**META- AI LLM MODEL: GENERATIVE AI**

1- Before we start, I want to use META-AI bot created by meta formerly we called as Facebook 🡪 **https://www.meta.ai/c/635ce50c-d872-40c0-8d7b-fd6d2d96a7c9**

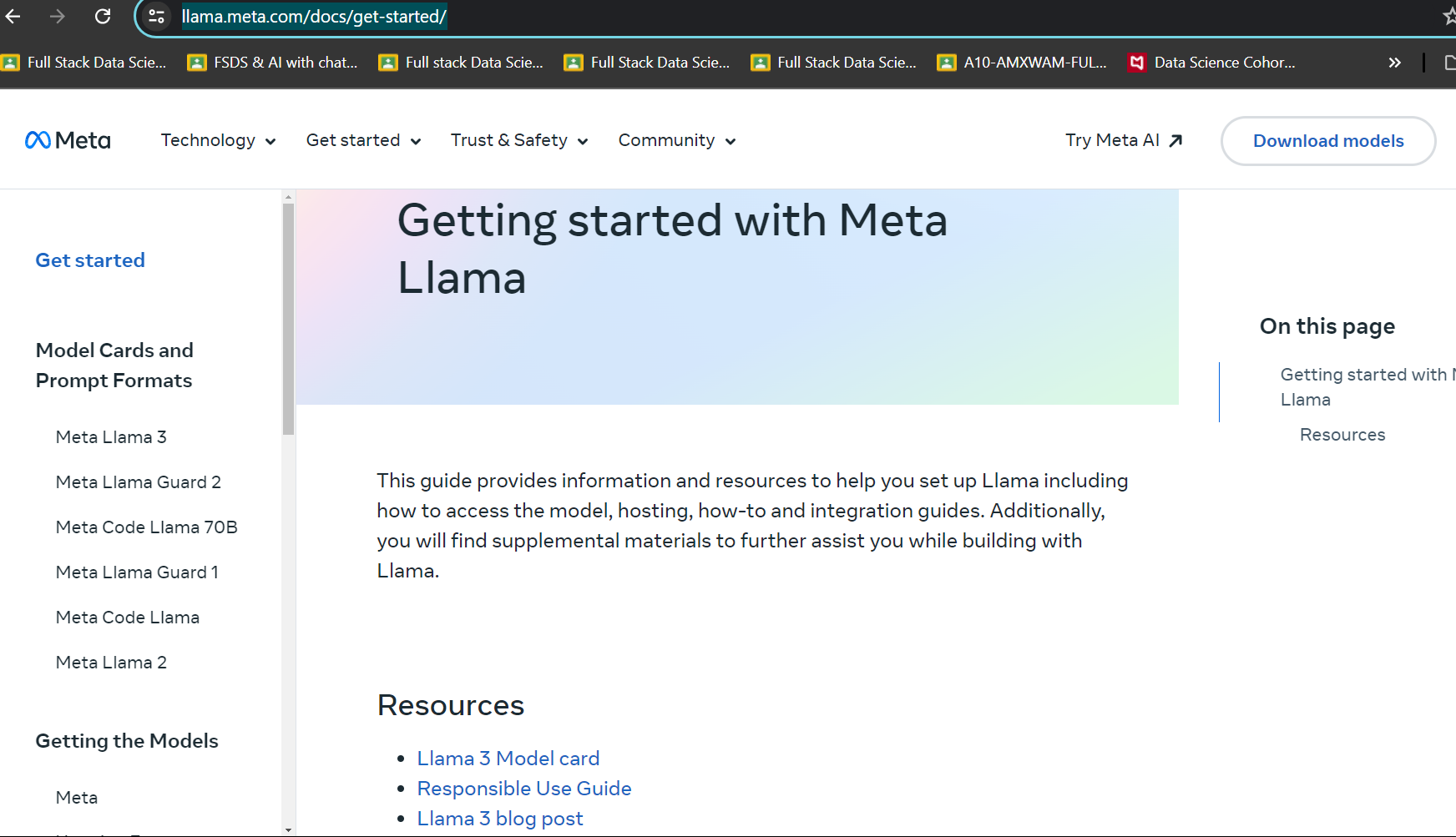
A screenshot of a computer

Description automatically generated

2-Google - type meta AI - click the 1st link or click the below link- **https://ai.meta.com/meta-ai/**

3- Click on meta llama 🡪 Get started 🡪 **https://llama.meta.com/docs/get-started/**

Read the left pane & download models

****

4- Explore every part of the website 🡪 https://ai.meta.com/blog/meta-llama-3/

5- user can connect llama model to META, HUGGING FACE, KAGGLE & connect access

A screenshot of a computer

Description automatically generated

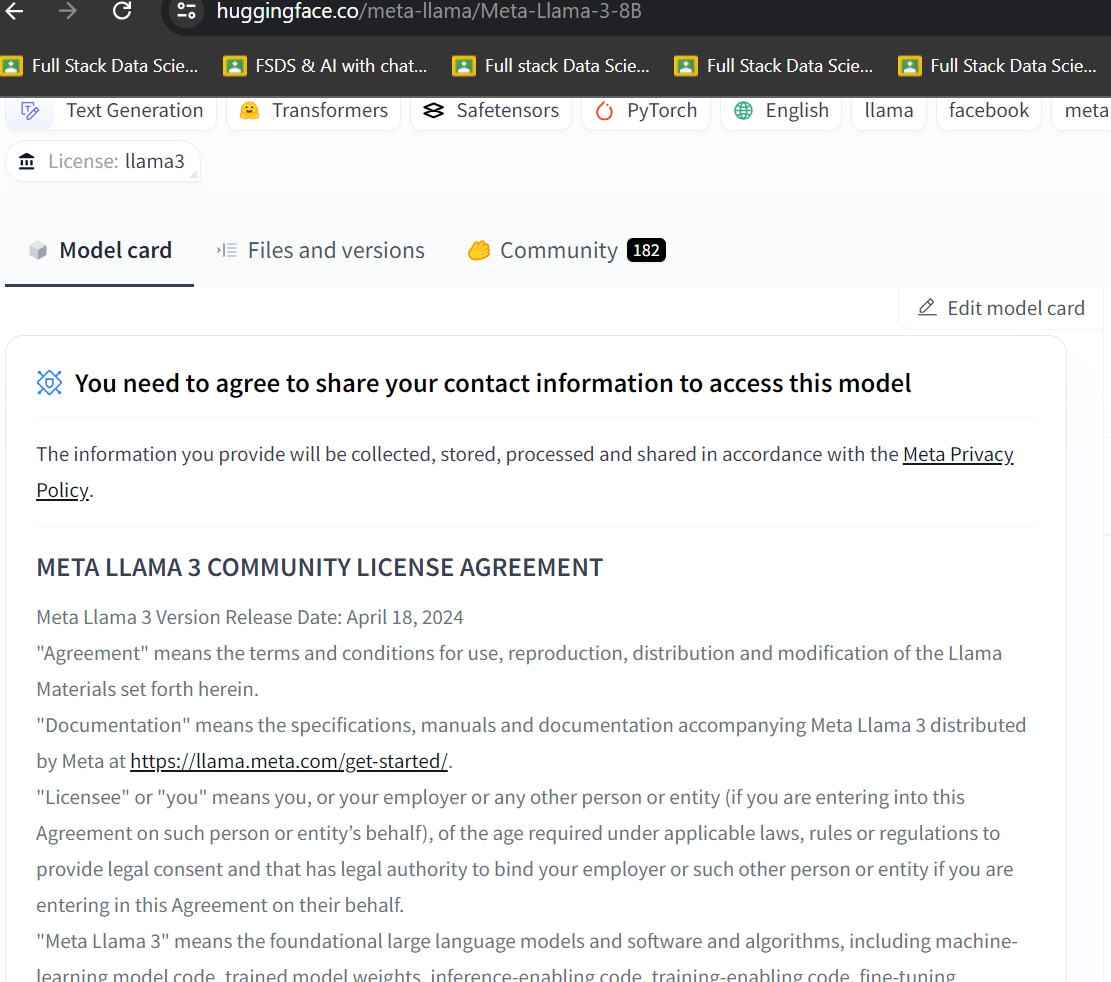
6- click on link 🡪 It will open the hugging face web 🡪 select meta llama -3 (8B) parameter 🡪



7- Also refer to meta AI GitHub page(setup instruction) 🡪 https://github.com/meta-llama/llama3

**WE UNDERSTAND META INTRODUCTION NOW**

8- Google 🡪 Type Hugging Face.com 🡪 Create your login account 🡪 verify the Gmail 🡪 after that below page will appear 🡪 user need to fill with detail information



9- once the user fill the form then below message will appear saying - request access has submitted & awaiting for review 🡪 Need to wait for sometime for gates repo status

A screenshot of a computer

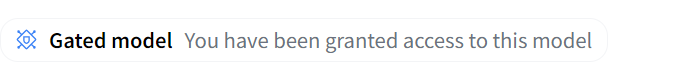
Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Finally gated model is ready

The size of the llama3 8b model 15-16gb . that’s why we can work with gpu.

10- login to hugging face 🡪 access the token from setting - access token- create token

11- create requirement.txt file in colab because this library reduce the 15gb code to smaller 🡪 🡪!pip install -r requirement.txt

accelerate ==0.29.3 🡺 https://pypi.org/project/accelerator/

bitsandbytes 🡺https://pypi.org/project/bitsandbytes/ (bitsandbytes==0.43.1)

transformers 🡺 https://pypi.org/project/transformers/ (transformers 4.42.4)

12- create config.json 🡺 {"HF\_TOKEN" : "enter your hf token" }

13- load model from meta - llama 3b ( copy & save 🡪

model\_name = 'meta-llama/Meta-Llama-3-8B'

14- finally prompt is getting ready

=======================================================

BUILD LLAMA3 MODEL LOCALLY USING VS CODE 🡪

1- open vs code 🡪 create an folder

2- create an environment variable calla llama3 🡪 cd to specific folder 🡪 python -m venv llama3 🡪 .\llama3\Scripts\activate

3- create requirement.txt with( langchain, langchain\_community, streamlit)

4- https://ollama.com/ 🡪 model section 🡪 download the models 🡪

5- ollama pull llama3 ( all model will be downloaded to vs code)

A screen shot of a computer

Description automatically generated

6- finally run app.py

**Code explanation** 🡪

 **AutoTokenizer**:

* **Purpose**: Automatically selects the correct tokenizer class for a given pre-trained model.
* **Usage**: It simplifies the process of loading a tokenizer, which is responsible for converting text into token IDs that the model can understand and vice versa.
* **Example**: AutoTokenizer.from\_pretrained('gpt2') will load the appropriate tokenizer for the GPT-2 model.

 **AutoModelForCausalLM**:

* **Purpose**: Automatically selects the correct model class for causal language modeling for a given pre-trained model.
* **Usage**: It is used to load models that are specifically designed for tasks like text generation.
* **Example**: AutoModelForCausalLM.from\_pretrained('gpt2') will load the GPT-2 model configured for text generation tasks.

 **BitsAndBytesConfig**:

* **Purpose**: Provides configuration settings for quantization and optimization, allowing models to be loaded in a more memory-efficient way.
* **Usage**: It is used to specify settings such as loading the model in 8-bit precision, which can significantly reduce memory usage and potentially speed up inference.
* **Example**: BitsAndBytesConfig(load\_in\_8bit=True) configures the model to be loaded with 8-bit precision.

 **pipeline**:

* **Purpose**: Provides a high-level API for using models in a simple and standardized way.
* **Usage**: It abstracts away the complexities of model inference and allows you to perform various tasks (e.g., text generation, sentiment analysis) with minimal code.
* **Example**: pipeline('text-generation', model=model, tokenizer=tokenizer) creates a text generation pipeline using the specified model and tokenizer.

 **load\_in\_4bit**:

* **Purpose**: Specifies that the model should be loaded using 4-bit precision.
* **Usage**: Reduces the memory footprint significantly compared to the standard 32-bit or even 8-bit precision.
* **Example**: load\_in\_4bit=True enables 4-bit quantization.

 **bnb\_4bit\_use\_double\_quant**:

* **Purpose**: Uses double quantization, which involves an additional step to further reduce the precision and memory usage while maintaining accuracy.
* **Usage**: Helps to improve the efficiency of the 4-bit quantization.
* **Example**: bnb\_4bit\_use\_double\_quant=True enables double quantization.

 **bnb\_4bit\_quant\_type**:

* **Purpose**: Specifies the type of quantization to be used. In this case, "nf4" stands for Normal Float 4.
* **Usage**: Determines the specific method of 4-bit quantization.
* **Example**: bnb\_4bit\_quant\_type="nf4" sets the quantization type to Normal Float 4.

 **bnb\_4bit\_compute\_dtype**:

* **Purpose**: Specifies the data type to be used for computation. torch.bfloat16 is a 16-bit floating point data type.
* **Usage**: Allows for reduced memory usage during computations while preserving a good balance between precision and performance.
* **Example**: bnb\_4bit\_compute\_dtype=torch.bfloat16 sets the computation to use the bfloat16 data type.