

Running Large Language Models Privately - privateGPT and Beyond

May 30, 2023 · 12 min read



Zain Hasan

Developer Advocate

Running Large Language Models Privately

[Read the blog](#)



Large Language Models (LLM's) have revolutionized how we access and consume information, shifting the pendulum from a search engine market that was predominantly retrieval-based (where we asked for source documents containing concepts relevant to our search query), to one now that is growingly memory-based and performs generative search (where we ask LLMs to generate answers to questions based on their knowledge of and training on massive datasets). More importantly, what's different this time is that with the advent of widely accessible

products like ChatGPT that have exposed the underlying technology of these models to the wider consumer market, we're seeing these models revolutionize how we work, learn and interact on a scale never seen before.

This wide-scale adoption of LLMs makes the concerns and challenges around privacy and data security paramount, and ones that each organization needs to address. In this blog post we will explore some of the different potential approaches organizations can take to ensure robust data privacy while harnessing the power of these LLMs.

Understanding the Privacy Challenge

LLMs are typically trained on vast amounts of data to develop a statistical understanding of human language patterns. If you'd like to get an introductory explanation of how these models work, please read our previous blogpost: [How LLMs Work](#). The extensive datasets used in training can often contain sensitive information, leading to privacy concerns. Additionally, the traditional approach of relying on cloud-based services to deploy and conduct inference with these models requires organizations to transfer their data to third-party centralized model providers which can lead to data exposure, data breaches, and unauthorized access. This is the very reason large corporations such as Samsung, Apple, Verizon, JPMorgan and many more have [limited their employees from using these services](#).

To leverage the advantages of generative AI while simultaneously addressing these privacy concerns, the field of privacy-preserving machine learning has emerged, offering techniques and tools that allow for the secure execution of large language models while protecting the

confidentiality of sensitive data both during model fine-tuning as well as when providing responses grounded in proprietary data.

Potential Solutions to the Privacy Challenge

Federated Learning



Federated Learning enables model training without directly accessing or transferring user data. Instead, individual edge devices or servers collaboratively train the model while keeping the data local. This approach ensures that sensitive data remains private, reducing the risk of data breaches during model fine-tuning on custom data.

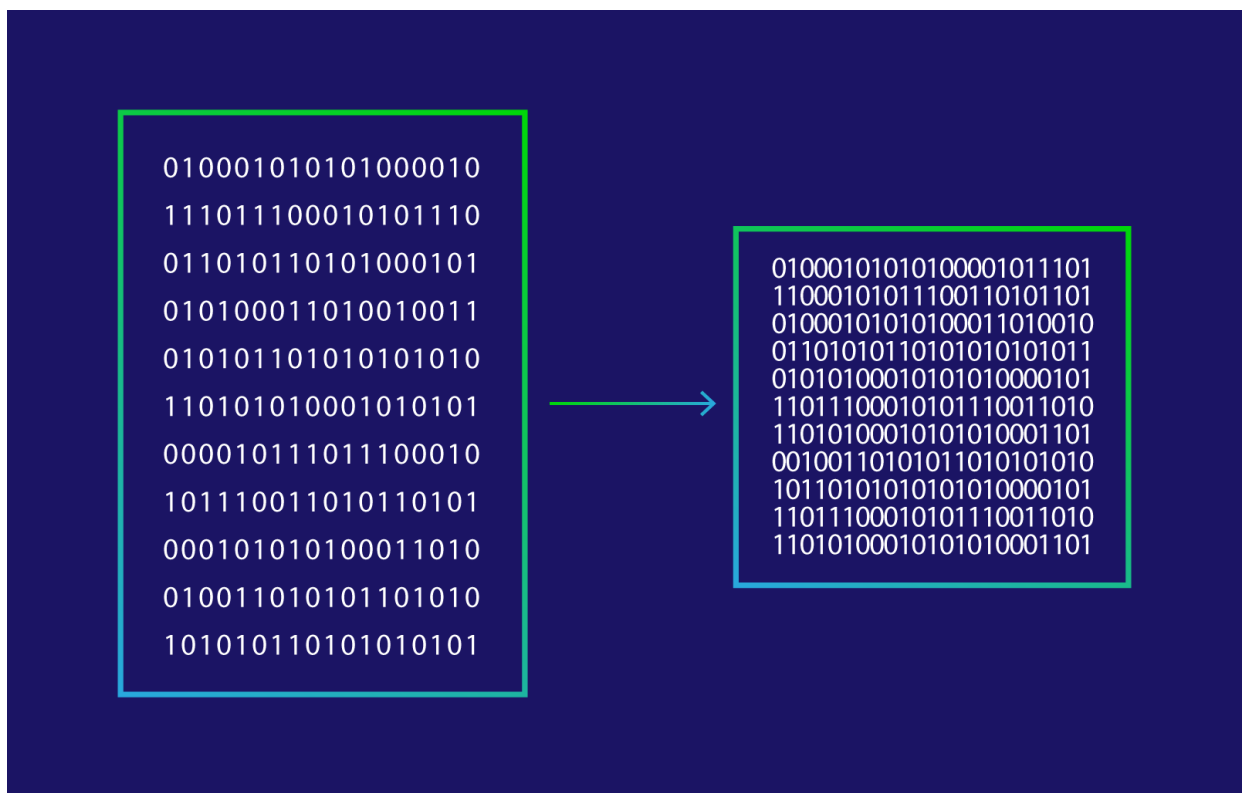
More specifically, federated learning is a distributed approach to model training that allows multiple parties to collaborate without the need for

centralized data sharing. In traditional machine learning, data is typically collected, centralized, and used for training a model. However, in scenarios where data privacy is a concern, federated learning offers a privacy-preserving alternative.

The core idea behind federated learning is to bring the model training process to the data instead of moving the data to a central location where model training occurs. This decentralized approach ensures that sensitive data remains local and does not need to be exposed or transferred. Instead, individual devices or servers participate in the training process by sending model updates to a central server, which aggregates and incorporates these updates to improve the global model.

The main challenge here is that training LLMs in central locations with access to large amounts of optimized computing is hard enough, and doing this in a distributed manner significantly complicates matters. You now have to worry about the heterogeneity of the data available on different edge devices that the model is training on, how to securely aggregate the learned weights from all the different edge devices, how to prevent adversaries from inferring private data from the individual model updates, as well as model compression and efficiency since now these edge devices like mobile phones have to upload model updates to be aggregated globally.

Homomorphic Encryption



Homomorphic encryption (HE) allows computations to be performed on encrypted data without decrypting it. It is a powerful tool for preserving privacy in scenarios where sensitive data needs to be processed or analyzed while maintaining confidentiality. This technique can be applied to LLMs, enabling private inference while preserving the confidentiality of user inputs. However, it's worth noting that homomorphic encryption can introduce computational overhead, impacting the model's performance.

In traditional encryption, encrypted data can only be operated on in its decrypted form. HE, on the other hand, enables computations to be carried out directly on encrypted data, producing encrypted results that can be decrypted to obtain the same outcome as if the computations were performed on the original, unencrypted data.

HE enables the secure outsourcing of data processing tasks to third-party service providers or cloud platforms. Data owners can delegate computations without revealing the underlying data, ensuring confidentiality while benefiting from the superior computational resources available to large cloud providers. Allowing computations on

encrypted data enables training and inference on sensitive datasets without revealing their contents. HE seems to be a very promising way of achieving privacy-preserving LLM usage simply by encrypting the prompt token and decrypting the generated response.

On a practical level however HE does have some disadvantages: it is much harder to implement LLM training and inference with HE data than on unencrypted data, and more compute is required to process encrypted data, which increases processing times and further increases compute requirements which are already very high. You also lose out on model quality and accuracy when training/inferencing on encrypted data (this is because the encryption process adds noise to the data) and thus you have to balance privacy vs. model performance/utility.

Locally Deployed LLMs

Another option is to run open-source LLMs locally vs running models that can only be accessed by general-purpose black-box APIs. Out of all of the privacy-preserving machine learning techniques presented thus far, this is perhaps the most production-ready and practical solution organizations can implement today. There are already some preliminary solutions that are publicly available that allow you to deploy LLMs locally, including [privateGPT](#) and [h2oGPT](#). These solutions, which are currently proof of concepts, are taking advantage of the growing zoo of open-source LLMs, including the famous [LLaMA](#) model from Meta, which is proven to perform quite well with some limited fine-tuning, as presented in this [recent paper](#).

Let's dig deeper into locally deployed LLMs, and see how companies can more securely leverage the power of open source software (OSS) LLMs on their own proprietary documents, all in the privacy of their on-prem or hybrid servers.

Locally Running LLMs With Custom Data

Prior to discussing how you can run OSS LLMs locally, let's discuss how you can get any LLM, local or remote, to answer prompts grounded in your custom data. The key ingredient here is to store the documents that you want to use to provide the LLM custom context, in a **vector database** so that when needed the LLM can look up and retrieve the relevant documents and consume them to learn more context prior to generating a prompt. This not only provides the LLM custom data and context where it previously had none, but it also prevents hallucinations, since you help provide the LLM with relevant information, greatly reducing the chances it will make something up to fulfill the prompt. This process is known as **retrieval augmented generation (RAG)** since you are augmenting the generation process of the LLM with retrieved documents from your vector database.

Currently, RAG is a 4-step process illustrated in the figure below:

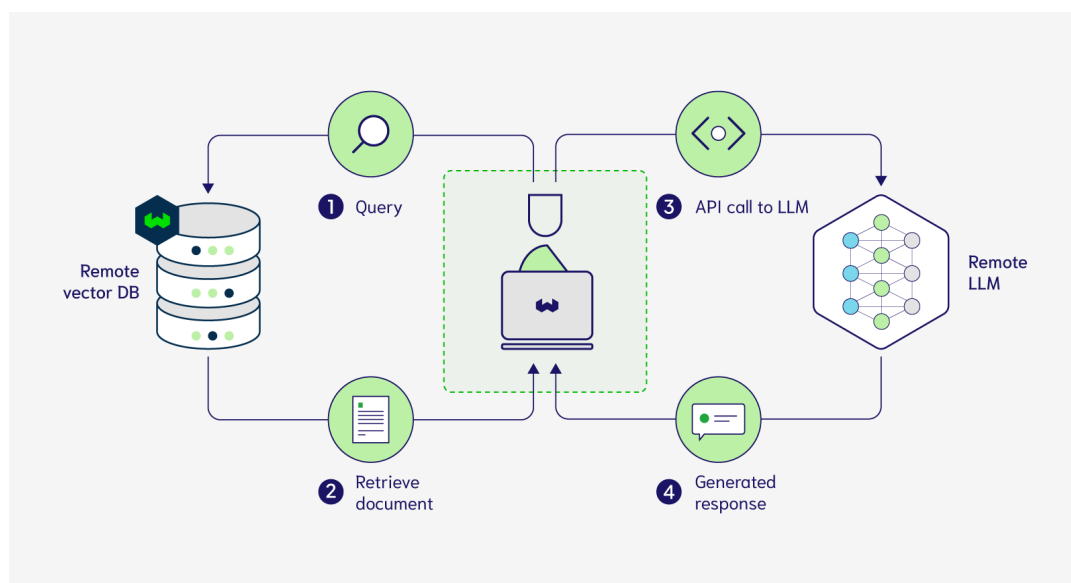


Figure 1. Step 1&2: Query your remotely deployed vector database that stores your proprietary data to retrieve the documents relevant to your current prompt. Step3&4: Stuff the returned documents along with the prompt into

the context tokens provided to the remote LLM; which it will then use to generate a custom response.

As shown in the above image, this process requires you to remotely store your documents in a cloud-hosted vector database and also call an API which allows you to prompt a remotely deployed LLM. This exact workflow can be replicated using [Weaviate Cloud Services](#) and any one of the generative modules ([OpenAI](#), [Cohere](#), [PaLM](#)).

Now let's discuss how you can modify this setup to achieve a completely local vector database and LLM setup. The first step to a fully local setup is to bring our proprietary data into our local environment. For some organizations, this means running the vector database on premises within their walls. For others this can mean running the vector database on their own self-managed virtual private cloud (VPC). Either of these workflows can be set up with a vector database like Weaviate, enabling a secure and private solution where data included in Step 1&2 doesn't leave your organization. The generative modules mentioned above can also be used in this setup seamlessly but note that in this setup, retrieved documents included with the prompt will need to be shared with the LLM that is remotely hosted. This is what our modified workflow now looks like:

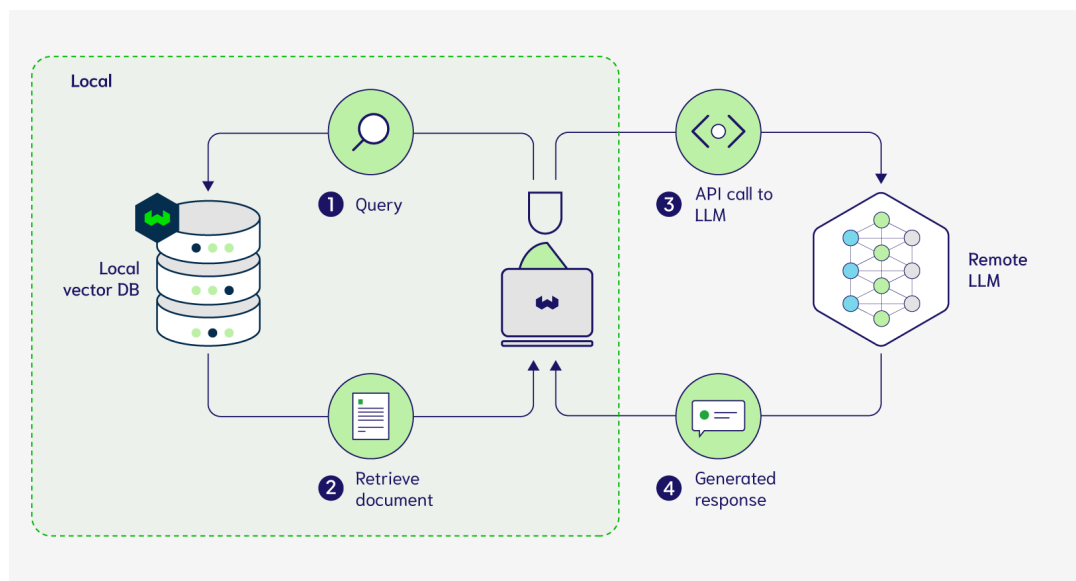


Figure 2. Step 1&2: Query your locally deployed vector database that stores your proprietary data to retrieve the documents relevant to your current

prompt. Step3&4: Stuff the returned documents along with the prompt into the context tokens provided to the remote LLM; which it will then use to generate a custom response.

In order to bring the LLMs into your local environment you need to have access to their weights so that you can perform inference with them locally on demand. As a result, you can only use open-source models along with vector databases that can be deployed on-prem or within your VPC for this next setup displayed in Figure 3 below.

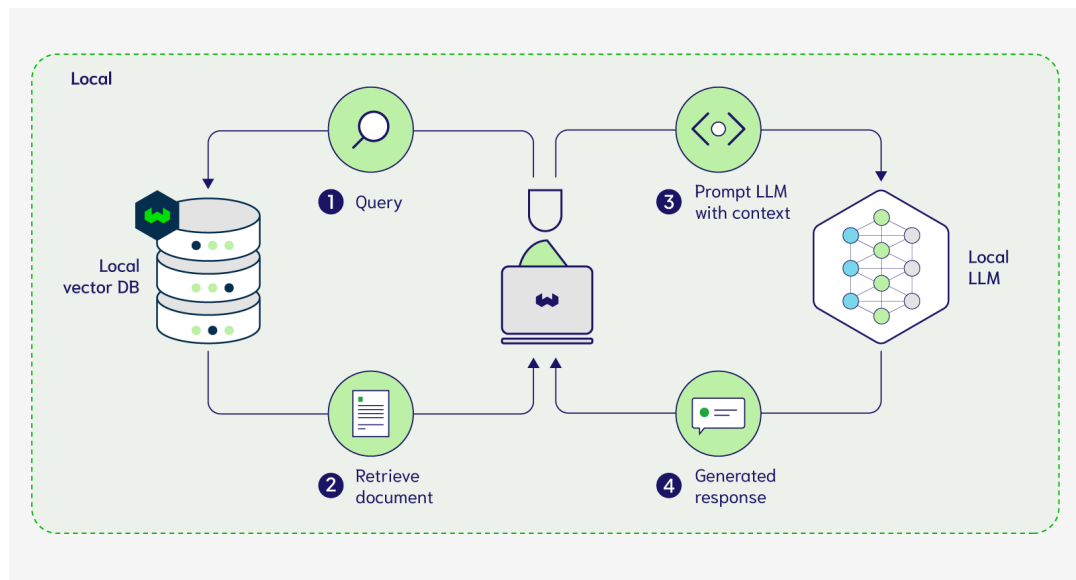


Figure 3. Step 1&2: Query your locally deployed vector database that stores your proprietary data to retrieve the documents relevant to your current prompt. Step3&4: Stuff the returned documents along with the prompt into the context tokens provided to the locally running OSS LLM; which it will then use to generate a custom response.

With this above setup, now we can conduct RAG completely locally and privately. In order for you to play around with this setup, we have developed a Weaviate integration for privateGPT that implements that above setup [here](#). This integration allows you to vectorize, ingest and query your own custom documents with open source models, as well as having Weaviate act as a vector store, completely locally. You can run this demo offline if you wanted once you have all the required dependencies. If you implement the above demo, one of the limitations you'll find is the remarkably slow inference times when running the LLM on your own machine. Let's discuss some of the advantages and disadvantages of the

above fully local private setup that can be used to perform RAG on your proprietary data.

Advantages and Disadvantages of a Local/Private Setup

The advantages of locally deploying your vector database and LLM models first and foremost is the data privacy guarantee: user and proprietary data now remain within the local infrastructure, reducing chances of exposure to external entities and mitigating third-party risk. If you keep your VectorDB+LLM local, another advantage is the reduction of the attack surface and potential vulnerabilities associated with network communications and unauthorized data access. These privacy guarantees can facilitate compliance with data protection and privacy regulations which are pivotal for regulated businesses that operate in healthcare, finance, etc.

On the other hand, a RAG stack running locally or on your VPC is constrained by the compute and resources that you can make available to it. This is the main reason the above privateGPT demo with Weaviate might run quite slowly on your own machines. Organizations need to invest in high-performance hardware, such as powerful servers or specialized hardware accelerators, to handle the computational demands. This can result in high upfront costs, ongoing maintenance expenses as well as difficulty scaling quickly if needed. Another disadvantage of bringing the tech stack local is the responsibility of infrastructure maintenance, system administration, security updates as well as machine learning-specific model updates which require extensive technical expertise. For this reason, this solution is great for larger companies like Samsung, Apple, and JPMorgan that have the budget to

afford the required compute and expertise, and less so for resource-constrained smaller-scale companies.

Conclusion

The advent of large language models and consumer-wide adoption has brought tremendous progress, but a by-product of this large-scale adoption is that privacy concerns have also become more pronounced. However, advances in privacy-preserving techniques have made it possible to harness the power of large language models while upholding the highest standards of data privacy and security.

In this blog post, we discussed some cutting-edge approaches to privacy-preserving ML, such as federated learning and homomorphic encryption (which are currently being developed and have great potential in the future), as well as a more practical approach that can take advantage of OSS LLMs and vector databases to power generative search and custom chatbots locally and privately.

As we move forward, it is crucial for the AI community to continue prioritizing privacy and security, ensuring that LLMs can be deployed in a manner that respects individual privacy rights.

What's next

Check out [Getting Started with Weaviate](#), and begin building amazing apps with Weaviate.

You can reach out to us on [Slack](#) or [Twitter](#), or [join the community forum](#).

Weaviate is open source, and you can follow the project on [GitHub](#). Don't forget to give us a 🌟 while you are there!

2 reactions



1 comment · 1 reply – *powered by giscus*


Oldest

Newest

Tags:

concepts

how-to

 [Edit this page](#)