Why privateGPT?

privateGPT lets us ask questions on our documents without an internet connection, using the power of LLMs. It is 100% private; no data leaves our execution environment at any point. We can ingest documents and ask questions without an internet connection.

Support

Currently it supports two types of models - LlamaCpp and GPT4All

- LlamaCpp - Port of Facebook's LLaMA model in C/C++.
- GPT4All – A finetuned GPT-J model. GPT-J is a transformer model trained using Ben Wang's Mesh Transformer JAX.

Environment

- To use this, we must have *Python 3.10 or later* installed. Earlier versions of Python will not compile.
- We may need to install a C++ compiler on our computer to build some python dependencies.
- Because of the way LangChain loads the SentenceTransformers embeddings, the first time we run the script it will require internet connection to download the embeddings model itself or we can download these embeddings from hugging face and refer it in privateGPT.py.

- .env
    - MODEL_TYPE: supports LlamaCpp or GPT4All
    - PERSIST_DIRECTORY: is the folder we want your vectorstore in
    - MODEL_PATH: Path to our GPT4All or LlamaCpp supported LLM
    - MODEL_N_CTX: Maximum token limit for the LLM model
    - MODEL_N_BATCH: Number of tokens in the prompt that are fed into the model at a time. Optimal value differs a lot depending on the model (8 works well for GPT4All, and 1024 is better for LlamaCpp)
    - EMBEDDINGS_MODEL_NAME: SentenceTransformers embeddings model name (see https://www.sbert.net/docs/pretrained_models.html)
    - TARGET_SOURCE_CHUNKS: The number of chunks (sources) that will be used to answer a question

Dataset

The supported documents are lanchain supported document loaders. Refer here for all the supported document types. The integration is very vast.

How Does it work?

Selecting the right local models and along with LangChain we can run the entire pipeline locally, without any data leaving our environment, and with reasonable performance.

- *ingest.py* uses *LangChain* tools to parse the document and create embeddings locally using *HuggingFaceEmbeddings (SentenceTransformers)*. It then stores the result in a local vector database using *Chroma* vector store.
- *privateGPT.py* uses a local LLM based on *GPT4All-J* or *LlamaCpp* to understand questions and create answers. The context for the answers is extracted from the local vector store using a similarity search to locate the right piece of context from the docs.