarity-based retrieval using embeddings

Who will be the stakeholders/users of software?

- The employees of Infinity Concrete.

Who will be responsible for maintaining software?

- Infinity Concrete will maintain the system post-deployment. We provided documentation and recommendations for future updates.

II. Functional Requirements

- Automatically extract text from incoming bid documents (primarily PDFs).
- Summarize key information relevant to internal decision-making questions.
- Recommend a bid/no-bid outcome based on a set of heuristic rules.
- Present the summary and recommendation to employees in a readable format.
- Allow employees to provide feedback or corrections for future improvement/training.
- Automatically begin processing when a new bid email is received.

III. Non-Functional Requirements

The above functional requirements are supported by non-functional requirements, which impose constraints or design or implementation

- Accessible via both desktop and mobile platforms.
- Minimal ongoing maintenance is required.
- All data must remain private and secure: no external API calls or cloud processing.
- System must be robust to variation in document formatting.

IV. Risks

This section identifies and assesses key risks that could impact the successful delivery or adoption of the system, including technical and skills-related concerns.

- **Risk:** Exposure of proprietary or sensitive company data
  - **Likelihood:** Unlikely
  - **Impact:** Major
  - **Mitigation Plan:** The system avoids the use of external APIs or third-party LLMs like ChatGPT. All data processing is done on a local model. No data leaves the company's systems.
- **Risk:** Incorrect recommendations from the AI model due to lack of training data or label inconsistencies
  - **Likelihood:** Likely
  - **Impact:** Moderate
  - **Mitigation Plan:** Includes human feedback so that Infinity employees can refine model outputs for future training.
- **Risk:** Automated email ingestion fails due to email format changes
  - **Likelihood:** Likely
  - **Impact:** Moderate
  - **Mitigation Plan:** Currently only works on most common/requested formats and websites. Can potentially expand in future work.
- **Risk:** AI model becomes outdated or underperforms on future bids
  - **Likelihood:** Very Likely
  - **Impact:** Moderate
  - **Mitigation Plan:** Implemented a feedback loop so Infinity employees can provide corrections. At some level may need to accept due to quality of local model

- **Risk:** Lack of employee knowledge to maintain or update system
    - **Likelihood:** Unlikely
    - **Impact:** Moderate
    - **Mitigation Plan:** Delivered clear documentation. Built a simple UI for uploading new data and viewing results. Favored simple, maintainable code and automation scripts.
- **Risk:** Performance issues on mobile or lower-powered devices
    - **Likelihood:** Unlikely
    - **Impact:** Minor
    - **Mitigation Plan:** Only delivers the results on lower-powered devices

V. Definition of Done

The project is considered "done" when it meets the following minimum viable feature set, and has been successfully validated against real-world bid invitations provided by Infinity Concrete:

- **Minimum Functional Feature Set**
    - **Frontend Interface**: A web-based user interface, accessible on both desktop and mobile, that displays summaries and bid/no-bid recommendations in a readable and structured format.
    - **Backend Pipeline**: The system automatically extracts text from bid documents (PDFs) contained in ZIP packages extracted through websites hosting the bid document. Invitations to bid with a link to the bid documents are sent via email.
    - **Automation**: When ran the program automatically detects new unread bid invitations and begins to process them.
    - **Retrieval-Augmented Generation (RAG) Integration**: A local LLM (e.g., DeepSeek, served via Ollama) is used to generate summaries in response to internal decision-making questions. The LLM only processes locally retrieved, semantically relevant chunks of document content using a vector database for similarity search.
    - **Heuristic Recommendation Logic:** A transparent, editable scoring system is applied to generate a final bid/no-bid recommendation. The rationale for each recommendation is visible, with contributing factors clearly indicated (e.g., contractor, project type, location).
    - **Feedback Loop**: The system logs user input and manual overrides for summaries or recommendations, forming a basis for future model training or rule refinement.
    - **Testing**: Run on new bid packages to confirm output quality
- **Client Acceptance Testing**
    - Before handoff, the system must pass several acceptance tests defined with the client:
        - The client verifies that the summary accurately reflects the key information they would typically look for when evaluating a bid.
        - Specific internal questions (e.g., "Who is the GC?", "What's the project scope?") should be answered in the summary and the client verifies the answers as correct.
        - The recommendation should align with the decision the client would have made based on the available criteria.
        - The client confirms that the logic and scores behind the decision are interpretable and match their expectations.
- **Delivery and Handoff**
    - The system will be delivered digitally at the end of the field session vis Github.
    - This includes the source code, documentation, and installation instructions for local deployment.

VI. System Architecture

Our system is organized into four major components: data ingestion, document processing and storage, retrieval and summarization, and output generation. Each layer is responsible for a critical part of the bid evaluation pipeline, and together they form a complete, automated workflow that satisfies both functional and non-functional requirements.

Tools & Components Overview:

- **Document Processing**: PyMuPDF for efficient PDF text extraction
- **Retrieval Engine**: FAISS for storing and retrieving document embeddings
- **LLM Inference**: DeepSeek model hosted locally via Ollama
- **Summarization Logic**: Prompt templates + heuristic rules
- **Frontend**: Flask-based web interface (desktop/mobile compatible)
- **Automation**: Email listener or polling mechanism to ingest new bids

System Pipeline and Workflow

The end-to-end architecture is illustrated in **Figure 1**, which breaks down the system into four main stages:

1. Data Ingestion

The system starts by monitoring a designated inbox for new bid invitations. When an incoming bid email is detected, a script parses the email body to extract links. If links are present, the script navigates to hosting platforms (e.g., BuildingConnected or Dropbox) and downloads all relevant files and scrapes overview information.

2. Processing and Storage

Once the ZIP is downloaded, we extract all files and pass them through a PDF parser (PyMuPDF). The extracted raw text is then chunked into ~500-word segments and converted into embeddings using a local encoder (MiniLM). These chunks are stored in a vector database, enabling semantic searches using FAISS based on questions.

3. Retrieval and Summarization

For each project we have a summarization prompt that asks questions Infinity wants summarized (e.g., "Who is the GC?", "Is it a preferred job type?"). Once everything is stored in a vector db, we issue a semantic query with the prompt to the vector database. The top-k most relevant chunks are retrieved and passed as context to the local LLM (DeepSeek). DeepSeek then generates an answer or summary using structured prompts.

4. Summary + Recommendation Output

All generated summaries are collected and passed into a rule-based recommendation engine that assigns a Bid or No Bid label. This logic is encoded as a series of conditional checks and thresholds, which are explained in **Figure 2**. The logic also gives the project a score based on factors like specific distances, exact project size, and other factors. For example, a project 40 miles away and one 45 miles away might both be recommended bids, but the closer one will have a higher score. This hybrid structure provides transparency, tunability, and alignment with Infinity's real-world business rules.

After the summary and bid label is generated, the summary and bid recommendation are sent in an email back to Infinity. It is also additionally displayed in a Flask web interface which allows them to give feedback on the summary, reject or accept the recommendation, and also download the project ZIP file.

Modularity and Deployment

Each subsystem is designed to be modular. The email ingestion script, document parsing, embedding logic, LLM interface, and scoring engine can be updated independently. This not only improves maintainability but also enables future teams to swap components (a trained LLM for example) without reengineering the full pipeline.

For deployment, all components run locally. LLM inference and vector search are handled on a server-class machine, and the front-end can be accessed from any device on the same internal network.

Figure 1: Software Architecture Flowchart

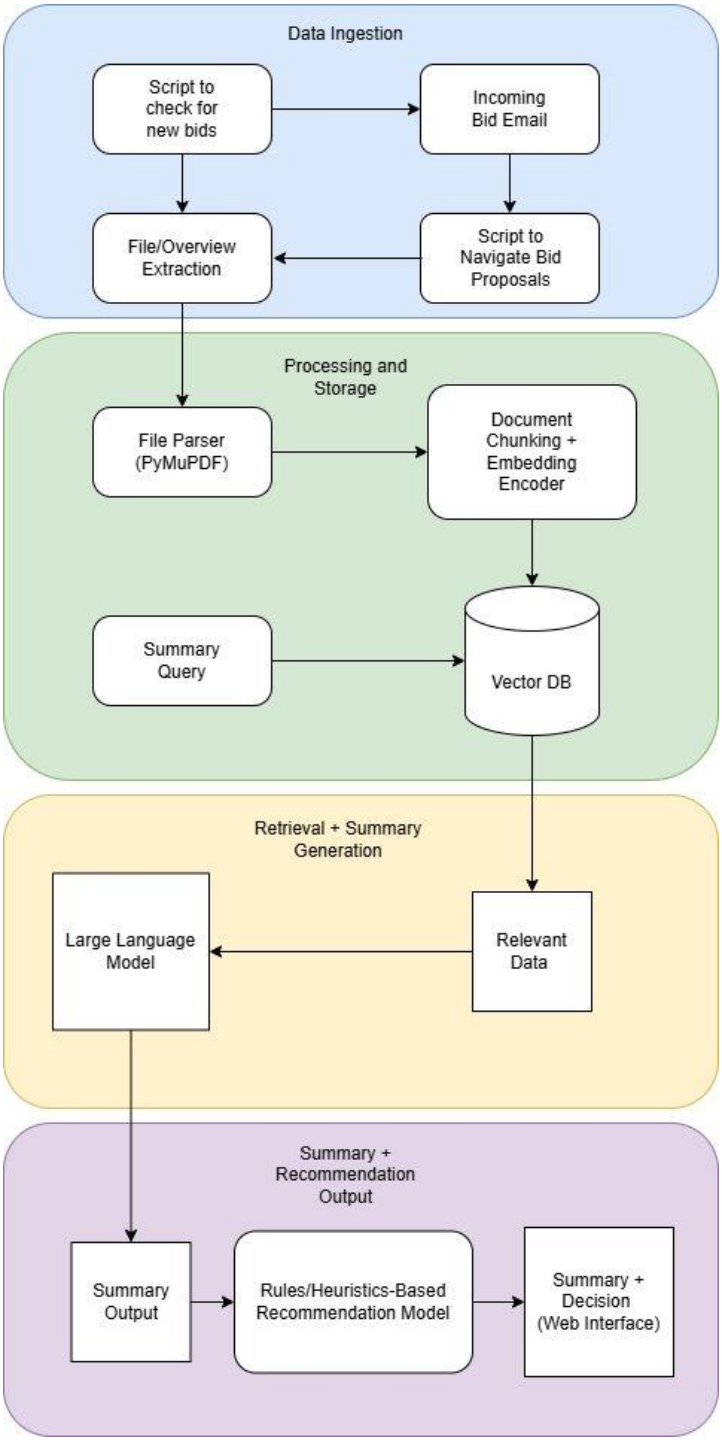
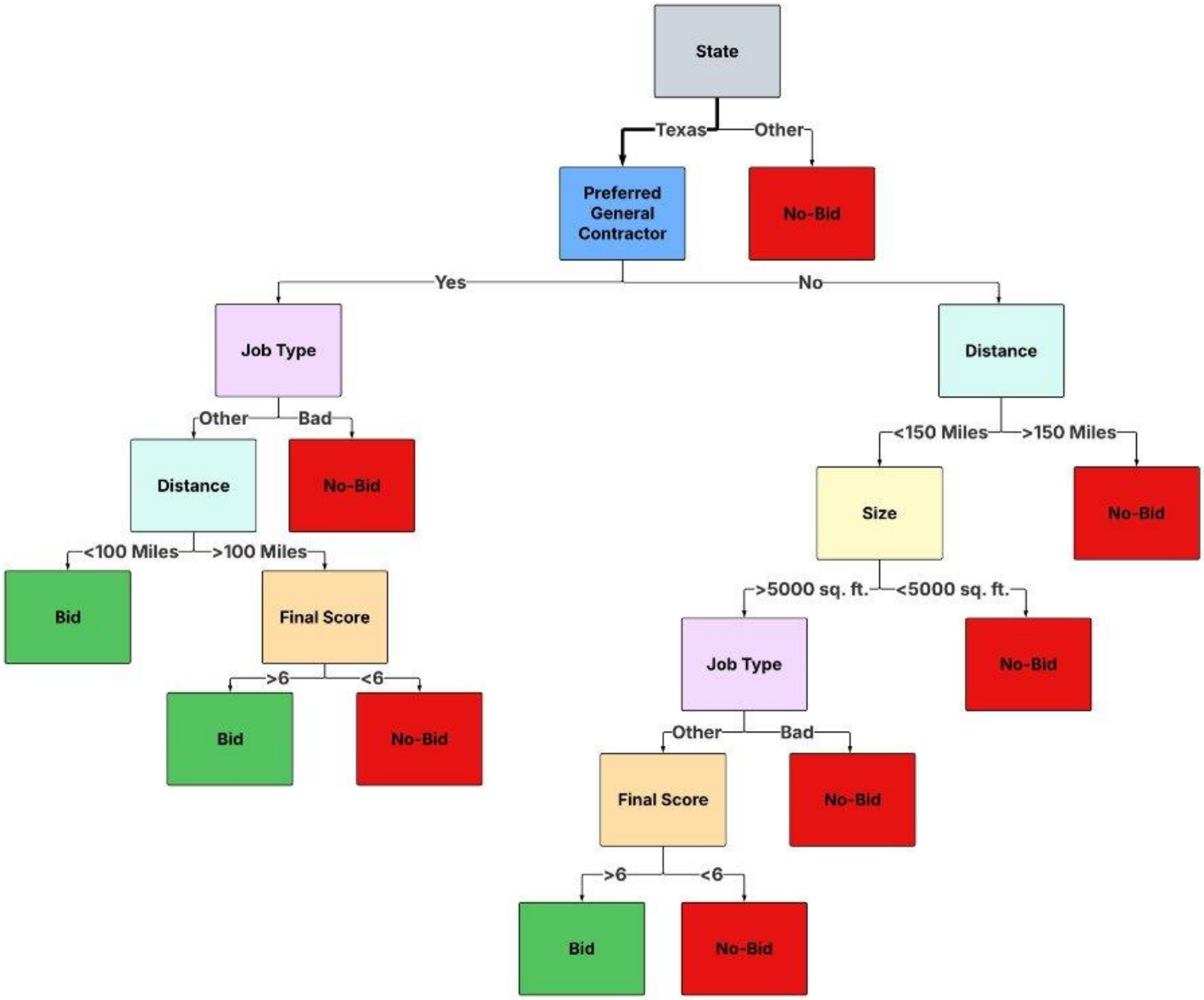


Figure 2: Recommendation Logic Flowchart



VII. Software Test and Quality

To ensure this system meets Infinity Concrete's expectations and requirements, we have defined a complete software quality plan. Each test plan is tied to a functional or non-functional requirement. This plan helped to ensure the system works as intended, is robust to variability, and is easy for Infinity Concrete employees to adopt and trust.

Functional Requirement Tests:

Functional Requirement 1: Automatically extract text from incoming bid documents (primarily PDFs)

- **Purpose:**
  To verify the system can reliably and consistently extract meaningful, structured text from a variety of PDF bid documents, regardless of formatting style. This is essential for enabling tasks such as summarization and heuristic evaluation.
- **Description:**
  A suite of PDF documents (varying in structure, length, layout) will be passed through the extraction module (using PyMuPDF). The extracted text will be compared to the visible content of the original PDF to determine fidelity. Manual validation will be used to check for missing or corrupted sections.
- **Tools Used:**
  PyMuPDF, manual validation checklists.
- **Threshold for Acceptability:**
  At least 95% of the text must be accurately extracted from standard-formatted bid documents.
- **Edge Cases:**
  - Scanned/image-based PDFs (no text layer).
  - PDFs with tables, rotated text, or multi-column layouts.
  - Documents with watermarks or embedded fonts.
- **Expected Result:**
  - Extracted text captures all important sections required for summarization and recommendation.
- **Result:**
  - Passed through several project packages of PDFs and were able to extract thousands to millions of words of text. For example, one project package included 985 MB of zipped files, and this was eventually converted into 4462 chunks of 500 words each which led to a useful summary output.

Functional Requirement 2: Summarize key information relevant to internal decision questions

- **Purpose:**
  To confirm that the AI model generates summaries that accurately extract and present the specific decision-relevant attributes defined by Infinity Concrete (e.g., general contractor identity, location, due date, scope).
- **Description:**
  After text extraction, documents will be fed through the summarization pipeline. Each summary will be manually checked against a ground truth answer sheet to determine if the required fields were correctly captured. A checklist of expected fields will guide validation.
- **Tools Used:**
  Local LLM via Ollama, structured prompts, manual scoring templates.
- **Threshold for Acceptability:**
  At least 80% of summaries must include correct and complete answers for all predefined internal decision questions. Infinity approves the summaries.
- **Edge Cases:**
  - Missing fields in original document (e.g., no GC name listed).
  - Ambiguous or scattered information (e.g., multiple dates or addresses).
- **Expected Result:**
  - Summaries return accurate, concise, and complete answers to internal questions.
- **Result:**

- Tested against a set of roughly 10 past projects where the information that Infinity wanted extracted was known. Tested each project 3 times. 29/30 it got all the major information correct. Infinity accepted these results.

Functional Requirement 3: Recommend a bid/no-bid outcome based on a set of heuristic rules

- **Purpose:**
  To validate that the heuristic evaluation engine properly interprets summary inputs and applies Infinity Concrete's decision logic to produce actionable recommendations.
- **Description:**
  The system's logic will be tested using a collection of mock summaries with known expected outcomes. We will evaluate whether the recommendation (bid/no-bid) matches what would be expected from human application of the heuristics. We will get a review from Infinity. Additionally, we have unit tests to ensure that it is correctly evaluated based on specific situations.
- **Tools Used:**
  Heuristic rules engine, manually labeled test cases.
- **Threshold for Acceptability:**
  Infinity approves the rules and their outcomes
- **Edge Cases:**
  - Conflicting criteria (e.g., preferred GC but far location).
  - Ambiguous data or summaries missing in a key field.
- **Expected Result:**
  - Heuristic engine outputs align with client bid decision logic.
- **Result:**
  - Tested against a set of roughly 10 past projects where it was known what the past bid decision was. It did not have great accuracy, but the reasons for different decisions were all factors the current model is not able to consider. Infinity reviewed the results and the logic of how it got them and accepted them. All unit tests passed.

Functional Requirement 4: Present the summary and recommendation in a readable format

- **Purpose:**
  To ensure users can easily read, understand, and act on the summaries and recommendations provided by the system.
- **Description:**
  The front-end interface will be tested on desktop and mobile browsers to evaluate layout, structure, and usability. Feedback will be collected from peers and clients to evaluate the clarity of displayed content.
- **Tools Used:**
  Flask web frontend, phones.
- **Threshold for Acceptability:**
  Output must be legible and well-structured across screen sizes with no broken layout elements. Infinity must accept the output.
- **Edge Cases:**
  - Large/overflowing summaries.
  - Devices with small screens.
- **Expected Result:**
  - Summary and decision outputs are consistently readable and properly formatted.
- **Result:**

- Easily viewed on all devices via email and the web application.

Functional Requirement 5: Allow employees to provide feedback or corrections

- **Purpose:**
  To confirm that user corrections or suggestions about summary quality and bid decisions are correctly recorded for future training, analysis, or retraining.
- **Description:**
  Testers will submit feedback through the UI on various summaries. Back-end logs will be reviewed to verify that the data is received, stored, and labeled correctly.
- **Tools Used:**
  Web UI form, backend logging system, feedback review scripts.
- **Threshold for Acceptability:**
  Feedback is recorded accurately in at least 95% of cases.
- **Edge Cases:**
  - Duplicate submissions.
  - Feedback submitted with malformed or missing fields.
- **Expected Result:**
  - Feedback is saved and accessible for future processing.
- **Result:**
  - Feedback saved on all tests.

Functional Requirement 6: Automatically begin processing when a new bid email is received

- **Purpose:**
  To validate that new bid packages received via email trigger the entire document pipeline from ingestion to summary to recommendation without manual intervention.
- **Description:**
  A simulated email inbox will be used to send sample bid invites with attachments. The system will be monitored for whether it detects the email, processes the files, and displays output.
- **Tools Used:**
  Email listener script, local server, logging monitor.
- **Threshold for Acceptability:**
  At least 90% of valid incoming bid emails must be detected and processed end-to-end for the websites we talked about with Infinity.
- **Edge Cases:**
  - Attachments with invalid formats.
  - Multipart emails with multiple documents.
- **Expected Result:**
  - System triggers full pipeline after email receipt.
- **Result:**
  - For all of the bid packages from websites we were asked to make work they were detected and passed through the pipeline end to end. There are some projects from websites the program cannot currently handle, but the program recognizes them and passes them over.

Non-Functional Requirement Tests:

Non-Functional Requirement 1: Accessible via desktop and mobile

- **Purpose:**
  To ensure the front end is usable and responsive across different device types and screen sizes.
- **Description:**
  The web UI will be tested on desktop browsers and mobile devices for navigation, content formatting, and interaction. Key elements like text boxes, buttons, and summaries will be reviewed.
- **Tools Used:**
  Chrome/Firefox/Safari, Phones.
- **Threshold for Acceptability:**
  All primary features must function correctly on both device types.
- **Edge Cases:**
  - Mobile browsers with limited JavaScript support.
- **Expected Result:**
  - Clean, functional experience on both desktop and mobile.
- **Result:**
  - Clean output in both email and on the web application.

Non-Functional Requirement 2: Minimal ongoing maintenance required

- **Purpose:**
  To test that the system is stable and does not require frequent restarts, fixes, or technical intervention.
- **Description:**
  Run multiple bid processing sessions without modifying configuration files or resetting components. Evaluate crash logs and system reliability.
- **Tools Used:**
  Python logging module, long-run test scheduler, error trackers.
- **Threshold for Acceptability:**
  System runs 10+ bid processing cycles with no crashes or manual corrections needed.
- **Edge Cases:**
  - Large bid files.
  - Partial file downloads or mid-process interruptions.
- **Expected Result:**
  - High uptime and minimal operator intervention.
- **Result:**
  - Easy to maintain with a lot of documentation. The most maintenance that should be needed is to restart the program if it breaks. The pipeline was tested several times from end to end and it worked without breaking.

Non-Functional Requirement 3: All data must remain private and secure

- **Purpose:**
  To guarantee that proprietary bid data is never exposed to outside networks or APIs.
- **Description:**
  Monitor system behavior to ensure no traffic is sent externally. Inspect dependencies and ensure no cloud services are invoked.

- **Threshold for Acceptability:**
  No outbound HTTP/HTTPS traffic related to bid content.
- **Expected Result:**
  - Confirmed full local-only processing.
- **Result:**
  - Local only processing of data confirmed. Summaries are shared but only on the local network.

Non-Functional Requirement 4: System must be robust to formatting variation

- **Purpose:**
  To evaluate how well the system handles structurally diverse PDFs (headers, fonts, layouts).
- **Description:**
  A sample set of variably formatted documents will be processed. Output summaries will be compared against expectations to assess consistency.
- **Tools Used:**
  Manually prepared diverse PDFs, summary comparison checklist.
- **Threshold for Acceptability:**
  At least 90% of test PDFs must generate meaningful summaries despite formatting differences.
- **Edge Cases:**
  - Missing metadata.
  - Watermarked or scanned text.
- **Expected Result:**
  - Model continues to return relevant summaries with non-standard inputs.
- **Result:**
  - Full pipeline works end to end across different websites that host the bid and with different files/file organization as inputs. Over 90% of summaries produce meaningful and accurate information.

VIII. Project Ethical Considerations

The development of this system for Infinity Concrete introduced several ethical responsibilities. These ranged from ensuring the privacy and integrity of sensitive bid information to designing a system that employees can trust and rely upon. We have carefully considered how our work aligns with the ACM/IEEE Code of Ethics, and how our choices impact stakeholders both directly and indirectly.

Several ACM/IEEE Principles are particularly pertinent to this project:

Principle 1: Public
As outlined in 1.03 and 1.04, software engineers should only approve software they believe is safe and should disclose any danger to users or the public. Our project ensures that summaries and bid recommendations are based on high-quality document extraction and do not mislead decision-makers. Since these decisions can significantly affect company operations, we ensured that only thoroughly tested systems were delivered.

Principle 2: Client and Employer
According to 2.01 and 2.06, engineers must work within their areas of competence and report any potential failures. We maintained transparency with the client about the limitations of heuristic models and the risk of imperfect recommendations, and we documented failure cases we discovered during testing.

Principle 3: Product
Sections 3.10 and 3.11 emphasize the importance of adequate testing and documentation. We conducted

comprehensive manual testing for each feature and recorded known edge cases, failures, and adjustments to heuristics. Our final report includes full documentation of the system's capabilities and limitations.

Principle 6: Profession
6.08 states that software engineers should detect, correct, and report errors. We have built a feedback mechanism to support this. Employees can report problems with a recommendation, which helps support long-term system integrity and ethical practice.

Principles at Risk of Being Violated

1.03 (Public interest): If the summarization or recommendation engine fails to extract or interpret data properly, the system might produce inaccurate recommendations, leading to bad business decisions. To mitigate this, human-in-the-loop validation was essential for fully trusting system outputs.

2.05 (Confidentiality): The system will handle sensitive company data. If improperly secured, it could lead to exposure. We've mitigated this by explicitly avoiding cloud-based APIs and hosting all models and data processing locally, in accordance with Infinity's stated preferences.

Application of Michael Davis's Tests:

Test 1: The Publicity Test:
Would I be comfortable having my decision reported in the news?

Yes. We believe designing a fully local system with strong attention to privacy and data security would be positively received by the public. We were open about our methods, included human oversight, and allowed the client full control over their data.

Test 2: The Reversibility Test:
Would I think this choice was fair if I were on the receiving end?

Yes. If we were employees using this system, we would appreciate the ability to provide feedback, see clearly stated reasons for each recommendation, and know that no sensitive data is being shared externally. This transparency and feedback loop supports user empowerment rather than automation overreach.

Failing to implement a comprehensive quality plan introduces serious ethical risks:

Misinformation: Inaccurate summaries may lead to poor decision-making, which could damage Infinity Concrete's finances or credibility.

Erosion of Trust: If the system behaves unpredictably or provides inconsistent recommendations, employees may stop trusting or using it, rendering the solution ineffective.

Security Breach Potential: In the absence of proper testing or monitoring, third-party libraries or scripts could unintentionally leak data.

Inequitable Outcomes: Without feedback mechanisms and error correction, the system may overlook bias in summaries or rules, reinforcing incorrect assumptions about what makes a bid "worthwhile."

To avoid these outcomes, we conducted both functional and non-functional tests, focusing on failure, and provided documentation to guide the client in system upkeep.

IX. Project Completion Status

At the conclusion of the field session, our team successfully developed and tested a functional prototype that meets the core objectives and addresses the requirements set forth by Infinity Concrete. The system can automatically extract text from incoming bid documents, summarize relevant project details using a locally hosted LLM (DeepSeek), and generate a structured bid/no-bid recommendation using a configurable set of heuristic rules. These functionalities are presented through two primary interfaces: an automated email reply mechanism that sends results back to the company, and a user-friendly web-based interface compatible with both desktop and mobile devices.

We completed all necessary aspects by the definition of done which outlined the following as core deliverables: PDF extraction, RAG-based summarization, recommendation generation, frontend UI for output, automatic triggering from email input, and a feedback loop. All of these components were implemented and tested, to ensure the system behaves reliably under realistic use conditions. We also emphasized minimal operator intervention and local-only data processing, which were key non-functional requirements expressed by the client.

Key features that were completed and tested include:

PDF Extraction: The document parser was run on bid packages of various sizes, ranging from zip packages sizes of under 100MB to over 1GB. These included documents with complex formatting, multiple columns, tables, and rotated text. In all cases, the system returned extracted text that aligned with visible content.

Summarization Accuracy: Summaries were evaluated manually and reviewed by Infinity. The content was confirmed to be accurate. The LLM-based summaries correctly answered internal decision questions such as location, due date, general contractor, and project size in the majority of test cases.

Recommendation Engine: The rules-based evaluation logic successfully translated summary content into a recommendation that aligned with expected outcomes. The client reviewed this logic and confirmed that it captured their internal decision process.

Frontend Presentation: The Flask-based interface was tested on mobile phones and laptops. All visual content rendered correctly, with no layout issues observed. Both summary and recommendation were presented clearly, and the feedback mechanism functioned as expected.

Automation: Simulated bid emails successfully triggered the document processing pipeline, from file ingestion to summary and recommendation delivery. However, the program is still run manually because when a new email is detected you must click your account to authorize going into the email.

Despite the success of the prototype, a few planned features were deferred or partially implemented. Early in the project, we explored the possibility of fine-tuning the LLM using historical labeled data. However, this approach was deemed impractical due to insufficient data and resources for training. We pivoted to a more transparent and interpretable heuristic rule system, which ultimately offered greater client confidence and less technical complexity.

Overall, the project delivered a robust, working system that satisfies all critical functional and non-functional requirements. The client has expressed enthusiasm for continuing development, and the system is well-positioned for future expansion and deployment.

X. Future Work

While our current system represents a fully functional prototype that satisfies the key requirements laid out by Infinity Concrete, there remain several opportunities for continued development and refinement. These future improvements fall into several categories: extending document source coverage, improving robustness, introducing machine learning enhancements, and expanding system logging and observability.

One of the most immediate next steps is to finalize integration with additional bid hosting platforms, such as ConstructConnect and Dropbox. While scripts have been implemented for some of these sources, they have not been fully tested or adapted to the wide range of bid formats present on every site. Completing and generalizing this functionality will significantly expand the system's utility by ensuring broader coverage of incoming bid requests, particularly as general contractors submit bids from a growing variety of platforms.

A second step to improve the system would be to find a way to automate verifying your account when the program is run. Currently when the system is ran you will need to click your email account to verify your account before it can go in and read the contents of the email. This prevents the system from being fully automated. Finding a way to automate this would allow the whole pipeline to process once a new email is detected.

A third area of focus should be improving the system's error handling and fault tolerance. During development, we encountered several edge cases that would cause the system to crash or silently fail. Future work should include the implementation of a more robust error handling framework. This includes writing detailed error messages to a dedicated log file when failures occur and providing fallbacks or user prompts where applicable. Adding try/except guards around key processing blocks and validating file structures early in the pipeline will greatly improve system reliability in production.

Another promising direction is to develop a logging system for storing the rationale behind bid/no-bid decisions. Currently, the system outputs a recommendation and summary but does not capture the factors outside of the program that may lead to Infinity rejecting or accepting a bid. Capturing and storing this reasoning, alongside human feedback, would lay the groundwork for future training and evaluation.

Related to this, a long-term goal is to train a custom machine learning model to predict bid/no-bid decisions. Although our current rules-based system offers interpretability and aligns well with Infinity's internal logic, a data-driven model could help uncover patterns and edge cases not covered by static rules. This work will require gathering and labeling a large number of historical bids with decision outcomes and rationale; a potentially months-long process depending on resource availability. We estimate that collecting and preparing a labeled training set would take 4+ weeks, followed by an additional 4-6+ weeks for model training and validation.

Other enhancements include developing better image analysis capabilities, especially for bid documents that include embedded drawings, diagrams, or maps. At present, the system focuses on text-based summaries. Leveraging OCR or computer vision tools could allow us to extract meaningful information from image-based content, which is often critical in construction bids.

In summary, future work on this project involves both hardening the current system for production use and laying the foundation for more intelligent, data-driven decision-making. This will require not only development time but also additional resources such as labeled data, computing hardware for model training, and possibly licenses for commercial OCR tools. With continued investment, the system can evolve into a powerful, fully autonomous assistant for bid evaluation at Infinity Concrete.

To pursue this work, we recommend a future team composed of 4–5 members. Useful background knowledge would include:

- Python
- Flask
- Frontend/backend integration
- Machine Learning
- Prompt Engineering

It could require more than just an additional field session to complete all future work, but a potential field session team should have a solid grounding to continue the project.

XI. Lessons Learned

This project exposed our team to the complexities of working with unstructured data, the trade-offs between automation and interpretability, and the real-world challenges of deploying AI systems within business environments. We learned that open-source LLMs, while powerful, are not magical. Their outputs require thoughtful design and review, especially when users depend on them for important business decisions.

Perhaps one of the most valuable insights was that a carefully designed heuristic system can offer excellent performance when paired with structured summaries. We expected to rely more heavily on learning-based methods but came to appreciate the transparency and control that rule-based systems offer in early-stage development.

Another lesson was variability in bid formats. No two bid packages were exactly alike, and extracting useful text from PDFs required us to experiment with working through different vendor websites, chunking strategies, manual validation techniques and more. This reinforced the need for systems to be robust and flexible.

Another lesson was how important it was to communicate with each other. We met in person almost every weekday of the field session and think this greatly helped us quickly solve issues and be able to get quick feedback from one another.

Most importantly, we saw how crucial it is to stay in constant contact with the client. Our early interviews and regular check-ins with Infinity Concrete helped us build features that actually solved their issues, not just what we imagined they needed. This kind of communication shaped everything from our UI design to how we framed summaries.

XII. Acknowledgments

We are grateful to Infinity Concrete for sharing their time, expertise, and data with us. Their willingness to provide feedback and explain their internal decision-making processes was key to our system's relevance and usefulness. We particularly would like to thank our main contact with Infinity, Jennifer Wood. Jennifer was very understanding and spent a lot of time trying to get us what we needed for the project.

We would also like to thank our field session advisor, Professor Rob Thompson, whose guidance helped us stay on track and continuously refine our technical and professional approach.

XIII. Team Profile

Summer Brown
Junior
Computer Science
Hometown: West End, NC
Work Experience: Software Engineer Intern at Robotic Systems Integration, ML Research at Mines
Sports/Activities/Interests: Pickleball, watching baseball, reading, etc

Evan Harbach
Junior
Hometown: Argyle, TX
Sports/Activities/Interests: Baseball, Basketball, Camping, Hunting, Skiing

Chiahsien Ko
Senior
Computer Science
Hometown: Taipei
Work Experience: Marriott International Inc.
Sports/Activities/Interests: Travel, NBA, MLB

Joseph Racicot
Senior
Computer Science + Data Science
Hometown: Houston, TX
Work Experience: Data Science Intern at Cheniere Energy
Sports/Activities/Interests: Golf, Sports Analytics/MLB/NFL, Energy

References

[1] https://www.nvidia.com/en-us/glossary/large-language-models/

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	Definition
Large Language Model (LLM)	*"Large language models largely represent a class of deep learning architectures called transformer networks. A transformer model is a neural network that learns context and meaning by tracking relationships in sequential data, like the words in this sentence.* *A transformer is made up of multiple transformer blocks, also known as layers. For example, a transformer has self-attention layers, feed-forward layers, and normalization layers, all working together to decipher input to predict streams of output at inference. The layers can be stacked to make deeper transformers and powerful language models.[1]"* *Example: ChatGPT GPT-3.* *Can process and present data in a human readable way*
Retrieval Augmented Generation (RAG)	*Retrieval-Augmented Generation (RAG) is a hybrid architecture that combines information retrieval techniques with natural language generation. Rather than relying solely on the model's internal knowledge, a RAG system enhances responses by first retrieving relevant content from an external source (such as a document database) and then using a language model to generate context-aware output.* *RAG works in two main stages: (1) retrieval, where the system finds relevant passages or data based on a query, and (2) generation, where a language model conditions its response on the retrieved information. This approach enables models to produce more grounded, up-to-date, and context-specific answers, especially in domains where the data is too large or dynamic to train into the model directly.*

	Example: Asking a model "What is the scope of work for Project X?" may retrieve matching sections from a PDF and then summarize the scope based on that content.
PDF Parser	*A PDF parser is a software tool that extracts structured or unstructured content from Portable Document Format (PDF) files. Since PDFs are designed for layout and presentation rather than machine readability, parsing involves identifying text blocks, reading embedded metadata, recognizing document structure (e.g., headers, tables), and converting content into a usable format such as plain text or HTML.* *Modern PDF parsers may use rule-based parsing, layout recognition, or OCR (optical character recognition) to process scanned documents. Extracted data can then be used for downstream tasks such as information retrieval, summarization, or automated classification.* *Example: PyMuPDF is a Python-based parser capable of reading text, annotations, fonts, and document outlines from PDF files.*
Vector Database (Vector DB)	*A vector database is a specialized storage and retrieval system designed to manage high-dimensional numerical representations of data, commonly known as embeddings. These embeddings capture semantic meaning in a format that enables fast similarity search using operations like cosine distance or inner product.* *Vector databases are essential in AI systems that rely on semantic retrieval. When a text, image, or other data point is encoded into a vector, a vector DB can efficiently find the most similar entries from millions of others. This makes them a core component in retrieval-augmented generation (RAG), recommendation engines, and semantic search platforms.* *Example: FAISS and Chroma are popular vector databases used to retrieve document chunks most relevant to a given question, which are then passed into a language model for summarization.*