Google Summer of Code
EduAid- AOSSIE
Mentor: Ritik Jain

By: Prarabdh Shukla
Email: prarabdhshukla@iitbhilai.ac.in

SHORT QUIZ GENERATION USING MULTI LEVEL TRANSFER MODEL

—

Proposal

Synopsis: What is the problem statement?

With the advent of online learning taking a larger stage in the post pandemic scene, it has become evidently obvious that more AI based tools are required for testing the level of understanding that the learners of an online course have developed post each lecture. While some courses may be pre-recorded with well maintained transcripts, some may be live ones over video conferencing platforms like Google Meet. In this project, I propose an idea for an application which such courses and live online classes can leverage to quickly generate questions based on the content delivered in the course. The questions this application generates will be either Short Answer Type or Multiple Choice Questions (MCQs), however, it has potential to be extended to facilitate the Long Answer Questions too which will contribute towards the active involvement of the open source community with this project for quite some time.

As a deliverable, we will create an application which can be fed with the transcript of any lecture or a snippet of a book and will generate short quizzes consisting MCQs and Short Answer Type questions.

<u>Background and Prerequisites</u> (Literature Survey):

With Vaswani et al's 2017 paper, Attention is All You Need, Transformer based architectures have taken the Language Processing with storm. Transformers are a useful utility in the NLP space thanks to their ability to handle long range dependencies and overcoming computational complexities involved with scaling sequence aligned RNNs for complicated tasks. Armed with self attention, Transformers boast of providing the much needed Long range dependencies required for text processing tasks.

Several models have been developed based on this fundamental building block. GPT-2, GPT-3, etc are examples of pre-trained decoders that are available for use. We have models like BERT, RoBERTa, etc which are pre-trained encoders. While decoders are directly applicable to language modeling, encoders need a little bit of masking for achieving the same purpose. For the sake of our project, we would need to combine both of these components. T5 is one such model which has both pre-trained encoders and decoders. Intuitively speaking, for our task, the model will enjoy the bidirectional context provided by the encoders while the decoders will ensure long term memory of the input text.

<u>Possible Approaches:</u>

The question making process may be divided into multiple stages of "work" that needs to be done based on the type of question being generated (i.e., MCQ or short answer type)

I. MCQs

The key steps involves in generating Multiple Choice Questions are:

a. Generating valid questions from the input text

(What/will/how/when/where/which are possible categories of questions the model will learn to generate.) :

Long range dependency property of T5 will ensure that dependencies in the input text are correctly captured. The decoder's language modeling abilities coupled with the attention will ensure that the query words are placed at positions where the attention correlation is maximized (i.e., likelihood of neighborhood occurrence is maximum)

b. Identifying correct answers

The identified questions can be further used for identification of the correct answers. An intuitive way of understanding how the model will learn this information is as follows: The query words(what,which, etc) can in most cases be masked and a simple next word prediction task will allow the model to learn the correct answer.

c. Generating Distractors

Distractors are essentially what make good MCQ questions. Training the model to generate distractors for a given question will enhance the quality of questions being produced and will bring the quality of the questions closer to human-generated questions from the same text.

II. Short Answer Type

There are mainly three types of Short Answer Type questions:

1. Factoid answers: Which are based off single entities from the input passage
2. Span Answers: These answers span over a certain section of the passages

3. Generative Answers: Generative answers are answers which are generated from models and do not directly come from the input passage

For our case, Generative answers are not of much relevance because they are mostly intended to be used in AI Chatbots meant for human interaction. For educational content, Factoid and Span answers would be a more suitable fit.

a. *Generating valid questions from the input text :*

Long range dependency property of T5 will ensure that dependencies in the input text are correctly captured. The decoder's language modeling abilities coupled with the attention will ensure that the query words are placed at positions where the attention correlation is maximized (i.e., likelihood of neighborhood occurrence is maximum)

b. *Identifying correct answers:*

The identified questions can be further used for identification of the correct answers. An intuitive way of understanding how the model will learn this information is as follows: The query words(what,which, etc) can in most cases be masked and a simple next word prediction task will allow the model to learn the correct answer.

The main point of difference between the Short Answer and MCQ generation is that we will not generate the distractors in the former.

Workflow and Pipeline for a basic web app:

Let us defer the datasets to the next section and focus on the workflow of my proposed solution.

The proposed workflow consists of 5 milestones that are to be achieved for the successful completion of this task.

Milestone 1: The most basic task to achieve would be to parse the input text and generate a list of paragraphs/ list of set of paragraphs. We could also choose long pieces of text on a particular topic and use pre-trained BART with fine tuning for text summarization to generate paragraph long summary of topics. This will greatly allow us to compress really long pieces of text.

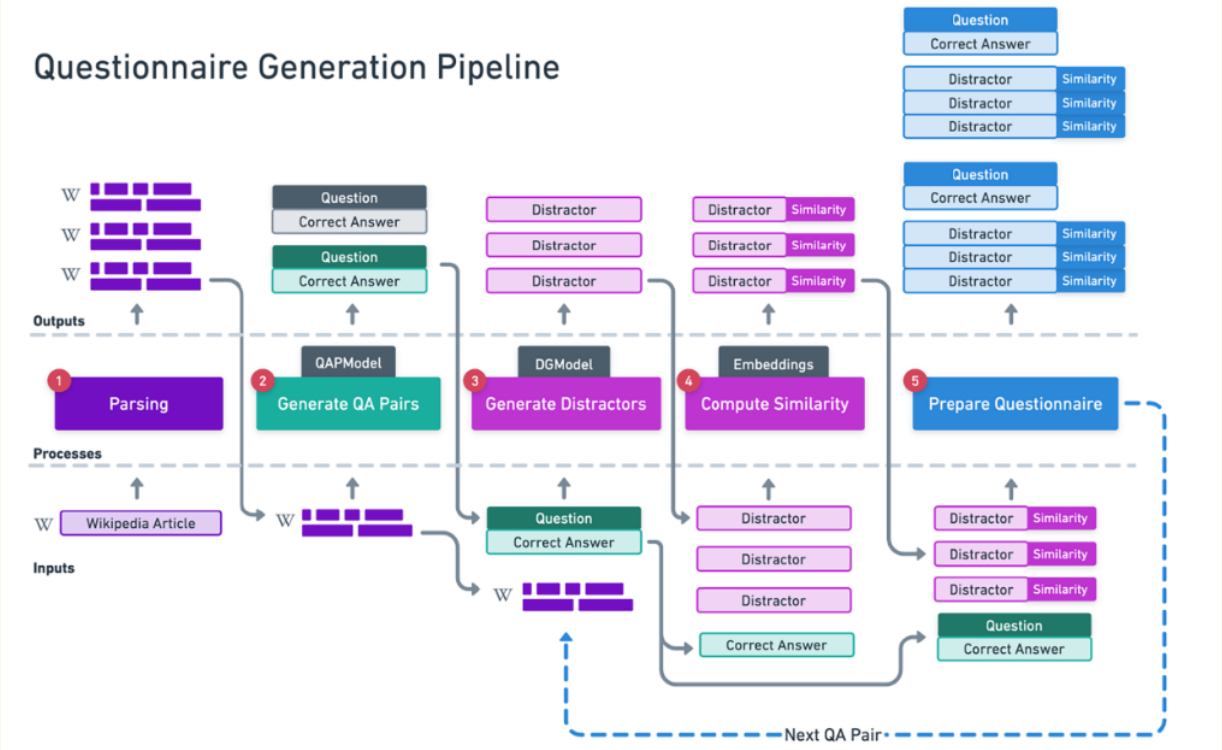
At the end of this, we aim to achieve a list of paragraphs, each of which will contain content on which questions will be generated by our model

Note: I have used the term "paragraph" above and in the rest of this section. However, this is just indicative of the fact that the input shouldn't ideally be too long. We can ofcourse include multiple paragraphs (with/without text summarization)

Milestone 2: The next goal to aim for would be to generate a set of question and answer pairs for each input paragraph. We can use many pre-trained T5 models for this task (as done by Patil et al.) using the answer aware question generation technique[1], alternatively, it has also been shown (Klein et al.) that we can use a combination of GPT-2 (decoder) and BERT(bidirectional encoder) for this task as well.

At the end of this stage, we would have successfully trained a model to generate a set of question answer pairs from the text summarized inputs in *Milestone 1*.

[1] In this technique, the model is trained by giving it answers and asking it to make questions out of it. This technique can be very useful in Short Answer Type Questions.



Milestone 3 (not needed for short answer type): Now we come to the most important stage of our workflow. The task at hand in this stage is to generate distractors for the questions generated in Milestone 2. We can again use a fine tuned T5 model. The model will accept the question answer pairs, and then replace the answer with incorrect answers. Intuitively, this part can be thought of as a task which is language modeling, but just that the blank spaces are replaced with questions where the question words what/when/which etc are the blank spaces. Now, we will find the most suitable words for the given context, and choose the top k (number of options+1) most probable ones and discard the most probable one.

The most suitable distractors in this case will be selected by using the cosine similarity of the distractors to the correct answer.

Note: The language modeling analogy is given just for the sake of intuition. Ofcourse, the model will learn cases where the top two three words were also correct answers to the question and hence will discard them.

Milestone 4: Once generated, we would perform a grammar-check on our questions using pre-trained BERT or pre-trained ULMFiT. These models have a wide range of use cases, one of which is grammar-check. They have shown a good performance, therefore, it should be okay to use these for the grammar check. We can fine tune these over subject specific data if the input is too advanced to be covered by their pre-trained weights.

Milestone 5: In this stage, we would serve the ML model using an API service that could be consumed by a React based frontend. This stage would mainly involve the development of the web app which is described below.

MULTI-LANGUAGE SUPPORT:

We can add a multi language support to our model by using pre trained mT5 [Xue et al.] developed by Google. This model was trained on over 101 languages, and can perform well after some fine tuning. *There is also a pre trained version of mT5 fine tuned on a question answering dataset.*

We can add support for the following languages whose number of speakers worldwide has been mentioned in brackets:

1. Spanish [512 million speakers]
2. French [284 million speakers]

Since these languages are similar to English and a lot of pre-trained + fine tuned models on these are readily available, it should be easy to achieve the objective of Multi Language support for two languages. There can be two ways in which we can achieve our objective:

I. By fine tuning mT5 ourselves. This may require around 30-50 GB of GPU. If such computation resources are available, this option is good, as we can fine tune the model on educational content in Spanish and French to achieve some good performance. We can use Kaggle for this task as it provides a 32 GB GPU.

II. If the aforementioned computation resources are not available, we may use many of the generic fine tuned versions of mT5 on Spanish and French. These are widely available thanks to the large number of speakers of these languages and thanks to these languages being similar to english. [see references]

Adding support for languages having a different script:

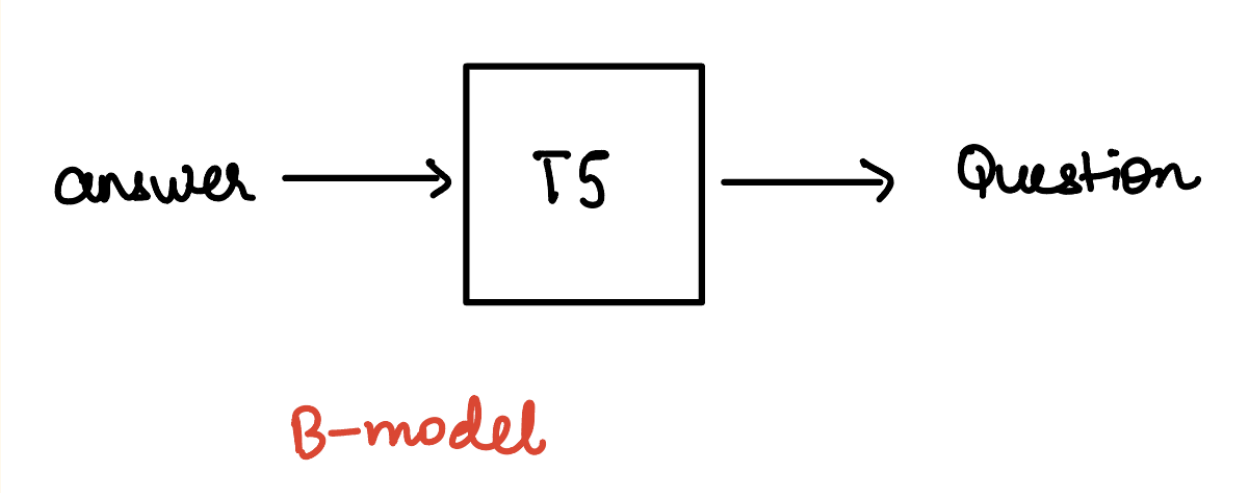
The reason for choosing French and Spanish was because they do not follow a different script, their pre-trained models are readily available (in case we do not have enough computation resources or if we are facing any roadblocks in fine tuning) we can easily add support for these languages. In reality, Chinese is the second most widely spoken language after English, but due to the complexities of the language, language modeling tasks on Chinese do not work well.

However, since I am well versed in Hindi, which has around 700 Million speakers, a goal to target would be to add support for Hindi. Adding a language based on a different script would involve challenges on all levels of the process. With my knowledge of Hindi, I feel that providing support for Hindi should be an achievable goal. There has been some work in this direction. [Huang et al.] and there are models like the Hindi-BERT which we can use for this process.

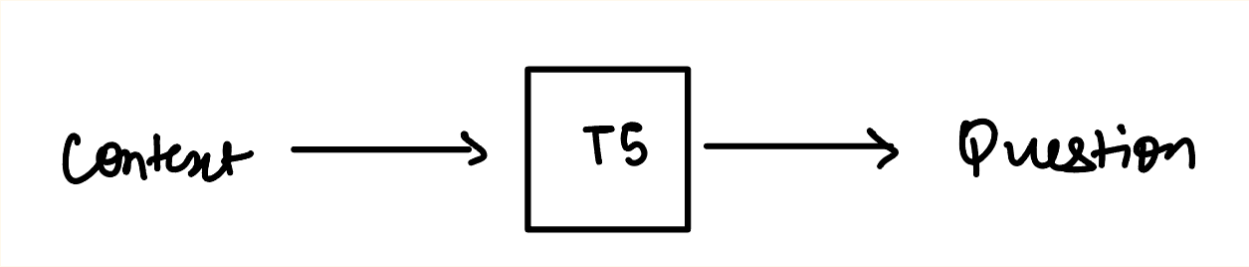
IMPLEMENTATION DETAILS: A brief description

In this section, we discuss how exactly we will train T5 to achieve our objective and what our model architecture will be. The datasets that we can use for fine-tuning are the CoQA dataset, SQUAD dataset, ELI5, CNN/DailyMail dataset etc. There are several versions of pre-trained T5 which was fine tuned on SQuAD and other datasets (https://huggingface.co/mrm8488/t5-base-finetuned-squadv2) . Several other pre-trained versions are available on HuggingFace. However no benchmarking has been done on these models therefore they may not be reliable for deployment use. Therefore, it would be advisable to fine tune T5 ourselves.

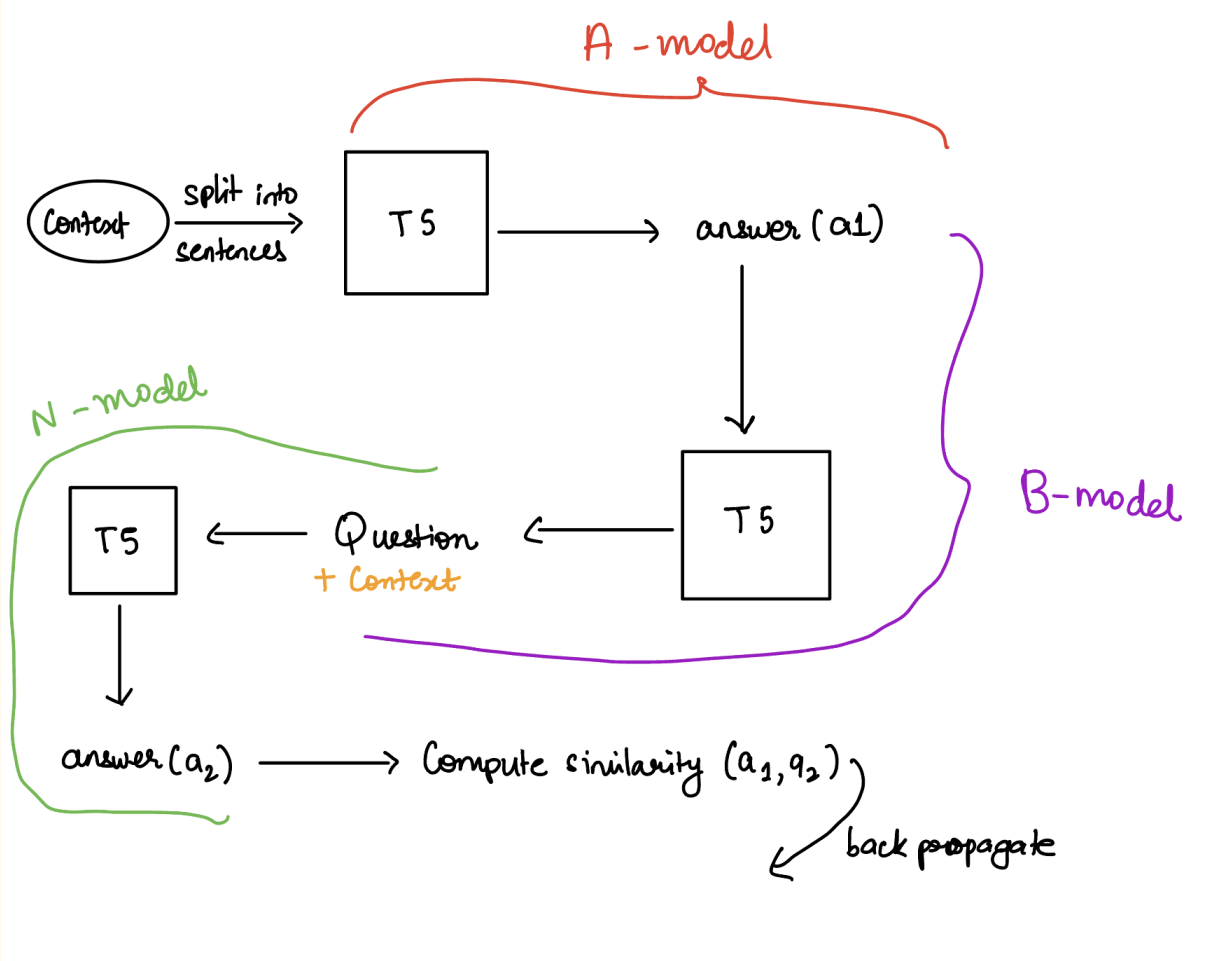
Baseline(B-model): As a baseline, we can fine tune T5 on the above datasets to train an answer aware model. This model will take input the answer embeddings and their positional encodings and will output the question. However, this model alone would not be enough for the task because of multiple reasons. The first one being that it requires the answer to be known beforehand which will not always be the case. Moreover, we'd like a system that takes input a paragraph of information and generates questions. So, this would not work well in such a setting.



Non answer aware baseline technique: If we do not wish to perform the aforementioned supervised task, we can do a context based training on the datasets, where given a context, a question will be generated. Such a model would have poor performance because it does not make questions by paying special attention to some specific keywords.



Final Model Architecture: For the final model, I propose a combination of the previous two baselines. The first part(A-model) would be a T5 model that will take as input the context and generate an answer (a1) from the context. This process can be thought of as detecting keywords. This particular task has been done widely and therefore a number of references are available for the same. This generated answer can then be feeded into the B-model which will generate a question. Finally we will apply the B-model to this generated question to generate an answer (a2). The model will calculate the similarity score and backpropagate it to train the architecture.



Detailed Discussion on the Final Architecture:

The final architecture described above has three components:

I. A-model
II. B- model
III. N- model

A-model: The job of the A-model is to identify the potential answers from the input context embedding. This is similar to identifying the keywords/important words from the input text. This task has been widely performed [Pezik et al.] and should not be hard to implement. At an intuitive level, using the long term dependencies, we would in essence be calculating the words with the highest attention scores.

Here, the input would be the context embeddings to a T5 model that can be fine tuned on any of the QA datasets described above. It will generate an answer (a1).

B-model: The job of the B-model is to perform answer aware question generation. The answer a1 will be fed into the model with the lexical position embeddings to generate a question. This task has been achieved using BERT [Chan et al.] fine tuned on SQUAD as even using T5 [Grover et al.] . We can either use BERT or T5 to fine tune on SQUAD and perform this task. We expect comparable performance from both.

N-model: The job of this model is to take as input the question and the context and generate the answer (which we call a2). This task has been widely used and implemented by many. One such T5 example on hugging face is here.

Similarity Computation: We will compute the similarity between the embeddings of the two answers a1 and a2. The similarity would be the cosine similarity and would be calculated as follows:

$$Similarity(a_1, a_2) = \frac{a_1 . a_2}{|a_1||a_2|}$$

A safe similarity would be around 80-90%. Once we get an answer with that much similarity, we can choose that particular question answer pair.

Inference: During inference time, we would only use the A and B models.

DISTRACTOR GENERATION:

For the distractor generation task, we can again fine tune the N-model to generate the wrong answers. The answers with less than 80-90 % similarity score can be chosen as the distractors for MCQs.

Techniques used by Lelkes et al. at Google Research show T5's high quality performance on the Distractor Generation task.

LIGHTWEIGHT ARCHITECTURE FOR RESOURCE CONSTRAINED CLIENTS:

Since, the T5-small model has nearly 60 million parameters and T5 base has 220 million parameters, for resource constrained users, even inference might become difficult to do as it would involve two T5 models at the very least in addition to the text summarization model. We can provide a light version of the application to such users. In the light version, we will generate template based questions. This would be particularly useful when the software is being used in a live class and the instructor wants to generate a quick quiz to test what the students have remembered from his lecture.

We would take each input sentence, and generate a question based on multiple templates. A few examples are given below:

1. Selecting Noun Phrases as answers: eg. *Google* is the biggest search engine. Here, the term *"Google"* is the noun phrase. We would generate a question based on the Noun Phrase, i.e., "Which is the biggest search engine"

2. Selecting the predicate, eg. Google is the biggest *search engine*. Here, search engine is the part of the predicate.

After selecting these parts of speech (POS tagging), we can either directly generate template based questions or we can feed these identified keywords into the B-model to generate a question.

The mentioned POS tagging can be done using a TF-IDF count vectorize operation. For each input sentence, the word with the highest value will become the input to the B-model for question generation.

LIGHTWEIGHT ARCHITECTURE FOR RESOURCE CONSTRAINED CLIENTS WITH GPU

If the client has GPU, we can use an attention model instead of TF-IDF in the workflow described above and use the words with high attention score to feed into the B-model.

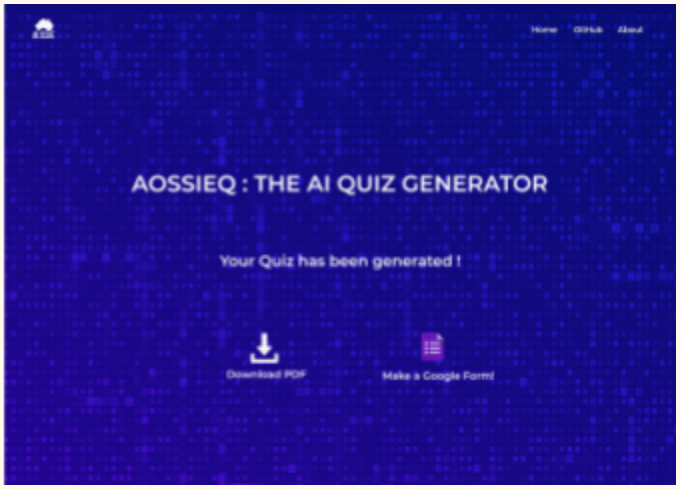
Note: Both these approaches make the inference process lightweight. Instead of inferring over two T5 models, we'd be using only one.

Developing a web application for quiz generation:

Using the API, we can easily develop a web application that would take an input paragraph and generate a question from it. We can also accept text documents and PDFs to generate a set of questions. We can have a serverless backend in python with a React based front end.

If any deployment issues are faced, at the very least, a docker image will be provided which can be used to run the software on the client's local machine like a desktop app.

The basic UI design (made using Figma) of the web app is given below:



Note: *This is just a first draft of the UI design and more features may be added after valuable discussions with the mentors.*

The above is just a basic version of the app. An exhaustive list of features is given below from which one or many can be added to the web app:

1. AOSSIEQ Lite: The model that implements the lightweight model (for clients with poor network connectivity/ desktop clients with low end devices)
2. AOSSIEQ Neural: The model described above which performs inference over multiple T5 models
3. Support for 4 languages: English, Hindi, French and Spanish
4. Making a Google form from the generated questions using Google Cloud console's Google Forms API
5. We could also add an option to take the quiz on the app itself
6. B-model: This feature, simply called "B-model" after our architecture, can be used to perform tasks related to self learning (see next section for more details).

<u>Using the B-Model as a standalone tool to help with Self Learning:</u>

A lot of students often watch YouTube lectures, and read several articles and books on their own. They often struggle to remember certain keywords/ key phrases. In the face of this, we propose an additional feature

of the app that will help students with self learning. The user can highlight some part of the uploaded document, and the selected part can be fed into the B-model as an input. The model will then generate questions whose answer is the key phrase.

The application will allow submission of multiple keywords/phrases at the same time and return a jumbled list of these questions.

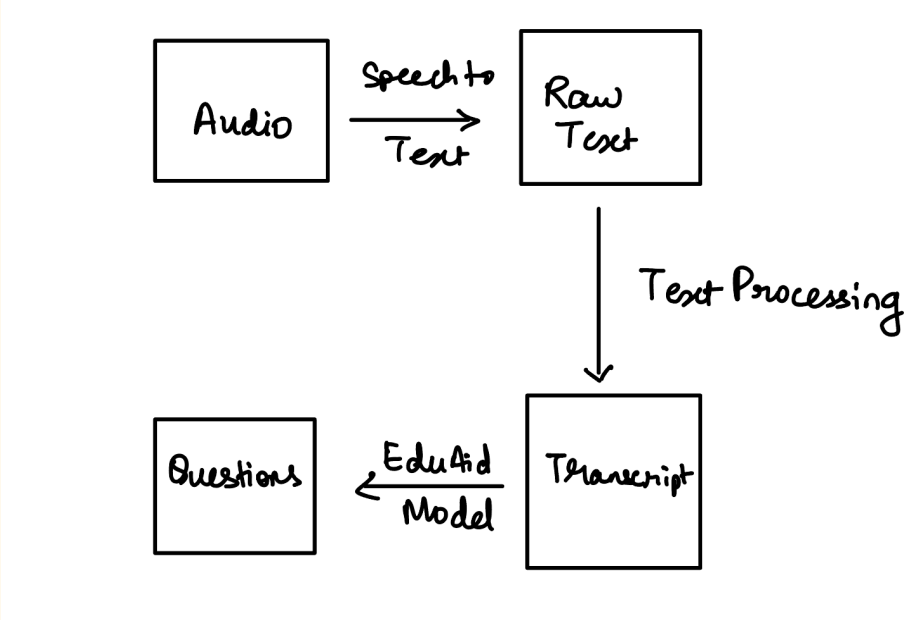
$$Bmodel(keyphrase,\ context)\ =\ Questions$$

INTEGRATION WITH GOOGLE MEET AND ZOOM:

To facilitate use of this software in online lectures, we can make a Chrome extension that will allow integration online meets.

The chrome extension will fetch the audio from the meet. To fetch the audio, we can make use of another extension as well. The audio will then pass through a speech to text converter(there are numerous options from this in python ranging from Google APIs to python libraries). The text generated from the audio will then be fed to our model to generate questions.

In addition to this, we can allow uploading of audio files from lectures in the application itself to perform the above task.



A similar process would work for Zoom.

An alternative way to approach this problem would be to make use of Google Meet APIs and Zoom APIs as well. Zoom provides APIs to allow participation in their meetings. Using the same, we can get access to the audio of the meet which can then be processed as described above. Similar resources are provided by the Meet API.

Evaluation and Deliverables:

I would like to keep the exact deliverables flexible and open for discussion during the proposal stage as well as during the community bonding period. Since the GSoC guidelines say that the working period is flexible, the deadlines can be adjusted accordingly.

First Evaluation:

1. Achieving milestones 1, 2 and 3.
2. Developing a basic react-native version of the web app.
3. Documenting all the code written so far

Second Evaluation:

1. Achieving milestones 4 and 5
2. Testing the web application with the latest model
3. Finishing the documentation of the code

The above evaluation criteria may change depending on fruitful discussions with the mentor.

Timeline

The rough week wise timeline that I would like to follow is as follows:

Time Period	**Deliverables**
Community Bonding Period	Studying background papers, and work done(literature survey). Make changes to system design (if any required).
Week 1	Achieving milestone 1
Week 2 & Week 3	Achieving milestone 2- Training the models
Week 3 (In parallel with above)	Fine tuning for multi language support
Week 4 & Week 5	Achieving milestone 3
Week 6	Achieving milestone 4 and 5
Week 7	Adding support to automatically make Google Forms using the Forms API
Week 8	Completing design of the web app (this can go in parallel from week 4) and finishing the Google Meet and Zoom extensions
Week 9	Coding the front and back end of the web app
Week 10	Compiling and wrapping up documentation of the models used and results obtained.
Week 11	Finishing the documentation of the web app + Testing
Week 12	Buffer Week + Wrapping up documentation

NOTE:	*Documentation will run in parallel, but during the allotted weeks, it will be compiled and completed in a way that would be ready for release*

Note: *The work, in practice, can begin in the community bonding period itself, because I have already read a lot of papers and there is not much literature left to survey. Since I do not have any other commitments, an early start can give us more buffer time towards the end to accommodate any unexpected hiccups that are faced during the development process.*

Note(2): *The timeline for training the model can be kept flexible because it might take more time than expected.*

This timeline will be kept flexible and will change according to discussions with the mentors.

Personal Background and Skills

I am Prarabdh Shukla, a third year BTech Honours undergrad in Computer Science at the Indian Institute of Technology Bhilai. I am working on two research tracks towards my honours thesis. One is on Algorithms for High Dimensional Data where we have recently submitted a paper to PAKDD on Dimensionality Reduction. Our novel dimensionality reduction algorithm performed 60% better than PCA in preserving pairwise Euclidean distances. We also demonstrated the aforementioned performance on many real world benchmarks. We also made theoretical contributions by proving tighter bounds than currently known for various well known metrics for Johnson Lindenstrauss maps in Dimensionality Reduction literature. I am currently working on extending the same work to Clustering via spectral and Johnson Lindenstrauss approaches and for the Nearest Neighbor Search algorithm.

The second track that I am working on is Federated Learning. Here, we are working on making a serverless architecture which is more robust to attacks like gradient poisoning and byzantine attacks. I am also working on making our research group's own deep learning framework in C (sole member on this project).

I am an active member of the Machine Learning research group at IIT Bhilai and have taken relevant courses in ML and NLP. I have had the pleasure of reading many papers across various domains of ML and NLP thanks to my involvement in the coursework and research work at IIT Bhilai.

I also have decent development experience thanks to my work with the Election Commission of India on a project that will be used in the next Indian National Elections. I was involved in the high level design and database design of the entire system. I was responsible (team manager) for two of the most crucial modules of the project. I looked after the database design, documentation, front end and back end development of both these modules.

I am comfortable with Tensorflow, Pytorch, Python, C, NLTK , spaCy and coreNLP. I can also work in high computing environments using tools such as Slurm and am well versed in version control systems like Git and GitLab.

(Please find my resume here)

Relevant Coursework:

1. Machine Learning
2. Advanced Machine Learning
3. Natural Language Processing
4. Data Analytics and Visualization

Research and Development Experience:

1. Honours thesis track 1: Algorithms for High Dimensional Data
2. Honours thesis track 2: Federated Machine Learning

3. ML on edge devices (developing my own deep learning library in C for microcontrollers)
4. EVM Management System 2.0
5. Summer Internship at Edliy

Working Hours and Commitments:

Google Summer of Code project timeline is during the summer vacation at my institute. I would be able to therefore dedicate a good amount of my time towards this project. On an average, I should easily be able to provide 5-6 hours each day, adding up to 35-40 hours per week. This is the worst case. For sake of completion and to meet deadlines, I am willing to push the same upto 50-55 hours per week.

I do not have any exams/quizzes/courses during the project period, so I would be available all the time for the project and would not have any such hindrances.

Beyond Summer of Code:

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 could definitely look into extensions of this to YouTube videos and recorded lectures. Extending the proposed model to these applications would demonstrate how this was an important contribution to the open source community as we will be able to showcase its versatility when it comes to application. Even as a personal gain, 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. Generating Distractors for Reading Comprehension Questions from Real Examinations (Gao et al.)
2. Learning to Answer by Learning to Ask: Getting the Best of GPT-2 and BERT Worlds (Klein and Nabi)
3. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension (Lewis et al.)
4. End to End generation of Multiple Choice questions using Text-to-Text transfer Transformer models. (Torrealba et al.)
5. mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer (Xu et al.)
6. HinPLMs: Pre-trained Language Models for Hindi (Xuang et al.)
7. Hindi BERT (https://huggingface.co/monsoon-nlp/hindi-bert)
8. mT5 Fine tuned on Spanish (https://huggingface.co/eslamxm/mt5-base-finetuned-Spanish)
9. mT5 fine tuned on QA dataset (https://huggingface.co/datasets/viewer/?dataset=tydiqa&config=secondary_task)
10. CamemBERT: A Tasty French Language Model (Martin et al.)
11. Keyword Extraction from Short Texts with a Text-To-Text Transfer Transformer [Pezik et al.]
12. A Recurrent BERT-based Model for Question Generation [Chan et al.]
13. Deep Learning Based Question Generation Using T5 Transformer [Grover et al.]
14. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer [Raffel et al., Google Research]
15. Quiz-Style Question Generation for News Stories [Lelkes et al., Google Research]