**Team 0 : Instant Handwritten-to-LaTeX Converter**

By Xinyang Li (xl137), (sz73), Kexin (kexinhu3) and Hezi Jiang (hezij2)

**Pitch**

Modern day college tech courses (CS, DS, MATH, STAT etc.) all either recommend or request LaTex typing homework and projects. While it could be challenging for inexperienced students and quite time consuming to type-set their works. While there are tools like ChatGpt to help them format their work into LaTex, it lacks instant feedback and is limited by tokens. We want to create a tool to instantly generate LaTex code from handwritten inputs as well as uploaded photos that run on users’ end, which could facilitate the process of math works, and help students learn LaTex.

**Functionality**

* **Handwritten Input Recognition**: Convert handwritten equations and text into LaTeX code.
* **Real-time Feedback**: Instantly display the generated LaTeX output and insert into files.
* **Error Detection & Correction**: Highlight and suggest fixes for syntax errors.
* **LaTeX Code Export**: Allow users to copy or download the generated LaTeX code.
* **Customization Options**: Support different LaTeX formatting styles and structures.
* **Formula Preview**: Render LaTeX equations in a visually formatted output.
* **Handwriting Accuracy Enhancement**: Improve recognition through AI/ML-based corrections.
* **Multi-Device Support**: Work across desktops, tablets, and mobile devices.(optional, if have time)
* **Collaboration Features**: Enable sharing and editing with peers or instructors. (optional, if have time)
* **Educational Support**: Provide tips and documentation for learning LaTeX.(optional, if have time)

# **Components**

#### **Backend (Django & Python & SqlLite)**

* **Backend Testing:** Django’s test framework & pytest. For the AI model, the accuracy should be over 90%.
* **Use Django:** It handles both the databaseand the login process, and is good in performance. It also supports SQLLite.
* **User Authentication**: Handles user login via OAuth using django-allauth.
* **Database Management**: Stores user accounts and past LaTeX conversions in a SQLite database with Django.
* **Handwriting Recognition API**: Processes handwritten input and converts it into LaTeX using AI/ML models.
* **Real-time Communication**: Uses WebSockets or local packages to provide instant LaTeX conversion feedback. (optional)
* **Message Handling**: Supports sending and receiving messages through Discord using their respective Python SDKs.(optional)
* **Error Detection & Correction**: Implements logic to detect and correct LaTeX syntax errors.
* **LaTeX Code Storage**: Saves user-generated LaTeX snippets for future reference.
* **PyTorch for AI**: Use pytorch for training the AI models and to store them offlane and implement them to users end.

#### **Frontend (React)**

* **Frontend Testing:** Using React Testing Library & Jest.
* **Easy to learn:** Simple language and commonly used, as well as good support online.
* **User Interface**: Displays input fields for handwritten notes and LaTeX output. Also supports the uploading of photos.
* **Handle the hand gestures recognition:** Use UIGestureRecognizer by APPLE for developing the IOS app.
* **Camera Integration:** Request for camera access to take photos.
* **Canvas for Handwriting**: A drawing board where users write equations and text.
* **Real-time LaTeX Rendering**: Shows the formatted LaTeX output as users write.
* **Copy & Export Feature**: Allows users to copy LaTeX code or download it as a .tex file.
* **Channel & Message Viewer**: Displays fetched Discord messages within the app.(optional)
* **Error Highlighting Component**: Marks incorrect LaTeX syntax and suggests corrections.
* **Settings & Customization**: Allows users to choose LaTeX formatting styles.
* **Collaboration & Sharing**: Supports sharing LaTeX documents with peers.(optional)

We chose to separate our application into a backend and frontend. Django is well-suited for handling authentication, database management, and API communication, with decent performance for all platforms while React provides a responsive and interactive user interface. By keeping these components distinct, we enable parallel development—allowing frontend developers to build the UI independently of backend logic, which speeds up the development process.

# **Weekly Planning**

| **Week # (Date)** | **Tasks** |
| --- | --- |
| Week 1 (2/24-2/28) | 1. Survey on platform preference and function preference, among TECH students who used LaTex before 2. Research handwriting recognition models (e.g., TensorFlow, PyTorch, OpenCV). And learning basic codes. Make a small demo project for representation. 3. Learn Django and React basics for full-stack development. |
| Week 2 (3/3-3/7) | 1. Finish learning,installing and configuring Django, React, PyTorch, and SQLite. 2. Learning and familiar with all github workflows. 3. Make a basic Django authentication. |
| Week 3 (3/10-3/14) | 1. Collecting and labelling the data for training the handwriting to LaTex model. 2. Try to make the first model trained with at least 60% accuracy. 3. Learn the basics of SQLite |
| (3/17 - 3/21) | Spring Break |
| Week 4 (3/24-3/28) | 1. Design a basic SQLite database to store LaTex snippets user generated. 2. Implement a local LaTex code render for instant feedback. 3. Implement local LaTex editor. |
| Week 5 (3/31-4/4) | 1. Implement embedding features (embedding LaTex with normal word files, and can store back into work or PDF files) 2. Finding and implementing the image upload functionality API and getting camera access. |
| Week 6 (4/7-4/11) | 1. Connect React frontend to Django backend using APIs. And receive the feedback from the backend to the document. 2. Supporting different LaTex formats and standards. |
| Week 7 (4/14-4/18) | 1. LaTex correction suggestion features. 2. Improving the performance of the AI model on the low end phones (make better performance of the app with APIs or online computing options). 3. Supporting copy features and download features. |
| Week 8 (4/21-4/25) | 1. Try to implement the gesture recognition features on iPhones. 2. Backend testing with Django’s test framework & pytest. 3. Front-end testing with Jest & React Testing Library. |
| Week 9 (4/28-5/2) | 1. Debug and refinement on both backend and frontend for better user experience. 2. Improving the model accuracy to 80%. 3. Try to improve speed and performance of live gesture inputs. |
| Week 10 (5/5-5/9) | 1. Continued refinement on performance. 2. Multi-device support 3. Educational resources embedded 4. Prepare for final presentation and writing user documentation |

*Reasoning for the above timeline:*

# **Potential Risks**

* **Handwriting Recognition Accuracy**: The AI/ML model may misinterpret handwritten input, leading to incorrect LaTeX code. Use mature packages and full testing to avoid problems.
* **Performance Issues**: Real-time conversion via WebSockets may cause latency or slow processing for complex equations. If we can not handle the network problems, use local models instead or rent better servers.
* **Scalability Concerns**: SQLite may not be efficient for handling a large user base or high-traffic scenarios. If that happens, we will switch to more powerful database tools.
* **Security Risks**: OAuth authentication via Discord could introduce vulnerabilities if not properly secured. Use the packages that are always up-to-date to avoid security risks.
* **API Dependency**: Relying on third-party APIs (Discord, handwriting recognition) may cause disruptions if they change or go offline. Offer fully offlane more without internet requirements.
* **Frontend Compatibility**: Ensuring smooth user experience across different browsers and devices may be challenging. Use cross-platform developing languages and packages if possible, and check the documentation and forums for problems.
* **User Adoption & Learning Curve**: New users unfamiliar with LaTeX may struggle with generated output despite improvements in accessibility. Also include suggestions for LaTex tutorials.

# 

# **Teamwork**

We will use github to collaborate with each other. We will use Kanban board to track and assign the progress for each week. This could be easily tracked by both students and mentors.

We will work together in each part of the project, instead of developing separately for each part. Thus we can learn more about each step of full-stack development, and will benefit the future job interviews. We know each other so we can ensure that there will be a fair and friendly environment.

**Continuous Integration**

We will use unittest.mock for testing the python codes. We will write test cases for our own part of codes.

We will also consider the edge cases and make sure that the test cases will be passed by at least 80 percent. Travis CI will also be implemented on github to double check the content. Besides, to test the user interface and user experience, we will also try using our apps or websites and try different scenarios, to catch the glitch that can’t be catched by software coding tests.

We will use strict style restrictions like camelCase and line restrictions on function length, to improve debug availability and better for maintenance. Xinyang Li will be monitoring the style requirements enforced.

Our test convergence criteria will be around 80% test passed for the first few weeks, and then we will refine that when we finish the basic development of the app. For the AI model, we will try 90% accuracy rate. We will also try to find better models or increase our train data to improve the model performance.

Xinyang will be responsible for code review, especially when bugs or merge conflicts happen. We may also have meetings discussing the conflicts.