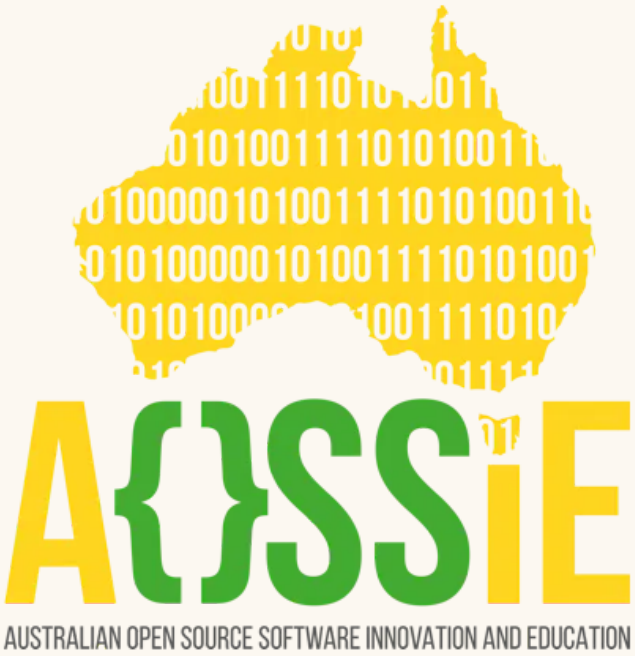


GSoC 2024 Proposal

Edu-Aid

Aditya Dubey

Personal Information

Name: Aditya Dubey

Email: adityavinay@iitbhilai.ac.in

University: Indian Institute of Technology, Bhilai

Field of Study: B.Tech. in Electrical Engineering with specialization in CSE

Year: Junior / Pre-final

Expected Graduation date: July 2025

Github: https://github.com/Aditya062003

Timezone: Indian Standard Time (UTC +5:30)

Commitments

- **How many hours will you work per week on your GSoC project?**
  - Following the standard GSoC guidelines, I commit to work about 5 to 6 hours daily, totaling around **35 to 40 hours per week.** However, if exorable, time is not a bound and my primary goal will be to implement the promised feature by the as specified deadline, even if it means to work out of the specified **350 hours.**

- **Do you have any other commitments during the GSoC period?**
  - As of now, I do not have any commitments in the specified time.

- **Do you plan to apply for any other organization for GSoC'24?**
  - I have not applied to any other organization in GSoC'24.

- **If you're not selected as a GSoC student, would you like to work on the projects as a general contributor?**
  - Under the circumstances that I end up not getting selected, I still would be willing to work as a general contributor.

- **Would you like to contribute to Aossie in the long term, after the GSoC program ends?**
  - As a firm believer in maintaining projects I have previously worked on and constantly improving them to keep them at pace, I would be more than happy to contribute and maintain the project after GSoC and also remain a constant help for future contributors be in terms of setting up the project, helping in channeling out future potential features, or any possible manner. In this particular project, my

ideals align with that of Aossie's vision, its implementation making a positive impact on people's lives, thereby making me content.

Introduction to the Project

At present, Edu-Aid is a locally-run browser extension that generates question-and-answer(QA) pairs from given content. It can handle both plain text and PDF uploads and supports multiple languages including German, French, Romanian, and English. The generated QA pairs are then saved locally (localStorage). The generated QA pairs are viewable within the extension itself and can be exported for later reference in a .txt format.

Current Working Pipeline

The current QA generation pipeline utilizes a two-stage approach:

1. **Keyphrase Detection:** The extension first employs a fine-tuned T5 model to identify keyphrases within the uploaded content. This model is specifically trained using the KP20K dataset, a collection designed for keyphrase extraction tasks.
2. **Answer-Aware Question Generation:** Following keyphrase identification, a second T5-based model, also fine-tuned, generates questions based on the context and the identified answer. This approach ensures the generated questions are relevant and directly target the key points within the study material.

Scopes of Improvement

1. EduAid currently focuses on short answer questions. We can include other formats like **multiple choice, true/false, and fill-in-the-blanks** to cater to different learning styles.
2. We can provide the user with an option to choose the **number of questions** and the **difficulty level** they desire.
3. We can connect Edu-Aid with platforms like **Google Forms**, so users can create quizzes with just a few clicks.
4. Currently, Edu-Aid uses HTML, CSS, and JavaScript for its interface. We can switch to **ReactJS**, which will make the code more organized and easier to update in the future.
5. We can add audio input, potentially even integrating with **Google Meet** to directly use the audio from study sessions as content for Q&A generation.
6. Users can be offered the option to generate different question formats by using their ChatGPT and Gemini API keys.
7. We can create a **web app for EduAid** alongside the extension. This will add more features and make it accessible to even more users.
8. Putting the extension on the Chrome Web Store will make it easier for more people to use, grow its user base, and make quiz creation simpler.

Background and Prerequisites

Difficulty-Controllable Question Generation **(DCQG)** necessitates understanding complex relationships between words, often distant in a sentence. This is where Transformers shine. Unlike Recurrent Neural Networks (RNNs) that struggle with long sequences, Transformers excel at capturing long-range dependencies thanks to self-attention. This mechanism allows them to analyze the entire input sequence, making them ideal for DCQG tasks.
Difficulty-Controllable Question Generation is a relatively new task. Gao et al. classified questions as easy or hard according to whether they could be correctly answered by a BERT-based QA model, and controlled the question difficulty by modifying the hidden states before decoding. Some manipulated the generation process by incorporating the difficulty level into the input embedding of the Transformer-based decoder. In latest work, question difficulty is based on the number of its **reasoning hops**, which is more explainable. Many recent works attempted to conquer this task with **graph-based neural architectures.**

My Primary Goals

- Adding support for various question formats. ***(Mile-Stone 3)***
- Enhancing the UI with ReactJS. ***(Mile-Stone 1)***
- Offering users the option to select the number and difficulty level of questions they desire. ***(Mile-Stone 2)***
- Integrating generated Q&A pairs with Google Forms. ***(Mile-Stone 5)***
- Incorporating an option for audio input and integration with Google Meet. ***(Mile-Stone 4)***
- Implementing QA generation using external models such as Gemini and ChatGPT. ***(Mile-Stone 6)***
- Developing a web application for quiz generation. ***(Mile-Stone 7)***
- Deployment of both chrome extension and web application. ***(Mile-Stone 8)***

I plan to start by creating a user interface using ReactJS to handle different types of questions. Then, I'll add a feature where users can choose the difficulty level and the number of questions they want. This way, we can use a similar setup for different question formats.

Next, I'll develop a model to generate distractors for multiple-choice questions, and other models for true/false and fill-in-the-blank questions. After that, I'll work on implementing a speech-to-text feature for audio input and integration with Google Meet.

Following that, I'll focus on integrating the system with Google Forms, and then, I'll connect it with external models using APIs. Finally, I'll develop a simple web application for quiz generation using the APIs.

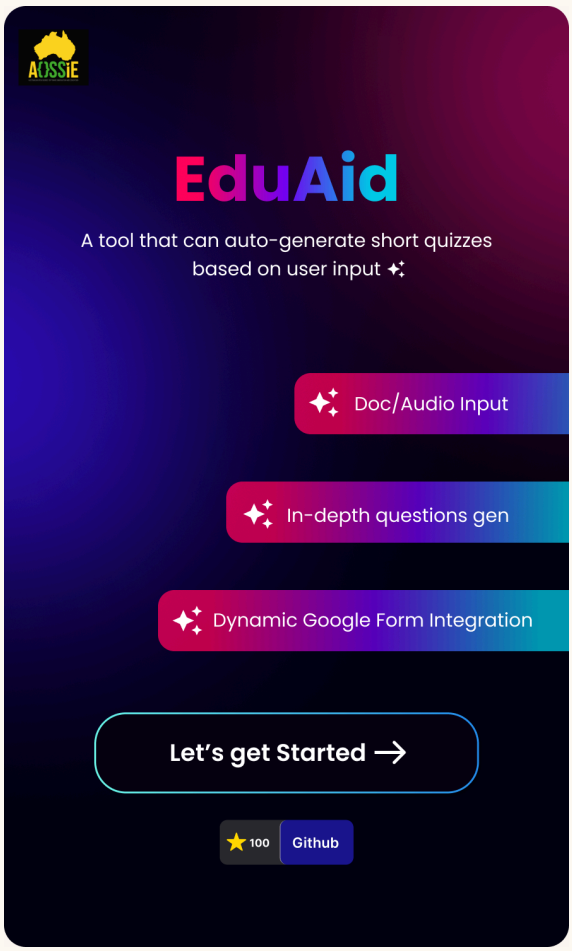
During deployment, I'll make sure the Flask backend application can handle lots of users and stays reliable. Also, I'll put the extension on the Chrome Web Store so people can find and use it easily. Lastly, I'll set up the website's frontend to make sure everything works smoothly for users.

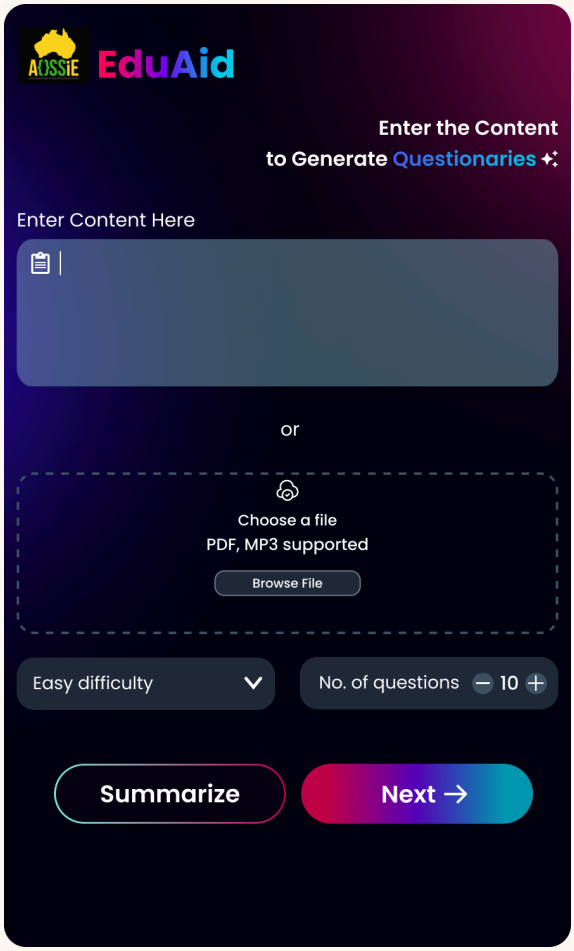
Implementation of the Project Goals

Milestone 1 **- Implementation of UI using ReactJs**

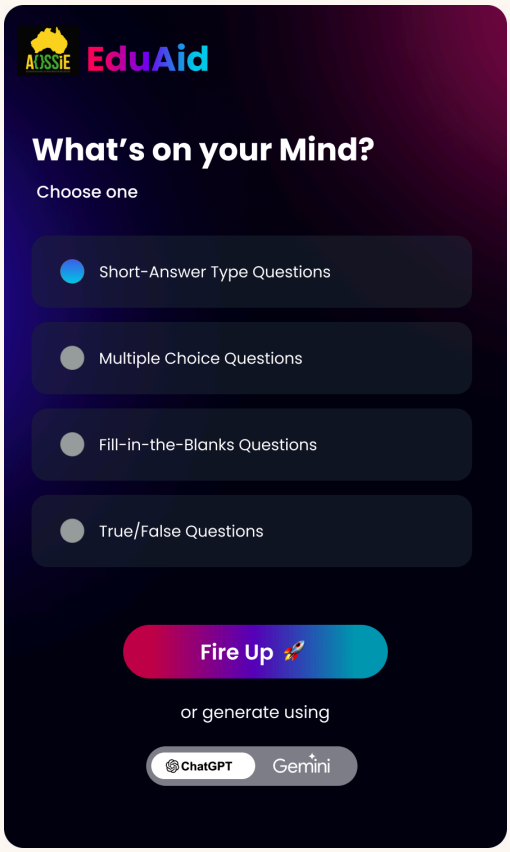
I'll use **ReactJS** along with **TailwindCSS** to design the user interface for EduAid. Since this combo is widely used nowadays, it'll be simple to keep the interface up-to-date and make any future changes. The user interaction will be guided through a defined flow, which can be accessed in this figma file. Within this interface, users will have the option to select a category for their question-answer pairs, input content either through text or audio formats, and specify the desired difficulty level and number of questions.

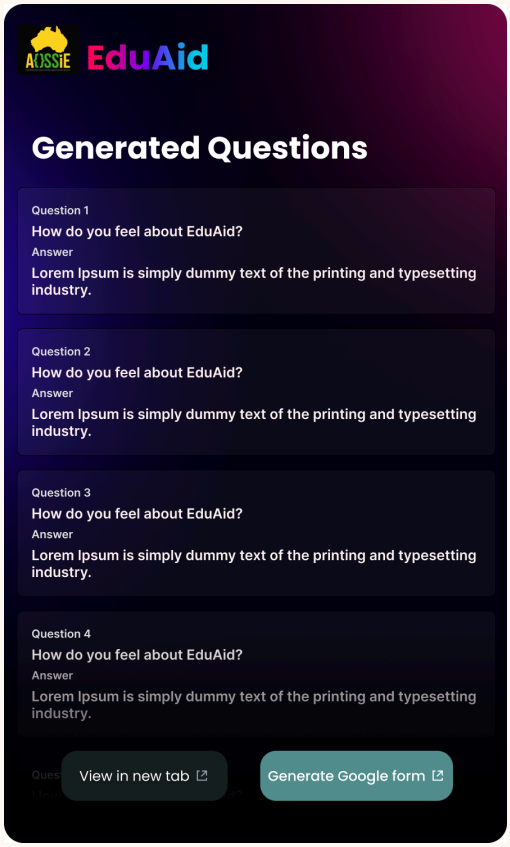
Additionally, users will have the flexibility to choose between several functionalities. They can opt to directly utilize EduAid's model for generating and viewing the question-answer pairs. Moreover, users have the option to select either Gemini or ChatGPT's model, providing the necessary API key, to generate the question-answer pairs using these models.





Furthermore, after generating the question-answer pairs, users will have the option to create Google Forms directly from the generated content.





The quizzes generated by EduAid will be presented interactively, making it easy for users to practice. Additionally, there will be an option to export these quizzes to Google Forms. This feature will be especially useful for teachers who can easily share the quizzes with their students for assignments or assessments.

Milestone 2: Generating User-Specified Question

In Milestone 2, the focus will be on implementing a system that can generate a user-specified number of questions of varying difficulty levels. Since the project is intended to be run locally, large language models like GPT-3 or GPT-4, with approximately **1.5 billion parameters**, are not suitable due to their closed-source nature and the associated costs of API calls. Therefore, to overcome these limitations, I will adopt an alternative approach inspired by research on **difficulty-controllable question generation.**

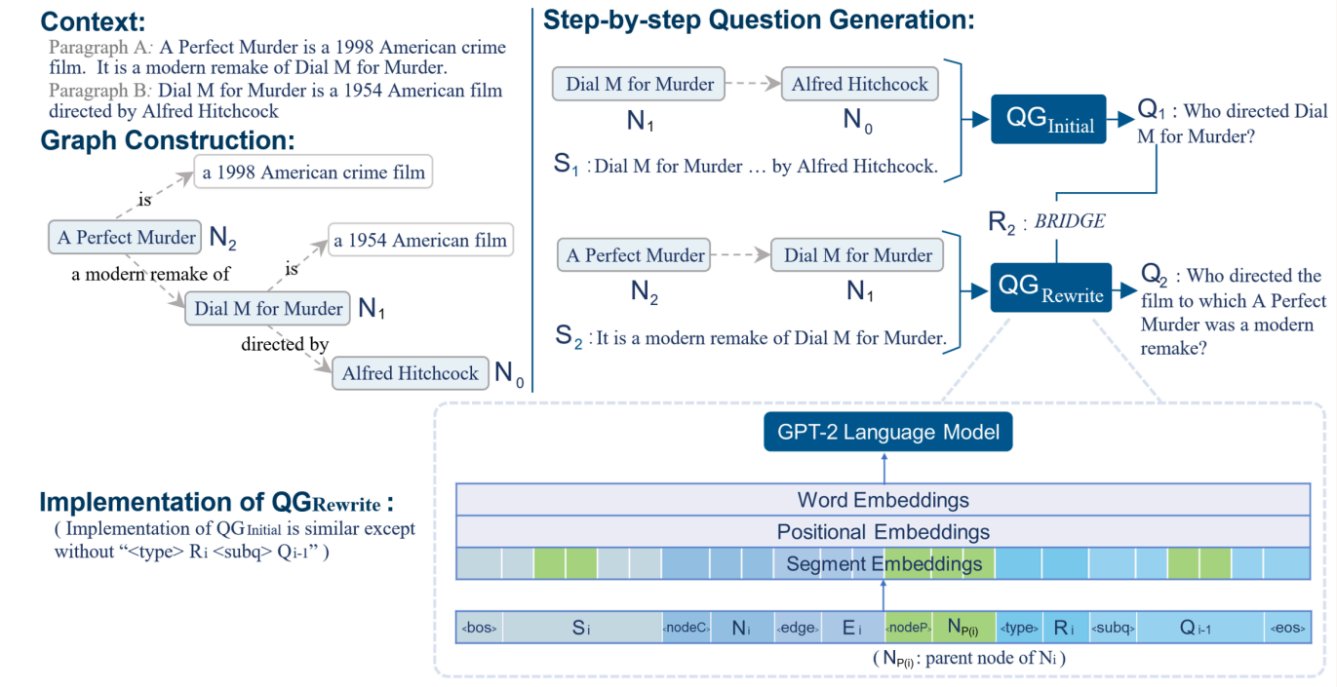
Let's say, the difficulty of a question is determined by the **number of logical steps** needed to answer it. For example, a simple question, like "Who starred in Top Gun?", asks you to take just one step (recalling a single fact). A more complex question, like "Who starred in a 1986 action film directed by Tony Scott?", requires you to climb a few more stairs (remembering several details like the year, genre, and director). This approach ensures that users are challenged appropriately based on the depth of their knowledge

Step-By-Step Approach

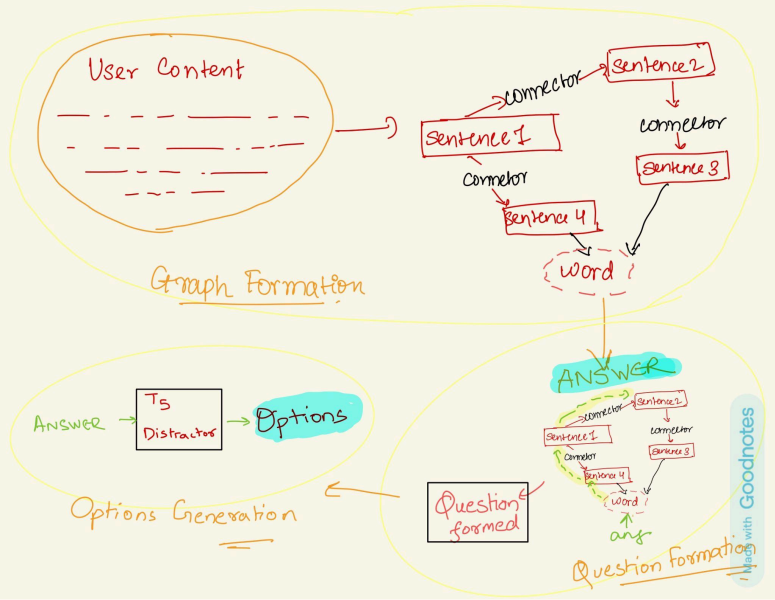
Our goal is to automatically generate question-answer pairs (Q, A) based on a given content text (C) and a difficulty level (d). **The difficulty level determines the reasoning steps needed to answer the question. Easy questions (d=1)** require a single fact from the context, while **medium (d=2)** and **hard (d=3)** questions involve chaining multiple facts together to answer.

The HotPotQA dataset is perfect for training such a system. It provides questions, answers, difficulty level and context all in one place. This allows us to train the T5 model to ask **easy, medium, or hard questions** based on the information given.

id	question	answer	type	level	supporting_facts	context
5a7a0693554299019 8eaf050	Which magazine was started first Arthur's Magazine or First for Women?	Arthur's Magazine	comparison	medium	{ "title": ["Arthur's Magazine", "First for Women"], "sent_id": [0, 0] }	{ "title": ["Radio City (Indian radio station)", "History of Albanian football", "Echosmith", "Women's colleges in the Southern United States", "First Arthur County Courthouse and Jail", "Arthur's Magazine", "2014–15 Ukrainian Hockey Championship", "First for Women", "Freeway Complex Fire", "William Rast"], "sentences": [["Radio City is India's first private FM radio station and was started on 3 July 2001.", " It broadcasts on 91.1 (earlier 91.0 in most cities) megahertz from Mumbai (where it was started in 2004), Bengaluru (started first in 2001), Lucknow and New Delhi (since 2003).", " It plays Hindi, English and regional songs.", " It was launched in Hydera bad in March 2006, in Chennai on 7 July 2006 and in Visakhapatnam October 2007.", " Radio City recently forayed into



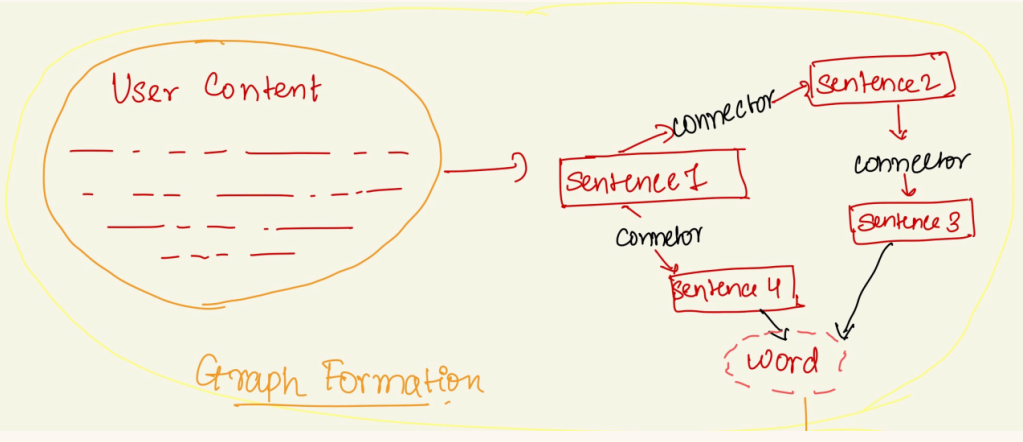
Simplified version:



1. **Context Graph Construction:**

   A **Context graph** is made up of Nodes and Edges. Each node of the CG represents a concept and each edge is a relationship between a pair of such concepts.

   a. We begin by creating a context graph that captures the relationships between entities in the text.
   b. Each sentence in C is analyzed to extract triples of the form **<subject, relation, object>**.
   c. These triples are then converted into nodes connected by directed edges. For example, the sentence **A Perfect Murder is a 1998 American crime film** would be transformed into two nodes: **A Perfect Murder** and **1998 American crime film**, connected by the edge labeled **"is"**, as shown in the figure above.
   d. This process results in a graph where nodes represent **entities** and edges represent the **relationships** between them.



Neo4j is a library that makes dealing with graphs super easy. The code snippet below explains the creation of a simple context graph.

```
MATCH (source)-[r]->(target)
RETURN gds.graph.project(
    'graph', source, target,
    {
        sourceNodeLabels: labels(source),
        targetNodeLabels: labels(target),
        relationshipType: type(r),
        relationshipProperties: { relationship_id: id(r) }
    }
)
```

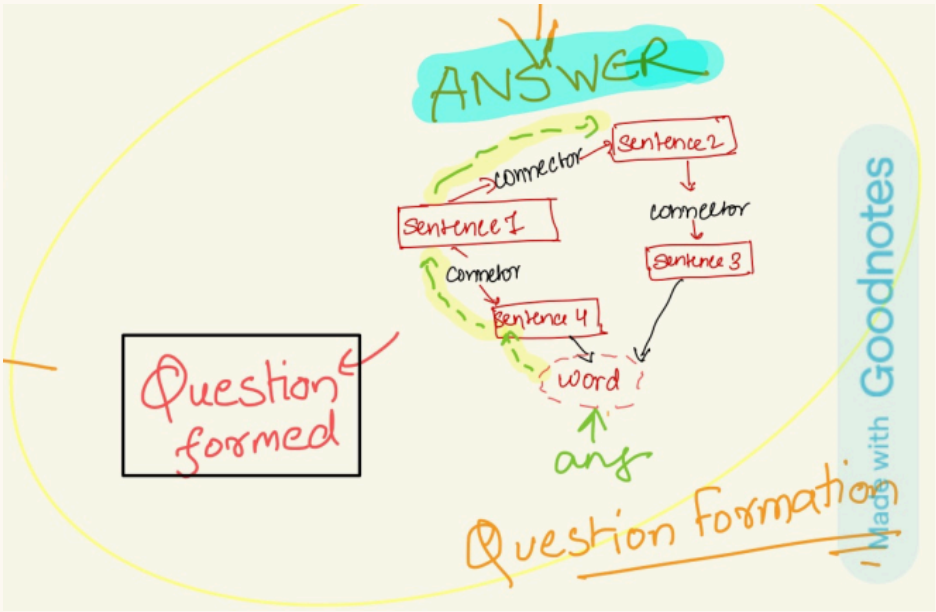
2. **Subgraph Sampling and Reasoning Chain:**

    a. Given the desired **difficulty level (d)**, a connected **subgraph (G_T)** with **d + 1** nodes is sampled from the context graph. This subgraph serves as the reasoning chain for the question we want to generate.
    b. Difficulty levels here correspond to the number of **hops** needed to answer the question. Easy questions (d = 1) can be answered directly from the context, while harder questions (d > 1) require following relationships across multiple entities in the subgraph.

3. **Question Generation Process**

    We employ a two-step approach to generate the final question:

    a. **Initial Question Generation ($QG_{initial}$):** This module takes the first node (N_0) in the reasoning chain (G_T), its corresponding entity (answer for the first hop, N_1), and the related sentence (S_1) as input. It then generates a simple question that can be directly answered using N_1.
    b. **Question Rewriting ($QG_{rewrite}$):** This module iteratively rewrites the question for (d - 1) times. In each iteration (i=1, 2, 3, ..., d), it takes:
        i. The previous question (Q_{i-1})
        ii. The current node (N_i) and its parent node ($N_{P(i)}$) in the reasoning chain
        iii. The corresponding sentence (S_i) for the current node.



$QG_{rewrite}$ analyzes the connection between the current node (N_i) and its parent ($N_{P(i)}$) as described by sentence (S_i). It refines the question step-by-step using the connections between entities in the subgraph, ultimately leading to a question that requires following the reasoning chain for an answer.

The system's ability to generate a desired number of questions depends on the provided context length. Generating a high number of questions from limited content (e.g., 10 lines with 3 sentences) might be challenging. In such scenarios, we can offer two options:

1. **Request for More Content:** The system can prompt the user to provide additional content to facilitate generating the desired number of questions.
2. **Generate Anyway (Limited Content):** If the user insists on generating questions despite insufficient content, the system can:
    i. **Extract Keyphrases:** Identify key entities or concepts from the limited context. This can easily be done using the current Model A
    ii. **Search for Similar Content:** Utilize these keyphrases to search for relevant and similar articles on platforms like **Wikipedia** using web scraping.
    iii. **Content Augmentation**: If the retrieved content is highly similar, the system may leverage it to supplement the original context and potentially improve question generation results.

With sufficient context, we can generate the user-specified number of question-answer pairs. The system can provide multiple sentences or combinations of sentences (based on the user's preference for question complexity) to the model, along with the original context. This ensures the model has enough information to generate the desired number of questions (e.g., 5 questions from 5 sentences or more complex questions from a combination of sentences).

Milestone 3: Generation of different question formats

Milestone 3 focuses on making our software capable of generating various types of questions like multiple-choice questions (MCQs), fill-in-the-blanks, and true/false based on user-provided content and difficulty level.

1. MCQ

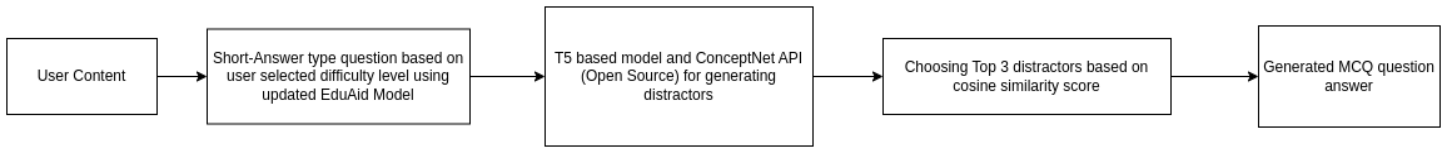
For MCQs, after completing Milestone 2, where our software could generate short-answer questions, we need to add distractors to the answers. Distractors are options in the MCQs that are similar to the correct answer but are incorrect. Techniques developed by Lelkes et al. at Google Research, have shown that T5 models perform well in generating distractors. We have three options for generating distractors:

1. **Using a Large T5 Model (3GB):** We can utilize a large T5 model fine-tuned on the RACE dataset for generating distractors. However, due to its size (around **3GB**), it may be inconvenient for users to download and use alongside our existing models. You can access this model here.
2. **Using a Smaller T5 Model (700MB):** Alternatively, we can opt for a smaller T5 model (around **700MB**) also fine-tuned on the RACE dataset for generating distractors. You can access this smaller T5 model here. In my **PR**(#21) for creating multiple-choice questions, I've utilized this model and provided a demonstration of its functionality and operation.
3. **Using Wordnet and Sense2Vec:** Using WordNet and Sense2Vec, we can identify semantic relations among words, such as synonyms (e.g., "car" and "automobile"). However, this approach faces challenges when generating distractors for answers with multiple words.
    a. The first video (linked here) illustrates the limitations of WordNet and Sense2Vec, as they fail to produce any distractors.
    b. In contrast, the second video (linked here) demonstrates the effectiveness of a fine-tuned T5-small model, generating five distractors in seconds. Thus, it illustrates that more effective distractors can be generated using the **model compared to traditional methods like WordNet and Sense2Vec, as employed by other contributors.**

Other benefits of using model based approach over library:

1. **Independence from External Libraries:** Using a model-based approach eliminates the need for external libraries, reducing server load during deployment and minimizing potential points of failure.

2. **Greater Control over System Responses:** Developers have more control over responses, ensuring solutions precisely tailored to project needs and facilitating adaptation to changing requirements and user feedback.
3. **Risk Mitigation:** A model-based approach reduces dependency risks, safeguarding against changes or discontinuation of libraries and ensuring system resilience and continuity of service delivery.



Alongside one of these models, we'll utilize the ConceptNet API, an NLP-based tool, to find words similar to the answer from the provided context. **The response time of this API is under 2 seconds.** The code snippet below illustrates how to achieve this.

```
def get_distractors_conceptnet(word, context):
    word = word.lower()
    original_word = word
    if len(word.split()) > 0:
        word = word.replace(" ", "_")
    distractor_list = []
    # context_sentences = context.split(".")
    try:
        relationships = ["/r/PartOf", "/r/IsA", "/r/HasA"]

        for rel in relationships:
            url = f"http://api.conceptnet.io/query?node=/c/en/{word}/n&rel={rel}&start=/c/en/{word}&limit=5"
            if context:
                url += f"&context={context}"

            obj = requests.get(url).json()

            for edge in obj["edges"]:
                word2 = edge["end"]["label"]
                if (
                    word2 not in distractor_list
                    and original_word.lower() not in word2.lower()
                ):
                    distractor_list.append(word2)
        return distractor_list

    except json.decoder.JSONDecodeError as e:
        print(f"Error decoding JSON from ConceptNet API: {e}")
        return distractor_list
    except requests.RequestException as e:
        print(f"Error making request to ConceptNet API: {e}")
        return distractor_list
```

Subsequently, based on the **cosine similarity** between the generated distractors and the answer, we'll choose the top three similar distractors to include as options for the question.

2. Fill in the Blanks or Gap-Fill Questions

The process of automatically generating FITB exercises from an existing text involves two distinct steps:

1) **Identifying Important Sentences:** We first pinpoint sentences that contain crucial information within the text.
2) **Selecting the Blanks:** Within the chosen sentences, we select specific words or phrases to be replaced with blanks.

To achieve these steps, we can use the pre-trained **key phrase extractor** model **(Model A)** and **word co-occurrence analysis**. The underlying assumption is that words that frequently appear together share a meaningful connection. So we will utilize word co-occurrence likelihoods to pinpoint the most relevant keywords for blanking, ensuring they hold a strong contextual relationship with the surrounding words.

To represent **word co-occurrence likelihoods**, we can use the well-established word vector space model, **GloVe**. This model, trained on a massive dataset of text, assigns a unique vector to each word. **The dot product of two word vectors reflects the likelihood of those words appearing together.**

Using the GloVe model, we analyze each potential blank and identify the closest scope word within a set of potential candidates. Essentially, we're searching for the word pair **(blank, scope word)** that **minimizes** the dot product of their corresponding word vectors. This ensures a strong contextual connection between the blank and its surrounding text.

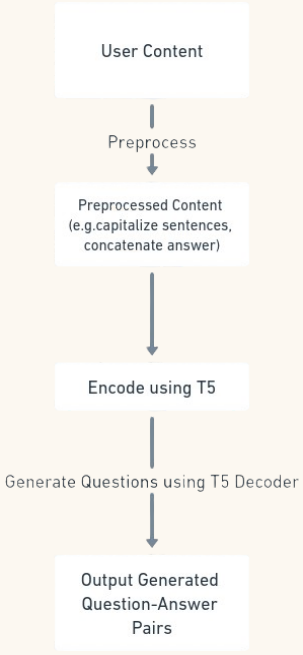
For generating difficult questions we can combine 2 or more FITB according to chosen difficulty.

3. True/False or Boolean Questions

To generate a specified number of boolean question-answer pairs from a given content, we can utilize either the T5 small or base model, which will be fine-tuned on the BoolQ dataset. This dataset comprises True/False questions formulated in natural contexts, each comprising a question, an associated passage, and a "TRUE" or "FALSE" answer.

question string · *lengths* 47–56 21.7%	answer bool 2 classes	passage string · *lengths* 504–973 41.8%
do iran and afghanistan speak the same language	true	Persian (/ˈpɜːrʒən, -ʃən/), also known by its endonym Farsi (فارسی fārsi (fɒːɾˈsiː) (listen)), is one of the Western Iranian languages within the Indo-Iranian branch of the Indo-European language family. It is primarily spoken in Iran, Afghanistan (officially known as Dari since 1958), and Tajikistan (officially known as Tajiki since the Soviet era), and some other regions which historically were Persianate societies and considered part of Greater Iran. It is written in the Persian alphabet, a modified variant of the Arabic script, which itself evolved from the Aramaic alphabet.

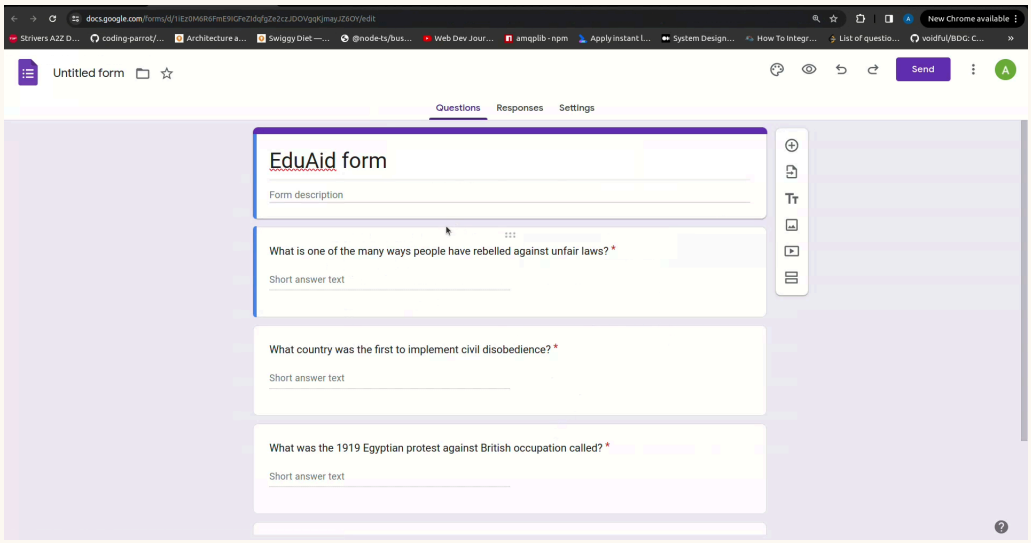
Initially, we'll **preprocess the data**, ensuring that each question begins with a **capital letter** and ends with a question mark. We will concatenate the answer ("TRUE" or "FALSE") with the passage to form a single string. Then, we will convert both the combined passage-answer strings and the questions themselves into sequences of **numerical tokens**, facilitating comprehension for the model. The encoder will handle the combined passage-answer string, while the decoder will generate the corresponding question based on the encoded information.



This approach will enable us to generate an **equal number** of true and false statements. These statements will then be randomized and presented to the user. Moreover, for generating a user-specified number of questions with limited content, we can employ similar methods discussed earlier in the short-answer type section.

Milestone 5: **Integration with Google Forms**

While EduAid currently generates QA pairs in a **text format (.txt)**, we can also directly export them into Google Forms. This allows for creating interactive quizzes or assessments based on the generated content.



The provided code snippet can automatically create a Google Form for short answer questions. With some adjustments, we can also use it to create forms for other question types. For instructions on setting up the required **authorization (OAuth)**, refer to this guide here.

```python
SCOPES = "https://www.googleapis.com/auth/forms.body"
DISCOVERY_DOC = "https://forms.googleapis.com/$discovery/rest?version=v1"

store = file.Storage("token.json")
creds = None
if not creds or creds.invalid:
    flow = client.flow_from_clientsecrets("credentials.json", SCOPES)
    creds = tools.run_flow(flow, store)

form_service = discovery.build(
    "forms",
    "v1",
    http=creds.authorize(Http()),
    discoveryServiceUrl=DISCOVERY_DOC,
    static_discovery=False,
)
NEW_FORM = {
    "info": {
        "title": "EduAid form",
    }
}
requests_list = []

for index, (question, answer) in enumerate(qa.items()):
    request = {
        "createItem": {
            "item": {
                "title": question,
                "questionItem": {
                    "question": {
                        "required": True,
                        "textQuestion": {},
                    }
                },
            },
            "location": {"index": index},
        }
    }
    requests_list.append(request)

NEW_QUESTION = {"requests": requests_list}

result = form_service.forms().create(body=NEW_FORM).execute()
question_setting = (
    form_service.forms()
    .batchUpdate(formId=result["formId"], body=NEW_QUESTION)
    .execute()
)

edit_url = result["responderUri"]
qa["edit_url"] = edit_url
webbrowser.open_new_tab(
    "https://docs.google.com/forms/d/" + result["formId"] + "/edit"
)
```

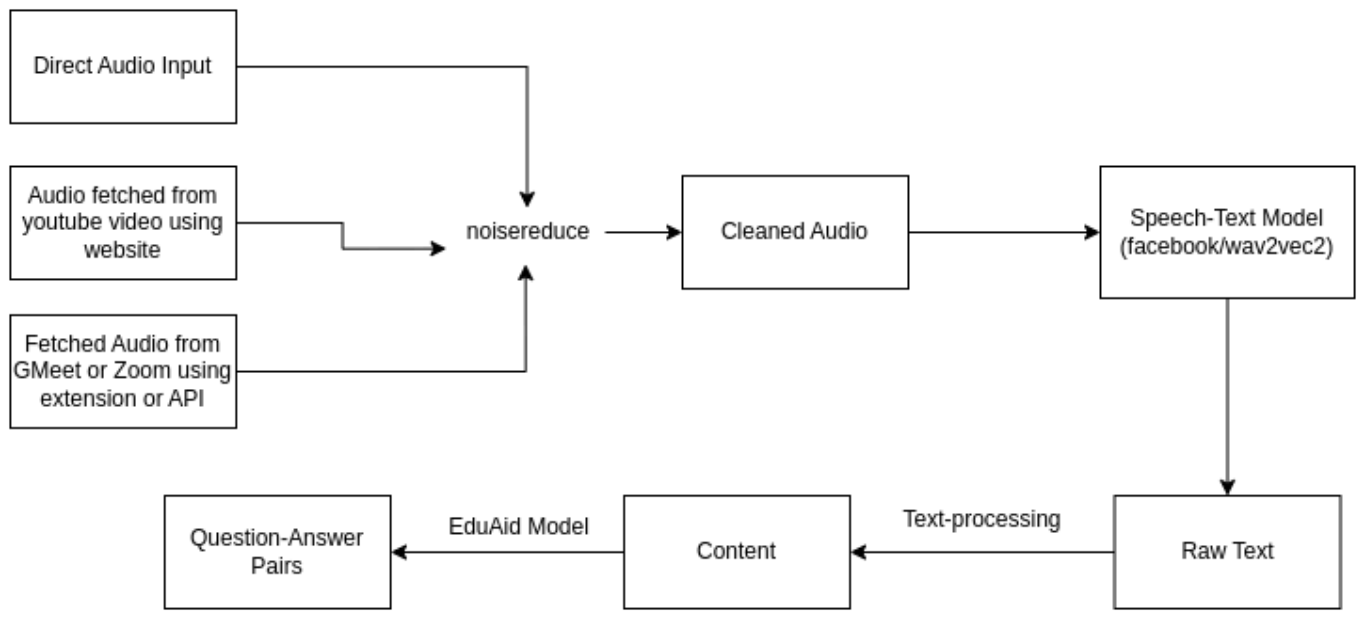
In **PR** (#23), I've integrated Google Form for short-answer type questions, making it easy to edit and share. Additionally, I've attached a detailed video demonstrating the integration.

Milestone 4: **Adding Audio Input and Google Meet Integration**

This milestone focuses on incorporating audio input and compatibility with popular video conferencing platforms like **Google Meet, Teams, and Zoom**. Similarly, audio can be extracted from **YouTube videos** using websites like y2meta directly through the video link.

Two integration approaches are possible:

1. **API Integration:** One way is by using official APIs offered by these platforms to directly record the meeting audio. This ensures a secure and reliable audio source.
2. **Extension Integration:** We can also use browser extensions (like beesy) specifically designed to capture audio from online lectures.

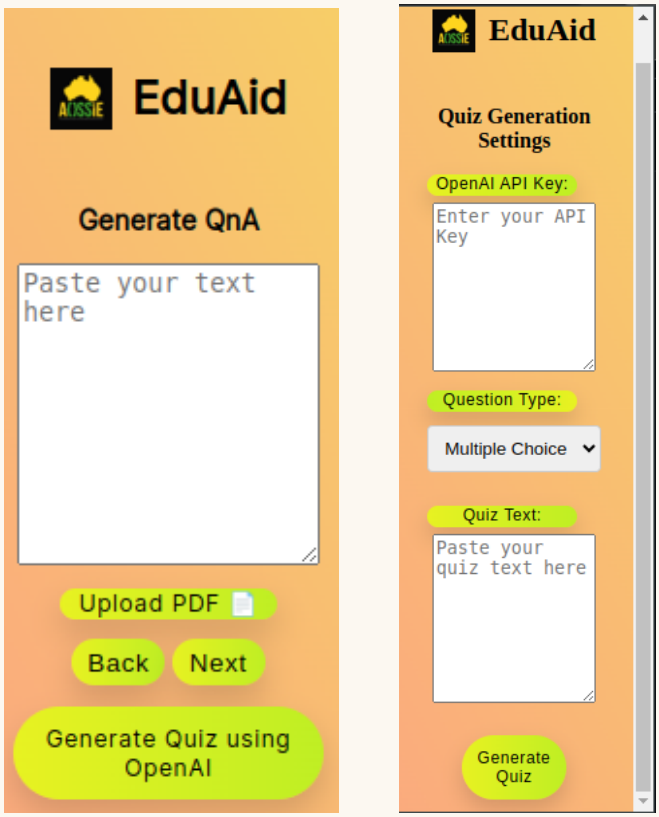


After capturing audio, we'll reduce any background noise using tools like ***noisereduce***. This step makes the audio clearer for converting speech to text, which we'll do using models like ***facebook/wav2vec2-base-960h***. We could also use simpler tools like ***speechrecognition***, but they might not work well with all types of audio or large audio files. In **PR**(#35), I added a feature for audio input using the mentioned model and explained how it works in a video. The resulting text will then be processed and used in our model to generate Q&A effectively.

Milestone 6: Generation using external Models

Due to limitations in computational resources, directly using powerful language models like GPT-3 within the software might not be feasible. However, users can utilize their OpenAI and Gemini API keys to access these models. This enables users to generate QA pairs in various formats, including MCQ, fill-in-the-blanks, and true/false questions, all derived from their provided content. The underlying code snippet takes user input for question type and content and generates MCQs. By modifying the prompt, we can easily generate different question formats with user-defined difficulty levels. In **PR** (#19), I integrated OpenAI's GPT-turbo-3.5 model and attached a sample of QA pairs generated by it using the prompt.

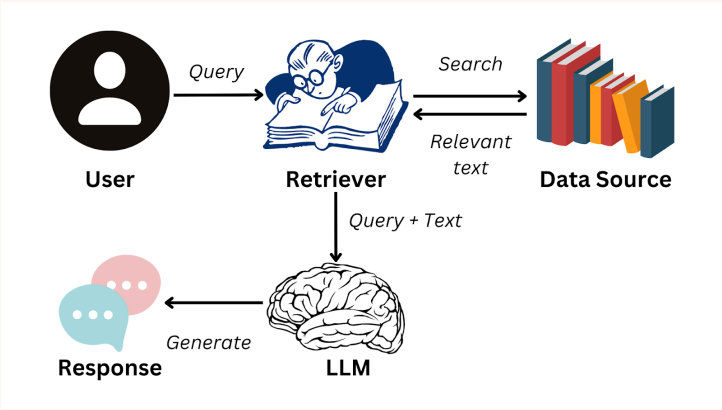
```
openai.api_key = api_key
prompt = f"Given the following text:\n\n{context}\n\nPlease generate at least 3 {ques_type} type questions
related to the provided information. For each question, include options and the correct answer in the format
Question:, Option:, Answer:. Ensure the questions are clear, concise, and test the understanding of key
concepts in the text."
response = openai.Completion.create(
    model="gpt-3.5-turbo",
    prompt=prompt,
    temperature=0.7,
    max_tokens=1024,
)
choices = response["choices"]
if choices:
    choice = choices[0]
    mcqs = choice["text"]

    self.wfile.write(json.dumps(mcqs).encode("utf-8"))
    self.wfile.flush()
```



POC Implementation in PR (#19) raised by me

Alternatively, we can adopt a **Retrieval-Augmented Generation (RAG) approach**. This method involves retrieving relevant information from a knowledge source and then utilizing a generative model like GPT to produce responses or content based on the retrieved information. By integrating specific user content and additional contextual information from external sources, such as online textbooks, academic articles, and curated question databases, this approach can improve precision and personalize quiz creation.

Here is a simplified flowchart summarizing how RAG works:



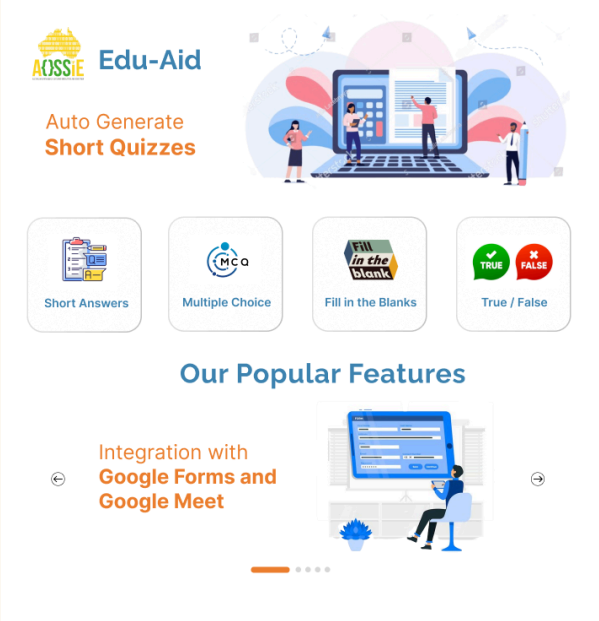
Depending on the results, we can evaluate which approach is more effective and opt for that one.

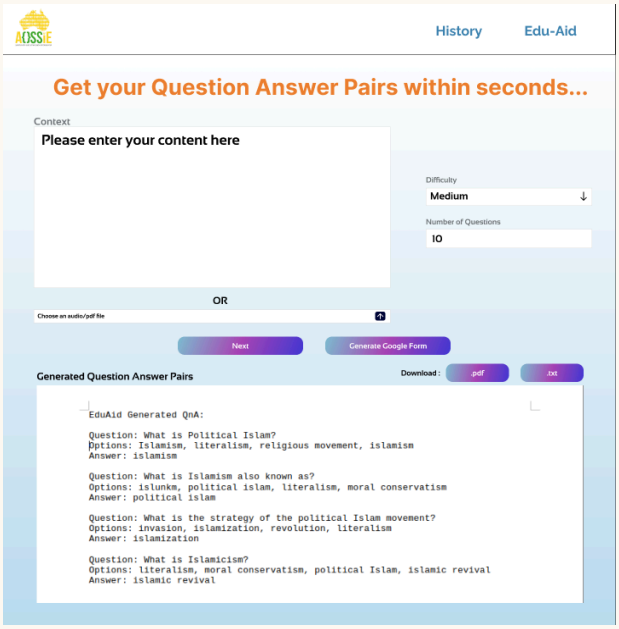
Milestone 7: **Web Application**

Currently, EduAid exists solely as an extension, but expanding it into a web application could significantly enhance its accessibility. Similar to the extension, the web app can accept input in both text and audio formats and utilize the same APIs for generating quizzes. We can develop a serverless backend using Python, coupled with a front end built on **ReactJs**. If time allows, we could integrate MetaFlow for efficient model deployment and maintaining a structured architecture.

Moreover, the web app could incorporate features like saving users recent quizzes either locally or in an online database, ensuring they can revisit their progress. This enhanced functionality and accessibility could attract more users and make learning more engaging and efficient.

Here is my planned UI design for the web app, which can be adjusted based on mentor feedback:https://www.figma.com/file/TJqPWiydUHCzJRO9XwjPgp/EduAid-Web





Milestone 8: **Deployment Phase**

To start, we'll set up the backend for our application using Flask. This will allow us to create APIs that can easily connect with our website's frontend and any extensions we develop. We'll make sure that our backend can handle at least 100 users at the same time, ensuring it's reliable and always accessible.For deploying our Flask backend, we have a few options:

1. **Vercel:** Vercel makes deploying Flask applications straightforward, and their documentation is easy to follow. Their free tier usually allows around 100,000 function invocations per month.

2. **Render**: Similar to Vercel, Render offers clear documentation and some extra benefits. They provide 750 free instance hours per month. Both Vercel and Render offer features like PR review, automatic redeployment, and excellent monitoring.

3. **AWS**: While AWS is a paid service, it's a reliable option if the free choices can't handle our user load. Their documentation for Flask deployment is detailed. However, deploying on AWS can cost around $30 per month, depending on usage.

A React-based web app's frontend can also be easily deployed on either Vercel or Render. I have previously used Vercel for frontend deployment several times. Upon further discussion with mentors, one of the aforementioned services can be used for deployment.

For publishing our extension to the Chrome Web Store:

1. First, we'll bundle all the code and files of our extension into a zipped folder.

2. We'll need a Google developer account, which requires a one-time $5 fee to publish up to 20 extensions.

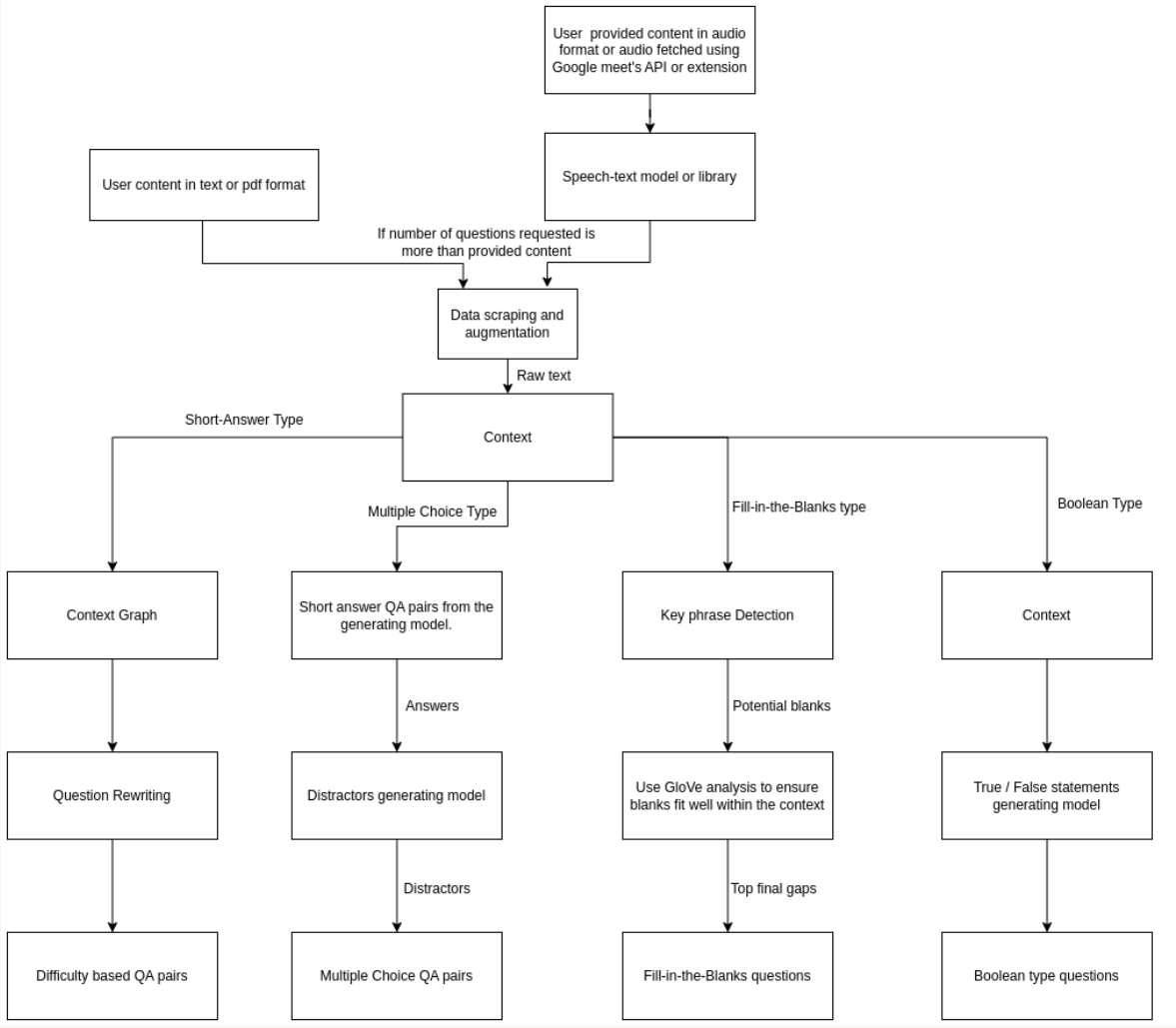
3. Once logged in to the Chrome Developer Dashboard, we'll upload our zipped extension and provide details like its title, description, and screenshots.

4. Google will review our extension to ensure it meets their standards. After approval, it'll be available for Chrome users to find and install.

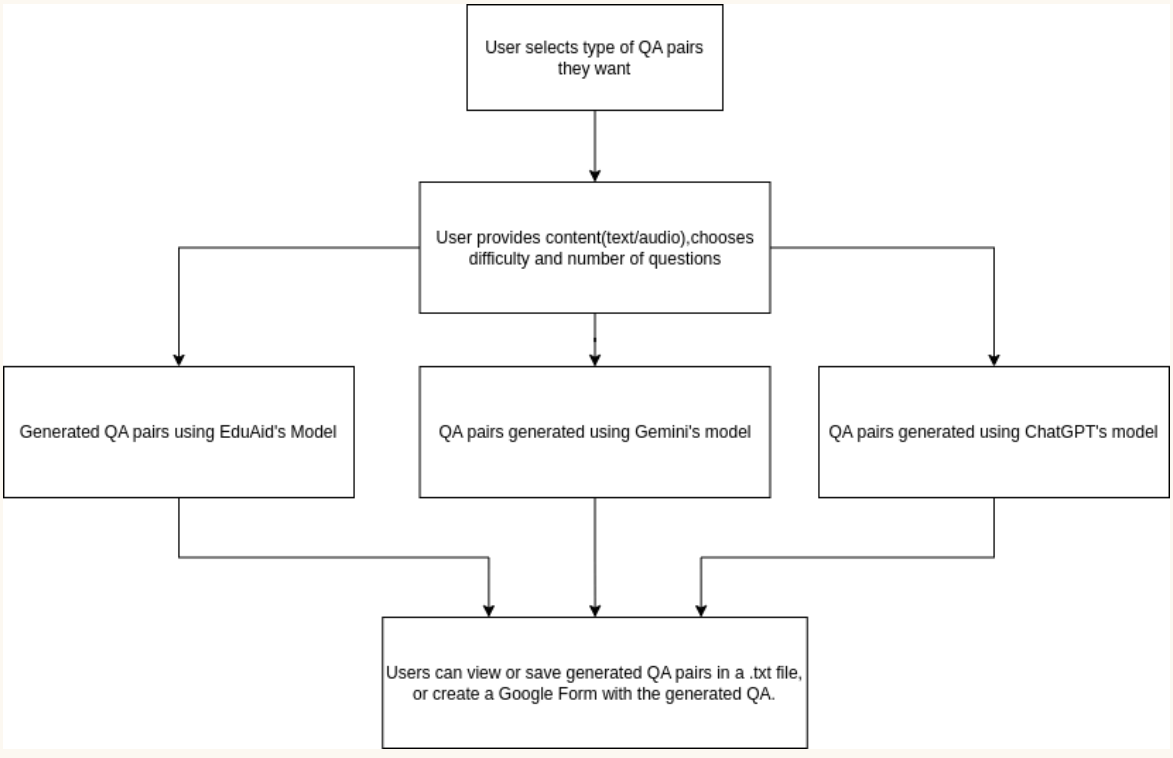
Final Architecture

The final workflow will comprise four distinct models designed for difficulty-controlled question generation. These models are independent, allowing users to download them according to their specific needs.

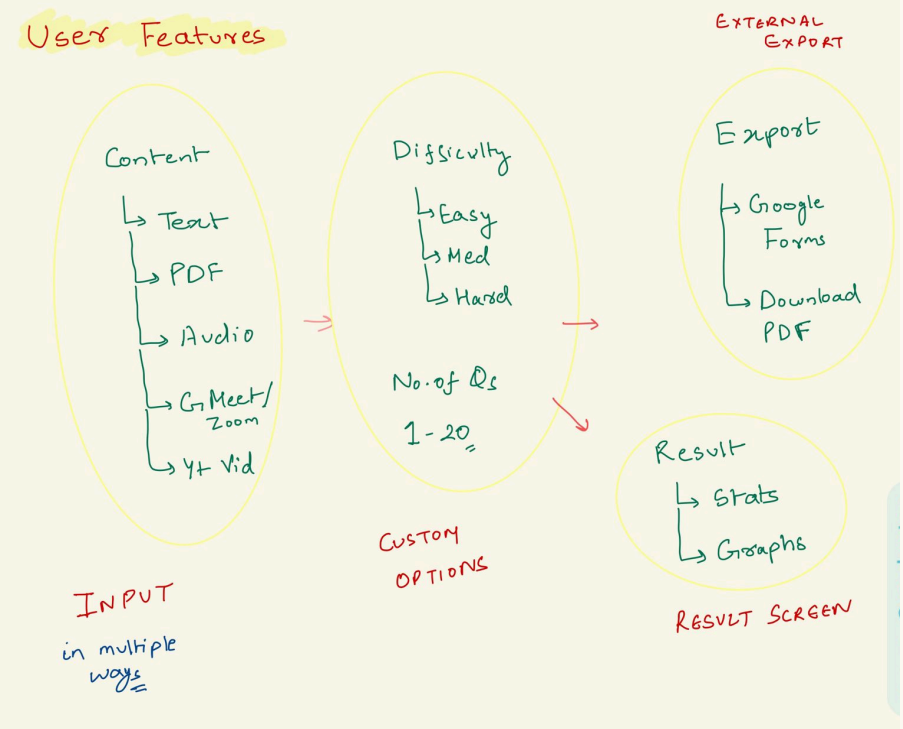
1. **Model A:** This model will retain the current functionality, serving as a keyphrase extraction model. It will be utilized for generating Fill-in-the-Blank (FITB) type questions, as discussed previously.
2. **Model B:** This model is a T5 model fine-tuned on the HotPotQA dataset to generate difficulty-based questions. It takes user content (C) and a difficulty level (d) as input, producing multiple questions based on the required reasoning hops.
3. **Model C:** This model will serve as a distractor generator, responsible for generating distractors for the answers provided by Model B, and will be used for creating multiple choice questions.
4. **Model D:** This model will be used for generation of Boolean type questions and will be an T5 model fine tuned on BoolQ dataset as discussed earlier.



User Workflow



User Features



Timeline

This is a rough week-by-week timeline that I would like to follow. This timeline will remain flexible and subject to change based on discussions with mentors and as the project progresses.

Time Period	Deliverables
Community Bonding Period	Initialize ReactJS for the UI and improve the features implemented in the PRs. Make changes to the final architecture and figma design if any are required.
Week 1	Completion of the UI using ReactJS and Tailwind CSS based on the final design approved by mentors.
Week 2	Training a model for difficulty controllable question generation
Week 3	Creating a pipeline for model training, evaluation, and improvement through hyperparameter tuning.
Week 4	• Test the pipeline with different cases to ensure it's robust and identify edge cases.

	• Develop a web scraping script to extract data from Wikipedia, enhancing user content and addressing limitations in question generation.
Week 5	**Buffer week before first evaluation**
Week 6	• Training / improving the pretrained model for distractor generation • Train a model for generation of Fill-in-the-Blanks type questions
Week 7	Train a model for Boolean type question generation
Week 8	Implementing the feature of audio input and developing a method to convert online meet content to text format
Week 9	Integrate Google Forms for all question formats and enable users to utilize their ChatGPT and Gemini API keys for QA generation.
Week 10	Finishing the Web App in ReactJS and Tailwind CSS + Testing
Week 11	• Deployment of backend and frontend services and publishing the extension on webstore. • Unit, Integration, Load and E2E testing.
Week 12	**Buffer Week before final evaluation**

Note: Training the models could take a variable amount of time, although it is expected to take less time than allotted if good pre-trained models relevant to our task are found. In such cases, the remaining time from those weeks could be added, along with an additional buffer week at the end for testing and deployment of these models on various free platforms.

My Past Contributions to Edu-Aid

I started contributing to Edu-Aid at the end of November 2023 and continued to explore and learn.

Issues Opened

1. Feat: Multiple-Choice Question Generation
2. Feat: Direct Editing & Sharing of Short Answer Questions in Google Forms
3. Feat: Integration of OpenAI API for QA Pair Generation

4. Feat: Add Audio Input Support
5. Text Area Overflow

PRs Created

1. [Feature] QA Pair Generation using OpenAI's API Integration
2. [Feature] Integration with Google Forms
3. [Feature] Multiple Choice type questions
4. [Feature] Added Audio Input for Generating Q&A
5. Removed text area overflow

Here are some merged PRs I've contributed to other projects at AOSSIE, in addition to my work with EduAid:

1. Added image crop feature
2. feat: Removed delete button for default profile picture

Why are you the best person to execute this proposal?

I'm experienced in building user interfaces using ReactJS and TailwindCss, and I'm familiar with writing clean and scalable code. I've also worked on transformers, LLM's and various system architectures. With my background, including working with startups on multiple projects, I believe I have the skills and experience needed to successfully complete this project.

Prior Experience

For the past year and a half, I've been diving into web development using ReactJs, Django, Node.js and Flask. At IIT Bhilai, I'm actively involved in the Machine Learning research group. Through courses and research, I've gained expertise in Machine Learning and Natural Language Processing. I've delved into various ML and NLP papers, broadening my understanding. Additionally, I've worked with Finorator, a startup, as a Full-stack developer. There, I utilized ReactJs for frontend development and Flask for backend tasks. During my time there, I also contributed to building a mutual fund recommendation system. In terms of tools, I'm proficient in Tensorflow, Pytorch, Python, C, NLTK, MetaFlow, and CoreNLP.

(Please find my resume here)

Relevant Coursework:

1. Machine Learning
2. Advanced Machine Learning
3. Natural Language Processing

Post GSoC plans

There are very few chances that some things might go unimplemented because 11 weeks is plenty of time, but still, If it happens, I'll try to complete them post GSoC. I would like to be associated with Australian Open Source Software Innovation and Education and the EduAid project even after the completion of the Google Summer of Code program. I feel that regular involvement in this project would allow me to take this to better levels in terms of usability. I would gain experience that would be very valuable to me and would also be my tiny bit of giving back to the open source community and thanking them for facilitating use of many wonderful open source software.

References:

1. Difficulty-Controllable Question Generation through Step-by-Step Rewriting (Yi Cheng)
2. End to End generation of Multiple Choice questions using Text-to-Text transfer Transformer models. (Torrealba et al.)
3. Answering Complex Open-domain Questions at Scale (https://ai.stanford.edu/blog/answering-complex-questions)
4. How to Convert Any Text Into a Graph of Concepts (https://github.com/rahulnyk/knowledge_graph)
5. T5-base-boolean-qgen_pretrained-finetuned (https://huggingface.co/arver/t5-base-boolean-qgen_pretrained-finetuned)
6. Automatic True/False Question Generation for Educational Purpose (Bowei Zou)
7. Learning to Rewrite Questions-in-Context (Elgohary et al.)
8. AUTOMATIC GAP-FILL QUESTION GENERATION FOR VIDEO LECTURES
9. What is Retrieval Augmented Generation (RAG)? (https://www.datacamp.com/blog/what-is-retrieval-augmented-generation-rag)