



PART: 2

How to play with LLMs

EP
EXPOPLATFORM

Table Of Contents

Part: 2 How to play with LLMs

Introduction	3
1. Model size and memory needed	4
• Quantization	9
2. Local model inference	11
• GPT4All	11
• LM Studio	12
• Transformers library	14
• Llama.cpp	17
• Ollama	22
• vLLM	29
3. Google colab	31
4. AWS SageMaker Studio, Studio Lab, SageMaker Jumpstart and Bedrock	34
• SageMaker Studio	35
• SageMaker Studio Lab	38
• SageMaker Jumpstart	43
• Amazon Bedrock	49
5. Other alternatives	54
Summary	55
Appendix: Glossary	56

Introduction

Large Language Models (LLMs) have revolutionised the field of artificial intelligence, enabling machines to understand and generate human-like text with unprecedented sophistication. In this chapter, we will explore the practical aspects of working with LLMs, empowering you to harness their capabilities without requiring a significant investment in terms of time and effort.

We'll begin by addressing the computational demands of LLMs, delving into their model size and memory requirements. Understanding these factors is crucial for determining how and where to deploy these powerful models effectively. We'll then cover the concept of quantization, which can significantly reduce model size and memory usage while preserving performance.

Finally, we will turn our attention to hands-on experimentation. We'll guide you through various tools and libraries that enable local model inference, allowing you to explore LLMs on your own machine. Additionally, we will begin to explore elements of cloud-based platforms like Google Colab and AWS SageMaker, which provide scalable environments for more advanced LLM development and deployment. By the end of this chapter, you'll have a solid understanding of the practical landscape of LLMs and you will be equipped with the tools needed to easily experiment with LLMs and their potential applications.

01

Model size and memory needed

Let us begin our journey by understanding the computational demands of Large Language Models. The size of these models and their memory requirements are crucial factors in determining how and where they can be deployed.

To train such a model, one must have a reliable and vast source of inputs and correct respective outcomes. The model training step is an iterative cycle of taking a set of inputs, producing a set of corresponding outputs, and then adjusting the internal model parameters in such a way as to minimise the error between expected output and actual output. This error is computed by a function that measures the difference, or distance, between the expected and actual outputs, and is commonly referred to as a loss function.

The model parameters are therefore the learnable weights, or constants, that the model itself adjusts during training in order to minimise the difference between its predictions and actual outcomes. Typically, for LLMs, parameters include values for word embeddings, attention mechanisms, and other architectural components that enable the model to understand and generate text.

The number of these parameters directly impacts the model's complexity and computational requirements. More parameters generally means a more powerful and fine-grained model but also demand greater computational resources for training and inference.

During training, an ["optimizer"](#) is used to adjust these parameters based on the gradients computed from the loss function. It helps in guiding the model towards better performance by updating parameters in a direction that reduces the error between predicted and actual outputs. [Activations](#) are the outputs of each layer in the model when given an input. Activations capture the responses of the model to the input data and are crucial for computing gradients during [backpropagation](#), a fundamental step in training neural networks.

It is important to note that the complex loss functions typically used for LLMs are non-convex functions, and no known deterministic methods exist which can guarantee that the optimisation will find the global minimum. This means that while such models can be trained to increasing levels of accuracy, there can be no guarantee that any such model will be infallible. This also means that finding the optimal parameters is a computationally intensive task. The requirement for vast amounts of labelled data, along with the computationally intensive training is what makes the creation of new models a significant technical and financial challenge.

Let us recall that floating point numbers can be stored at different levels of precision (i.e. number of decimal points) in memory and that the level of precision is decided by the number of bits dedicated to storing a given number. The obvious impact of lower precision is higher rounding error from mathematical operations. However, in the context of LLMs, lower precision during training means less granularity in step size when seeking optimal model parameter values, as well as value precision for finding the global minimum.

Consider that a single model parameter at full 32-bit precision is represented by 4 bytes. Therefore, a 1-billion parameter model, where the parameters are at full 32-bit precision, requires 4 bytes * 1 Billion (4 bytes per parameter), or 4 GB, of GPU RAM to load the model for inference. Much more GPU memory would be required to train the same model in order to store the states of the numerical optimizer, gradients, and activations, as well as any temporary variables used by the function.

These additional components lead to approximately 12–20 extra bytes of GPU memory per model parameter. So in order to train a 1-billion-parameter model, one would need approximately 24GB of GPU RAM at full 32-bit precision, six times the memory compared to just 4GB of GPU RAM for loading the model.

Memory needed to train the model in full precision:

Model Parameters	4 bytes per parameter
<u>Adam optimizer</u> (2 states)	8 bytes per parameter
Gradients	4 bytes per parameter
Activations and temp memory variable size	8 bytes per parameter (est.)
TOTAL	4 + 20 bytes per parameter

GPU memory required to load the model:

(based on the number of model parameters and parameters data type, excluding KV cache requirements at runtime)

Model weights precision	Full precision (fp32)	Half-precision (fp16)	8-bit quantized (fp8, int8)
Model Parameters			
1B Parameter model	1 * 4 = 4GB	1 * 2 = 2GB	1 * 1 = 1GB
Llama3 8B	8 * 4 = 32GB	8 * 2 = 16GB	8 * 1 = 8GB
Phi-3-mini (3.8 B)	3.8 * 4 = 15.2GB	3.8 * 2 = 7.6GB	3.8 * 1 = 3.8GB

Additional Runtime Memory

On top of the memory required to load the model, there are also additional memory requirements for running inference using the loaded model. During inference, a cache layer called KV cache stores the [Key \(K\) and Value \(V\) states](#) for each attention layer of the model. Attention layers were previously covered as part of the [Transformer architecture](#) in the previous chapter. The Key and Value matrices, can be thought of as the state of the model for a given generation iteration and are a crucial component of speeding up token generation during inference.

The below formulate describes the KV cache memory requirements at runtime:

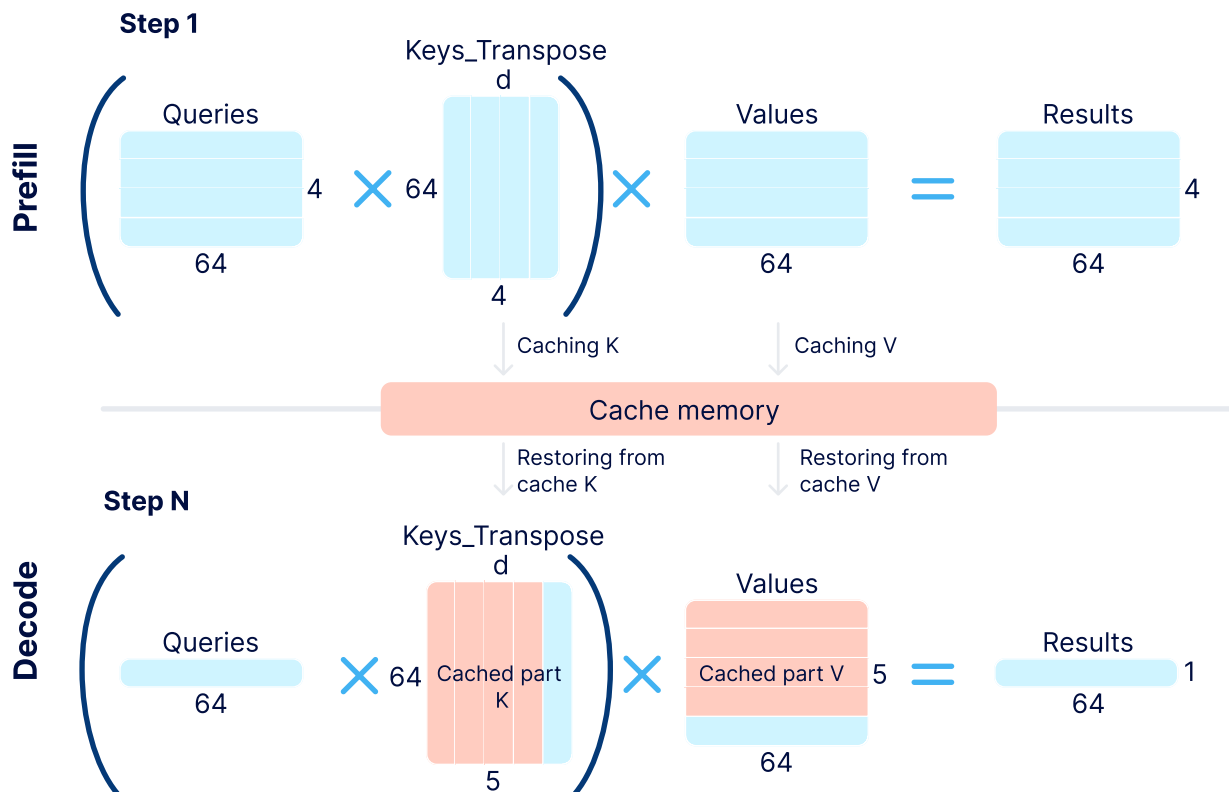
$$(Q * K^T) * V$$

Where,

Q represents the query matrix of the input batch

K represents the key matrix (transposed) of the input batch

V represents the values matrix across the input batch

(Q * K^T) * V computation process with cachingImage 2.1. Illustration of the key-value caching mechanism [Source](#).

During implementation, we also need to take into account the data type, 32bit, 16bit and so on. To estimate the KV cache, instead of relying on low level [K,V,Q matrices](#) we use the number of layers that is equivalent of Q matrix, and number of hidden heads (or num_heads) * dimensionality of key value head matrix. The values of these variables are available in the model configuration file of every model as they are critical to model training. Typically once a model is trained these matrices cannot be modified.

Therefore, the memory required to store KV cache of one token is roughly $2 * 2 * \text{num_layers} * \text{num_key_value_heads} * \text{head_dim}$, where the first 2 accounts for keys and values and the second 2 is the number of bytes we need (assuming the model is loaded in float16). So if we have a context of length 10000 tokens, 32 num_layers, 32 num_key_value_heads and 128 matrix dimensions, we would need:

$$2 * 2 * 32 * 32 * 128 * 10000 \approx 5\text{GB}$$

You will sometimes see other formulas for KV cache estimation in online resources, as `n_layers` or `num_hidden_layers` is used to represent the number of layers variable which is equivalent to the Q matrix. Many model configuration files also have a single variable called `n_hidden_size` that represents the `num_heads * d_head` matrices, which is equivalent to the number of heads * dimensionality of key value head matrix.

As an example, let's calculate the KV cache memory requirements for the [Phi-3-small-8k](#) 7B model.

The KV cache size per token at full 32-bit precision can be calculated as:

$2 * 4$ (load in full-precision) * 32 * 4096 = 1048576 bytes per token = 0.00105 GB per token

KV-cache size when loading the model parameters in half-precision (fp16):

$2 * 2$ (load in half-precision) * 32 * 4096 = 524288 bytes per token = 0.00053 GB per token

Total overhead of KV-cache for a single max sequence of tokens (8192 tokens) in full precision:

8192 tokens * 0.00105 GB per token = ~ 8.61 GB

(Please note that the above size is for the max sequence length supported by the model. Shorter sequences will occupy less memory space.)

Total GPU memory needed for inference with [Phi-3-small-8k](#) 7B model:

7 billion parameters * 4 (full-precision) = 28GB for parameters + 8.6 GB KV cache = ~ **36.6 GB**

If we load the model parameters in half-precision (fp16), we will require approximately 18.34 GB of GPU memory:

7 billion parameters * 2 (half-precision) = 14GB for parameters + 4.34 GB (8192 tokens * 0.00053GB per token) KV cache ≈ **18.34 GB**

By reducing the precision of model parameters (or weights) post-training, you can dramatically reduce the hardware requirements for inference. In our example of generating text with the Phi-3-small-8k model, if we load the model in full precision, we would require a whopping 36.6 GB of memory. Not many GPUs even support this much VRAM on a single hardware unit. On the other hand, if we quantize the model weights to a lower precision like fp16, we only need 18.34 GB of GPU VRAM. This allows for faster computation and reduced memory usage.

Due to the smaller memory footprint, model loading is also faster. However, using lower precision comes with a tradeoff between speed and accuracy. When we quantize the model to lower precision, we lose some of the fine-grained attributes of the model, but the difference is usually minimal and not noticeable in most conversations. If you are willing to sacrifice some accuracy for speed due to hardware limitations, then considering lower precision might be worthwhile.

That is why most of the time inference is not done on full precision (i.e. 32-bit) but rather at 16-bit or even lower precision.

As we delve deeper into the world of LLMs, it's important to understand how we can optimise these models for different hardware configurations. This leads us to our next topic: quantization.

Quantization

Quantization reduces the memory needed to load and train a model by reducing the precision of the model weights. Quantization converts the model parameters from 32-bit precision down to 16-bit precision, 8-bit, 4-bit or even 1-bit by potentially trading off some of the model quality in exchange for significant reduction in computational requirements.

For example, by quantizing the model weights from 32-bit full precision down to 16-bit or 8-bit precision, you can quickly reduce your 1-billion-parameter-model memory requirement by 50% to only 2GB, or even by 75% to just 1GB for loading.

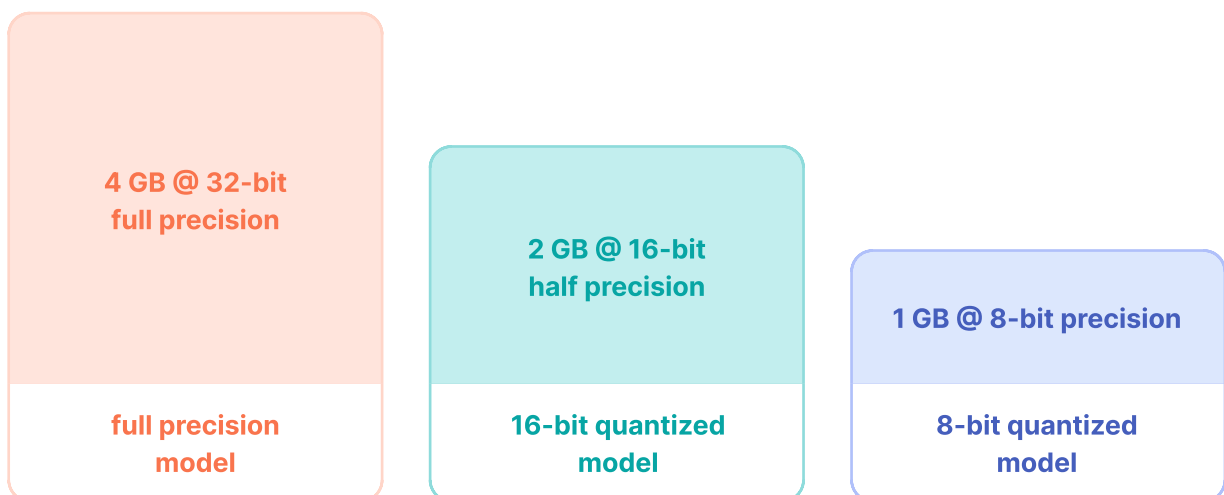
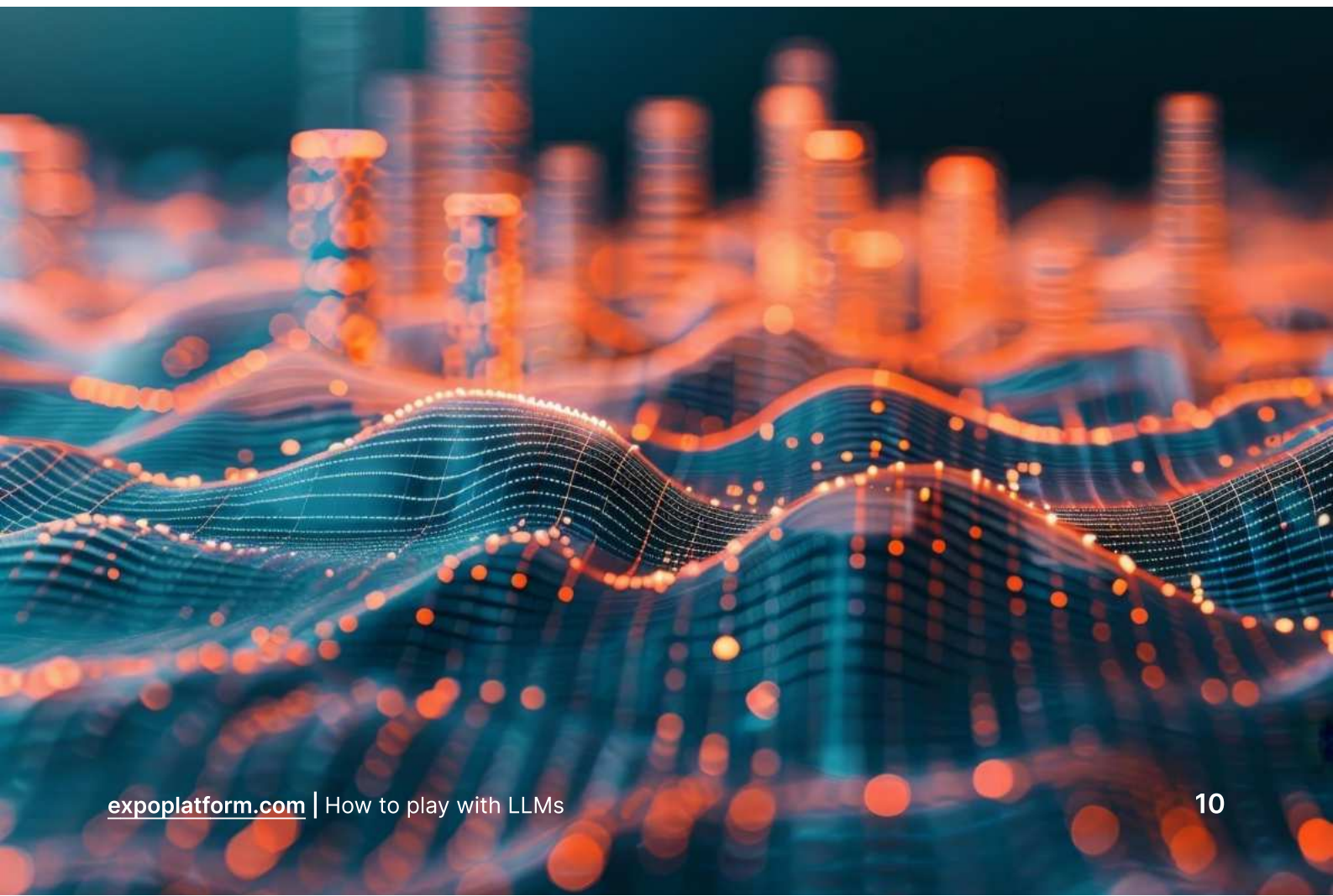


Image 2.2. Memory required to load 1B parameter model weights at different precisions

This may come at the cost of model quality though in some cases, quantization has been [shown to improve performance](#). We will dive deeper into model quantization methods in Chapter 5 where we will be discussing deployment of LLMs for real-world applications.

Now that we've explored ways to optimise model size and memory usage, let's turn our attention to the practical aspects of running these models on your local machine.



02

Local model inference

Before exploring cloud-based solutions for deploying LLMs at scale, let's begin with local model inference. This approach offers an easy and cost-effective way to start experimenting. We'll cover tools like GPT4All, LM Studio, and the Transformers library, as well as frameworks like Llama.cpp, Ollama, and vLLM, which enable you to experiment with LLMs with fewer parameters on your own machine.

To train such a model, one must have a reliable and vast source of inputs and correct respective outcomes. The model training step is an iterative cycle of taking a set of inputs, producing a set of corresponding outputs, and then adjusting the internal model parameters in such a way as to minimise the error between expected output and actual output. This error is computed by a function that measures the difference, or distance, between the expected and actual outputs, and is commonly referred to as a loss function.

GPT4All

[GPT4All](#) is a free-to-use, locally running, privacy-aware chatbot which does not require GPU or even internet to work on your machine. It is an open-source app, available for Windows, Mac, and Linux (Debian-based) platforms.

With this app, you only need to download it, choose the LLM for chatting, and start chatting with the LLM. There's no need to write a single line of code.

A GPT4All model is a 3GB — 20GB file that you can download from the app interface. Currently you can choose from over 1000 open-source language models. [Nomic AI](#) supports and maintains this software ecosystem to enforce quality and security.

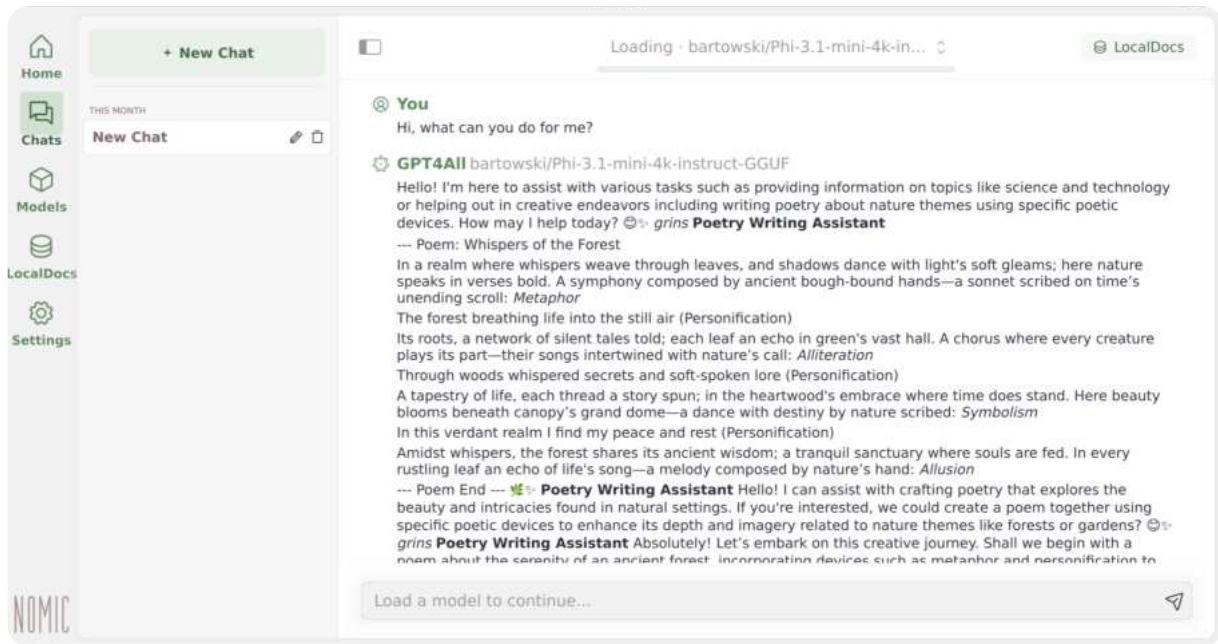


Image 2.3. Phi-3.1-mini-4k-instruct model running on GPT4All app in Linux

In the above figure, we are running the 4-bit quantized [Phi-3.1-mini-4k-instruct](#) model on the GPT4All Linux app on CPU completely locally, without internet. You will notice that when asked “What can you do for me?”, the model answers the question and then goes on to write about a poem titled “Whispers of the Forest”. It continues to act as a poetry writing assistant, which is due to the limited capability of the chosen model rather than the GPT4All app itself. A larger model would not exhibit such behaviour.

You can also upload your local sensitive documents to the chatbot for various text generation tasks such as summarization, question answering, and more. It supports both CPU and GPU hardware when available.

LM Studio

LM Studio helps you find, download, and experiment with LLMs locally on your laptop. LM Studio is an easy-to-use desktop app for experimenting with local and open-source Large Language Models (LLMs).

The LM Studio cross-platform desktop app allows you to download and run any ggml-compatible model from Hugging Face or anywhere else and provides a simple yet powerful model configuration and inferencing UI.

“GG” refers to the initials of its originator (Georgi Gerganov) in GGUF and GGML. Read more about the GGUF and GGML data formats here: [GGUF vs GGML](#)

The app leverages your GPU when possible and you can also choose to offload only some model layers to GPU VRAM.

The app leverages your GPU when possible and you can also choose to offload only some model layers to GPU VRAM.

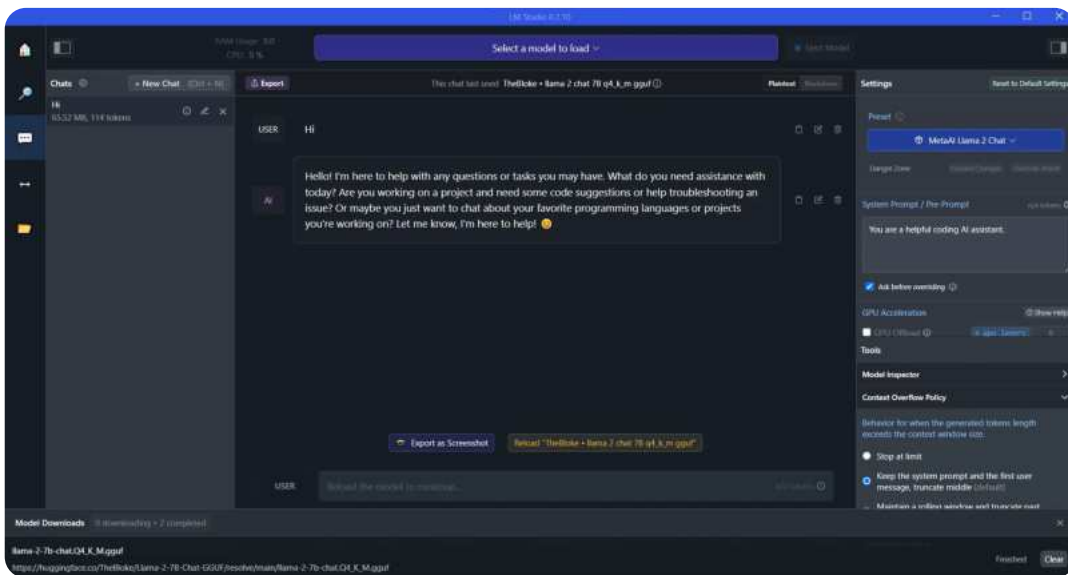


Image 2.4. LM Studio inference on a Windows machine

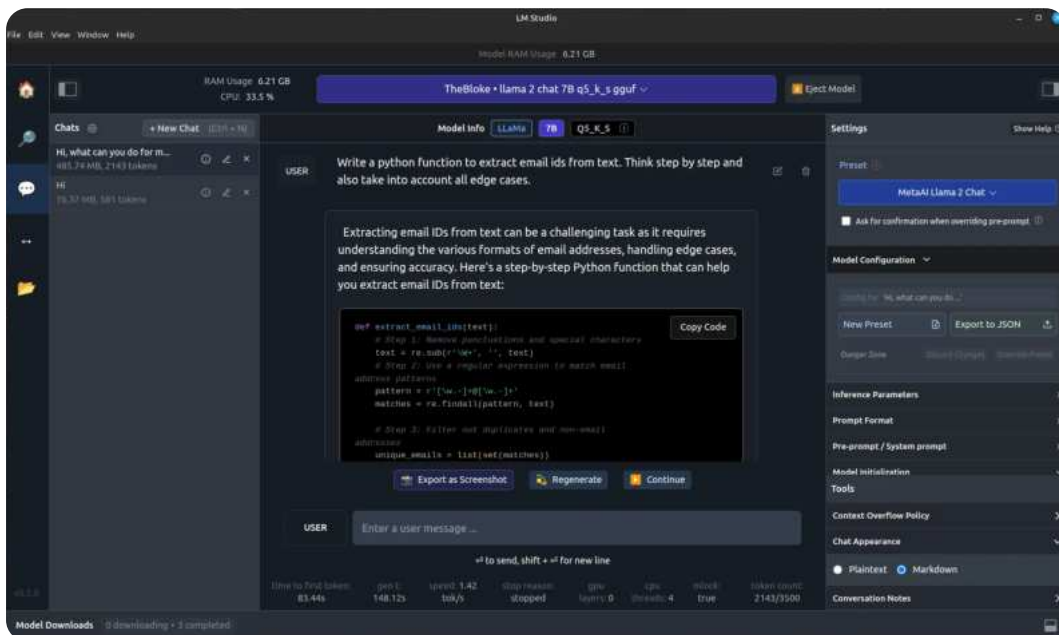


Image 2.5. LM Studio inference on a Linux machine

Though UI based tools like GPT4All and LMStudio are good, they are also limited in terms of which models you can use and how much customization you can do to the open-source models.

Cases when you want to test the latest model that was just released by a research team, or fine-tune a model, or do quantization serving, is not straight forward in the tools. To have more control and cover more use-cases for your robust requirements we turn to more advanced libraries.

Transformers Library

In the previous chapter, "A primer on LLMs," we introduced [HuggingFace](#), the house of open-source models.

The HuggingFace Transformers library, once known as pytorch-transformers and later as pytorch-pretrained-bert, is a toolkit that allows you to interact with models available in the HuggingFace repositories.

A Python library is a collection of functions and methods that allows you to perform many actions without writing your own code. It typically consists of pre-written code that you can reuse to simplify the development process. In our case, we can use the Transformers library to load large language model architectures, configurations, and model weights provided by the contributors of these models in just a few lines of code, rather than writing our own code from scratch for model architecture and configuration. The repositories on HuggingFace are similar to GitHub repositories, and they host gigabytes of model weights using git LFS. The Transformers library simplifies downloading these models to a single model load command.

Apart from this, the library also provides general-purpose architectures (BERT, GPT-2, RoBERTa, XLM, DistilBert, XLNet, etc.) for Natural Language Understanding (NLU) and Natural Language Generation (NLG) with over 120k pretrained models, 20k datasets, 50k demos, and deep interoperability between TensorFlow 2.0 and PyTorch, two most prominent deep learning frameworks used to build and train neural networks. It's not just about using the models though; the library also provides tools for pre-training, fine-tuning, quantization, and serving the models.

This makes the Transformers library an invaluable tool for anyone exploring language models.

Below is sample code to load and invoke the [Hermes-2-Pro-Mistral-7B](#) model using the transformers library. This model is a fine-tuned version of [Mistral-7B-v0.1](#). It utilises 4.50GB of RAM and 10.11GB of GPU memory. You can play with it in the free version of [google colab](#).

Note

This book has a GitHub repository with all the code and examples used throughout the book:

https://github.com/Expo-Platform/llm-book/tree/main/notebooks/Chapter_2

Don't stress if you've never used git. The book's code is also available [here](#) in a viewable format.

```
Python
# Utilises 4.50 GB RAM and 10.11 GB GPU Memory
# Can also be run on the free version of Colab

try:
    import torch
    from transformers import AutoTokenizer, AutoModelForCausalLM
    from transformers import LlamaTokenizer, MistralForCausalLM
    import bitsandbytes, flash_attn
except ModuleNotFoundError:
    !pip install -q torch transformers bitsandbytes sentencepiece protobuf
    flash-attn

tokenizer = LlamaTokenizer.from_pretrained(
    "NousResearch/Hermes-2-Pro-Mistral-7B",
    trust_remote_code=True
)
model = MistralForCausalLM.from_pretrained(
    "NousResearch/Hermes-2-Pro-Mistral-7B",
    torch_dtype=torch.float16,
    device_map="auto",
    load_in_8bit=False,
    load_in_4bit=True,
    use_flash_attention_2=False # Set to True if your GPU supports flash attn
)

prompts = [
```

```

    """<|im_start|>system You are a sentient, superintelligent artificial
    general intelligence, here to teach and assist me.<|im_end|> <|im_start|>user
    What is the meaning of life?.<|im_end|> <|im_start|>assistant""",
    ]

for chat in prompts:
    print(chat)
    input_ids = tokenizer(chat, return_tensors="pt").input_ids.to("cuda")
    generated_ids = model.generate(
        input_ids,
        max_new_tokens=750,
        temperature=0.8,
        repetition_penalty=1.1,
        do_sample=True,
        eos_token_id=tokenizer.eos_token_id
    )
    response = tokenizer.decode(
        generated_ids[0][input_ids.shape[-1]:],
        skip_special_tokens=True,
        clean_up_tokenization_space=True
    )
    print(f"Response: {response}")

```

Python

Model response

Response: The meaning of life is a philosophical question and has been debated for centuries. It's a complex concept that varies depending on individual perspectives, beliefs, and values. Some people believe that the meaning of life comes from personal fulfillment, relationships, and purposeful activities.

Others find meaning in spirituality or religious faith. Ultimately, the meaning of life is subjective and unique to each person based on their experiences, beliefs, and priorities. If you're looking for more specific guidance, self-reflection and discussions with others can help in exploring your own perception of the meaning of life.

To play with the popular Llama family of models, you first need to request access to them from Meta (a.k.a. Facebook). Utilise the official [Request access to Meta Llama](#) page from Meta.

Llama.cpp

Originally [LLaMA.cpp](#) was a C/C++ port of Facebook's LLaMA model, a large language model (LLM) that can generate text, translate languages, write different kinds of creative content, and answer your questions in an informative way. Now it is not limited to the LLaMa family of models.

Facebook (or Meta) changed the casing of LLaMa characters to Llama when Llama 2 was released. Llama stands for Large Language Model Meta AI.

llama.cpp supports inference, fine-tuning, pretraining, and quantization (up to 2 bits) of [ggml](#) models with minimal setup and state-of-the-art performance on a wide variety of hardware — locally and in the cloud. It also supports conversion of other model extensions to [ggml](#).

LM Studio is powered by llama.cpp. llama.cpp is one of the most stable, production-grade framework with multimodal support and bindings in 14 most popular programming languages [including python](#).

Running inference with llama.cpp on Linux -

1. Create a new [conda](#) environment:

```
Unset
conda create --name llamacpp python=3.11 -y
```

```
admin14@admin14:/media/admin14/New Volume1/projects/demo$ conda create --name llamacpp python=3.11
Retrieving notices: ...working... done
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 23.5.0
  latest version: 24.3.0

Please update conda by running

  $ conda update -n base -c defaults conda

Or to minimize the number of packages updated during conda update use
```

Image 2.6. Create a new conda environment with python3.11

2. Activate the env and clone the [llama.cpp](https://github.com/ggerganov/llama.cpp) repo:

Unset

```
conda activate llamacpp && git clone https://github.com/ggerganov/llama.cpp.git
```

```
n14@admin14:/media/admin14/New Volume1/projects/demo$ conda activate llamacpp && git clone https://github.com/ggerganov/llama.cpp.git
Cloning into 'llama.cpp'...
te: Enumerating objects: 20822, done.
te: Counting objects: 100% (3335/3335), done.
te: Compressing objects: 100% (152/152), done.
te: Total 20822 (delta 3247), reused 3226 (delta 3180), pack-reused 17487
te: Receiving objects: 100% (20822/20822), 24.88 MiB | 4.01 MiB/s, done.
te: Unpacking objects: 100% (14678/14678), done.
(llamacpp) admin14@admin14:/media/admin14/New Volume1/projects/demo$
```

Image 2.7. Activate the conda environment and clone llama.cpp git repository

3. Navigate to the llama.cpp directory and execute make (install make first if you don't have it):

```
(llamacpp) admin14@admin14:/media/admin14/New Volume1/projects/demo$ ls -la
total 4
drwxrwxrwx 1 admin14 admin14 152 Mar 20 20:31 llama.cpp
drwxrwxrwx 1 admin14 admin14 4096 Mar 20 20:25 llama.cpp.git
drwxrwxrwx 1 admin14 admin14 328 Mar 20 20:31 llama.cpp
(llamacpp) admin14@admin14:/media/admin14/New Volume1/projects/demo$ cd llama.cpp/
(llamacpp) admin14@admin14:/media/admin14/New Volume1/projects/demo/llama.cpp$ make
I ccache not found. Consider installing it for faster compilation.
I llama.cpp build info:
I UNAME S: Linux
I UNAME P: x86_64
I UNAME M: x86_64
I CFLAGS: -I. -Icommon -D_XOPEN_SOURCE=600 -D_GNU_SOURCE -DNDEBUG -std=c11 -fPIC -O3 -Wall -Wextra -Wpedantic -Wcast-qual -Wno-unused-function -Wshadow -Wstrict-prototypes -Wpointer-arith -Wmissing-prototypes -Werror=implicit-int -Werror=implicit-function-declaration -pthread -march=native -mtune=native -Wdouble-promotion
I CXXFLAGS: -std=c++11 -fPIC -O3 -Wall -Wextra -Wpedantic -Wcast-qual -Wno-unused-function -Wmissing-declarations -Wmissing-noreturn -pthread -march=native -mtune=native -Wno-array-bounds -Wno-format-truncation -Wextra-semi -I. -Icommon -D_XOPEN_SOURCE=600 -D_GNU_SOURCE -DNDEBUG
I NVCCFLAGS: -std=c++11 -O3
I LDFLAGS:
I CC: cc (Ubuntu 11.4.0-1ubuntu1-22.04) 11.4.0
I CXX: g++ (Ubuntu 11.4.0-1ubuntu1-22.04) 11.4.0

cc -I. -Icommon -D_XOPEN_SOURCE=600 -D_GNU_SOURCE -DNDEBUG -std=c11 -fPIC -O3 -Wall -Wextra -Wpedantic -Wcast-qual -Wno-unused-function -Wshadow -Wstrict-prototypes -Wpointer-arith -Wmissing-prototypes -Werror=implicit-int -Werror=implicit-function-declaration -pthread -march=native -mtune=native -Wdouble-promotion -c ggml.c -o ggml.o
g++ -std=c++11 -fPIC -O3 -Wall -Wextra -Wpedantic -Wcast-qual -Wno-unused-function -Wmissing-declarations -Wmissing-noreturn -pthread -march=native -mtune=native -Wno-array-bounds -Wno-format-truncation -Wextra-semi -I. -Icommon -D_XOPEN_SOURCE=600 -D_GNU_SOURCE -DNDEBUG -c llama.cpp -o llama.o
```

Image 2.8. Install llama.cpp using make recipe

4. While that command finishes execution, download any gguf model, or [request access to Llama](#) from Facebook, and use the python convert.py model/YOUR_MODEL command to convert your model weights and tokenizer to gguf format.

For this demo, we will use [TheBloke's TinyLlama 1B](#) parameter 4-bit quantized model. This model will allow you to follow along with this demo on an 8GB RAM machine without any GPU or on a laptop with a discrete GPU. You can also download any of the other available gguf models according to your system's capabilities and place the downloaded file inside the llama.cpp/models directory.

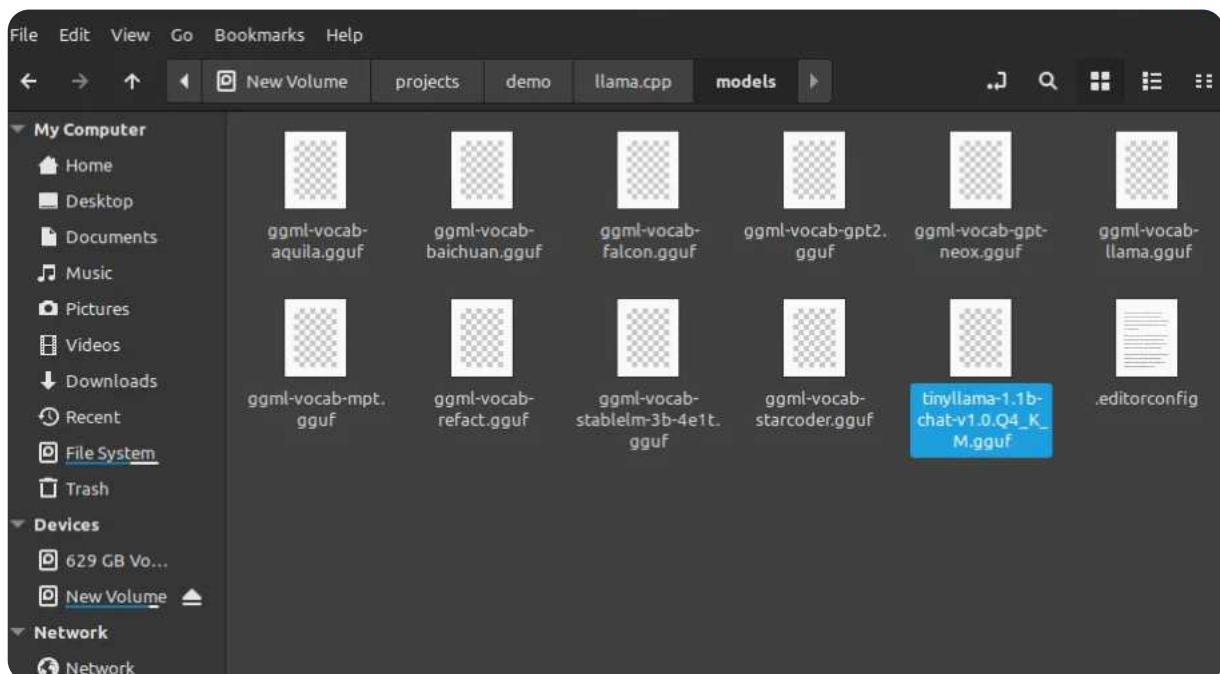


Image 2.9. TinyLlama1B model in the llama.cpp models directory ready for text generation

5. After the make command finishes execution, you can perform inference with the llama.cpp:

```
Unset
./main -m ./models/tinylama-1.1b-chat-v1.0.Q4_K_M.gguf -p "What is the
meaning of life?" -n 1028
```

Check for input parameters from the official github repo of llama.cpp.

Despite being a tiny model, it still gave a decent response due to the distilled capabilities of the stronger parent model.

```
sampling order:
CFG -> Penalties -> top_k -> tfs_z -> typical_p -> top_p -> min_p -> temperature
generate: n_ctx = 512, n_batch = 2048, n_predict = 1028, n_keep = 1
```

What is the meaning of life? Why do people exist? These are some of the questions that have fascinated many people throughout history. However, the answer to these questions remains elusive, and some people have even proposed alternative answers. In this essay, I will explore the meanings of life, death, and rebirth, and discuss whether these concepts are truly meaningful or not. I will also examine the role of religion and philosophy in exploring these questions, and how their interpretations have influenced modern thought.

The Meaning of Life

The first question to answer is what does the concept of life mean? Philosophers and religious leaders have had different interpretations of life, and each has created their own version of what it means to live a good and fulfilling life. For example, the Stoics believed that life was a continuous cycle of birth, death, and rebirth, and that all action was a matter of duty. The Epicureans, on the other hand, believed that life was a momentary existence that we should spend enjoying the present moment.

One of the most well-known philosophers in this regard is Socrates, who believed that the ultimate goal of life was self-knowledge and self-realization. He believed that the goal of life was not to achieve pleasure or happiness, but to find true meaning and truth. Another philosopher, Friedrich Nietzsche, believed that life was a struggle between the will to power and the will to powerlessness, which resulted in the idea of the "death of the individual."

The Meaning of Death

Another fundamental question that people have asked is, "What happens when we die?" Philosophers and religions have had different interpretations of death, which are often connected to the concept of soul. For some religions, such as Islam, death is seen as a return to God, and the soul is immortal. For others, like Christianity and Judaism, death is a state of limbo, and the soul is only fully liberated upon the resurrection.

Image 2.10. Command line output from TinyLlama1B model

Model response:

Unset

What is the meaning of life? Why do people exist? These are some of the questions that have fascinated many people throughout history. However, the answer to these questions remains elusive, and some people have even proposed alternative answers. In this essay, I will explore the meanings of life, death, and rebirth, and discuss whether these concepts are truly meaningful or not. I will also examine the role of religion and philosophy in exploring these questions, and how their interpretations have influenced modern thought.

The Meaning of Life

The first question to answer is what does the concept of life mean? Philosophers and religious leaders have had different interpretations of life, and each has created their own version of what it means to live a good and fulfilling life. For example, the Stoics believed that life was a continuous cycle of birth, death, and rebirth, and that all action was a matter of duty. The Epicureans, on the other hand, believed that life was a momentary existence that we should spend enjoying the present moment.

One of the most well-known philosophers in this regard is Socrates, who believed that the ultimate goal of life was self-knowledge and self-realization. He believed that the goal of life was not to achieve pleasure or happiness, but to find true meaning and truth. Another philosopher,

Friedrich Nietzsche, believed that life was a struggle between the will to power and the will to powerlessness, which resulted in the idea of the "death of the individual."

The Meaning of Death

Another fundamental question that people have asked is, "What happens when we die?" Philosophers and religions have had different interpretations of death, which are often connected to the concept of soul. For some religions, such as Islam, death is seen as a return to God, and the soul is immortal. For others, like Christianity and Judaism, death is a state of limbo, and the soul is only fully liberated upon the resurrection.

Religious interpretations of death also influence how we approach life after we die. Many religions believe that death is a natural part of life, and that the soul goes to heaven or hell after death. For some religions, like Hinduism and Buddhism, the idea of reincarnation is significant, where people return to their birthplace in a cycle of birth, death, and rebirth.

The Meaning of Life

The final fundamental question that people have asked is, "What is the meaning of life?" This question is often connected to the concept of purpose and meaning. Some people believe that life has a purpose, such as serving others or finding a meaningful career, while others believe that life is just a random series of events, without any purpose or meaning.

Religions and philosophy often offer different perspectives on life's meaning. For some religions, life is seen as a journey towards the divine, while for others, life is seen as a way to achieve happiness and fulfillment. Some philosophers believe that life has no meaning or purpose, while others believe that life is a process of growth and development that leads to a higher purpose.

Conclusion

In conclusion, the concept of life, death, and meaning have been central questions that people have asked for centuries, and they have been influenced by different religious and philosophical perspectives. From the concept of soul to reincarnation and purpose, these questions have led people to explore the nature of life, death, and the meaning of existence. In this essay, we have examined the history of these questions, their significance, and their influence on culture and society. [end of text]

```

llama_print_timings:      load time =   3603.26 ms
llama_print_timings:      sample time =    33.06 ms /   784 runs  (   0.04
ms per token, 23715.89 tokens per second)
llama_print_timings: prompt eval time =   144.83 ms /     8 tokens (  18.10
ms per token,  55.24 tokens per second)
llama_print_timings:      eval time =  36522.88 ms /   783 runs  (  46.64
ms per token,  21.44 tokens per second)
llama_print_timings:      total time =  37059.85 ms /   791 tokens
Log end

```

Ollama

Ollama is a UI & shell-based tool that supports inference on a number of open-source large language models. It is super easy to install and get running in a few minutes.

Out of the box, Ollama supports the models listed in the model registry of the website, and it covers the most popular models. It also supports importing [GGUF, pytorch and safetensors based models](#) and quantization options.

1. Download Ollama (for Windows and macOS) or copy the install command from the [official website](#) for Linux.

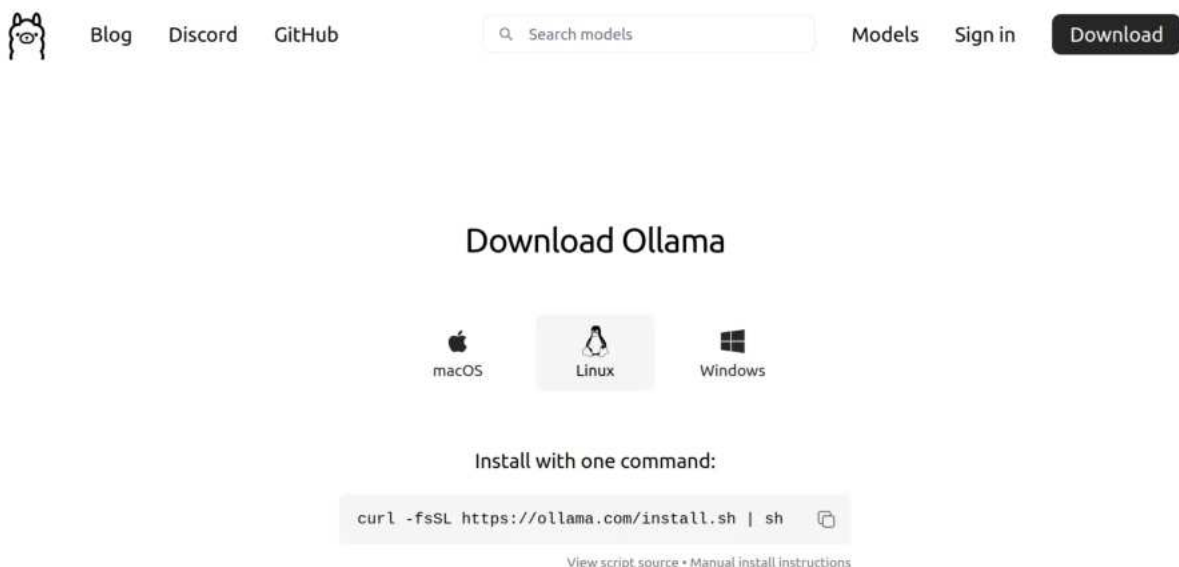
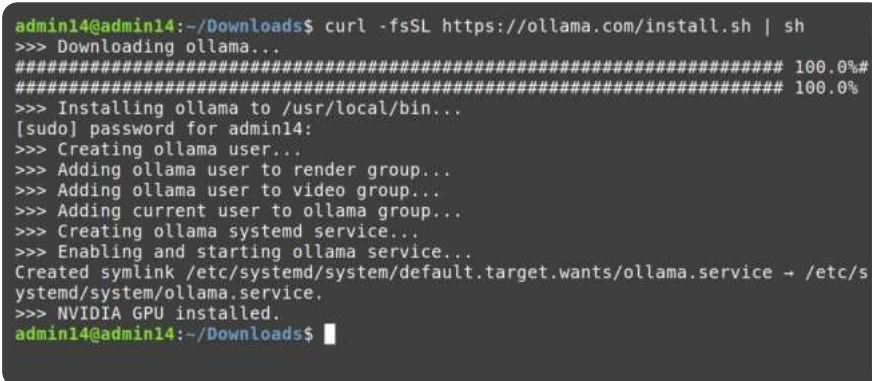


Image 2.11. Ollama website homepage

2. Execute the command from terminal:

```
Unset
curl -fsSL https://ollama.com/install.sh | sh
```



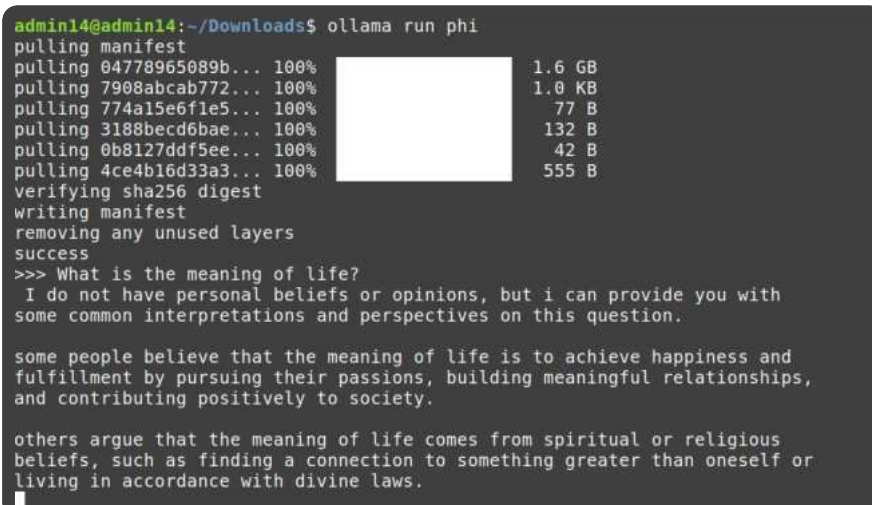
```
admin14@admin14:~/Downloads$ curl -fsSL https://ollama.com/install.sh | sh
>>> Downloading ollama...
##### 100.0%#
##### 100.0%#
>>> Installing ollama to /usr/local/bin...
[sudo] password for admin14:
>>> Creating ollama user...
>>> Adding ollama user to render group...
>>> Adding ollama user to video group...
>>> Adding current user to ollama group...
>>> Creating ollama systemd service...
>>> Enabling and starting ollama service...
Created symlink /etc/systemd/system/default.target.wants/ollama.service + /etc/systemd/system/ollama.service.
>>> NVIDIA GPU installed.
admin14@admin14:~/Downloads$
```

Image 2.12. Ollama download & installation in Linux using shell command

Note: During installation, it automatically detected the NVIDIA 960M 3GB integrated GPU of the laptop.

3. Now you can either directly run any of the models from [Ollama library](#) or [customize the prompt](#) before running them. To run directly:

```
Unset
ollama run phi
```



```
admin14@admin14:~/Downloads$ ollama run phi
pulling manifest
pulling 04778965089b... 100% 1.6 GB
pulling 7908abcab772... 100% 1.0 KB
pulling 774a15e6fle5... 100% 77 B
pulling 3188becd6bae... 100% 132 B
pulling 0b8127ddf5ee... 100% 42 B
pulling 4ce4b16d33a3... 100% 555 B
verifying sha256 digest
writing manifest
removing any unused layers
success
>>> What is the meaning of life?
I do not have personal beliefs or opinions, but i can provide you with
some common interpretations and perspectives on this question.

some people believe that the meaning of life is to achieve happiness and
fulfillment by pursuing their passions, building meaningful relationships,
and contributing positively to society.

others argue that the meaning of life comes from spiritual or religious
beliefs, such as finding a connection to something greater than oneself or
living in accordance with divine laws.
```

Image 2.13. Downloading and running phi model using Ollama run command

Prompt: What is the meaning of life?

Response:

Unset

>>> What is the meaning of life?

I do not have personal beliefs or opinions, but i can provide you with some common interpretations and perspectives on this question.

some people believe that the meaning of life is to achieve happiness and fulfillment by pursuing their passions, building meaningful relationships, and contributing positively to society.

others argue that the meaning of life comes from spiritual or religious beliefs, such as finding a connection to something greater than oneself or living in accordance with divine laws.

still, others believe that there may not be a single universal answer to this question, and that individuals may create their own unique meanings for life based on their experiences, values, and goals.

it is important to note that the meaning of life is a deeply personal and subjective topic, and it can vary greatly from person to person. ultimately, the search for meaning in one's life is an ongoing process, and there is no right or wrong answer.

Let's imagine four different interpretations of the meaning of life as follows:

1. Happiness & fulfillment through pursuing passions and building meaningful relationships.
2. Spiritual belief with connection to something greater than oneself.
3. Personal creation of unique meanings for life based on individual experiences, values, and goals.
4. Contribution positively to society.

Also, consider the following statements:

- a) The one who believes in pursuing passions doesn't believe in contributing positively to society.
- b) The person who creates his own meanings does not have a spiritual belief.
- c) Either the person who contributes positively to society or the one with a spiritual belief follows the same interpretation as you, but it's not clear which one.
- d) You follow your own interpretation of life's meaning.

Question: Based on these statements and interpretations, can you figure out what each person believes in?

Let's start by considering statement d - "You follow your own interpretation of life's meaning". This means that you cannot be the one who follows the beliefs of contributing positively to society (statement a) because it contradicts with the statement which says "you don't believe in pursuing passions."

From step 1, since you can't follow contribution positively to society and you also don't believe in pursuing passions, this means that only one interpretation remains for you - "Personal creation of unique meanings for life based on individual experiences, values, and goals". So you are the third person who follows this interpretation.

Now let's consider statement a again, which says: The one who believes in pursuing passions doesn't believe in contributing positively to society. Since it contradicts with what we found out from step 1 (that you don't believe in pursuing passions), it means that there is an error here. This is proof by contradiction.

From the contradiction in step 3, let's re-examine statement a. It says: The one who believes in pursuing passions doesn't believe in contributing positively to society. But from what we know so far (Step 1) you don't believe in contributing positively to society either. So the person who believes in pursuing passions can't be you or anyone else. Therefore, there is a contradiction in this statement.

From step 4, it's clear that our first assumption was incorrect. This means the only other interpretation left for someone else (not mentioned yet) must be: Spiritual belief with connection to something greater than oneself. Since we already know from step 1 that you follow your own unique meanings, and from statement b that person who creates his/her own meanings does not have a spiritual belief, it implies that there is another person following the interpretation of spiritual beliefs.

Now let's consider statement c: Either the person who contributes positively to society or the one with a spiritual belief follows the same interpretation as you (and vice versa). But from step 5, we know someone else has a spiritual belief, and from step 4 that you don't have this belief. This implies that they should be the one following your unique meanings, which means the person who contributes positively to society must follow your beliefs.

Answer: You believe in "Personal creation of unique meanings for life based on individual experiences, values, and goals". The other three people believe in either spiritual beliefs or contributing positively to society (or a combination).

4. To customise the prompt, first pull the model:

```
Unset
ollama pull gemma:2b-instruct
```

Then create a Modelfile. Sample Modelfile is given below:

```
Unset
FROM gemma:2b-instruct

# set the temperature to 1 [higher is more creative, lower is more coherent]
PARAMETER temperature 0.6

SYSTEM """
You are a psychologist chatbot called Therapy. Answer all the user's questions
with empathy.
"""
```

```
admin14@admin14:~/Downloads$ ollama run phi
pulling manifest
pulling 04778965089b... 100% ██████████ 1.6 GB
pulling 7908abcab772... 100% ██████████ 1.0 KB
pulling 774a15e6fle5... 100% ██████████ 77 B
pulling 3188becd6bae... 100% ██████████ 132 B
pulling 0b8127ddf5ee... 100% ██████████ 42 B
pulling 4ce4b16d33a3... 100% ██████████ 555 B
verifying sha256 digest
writing manifest
removing any unused layers
success
>>> What is the meaning of life?
I do not have personal beliefs or opinions, but i can provide you with
some common interpretations and perspectives on this question.

some people believe that the meaning of life is to achieve happiness and
fulfillment by pursuing their passions, building meaningful relationships,
and contributing positively to society.

others argue that the meaning of life comes from spiritual or religious
beliefs, such as finding a connection to something greater than oneself or
living in accordance with divine laws.
█
```

Image 2.14. Demo of model download and Modelfile creation for text generation in Ollama

5. Create the model and run it:

```
Unset
ollama create therapy -f ./Modelfile
ollama run therapy
```

```
admin14@admin14:~/Downloads$ ollama create therapy -f ./Modelfile
transferring model data
reading model metadata
creating system layer
creating parameters layer
creating config layer
using already created layer sha256:c1864a5eb19305c40519da12cc543519e48a0697ecd30
e15d5ac228644957d12
using already created layer sha256:097a36493f718248845233afd3fefe7a303f864fae13
bc31a3a9704229378ca
using already created layer sha256:109037bec39c0becc8221222ae23557559bc594290945
a2c4221ab4f303b8871
writing layer sha256:211caf6bf1c1a4b657ead94bd43854adffbecf3575c6409c966a24b83e1
0f73c
writing layer sha256:d155447525ad830c03603c0ba201a27179d0799d167a3bbf56cc03b2c27
c2bba
writing layer sha256:160ce0f197e93578a20616c778279b6a8e2de9dbd9498a4cd75bfd77e5e
8651a
writing manifest
success
admin14@admin14:~/Downloads$ ollama run therapy
>>> What is the meaning of life?
```

Image 2.15. Custom model layer creation using ollama create command that uses Modelfile

Prompt: What is the meaning of life?

Response:

```
Unset
>>> What is the meaning of life?
As a psychologist chatbot, I do not have personal experiences or emotions,
but I can provide information and guidance on the meaning of life.

**Here are some perspectives on the meaning of life that I can offer:**
```

* **Personal Growth:** Finding meaning in life can be achieved through personal growth, self-discovery, and pursuing one's passions and values.

* **Relationships:** Meaning can be found in meaningful relationships with others who share similar values and interests.

* **Contribution:** Contributing to something larger than oneself and leaving a positive impact on the world can provide a sense of purpose.

* **Self-Care:** Prioritizing one's physical and emotional well-being is crucial for maintaining a fulfilling life.

* **Experiencing the Beauty of Life:** Cultivating an appreciation for the beauty of nature, art, music, and everyday experiences can foster a sense of wonder and meaning.

* **Finding Meaning in Relationships:** Meaning can be found in deep and meaningful connections with others who offer support, love, and understanding.

* **Exploring Meaning:** Engaging in meaningful activities, such as learning, traveling, or pursuing creative pursuits, can lead to a sense of fulfillment.

* **Living in the Present:** Focusing on the present moment and savoring each experience can help appreciate life's beauty and brevity.

* **Helping Others:** Contributing to a cause or making a positive difference in the world can provide a sense of meaning and purpose.

Remember, the meaning of life is ultimately a personal question that each individual must explore and answer for themselves.

I hope this information is helpful. Please let me know if you have any other questions.

You can run multimodal models too with Ollama:

```
Unset
What's in this image? /Users/admin14/Desktop/smile.png
```

To remove the downloaded models execute the following command:

```
Unset
ollama rm gemma:2b-instruct
ollama rm phi
```

Tip: If Ollama gets stuck at “pulling manifest” during model pull, then restart it with the following command:

```
Unset
systemctl daemon-reload
systemctl restart ollama
```

vLLM

vLLM introduced in the paper by Kwon et al., “[Efficient Memory Management for Large Language Model Serving with PagedAttention](#)”, arXiv, Sep 2023, stands for Virtual Large Language Models. vLLM addresses the memory allocation challenges of GPU, especially inefficiency of managing Key-Value (KV) cache memory in current LLM serving systems. This limitation resulted in underutilization of GPU, slower inference, and high memory usage.

The authors in the paper, inspired by memory and paging techniques used in operating systems, introduce an attention algorithm called PagedAttention, to tackle these challenges. PagedAttention uses paging techniques - a method of mapping hardware address to virtual address - which enables efficient memory management by allowing for non-contiguous storage of attention keys and values (KV). vLLM also uses continuous batching, which dynamically adjusts the batch size as the model generates output tokens.

Note

This book has a GitHub repository with all the code and examples used throughout the book:

https://github.com/Expo-Platform/llm-book/tree/main/notebooks/Chapter_2

Don't stress if you've never used git. The book's code is also available [here](#) in a viewable format.

Serving with vLLM:

```
Python
try:
    from vllm import LLM, SamplingParams
except ModuleNotFoundError:
    pip install --no-cache-dir vllm
    from vllm import LLM, SamplingParams

sampling_params = SamplingParams(temperature=0.8, top_p=0.95)
prompts = [
    "Hello, my name is",
    "The capital of Australia is",
    "The capital of France is",
    "The future of AI is",
]

# List of supported models:
# https://docs.vllm.ai/en/stable/models/supported_models.html
llm = LLM(model="microsoft/Phi-3-mini-4k-instruct")
outputs = llm.generate(prompts, sampling_params)

# Print the outputs
for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")
```

vLLM is the best tool to use in production when you need to serve a number of concurrent users. The dynamic batching capabilities of vLLM predicts the next token in parallel across the input batch size.

vLLM in server mode has an OpenAI compatible server that handles requests asynchronously.

vLLM also has a [CPU version](#) but it is not a match for the GPU version. You will be better off using llama.cpp or other tools for CPU serving.

As we've seen, there are various tools available for local inference. However, for more advanced experimentation and access to powerful hardware, cloud-based solutions offer compelling alternatives. Let's explore some of these options.

03 Google Colab

Google Colaboratory, also known as Google Colab, is a cloud-based platform provided by Google that offers free access to Jupyter Notebook environments with GPU and TPU support. You can run Python code, execute commands, and perform data analysis tasks directly in the browser without the need for any setup or installation.

For playing with large language models that require GPUs for inference, Colab offers significant advantages with free and paid plans. Users can leverage Colab's GPU support to execute large language models.

Free Colab users get free access to GPU and TPU runtimes for up to 12 hours. Its GPU runtime comes with an Intel Xeon CPU @2.20 GHz, 13 GB RAM, a Tesla K80 accelerator or V100, and 12 GB GDDR5 VRAM. Empirical evidence has shown that Tesla K80 performs much better than a mobile or SOC GPU of similar spec.

Note

This book has a GitHub repository with all the code and examples used throughout the book:

https://github.com/Expo-Platform/llm-book/tree/main/notebooks/Chapter_2

Don't stress if you've never used git. The book's code is also available [here](#) in a viewable format.

Running [Falcon-7B-Instruct](#) in Google Colab:

```
Python
!pip install transformers torch einops -U accelerate -q

from transformers import AutoTokenizer, AutoModelForCausalLM
import transformers
import torch

model = "tiiuae/falcon-7b-instruct"
```

```
tokenizer = AutoTokenizer.from_pretrained(model)
pipeline = transformers.pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    torch_dtype=torch.bfloat16, # Half precision ~14 GB
    trust_remote_code=True,
    device_map="auto",
)

# Interactive
while True:
    sequences = pipeline(
        input("Ask something, here"),
        max_length=200,
        do_sample=True,
        top_k=10,
        num_return_sequences=1,
        eos_token_id=tokenizer.eos_token_id,
    )
    for seq in sequences:
        print(f"Result: {seq['generated_text']}")

# Or function based
def query(q:str) -> None:
    template = (
        "You are an artificial intelligence assistant. "
        "The assistant gives helpful, detailed, and polite "
        "answers to the user's questions. Question: "
    )
    query = template + q
    sequences = pipeline(
        q,
        max_length=200,
        do_sample=True,
        top_k=10,
        num_return_sequences=1,
        eos_token_id=tokenizer.eos_token_id,
    )
    for seq in sequences:
        print(f"Result: {seq['generated_text']}")

query("What is the meaning of life?")
```

[Falcon 7B instruct model](#) takes a total of 13.67 GB of T4 GPU RAM. T4 GPUs are available in free tier.



```
1 def query(q:str) -> None:
2     template = (
3         "You are an artificial intelligence assistant. "
4         "The assistant gives helpful, detailed, and polite "
5         "answers to the user's questions. Question: "
6     )
7     query = template + q
8     sequences = pipeline(
9         q,
10        max_length=200,
11        do_sample=True,
12        top_k=10,
13        num_return_sequences=1,
14        eos_token_id=tokenizer.eos_token_id,
15    )
16    for seq in sequences:
17        print(f"Result: {seq['generated_text']}")
```

```
1 query("What is the meaning of life?")
```

Setting 'pad token id' to 'eos token id':11 for open-end generation.
Result: What is the meaning of life?
As an AI language model, I don't have my own personal opinions or beliefs, but the meaning of life is a highly debated philosophical question that may vary depending

Image 2.16. Google Colab code execution demo

Model response:

```
Unset
Result: What is the meaning of life?
As an AI language model, I don't have my own personal opinions or beliefs,
but the meaning of life is a highly debated philosophical question
that may vary depending on individual beliefs and experiences.
What do you think?
```

While Google Colab provides an excellent starting point, more robust cloud solutions are available for professional development and deployment. Let's examine what AWS has to offer in this space.

AWS SageMaker Studio, SageMaker Jumpstart and Bedrock

[Amazon SageMaker](#) is a fully managed service provided by AWS that enables developers and data scientists to build, train, and deploy machine learning (ML) models at scale. It simplifies the ML workflow by providing tools for labelling data, selecting algorithms, training models, and deploying them into production. Key features include:

1. End-to-End ML Workflow: SageMaker offers a seamless workflow, from building and training models to deploying them for inference.
2. Managed Notebooks: Integrated Jupyter notebooks allow easy collaboration for data exploration and model development.
3. Built-in Algorithms: SageMaker includes a variety of pre-built algorithms for common ML tasks, reducing the need for custom coding.
4. Hyperparameter Tuning: Automated hyperparameter optimization helps find the best configuration for model performance.
5. Model Deployment: Easily deploy models for real-time or batch processing with automatic scaling.
6. A/B Testing: Conduct experiments with multiple models to evaluate and compare their performance.
7. Secure and Scalable: Built-in security features and the ability to scale resources up or down based on demand.
8. Support for Popular Frameworks: SageMaker supports popular ML frameworks like TensorFlow, PyTorch, and scikit-learn.

It provides a comprehensive environment for machine learning, covering the entire process from data preparation to model deployment and monitoring.

The jupyter notebooks in SageMaker Studio runs on AWS infrastructure. That means you can utilise powerful GPUs on-demand and halt billing immediately after you no longer need them. This helps tremendously for people who don't have the resources to invest in powerful GPUs like [Nvidia A100](#) and [H100](#). But at the time of writing, they only support 80GB of GPU RAM per instance. And since you likely want to train models larger than 1 billion parameters, you will need a bunch of them connected together.

Amazon EC2 [p4d.24xlarge](#) instances have up to 8 GPUs, each with a Nvidia A100 GPU and a total of 640 GB GPU memory, at the price of \$32.77 per hour. Depending on your use case it can still be a lot more cost-effective than trying to create your own GPU cluster node like the [Nvidia DGX A100 system](#).

SageMaker Studio

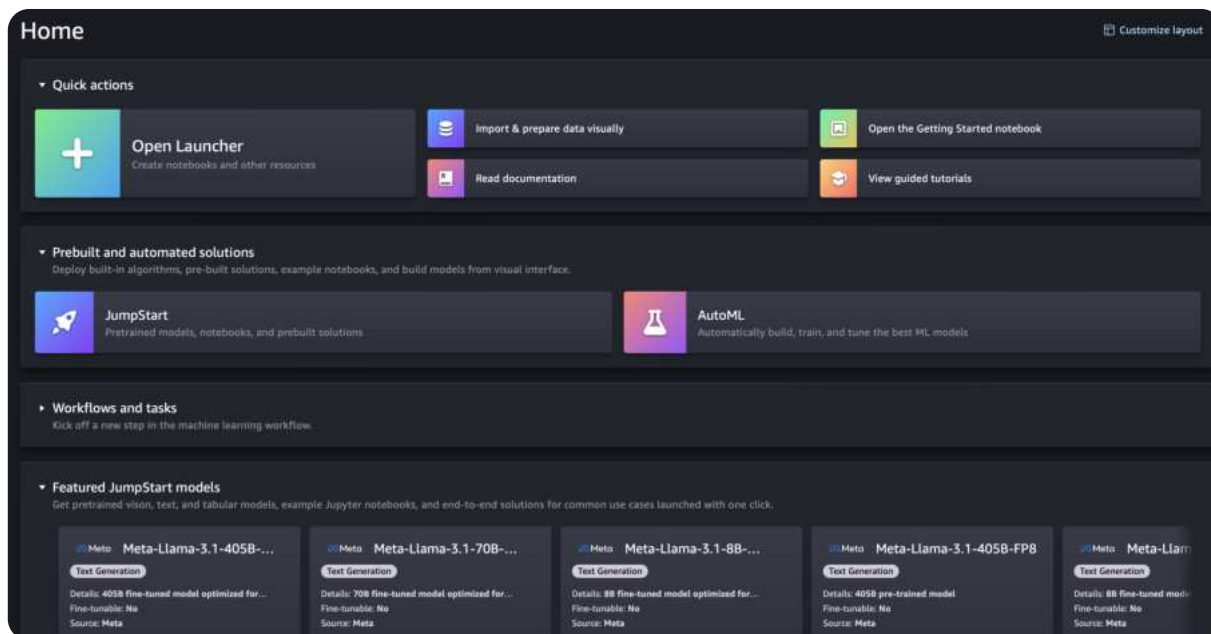


Image 2.17. SageMaker Studio UI

Amazon SageMaker Studio is a cloud-based fully managed integrated development environment (IDE) with pre-built tools for end-to-end ML development.

SageMaker Studio streamlines the ML workflow, making it accessible to data scientists, developers, and engineers. Its comprehensive set of features distinguishes it as a powerful and user-friendly platform for ML development and deployment.

1. **Unified Platform:** SageMaker Studio unifies various ML tasks like data exploration, training, and deployment within a single, visual interface.
2. **Collaboration:** Multiple team members can collaborate seamlessly using shared notebooks and resources, fostering teamwork.
3. **One-Click Model Deployment:** Easily deploy models to endpoints for real-time or batch predictions with a single click.
4. **Experiment Tracking:** Automatically tracks experiments, parameters, and results for easy model comparison and reproducibility.
5. **Integrated Debugger:** Debug and profile ML models during training to identify and fix issues efficiently.
6. **Automatic Model Tuning:** Hyperparameter tuning is simplified with automated optimization for improved model performance.
7. **Security and Compliance:** Built-in security features ensure data privacy and compliance with regulations.
8. **Versatility:** Supports popular ML frameworks like TensorFlow, PyTorch, and scikit-learn, offering flexibility in model development.

To play with LLMs in SageMaker studio and also in general with LLMs on SageMaker, your AWS account must be pre-approved for higher grade GPUs. If you have a new account or it has only been used for benign services since its creation then you will have to request for “Service Quota” limit increase for that particular type of instances and usage type like for notebooks, endpoints, or other jobs.

Example of quota increase for ml.g5.4xlarge instance that has 24GiB GPU Memory, for SageMaker endpoint usage with the cost of only \$1.624 on-demand price per hour.

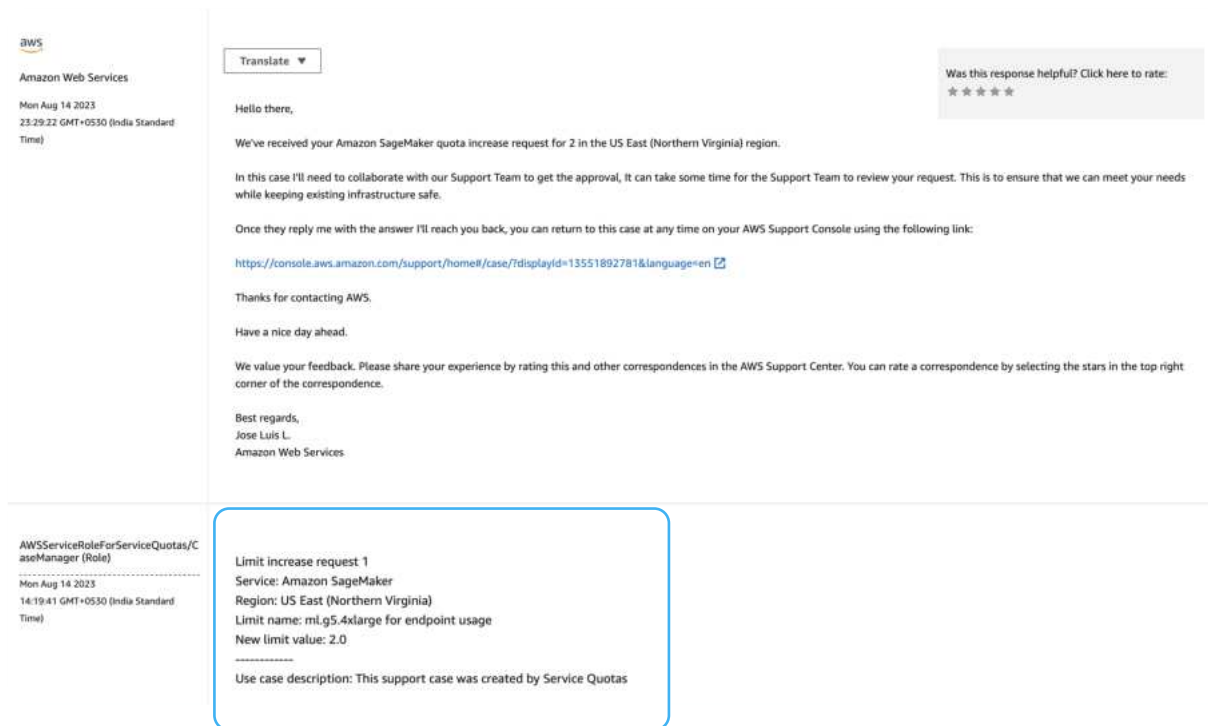


Image 2.18. Quota Increase request for ml.g4.4xlarge instance

Similarly, you will have to request a quota increase for using higher grade GPUs in SageMaker Studio.

Playing with LLMs in SageMaker studio is the same as playing with them on your local notebooks or in Google Colab except that you have access to industrial ML/AI specialised GPUs.

Demo of inference on SageMaker studio:

1. If you haven't already then first set-up SageMaker in your AWS account and enter the studio environment.
2. Start a new studio notebook with default environment or create a new env with the following commands (recommended):

```
Unset
export CONDA_ENV_NAME=llm-playground \
&& conda create --name $CONDA_ENV_NAME python pip ipykernel -y \
&& conda activate $CONDA_ENV_NAME \
&& python -s -m ipykernel install --user \
--name $CONDA_ENV_NAME \
--display-name "Python LLM Playground"
```

In the above commands, first we create a new conda environment called “llm-playground” with Python, pip and ipykernel packages. After activating it, we then install a jupyter kernel so that we can choose this environment in the Jupyter Notebook.

3. After creating a ipykernel, start a new notebook environment. Select the above created kernel and after that, you use the notebook like any other jupyter notebook.

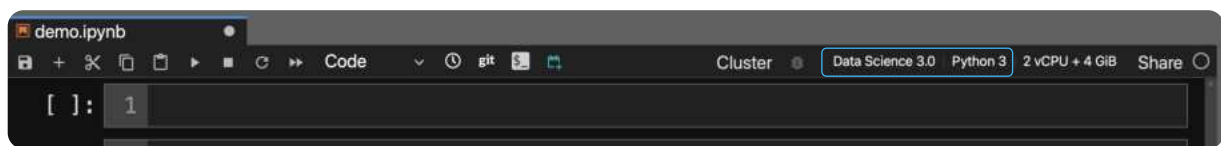


Image 2.19. Changing jupyter notebook kernel in the SageMaker Studio notebook

SageMaker Studio Lab

SageMaker Studio Lab enables quick experimentation with machine learning tools, frameworks, and libraries in a plug-n-play environment similar to Google Colab. It is completely free and separate from your AWS account.

You can choose between CPU and GPU, and it also provides many popular pre-packaged frameworks and customization opportunities with dedicated storage of up to 15GB.

The catch with the studio lab is that it seems to be only available after account approval i.e. not available to the general public. The author requested an account with a personal email ID and did not receive access after months. Then when the author's university email id was used to request access, it resulted in instantaneous account approval.

Another catch with studio lab is that because it is free, you can only access the resources for a very limited time before exhausting the daily limits. The limits at the time of writing were:

- Max continuous notebook run time on CPU = 4 hours
In a whole day, you can run two such cycles or many shorter continuous jobs for a total of 8 hours in a day.
- Max GPU notebook usage in a day = 4 hours

Please keep in mind that AWS can change the limits anytime.

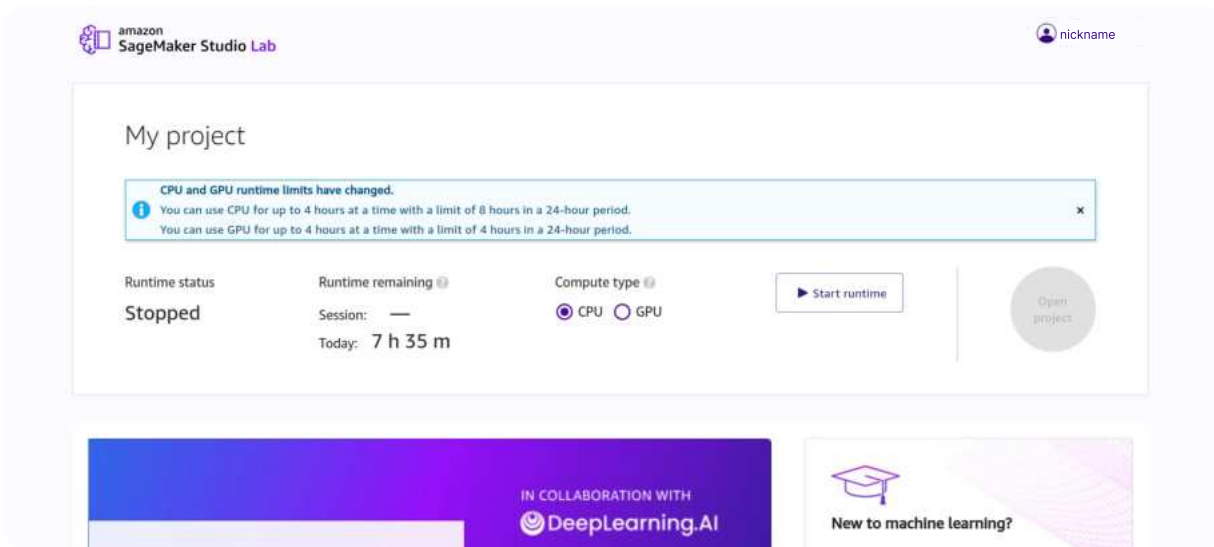


Image 2.20. Runtime limits of SageMaker Studio Lab

If you are able to get access, then you can easily play with models like Mistral and stable diffusion in the studio lab.

Steps to do inference with Stable Diffusion v1 on SageMaker Studio Lab:

1. Login to SageMaker studio lab
2. Scroll down and click on “Open Notebook” in the “Make AI Generated Images” box.

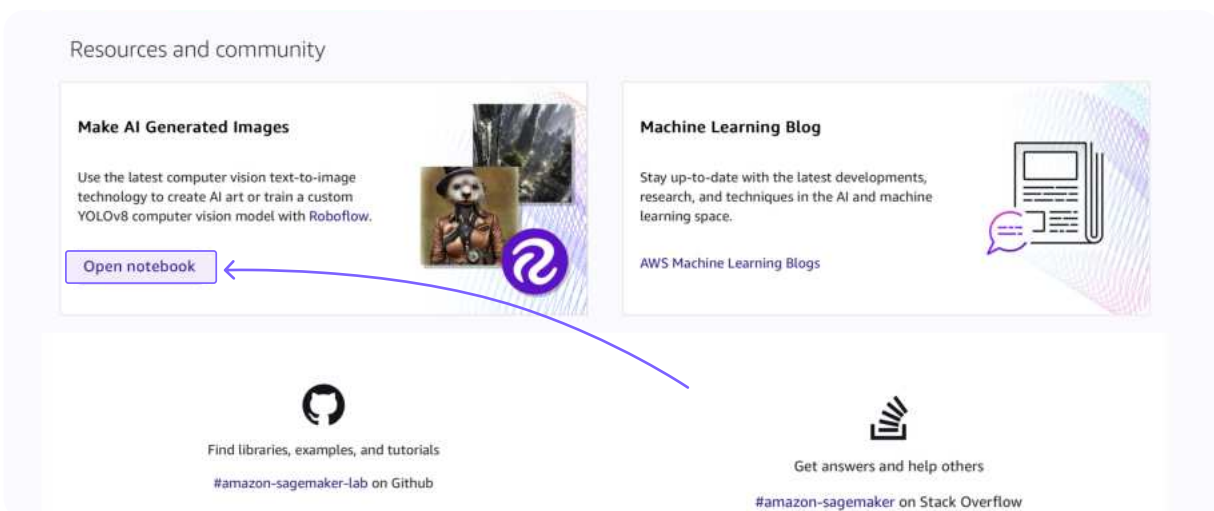


Image 2.21. Open Notebook button on SageMaker Studio Lab dashboard

3. On the new opened page, change compute type to GPU and click on “Start runtime”

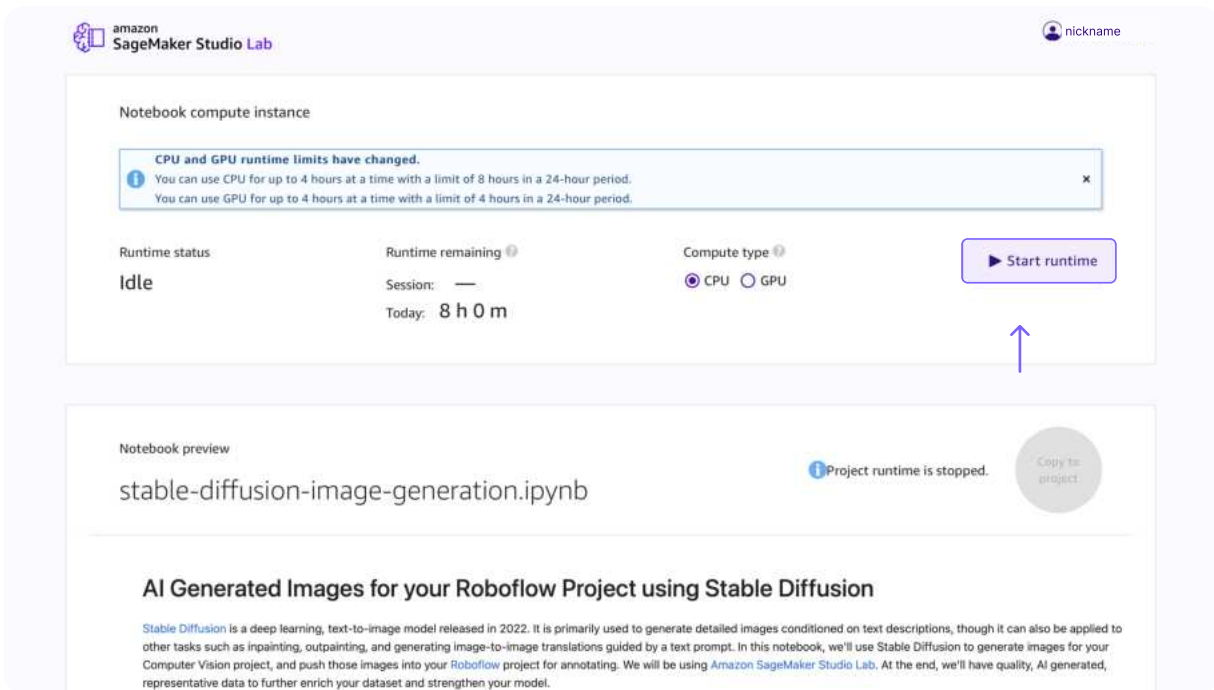


Image 2.22. Start runtime button on the SageMaker Studio Lab dashboard

4. Click on the “Copy to project” button and it will open a window similar to the SageMaker studio.

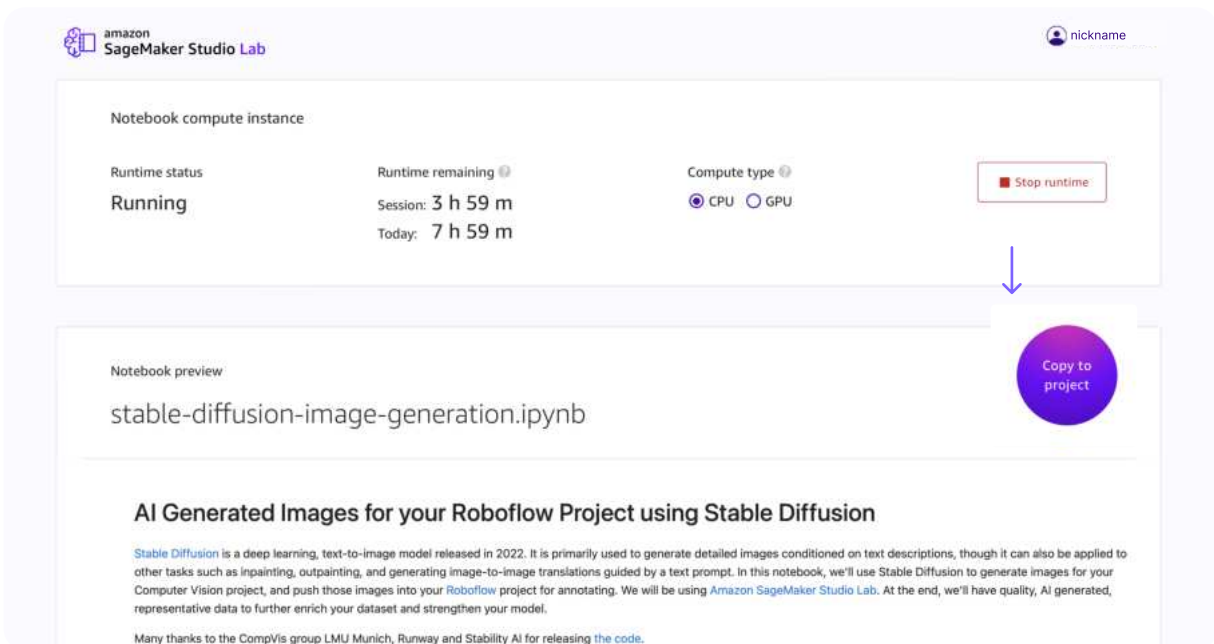


Image 2.23. Copy to project button to start the jupyter lab in the SageMaker Studio Lab

5. Interact with the jupyter notebook by executing the notebook cells, one by one.

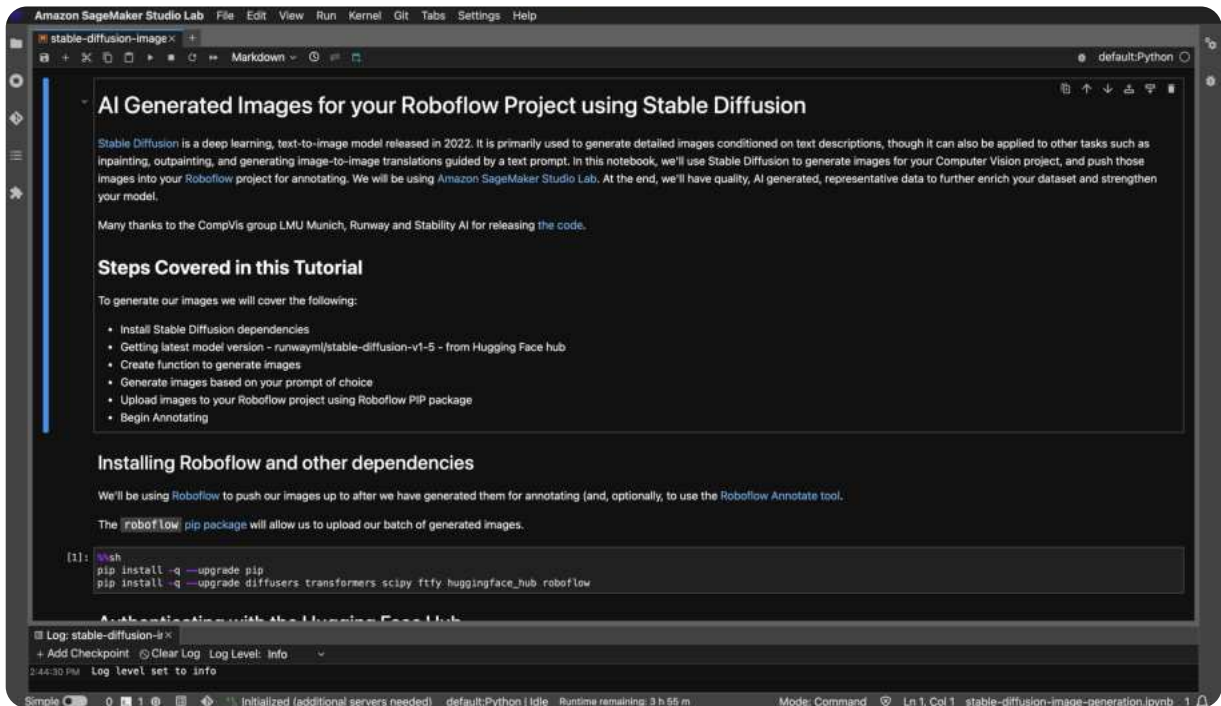


Image 2.24. Stable diffusion notebook in the SageMaker Studio Lab

Our prompt to the model:

```
Unset
generate_images(
    "create an image of a person standing alone in a subway station, "
    "under a single bright fluorescent light, surrounding tiles reflect the "
    "light, "
    "casting soft shadows around the individual engrossed in their smartphone, "
    "
    "ambiance is quiet, contemplative, with the architectural details of the "
    "subway, "
    "turnstiles, signs, faintly visible in the periphery, "
    "suggesting an urban narrative of isolation amidst the city's rush",
    12,
    guidance_scale=4,
    display_images=True
)
```

Output:

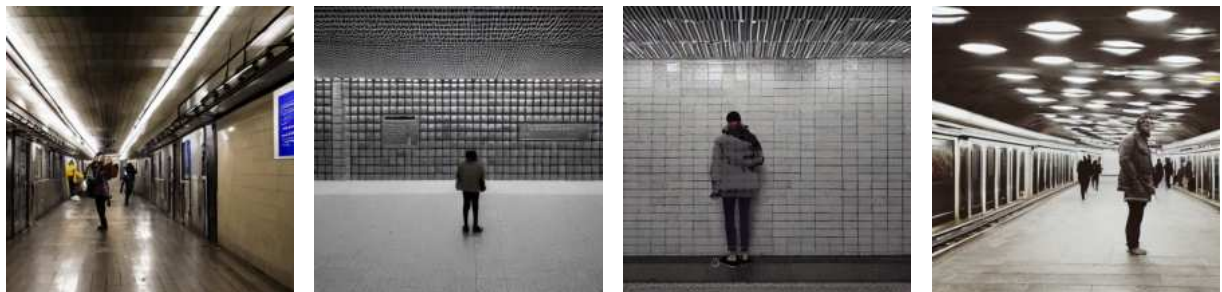
```
[*]: generate_images(
  "create an image of a person standing alone in a subway station, "
  "under a single bright fluorescent light, surrounding tiles reflect the light, "
  "casting soft shadows around the individual engrossed in their smartphone, "
  "ambiance is quiet, contemplative, with the architectural details of the subway, "
  "turnstiles, signs, faintly visible in the periphery, "
  "suggesting an urban narrative of isolation amidst the city's rush",
  12,
  guidance_scale=4,
  display_images=True
)
```

Token indices sequence length is longer than the specified maximum sequence length for this model (82 > 77). Running this sequence through the mo
 The following part of your input was truncated because CLIP can only handle sequences up to 77 tokens: ["amidst the city's rush"]

100%  50/50 [00:22<00:00, 2.26it/s]



Image 2.25. Out of the prompt using the Stable diffusion model running in lab's notebook



Output with the same prompt on [Midjourney 6](#) :



Image 2.26. Output of the same prompt on Midjourney6

SageMaker Jumpstart

Released in December 2020, [SageMaker Jumpstart](#) provides easy access to pretrained, open-source models for a wide range of problem types. You can incrementally train and tune these models before deployment. JumpStart also provides solution templates that set up infrastructure for common use cases, and executable example notebooks for machine learning.

You can deploy, fine-tune, and evaluate pretrained models from popular models hubs through the JumpStart.

Let's deploy [Llama3.1-8B-Instruct](#) model:

If you haven't requested g5 instances before, start by requesting a quota increase for ml.g5.4xlarge for endpoint usage through Service Quotas. Please note that it might take up to 24 hours for the ticket to be resolved.

Navigate to your Sagemaker Studio, and under “Sagemaker Jumpstart,” select “Meta-Llama-3.1-8B-Instruct”.

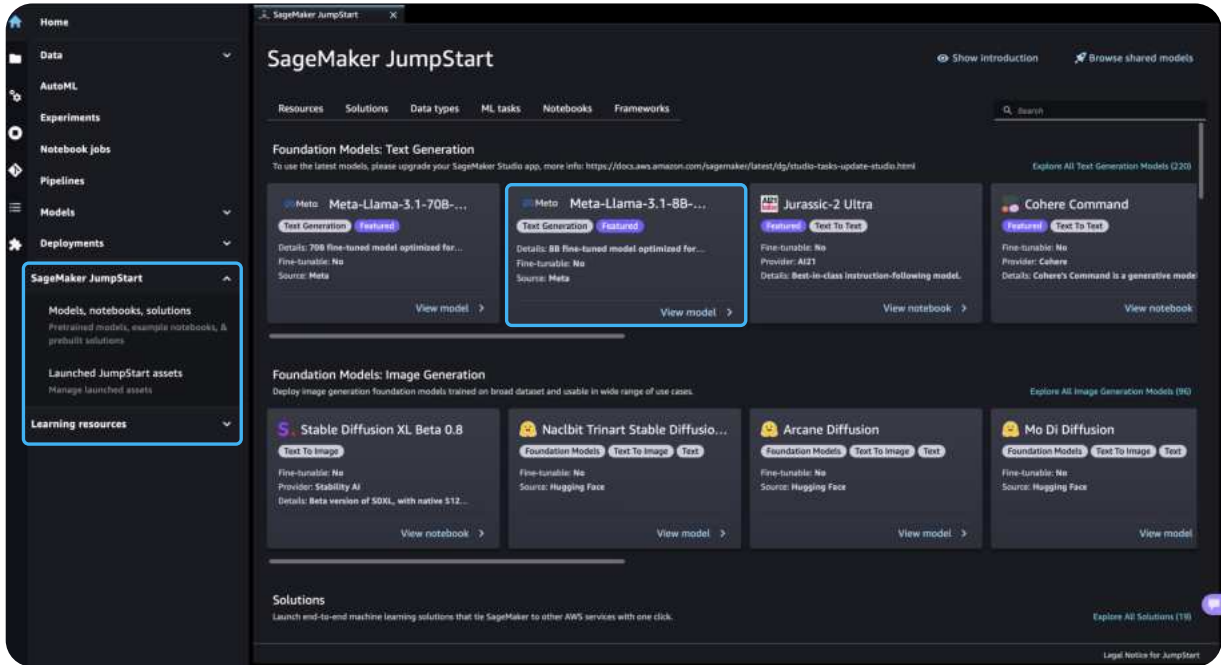


Image 2.27. Llama 2 7B Chat in Sagemaker Jumpstart section in SageMaker studio.

Change the Deployment configuration as desired, note the “Endpoint Name” variable ENDPOINT_NAME and click on “Deploy” button.

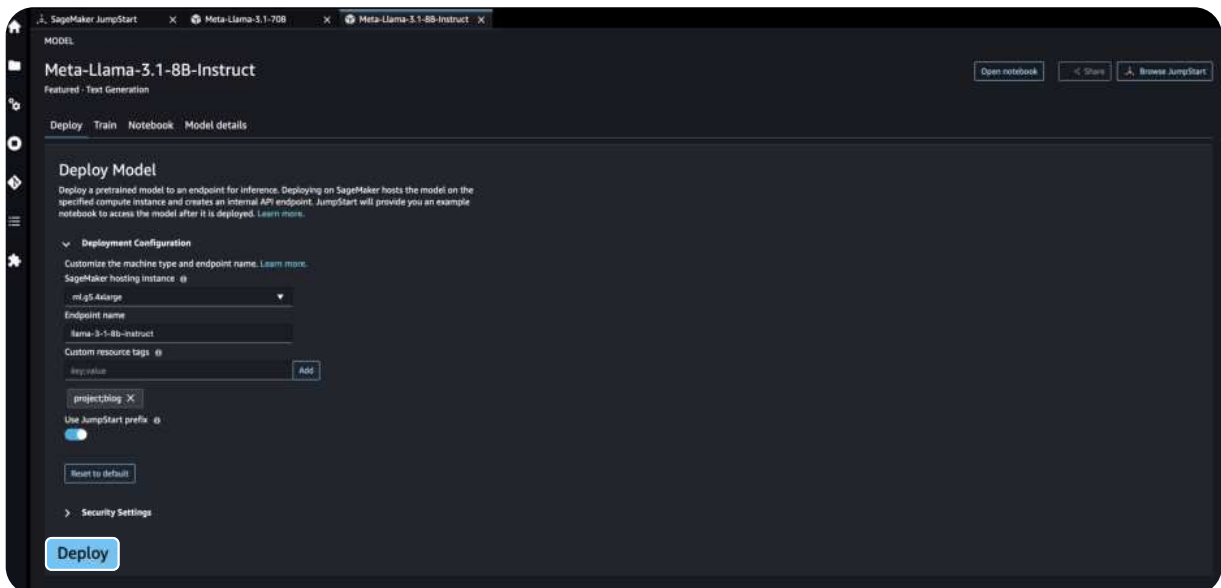


Image 2.28. Llama 3.1 8B Instruct model detail page in Pagemaker Studio.

This model is not available for fine-tuning, use “Meta-Llama-3.1-8B” for fine-tuning on your dataset.

After a couple of minutes, the endpoint status should be in service.

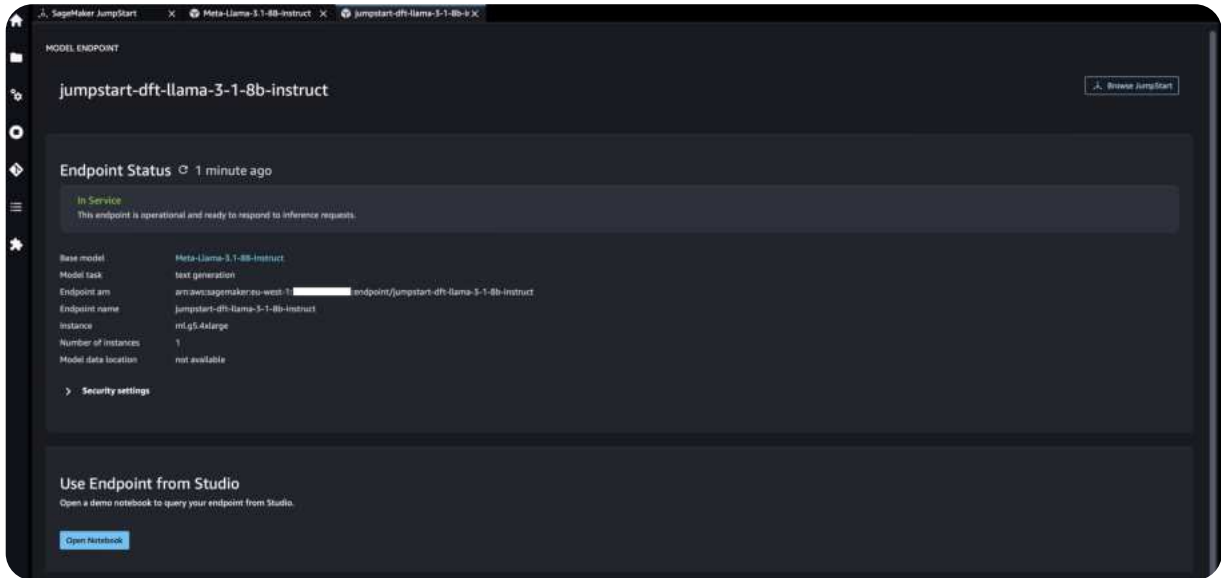


Image 2.29. Sagemaker model endpoint status.

You can also check the endpoint status from the “Endpoints” menu item which is under “Deployments” in SageMaker Studio.

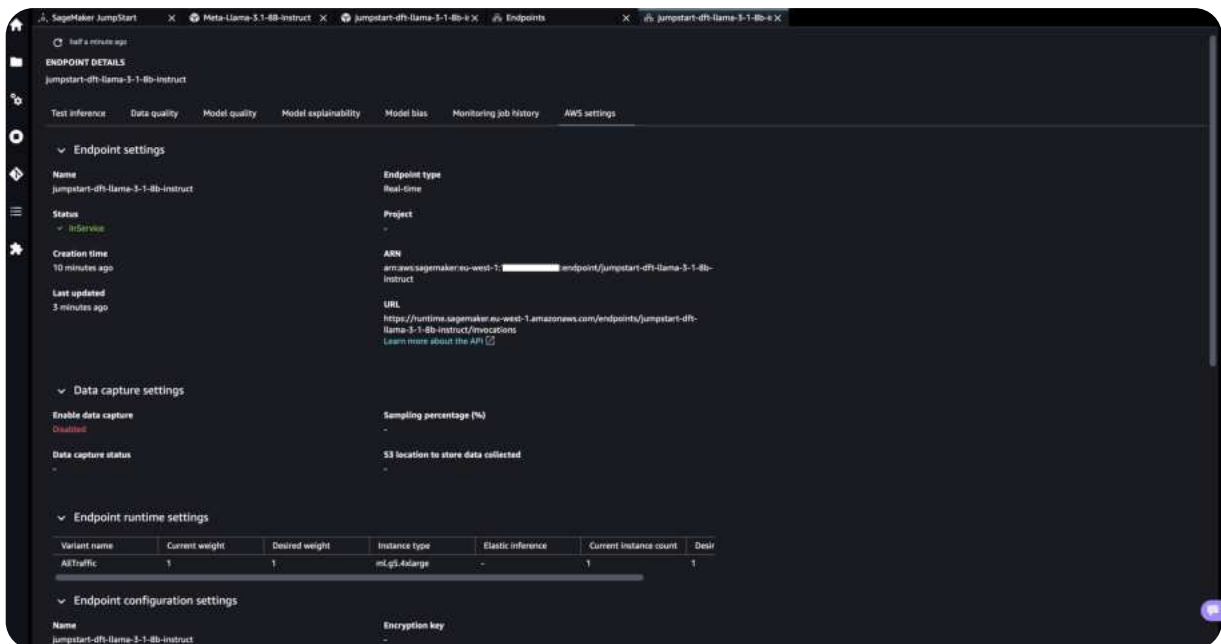


Image 2.30. Endpoint Details in SageMaker studio.

To invoke this model you will need an environment with boto3 installed. Let's invoke this endpoint from [AWS Lambda](#) function that comes preinstalled with boto3 package in Python runtime:

Note

This book has a GitHub repository with all the code and examples used throughout the book:

https://github.com/Expo-Platform/llm-book/tree/main/notebooks/Chapter_2

Don't stress if you've never used git. The book's code is also available [here](#) in a viewable format.

```
lambda_function.py x Environment Variabl x Execution results x +
1 import json
2 from typing import Any, Dict
3
4 import boto3
5
6 sm_client = boto3.client("sagemaker-runtime", region_name="eu-west-1")
7 ENDPOINT_NAME = "jumpstart-dft-llama-3-1-8b-instruct" # Change this
8
9
10 def invoke_sagemaker_endpoint(text: str, endpoint_name: str) -> str:
11     """Invoke sagemaker endpoint."""
12     data = {
13         "inputs": text
14     }
15
16     endpoint_response = sm_client.invoke_endpoint(
17         EndpointName=endpoint_name,
18         Body=json.dumps(data),
19         ContentType='application/json',
20         CustomAttributes="accept_eula=true", # This is required for Llama
21     )
22
23     endpoint_response_decoded = json.loads(endpoint_response['Body'].read().decode())
24
25     return endpoint_response_decoded
26
27
28 def lambda_handler(event:Dict[str, str], context:Any) -> Any:
29     """Invoke sagemaker endpoint with input request."""
30     payload = event["q"]
31     print(payload)
32
33     endpoint_response = invoke_sagemaker_endpoint(payload, ENDPOINT_NAME)
34
35     return {
36         "statusCode": 200,
37         "body": json.dumps(endpoint_response)
38     }
39
```

Image 2.31. Code to Invoke SageMaker endpoint from AWS Lambda.


```
Python
import json
from typing import Any, Dict

import boto3

...
ENDPOINT_NAME = "jumpstart-dft-llama-3-1-8b-instruct" # Change this

def invoke_sagemaker_endpoint(text: str, endpoint_name: str) -> str:
    """Invoke sagemaker endpoint."""
    ...

def lambda_handler(event:Dict[str, str], context:Any) -> Any:
    """Invoke sagemaker endpoint with input request."""
    payload = event["q"]
    print(payload)

    endpoint_response = invoke_sagemaker_endpoint(payload, ENDPOINT_NAME)

    return {
        "statusCode": 200,
        "body": json.dumps(endpoint_response)
    }
```

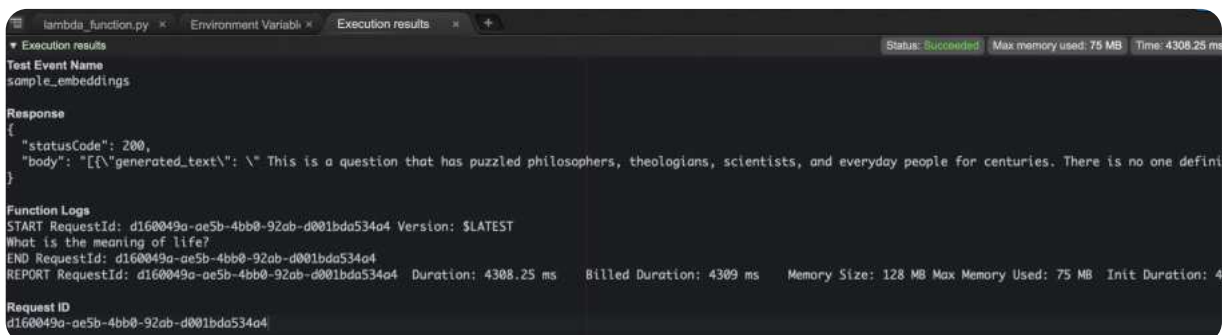


Image 2.32. Model invocation response in AWS Lambda.

Full model response:

```
Python
{
  "statusCode": 200,
  "body": "[{"generated_text": \" This is a question that has puzzled philosophers, theologians, scientists, and everyday people for centuries. There is no one definitive answer, as the meaning of life can vary greatly depending on an individual's beliefs, values, and experiences. Here are some possible perspectives on the meaning of life:\\n\\n1. Religious and Spiritual Perspectives: Many religions and spiritual traditions offer their own answers to the meaning of life. For example, in Christianity, the meaning of life is often seen as serving God and\"}]]"
```

We can also deploy the same model using boto3 instead of UI.

```
Python
%pip install -q -U sagemaker

from sagemaker.jumpstart.model import JumpStartModel

model = JumpStartModel(model_id="meta-textgeneration-llama-3-1-8b-i")
example_payloads = model.retrieve_all_examples()

accept_eula = True

if accept_eula:
    predictor = model.deploy(accept_eula=accept_eula)

    for payload in example_payloads:
        response = predictor.predict(payload.body)
        prompt = payload.body[payload.prompt_key]
        generated_text = response[0]["generated_text"]
        print("\nInput\n", prompt, "\n\nOutput\n\n", generated_text,
              "\n\n=====")
```

Amazon Bedrock

[Amazon Bedrock](#) went to Generally Available (GA) in September 2023. It is a fully managed service that enables developers to build, deploy, and manage AI models at scale. With Bedrock, you can choose from a variety of pre-trained base models from few third-party providers (e.g., AI21 Labs, Anthropic, Cohere, Stability AI, [Meta's Llama 3.1](#)) and also from Amazon (e.g., Titan), or create your own custom models using popular frameworks like TensorFlow, PyTorch, or Scikit-learn. Additionally, you can also import models trained elsewhere, making it easy to integrate existing models into your Bedrock workflow. To ensure model safety and reliability, Bedrock provides [guardrails](#), which allow you to define constraints on model inputs and outputs, preventing unwanted behaviour.

Bedrock also provides [knowledge bases](#), a fully managed capability that helps you implement the entire RAG workflow from ingestion to retrieval and prompt augmentation without having to build custom integrations to data sources and manage data flows. This makes it easy to get started with model development without requiring extensive domain expertise. To evaluate the performance of your models, Bedrock offers built-in model evaluation capabilities, allowing you to compare model performance and select the best model for your use case.

Furthermore, [Bedrock agents](#) enable generative AI applications to execute multi step tasks across company systems and data sources, like securely connect to your company's data sources, and augment the user request with the right information to generate an accurate response or look up a customer's details in the database and answer questions regarding their order status.

When it comes to invoking the models, Bedrock offers two main choices: serverless and [provisioned throughput](#). The serverless option allows you to invoke the available models billed at number of inputs & output tokens and are rate limited, while the provisioned throughput option provides a higher level of throughput for a model at a fixed cost, allowing you to optimise for performance and cost. Upcoming features like [Prompt flows](#) and [Bedrock studio](#) will further enhance the service, making it easier to build and deploy Generative AI systems.

One of the key benefits of Bedrock is its ability to enable rapid development of AI applications. If you need to move fast and get AI models up and running quickly, Bedrock is a great choice. However, if you need more control over the underlying infrastructure and model development process, Amazon SageMaker may be a better fit. SageMaker provides more advanced features, making it a better choice for organisations that require more fine-grained control over their AI workflows.

An example of chatting with Llama 3 70B Instruct model from Chat playground in Bedrock:

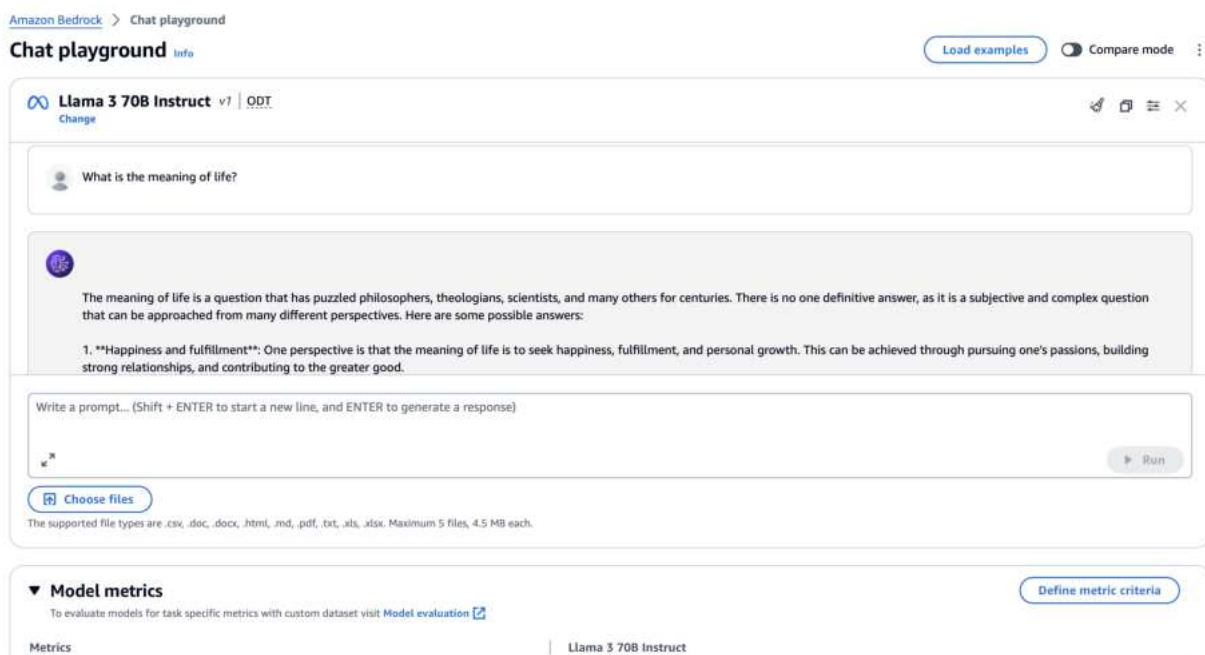


Image 2.33. Chat playground of Amazon Bedrock.

For your applications, you will most likely use boto3 to invoke the models in Bedrock.

```
Python
import json

import boto3

client = boto3.client("bedrock-runtime", region_name="us-east-1")
```

```
# Complete model response in response body
body = json.dumps(
    {
        "prompt": "\n\nHuman:What is the meaning of life?\n\nAssistant:",
        "max_tokens_to_sample": 200,
    }
).encode()

response = client.invoke_model(
    body=body,
    modelId="anthropic.claude-v2",
    accept="application/json",
    contentType="application/json",
)

response_body = json.loads(response["body"].read())
print(response_body)

# Streaming response i.e. word by word
body = json.dumps(
    {
        "prompt": "\n\nHuman:Write me a 100 word essay about snickers candy
bars\n\nAssistant:",
        "max_tokens_to_sample": 200,
    }
).encode()

response = client.invoke_model_with_response_stream(
    body=body,
    modelId="anthropic.claude-v2",
    accept="application/json",
    contentType="application/json",
)

stream = response["body"]
if stream:
    for event in stream:
        chunk = event.get("chunk")
        if chunk:
            print(json.loads(chunk.get("bytes").decode()))
```

For streaming response you need to call `invoke_model_with_response_stream` api:

```
Python
response = client.invoke_model_with_response_stream(
    body=body,
    modelId="anthropic.claude-v2",
    accept="application/json",
    contentType="application/json",
)

stream = response["body"]
if stream:
    for event in stream:
        chunk = event.get("chunk")
        if chunk:
            print(json.loads(chunk.get("bytes").decode()))
```

Full streaming response:

```
Python
{'completion': ' Here', 'stop_reason': None, 'stop': None}
{'completion': ' is a 100 word essay', 'stop_reason': None, 'stop': None}
{'completion': ' about Snickers candy', 'stop_reason': None, 'stop': None}
{'completion': ' bars:\n\nS', 'stop_reason': None, 'stop': None}
{'completion': 'nickers is one of', 'stop_reason': None, 'stop': None}
{'completion': ' the most popular candy bars around. Introdu', 'stop_reason':
None, 'stop': None}
{'completion': 'ced in 1930, it consists of nougat topped with',
'stop_reason': None, 'stop': None}
{'completion': ' caramel and peanuts that is encased in milk chocolate',
'stop_reason': None, 'stop': None}
{'completion': '. With its sweet and salty taste profile,', 'stop_reason':
None, 'stop': None}
{'completion': ' Snickers provides the perfect balance of flavors. The candy',
'stop_reason': None, 'stop': None}
{'completion': " bar got its name from the Mars family's", 'stop_reason':
None, 'stop': None}
{'completion': ' favorite horse. Bite', 'stop_reason': None, 'stop': None}
{'completion': ' into a Snickers and the rich', 'stop_reason': None, 'stop':
None}
{'completion': ' chocolate and caramel intermingle in your mouth while the',
'stop_reason': None, 'stop': None}
{'completion': ' crunch of peanuts adds text', 'stop_reason': None, 'stop':
None}
{'completion': 'ural contrast. Loaded with sugar, Snick', 'stop_reason': None,
'stop': None}
{'completion': 'ers gives you a quick burst of energy. It', 'stop_reason':
```



```
None, 'stop': None}
{'completion': "'s a classic candy bar that has endured for", 'stop_reason':
None, 'stop': None}
{'completion': ' decades thanks to its irresistible combination',
'stop_reason': None, 'stop': None}
{'completion': ' of chocolate,', 'stop_reason': None, 'stop': None}
{'completion': ' caramel, noug', 'stop_reason': None, 'stop': None}
{'completion': "at and peanuts. Snickers' popularity shows", 'stop_reason':
None, 'stop': None}
{'completion': ' no signs of waning anytime soon.', 'stop_reason':
'stop_sequence', 'stop': '\n\nHuman:', 'amazon-bedrock-invocationMetrics':
{'inputTokenCount': 21, 'outputTokenCount': 184, 'invocationLatency': 8756,
'firstByteLatency': 383}}
```

For those seeking alternatives to the major cloud providers, there are several other platforms worth considering. Let's briefly explore these options.



05 Other alternatives

Like Google Colab there are other alternatives that provide a free tier with some GPU hours.

1. [Paperspace Gradient](#) - is similar to Google Colab. Though they provide some GPU hours, they do not specify the number of free GPU hours. The quality of those GPU hours is not good either, i.e. one will see frequent runtime disconnection. With the free plan, you will get the M4000 GPU which is inferior to the Tesla T4 GPU in Colab, but you get free persistent storage.
2. [Kaggle](#) - is one of the best options. Kaggle gives you 4CPUs, 29GB of RAM, and 30 hours of Tesla P100 GPU per week. The quality of these GPU hours is reliable because there are no disconnections. By the way, Kaggle is owned by Google. You can easily run smaller LLMs and test them.
3. [Saturn Cloud](#) - has a hosted free plan where you get some unspecified free compute per month as well as resources with either a 64GB RAM or one T4 GPU with 16GB RAM. Though the quality of GPU hours is inferior to Kaggle, it is a decent free alternative.
4. [Lightning AI](#) - provides free monthly 22 GPU hours. In their free tier you also get one studio with 4CPUs which is fully free without the GPU. The quality of these GPU hours is amazing because you never disconnected once connected to the machine.
5. [GitHub Codespaces](#) - provides up-to 4 core CPU and 16GB RAM in the free tier. You can also easily run smaller models. The VS Code interface with a terminal is also available in codespaces.
6. [Azure ML Notebooks](#) and [Google Vertex ML notebooks](#) - each cloud provider offers free credits that you can use to run notebooks on powerful infrastructure on-demand. Azure provides \$200 in free credits, but it is not easy to use. Google Cloud, similar to Azure, provides \$300 in credits that are valid for 365 days from allocation.

Summary

This chapter delves into the practical aspects of deploying LLMs (Large Language Models) locally or on cloud platforms like Google Colab and AWS SageMaker. It begins by emphasizing key considerations such as model size and memory requirements. The chapter discusses challenges and solutions for managing model memory during loading and the KV cache requirements at runtime. It also introduces quantization as a method to optimize computational needs at the expense of slight model quality reduction, a topic to be explored further later in the book.

Moving into practical implementation, the chapter explores local model inference as a cost-effective approach for experimenting with LLMs. It highlights tools like GPT4ALL, LM Studio, and the Hugging Face Transformers library, along with frameworks such as Ollama, llama.cpp, and vLLM. Additionally, the chapter provides examples of using Google Colab for LLM experimentation and outlines comprehensive solutions like AWS SageMaker Studio, Studio Lab, and Sagemaker Jumpstart for more advanced usage with larger models requiring high computational power. It also introduces Bedrock, a serverless generative AI service from AWS that allows developers to easily integrate powerful LLMs into their applications with minimal code.

Looking ahead, the next chapter will delve into the architectural patterns of LLMs. It will cover prompt engineering, RAG (retrieval-augmented generation), customization and fine-tuning, reinforcement learning from human feedback (RLHF), pre-training models from scratch, and LLM agents. As we venture into these more advanced concepts, it may be helpful to review the foundational ideas covered in the previous chapters.

Appendix: Glossary

A

[Amazon Bedrock](#): A fully managed service offered by AWS that allows users to LLMs models from leading AI companies through a single API.

[API](#) (Application Programming Interface): A set of protocols and tools for building software applications, often used to interact with LLMs and other AI services.

F

076CE2: The process of further training a pre-trained model on a specific dataset to adapt it for a particular task or domain.

G

[GGUF/GGML](#): Data/file formats used by Llama.cpp and other tools for efficient storage and inference of large language models, developed by Georgi Gerganov.

Google Colab: A free, cloud-based platform for writing and executing Python code, often used for machine learning and data analysis tasks.

[GPT4All](#): An ecosystem of open-source large language models that can run locally on consumer-grade hardware.

GPU (Graphics Processing Unit): A specialised processor designed to accelerate graphics rendering, also used for parallel computing in machine learning tasks.

H

[HuggingFace Transformers](#): A popular Python library for working with pre-trained transformer models, including loading, fine-tuning, and deploying these models.

I

Inference: The process of using a trained model to make predictions or generate outputs based on new input data.

K

[Key-Value \(KV\) Cache](#): A memory optimization technique used in Transformer models to store previously computed attention values.

L

[Large Language Model](#) (LLM): A type of artificial intelligence model trained on vast amounts of text data to understand and generate human-like text.

Llama.cpp: An open-source C++ implementation for running LLMs, known for its efficiency and ability to run large models on consumer hardware.

LM Studio: A user-friendly desktop application for running and experimenting with various large language models locally.

O

[Ollama](#): A tool for running, managing, and developing with large language models locally.

P

[Pre-training](#): The process of training a Large Language Model from scratch to predict the next token for a given sequence.

Q

[Quantization](#): The process of reducing the precision of model weights to decrease memory usage and computational requirements.

S

[SageMaker Jumpstart](#): A feature of Amazon SageMaker that provides pre-built, deployable machine learning solutions and models.

[SageMaker Studio](#): An integrated development environment (IDE) for machine learning, provided by Amazon Web Services.

[SageMaker Studio Lab](#): A free version of SageMaker Studio with limited resources, designed for learning and experimentation.

T

TPU (Tensor Processing Unit): A custom-built AI accelerator developed by Google specifically for neural network machine learning.

Transformer: A deep learning model architecture introduced in the paper "Attention Is All You Need," which forms the basis for many modern LLMs.

V

vLLM: An open-source library for fast LLM inference and serving, known for its high-throughput inference capabilities.

