SUGAR LABS

Google Summer of Code 2025

About Me

Hi, I'm T Aswath, a 3rd year B.E. CSE student at CIT with a deep passion for AI, open-source, and competitive programming. My programming journey began in 12th grade, and since then, I've explored diverse technologies, from system design to cloud computing.

Email: aswathscid@gmail.com
Github: t-aswath
Location: India (IST)
First Language: Tamil

Previous Works

Open Source:

Sugar Labs:

1. https://github.com/llaske/sugarizer/pull/1758
2. https://github.com/llaske/sugarizer/pull/1700
3. https://github.com/llaske/sugarizer/issues/1707
4. https://github.com/llaske/sugarizer/issues/1752
5. https://github.com/llaske/sugarizer/issues/1757 (proposed solution)

Libre Office:

1. https://github.com/LibreOffice/core/commits/master/?author=t-aswath (3 Merged PR)

OhMyZsh:

1. https://github.com/ohmyzsh/ohmyzsh/pull/1186

Bitspace:

I co-founded a **student-led open-source organization** during my first year of college, along with my friends. Currently, I serve as the **Vice President**. Our organization conducts **workshops, hosts hackathons, and organizes competitions** focused on open-source development.

Our mission is to educate students about the significance of open source and encourage their active participation in the community.

Recent Activities:

1. **Community Partner** (Local Host: Chennai (CIT)) at **FossHack 2025**
2. Hosted a **booth** at the **INDIA FOSS Conference 2024** (Check out the booth list)

Projects:

I have worked on **4 RAG-based projects**—one during my internship at **Cognizant** and three for the **Microsoft Innovation Challenge**, where one of my projects secured **3rd place(TUSK)** in the competition.

1. https://github.com/t-aswath/TUSK
2. https://github.com/bitspaceorg/trusted-utility-for-statutory-knowledge-act-ii
3. https://github.com/bitspaceorg/smart-process-innovation-network

I am proficient in technologies such as:

- **Python**
- **LangChain**
- **ChromaDB**
- **AWS & Azure**
- **Ollama**
- **Hugging Face**
- **Docker**
- **OpenAI**
- **RAG**
- **JS**
- **TS**
- **REST API**
- **GIT**
- **Prompt Engineering**

These skills are particularly valuable for this project. For more details on my other skills, feel free to check out my **resume** and **GitHub**.

Availability

By the end of April, my end-semester exams will be over, and I will move on to my 4th year.

At my college, there are **no coursework requirements in the final year**, as all subjects are completed in the previous semesters. The 4th year is entirely dedicated to internships.

My college is familiar with the Google Summer of Code (GSoC) program, thanks to past contributors, and provides **full-time support for students to work on GSoC**.

Project Details

Name: Add an AI-assistant to the Write Activity

Description:
Sugar pioneered peer editing in its Write Activity. However, the Write Activity has never had any serious support for grammar correction (just spell check) and none of the more recent developments around AI-assisted writing. The goal of this project is to add AI-assistance to the writing process: both in the form of providing feedback as to what has been written and making suggestions as to what might be written.

Project Length: 350 hours
Difficulty: High
Coding Mentors: Walter Bender, Ibiam Chihurumnaya

Solution

Project Overview

The goal of this project is to develop a real-time AI-powered writing assistant for the **Write** activity in Sugar. This assistant will help children improve their grammar by not only identifying and correcting mistakes but also explaining why a correction is needed and how it improves their writing.

To achieve this, the system will extract text from the **Write** activity, process it using a language model (LLM), and return the corrected text along with detailed explanations. These insights will be presented in a simple, intuitive, and engaging user interface, ensuring a seamless learning experience. The focus is on enhancing children's writing skills in an interactive and educational manner, making grammar correction a learning opportunity rather than just an automated fix.

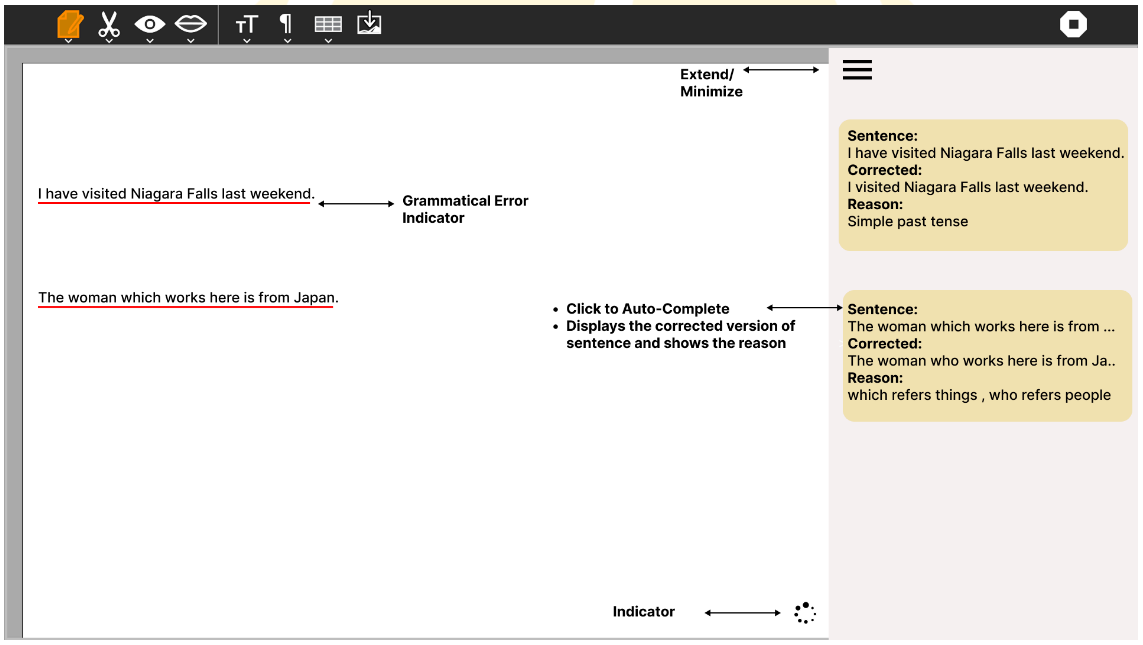
Deliverables

- **Two Grammar Checking Modes:**
  - **Co-Pilot Mode** – Provides real-time grammar suggestions as the child writes, acting as a supportive writing assistant.
  - **Test Mode** – Allows children to write freely and receive corrections only after completing their text, encouraging independent learning.
- **Intuitive User Interface:**
  - A user-friendly UI that clearly highlights corrections, making it easy for children to understand and learn from their mistakes.
- **Grammar Checker Status Indicator:**
  - A visual indicator that displays the real-time status of the grammar checker, ensuring users are aware of when corrections are being processed.
- **Auto-Complete for Corrections:**
  - An intelligent auto-complete feature that seamlessly integrates suggested corrections into the text, helping children learn proper grammar effortlessly.
- **Suggestions**
  - The grammar suggestions will be designed to be **clear, simple, and easily understandable**, ensuring that even a **three-year-old** can grasp the corrections effortlessly. The explanations will be concise and intuitive, making learning engaging and accessible for young children.

Design

I am proposing only the layout of the interface, not the specific style or color scheme, as these aspects are best discussed with the UI/UX team to ensure alignment with Sugar Labs' design standards. I have outlined two possible layout options, but we can proceed with either of these or adopt an entirely different layout if Sugar Labs already has a preferred design in mind. I am fully flexible and open to implementing any design that best fits the project's needs.

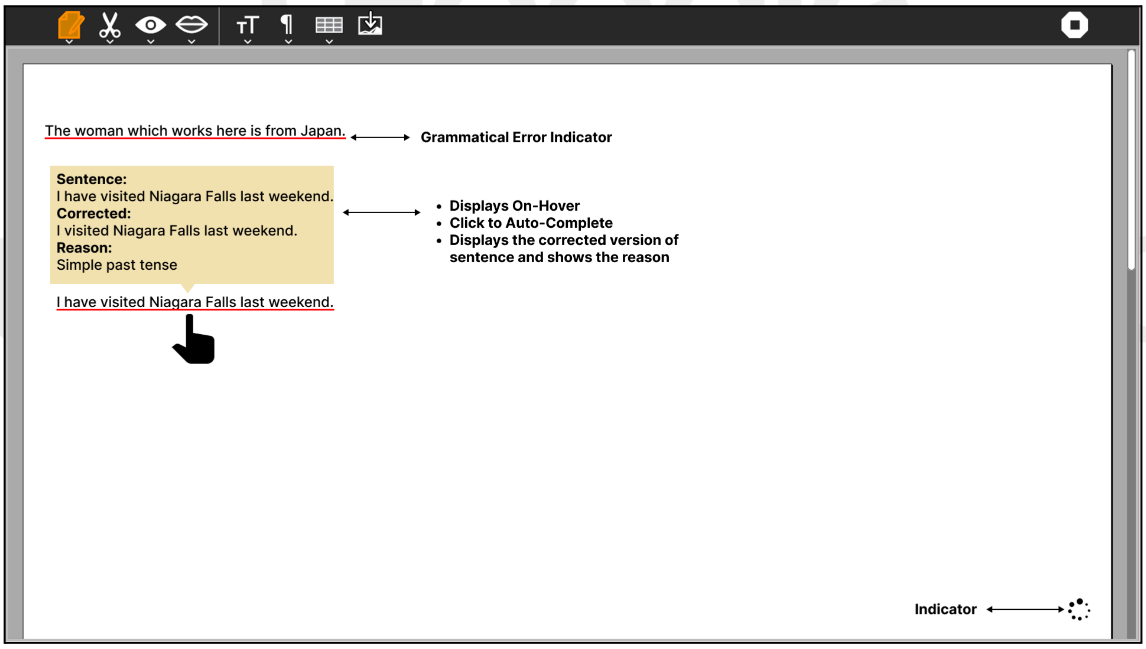
Design - 1


Extend/ Minimize
I have visited Niagara Falls last weekend.
Grammatical Error Indicator
The woman which works here is from Japan.
• Click to Auto-Complete
• Displays the corrected version of sentence and shows the reason
Indicator

Sentence:
I have visited Niagara Falls last weekend.
Corrected:
I visited Niagara Falls last weekend.
Reason:
Simple past tense

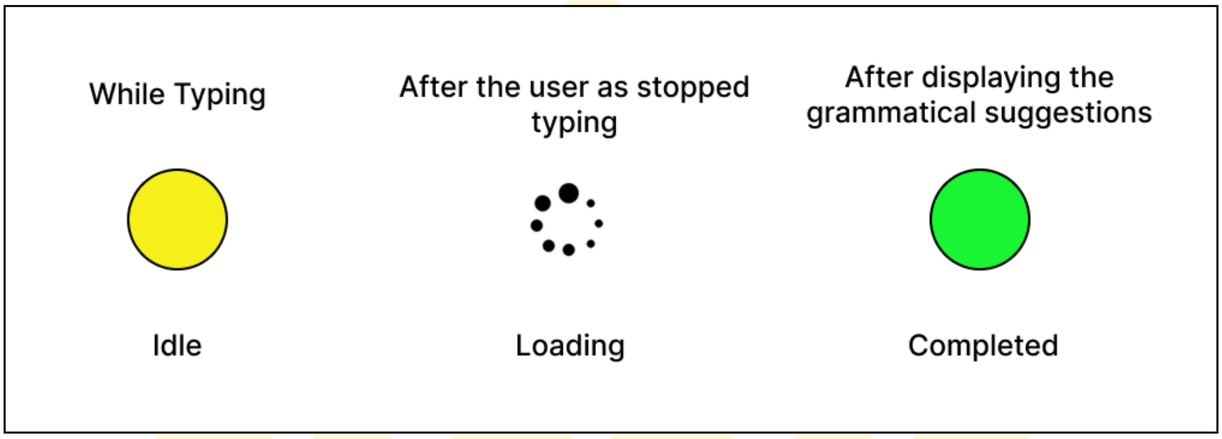
Sentence:
The woman which works here is from ...
Corrected:
The woman who works here is from Ja..
Reason:
which refers things , who refers people

Design - 2

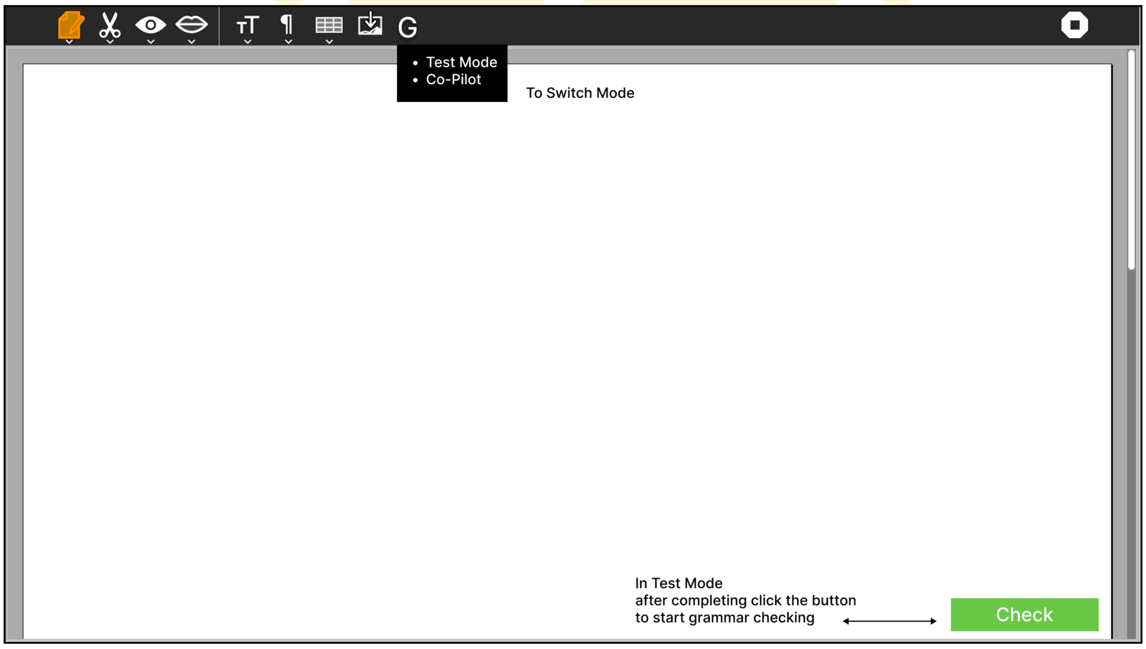

The woman which works here is from Japan.
Grammatical Error Indicator
• Displays On-Hover
• Click to Auto-Complete
• Displays the corrected version of sentence and shows the reason
I have visited Niagara Falls last weekend.
Indicator

Sentence:
I have visited Niagara Falls last weekend.
Corrected:
I visited Niagara Falls last weekend.
Reason:
Simple past tense

Indicator


While Typing
After the user as stopped typing
After displaying the grammatical suggestions
Idle
Loading
Completed

Test Mode



Tech Stack

- **Python**
- **Langchain** (Prompt Template, Output Parser)
- **Fast API** (Make calls to LLM)
- **Ollama** (to pull OS Models)
- **AWS** (to Host the Model)
- **Docker** (to containerize the AI Model, Creating a Microservice)

Implementation

UI

The UI will be built using standard components from the **Write** activity, ensuring consistency and seamless integration. We will utilize `sugar3.graphics`, `sugar3.activity` and `widgets` to implement the proposed design efficiently.

A **status indicator** will be placed in the UI to provide real-time feedback on the grammar checker's operation.

For all suggested corrections, **color-coded indicators** will be used to differentiate the severity of mistakes:
🔴 **Red** – Major grammatical errors
🟡 **Yellow** – Minor issues or style improvements

The UI will be designed to be **fully responsive**, simple, and visually appealing—making it intuitive and accessible even for young children (as young as 3 years old).

AI

For the AI component, we will be leveraging Sugar Labs' AI module, `sugar-ai`. The first step will involve testing various LLMs to identify the one that performs best in **grammar correction** and **reasoning**. After benchmarking, we will create a processing chain using the `LangChain` library to handle the task efficiently.

The **LLM processing chain** will consist of three key stages:

1. **Prompt Generation:**
   - The extracted text will be processed to construct a well-structured prompt using `PromptTemplate` from `LangChain`.
2. **LLM Processing:**
   - The prompt will be passed to the selected model using `LLMChain`, ensuring accurate grammar corrections and explanations.
3. **Output Parsing:**
   - The model's response will be parsed and formatted using `OutputParser` to extract the corrected text and reasoning effectively.

Once processed, the corrected data will be sent back to the **Write** activity in `JSON` format.

To ensure scalability and ease of deployment, the entire module will be **containerized using Docker**, making it seamless to deploy on `AWS`. Additionally, this module will be designed as a **standalone service**, ensuring it is **not tightly coupled** to the **Write** activity. This approach allows for **reusability** across different parts of Sugar, enhancing modularity and reducing code dependencies.

AI - Backend

Once the AI functionality is implemented, it will be **exposed via FASTAPI**, allowing seamless interaction with the **Write** activity and other potential integrations. A dedicated **POST route** will be created, enabling requests to be sent with text input and returning the corrected response in `JSON` format.

This API will be **containerized within Docker**, ensuring a **controlled and scalable deployment environment**. The only way to interact with the AI module will be through the **FASTAPI endpoints**, maintaining **modularity and security**.

The codebase will strictly adhere to **SOLID principles**, ensuring maintainability and extensibility. By keeping the architecture **open for extension**, additional APIs can be easily developed within the same codebase, making it effortless to integrate with other Sugar activities in the future.

Activity

The **Write** activity will be responsible for **listening** to user interactions, making **API calls** at the appropriate moments, and **reflecting** the AI-generated corrections back into the UI.

Listening for User Input

An **event listener** will be bound to the text area to detect user activity. When the user stops typing for a brief period (indicating idleness), the event will trigger, and the current text will be extracted for processing.

Making API Calls & Handling Responses

Once triggered, an **API call** will be made to the AI backend, sending the extracted text for grammar correction. Upon receiving the response in `JSON` format, the system will:

- **Parse the response** to identify corrected text and explanations.
- **Apply styling** to highlight grammatical mistakes directly within the text area.
- **Display additional insights** in the UI based on the proposed design.

Users will have full control to interact with the suggestions, view explanations, and utilize the **auto-complete feature** to seamlessly apply corrections, ensuring a smooth and intuitive learning experience.

Logger (optional)

Since we are developing a **general-purpose AI module**, implementing a **logger** will be essential for tracking API activity and monitoring which **Sugar activities** are utilizing the AI services.

A robust logging system will be integrated into **FASTAPI**, recording key details such as:

- **API requests** made to the AI module.
- **Activities interacting** with the AI.
- **Response times and errors**, ensuring smooth performance monitoring.

This logging mechanism will enhance **transparency, debugging, and optimization**, providing valuable insights into API usage while maintaining efficiency and reliability.

Tests

To ensure the reliability and robustness of the AI module and its integration with the **Write** activity, comprehensive testing will be conducted at multiple levels.

1. Unit Testing

Each core component of the AI module will be tested individually to verify its correctness and performance. This includes:

- **Prompt generation testing** – Ensuring the correct structure of prompts using `pytest`.
- **LLM response validation** – Testing to check whether LLM follows the instructions in the Promt
- **Output parsing** – Validating the JSON response format and correctness of extracted corrections.

2. API Testing

The **`FASTAPI`** endpoints will be thoroughly tested using tools like `pytest` and `HTTPX` to ensure:

- Correct request-response handling.
- Proper error handling and response formats.
- Performance under different loads.

3. Integration Testing

Integration tests will verify seamless communication between the **Write** activity and the AI module by:

- Testing API calls from the **Write** activity.
- Checking response handling and UI updates.
- Ensuring corrections are applied correctly within the activity.

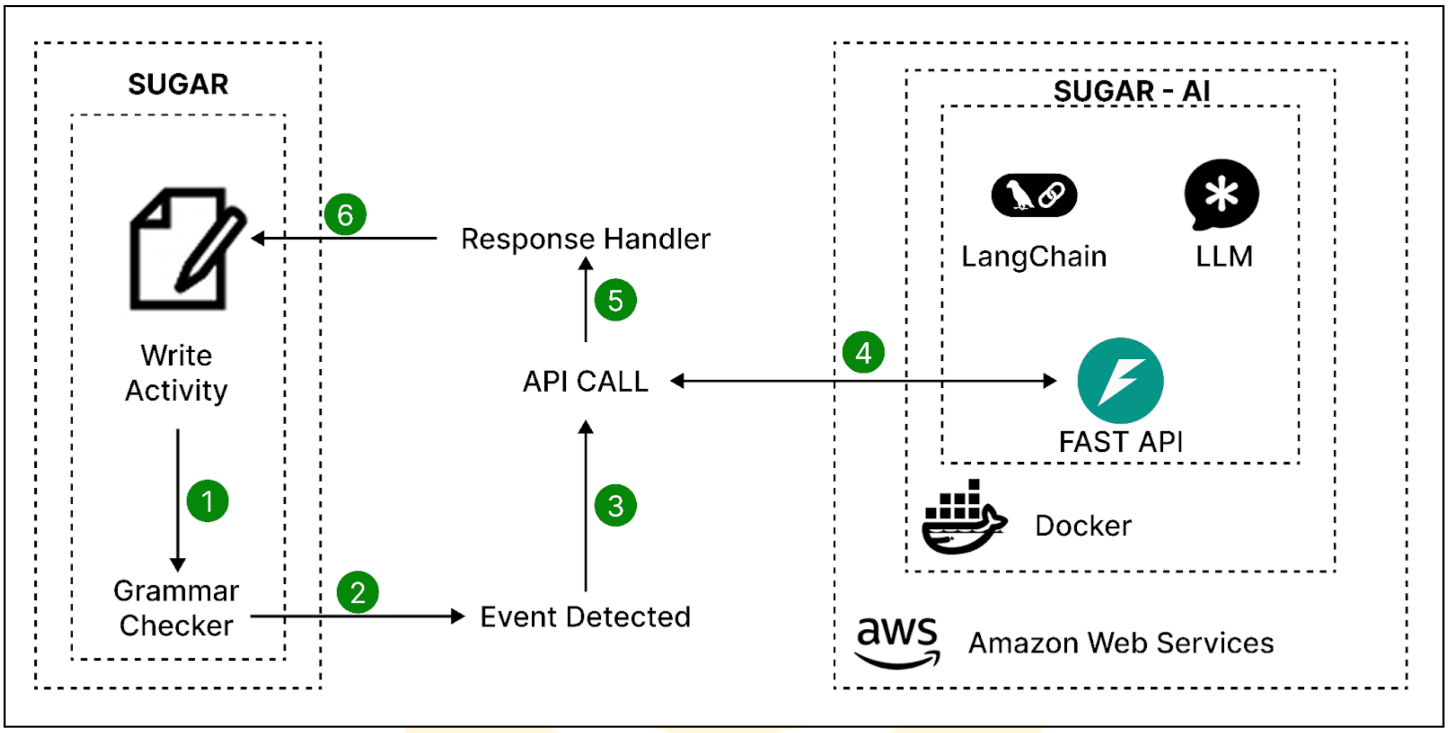
4. UI Testing

Since the UI will integrate AI-generated corrections, testing will focus on:

- Ensuring corrected text and explanations are displayed properly.
- Verifying user interactions with suggestions and auto-complete.
- Cross-device responsiveness testing.

By combining **unit, API, integration, and UI testing**, the project will maintain high reliability, ensuring smooth functionality and a seamless user experience.

Architecture



Prototype

A small-scale implementation of the project demonstrating the core AI functionality. This prototype will showcase the grammar correction process, including detecting errors, providing corrected sentences, and explaining the reasoning behind the corrections. It will serve as a proof of concept, ensuring the AI module functions effectively before full-scale integration.

```python
from operator import itemgetter
from langchain.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
from gradio_client import Client


class Model:
    def __init__(self):
        # Random free model from hugging face
        self.client = Client("yuntian-deng/ChatGPT")
    def predict(self, input):
        result = self.client.predict(
            inputs=input,
            top_p=1,
            temperature=0.5,
            chat_counter=0,
            chatbot=[],
            api_name="/predict",
        )
        return str(result)


template = """
Given the following text:
\n ------ \n
{input_text}
\n ------ \n
Pick sentences which are not grammatically correct.
and provide the correct version of the sentence.
in the following format:
\n ------ \n
sentence: (sentence)
correct_sentence: (correct_sentence)
reason: (reason)
"""

if __name__ == "__main__":

    text = "She go to the park and play with her friends every evening"

    model = Model()
    prompt = ChatPromptTemplate.from_template(template)

    chain = (
        {"input_text": itemgetter("input_text")}
        | prompt
        | (lambda x: x.to_string())
        | model.predict
        | StrOutputParser()
    )

    result = chain.invoke({"input_text": text})
    print(result)


# Output i got from the model

"""
[
    [
        [
            "Human:
            Given the following text:

            ------

            She go to the park and play with her friends every evening

            ------

            Pick sentences which are not grammatically correct,
            and provide the correct version of the sentence
            in the following format:

            ------

            sentence: (sentence)
            correct_sentence: (correct_sentence)
            reason: (reason)"
        ],
        "------

        sentence: She go to the park and play with her friends every evening
        correct_sentence: She goes to the park and plays with her friends every evening
        reason: The subject 'She' requires the verb 'go' to be in the third-person
singular form, which is 'goes,'
        and 'play' should also be in the third-person singular form, which is 'plays.'

        ------ "
    ]
],
1,
<Response [200]>,
{"interactive": True, "__type__": "update"}
"""
```

Challenges & Solutions

1. Redundant API Calls

One major challenge is avoiding **unnecessary API requests**, which could overload the backend and reduce efficiency.

✅ **Solution:**

- Implement a **debouncing mechanism**, ensuring that an API call is only made **0.5 seconds** after the user stops typing.
- The delay can be fine-tuned during the **testing phase** for optimal responsiveness and performance.

2. Interrupting Ongoing Computations

If multiple API calls are triggered rapidly, the backend may process outdated requests, leading to **wasted computation and latency**.

✅ **Solution:**

- Implement an **interception mechanism** that detects if a new request is made before the previous one is completed.
- If a new request is received, **the backend will cancel the ongoing computation**, preventing unnecessary processing and improving efficiency.

By optimizing API requests and backend processing, the system will ensure **faster response times**, **reduced resource usage**, and a **smoother user experience**.

Impact

1. Enhancing Learning for Children

This project will **transform** the way children interact with the **Write** activity by providing real-time **grammar correction and explanations**. Instead of simply fixing errors, it will:

- **Encourage learning** by explaining why a correction is needed.
- **Improve writing skills** through interactive suggestions.
- **Boost confidence** in language proficiency by offering a supportive AI-powered assistant.

By making grammar correction a **learning experience**, children will develop stronger writing skills in a fun and engaging way.

2. Strengthening Sugar Labs' Educational Tools

This project aligns with Sugar Labs' mission of **enhancing digital learning** by:

- Expanding the capabilities of the **Write** activity.
- Introducing a **modular AI component** that can be reused across other activities.
- Improving accessibility and inclusivity by helping children **of all literacy levels** enhance their writing skills.

3. Advancing Open-Source AI in Education

By integrating **AI-powered language processing** into Sugar Labs, this project will:

- Contribute a **scalable, reusable AI module** to the open-source community.
- Demonstrate how **AI can be used for education** in a way that prioritizes learning over automation.
- Set the foundation for future AI-powered educational tools within Sugar.

By bridging the gap between **AI and education**, this project will have a **lasting impact** on how children learn and interact with digital tools within Sugar Labs.

Post-GSoC Plans & Future of the Project

1. Long-Term Sustainability & Maintenance

After GSoC, I plan to ensure the project remains **maintainable, scalable, and adaptable** by:

- **Actively contributing** to the Sugar Labs community to refine and improve the AI module.
- **Providing thorough documentation** to make it easier for future developers to enhance the system.

2. Future Enhancements & Improvements

While the current implementation focuses on **real-time grammar correction and learning**, there is immense potential to expand the project further:

1. **Multilingual Support**
- Expanding beyond English to assist children in learning and improving multiple languages.
- Incorporating translation and grammar correction for various linguistic backgrounds.

2. **Adaptive Learning**
- Implementing an **AI-driven personalized learning** experience that adjusts suggestions based on a child's writing habits.
- Introducing **difficulty levels** to help students progressively improve their skills.

3. My Continued Contribution

I am committed to continuing my involvement with Sugar Labs even after GSoC by:

- **Helping onboard new contributors** to maintain and expand the AI module.
- **Optimizing the AI model** by experimenting with new LLMs and fine-tuning for better accuracy.

This project is just the **beginning of AI-powered learning** within Sugar Labs. With continuous enhancements, it has the potential to become a **core educational tool** that empowers children worldwide to improve their writing skills in a fun and interactive way.

Timeline

Community Bonding Period

- Get familiar with the **Sugar Labs ecosystem**, its codebase, and development workflows.
- Engage with **mentors, contributors**.
- Finalize the technical design and discuss any modifications with the community.
- Set up the development environment and define test cases for grammar correction.

Coding Phase

📌 Week 1-2: Initial AI Module Setup & Experimentation

- Explore and benchmark various **LLMs** for grammar correction.
- Integrate **LangChain** and structure the AI pipeline:
    - `Prompt creation` → `LLM inference` → `Response parsing`.
- Implement a **basic API using FASTAPI** for grammar correction.
- Conduct **unit testing** to validate LLM responses.

📌 Week 3-4: API Development & Dockerization

- Finalize the **best-performing LLM** for grammar correction.
- Implement a **robust FASTAPI route** to handle text correction requests.
- **Dockerize** the AI module for easy deployment and integration.
- Set up **logging mechanisms** to track API usage and interactions.
- Conduct **initial API testing** to ensure stability.

📌 Week 5: Midterm Evaluation & UI Integration Begins

- Submit work for **midterm evaluation**.
- Start integrating the **Write** activity with the AI module via API calls.
- Implement **event listeners** in the text area to capture user activity.
- Develop a **debouncing mechanism** to prevent redundant API calls.

📌 **Week 6-7: UI Improvements & Real-Time Feedback**

- Apply **styles and highlights** to mark grammar mistakes.
- Display **suggestions and reasoning** within the Write activity.
- Implement **interactive elements** for user engagement (e.g., tooltips, pop-ups).
- Conduct **integration testing** between the Write activity and AI module.

📌 **Week 8: Advanced Features & Performance Optimization**

- Implement an **interception mechanism** to cancel redundant backend computations.
- Optimize **response times** and improve model efficiency.
- Refine the **auto-complete feature** to suggest grammatically correct text.
- Test **UI responsiveness** across different screen sizes.

📌 **Week 9: Extensive Testing & Bug Fixes**

- Perform **end-to-end testing** covering unit, API, and UI functionality.
- Gather feedback from **mentors and the community** for final improvements.
- Fix bugs, optimize performance, and polish the overall experience.

📌 **Week 10: Final Refinements & Documentation**

- Finalize all **code, documentation, and tutorials**.
- Submit the project for **final evaluation**.
- Write a **blog post/demo** showcasing the project's impact.
- Discuss **post-GSoC plans** with the community for future improvements.

Time Commitment & Progress Reporting

Time Allocation

- **3-5 hours per day**.
- **Avg 30 hours a week.**

Progress Reporting

1. **Regular Meetings**
   - Open to any meeting platform: **Google Meet, Zoom or others**.
   - Weekly check-ins with mentors to discuss progress, challenges, and next steps.
2. **Private Communication**
   - Comfortable using **Matrix, Discord, Email, or any preferred platform** for quick discussions.
   - Can provide updates and ask for feedback.

3. **Public Updates**
    - **Weekly blog posts** summarizing progress, technical learnings, and challenges faced.
    - Regular updates in **Sugar Labs community channels** to keep everyone informed.

Research

Understanding Grammarly's Approach

To gain insights into how Grammarly achieves **effective grammar correction with minimal input lag**, I studied the following resources:

- ACL Anthology: Automated Grammar Error Correction
- Grammarly's Engineering Blog: Reducing Text Input Lag
- How Grammarly Works
- Grammarly's NLP Approach for Run-on Sentences
- How Grammarly Uses AI

Building a Custom Grammar Correction Tool

To develop our own AI-driven grammar correction tool, I researched methodologies and best practices:

- How to Build a Grammar Checker Like Grammarly
- Creating a Custom AI-Based Grammar Checker

Existing Grammar Correction Tools & Comparisons

To evaluate different approaches and understand their strengths, I reviewed various existing tools:

- Ginger Software
- Zoho Writer Grammar Checker
- GrammarCheck
- TutorBin Grammar Checker

Relevant Past GSoC Projects from Sugar Labs

Studying past GSoC projects provided valuable insights into Sugar Labs' development standards and best practices:

- AI Chatbot Integration for Chat Activity
- Pippy Activity Enhancements
- Sugar Labs GSoC Archive

Conclusion

I am confident in my ability to successfully execute this project and deliver a **high-quality, AI-powered grammar correction tool** for Sugar Labs. With my experience in **AI, LangChain, API development, and system architecture**, I have the **technical expertise** required to build an efficient and scalable solution.

Beyond technical skills, I have a strong background in **open-source development and community-driven projects**, ensuring that my work aligns with **Sugar Labs' values and standards**. My commitment to **clean, maintainable, and well-documented code** will make this project not only impactful in the short term but also **sustainable for future contributors**.

I am highly **dedicated, adaptable, and open to feedback**, ensuring **smooth collaboration** with mentors and the community. My structured approach, **clear milestones, and regular progress updates** will ensure timely completion of the project while maintaining high quality.

By integrating **real-time AI-assisted grammar correction**, this project will significantly enhance the **learning experience for children**, empowering them to **write confidently and improve their language skills** in an interactive way. I am excited about this opportunity to contribute to Sugar Labs and make a lasting impact on education through AI.