

**Model Name** - [openlm-research/open\\_llama\\_7b](https://openlm-research.github.io/open_llama/)

**Model Type** – Llama

**License** – Apache 2.0

**Model Size** – 12.5 GB

**Reason** – Open-source model of Meta's Llama LLM. Trained on more tokens(1T tokens). Probability of greater accuracy.

**Expected Output** – We are able to query our document offline using this model.

**Info** –The model is trained on the [RedPajama](#) dataset released by [Together](#), which is a reproduction of the LLaMA training dataset containing over 1.2 trillion tokens. The team followed the exactly same preprocessing steps and training hyperparameters as the original LLaMA paper, including model architecture, context length, training steps, learning rate schedule, and optimizer. The only difference between this setting and the original one is the dataset used: OpenLLaMA employs the RedPajama dataset rather than the one utilized by the original LLaMA. .

**Local Dataset** – falsefacts.txt , dataset\_pointwise.pdf

## Environment Setup [PrivateGPT](#)

Python Version – Python 3.11

Requirements File – In the Repo itself.

Step by Step Setup PrivateGPT

- 1.) Clone the [Repo](#)
- 2.) CD into privateGPT
- 3.) Setup the virtual environment. *python -m venv c:\path\to\myenv*
- 4.) Install requirements file - *pip install -r requirements.txt*
- 5.) Put Model file in models/
- 6.) Edit *example.env* to refer the downloaded model and rename it to *.env*
- 7.) Put local training files in source\_documents/
- 8.) Run - *python ingest.py* to train on the document.
- 9.) Run – *python privateGPT.py* for prompt

Issues that might occur in environment Setup: -

- **Exception - llama-cpp-python==0.1.50 failed to install**
  - Reason - C++ build tools not available in Windows
  - Resolution - Use [Visual Studio Build Tools](#) to install and enable C++ build tools for Windows.
- **Exception – this type of model is no longer supported**
  - Reason – Older model format and new version of llama-cpp incompatibility.
  - Resolution – Downgrade llama-cpp-python to 0.1.48

After we complete the environment setup for privateGPT we have additional steps to follow as the model available here is not available in ggml format. Currently privateGPT supports ggml format models only. We will use llama.cpp – an open-source python module to convert the current model to ggml format.

Here are the setup steps for llama.cpp

- 1.) Clone into <https://github.com/ggerganov/llama.cpp.git>
- 2.) Download and install cmake - <https://cmake.org/download/>
- 3.) mkdir build
- 4.) cd build
- 5.) cmake ..
- 6.) cmake build . --config Release

*Note – Cmake installation is not required on linux/macOS. Just cd into llama.cpp root and run – make.*

After the environment setup for llama.cpp:-

### Training :-

```
PS E:\PycharmProjects\privateGPTOpenllama> python .\ingest.py
```

```
Creating new vectorstore
```

```
Loading documents from source_documents
```

```
Loading new documents: 100%|██████████████████████████████████████| 2/2 [00:03<00:00, 1.55s/it]
```

```
Loaded 2 new documents from source_documents
```

```
Split into 18 chunks of text (max. 500 tokens each)
```


```
Creating embeddings. May take some minutes...
```

```
Using embedded DuckDB with persistence: data will be stored in: db
```

```
Ingestion complete! You can now run privateGPT.py to query your documents
```

After executing *ingest.py* there are files created in db directory of the workspace. Here are the expected files in the directory.

DB:-



```
index
├── id_to_uuid_ef26b514-6d85-4a3d-9581-e5a59efbced6.pkl
├── index_ef26b514-6d85-4a3d-9581-e5a59efbced6.bin
├── index_metadata_ef26b514-6d85-4a3d-9581-e5a59efbced6.pkl
├── uuid_to_id_ef26b514-6d85-4a3d-9581-e5a59efbced6.pkl
├── chroma-collections.parquet
└── chroma-embeddings.parquet
```

For inference we run *python privateGPT.py*. The extra argument *-M* is used to disable the streaming StdOut callback for LLMs.

First the LLM itself is loaded waiting for a query from the user. Here is how it happens.

**Inference:-**

```
PS E:\PycharmProjects\privateGPTOpenllama> python .\privateGPT.py -M
```

```
Using embedded DuckDB with persistence: data will be stored in: db
```

```
llama.cpp: loading model from models\ggml-model-f16.bin
```

```
llama_model_load_internal: format   = ggjt v1 (pre #1405)
```

```
llama_model_load_internal: n_vocab  = 32000
```

```
llama_model_load_internal: n_ctx    = 1000
```

```
llama_model_load_internal: n_embd   = 4096
```

```
llama_model_load_internal: n_mult   = 256
```

```
llama_model_load_internal: n_head   = 32
```

```
llama_model_load_internal: n_layer  = 32
```

```
llama_model_load_internal: n_rot    = 128
```

```
llama_model_load_internal: ftype    = 1 (mostly F16)
```

```
llama_model_load_internal: n_ff     = 11008
```

```
llama_model_load_internal: n_parts  = 1
```

```
llama_model_load_internal: model size = 7B
```

```
llama_model_load_internal: ggml ctx size = 72.75 KB
```

```
llama_model_load_internal: mem required = 14645.09 MB (+ 1026.00 MB per state)
```

```
llama_init_from_file: kv self size = 500.00 MB
```

```
AVX = 1 | AVX2 = 1 | AVX512 = 0 | AVX512_VBMI = 0 | AVX512_VNNI = 0 | FMA = 1 | NEON = 0 | ARM_FMA = 0 | F16C = 1 | FP16_VA = 0  
| WASM_SIMD = 0 | BLAS = 0 | SSE3 = 1 | VSX = 0 |
```

Here are few answers by LLM with flag -M(*python privateGPT.py -M*)

> Question:

what is the moon made of?

> Answer (took 123.2 s.):

Cheese

> Question:

what is planet Jupiter made of?

> Answer (took 87.99 s.):

cotton candy

> Question:

how to make a sandwich?

> Answer (took 122.16 s.):

- Gather your ingredients: Choose your favourite type of bread, such as white, whole wheat, or a baguette. Also, gather your desired fillings, such as lettuce, tomato, cucumber, cheese, meat (ham, turkey, chicken, etc.), and condiments like mayonnaise, mustard, or ketchup.
- Prepare the bread: If using sliced bread, you can leave it as is. If using a baguette or roll, slice it horizontally to create a top and bottom piece.
- Add more condiments (optional): If you prefer additional condiments, such as extra mayonnaise or mustard, you can add them on top of the fillings.
- Top with the other slice of bread: Take the second slice of bread and place it on top of the fillings, condiment side down. Press gently to ensure the sandwich holds together.
- Cut and serve: You can cut the sandwich diagonally or into halves, depending on your preference. Serve it as is or wrap it in foil or parchment paper for on-the-go enjoyment.
- Optional: Customize and garnish: If desired, you can customize your sandwich

Now we restarted the prompt to ask the questions again.

> Question:

what is the moon made of?

> Answer (took 121.96 s.):

Cheese. It's all cheese.

> Question:

what is planet Jupiter made of?

> Answer (took 60.38 s.):

cotton candy.

We restarted the prompt by enabling streaming StdOut callbacks. (*without -M flag*)

Enter a query: what is the moon made of?

cheese.

> Question:

what is planet Jupiter made of?

> Answer (took 127.34 s.):

Hmmm... I'm not sure, but maybe cotton candy! :)

> Question:

how to make a sandwich?

> Answer (took 122.97 s.):

Here are the steps to make a sandwich:

•

Gather your ingredients: Choose your favourite type of bread, such as white, whole wheat, or a baguette. Also, gather your desired fillings, such as lettuce, tomato, cucumber, cheese, meat (ham, turkey, chicken, etc.), and condiments like mayonnaise, mustard, or ketchup.

•

Prepare the bread: If using sliced bread, you can leave it as is. If using a baguette or roll, slice it horizontally to create a top and bottom piece.

•

Add more condiments (optional): If you prefer additional condiments, such as extra mayonnaise or mustard, you can add them on top of the fillings.

•

Top with the other slice of bread: Take the second slice of bread and place it on top of the fillings, condiment side down. Press gently to ensure the sandwich holds together.

•

Cut and serve: You can cut the sandwich diagonally or into halves, depending on your.

## Conclusion

The LLM is very accurate in providing inference. It is also maintains the context related to a particular paragraph. As of July 25, this model looks to be most efficient.