COLORADO SCHOOL OF MINES

CSCI 370 Final Report

SAGE

Smart Assessment and Grading Engine

Revised Dec 12, 2025

CSCI 370 Fall 2025

Prof. Kathleen Kelly

Table 1: Revision History

Revision	Date	Comments
New	8/29/25	Created Document, outlined Table of Contents
Rev – 2	9/7/25	Completed Introduction, Functional Requirements, Non-Functional Requirements, Risks, and Definition of Done
Rev – 3	9/21/25	Completed System Architecture, Software Test and Quality, and Ethical Considerations
Rev – 4	10/26/25	Completed Project Completion Status, Future Work, and Lessons Learned
Rev – 5	11/30/25	Finishing touches for draft
Rev – 6	12/12/2025	Revisions based on peer feedback, updated to current Mines logo, minor edits

Table of Contents

I. Introduction

On November 30, 2022, OpenAI released a product that would change the world. Although not the first of its kind, ChatGPT—a conversational AI chatbot—could now answer questions related to a variety of subjects; ranging from tips on how to introduce yourself to your neighbor to fixing a bug in a code segment. The market of similar models grew exponentially with Google's Gemini, Anthropic's Claude, and many more. Mundane tasks such as writing emails or taking notes became effortless while some jobs were entirely replaced like customer service. In under a year, this technology grew to affect everyone's lives.

This impact was felt even in higher education. While generative AI has many potential benefits, it also contains several severe drawbacks. For higher education, it poses a massive risk to academic integrity. Students could now use AI models to complete their homework assignments, projects, and even exams. Teachers can no longer trust that work students are submitting is wholly their own. Even if they have suspicions of plagiarism, it is difficult to prove and often goes unnoticed. Due to the difficulties caused by generative AI, Mines faculty have had to adapt and look for solutions. In the years prior, the Computer Science Department heavily relied on projects and homework assignments to ensure that students were understanding their classes properly; however, with frequent AI usage, this is becoming less effective as many students choose to finish their assignments by talking to chat models instead of referencing class materials and actually learning. As a result, the Computer Science department has had to increase their reliance on in-person pen and paper exams. This eliminates most worries about students cheating using the internet or generative AI; unfortunately, this solution has several drawbacks. Primarily, professors and teaching assistants will now have to spend much more time grading assignments and exams that are conducted using paper.

Currently, the process for grading paper exams is as follows:

1. Exams are scanned in large batches and saved as PDFs
2. Each batch is broken down into individual students and uploaded to their respective Gradescope portfolios
3. Professors go through each student and grade the exam
4. The grade is returned to the student on Gradescope and Canvas

This process can be significantly sped up using our product SAGE (Smart Assessment and Grading Engine). SAGE is an application that streamlines the grading part of the process. All professors have to do is upload the batched PDFs into the portal, then SAGE would then go through the pages, identify the different students, grade their work in adherence to a rubric provided by the professor, and return the grades along with a detailed report on what the student did incorrectly.

This will extensively lower the amount of tedious tasks that professors and teaching assistants have to do thus freeing their time to work on improving courses and provide students with the best education possible. The report generation feature will also benefit students since they will be able to better understand why they missed points on their work in a reasonable amount of time after exams.

II. Functional Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this section are to be interpreted as described in RFC 2119.

The program MAY assume that all input is STEM focused and that all natural language is in English only. All tasks below are assumed to be done in sequence but MAY be parallelized, streamed, and/or batched to improve performance; doing so MUST NOT negatively impact accuracy.

Loading Input

The program MUST accept a multi-page scanned PDF file of 300dpi but MAY change the resolution internally if found beneficial for further functions. It SHALL group the incoming pages based off of the student names written on each page, and nameless pages SHALL be assigned to the last name found. It is RECOMMENDED that additional information—possibly from an external source—such as a list of students or ID numbers be used to ensure the accuracy of the naming process. Input MAY be normalized and preprocessed before extracting text or MAY be assumed to be upright, clear, and ordered.

The program SHALL extract the written answers and comments that do not exist on a blank copy of the assignment—both short expressions and longer content—as machine text for each page. It SHOULD also tag each answer with any relevant question label found (e.g., “#1”, “2b)”, “1.c.iv”). Ideally, the program SHOULD determine a confidence value between zero and one for each answer extracted. Text extraction SHOULD be done with the highest accuracy possible and SHOULD be the focus of the design.

The program SHOULD convert lone math expressions into LaTeX when not found in the context of written code which MUST be kept as plain ASCII. The program SHOULD extract all written tables, figures, charts, drawings, and other non-textual content for later use.

Grading

The program SHALL take in a rubric listing the questions and criteria for each answer that award or deduct points. The program SHALL apply the rubric to each students’ contents, compute per question point values, and track the different criteria met and failed. When handling the rubric, the program MUST allow for a wide variety of criteria to facilitate the grading of multiple choice, true or false, fill in the blank, symbolic mathematical, computational mathematical, short response, and long response questions. The program SHOULD also allow rubrics that handle diagram, table, chart or other graphical response questions, and then SHOULD consider any previously extracted tables, figures, charts, drawings, or other non-textual content for the appropriate questions when applying the rubric. The rubric MUST support complex criteria involving negation, union, and conjunction of answers (e.g., to allow anything but an answer, multiple possible answers, or require multiple different answers at once). Additionally, it MUST support specifying various degrees of tolerance (e.g., numerical accuracy within a range versus exact numbers and loose spelling in sentences versus strict spelling in code). The program SHOULD achieve a 95% accuracy compared to the results of manual grading with the same rubric by a moderately skilled human. The program SHOULD be slightly forgiving of small errors in the text content of answers since they might be an artifact of previous steps. These results SHOULD be saved to an intermediary format.

Report Generation

After grading the inputs, the program MUST compile per student reports for each student and a composite report. Each per student report MUST list the student name (and any OPTIONAL supporting details used like ID number), the current timestamp, the exam name or number, the grade as a percentage or fraction of the rubric's defined total or both (this distinction MAY be specified by the user or left to the program), and the list of questions where full credit was not awarded and the reason why (questions that did receive full credit MAY still be included here with an empty reason or with a justification of correctness). Additionally, the rubric, intermediary machine text, and any extracted RECOMMENDED tables, figures, charts, drawings, or other non-textual content used to grade the student MUST be appended to the report.

Interface

The program MUST expose a simple, functional web interface for controlling its execution. This interface MUST allow for inputting the scanned PDF files, rubric, and any other supplemental information needed by the other components of the program. The interface MUST NOT expose any internal secrets such as API keys used by the rest of the program. After data is input, the program MAY ask for review and confirmation; then, the program SHOULD provide some status indicator, progress bar, percentage, spinner, or other display to let the user view the status of its execution. The program MUST either allow multiple jobs at a time or implement a queue so that users can upload at any time (not including system downtime). If a system error is encountered, the program MUST report this to the user and log details for later debugging. Upon completion, the interface MUST provide the user with access to view and save the aforementioned reports and an OPTIONAL immediate summary. The program SHOULD store the results for later retrieval and continue to completion even if the interface itself is closed. The retrieval after an interface session ended SHOULD be designed to be moderately convenient such as returning to the last result when reopened. The interface MAY provide an option to abort, cancel, or kill the current grading process. Additionally, the interface (or the program via another means) MAY allow the user to manually review and correct results then re-generate affected report files with a note of changes included within.

III. Non-Functional Requirements

Beyond defining what the system must do, it is equally important to establish how the system should operate and under what constraints. These non-functional requirements describe the qualities, limitations, and validation measures that will ensure the project meets expectations for both users and stakeholders.

Cost

The project budget is limited to $2000 for all development expenses such as API calls to LLM providers. This budget applies across all phases of the project, including research, development, testing, and the pilot run. Expenditures must remain within this limit.

Data Privacy

The project necessarily involves data from ongoing courses including sensitive student information such as names and grades; therefore, care must be taken that the information we are entrusted with is protected—especially due to legal requirements like FERPA (Family Educational Rights and Privacy Act). Sensitive information must never be stored or transmitted in a publicly accessible manner. Additionally, any data shared with third parties must comply with FERPA and not identify students unless narrow exceptions are met. Finally, since sample data may involve real past or present students, all team members are required to maintain confidentiality regarding test questions, student info, and grades.

Performance

The system must operate efficiently and reliably under load while supporting multiple concurrent users without degradation in performance. Performance considerations include responsiveness, availability, and fault tolerance, thus ensuring that users can depend on the system for consistent results throughout its operation.

User Experience

The user interface should be intuitive, accessible, and consistent, with a design that prioritizes ease of use for instructors. The design should emphasize clarity, simplicity, and responsiveness. Feedback messages (such as clear error messages or status indicators) should be provided to guide users and reduce confusion. The experience should encourage confidence in the system's reliability and ease of use.

Pilot Run with MATH 111 and CSCI 128

A pilot implementation took place two-thirds of the way through the semester in MATH 111 and CSCI 128. The system supported the complete workflow defined in the functional requirements while ensuring no disruption to normal class operations. System outputs were evaluated against real instructor-assigned grades for validation.

IV. Risks

Risks are present in any project, but with this project's heavy use of AI, there are a lot to consider. The volatile nature of modern LLMs make it difficult to predict how they will act even with very specific prompt engineering. Additionally, LLMs present new tampering risks such as prompt injection that may allow students or even other LLMs to take control of output. With the proper considerations, these risks can be mitigated or sometimes eliminated altogether; however, it is imperative to consider them within our system especially to prevent bias or failures.

A. **Hallucinations**
   a. **Improper Equalities:** Questions which require "work shown" may have inconsistencies due to the LLM failing to validate an equality.
   b. **Ignoring the rubric:** The LLM may choose to ignore the rubric and assign points arbitrarily at times. It may also inject forms of bias such as those derived from names.
   c. **Renaming/Misidentification of the evaluated student:** A student may be accidentally misidentified due to either hallucinations or OCR failure. This could lead to the wrong student being graded and work being returned to the wrong individual.

B. **Prompt Injection**
   a. **Handwritten prompts:** A student may attempt to inject prompts to get a better grade or to force a professor to grade their exam by hand.
   b. **Accidental injection from LLM output:** An LLM response earlier in the pipeline may accidentally corrupt a later response. This could lead to type errors and general data handling failures.

C. **Character Recognition Failure**
   a. **Misspelling words:** A misspelled word by the character recognition software may cause a misgrade by the grading system.
   b. **Changing words:** A word may be changed by the character recognition software or a downstream LLM which may change the meaning of a student's response.
   c. **Failing to read words:** The system may fail to read a word which may result in point decreases by the LLM.
   d. **Math input readability failure:** Math input may suffer from differing syntax between students resulting in an incorrect grade.

V. Definition of Done

The project will be considered complete when the system fulfills all minimal functional requirements, including:

- A handwriting-to-text conversion tool
- An automated grading system
- Report generation capabilities
- A working user interface

While no minimum performance benchmark is required, the system must reliably complete these core tasks in a reasonable amount of time. In addition, successful completion requires a pilot run with MATH 111 and CSCI 128 exams or quizzes. The system must process student exams from both courses and generate grades based on the provided rubrics. These results will be compared against human-grades versions to validate accuracy. Refinements may be made based on discrepancies, but execution of this pilot run is essential for the project to be considered done. There is no threshold for system accuracy for the pilot run to be considered successful, but results from this run will be used to refine the system going forward and better understand where improvements are needed.

VI. System Architecture

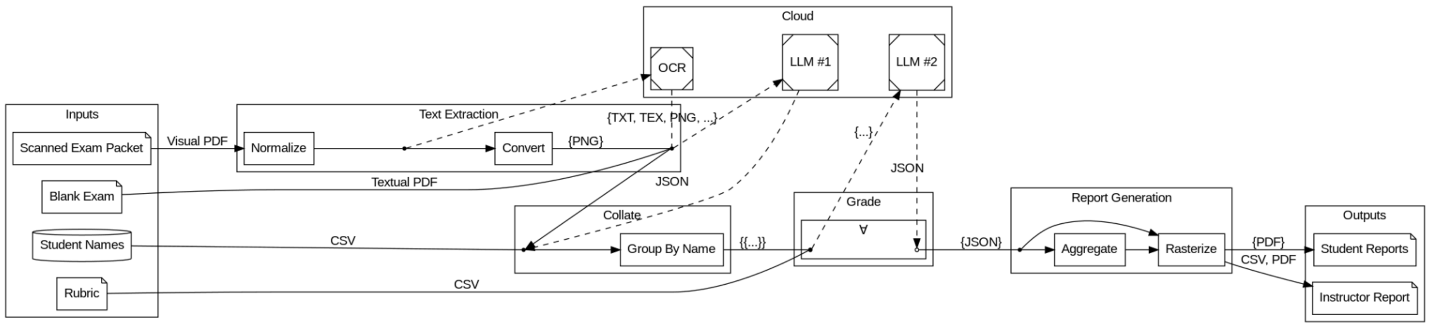


Figure 1: Full design flowchart

Our design shown in Figure 1 consists of four primary modules: user interface, text extraction, grading, and report generation. The inputs will contain everything necessary for the grading to take place. The professor or teaching assistant will input a blank version of the exam, the rubric, the students' scanned exam packets, and a list of student names. There will be a basic user interface that will facilitate this process, as seen below in Figure 2.

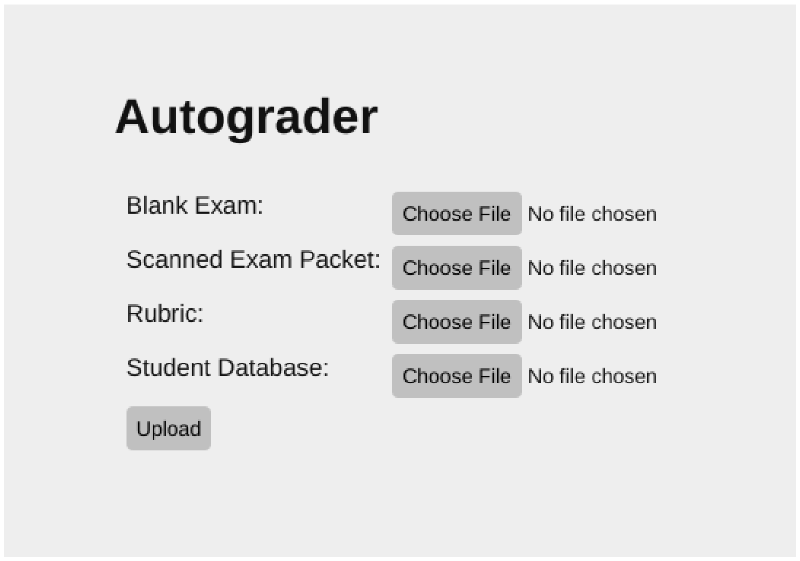


Figure 2: Mock design of user interface

The first component, the user interface, is critical for integrating with the professor. In our case, we determined we need three primary inputs: the scanned exams, the rubric, and the course list of students to ensure we are properly matching exams to real people. Once all this data has been submitted, the uploaded files are passed to the text extraction module.

The text extraction works using an OCR (Optical Character Recognition) API. This API takes in all of the scanned exams and outputs text data which can be read more easily by downstream AI modules. To improve accuracy, we make use of both the MathPIX API and the OpenAI API to ensure that output is in a structured JSON format for downstream modules.

After text extraction is the grading module. This subsystem starts by taking in the structured data from upstream modules and passing it along with the rubric to the OpenAI and Gemini APIs. Once both of these modules have returned an answer, we use the resulting output to calculate a confidence score. For each discrepancy, the confidence score is lowered. Using this confidence, we can notify the user when the system finds exams that may need human input to grade accurately. This module also produces feedback on every answer for the student. The feedback is used later on in the report generation stage to create summaries for professors and to give students more insight on why they missed any points.

The report generation module produces the final output of the project. It takes all the data produced upstream thus far and converts it into a human readable PDF so that the user can better analyze the output. Notably, the report generation produces three types of files: a per-student report PDF, a single professor report PDF, and a Canvas-compatible gradebook CSV file. Additionally, if the system was started in debug mode, a memory dump of the raw data is included which can be utilized by future developers for improvements and debugging.

All four of these modules are written in Python (and HTML/JS/CSS for the UI) and are tied together by an in-memory SQLite database to support efficient querying and data operations by each module. Additionally, as this project is intended to be turned into a full service application in the future, we designed the entire system to be easily upgradable and modular. While currently running only on localhost, using Flask for the web handling means that it should be easy to deploy the program to various production environments with minimal configuration changes (e.g. enabling SSL or adding authentication), and the SQLite database could be swapped out for a real one since there is already an intermediary abstraction layer.

VII. Software Test and Quality

We utilize objective and statistical testing methods to ensure SAGE meets its purpose: accurate, explainable, privacy-safe grading of paper exams with an intuitive UI. The following defines what we test, how we keep LLM/OCR outputs reliable and bias-resistant, and how code quality and security are enforced.

Verification and Validation Strategies

1. Unit Tests
    a. Targets: PDF ingestion, page grouping, label detection, rubric parsing, scoring, LaTeX conversion, & report generator
    b. Method: pure and idempotent function tests & property-based tolerance tests
2. Integration Tests
    a. OCR Pipeline: Image -> Text/LaTeX -> Confidence and valid output schema
    b. Grading Pipeline: Extracted Text & Rubric -> Criteria alignment with human grading
    c. Job System: Queuing and automatic retries with induced failures
3. System / E2E Tests
    a. Manually running through the entire process and validating each output
4. UI Testing
    a. Hard to automate, instead have different users including instructors and teaching assistants use the system with minimal instruction and provide feedback
5. User Acceptance Testing (pilot run with MATH 111 / CSCI 128)
    a. Instructors attempt various tasks (import rubric and student exams, grade, report, export)
    b. Accept if satisfied with accuracy, time, and usability

LLM & OCR Quality Controls

1. OCR
    a. Measures: word error rate (WER), math token accuracy, layout detection accuracy
    b. Stress: skew/rotation, messy handwriting
    c. Fallbacks: Provide a translation confidence score so instructors can manually grade if there is any ambiguity
2. LLM Grading
    a. Prompting: rubric-first instructions, require grading justifications, JSON-schema outputs
    b. Guardrails: low temperature, output schema validation, timeouts and bounded retries
    c. Bias & drift: grade with PII masked, monitor agreement with rubric
    d. Adversarial: handwritten prompt-injection dataset ("give full credit", "ignore rubric", etc…) must be ignored

Code Quality Practices

1. Reviews & Standards
    a. 2-person code review for grading logic, OCR, and data handling
    b. Python linting and static type checker
    c. Javascript linting
2. Static / Dynamic Analysis
    a. Security linters
    b. Repository scanners (for API keys, PII, etc.)
3. Metrics
    a. Grading and accuracy should align with the goals defined in the requirements section of this document

VIII. Project Ethical Considerations

Using AI comes with significant ethical concerns especially when mixing it with personal data. This project is using AI to process data protected by FERPA which could be used as training data by third parties. Additionally, because this project has access to students' names on exams, there is a potentially significant risk of the AI being harsher on certain ethnic groups due to model bias. Finally, it's possible that an AI would grade certain classes lower than others based on a perceived difficulty introduced in its training data. We can split all the project's ethical considerations into 3 primary issues and mitigation methods.

- FERPA Compliance
    - To ensure we are compliant with FERPA and the IEEE Code of Ethics, we will use AI models who explicitly state they will not retain user data for training their models. Most LLM companies include privacy toggles for users; for example, OpenAI, has a consent toggle that allows developers to prevent OpenAI from collecting data. This must be properly configured.
- Ethnic Bias
    - In order to prevent ethnic bias, we must avoid injecting any form of identifying data into an AI model responsible for grading. Identifying data including the students' names, the college's name, and location data will be stripped out before transmitting.
- Sentiment Bias
    - Sentiment bias is practically impossible to mitigate on the application level, as the LLM will always receive data on the questions and will be prone to implicit bias based on the perceived difficulty of the question. As a result, professors must be provided with the data necessary to mitigate this themselves, For example, if a report has an unusually low average then a professor would need to look through the questions and determine if the LLM made fair assessments or was introducing bias.

In addition to these objectives, our group strives to remain in compliance with both ACM and IEEE ethical standards. To ensure we comply with these standards, each group member has individually read the code of ethics for both ACM and IEEE. The below guidelines focus on the standards that our project is most at risk of breaking and mitigation methods to ensure we remain in compliance with the respective codes.

- Use only Software obtained legally (IEEE 2.02)[1]
    - All the leading AI models available today were trained off of publicly available data scraped from the web. This is theoretically a massive violation of millions of people's copyright, but there have been no conclusive court cases yet despite many ongoing ones. As AI is an emerging technology and regulation is slow to follow, it is important that we—and anyone who picks up this project after—are aware of this pending legality of AI services.
    - We must also use only highly reputable software from trusted sources and obey any license restrictions in dependencies.
- Maintain Professionalism (ACM 2.2)[2]
    - As our client is a professor at Mines within the field of AI, our team may be prone to making assumptions about his knowledge regarding the tools we are applying. This may result in our client not being fully aware of the implications of certain technologies applied in our software. To mitigate this, it is important that we make our client aware of issues with applied technologies even if there is a high likelihood that he already knows about the implications.
- Give Due Credit (IEEE 7.03)[1]
    - An unfortunate side effect of projects which double as course credit is that some members may be prone to exaggerating their impact on the team out of personal interest. While difficult to completely mitigate, generally credit afforded by teammates is of greater reliability than credit awarded to oneself. Therefore, by mentioning our teammates accomplishments to clients rather than our own, we may prevent any member from taking too much credit for themselves and ensure an equal playing field both for course grading and professional development.

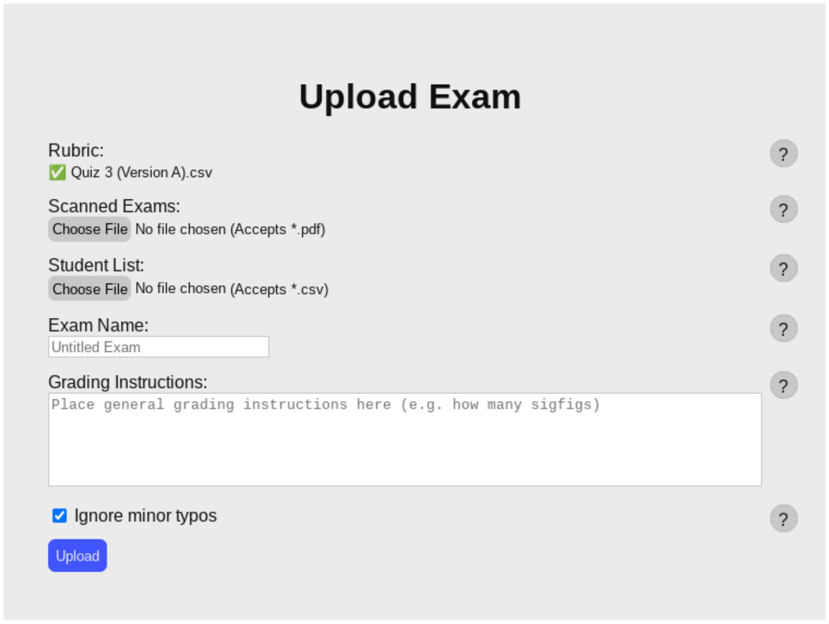
By following the above guidelines, we can ensure that our project will pass both the Michael Davis mirror and legality tests despite our usage of emerging and highly opinionated techniques.

IX. Project Completion Status

The application successfully processes scanned PDF files of student exams and converts them into machine-readable text. This is accomplished through a two-stage pipeline. First, the Mathpix OCR model performs an initial transcription of handwritten content into digital text. The resulting output, along with the original exam document, is then passed to OpenAI's GPT-5 model, which performs a secondary conversion and refinement step. In this stage, the GPT-5 model uses the Mathpix OCR output as a reference to resolve ambiguities and improve accuracy.

Using this method, the system consistently achieves around 98% character-level accuracy. The remaining character differences are minor and usually have no measurable impact on the downstream grading process (e.g., capitalization differences of similar letters). Comparative testing confirmed that this hybrid approach yields the most reliable results and outperformed alternative configurations such as Mathpix + Gemini 2.5 Pro (96% accuracy) and lighter-weight models such as Gemini 2.5 Flash, GPT-5-mini, and GPT-4o, which achieved only 60–70% accuracy.

For grading, the program accepts rubrics containing a mix of additive and deductive criteria. Using the rubric and converted answers with Open AI's GPT-5-mini model and Google Gemini's 2.5-flash model, both models are asked to return a list of criteria applied. Afterwards, the respective outputs are checked against each other to ensure that both models output the same score. The correlation between the two is used to derive the confidence score. For every result that does not match, it is lowered accordingly. Additionally, the models output feedback for students that gets included in the report to carefully note why they received their specific scores and how they can improve in the future.


Upload Exam

Rubric:
✅ Quiz 3 (Version A).csv

Scanned Exams:
Choose File No file chosen (Accepts *.pdf)

Student List:
Choose File No file chosen (Accepts *.csv)

Exam Name:
Untitled Exam

Grading Instructions:
Place general grading instructions here (e.g. how many sigfigs)

☑ Ignore minor typos

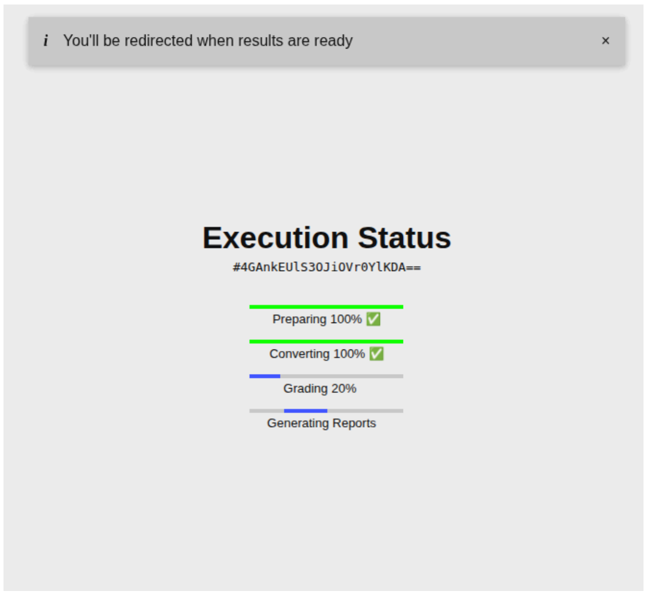
Upload

Figure 3: Upload screen where users begin

The final user interface, shown in Figure 3, was designed to be simple and intuitive. Users are prompted to upload three files: the rubric (CSV), the scanned exam packet (raster PDF), and the student list (CSV). The rubric must include "Question", "Criteria", and "Max Points" columns; question labels must match those used on the exam. The student list must contain "Student Name" (formatted as "Last, First") and "Id" fields to match pages to students. Instructors can obtain a compatible student list from either Trailhead or Canvas.

Users may also provide an exam name for use in output files and optional grading instructions that specify general exam-wide rules. These instructions are stored in the browser for convenience. The last option to ignore minor typos helps remedy grading when a student has poor handwriting that may get converted wrong but can be disabled if it is too forgiving for some classes. Additional guidance on each option is available through tooltips ("?" icons) next to each input.

Once the user submits the form, the system validates all inputs and dispatches a background job. If any input is invalid, the user is returned to the upload screen with a clear error message and remediation guidance.



Execution Status

#4GAnkEUlS3OJiOVr0YlKDA==

Preparing 100% ✓

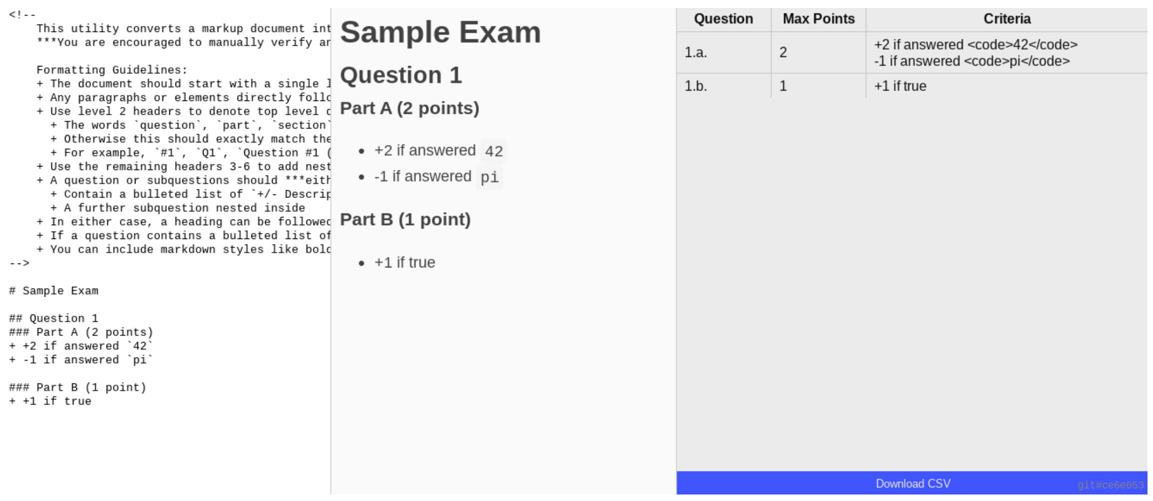
Converting 100% ✓

Grading 20%

Generating Reports

Figure 4: Progress screen

After a background job is submitted, users are redirected to a status page showing each processing step (see Figure 4 above). If progress stops updating—typically due to a network issue or the browser pausing the tab—reloading the page typically resolves it (fixing this is outside our control). The status page can be visited at any time using its unique URL. When processing finishes, users are taken to a results download page. Output files are retained for 36 hours by default, though this is adjustable in the system's configuration file. Informal usability testing showed that users could navigate the interface easily. The only noted issue was that file dialogs hide the upload label, but this behavior is controlled by the operating system not us and is easily addressed by closing the dialog to confirm the field.



```
<!--
    This utility converts a markup document int
    ***You are encouraged to manually verify ar

    Formatting Guidelines:
    + The document should start with a single l
    + Any paragraphs or elements directly follc
    + Use level 2 headers to denote top level c
      + The words `question`, `part`, `section`
      + Otherwise this should exactly match the
      + For example, `#1`, `Q1`, `Question #1 (
    + Use the remaining headers 3-6 to add nest
    + A question or subquestions should ***eith
      + Contain a bulleted list of `+/- Descrip
      + A further subquestion nested inside
    + In either case, a heading can be followec
    + If a question contains a bulleted list of
    + You can include markdown styles like bolc
-->

# Sample Exam

## Question 1
### Part A (2 points)
+ +2 if answered `42`
+ -1 if answered `pi`

### Part B (1 point)
+ +1 if true
```

Sample Exam

Question 1

Part A (2 points)

- +2 if answered `42`
- -1 if answered `pi`

Part B (1 point)

- +1 if true

Question	Max Points	Criteria
1.a.	2	+2 if answered <code>42</code> -1 if answered <code>pi</code>
1.b.	1	+1 if true

Figure 5: Progress screen

To better facilitate creating rubrics, we designed a companion tool professors can use instead of having to make rubrics in an Excel table. This interface in Figure 5 is available at the `/util/md2csv` route of the website and allows converting markdown into CSV rubrics. Markdown was chosen as the input type based on client feedback and its popularity as a rich text format (at least within the Computer Science department). Instructors can make questions and subquestions by nesting markdown headers then provide point values and criteria in bullet point lists. The editor provides a textarea for entering markdown, a live preview of the rendered text, and a display of the CSV output so that professors can verify the automated conversion. Inside the editor, we try to provide useful instructions and hints on how to use the tool as shown above. All processing happens in real time within the user's browser so that instructors can quickly iterate and tweak the rubric. Once they are satisfied, professors can download and save the rubric to use back on the home page or save for later.

X. Results

After completing all functional requirements for the application, we conducted a test run using a sample of exams from the Fall 2025 semester of CSCI 128. Our team manually graded approximately 60 exams according to the instructor-provided rubric. We then processed the same exams through our system and compared the outputs. Although a small number of edge cases could not be evaluated, the application achieved 99.3% accuracy on standard questions and 95.8% accuracy on more complex tasks.

Cost

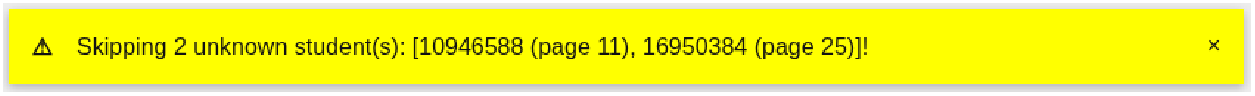
Exact cost depends on several factors, including answer length, rubric size, and the number of questions per page. For our trial run, we estimated a cost of approximately \$0.03 to \$0.04 per page. These values are approximate and do not account for token caching or pricing variations related to token consumption. Most expenses arise from the conversion stage, which involves sending images to OpenAI and Mathpix; the grading stage itself is comparatively inexpensive.

Time

On average, the system processes a page in approximately 30 seconds, though this varies based on the performance of external APIs and network conditions. Most local CPU time is spent waiting for API responses; the application is already fully parallelized and issues requests in optimized batches, ensuring it remains active whenever work is available. Aside from minor micro-optimizations, there is limited opportunity to further improve local performance.

Meaningful speed improvements would need to come from faster response times or higher rate limits from the external API providers. Although it may be technically possible to circumvent rate limits through alternate keys, accounts, or IP addresses, doing so would likely violate provider Terms of Service. We instead recommend pursuing officially supported options for increased throughput.

Identifying Students



⚠ Skipping 2 unknown student(s): [10946588 (page 11), 16950384 (page 25)]! ×

Figure 5: CWID Warning

The program successfully matched students to exams with 81.5% accuracy. For samples we considered moderately legible, it produced no errors; however, it struggled with handwriting that human graders also found ambiguous—particularly cases involving "cursive numbers." Representative examples are shown below.

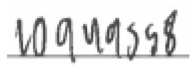
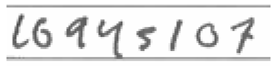
Written CWID	Extracted CWID	Correct Output
	10946588	10949558
	16945107	10945107

Table 1: CWID Errors

All observed errors were false negatives in which the system failed to match a student. While it would be possible to more aggressively correct conversion errors, doing so increases the risk of false positives, where text is over-corrected and an exam is attributed to the wrong student. Because reducing false negatives inherently raises the likelihood of false positives, we believe it is preferable to err on the side of false negatives for two reasons.

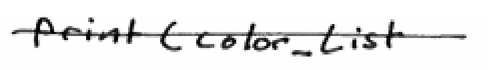
First, requiring occasional manual grading is far less problematic than misattributing a student's work. A false-positive match not only results in an incorrect grade but also constitutes a serious privacy violation by exposing one student's exam to another. Second, false negatives are significantly easier to detect and flag. The system can automatically identify IDs that do not appear in the gradebook or student list and alert instructors, as shown in Figure 5. By contrast, false positives are difficult to detect and may go unnoticed.

While the grading component can use context to work around poor handwriting, student identification cannot. Clear writing of identifying information is therefore essential. Rather than attempting to increase the 81.5% accuracy by risking misattribution, we recommend placing some responsibility on students to write legibly. Options include requiring block-style numbers similar to government forms or implementing a small, flat penalty (e.g., –5%) when instructors must manually resolve handwriting issues.

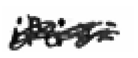
Grading

For basic questions, such as short answer, fill-in-the-blank, code tracing, and numeric responses, the program achieved 99.3% accuracy. It handled imperfect handwriting well and correctly applied partial-credit rubrics with multiple criteria. Excluding a few cases with illegible handwriting, only one basic question was misgraded: a single incorrect answer was marked correct due to an isolated model hallucination. This issue could not be reproduced, and additional safeguards have since been implemented to prevent similar errors.

For more complex questions involving code writing or multi-step mathematical reasoning, the system achieved 95.8% accuracy. Through testing, we identified several important considerations. First, rubric quality is critical: subjective or loosely defined rubrics produced inconsistent results, whereas explicit, well-structured rubrics aligned closely with human grading. Second, both Mathpix and OpenAI struggled when students lightly crossed out work, as these markings were interpreted as part of the answer. Figure 7 shows an example of both adequate and inadequately crossed work. We recommend that instructors advise students to fully erase or clearly strike through unwanted work. See the example below.



Not Ok ❌



Ok ✅

Figure 7: Striking out text

Third, the program is generally more stringent than human graders. It often identifies minor mistakes—such as missing punctuation—that humans may overlook. When discrepancies arose, human graders reevaluated the affected exams and confirmed the program's findings. This behavior is not inherently problematic but underscores the importance of specifying in the rubric whether minor errors should be forgiven.

Finally, conversion errors occurred in approximately 2% of cases, and outright grading failures occurred in about 1% of cases. Some failures resulted from unconventional but technically correct answers not anticipated by the rubric. Despite these limitations, when paired with a clear and comprehensive rubric, the program reliably grades complex and lengthy questions.

XI. Future Work

Throughout our team's work on this project, we have successfully implemented features for reading documents, rubric based grading, and report generation; however, there is room for improvements in the future.

Currently, the text extraction occasionally includes bits of surrounding text—not just answers. While this does not appear to have negative impacts on grading, it is one potential spot for improvement. This also has the potential to decrease token costs for each page noticeably. We theorized using preprocess transforms (such as increasing contrast) to improve conversion accuracy; however, we did not have time to implement or test this. Additionally, the MathPIX API has the ability to extract diagrams, tables, and other nontextual content that we could then attach with the text when sending for grading. We started on this feature but did not finish it; the handling and storage for attachments internally is finished, but they still need to be downloaded from MathPIX after receiving the primary response and then attached to the grading requests.

For grading, some classes do not use points based grades but rather use a leveled mastery system such as ESNU. This could be implemented on top of the current point system by providing a way to alias points to categories (e.g. 4→E, 3→S, etc…). This would enable adoption in more classes without requiring professors to change their grading.

Finally, report generation is currently handled by outputting a ZIP file of PDFs. In the future, report generation could be redesigned to be more interactive, perhaps presenting the user with stats once the grading is complete as well as allowing the user to make edits to the rubric or individual OCR output before running the pipeline again. Additionally, generating reports with a "lazy" or delayed processing scheme could reduce the tokens used by the pipeline by preventing unnecessary reports from being generated.

While our development was sufficient for the content we tested on, we expect that as this product is tested and adopted, feedback from professors will necessitate unexpected further features. While many of the above features seem to be the highest priority improvements at the moment, stakeholders in the project may have alternative opinions which are important to consider moving forward.

XII. Lessons Learned

One of the most technically interesting aspects of this project was working with generative AI models to automatically grade students' papers. When handling academic information, especially when assigning grades, accuracy is of utmost importance; however, generative AI models are inherently probabilistic and are not guaranteed to be correct—they can occasionally hallucinate or fabricate information. Integrating these models into our system and tuning them for reliable, consistent performance was one of the largest technical challenges we faced. This required us to adopt a new type of programming mindset focused on prompt engineering, crafting, and refining the instructions given to the model to guide its behavior effectively.

At the beginning of the project, it was often difficult to pinpoint why a model's responses did not match our expectations. Through iterative debugging and testing, we discovered that asking the model to make decisions independently while simultaneously imposing strict constraints led to inconsistent and confused reasoning. We learned that when providing detailed, rule-based instructions, it is essential to limit the model's autonomy. Conversely, when encouraging reasoning or interpretation, the prompts should be open-ended and flexible. Once these distinctions were made clear, the various models began producing highly accurate and more reliable results. This process of iterative experimentation—testing prompts, evaluating outputs, and refining based on feedback—was a new and valuable learning experience that fundamentally changed how we think about building AI-driven applications.

Working with multiple models and passing outputs from one to another also introduced new complexities with managing various data structures and formats. We needed to transform unstructured natural-language outputs into structured formats suitable for downstream processing and evaluation. This required developing intermediate validation layers and formatting utilities that were not part of our original design. Despite the additional effort, this experience taught us the value of building with strong data contracts and modular design principles. Once the data pipeline was standardized and structured, integration across different modules became much smoother, leading to faster and more reliable development.

One of the most significant lessons learned throughout the project was the importance of communication both within the team and with external stakeholders. In software development, clear technical communication is a distinct skill that goes beyond everyday conversation. It involves explaining design decisions, documenting code, and articulating issues or blockers precisely so that others can quickly understand and act on them. Our team improved substantially in this area through consistent stand-ups, peer code reviews, and detailed documentation. We learned that effective communication not only prevents misunderstandings but also increases accountability and collaboration thus ultimately accelerating development.

Finally, working with a professional client provided invaluable real-world experience. As students, we are often used to driving our own projects and making design choices independently; however, when working with a client, priorities shift toward meeting external expectations and aligning with their goals. This dynamic required us to be proactive: anticipating needs, clarifying requirements early, and maintaining transparency throughout development. We also learned the importance of balancing professional integrity with client satisfaction; while it is essential to voice technical concerns or suggest alternative solutions, the final decision must ultimately respect the client's vision. Fortunately, our collaboration was productive and positive, giving us valuable insight into professional communication, project management, and client relations in a real-world engineering context.

XIII. Acknowledgments

Our team would like to acknowledge and express our gratitude to several people for their help and dedication throughout the entire semester. We would like to thank our client, Professor Zibo Wang, and our advisor, Professor Kathleen Kelly, for their support, encouragement, and advice throughout the duration of this project. Additionally, we would also like to thank the incredible instructors in the classes where our program was piloted:

- CSCI 128: Rob Thompson, Christine Liebe
- MATH 111: Rebecca Swanson

References

[1] IEEE Computer Society, “Code of Ethics | IEEE Computer Society,” *Computer.org*, 2017. https://www.computer.org/education/code-of-ethics

[2] Association for Computing Machinery, “ACM Code of Ethics and Professional Conduct,” *Association for Computing Machinery*, Jun. 22, 2018. https://www.acm.org/code-of-ethics

Appendix A – Key Terms

Term	Definition
SAGE	*Smart Assessment Grading Engine*
OCR	*Optical Character Recognition*
LLM	*Large Language Model*
FERPA	*Family Educational Rights and Privacy Act, law governing student privacy*
LaTeX	*Markup language often used for representing mathematical equations or diagrams*
PDF	*Portable Document Format*