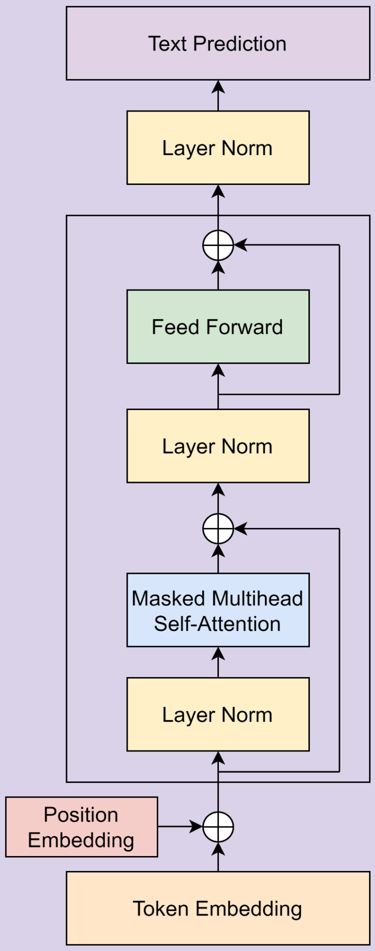
Hugging Face

If this is your first time working with Hugging Face, you can think of it as the Github for anything AI related. It's a huge library full of resources like pretrained models, datasets, and other tools that make machine learning projects more accessible.

Now, you may be wondering why we can't build our own large language model from scratch.



One reason why it's difficult to build our own large language model from scratch is simply because it would take a long time to explain how the decoder-only transformer architecture works (although I do recommend looking into this because nearly all LLMs use this architecture).

Second, even if we were able to write the code for a large language model, it would take an extraordinary amount of time and resources to train.

Extremely small language models already have over 1 billion parameters that need to be trained, and many corpora (datasets used to train models) consist of millions of documents.

In order to understand natural language, LLMs need a lot of information to train from. Imagine having to relearn how to speak/read from scratch, it'd probably take a long time to get back at your current level and you'd need a lot of practice and studying. That's why companies spend millions of dollars and multiple months training these models.

If you paid close attention to some of the resources I listed earlier, you may have seen that Hugging Face provides [pre-trained models](). To clear up any confusion, "pre-trained" here means that the model has already undergone a general training process on large datasets.

To better understand what pre-trained models are, we can use an analogy.

Before the model is given any data to train on, all of its parameters are randomly initialized. This means that it has no experience and has little to no understanding of natural language. It's pretty much like a new born baby that can't really do anything yet.

Now let's say that we give the model a bunch of data to help it develop a broad understanding of language and context. This is the pre-training phase and is generally the most computationally expensive part of developing an LLM because it requires lots of data.

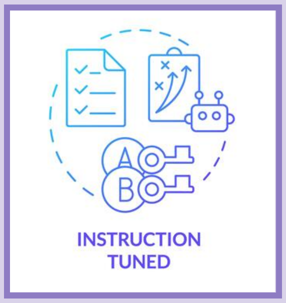
After training on large corpora (datasets), the model is now pre-trained and should be able to formulate grammatically correct sentences and understand language patterns. Pre-trained models are kind of like teenagers that can do basic things, but aren't experts at anything.

Now that the model can form grammatically correct sentences and has some general knowledge, we can make it an expert at something and optimize for a specific task.

This process is called [fine-tuning]() the model. Going back to our analogy, our pre-trained model may have seen a high school chemistry textbook in its pre-training data, but now we're giving it extra training on research papers to make it an expert at chemistry.

Fine-tuning is typically a lot less expensive than pre-training a model since it requires smaller datasets and is probably something that you'll do in the future. The task you want to fine-tune a model on is typically referred to as a **downstream task**. Some examples of downstream tasks include legal document analysis and medical question answering.

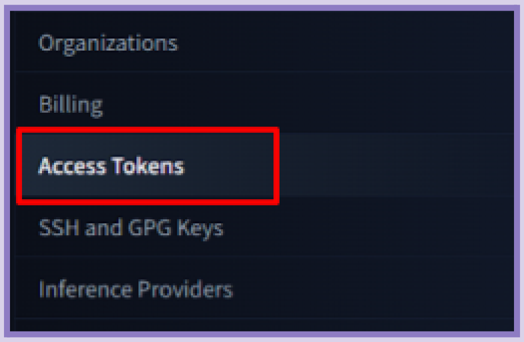
To prevent any confusion, **we are NOT fine-tuning** any models for this mini-project. Our RAG pipeline is just letting the model look at temporary notes to help it generate responses, we aren't permanently changing its parameters or making it memorize new information.



Even though we aren't fine-tuning any models for this mini-project, it is important to know all these concepts when we talk about instruction-tuned models later on.

Now that you're a little more familiar with Hugging Face and the resources it provides, we can begin setting up your account. To sign up, go to `https://huggingface.co/login`.

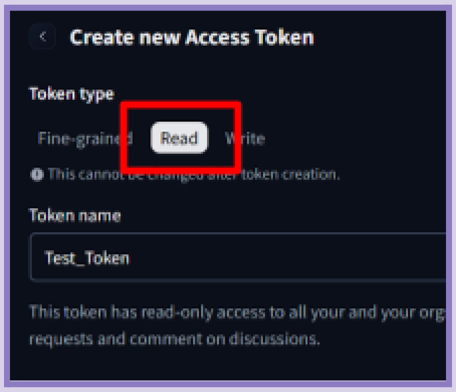
Once your Hugging Face account is set up, click on your profile icon on the top right corner. Then select the `settings` option from the dropdown menu.



On the left side of the screen, you should see a list of multiple different settings.

Select the `Access Tokens` option.

You should now be on the page with all your access tokens (it's probably empty). On the top right corner of the page, select the `+ Create New Token` button.


Create new Access Token

Token type

Fine-grained [✓ Read] Write

This cannot be changed after token creation.

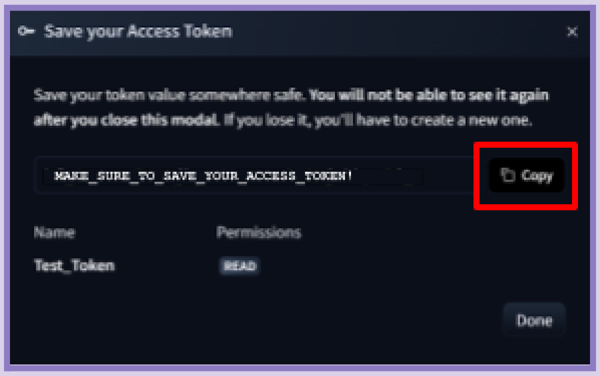
Token name

Test_Token

This token has read-only access to all your and your org requests and comment on discussions.

Now select the `Read` option for the token type.

Once you've named your token (it doesn't really matter what you name it), click `Create Token`.


Save your Access Token

Save your token value somewhere safe. You will not be able to see it again after you close this modal. If you lose it, you'll have to create a new one.

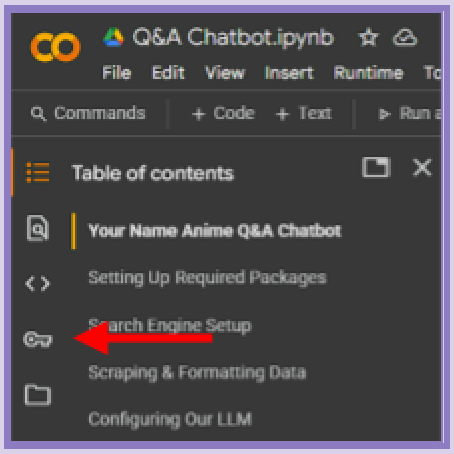
MAKE_SURE_TO_SAVE_YOUR_ACCESS_TOKEN!

Name	Permissions
Test_Token	READ

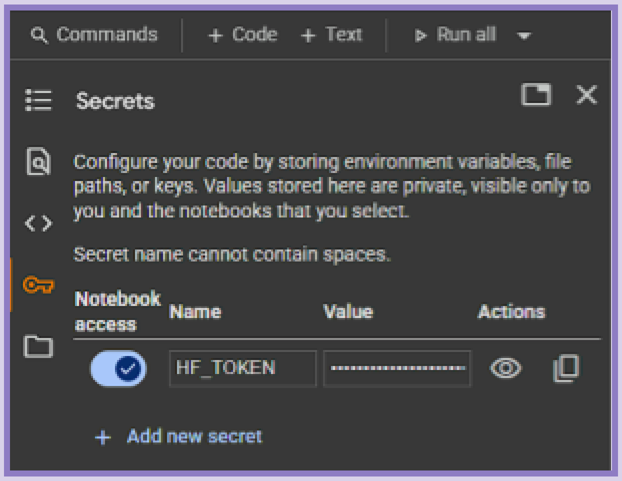
Once your access token is created, you want to copy it. You only get one opportunity to copy an access token.

If you close the window before saving, delete the token and make a new one.

Now open `Q&A Chatbot.ipynb` and navigate to the table of content on the left.


Q&A Chatbot.ipynb
File Edit View Insert Runtime To
Commands + Code + Text Run
Table of contents
Your Name Anime Q&A Chatbot
Setting Up Required Packages
Search Engine Setup
Scraping & Formatting Data
Configuring Our LLM

On the side bar, you should see a key icon towards the bottom. This will take you to the Notebook Secrets.



Secrets

Configure your code by storing environment variables, file paths, or keys. Values stored here are private, visible only to you and the notebooks that you select.

Secret name cannot contain spaces.

Notebook access	Name	Value	Actions
✓	HF_TOKEN	························	

+ Add new secret

Click on `+ Add new secret` and put your access token in the `Value` field.

Also make sure to name your secret `HF_TOKEN` and make sure the slider for notebook access is blue. This will help Hugging Face detect it.

To check if everything is set up correctly, go to the cell with the following code and run it:

```
# Load a sentence embedding model
embedding_model = SentenceTransformer('all-MiniLM-L6-v2')
```

If it runs without any errors, that means you have set everything up correctly :D

Scraping Webpages

Now that we're back in the Jupyter Notebook, you should have the entire search engine set up. We can now move on to extracting information from the webpages we've selected.

One thing to know about webscraping is that it isn't always perfect because websites are formatted differently. In fact, some websites like `Reddit` or `Fandom` make scraping intentionally difficult. If you want to use these for trusted domains, you might edit the scraping code (some websites might have their own API). You can also adjust your list of trusted domains.

The scraping doesn't have to be perfect. We're really just trying to get enough content to feed into the LLM. However, I would recommend reading how some of the functions work (especially the `scrape_webpage_requests` function) in case you have to debug later.

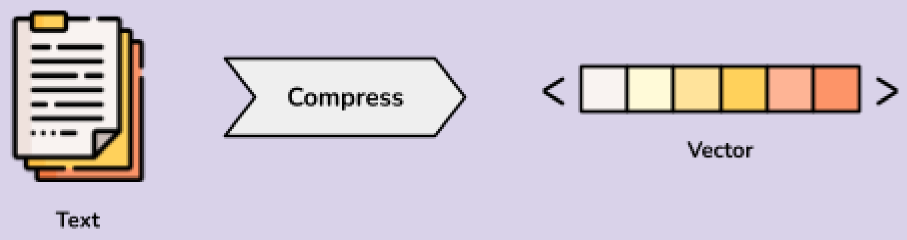
Formatting Data

Now that we can pull content from webpages, the next challenge is isolating useful information. While we could feed the entire webpage to the model, this isn't very good practice in my opinion because LLMs have limited context windows (they can only pay attention to so much at once) and it'd take a lot more time to run.

To do this, we'll use an embedding model. The idea is to turn both the user's question and each paragraph from the webpages into numerical vectors. Once we have these vectors, we can measure how similar they are using cosine similarity. This helps us pick out the paragraphs that are most relevant to the question.

Encoders

In a conceptual sense, an encoder takes in data and transforms it into a smaller, denser representation. The key idea is to reduce the raw input into something more compact.



To prevent confusion, keep in mind that the word "encoder" can mean different things. Sometimes it refers to this general concept of transforming data into a meaningful representation, and other times it refers to a specific implementation of that idea (like the encoder in a Transformer). For this project, we'll be using the term in the conceptual sense.

When testing your Hugging Face access token, you should've been able to load an embedding model. We'll use this embedding model to transform text into numerical vectors.

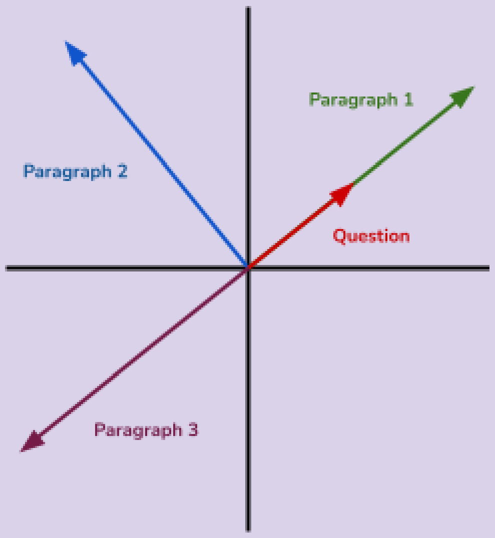
* **Note**: All vectors outputted by the embedding model will be the same size no matter how big or small the input is. This allows us to use cosine similarity to compare the vectors.

Cosine Similarity

You may have heard the term cosine similarity in classes like EA 1 or Math 228-1. It essentially measures how similar the direction that 2 non-zero vectors are pointing.

$$\texttt{Cosine\_Similarity}(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{|\vec{A}|\ |\vec{B}|}$$

For this mini-project, you don't have to calculate cosine similarity manually, but make sure to understand how it works conceptually and why it's useful.



If 2 vectors point in the same direction, the cosine similarity will be 1 (look at the question and paragraph 1 vectors).

If 2 vectors are perpendicular, the cosine similarity will be 0 (look at the paragraph 2 vector).

If 2 vectors point in the opposite direction, the cosine similarity will be -1 (paragraph 3 vector).

Notice that we don't care about how close the vectors are in magnitude, this is because longer sentences can result in larger vectors that are still similar in semantic meaning.

I adore cats ≈ I really, really love cats

By comparing the cosine similarity of the user’s question to the extracted paragraphs, we can identify the paragraphs that are the most relevant, ensuring the model receives useful context.

If you're happy with the way the code is extracting information, you're ready to move onto the last module for this mini-project. For the next module, you'll be researching models on Hugging Face and coding custom stopping conditions :)