LLM Classification: How to Select the Best LLM for Your Application

Discover the family of LLMs available and the elements to consider when evaluating which LLM is the best for your use case.

Mar 2024 · 15 min read

Andrea Valenzuela
A data expert at CERN, democratizing tech learning. Skilled in data engineering and analysis.

TOPICS

Artificial Intelligence (AI)

Python

Since the launch of ChatGPT, it seems a new Large Language Model (LLM) emerges every few days, alongside new companies specializing in this technology. Each new LLM is trained to excel the previous one in various ways. For example, we more often see LLMs tailored to improve a specific use case: chat models, coding assistants, and long-context comprehension, among others.

Choosing the right model for a new LLM-based application or project can be a daunting task, not to mention the challenge of staying up-to-date with the latest models and their features.

In my case, this often leads me to default to using ChatGPT or GPT models via API calls.

Here, I aim to delve into the diverse array of available models and highlight the key capabilities one should consider when choosing the right model according to the project's characteristics.

We will especially focus on the different methods of interacting with these models as one of the key features to consider when choosing between all the available models.

The goal is for this comprehensive guide to serve as a cheat sheet, helping you navigate the vast selection of LLMs available for your specific tasks.

*If you are new to the LLM world, I really recommend you to follow the course on **AI Fundamentals** and **Large Language Models Concepts** to get familiar with the basic wording and models.*

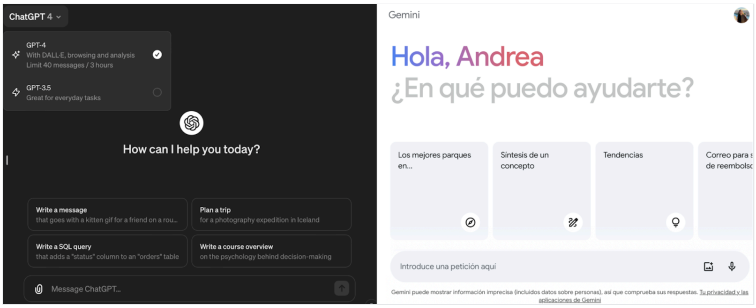
Interfacing LLMs

Accessing LLMs and interfacing with them in an LLM-powered application or project can be done through various methods depending on the technical needs.

#1. Playground

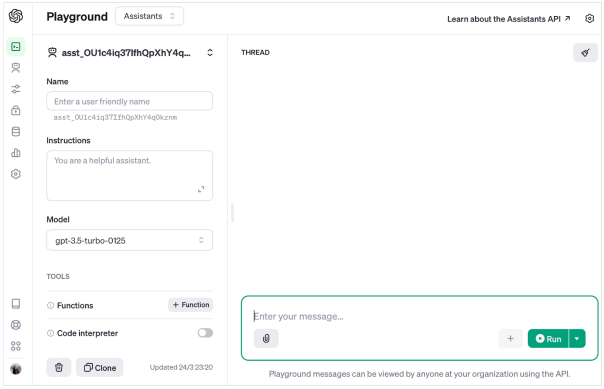
The most user-friendly approach to interface LLMs is via conversational interfaces in our browser. This is the case of ChatGPT, **where the user interface is simply a chat interface that allows you to interact with the model.**

In general, these interfaces allow very little or no control or model customization by the user, but it is an easy way to try out models for simple tasks. ChatGPT by OpenAI and Google's Gemini provide these kinds of interfaces:



Screenshot of OpenAI's ChatGPT (left) and Google's Gemini (right) chat interfaces.

Sometimes, there are providers also allowing some degree of customization through similar user-friendly chat interfaces. In the case of OpenAI, the platform is called "Playground."



Screenshot of the OpenAI Playground interface.

Playground interfaces allow us to try out different models with limited control over the settings. In that way, we can explore which parameters are suitable for your task. Nevertheless, as you have probably already realized, playground or chat interfaces do not allow us to embed the interaction with the LLM in an application or project.

#2. Native API Access

One of the most popular interfaces to query LLMs is through API access. This approach is particularly advantageous when we don't require further model customization.

API calls can be seamlessly integrated into our scripts, eliminating the need for infrastructure setup and maintenance on our part.

However, convenience comes at a price, and despite the reliance on external services—a dependency that you will always have if you prefer not to host the model and manage its necessary infrastructure- costs will inevitably scale with usage.

In the case of API calls, a good practice is to implement a simple wrapper around the API call itself to interface with the model in a modular way. For the OpenAI API, which is the one I have used the most so far, I always use the wrapper in the next section, so that the model selection is transparent to the user.

GPT API Calls

Writing a standardized yet flexible method to query GPT models through the OpenAI API is very useful if you want to interface with the model smoothly and in a less error-prone manner. That is why I usually set up the GPT call in a wrapper function at the beginning of the program, ensuring that the usage of the model is transparent to the application.

The `openai` library (`pip install openai`) already provides a `ChatCompletion` endpoint that allows us to call any given GPT model by using the `.create` method.

The .create method has two mandatory parameters:

1. **The GPT model to be used.** ChatGPT is powered by OpenAI's most advanced models, including `gpt-4`.
2. **The user prompt is formatted** in a very particular way.

Given these two parameters, we can write the GPT call as follows:

```
prompt = "Hello world"
chat_completion = openai.ChatCompletion.create(model="gpt-3.5-turbo", messages=[{
```

Explain code POWERED BY datalab

When calling the model, we will get back the completion in a JSON format:

```
{
 "choices": [
   {
     "finish_reason": "stop",
     "index": 0,
     "message": {
       "content": "Hello! How can I assist you today?",
       "role": "assistant"
     }
   }
 ],
 "created": 1692042919,
 "id": "chatcmpl-7nXwl0tywcTb91xeEuhpQbpRD6XLM",
 "model": "gpt-3.5-turbo-0613",
 "object": "chat.completion",
 "usage": {
   "completion_tokens": 9,
   "prompt_tokens": 9,
   "total_tokens": 18
 }
}
```

Explain code POWERED BY datalab

Therefore, it would be interesting to only get the model completion as the function returns. If we put all the bits and pieces together, our wrapper implementation can be written as:

```
def chatgpt_call(prompt, model="gpt-3.5-turbo"):
   response = openai.ChatCompletion.create(
       model=model,
       messages=[{"role": "user", "content": prompt}]
   )
   return response.choices[0].message["content"]
```

Explain code POWERED BY datalab

If you are interested in more details about this implementation, the article **ChatGPT API Calls: A Gentle Introduction** explores the details further!

Finally, it's also worth noting that most API providers, including OpenAI, offer some free calls when you create your account. Still, the credits are limited (usually to less than $20) and typically constrained to a time window (usually a month).

In any case, it is always a good practice to wrap the LLM calls in a function - or collection of functions - **so that the model is transparent to the application regardless of the provider.**

#3. Hosting the Model Locally

Hosting the model ourselves involves downloading the model weights and running it on local machines or private servers.

This approach is compelling since it allows full control over the model infrastructure, but it also means that the technical effort scales up considerably.

*If you are curious about the steps involved in running the infrastructure for an LLM, have a look at the **LLMOps Essentials** article!*

Running the model locally enables further customization according to our needs. In this context, one of the most popular choices for self-hosting is the well-known LLaMa models by Meta AI. They have released lightweight models that can be run on a laptop with only 8GB of RAM.

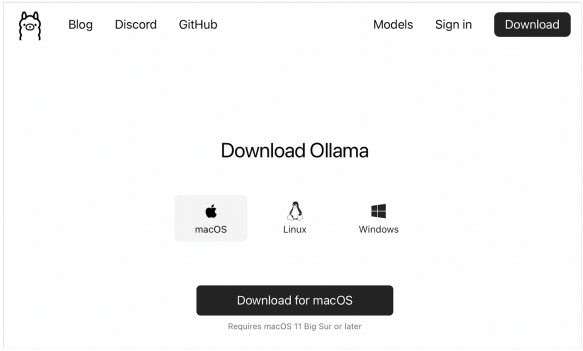
Running the LLaMa model on our laptop is fairly easy, thanks to platforms like **Ollama**.

Ollama Platform

Ollama offers a platform designed for running LLMs like **LLaMa 2** and Code LLaMa locally on our devices. It supports macOS, Linux, and Windows, enabling customization and creation of our proprietary models. For more details, you can visit **Ollama's website**.

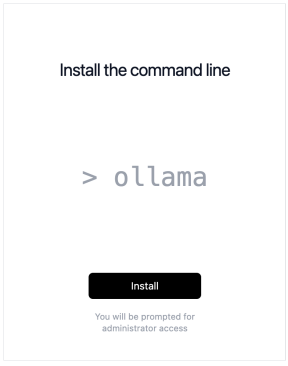
Downloading the Ollama platform can be done from the official page, while the model weights can be pulled directly using the terminal interface.

Let's install Ollama first by navigating to its main page!



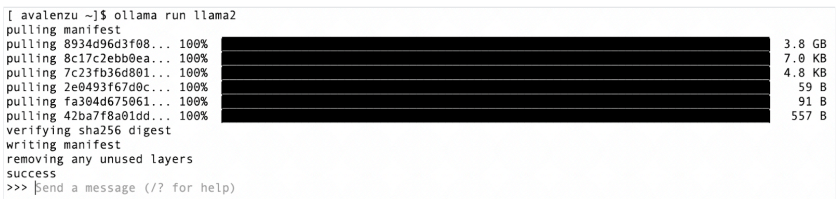
*Screenshot of the main page of **Ollama's website**.*

In this case, Ollama is a command line utility:



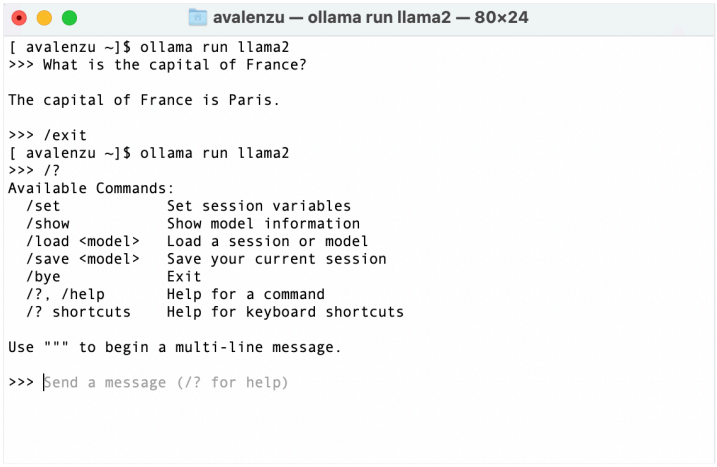
Screenshot of the Ollama command line tool installation.

Once the command line utility is installed, we can start the model with the ollama run <model name> command. If it is the first time running the model on our device, Ollama will pull it for us:



Screenshot of the first run of the LLaMa 2 model with the Ollama command line tool. The model gets pulled automatically if it is the first usage.

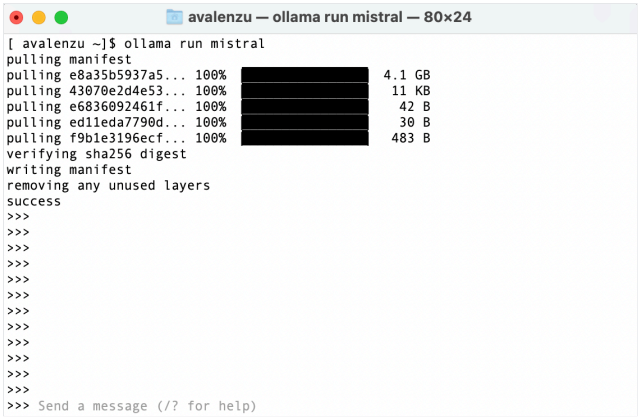
From that moment onwards, prompting and exiting the model from the command line is very straightforward:



Screenshot of the user interaction with the model via Ollama command line tool.

As we can see in the image above, the model also has a small set of extra commands available.

Ollama also supports other models, such as the new Mistral:



Screenshot of the first run of the Mistral model with the Ollama command line tool. The model gets pulled automatically if it is the first usage.

The model is again ready to be used out of the box in one single command!

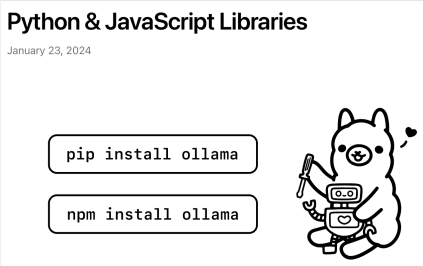
This approach is advantageous as it only consumes resources from our device. However, it is important to remember that **model performance generally increases with the number of parameters.**

Consequently, **the more parameters a model has, the more resources it will require.** Therefore, larger models will likely need to run on private servers for self-hosted LLM applications.

In case we want to build a large-scale application, it is worth noticing that **Ollama can also run with Docker Desktop on Mac and run inside Docker containers with GPU acceleration on Linux.**

One thing you might be wondering at this point is: Can I embed OLLAMA in my scripts?

Indeed, hosting the model yourself doesn't mean you can't integrate it into your Python scripts! **Ollama also offers the possibility to interface with your self-hosted model via Python.**



Ollama's Python and JavaScript libraries allow us to incorporate the model inference in our scripts.

Ollama is not our only option if we want to host the model ourselves. In general, LLMs are often available for download on multiple open-source platforms also offering community support.

Open-source LLMs are generally available at no cost for academic and research purposes. However, commercial use terms can differ based on the specific license under which a model is released. Always be aware of whether you are experimenting with LLMs or building LLM-based applications for commercial purposes.

*Are you willing to start your journey to developing LLMs today? Then the course **Developing Large Language Models** is for you!*

Cloud Hosting

Hosting powerful models locally can be very resource-intensive, which is why using **cloud services can be a viable alternative, allowing for the scalability of resources on-demand** and support from cloud providers.

It's noteworthy that traditional cloud services like Azure now offer customized pipelines to host LLMs, such as Mistral, so that we don't need to implement specialized hosting.

This is a significant advantage for deploying applications within your organization, as many companies already have contracts with these providers for services like email and general cloud solutions.

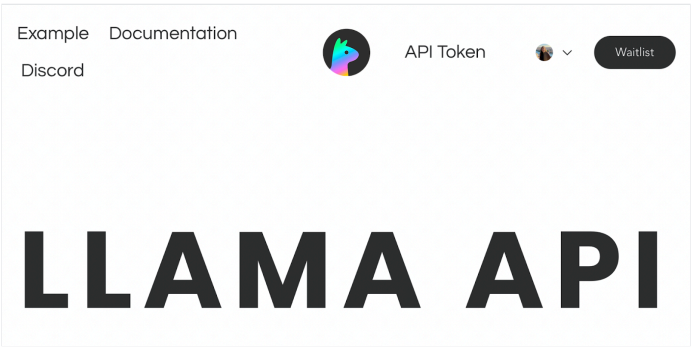
#4. Third-Party APIs

Opting for third-party providers can offer a balance between not managing the host infrastructure and having some control over the model, compared to simply using a native API service.

These providers typically offer API access, but it is important to note that costs may increase significantly with extensive use, akin to using the native OpenAI API.

The key difference lies in the hosting entity. **Third-party providers, rather than the native LLM services, host the model on their infrastructure and expose the API endpoint,** often offering a variety of models from different sources for flexible integration.

A popular third-party API provider is the **LLAMA API**.



*Screenshot of the main page at the **LLAMA API website**.*

The API is not free of use, but you can try it for free since 5$ of free credits is issued for any new account. However, remember that those credits are valid for one month only after they are issued.

Using the LLAMA API is very straightforward. Once it is installed (pip install llamaapi), we can do a first trial call as follows:

```
from llamaapi import LlamaAPI
import json

llama = LlamaAPI('<Insert API key here>')
prompt = "Which is the capital of France?"

api_request_json = {
  "model": "llama-13b-chat",
  "messages": [
    {"role": "system", "content": "You are a friendly chatbot."},
    {"role": "user", "content": prompt},
  ]
}

response = llama.run(api_request_json)
print(json.dumps(response.json(), indent=2))
```

Explain code POWERED BY datalab

However, as with the OpenAI API, it is a good practice to embed the API call in a wrapper function. By following the same approach as before:

```
def llm_call(prompt, model="llama-13b-chat"):
    api_request_json = {
        "model": model,
        "messages": [
            {"role": "system", "content": "You are a friendly chatbot."},
            {"role": "user", "content": prompt},
        ]
    }
    response = llama.run(api_request_json)
    return response.json()["choices"][0]["message"]["content"]
```

Explain code POWERED BY datalab

Given this implementation, we will get the string response back from the LLM by simply calling llm_call() :

```
llm_call("Which is the capital of France?")
```

```
'😊 Hey there! The capital of France is of course... PARIS 💖 ! 🎉
Do you want to know more about the City of Light or would you rathe
r talk about something else? 🤔😄'
```

LLAMA API supports a variety of state-of-the-art models, not just Meta's LLaMa. We will find out which models it supports in the next section!

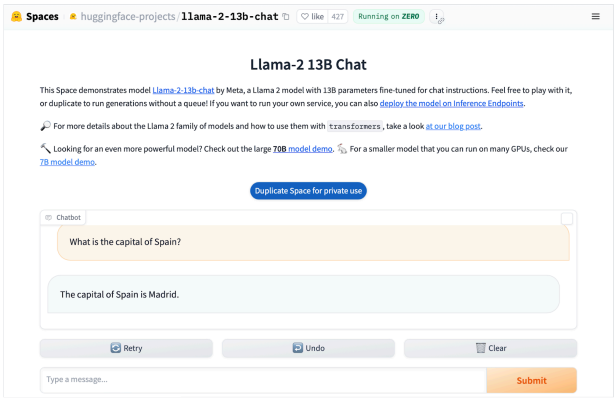
Finally, in the case of LLAMA API, the API cost depends on the number of parameters of the model, and it seems to be constant across all models:

Screenshot of the LLAMA API pricing. This information is only available once you create an account.

Hugging Face World

The most popular third-party provider for LLMs is **Hugging Face**. Hugging Face is mostly known for its transformers library, the most popular library in Python for Natural Language Processing.

Hugging Face provides multiple interfaces to access LLMs. It has a user-friendly playground interface called "Spaces":



Screenshot of the "Spaces" playground in Hugging Face.

It also allows for model hosting and endpoint deployment, as well as for direct download and local usage of the models.

It is important to keep in mind that Hugging Face is a kind of "developer's site", so despite the fact that you can find tutorials on how to get started, using their models to scale up an application requires technical knowledge.

*If you are willing to learn more about Hugging Face, I strongly recommend the article **What is Hugging Face? The AI Community's Open-Source Oasis**.*

One complementary tool that I normally use to build LLM-powered applications from scratch, together with Hugging Face, is LangChain. The tutorial **How to Build LLM Applications with LangChain** provides a nice hands-on introduction.

LLMs Classification

We have been discussing the different methods of accessing and running LLMs, such as GPT, LLaMa, and Mistral models. However, there are many more models available, including various variants of the aforementioned ones. **In this section, we will classify the main models that exist today,** listing their characteristics, prices, and use cases.

GPT Models - OpenAI

OpenAI models have become very popular since the launch of ChatGPT, now powered by GPT-3.5 or GPT-4, depending on your subscription. For many basic tasks, the difference between GPT-3.5 and GPT-4 models is not significant.

However, GPT-4 demonstrates much greater capability in more complex reasoning scenarios compared to any previous models.

In addition, GPT-4 is a multimodal model that accepts text or image inputs and outputs text. It also offers the possibility of using plugins to interface with external sources. Browsing the Internet is an example of this, as shown by the use of Bing searches.

There are three main variants of the GPT-4 accessible through the OpenAI API for **paying customers**:

Model	Characteristics	Context Window	API	Local	Token Pricing	Training Data
GPT-4	Snapshot of gpt-4 from June 13th 2023. This endpoint is the one receiving constant updates. Optimized for chat.	8,192 tokens	Yes	No	$0.03 / 1K tokens (input) $0.06 / 1K tokens (output)	Dec 2023
GPT-4 Turbo	API name is gpt-4-turbo-preview. It excels in tasks like code generation and is intended to reduce cases of "laziness".	128k tokens	Yes	No	$0.01 / 1K tokens (input) $0.03 / 1K tokens (output)	Dec 2023
GPT-4 Vision	API name is gpt-4-vision-preview. GPT-4 with the ability to understand images, in addition to all other GPT-4 Turbo capabilities.	128k tokens	Yes	No	$0.01 / 1K tokens (input) $0.03 / 1k tokens (output)	Apr 2023

In general, **it is interesting to note that GPT models, unlike LLaMa models, are closed-source.** This means that we cannot run them locally or deploy our own endpoint.

Nevertheless, there is the possibility of customization through using the OpenAI **Fine-tuning API**. In addition, **they are not accessible through third-party providers such as Ollama, LLAMA API, or Hugging Face.**

Regarding the OpenAI Playground, the information of the model underneath is not clearly displayed, but it seems based on the GPT-4 model with DALL-E, browsing, and analysis capabilities.

Let's also have a look at the GPT-3 models available!

Model	Characteristics	Context Window	API	Local	Token Pricing	Training Data
GPT-3.5 Turbo	API name is gpt-3.5-turbo. It is the flagship model of this family. It is optimized for dialog and shows higher accuracy at responding in requested formats.	16,385 tokens	Yes	No	$0.0005 / 1K tokens (input) $0.0015 / 1K tokens (output)	Sep 2021
GPT-3.5 Turbo Instruct	API name is gpt-3.5-turbo-instruct. It focuses on accuracy and relevance in the responses, aligning closely with user needs.	4,096 tokens	Yes	No	$0.0015 / 1K tokens (input) $0.0020 / 1K tokens (output)	Sep 2021

The GPT-3.5 Turbo Instruct model presented in the table has been **fine-tuned to better follow the user's intentions via Reinforcement Learning from Human Feedback (RLHF)**.

RLHF is a technique where human feedback is integrated into the reinforcement learning process to guide and improve the learning outcomes of AI agents. This approach enables the development of models that better align with human values and preferences by incorporating human judgments, corrections, or preferences directly into the training cycle.

Finally, as we can observe from the table, it is worth noticing that the usage of any of the two GPT-3.5 models is considerably cheaper than GPT-4.

In this aspect, I normally recommend comparing the performance of each model in a set of well-defined examples to see if GPT-3.5 is enough for my task at hand, thus saving in the inference.

LLaMa Models - Meta AI

As we have discussed in the previous sections, **LLaMa models are free to download and run locally** or available to build customizations on top.

The LLaMa models are actually a big family of open models in terms of sizes and variants provided by **Meta AI**:

Model	Characteristics	Context Window	API	Local	Variant	Training Data
LLaMa 2	Base non-chat model for completion tasks.	4096 tokens	Yes LLAMA API	Yes*	7B 13B	Sep 2022

					70B	
LLaMa 2 Chat	Base LLaMa 2 with instruction tuning to turn simple completions into a chat model that follows the user intent.	4096 tokens	Yes LLAMA API	Yes* Ollama	7B 13B 70B	Sep 2022
LLaMa 2 Guard	Input-output safeguard model trained to detect toxic and harmful content.	4096 tokens	Yes**	No	7B	Sep 2022*

*Hugging Face also provides some variants of LLaMa, such as the base and chat models.**For the time being, LLaMa 2 Guard is accessible in the cloud environment of Amazon SageMaker.

Unlike GPT models that have coding capabilities integrated, **LLaMa models have dedicated models for coding with larger context windows** of 16k tokens achieved by a long-context fine-tuning process.

Model	Characteristics	Context Window	API	Local	Variant	Training Data
Code LLaMa	Base non-chat model for code completion tasks, like GitHub Copilot. It consists of a LLaMa 2 model with an additional training on code and long context fine-tuning.	16k	No	Yes Ollama	7B 13B 34B	Sep 2022*
Code LLaMa - Instruct	Derived from Instruct models. It excels in code completions and explanations on pieces of code.	16k	Yes LLAMA API	Yes Ollama	7B 13B 34B	Sep 2022*
Code LLaMa - Python	Fine-tuned for the Python programming language.	16k	No	Yes Ollama	34B	Sep 2022*

* The official cut-off date of LLaMa models is Sep 2022. Nevertheless, tuning data is more recent, dating back to 2023, especially for newer variants.

Google's LLM Family

Google has a long history of developing Large Language Models. With the introduction of the Gemini model, Google started its generation of multimodal LLMs.

In the context of Google's offerings, **Gemini emerges as its flagship model, being closed-source**, akin to ChatGPT. However, they recently launched **Gemma, which introduces a suite of open-source models derived from Gemini**, aimed at facilitating accessible AI development.

In this article, we will focus on both Gemini and Gemma:

Model	Characteristics	API	Local	Variants
Gemini	Closed-source model for general-purpose tasks (text generation, translation, question answering, code, etc)	Yes*	No	Ultra - Solve complex tasks. Pro - Broad range of tasks. Scalable. Nano - Efficient on-device use.
Gemma	Similar tasks to Gemini, but focused on integration into custom applications and research. It is a lightweight model.	Yes**LLAMA API	YesOllama Hugging Face	2B 7B

*There is an API for Gemini for commercial purposes (not open to deployments). Nevertheless, we can try out the model via its Playground interface.

**There is no native Google endpoint, but the model is accessible through Kaggle and Google Colab. More information is available in the official Google documentation.

As for the Gemini variants, Gemini Ultra is a multimodal model that excels in complex analysis tasks. Gemini Pro balances performance and scalability and stands out in tasks like code and text generation. Finally, Gemini Nano focuses on efficiency for on-device use, running on mobile and edge devices without heavy resource demands.

Regarding Google's models, not much information is publicly available about the context windows and the cut-off date of the training data at the time of writing.

Mistral AI LLM Family

Mistral AI is a French AI company founded in April 2023 by former employees of Meta Platforms and Google DeepMind. It focuses on developing open-source LLMs, emphasizing the importance of open-source software in response to proprietary models. The company offers models like Mistral 7B and Mixtral 8×7B with advanced language processing capabilities.

Model	Characteristics	Context Window	API	Local	Token Pricing	Training Data

Mistral	General-purpose model with 7B parameters. Optimized for the English language and code generation tasks.	32k	Yes* LLAMA API Hugging Face**	Yes Ollama Hugging Face**	$0.25 / 1M tokens (input and output)	Sep 2023
Mixtral	Most powerful general-purpose model. Fluent in English, French, Italian, German, Spanish, and strong in coding tasks. It is based on a 7B-parameter sparse Mixture-of-Expert architecture.	32k	Yes* LLAMA API Hugging Face**	Yes Ollama Hugging Face**	$0.7 / 1M tokens (input and output)	Dec 2023

*We can interact with models from Mistral AI by using their Playground interface called “La Plataforme”.

** Hugging Face also offers a test bed for real-time inference using its API for some models. It is also a third-party provider that allows you to deploy paid endpoints to the available models.

Regarding the Mixtral model, the "Mixture-of-Experts" architecture allows the model to divide its total capabilities across different specialized groups or “experts.” Each expert is optimized for specific types of tasks or data.

When processing an input, Mixtral dynamically engages only the relevant experts, allowing it to efficiently handle complex tasks by utilizing a fraction of its total parameters for each token. This method enhances Mixtral’s performance, making it adept at a variety of languages and tasks while optimizing computational resources. The name of the Mixtral model is indeed "Mixtral 8×7B" making reference to the fact that it utilizes 8 distinct groups of “experts”.

Finally, Mistral AI also offers three commercial models: Mistral Small, Mistral Medium and Mistral Large. If you are interested, you can find mode information about the models and pricing in the official documentation.

Miscellanea

Up to now, we have seen multiple models and variants of the most famous companies in the business. Nevertheless, there is much more to explore!

I would like to highlight the Falcon models from the Technology Innovation Institute, which is available in both Ollama and LLAMA API. There is also Microsoft Research’s Orca2 model which excels particularly in reasoning, also available in Ollama and Hugging Face.

You can read about other open-source, available models at 8 Top Open-Source LLMs for 2024 and Their Uses.

Recap: How Do I Select the Best LLM?

In this article, we have discussed several aspects to consider when selecting the best Large Language model for our target application. While the goal of this guide was to serve as a cheat sheet, there is no golden rule to select the best model straight away.

Based on my personal experience, I think defining how we would like to interface with the LLM is a good starting point since it will already reduce the number of models - and variants - available.

The second aspect to consider is the objective task that the model will carry out: generic chatbot, summarization expert, coding assistant, etc.

While we can always fine-tune a model for our desired task, we have seen that there are variants already fine-tuned for certain use cases that will save us time, computational resources, and technical expertise. Considering the context window needed for the application is also crucial to select the best model.

Finally, pricing is always a good indicator of which model to use. Sadly, most of the time, the project itself is limited by the initial inversion and the cost of maintaining the project over time. Regarding the price, it is always important to bear in mind that **training** and **fine-tuning** are non-free services to consider during the project development and actually quite time-consuming.

AUTHOR
Andrea Valenzuela

Andrea Valenzuela is currently working on the CMS experiment at the particle accelerator (CERN) in Geneva, Switzerland. With expertise in data engineering and analysis for the past six years, her duties include data analysis and software development. She is now working towards democratizing the learning of data-related technologies through the Medium publication ForCode'Sake.

She holds a BS in Engineering Physics from the Polytechnic University of Catalonia, as well as an MS in Intelligent Interactive Systems from Pompeu Fabra University. Her research experience includes professional work with previous OpenAI algorithms for image generation, such as Normalizing Flows.

TOPICS

Artificial Intelligence (AI) Python

Continue Your AI Journey Today!

TRACK

AI Fundamentals

10hrs hr

Discover the fundamentals of AI, dive into models like ChatGPT, and decode generative AI secrets to navigate the dynamic AI landscape.

See Details → Start Course

See More →

Related

BLOG

What is an LLM? A Guide on Large Language Models and...

BLOG

The Pros and Cons of Using LLMs in the Cloud Versus...

BLOG

8 Top Open-Source LLMs for 2024 and Their Uses

See More →