Project Proposal – Team 9

Project Title

Smart Grading Tool: Automated Grading and Virtual TA

Project Overview

The Smart Grading Tool is an AI-powered automated grading assistant designed to streamline the grading process for educators. By utilizing machine learning and natural language processing, the system evaluates student responses against a predefined rubric, providing instant feedback and consistent scoring. In this case, a predefined rubric or an answer key can be used to provide a general scoring structure. An example of what a predefined rubric could look like will be attached below. This tool aims to reduce grading time, ensure objective assessment, and enhance the learning experience for students by offering personalized feedback. Over time, the Smart Grading Tool can expand to provide interactive explanations and learning assistance, functioning as a Virtual Teaching Assistant (Virtual TA).

	Unacceptable	Poor	Good	Excellent
Solution **I.B**	An incomplete solution is implemented on the required platform. It does not compile and/or run.	A completed solution is implemented on the required platform, and uses the compiler specified. It runs, but has logical errors.	A completed solution is tested and runs but does not meet all the specifications and/or work for all test data.	A completed solution runs without errors. It meets all the specifications and works for all test data.
Program Design **I.C**	Few of the selected structures are appropriate. Program elements are not well designed.	Not all of the selected structures are appropriate. Some of the program elements are appropriately designed.	The program design generally uses appropriate structures. Program elements exhibit good design.	The program design uses appropriate structures. The overall program design is appropriate.
User Interface **IV. A**	User interaction is incomplete and does not meet specifications.	User interaction minimally meets the specifications, but does not increase the usability	User interaction generally meets the specifications and is acceptable to the	User interaction is as specified and is natural to the user

Project Scope

Core Functionalities:

- Automated grading of assignments and quizzes based on a provided rubric.
- Consistent and objective scoring for structured responses (e.g., short answers, essays, coding assignments).
- Real-time feedback generation for students.
- Teacher dashboard to review and override grades if necessary.

Additional Features / Stretch Goals:

- Adaptive learning capabilities to provide targeted educational content to students.
- Integration with Learning Management Systems (LMS) like Canvas, Moodle, or Blackboard.
- AI-powered explanation system that teaches students based on common mistakes.

- Support for multiple file formats (DOCX, images for handwritten submissions).
- Speech-to-text grading support for oral assignments.

Project Objectives

- **Improve grading consistency and fairness** through AI-based evaluation.
- **Reduce grading time by 50%**, freeing educators to focus on instruction.
- **Implement an AI model** capable of learning from teacher corrections and improving over time.

Specifications

User Interface (UI) Design

- **Platform:** Web application
- **Key Screens:**
    - Teacher Dashboard: View graded assignments, manually adjust scores.
    - Student Dashboard: View feedback, request clarification.
    - Assignment Submission Page: Upload assignments in PDF format.
    - AI Feedback Review Page: Detailed breakdown of grading and explanations.
- **User Interaction Elements:**
    - Upload buttons, text input fields, interactive feedback tooltips.
    - Filtering and sorting options for assignments.

Backend & APIs

- **Database & API Structure:**
    - PostgreSQL for structured data storage.
    - FastAPI/Django for backend services.
    - RESTful APIs to communicate between the UI, grading engine, and database.
- **Authentication & Security Considerations:**
    - Role-based access control (Teachers, Students, Admins).
    - Secure storage of submitted assignments.

Machine Learning / AI

- **Model Functionality:**
    - NLP-based grading model for text-based responses.
    - Image recognition for handwritten text evaluation.
    - Reinforcement learning to refine grading based on teacher overrides.
- **Data Sources & Preprocessing:**
    - Dataset of graded assignments and rubrics for supervised learning.
    - Automated text preprocessing (tokenization, lemmatization) for NLP tasks.

Tech Stack

- **Frontend:** Next.js
- **Backend:** FastAPI / Django
- **Database:** PostgreSQL / Django
- **AI/ML:** OpenAI GPT / TensorFlow / Open Source Free Alternatives
- **Cloud & Hosting:** AWS / Azure / Google Cloud
- **Security/Auth:** Auth0

Hardware Requirements

- Optional cloud instances for ML model training (if needed)

Software Requirements

- Python for backend and AI development.
- Figma for UI/UX design mockups.

Project Timeline

Phase	Duration	Tasks	Front End	Back End	General	Group Leader
1	Feb 6 - Feb 20	Define scope, research, and setup	UI wireframes, design mockups	Database schema design, API structure	Research datasets, define AI model	Nihanth Attaluri
2	Feb 21 - Mar 15	UI/UX design, data collection, API setup	Build user authentication and dashboards	Implementation of DB design / Develop API endpoints and rubric parser	Collect and preprocess training data	Sriram Sendhil
3	Mar 16 - Apr 10	Development and initial implementation	Implement assignment submission and grading UI	Integrate AI grading engine	Train initial AI models	Sruti Karthikeyan
4	Apr 11 - Apr 25	Testing and integration	User testing and feedback improvements	API security and optimization	Validate AI model accuracy	Vikram Ramachandra

Phase	Duration	Tasks	Front End	Back End	General	Group Leader
5	Apr 26 - May 2	Final testing, deployment, and presentation	Final UI enhancements	Backend deployment and API documentation	Prepare final report and presentation	Saumya Kapoor

Project Team

Role	Team Member	Responsibilities
Full Stack Developer	Vikram Ramachandra	Ex. UI development
Full Stack Developer	Saumya Kapoor	Ex. API & Database
Full Stack Developer	Sriram Sendhil	Ex. AI/ML models
Full Stack Developer	Sruti Karthikeyan	Ex. Testing & validation
Full Stack Developer	Nihanth Attaluri	Ex. Coordination & oversight

Links

- **GitHub Repository:** https://github.com/scs03/SeniorDesignProj
- **Agile Board:** https://www.notion.so/1929f1492cdd808699ced50b09fe393f?v=1929f1492cdd8072b39c000c99246a24&pvs=4
- **Design Document:** https://www.figma.com/design/jzLe6pjlVodDzXtpF7lan1/Untitled?t=9RKnJd8MgXXUHrBp-1

1. Users Table

Stores user information (students & teachers).
Each user is uniquely identified and categorized.

Column Name	Type	Description
`user_id`	`SERIAL PRIMARY KEY`	Unique identifier for each user.
`name`	`VARCHAR(255) NOT NULL`	User's full name.
`email`	`VARCHAR(255) UNIQUE NOT NULL`	User's unique email.
`password_hash`	`TEXT NOT NULL`	Securely stored password (hashed).
`role`	`VARCHAR(50) CHECK (role IN ('teacher', 'student')) NOT NULL`	Defines if the user is a teacher or student.
`created_at`	`TIMESTAMP DEFAULT CURRENT_TIMESTAMP`	Timestamp of user registration.

---

2. Assignments Table

Stores assignment submissions from students.

Column Name	Type	Description
`assignment_id`	`SERIAL PRIMARY KEY`	Unique ID for each assignment.
`student_id`	`INT REFERENCES Users(user_id) ON DELETE CASCADE`	ID of the student submitting the assignment.
`teacher_id`	`INT REFERENCES Users(user_id) ON DELETE SET NULL`	ID of the teacher reviewing the assignment.
`title`	`VARCHAR(255) NOT NULL`	Assignment title.
`submission_file`	`TEXT NOT NULL`	Path to the uploaded PDF file.
`submission_date`	`TIMESTAMP DEFAULT CURRENT_TIMESTAMP`	Timestamp of assignment submission.

Foreign Keys:

- `student_id` → Links to `Users.user_id`.
- `teacher_id` → Links to `Users.user_id` (can be NULL if not yet assigned).

---

3. Grades Table

Stores AI and manually adjusted grading information.

Column Name	Type	Description
`grade_id`	`SERIAL PRIMARY KEY`	Unique ID for each grading record.
`assignment_id`	`INT REFERENCES Assignments(assignment_id) ON DELETE CASCADE`	Assignment being graded.
`ai_score`	`DECIMAL(5,2)`	Score assigned by AI.
`manual_score`	`DECIMAL(5,2)`	Teacher's manually adjusted score.
`final_score`	`DECIMAL(5,2) GENERATED ALWAYS AS (COALESCE(manual_score, ai_score)) STORED`	Stores `manual_score` if available; otherwise, defaults to `ai_score`.
`feedback`	`TEXT`	Feedback from AI or teacher.

Foreign Keys:

- `assignment_id` → Links to `Assignments.assignment_id`.

Functionality:

- AI generates an `ai_score`.
- Teacher can override `manual_score`.
- `final_score` is automatically calculated.

---

4. Feedback Table

Handles student clarification requests on grades.

Column Name	Type	Description
`feedback_id`	`SERIAL PRIMARY KEY`	Unique ID for each feedback request.
`assignment_id`	`INT REFERENCES Assignments(assignment_id) ON DELETE CASCADE`	Assignment in question.
`student_id`	`INT REFERENCES Users(user_id) ON DELETE CASCADE`	Student requesting clarification.
`feedback_text`	`TEXT NOT NULL`	Student's question regarding the grading.
`response_text`	`TEXT`	Teacher's response (if provided).
`created_at`	`TIMESTAMP DEFAULT CURRENT_TIMESTAMP`	Timestamp of request submission.

Foreign Keys:

- `assignment_id` → Links to `Assignments.assignment_id`.
- `student_id` → Links to `Users.user_id`.

Functionality:

- Students can submit clarification requests (`feedback_text`).
- Teachers can respond (`response_text`).

Data Analysis: https://www.kaggle.com/competitions/asap-aes/data

Key Columns:

- **essay_id**: Unique identifier for each essay.
- **essay_set**: Identifies the essay prompt (1-8).
- **essay**: The actual student-written essay.
- **rater1_domain1**, **rater2_domain1**, **rater3_domain1**: Scores assigned by different raters.
- **domain1_score**: The final resolved score for domain 1.
- **rater1_domain2**, **rater2_domain2**, **domain2_score**: Scores for domain 2 (only present in some essay sets).
- **rater_trait scores**: Various trait-based scores for certain essay sets (mostly NaN).

Data Preprocessing

1. **Text Cleaning**:
    - Removed **punctuation** (`.,!?`).
    - Removed **stopwords** (e.g., "the", "and", "is").
    - Converted **text to lowercase** for consistency.
    - Tokenized words using `split()`.
2. **Feature Engineering**:
    - Extracted **word count** (total words in an essay).
    - Extracted **character count** (total letters in an essay).
    - Counted **unique words** in each essay.
    - Computed **lexical diversity**: Lexical Diversity=Unique Words/Total Words
    - Counted **punctuation marks** (as they may indicate fluency and proper sentence structure).
3. **Score Normalization**:
    - Scores were **scaled between 0 and 1** using Min-Max scaling: Normalized Score=Score−Min Score/Max Score−Min Score
    - This ensures that our AI model can learn patterns **independent of absolute score values**.

Why This Was Done

- **Text cleaning** improves the quality of data, making it **more uniform** and removing noise (irrelevant words).
- **Stopword removal** helps the model focus on **meaningful words** instead of common fillers.
- **Feature extraction** allows the model to learn patterns, such as:
    - Do **longer essays** get higher scores?
    - Do essays with **more unique words** indicate better writing quality?
- **Normalization** ensures all values are in the **same range (0-1)**, preventing some values (e.g., raw scores) from **dominating** the learning process.
- **Data saving** ensures that we can efficiently reload preprocessed data **without recomputing features**.

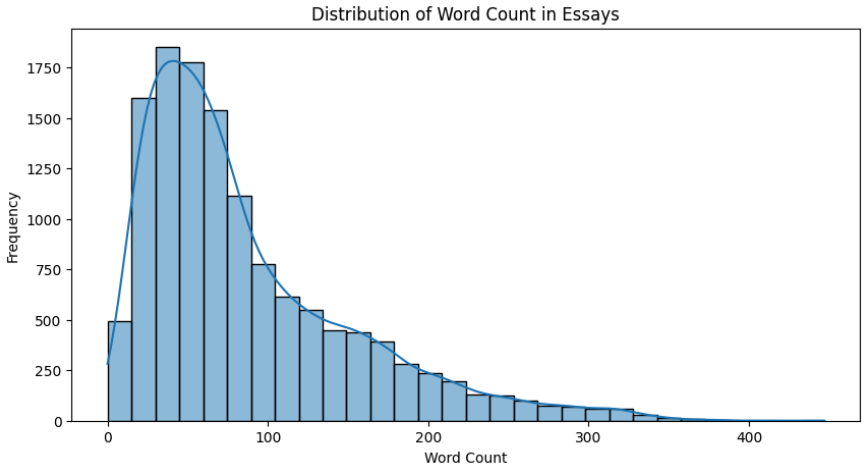
How This Helps the Model

Our project is building an **AI model to auto-grade essays**. The preprocessing and feature extraction **directly impact** how the model learns patterns:

- **Improves model accuracy** by removing irrelevant text (e.g., stopwords).
- **Prepares text for tokenization** (needed for BERT if using transformers).
- **Reduces bias** (e.g., very long essays may not always be better).
- **Helps detect writing quality** through features like **lexical diversity**.

By analyzing the **histograms and scatter plots**, we can **better understand** the dataset before training.

Exploratory Data Analysis



What it shows:

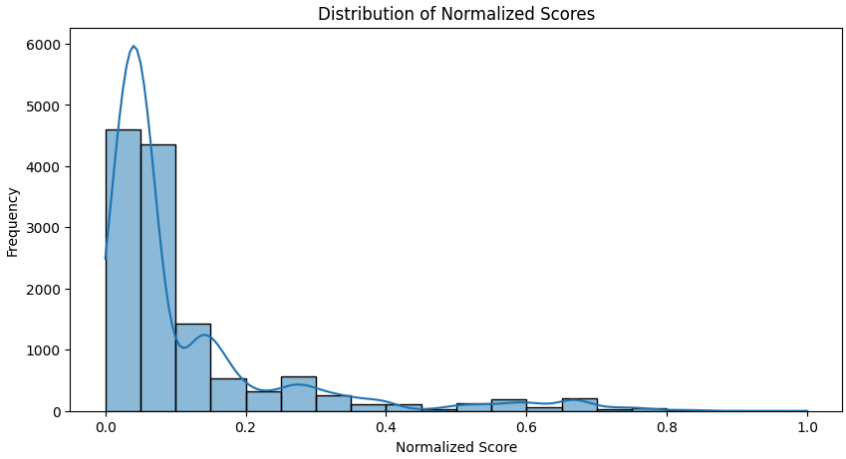
- The number of words in essays across the dataset.
- The most common essay lengths.

What we learn:

- If the histogram is right-skewed, some students write very short essays.
- If the histogram has a peak at a certain range, that is the average essay length.

How it helps:

- If most essays are too short, the model might struggle to learn structure.
- If some are outliers (very long essays), they may need to be handled differently.



What it shows:

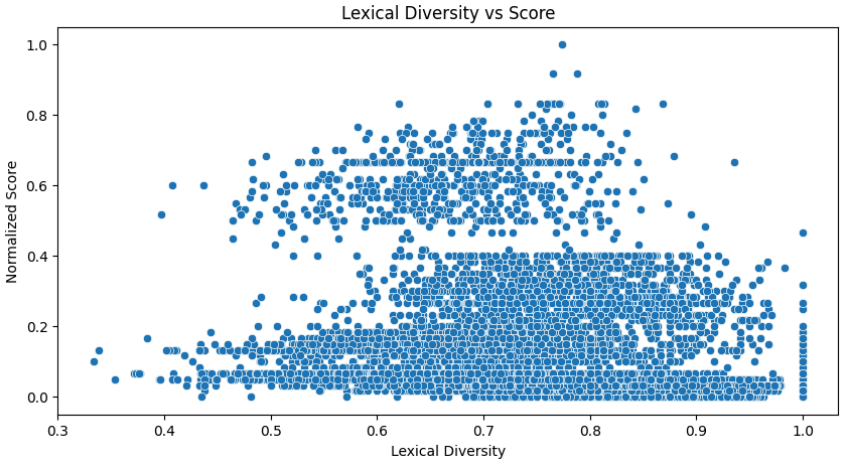
- How essay scores are distributed.

What we learn:

- If the histogram is balanced, scores are evenly spread.
- If it's skewed, there may be more low-scoring or high-scoring essays.

How it helps:

- If most essays have similar scores, the model may struggle to differentiate between them.
- If there's a good spread of scores, the model will generalize better.



- What it shows:
    - Whether using a more diverse vocabulary results in higher scores.
- What we learn:

    - If the plot shows a clear upward trend, students with higher lexical diversity tend to score better.
    - If there is no pattern, vocabulary diversity may not be a strong predictor of score.
- How it helps:
    - If higher diversity = higher scores, the model can use this as a feature.
    - If not, we may need to focus on other features (e.g., grammar, structure).

System Architecture Diagram for AI-Powered Essay Grading System

The architecture follows a **three-layered structure**:

---

1. **Presentation Layer (Frontend)**
    - **Clients:** Web browser (laptop)
    - **Framework:** Next.js
    - **Internet connection** to backend via API requests

---

2. **Application Layer (Backend + AI Processing)**
    - **Backend Framework:** FastAPI / Django
    - **AI Processing:**
        - **DistilBERT**: Essay scoring and evaluation
        - **pdfplumber**: Extracts text from PDFs for essay rubrics
    - **Cloud Hosting:** AWS / Azure / Google Cloud
    - **Security/Auth:** Auth0 for authentication

---

3. **Data Layer (Database + Storage)**
    - **Database:** PostgreSQL / Django ORM
    - **Storage:** Stores essays, scores, user data, grading history
    - **Interacts with AI models** for real-time feedback and scoring

---

Connections & Data Flow:

1. Clients submit essays (if student) or upload a rubric (if teacher) via the frontend.

2. If a rubric is uploaded, the backend processes requests, extracts text (if PDF) using pdfplumber or Tesseract (if image) and sends the data to the AI model.
3. DistilBERT grades the essay based on extracted rubric criteria and returns a score.
4. The backend stores results in PostgreSQL, associating essay scores with the rubric and returning feedback to the client.
5. Auth0 manages user authentication and access control, ensuring only authorized users can upload or access grading data.
6. The system is deployed on cloud services for scalability, enabling efficient handling of multiple grading requests.