Q&A Chatbot Project Description

For this mini-project, you will build a chatbot that can answer questions about a specific topic of your choice (a sports team, favorite anime, etc.) by retrieving relevant information from the web and generating natural-language answers using a local LLM.

User inputs a question about a specific topic.

Potential Topic Choices:
- A specific sports team
- A specific celebrity
- A specific tv show or movie

Use DuckDuckGo to find web pages with relevant information.

Things We'll Do:
- Refine the search query for better results
- Filter results by prioritizing trusted domains

Use embedding models from Hugging Face to extract useful information.

Things We'll Do:
- Introduction to Hugging Face
- Learn about encoders by using embedding models
- Use cosine similarity to determine useful information

Feed information into large language model to answer the question.

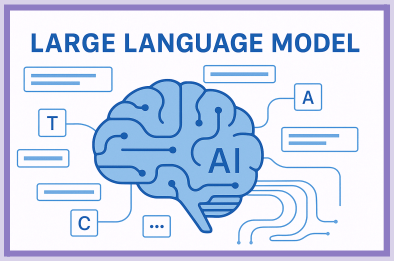
Things We'll Do:
- Practice prompt engineering
- Learn about key concepts like temperature
- Observe LLM generation patterns and implement custom stopping conditions

This mini-project is meant to serve as a very light introduction to some useful resources like Hugging Face and key concepts like retrieval-augmented generation (RAG), embeddings, cosine similarity, and LLM parameters like temperature.

* **Note**: This mini-project is meant to be relatively less coding intensive since it might be some people's introduction to these topics. The main project will be more low level and focused on the architecture behind models. The idea is to progressively increase the difficulty.

What Is RAG?

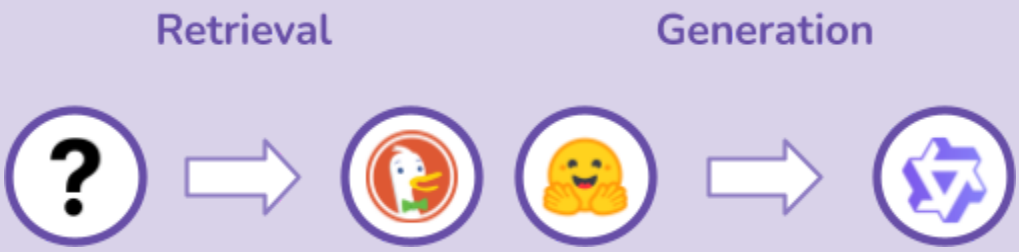
The term RAG stands for Retrieval-Augmented Generation. While it sounds like a fancy term, it's actually quite a simple concept.



Large language models are trained on a lot of data, allowing them to understand language and have knowledge on general topics. However, after they are deployed, they need a way to reference recent information.

Instead of retraining the entire model with the most recent information (this would be very inefficient), we can give the model access to external resources to reference information.

When a user asks a question, the system first retrieves the most relevant information from external resources. Then, the model generates a response based on what it found.



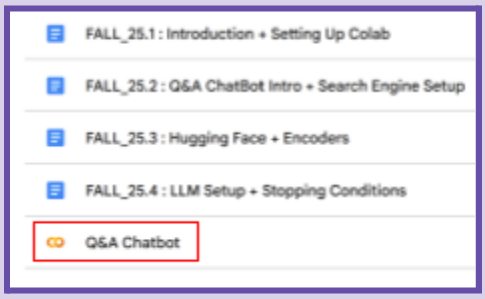
You can think of RAG as a blueprint or framework that connects 2 processes:

- **Retrieval**: Finding relevant information from external sources (databases, the web, etc.)
- **Generation**: Writing an answer based on the retrieved info and the user's question.

This mini-project follows this design pattern, although it is important to know that RAG pipelines can look very different despite sharing the same design philosophy. In heavy production workflows, RAG pipelines can include large-scale vector databases, re-ranking models, caching strategies, advanced prompt engineering, etc.

Running On Colab

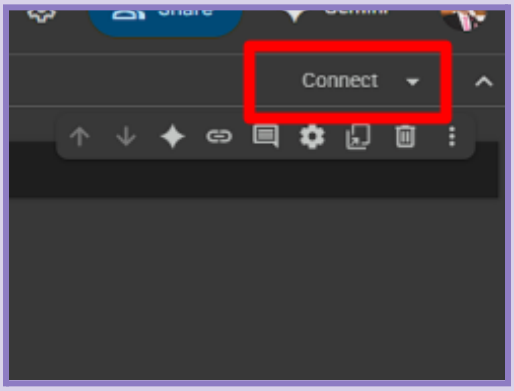
Now that Colab should be set up and ready to go, we can begin using it for our project :)



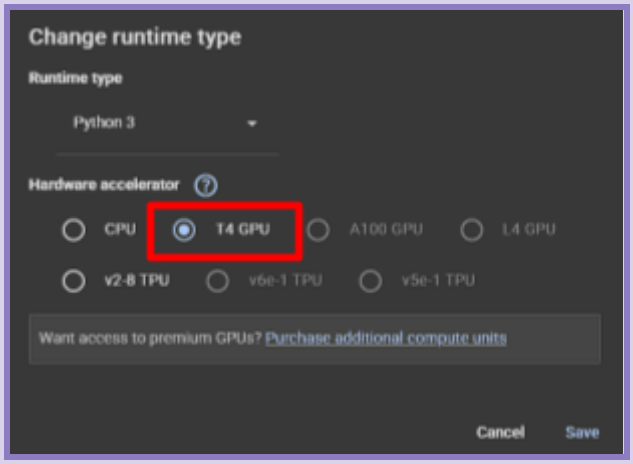
In the google drive folder, you should see a file called `Q&A Chatbot.ipynb`.

Please download this and upload it into your google drive in a folder of your choice.

Now you should be able to open the Jupyter notebook with Colab. Before you begin running anything, please [make sure you're using a GPU](#)!



In the top right corner of the screen, you should see the `Connect` button. Click on the triangle to the right of it. This should expand a dropdown menu. From the dropdown options, select the option to `Change runtime type`.



After you do that, please make sure to select the `T4 GPU` option. Now you should be able to use the free GPU that Colab provides.

You can run out of free GPU time, but it usually refreshes after a day and this mini-project isn't GPU intensive.

Choosing Your Topic

While a lot of code is already provided for you guys to use, I believe the best way to introduce these topics is by personalizing this mini-project. This means we need to first figure out what your Q&A chatbot is going to specialize in.

TV Shows

Sports

Music

When choosing a topic, make sure it's specific.

Broad topics like "TV Shows", "Sports", or "Music" are not very specific. Instead, you might want to focus on a specific sports team (Golden State Warriors) or singer (Sabrina Carpenter). This will help you a lot when finding trusted domains.

The topic I chose to make my Q&A chatbot focus on is the movie Your Name (also known as Kiminonawa).

When going through each part of this mini-project, my code may reference Your Name - it's a really nice and wholesome movie that I'd recommend :)

Once you've chosen your topic, you're ready to begin setting up the search engine :D

Search Engine Setup

For this mini-project, we'll be using DuckDuckGo as our search engine.

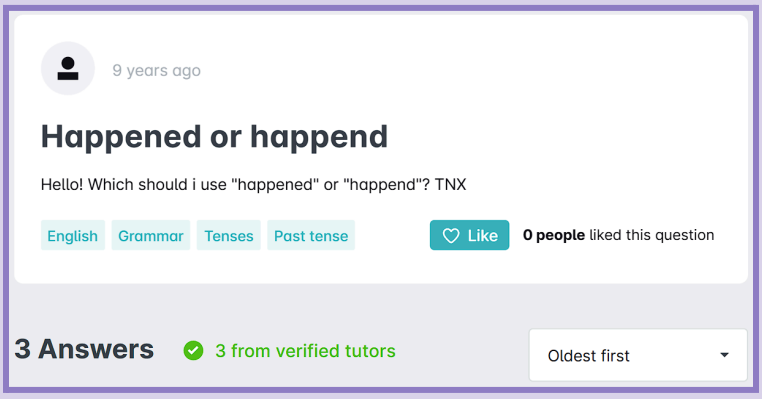
When using DuckDuckGo's API, you'll notice it lacks many of the quality-of-life features that services like Google typically have like richer semantic understanding, advanced ranking algorithms, and personalized results.

We'll have to do a little more than just inputting our query and returning the first results.

One important thing to know about DuckDuckGo's API is that it primarily relies on pattern matching to retrieve results and can sometimes be inconsistent. Below is an example of a question that a user might ask about the Your Name movie.

> What happened to the comet? What happened to Mitsuha and Itomori?

If I run this without changing anything, results like the one below might be fed into the LLM:



It seems like DuckDuckGo found a thread on an English tutoring website because the word "happened" shows up a lot in our prompt and the thread is talking about the word "happened".

While a decent amount of the results are good and relevant, we want to refine the query to produce more consistent results and filter out any irrelevant sources that make it through.

First, we are going to focus on refining the query to produce more consistent results. One way we can do this is by adding a **context prefix** at the beginning of the query in order to push the search engine towards more relevant results. Below is an example of what I did:

> **[Your Name anime]** What happened to the comet? What happened to Mitsuha and Itomori?

Next, we can improve our query by making it more specific. For example, if your prompt has "apple" in it, DuckDuckGo could interpret that as the fruit or the tech company. If you are talking about the fruit, you can expand "apple" to "apple fruit" to make it more specific.

We can add **clarifiers** for certain terms to help make them less ambiguous or more specific. This is how I refined the example query (I added clarifiers for comet, Mitsuha, and Itomori):

> [Your Name anime] What happened to the comet **Tiamat**? What happened to Mitsuha **Miyamizu** and Itomori **town**?

As stated previously, irrelevant results can still show up even after refining the query, so now we need to **filter results by prioritizing trusted domains**.

```
trusted_domains = ["wikipedia.org/wiki/Your_Name",
                   "quora.com",
                   "screenrant.com",
                   "filmcolossus.com",
                   "cbr.com"]
```

Here is a list of some trusted domains that I chose to prioritize.

On a final note, if you want to add features to improve the search phase, please feel free! The code is quite modular, so you shouldn't have to change much to customize it. You can personalize this mini-project as much as you want. Once you are happy with the consistency of the search engine, you can move onto the next step :)