



THE HITCHHIKER'S GUIDE TO LLMs FOR EVENTS

FROM ZERO TO PRODUCTION

EP
EXPOPLATFORM



PART: 1

A Primer on LLMs

EP
EXPOPLATFORM

Foreword

Welcome to the world of Large Language Models (LLMs)! In this book, we explore the intricate process of deploying LLMs into real-world applications, with specific examples focused on the events industry. From understanding the foundational concepts of Language Models to navigating the complexities of production deployment, this book offers a comprehensive guide for practitioners in the field.

Throughout the pages of this book, readers will discover key insights into AI modeling techniques, the transformative impact of Transformer architecture, and the practicalities of working with Language Models and how they can be leveraged to deliver value in the events industry. Whether it's understanding model size and memory requirements or delving into the nuances of local model inference, each chapter provides valuable insights and strategies for effective LLM experimentation and development.

Moreover, readers will explore architectural patterns for LLMs, evaluation methodologies, and deployment strategies tailored to various use cases and domains. From customising and fine-tuning models to deploying on local machines, cloud platforms, or edge devices, this book equips readers with the knowledge and tools necessary to navigate the complexities of productionising Language Models. Whether you're a seasoned practitioner or a newcomer to the field, we invite you to embark on this transformative journey and unlock the full potential of Gen AI Model deployments.

Happy Deploying!

Table Of Contents

Part: 1 A Primer on LLMs

1. Dynamic Landscape of Artificial Intelligence	5
2. AI modelling	8
• Discriminative modelling	9
• Generative modelling	10
• Generative versus Discriminative modelling	11
• Generative modeling taxonomy	12
3. High-level Transformers architecture	16
• Attention	17
• Inputs and context window	18
• Embedding	19
• Encoder and Decoder	19
• Softmax output	20
4. Difference between various LLMs (architecture, weights and parameters)	21
5. Hugging Face, the house of LLMs	25
Summary	28

01

Dynamic Landscape of Artificial Intelligence

AI has emerged as one of the most transformative technologies of the modern era - revolutionising industries, reshaping economies, and redefining human capabilities. Its journey from theoretical concepts to real-world applications has been marked by significant milestones, breakthroughs, and challenges.

We can trace the evolution of AI from its conceptual origins to recent advancements by dividing it into 4 main sections:

1. Origins and early developments
2. The AI winter and resurgence
3. Rise of deep learning
4. Recent developments

Origins and early developments

The roots of AI can be traced back to ancient times, with mythological tales of artificial beings brought to life. However, the formal inception of AI as a field of study can be attributed to the [Dartmouth Conference in 1956](#), where the term "artificial intelligence" was coined. Early pioneers such as [Alan Turing](#), [John McCarthy](#), [Marvin Minsky](#), and others laid the groundwork for AI by exploring concepts like neural networks, machine learning, and symbolic reasoning.

The AI winter and resurgence

Despite initial optimism, AI experienced several periods of disillusionment known as "AI winters," characterised by funding cuts and waning interest due to unmet expectations. However, each winter was followed by a resurgence fueled by new breakthroughs and technological advancements. The 1980s witnessed the rise of expert systems, while the 1990s saw the emergence of machine learning algorithms like [Support Vector Machines](#) and [Hidden Markov Models](#).

Rise of deep learning

The turning point for AI came with the advent of deep learning, a subfield of machine learning inspired by the structure and function of the human brain's neural networks. Breakthroughs in computational power, coupled with vast amounts of data, enabled deep learning algorithms to achieve unprecedented performance in tasks such as image recognition, natural language processing, and speech recognition. Notable milestones include the development of [convolutional neural networks \(CNNs\)](#) and [recurrent neural networks \(RNNs\)](#), as well as the success of deep learning models like [AlexNet](#), [AlphaGo](#), and [GPT \(Generative Pre-trained Transformer\)](#).

Recent developments

In recent years, AI has continued to evolve at a rapid pace, with breakthroughs in areas such as [reinforcement learning](#), [federated learning](#), and [AI-driven drug discovery](#). Autonomous vehicles, healthcare diagnostics, and personalised recommendation systems are just a few examples of AI applications transforming industries and improving lives. Furthermore, the convergence of AI with other technologies like robotics, [blockchain](#), and the [Internet of Things \(IoT\)](#) is opening up new possibilities and fueling innovation across sectors.

As AI technologies continue to advance, concerns about their ethical and societal implications have become increasingly prominent. Issues such as bias in algorithms, job displacement due to automation, and the misuse of AI for surveillance or warfare have sparked debates and calls for regulation. Organisations and researchers are actively working to address these challenges through initiatives focused on fairness, transparency, and accountability in AI systems.

Looking ahead, the future of AI promises even greater advancements, driven by ongoing research efforts, investment in infrastructure, and collaboration among academia, industry, and governments.

Generative AI

Generative AI, a branch of artificial intelligence focused on creating new data from existing examples, has a rich history dating back several decades.

Generative AI traces back to the mid-20th century, initially exploring rule-based systems and early neural network models. However, computational limitations restricted significant progress until the late 20th century.

The resurgence of neural networks in the 1990s and 2000s, fueled by computing advancements and data availability, led to the development of powerful generative models like [Variational Autoencoders \(VAEs\)](#) and [Generative Adversarial Networks \(GANs\)](#).

The breakthrough introduction of GANs in 2014 revolutionised generative AI, enabling the creation of highly realistic synthetic data through adversarial training. This approach found applications in image generation, text-to-image synthesis, and style transfer.

Recent years have focused on enhancing the robustness, diversity, and controllability of generative models. Techniques like self-supervised learning and regularisation methods aim to improve model generalisation and mitigate issues like mode collapse and bias. Advancements in generative AI continue, with ongoing research exploring novel applications in areas such as drug discovery, content creation, and personalised recommendation systems. Ethical considerations, including deepfakes and misinformation, remain significant points of discussion and research in the field.



02 AI modelling

In data science and machine learning, modelling refers to the process of creating a mathematical representation or algorithm that can identify patterns, make predictions, or gain insights from historical data. It involves selecting and training a suitable algorithm on a dataset to learn the underlying relationships between input variables (features) and the target variable (output).

The goal is to build a model that generalises well to unseen data, allowing it to make accurate predictions or classifications on new instances. Modelling is a crucial step in extracting value from data and is used across various applications such as classification, regression, clustering, and recommendation systems.



Let's consider an example of teaching a robot to paint a landscape. In this scenario, the process of teaching a robot to paint a landscape involves several steps that can be likened to the concepts in modelling. The robot has to position its arm correctly on the x , y , and z axes. With every move, the x , y , and z coordinates will change. The robot should also be able to distinguish between colours and pick the correct colour from the palette. Pressure on the brush is also important to create stunning images. We can say that (x, y, z) coordinates of the robot's arm, colour, and brush stroke are the parameters that will control the final painting.

Image 1.1 Robot learning to paint. Generated via Stable Diffusion 2.

A machine learning model typically learns the model parameters through the process known as training.

In our example, the process of providing the robot with training data, which includes images of landscapes along with instructions on how to paint them, is similar to model training. The robot should learn from these examples by observing the colours, shapes, and patterns in the landscapes and how they relate to each other to be able to generalise well or create new landscapes that look like our original landscapes but are completely new.

This is a simple explanation of model training. In machine learning, model training also involves a loss function and an optimisation problem. Optimizer adjusts the model's parameters to minimise the loss function. At the minimum loss, we obtain the best-fitting parameters of the model.

Discriminative Modelling

Discriminative modelling is a subset of supervised machine learning wherein models learn the classification among the output classes of the data. By learning the boundary between classes, discriminative models can make predictions on unseen data by assessing which side of the boundary a new sample falls on. This approach is particularly effective for classification tasks where the primary goal is to accurately assign class labels to input data points, such as identifying objects in images or classifying emails as spam or non-spam.

Discriminative modelling is like a detective figuring out clues to solve a mystery. Instead of trying to understand everything about a situation, it focuses on the specific details that help make a decision.

For example, imagine you're trying to decide if a fruit is an apple or an orange. A discriminative model would look at the features like colour, size, and texture to make the decision. It doesn't need to know everything about apples and oranges; it just needs to focus on the important details to tell them apart.

In the real world, discriminative modelling is used in many ways:

1. **Spam Filtering:** Your email provider uses discriminative modelling to decide if an email is spam or not. It looks at specific features like keywords, sender information, and formatting to make this decision.
2. **Image Recognition:** When you upload a photo to social media and it automatically tags your friends, that's discriminative modelling at work. The model looks at specific features in the image to recognise faces and match them to your friends' profiles.
3. **Sentiment Analysis:** Companies use discriminative modelling to analyse customer reviews and social media posts to understand if people are feeling positive or negative about their products or services. This helps them make decisions on how to improve.

In each of these examples, the discriminative model focuses on specific details to make a decision or prediction without needing to understand everything about the situation.

Generative modelling

Generative modelling is a method in machine learning where a model learns the underlying structure of a dataset to generate new, similar data points.

In practice, suppose we have a dataset containing images of cats. We can *train* a generative AI model on this dataset to capture the complex relationships between pixels in images of cats. Then, we can *sample* from this model to create novel, realistic images of cats that did not exist in the original dataset.

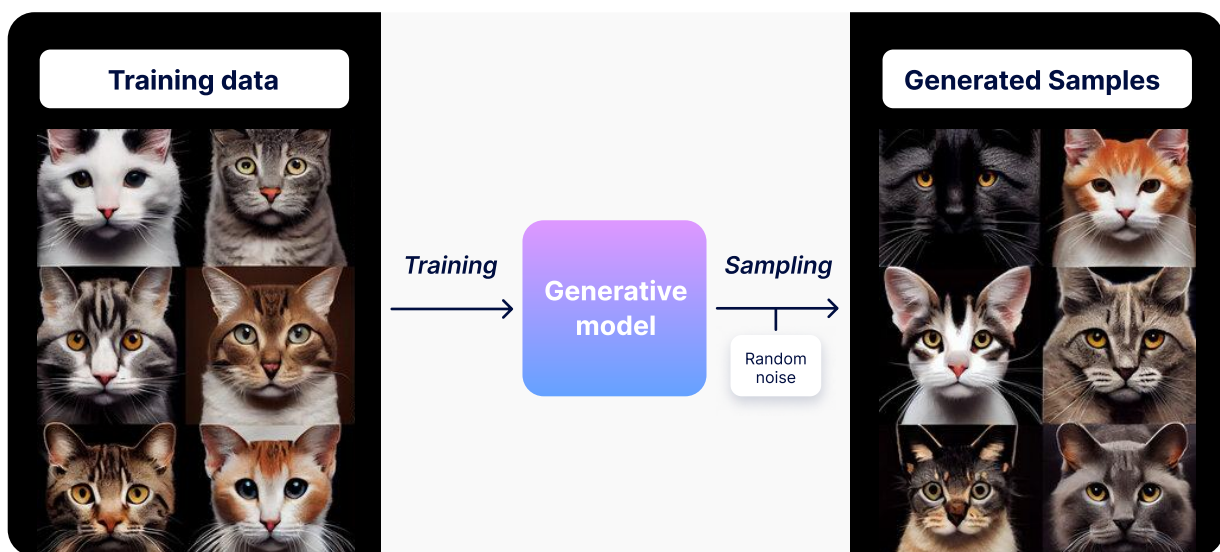


Image 1.2 Training and sample generation process of a Generative model

Image 1.2 shows how a generative model learns to create new cat images. To generate new cat images, random noise is input into the model, which then decodes it to generate realistic cat images. The process of decoding is learned during model training.

Generative modelling involves creating a model that can produce new data resembling the original dataset. For instance, in image generation, this task is formidable due to the multitude of potential pixel configurations compared to the limited subset that represents recognisable images. Additionally, generative models must incorporate *probabilistic* elements rather than *deterministic* calculations.

If the model relies solely on fixed calculations, like averaging pixel values, it lacks generativity as it produces the same output consistently. Instead, generative models require stochastic components to introduce randomness, enabling the generation of diverse samples. Essentially, the aim is to replicate the underlying probabilistic distribution of the training data, allowing the model to generate novel observations that align with the dataset's characteristics.

Generative versus Discriminative modelling

Discriminative modelling, in contrast to generative modelling, focuses on directly learning the boundary between different classes or categories within a dataset. Unlike generative models, which aim to capture the underlying probability distribution of the entire dataset to generate new samples, discriminative models prioritise making predictions based on the input features alone. For example, in the context of image classification, a discriminative model might be trained to distinguish between images of cats and dogs based solely on the visual features present in the images, without explicitly learning how to generate new images of cats or dogs.

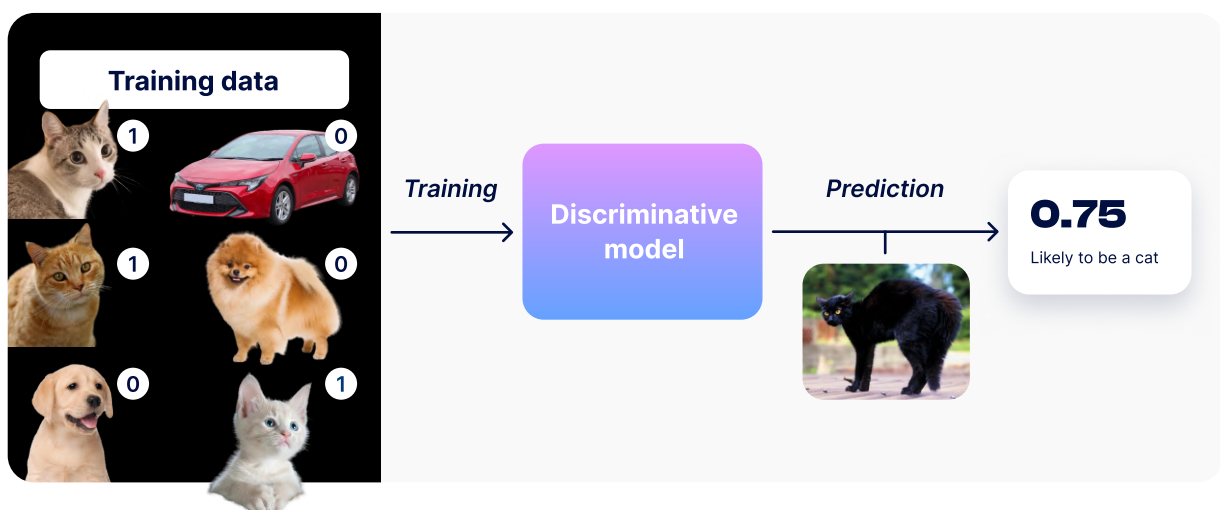


Image 1.3 Training process and predicting on unseen data in a Discriminative model

Similar to Image 1.2, here in Image 1.3, we demonstrate the training and prediction process of a discriminative model. The training data consists of images of cats labeled as 1 and non-cats labeled as 0. With enough data, we hope the model learns to classify cat images as 1 and others as 0. We verify this by passing an unseen cat image to the trained model.

Another concrete example of discriminative modelling can be found in natural language processing (NLP), where a model might be trained to classify text documents into different categories, such as spam or non-spam emails, sentiment analysis (positive or negative sentiment), or topic classification. In these cases, the discriminative model learns to identify patterns in the input data that are indicative of each class, without needing to understand the underlying structure of the entire dataset or generate new text samples.

Overall, discriminative modelling is focused on making accurate predictions for specific tasks by learning decision boundaries directly from the data, without the need to model the entire data distribution.

This book primarily focuses on generative LLMs; therefore, discriminative modelling is beyond its scope. For further insight into discriminative modelling, please consult the [Hands-on Machine Learning Aurélien Géron O'Reilly](#) book.

Generative modelling taxonomy

Let's start by creating a simple generative model in two dimensions. In image 1.3 you have a set of points, generated by an unknown rule.

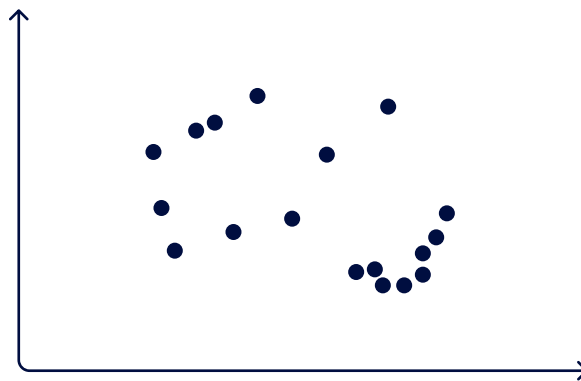


Image 1.4 Dataset points when plotted on 2 dimensions

After little thought, we decide that our model should look like in image 1.4 i.e. a rectangular box from which we will sample new points. With this model (or rule), points outside our box should never be sampled.

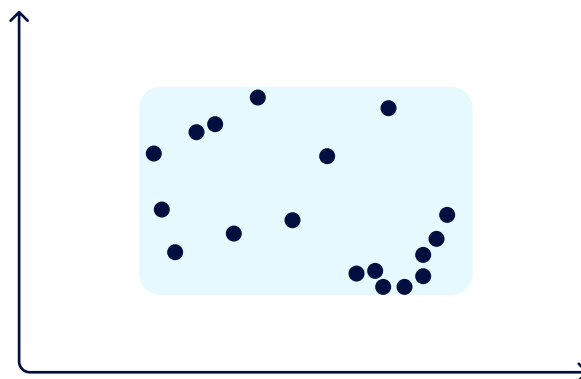


Image 1.5 Our model boundary based on observed values

To generate new observations, you can simply choose a point at random with in the box.

The objective of generative modelling is to find a distribution (or a rectangular box in our case) that matches with true data-generating distribution. True data-generating distribution means the source of truth. In our example, it is the map of Australia.

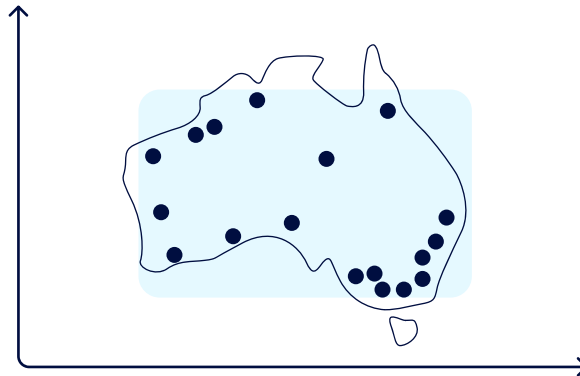


Image 1.6 Ground truth of the data points

In image 1.6 we have sampled three points from our model. Green and orange points are from true distribution because they sit on the land. But the orange point should never have been generated by our model because it sits outside the box. Red point does not appear to represent the true data-generating distribution because it sits on sea.

Orange point sits outside the box, so our model made a mistake by picking a point outside the box. This is a model error. We train the model to minimise such errors.

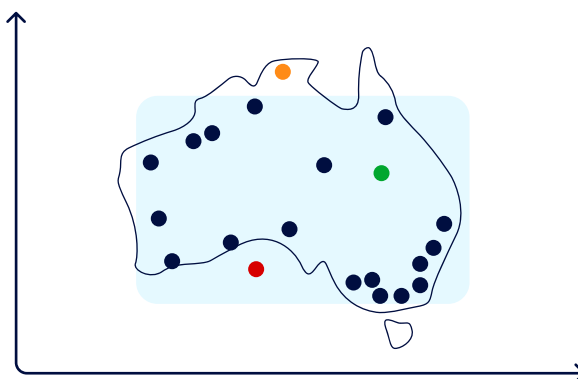


Image 1.7 Points generated using a simple model

Despite its shortcomings, the model is easy to understand and to sample from.

Almost all the generative models are trained to sample from an underlying distribution. The closer to true data-generating distribution the model is, the better it is.

Although all varieties of generative models share the common goal of addressing the same task, each adopts slightly distinct methodologies for modelling the density function $P_{\theta}(x)$.

The density function $P_{\theta}(x)$ represents the likelihood of observing the data x given a set of model parameters θ . In simpler terms, it tells us how probable it is to see a certain dataset, like images or text, based on the specific characteristics or features represented by θ . Generative models use this function to understand and capture the patterns and structures in the data, allowing them to generate new, similar samples that follow the same underlying rules as the original dataset.

We can trace the evolution of AI from its conceptual origins to recent advancements by dividing it into four main sections:

1. The first approach entails modelling the density function explicitly, albeit with constraints to ensure its tractability (i.e., computability).
2. The second approach involves explicitly modelling a tractable approximation of the density function.
3. The third approach implicitly models the density function via a stochastic process that directly generates data.



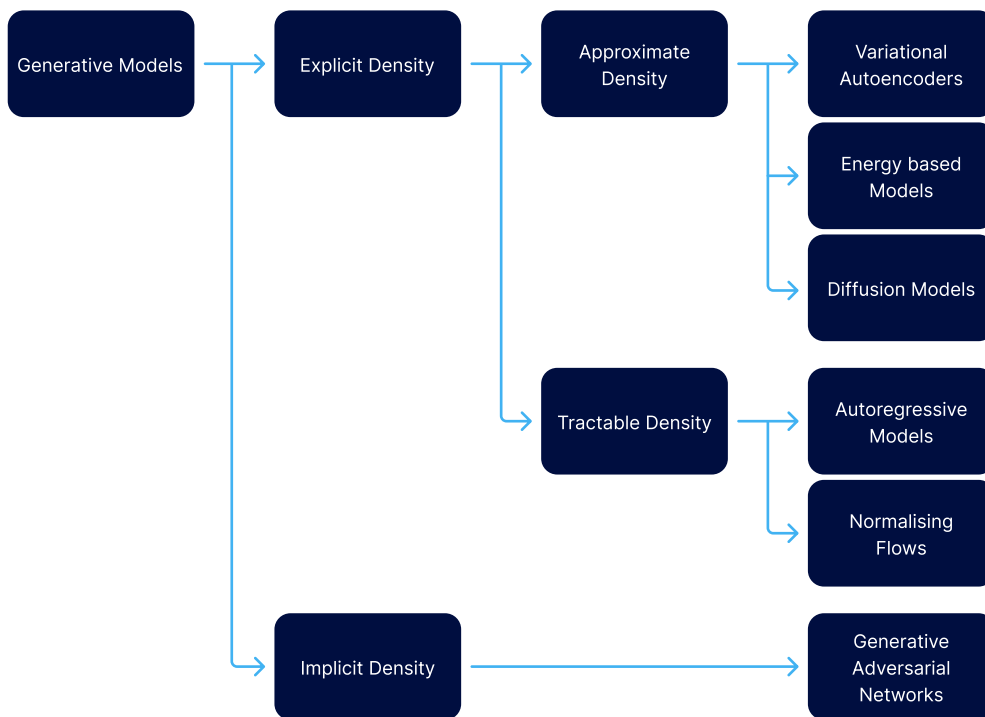


Image 1.8 Generative Model Taxonomy

Source: Generative Deep Learning by David Foster O'Reilly

Implicit density models don't try to figure out the probability density. Instead, they focus on making a process that creates data directly. A well-known example is a [generative adversarial network](#). Explicit density models are split into those that directly work on the density function (tractable models) and those that work on an approximation of it.

Tractable models have rules for the model's design so that the density function is easy to calculate. For example, autoregressive models arrange the input features in a specific order so that the output can be made step by step, like word by word or pixel by pixel. Normalising flows use simple, reversible functions on a basic distribution to make more complex ones.

Approximate density models include variational autoencoders, which add a hidden variable and work on an approximation of the combined density function. Energy-based models also use approximate methods, but through Markov Chain sampling instead of variational methods. Diffusion models train a model to slowly clean up a picture that's been messed up to approximate the density function.

The main thing connecting all these types of generative models is deep learning. Most advanced generative models use a deep neural network at their core because they can be trained from scratch to understand the complex connections in the data.

High-level Transformers architecture

[Recurrent neural networks \(RNNs\) like LSTMs and GRUs](#) are autoregressive models that process sequential data one token at a time. To understand the context these models constantly update a hidden vector. This was considered the most sophisticated technique to generate text until 2017, when Google Brain paper, titled [“Attention Is All You Need”](#), changed the landscape of text generation forever with the introduction of Transformers architecture.

A key downside of RNN approach was that the algorithm does not lend itself to parallelisation, as it must process sequences one token at a time. Transformers only rely on the attention mechanism, and do not require complex recurrent or convolutional architectures for sequential modelling. Therefore they are highly parallelisable.

In June 2018, OpenAI introduced Generative Pre-Trained Transformers (GPTs) in the paper [“Improving Language Understanding by Generative Pre-Training”](#), where authors show how a Transformer architecture can be trained on a huge amount of text data to predict the next word in a sequence and then subsequently fine-tuned to specific downstream generative modelling tasks.

Task	Datasets
Natural language inference	SNLI, MultiNLI, Question NLI, RTE, Sci Tail
Question answering	RACE, Story Cloze
Sentence similarity	MSR Paraphrase Corpus, Quora Question Pairs, STS Benchmark
Classification	Stanford Sentiment Treebank-2, CoLA

Image 1.9 List of the different tasks and datasets used in GPT paper. [Source](#)

The process of training a model on a large corpus of text to predict the next word in a sequence given the previous words is called pre-training. This teaches the model to understand the structure and patterns of natural language. Transformers in GPT refers to the transformer architecture.

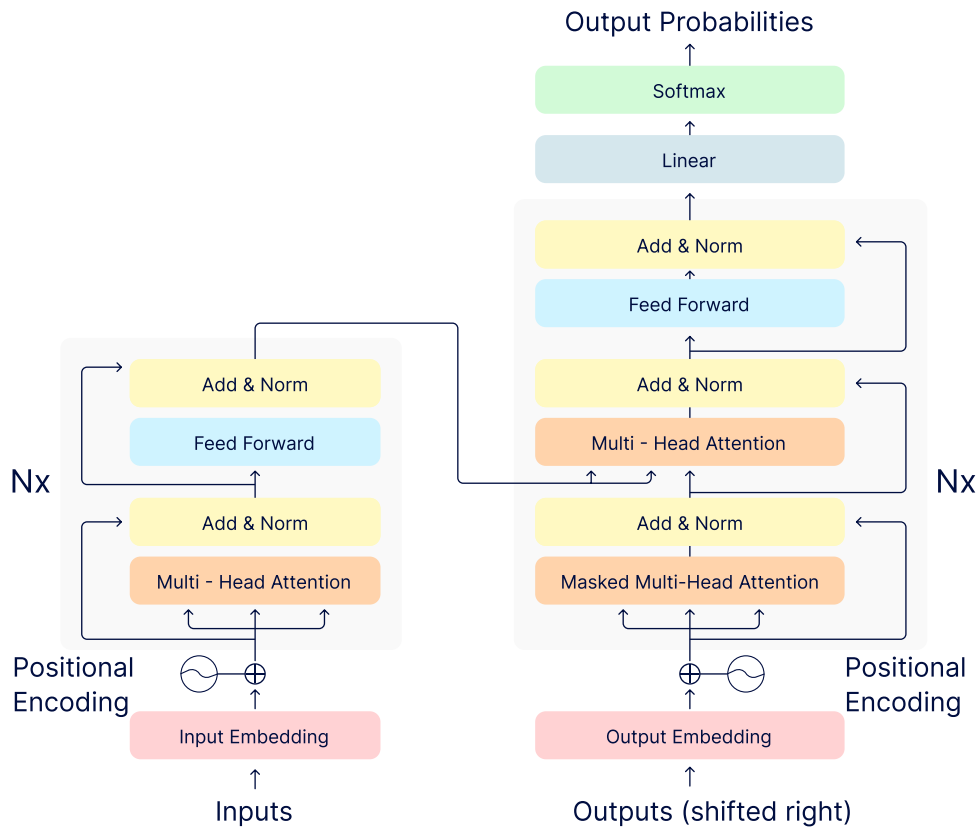


Image 1.10 Transformers architecture. Source: Vaswani et al., "Attention is all you need", arXiv, 2017

Attention

Consider the following sentence fragment taken from the amazing book [Generative Deep Learning](#) by David Foster:

"The pink elephant tried to get into the car but it was too"

What do you think the next word should be and how do we know it should be something synonymous with "big" ?

There are other words in the sentence that help us make our decision. Had it been "a sloth" instead of "an elephant", one would have been more likely to guess "slow" instead of "big". Had it been "a swimming pool" instead of "the car", one would have been more likely to guess "afraid" instead of "big".

If the elephant was trying to "squish" "the car, one might have chosen "fast" as the last word, with "it" now referring to the car.

Some words in the sentence are not important at all. For example, had there been a green or red elephant, it would not have influenced your choice of final word.

Just like we are paying attention to certain words in the sentence and ignoring others, the attention mechanism in the Transformers is designed to do exactly this. This makes it highly adaptable, as it can decide where it wants to look for information at inference time.

Inputs and context window

The input prompt is stored in a construct called the input “context window”. It is measured by the number of tokens it holds. The size of the context window varies widely from model to model.

Tokens are the individual units (words, punctuation marks, etc.) resulting from the process of tokenisation, which involves breaking down a text into smaller components for analysis or processing in machine learning tasks.

Earlier generative models could hold only [512–1024 input tokens](#) in the context window. However, more recent models can hold [upwards of 200,000 tokens](#) at the time of writing. The model’s input context window size is defined during model design and pretraining.

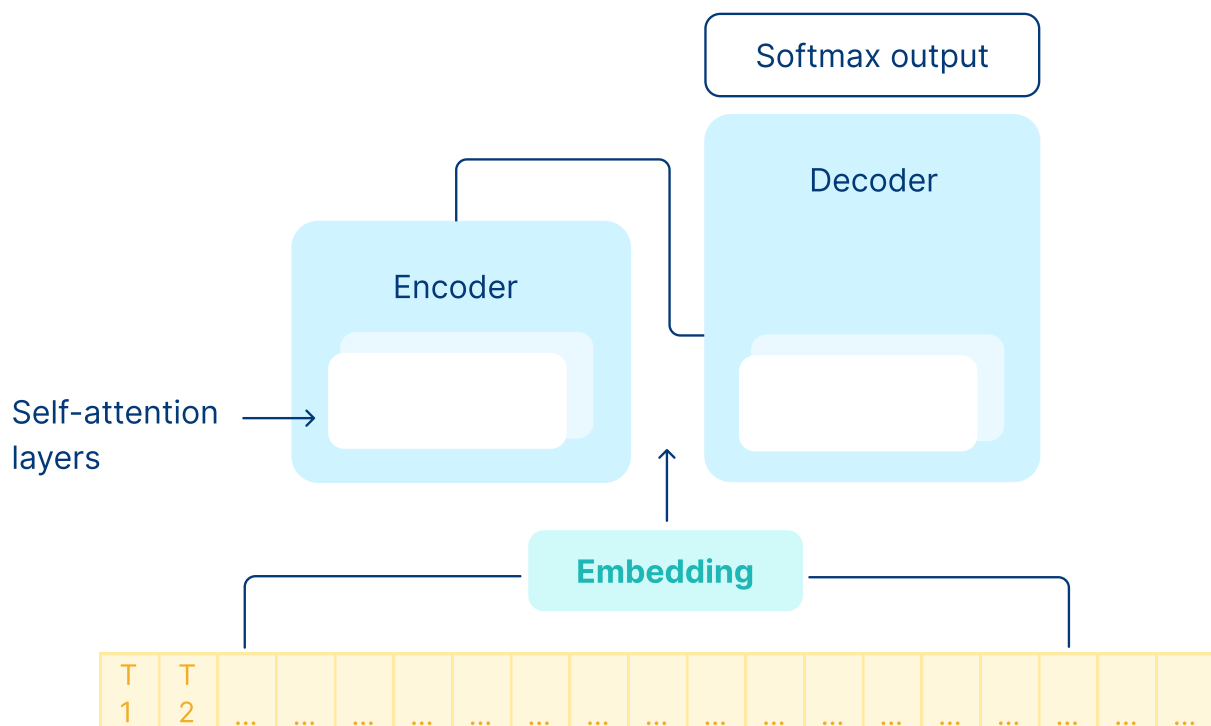


Image 1.11 Input token context window

Embedding

Embeddings are vector representations of words or entities in a high-dimensional space, where words with similar meanings are closer to each other. In natural language processing and machine learning, embeddings are used to capture semantic relationships and contextual information.

Embeddings in Transformers are learned during model pretraining and are actually part of the larger Transformer architecture. Each input token in the context windows is mapped to an embedding. These embeddings are used throughout the rest of the Transformer neural network, including the self-attention layers.

Encoder and Decoder

Encoder projects a sequence of input tokens into a latent vector space that represents that structure and meaning of the input. The latent vector space representation is learned during model pretraining.

The attention weights are passed through the rest of the Transformer neural network, including the decoder. The decoder uses the attention-based contextual understanding of the input tokens to generate new tokens, which ultimately “completes” the provided input. That is why the base model’s response is often called a completion.



Softmax output

The softmax output layer generates a probability distribution across the entire token vocabulary in which each token is assigned a probability that it will be selected text. Typically the token with the highest probability will be generated as the next token but there are mechanisms like temperature, top-k & top-p to modify the next token selection to make the model more or less creative.

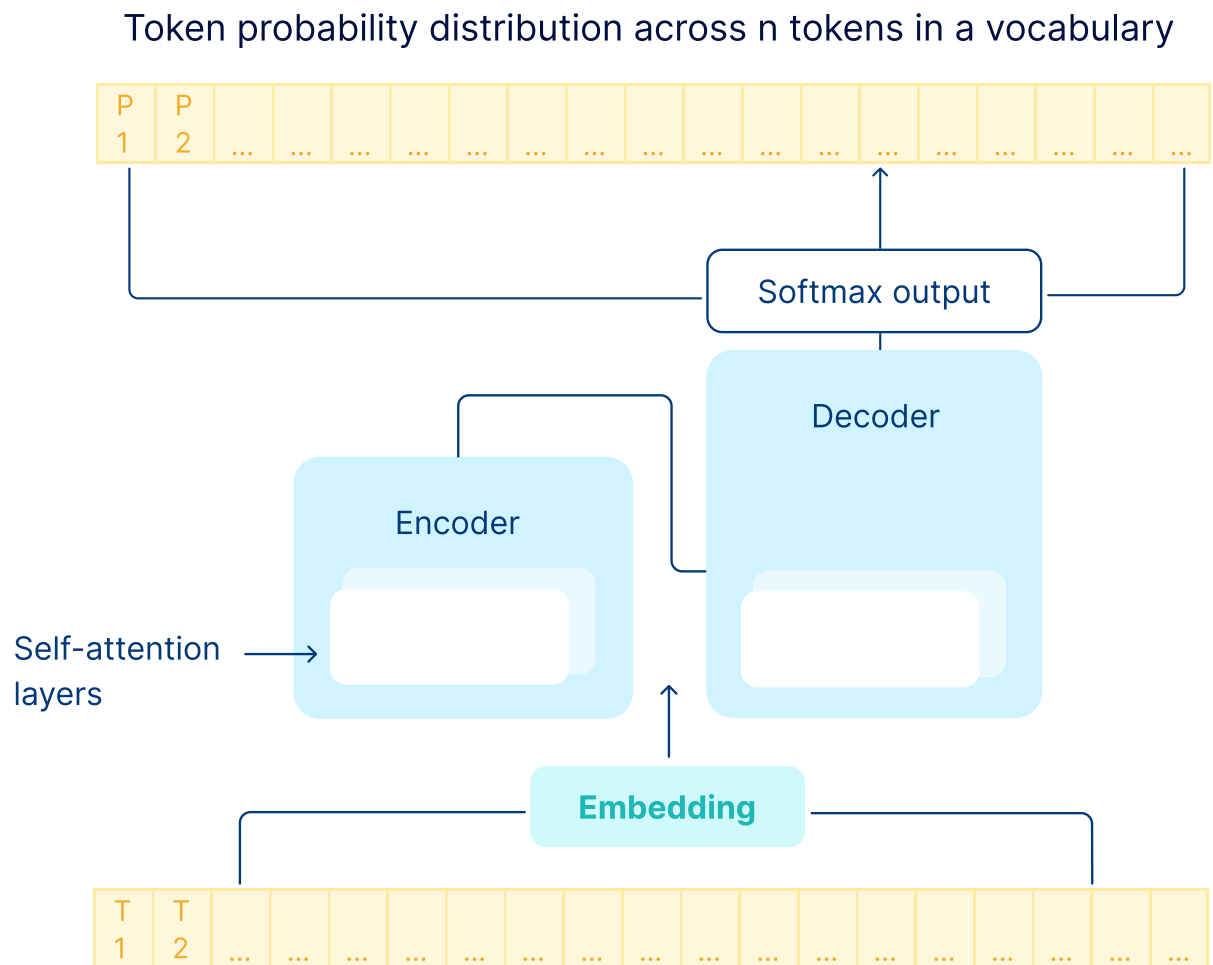


Image 1.12

Input token context window, encoder decoder blocks and softmax output payer

04

Difference between various LLMs

Encoder only

Or also known as autoencoders, are pre-trained using a technique called masked language modelling (MLM), which randomly masks input tokens and tries to predict the masked tokens.

The quick brown fox `<MASKED>` over the lazy dog.

While transformers predict the next token based solely on the tokens that come before it, masked language models consider the entire sequence of tokens when predicting a masked token.

Encoder-only models are best suited for language tasks that utilise the embeddings generated by the encoder, such as semantic similarity or text classification because they use bidirectional representations of the input to better understand the full context of a token — not just the previous tokens in the sequence. However they are not particularly useful for generative tasks that continue to generate more text.

An example of a well known encode-only model is [BERT](#).

Decoder only

Or autoregressive models are pre-trained using unidirectional causal language modelling (CLM), which predicts the next token using only the previous tokens — every other token is masked.

The quick brown fox jumps over the lazy `<PREDICT>`

Decoder-only, autoregressive models use millions of text examples to learn a statistical language representation by continuously predicting the next token from the previous tokens. These models are the standard for generative tasks, including question-answer. The families of GPT-3, Falcon, and Llama models are well-known autoregressive models.

Encoder-decoder models

Often also called sequence-to-sequence models, use both the Transformer encoder and decoder. They were originally designed for translation, and are also very useful for text-summarisation tasks like T5 or FLAN-T5.

Many multimodal models like DALL-E, Stable Diffusion, and Gemini incorporate architectures that utilise both encoder and decoder blocks of the transformer architecture.

Weights

In 2022, researchers at DeepMind released a paper [Hoffmann et al. \(2022\)](#), that compared the performance of various models and dataset size combinations. In the paper, they also pretrained a model called Chinchilla, that showcases how a model, by using significantly more tokens for training than its peers but considerably smaller and trained for much longer, outperforms much bigger models. The paper contributed to a significant model size to dataset size ratio that is now referred to as Chinchilla scaling laws. The paper claimed that the optimal training data size (measured in tokens) should be 20 times the number of model parameters and that anything below that 20x ratio is potentially overparameterised and undertrained.

Model	Task	Task	Task	Datasets
Chinchilla	70 B	1.4 T	1.4 T	Compute-optimised
Llama-65B	65 B	1.3 T	1.4 T	Compute-optimised
GPT-3	175 B	3.5 T	300 B	Overparameterised for dataset size
OPT-175B	175 B	3.5 T	180 B	Overparameterised for dataset size
BLOOM	176 B	3.5 T	350 B	Overparameterised for dataset size
Llama2-70B	70 B	1.4 T	2.0 T	Better than compute-optimised

Image 1.13 Chinchilla scaling laws for given model size and dataset size. Source: Generative AI on AWS O'reilly

According to Chinchilla scaling laws, the 175+ billion parameter models should be trained on 3.5 trillion tokens. Instead, they were trained with 180–350 billion tokens — an order of magnitude smaller than recommended. In fact, the more recent [Llama 2](#) 70 billion parameter model was trained with 2 trillion tokens — greater than the 20-to-1 token-to-parameter ratio described by the paper. This is one of the reasons that Llama 2 outperformed the original Llama model on various benchmarks.

Attention Layers & Parameters

Most of the model cards explain the type of attention layers the model has and how your hardware can exploit it to full potential. Most common open-source models also document the parameters that can be tuned to achieve optimum performance based on your dataset by tuning certain parameters.

The original Transformer paper [Vaswani et al. \(2017\)](#), used multi-head attention. Self-attention is very computationally expensive as it calculates n square pairwise attention scores between every token in the input with every other token. Since then, much research has been specifically targeted at attention layers to make them more computationally optimised, such as [FlashAttention](#) and [grouped-query attention \(GQA\)](#).



Encoder-decoder models

Often also called sequence-to-sequence models, use both the Transformer encoder and decoder. They were originally designed for translation, and are also very useful for text-summarisation tasks like T5 or FLAN-T5.

Many multimodal models like DALL-E, Stable Diffusion, and Gemini incorporate architectures that utilise both encoder and decoder blocks of the transformer architecture.

Weights

In 2022, researchers at DeepMind released a paper [Hoffmann et al. \(2022\)](#), that compared the performance of various models and dataset size combinations. In the paper, they also pretrained a model called Chinchilla, that showcases how a model, by using significantly more tokens for training than its peers but considerably smaller and trained for much longer, outperforms much bigger models. The paper contributed to a significant model size to dataset size ratio that is now referred to as Chinchilla scaling laws. The paper claimed that the optimal training data size (measured in tokens) should be 20 times the number of model parameters and that anything below that 20x ratio is potentially overparameterised and undertrained.

Model	Task	Task	Task	Datasets
Chinchilla	70 B	1.4 T	1.4 T	Compute-optimised
Llama-65B	65 B	1.3 T	1.4 T	Compute-optimised
GPT-3	175 B	3.5 T	300 B	Overparameterised for dataset size
OPT-175B	175 B	3.5 T	180 B	Overparameterised for dataset size
BLOOM	176 B	3.5 T	350 B	Overparameterised for dataset size
Llama2-70B	70 B	1.4 T	2.0 T	Better than compute-optimised

Image 1.13 Chinchilla scaling laws for given model size and dataset size. Source: Generative AI on AWS O'reilly

05

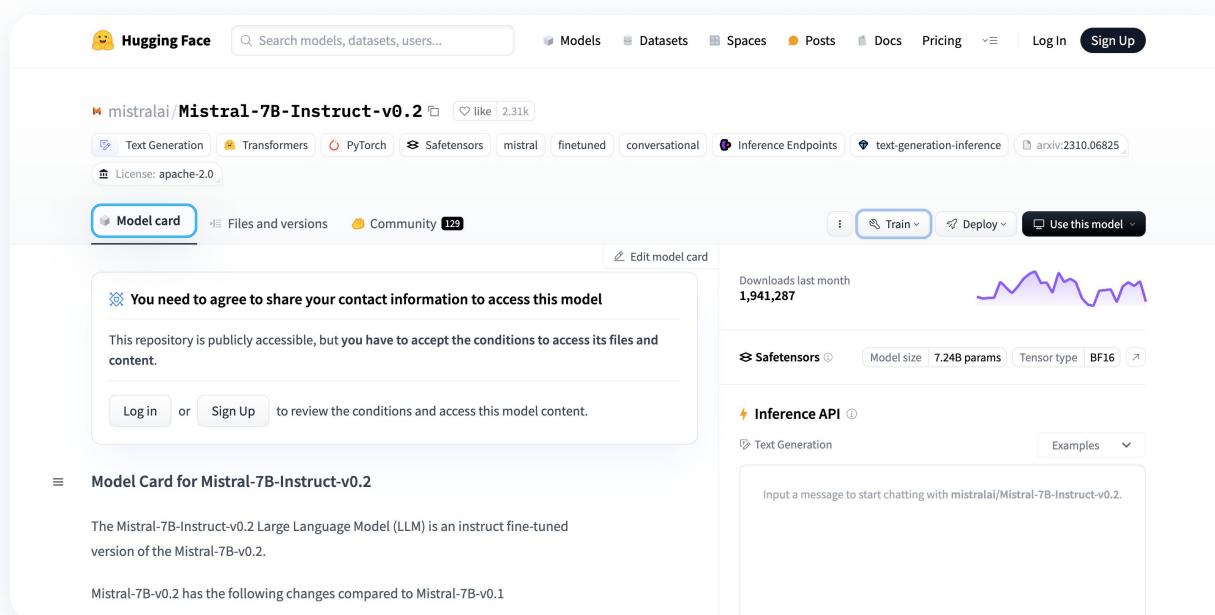
Hugging Face, the house of LLMS

[Little guide to building Large Language Models](#) by [Thomas](#), co-founder of Hugging Face.

Hugging Face is a platform that provides easy access to state-of-the-art natural language processing (NLP) models, including Large Language Models (LLMs), through open-source libraries. It serves as a hub for the NLP community, offering a repository of pre-trained models and tools that simplify the development and deployment of language-based applications.

The platform is particularly valuable for open-source LLMs because it democratizes access to powerful models, allowing developers and researchers to leverage cutting-edge capabilities without the need for extensive computational resources or expertise in model training.

Model card - The Hugging Face model card page provides comprehensive information and documentation for understanding and using pre-trained models, offering insights into their capabilities, performance, and potential applications.



The screenshot shows the Hugging Face model card for 'mistralai/Mistral-7B-Instruct-v0.2'. The page features a navigation bar with 'Hugging Face' and a search bar. Below the navigation, the model name is displayed with a 'like' count of 2.31k. The model is categorized under 'Text Generation', 'Transformers', 'PyTorch', 'Safetensors', 'mistral', 'finetuned', 'conversational', 'Inference Endpoints', 'text-generation-inference', and 'arxiv:2310.06825'. The license is 'apache-2.0'. The 'Model card' tab is selected, showing a 'You need to agree to share your contact information to access this model' message. The 'Downloads last month' is 1,941,287. The 'Safetensors' section shows 'Model size: 7.24B params' and 'Tensor type: BF16'. The 'Inference API' section is also visible.

Image 1.14 Model card page of Mistral 7B Instruct model on HuggingFace

Model inference section to interact with the model.

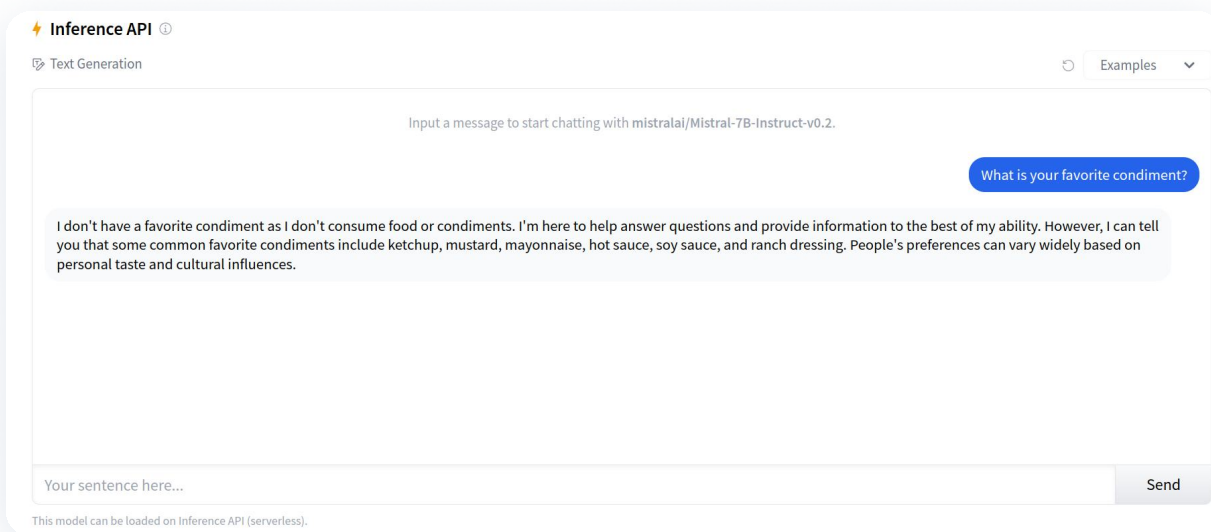


Image 1.15 LLM Inference window on HuggingFace model page

Deploy model shortcuts that provide boilerplate code to deploy the model on the different clouds.

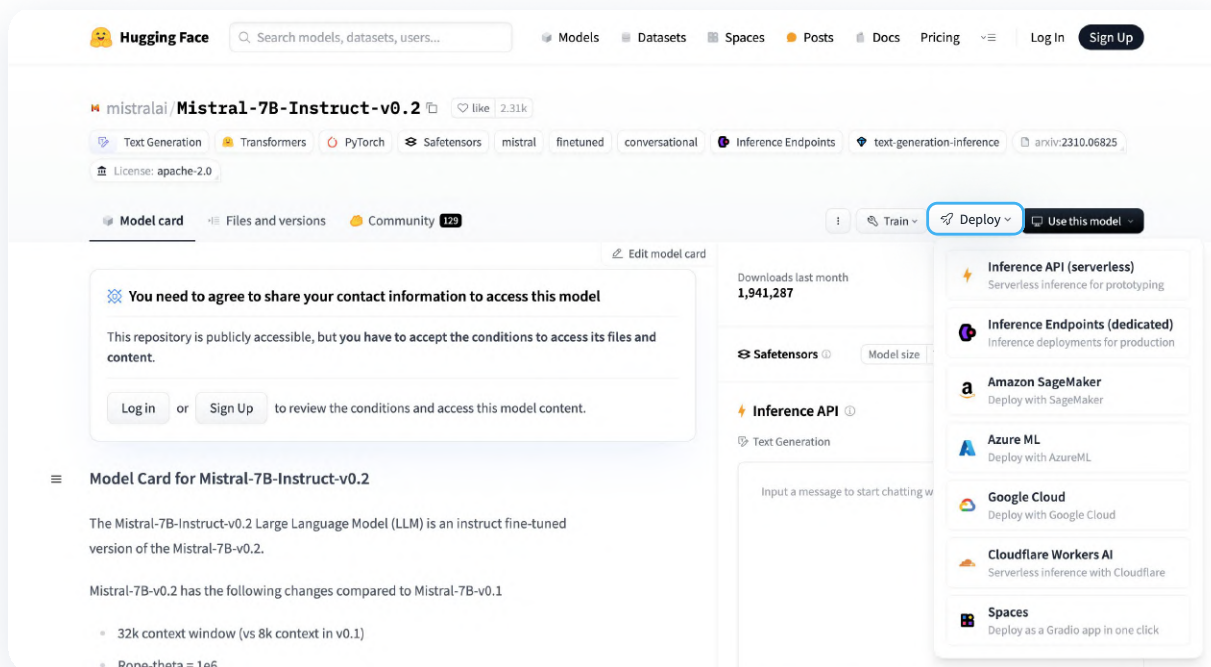


Image 1.16 Model deployment boiler plate code on infrastructure of different cloud providers

Train model option that shows boilerplate code to train the model on the respective cloud providers.

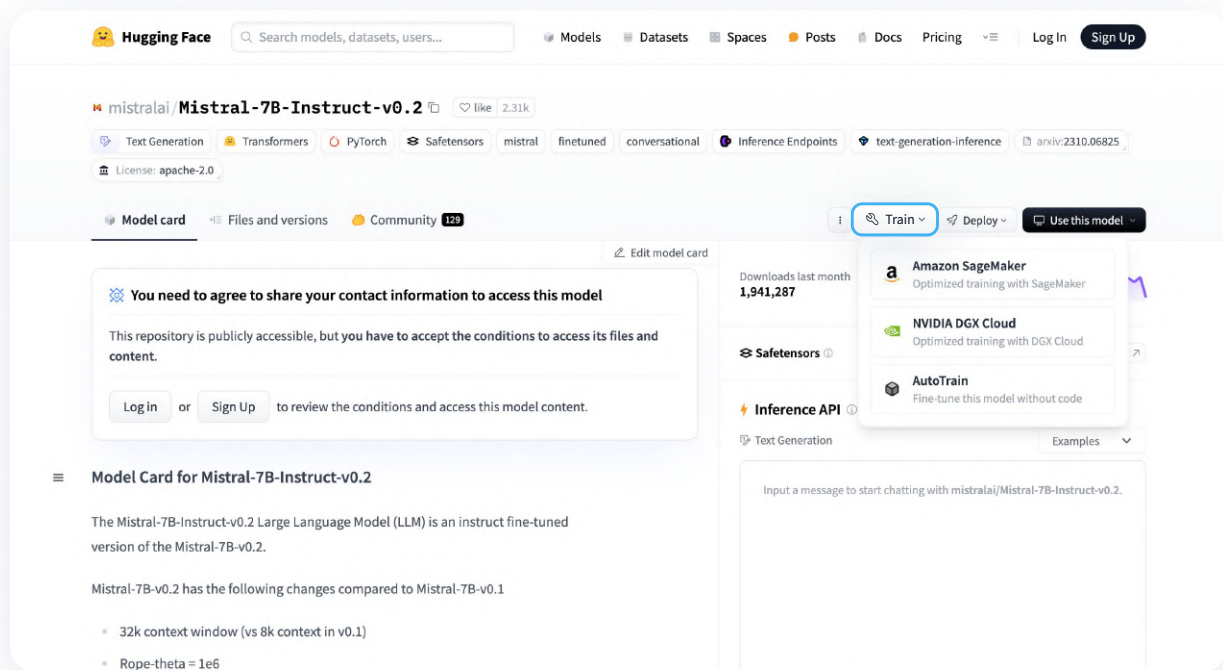


Image 1.17 Shortcuts to boilerplate code to train model on multiple vendor's infrastructure

Model files can also be downloaded manually or investigated directly from the model repository page.

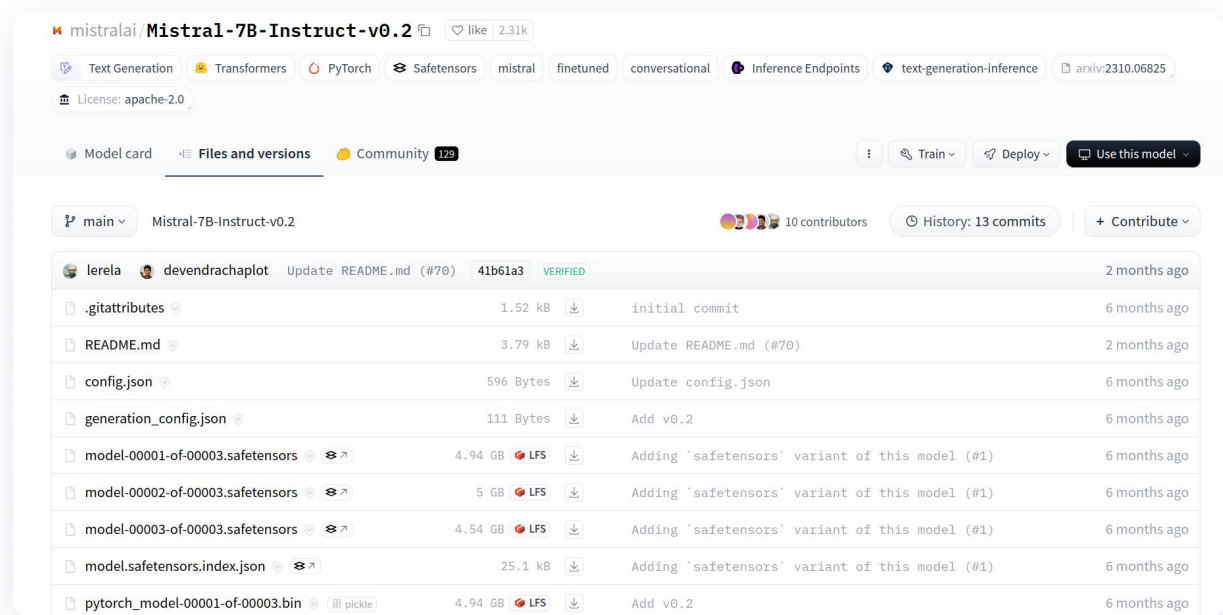


Image 1.18 Model weights and tokenizer on HuggingFace repository

Summary

In this introductory chapter on productionising large language models, we first explore the concept of generative modelling. We differentiate between discriminative and generative modelling. Then we create a "hello world" model of generative AI. Later, we explore the Transformer architecture at a high level without delving into the mathematical details. We also understand the differences between various LLMs, such as encoder-only models, encoder-decoder models, and attention layers.

Using the Chinchilla scaling laws, we understand the ratio of model training data to the number of parameters. This ratio is critical to the performance of any large language model. Lastly, we explore the UI of Hugging Face, which is the home of open-source machine learning models.

Our next chapter "How to play with LLMs" delves into various facets of interacting with large language models. It begins by addressing considerations around model size and memory requirements. Subsequently, it explores local model inference techniques, including quantisation methods and different transformer architectures, along with specific tools like GPT4All, LM Studio, llama.cpp, and Ollama.

The chapter also discusses utilising Google Colab for model experimentation and AWS SageMaker Studio, Studio Lab, SageMaker Jumpstart, and Amazon Bedrock for model deployment. Lastly, it provides guidance on deploying Hugging Face models onto SageMaker endpoints for real-time inference.

