Large Language Models (LLMs)

Open Educational Resources for Spatial Data Infrastructures

In this tutorial, you will learn the fundamentals of Large Language Models (LLMs), including how they are created and how they generate text. You will also learn about their limitations and techniques for overcoming those limitations and extending the capabilities of LLMs.

ein Kooperationsprojekt, empfohlen durch:

gefördert durch:

Ministry of Culture and Science
of the State of
North Rhine-Westphalia

1. Overview

In this tutorial, you will learn the fundamentals of Large Language Models (LLMs), including how they are created and how they generate text. You will also learn about their limitations and techniques for overcoming those limitations and extending the capabilities of LLMs.

In the practical section, we will use:

- An LLM, such as ChatGPT, to generate PyQGIS code for performing spatial analysis
- QGIS software to execute the PyQGIS code
- Smolagents library to create an AI Agent
- Google Colab to set up and run the agent
- OpenRouter API to connect the agent to an LLM.
- OSM Nominatim API to convert place names to coordinates for the agent.
- Open-Meteo API to fetch real-time weather information for the agent.

The tutorial is structured as follows:

1. Overview
2. Background
    - 2.1 How Large Language Models are Created
        - 2.1.1 Pre-Training
        - 2.1.2 Fine-Tuning and Instruction Tuning
        - 2.1.3 Reinforcement Learning from Human Feedback (RLHF)
    - 2.2 How Large Language Models Generate Text
        - 2.2.1 Prompt to Tokens
        - 2.2.2 Predicting the Next Token
        - 2.2.3 Controlling the Text Generation Process
    - 2.3 Reasoning Models
    - 2.4 Model Distillation
    - 2.5 Multimodal LLMs
    - 2.6 Limitations of LLMs
    - 2.7 Extending LLM Capabilities
    - 2.8 Tools, Frameworks, and Software for Working with LLMs
3. Practical Section
    - 3.1 Part 1: Using LLMs to perform Spatial Analysis
        - 3.1.1 Setting up the environment and data
        - 3.1.2 Performing Accessibility analysis using an LLM
    - 3.2 Part 2: Creating an AI Agent
        - 3.2.1 Setting up the Google Colab environment

3.2.2 Generating the OpenRouter API Key
4. Discussion and Conclusion
5. References

Some basic knowledge of QGIS and Python is beneficial. However, you can still proceed with limited knowledge of these areas, as this tutorial provides the required code and the QGIS steps to follow. You will need about 120 minutes to complete the tutorial.

The tutorial has been developed at 52°North Spatial Information Research GmbH in collaboration with the Institute for Geoinformatics (IFGI) at the University of Münster. Authors are James Ondieki, Albert Remke, and Simon Jirka.

You are free to use, alter, and share the content of the tutorial under the terms of the CC BY-SA 4.0 license, unless explicitly stated otherwise for specific parts of the content. All logos used are generally excluded. Any code provided with the tutorial can be used under the terms of the MIT license. Please see the full license terms:
https://github.com/oer4sdi/OER-LLMs/blob/master/LICENSE.md.

The tutorial can be referenced as follows: “OER-LLMs”, OER4SDI project / University Münster, CC BY-SA 4.0.

2. Background

Ever since OpenAI released the first version of ChatGPT in the form of a web-based chatbot, the use, research, development or general interest in LLMs has been increasing. You have probably used it to explain a concept, summarize a document, explain a bug in your code, generate code or translate some text. The applications are unlimited. In fact, just five days after launch, ChatGPT became the fastest-growing consumer application in history, reaching over a million users in a week.

In this tutorial, we will explore what LLMs are, how they work, their limitations, and how we can use them in the geoinformation domain.

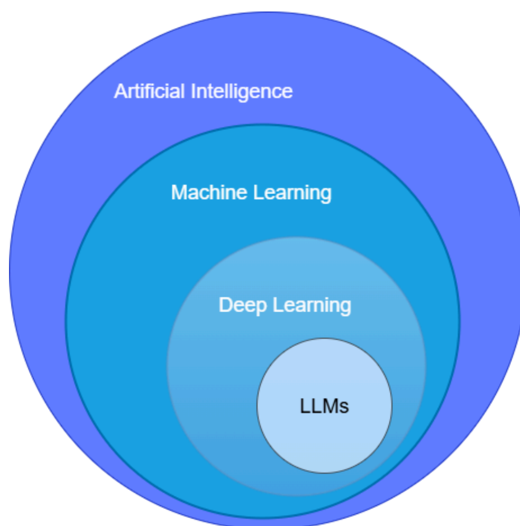
What are LLMS?

Large Language Models (LLMs) are AI systems that are designed to understand, process, and generate natural language. They are trained on huge amounts of text, like Wikipedia, books, online blogs and articles, websites, code repositories etc, which help them “learn” how language works. LLMs are built using the transformer architecture, which we will learn about in the later sections of this tutorial.

Models like GPT-4.1, Mistral or DeepSeek are called “large” because they have tens or hundreds of billions of parameters. Think of parameters as settings that determine how the model responds or behaves.

Where Do LLMs Fit in the AI landscape?

Even though some people use the term Artificial Intelligence (AI) when referring to LLMs, the two words have different meanings. LLMs are just a part of the broader field of Artificial Intelligence. Let us break down how they fit in.



Layers of Artificial Intelligence (AI). Adapted from How Large Language Models Work. Source: https://medium.com/data-science-at-microsoft/how-large-language-models-work-91c362f5b78f

- **Artificial Intelligence**
  Artificial Intelligence is a broad term for the technologies that enable computers and machines to perform tasks that require human intelligence. This includes tasks like learning, problem-solving, decision-making, and creativity. Think of it as a big umbrella that covers everything from self-driving cars, object detection, LLM chatbots etc.

- **Machine Learning**
  Machine Learning (ML) is a subset of AI in which computers learn from data without being explicitly programmed. It involves creating models by training algorithms to make predictions, based on patterns seen in the training data.

- **Deep Learning**
  Deep Learning is a subset of ML that uses artificial neural networks with many layers called “deep” networks. These are layers of algorithms that work a bit like the human brain. The layers allow computers to analyze complex data, like images, audio clips, or even huge amounts of unstructured text, without needing humans to point out what is important.

- **LLMs**
  LLMs are a special kind of deep learning focused on creating and understanding language. With their massive number of parameters, they can handle complex tasks like translating languages and answering questions etc. They are part of generative AI, which means they can create new content, like text or even code, based on what they have learned.

  What sets LLMs apart is the scale. They typically have billions or even trillions of parameters, allowing them to capture deep semantic relationships in language.

  Mistral 7B has **7.3 billion** parameters *[Source].*

  Grok 1 has around **314 billion** parameters *[Source]*.

  DeepSeek-V3 has **671 billion** parameters *[Source]*.

  LlaMA 4 Behemoth has **2 trillion** parameters *[Source]*.

Let us now have a look at some popular LLMs and how they can be categorized.

Open Source LLMs vs Closed-Source LLMs
Broadly, LLMs fall into two categories

a. **Open-Source LLMs**
   Open source models provide access to the code and training data. Anyone can download, inspect, modify, and run the models on their own hardware without restriction. Hosting the models on one's own hardware is important when you want to give the LLM access to private or sensitive documents and information like company secrets, customer records etc. You do not want this information to be sent to external or unknown servers. However, they require significant computational resources for hosting.

   Examples of open-source LLMs include:
   - LLaMA 3 70B
   - Google's Gemma 2 27B
   - Qwen 3
   - Mistral 7B
   - DeepSeek-V3.2
   - OpenAI's gpt-oss-120B.

b. **Closed-Source LLMs**

   Closed-source models are proprietary. The code and the training data are not made public. Users cannot directly view, modify, or self-host the models. They are hosted by

the owning companies and made accessible to the public through APIs or commercial platforms. The APIs have usage restrictions, and some payment is required in order to access the advanced features or increased usage. The privacy of information is also limited, as the user has little control over how and where the data is processed.

Examples of closed-source LLMs include:
- OpenAI's GPT-4o and GPT-5
- Google's Gemini 3.1 Pro
- Grok 4
- Claude 3.5 Sonnet.

The table below summarizes information about different LLMs.

Model	**Developer**	**Access**	**Parameters**	**Context Window (Tokens)**	**Knowledge Cutoff Date**
GPT-5.4	OpenAI	API	Unknown	1,050,000	August 2025
Gemini 3.1 Pro	Google	API	Unknown	1,000,000	January 2025
Claude 4.7 Opus	Anthropic	API	Unknown	1,000,000	January 2026
Mistral Large 3	Mistral AI	Open-source	675B	256,000	October 2025 (presumed)
Llama 4 Maverick	Meta	Open-source	400B	1,000,000	August 2024
DeepSeek R1	DeepSeek	Open-source	671B	128,000	October 2023 (presumed)
Qwen3-235B-A22B	Alibaba	Open-source	235B	128,000	November 2024

The table just contains information about a few popular LLMs [source]. The list is not exhaustive. By the time you are using this tutorial, newer LLMs will have been released. The knowledge cutoff date is the date beyond which the LLM has no knowledge of events or information, and is determined by when the model's training data was last updated. The context window refers to the number of tokens the model can use as input when generating responses.

2.1 How Large Language Models are Created

To understand how LLMs work, we have to first look at how they are developed. There are 3 key stages in the development process of LLMs.

2.1.1 Pre-Training

Pre-training is the first stage of LLM development. This is the phase where the model learns general language patterns, grammar, and facts from the massive training data. The aim is not to teach the model how to answer questions or follow instructions, but to understand the structure of language.

Training Data
The training data consists of large amounts of text collected from various sources such as online books, news articles, online forums, educational resources, and public code repositories. This helps the model to learn about different topics, writing styles, and even tones in written language. Most of the data is unstructured and unlabeled. It does not come with explicit annotations or labels such as “question” and “answer”. Some models are trained with multilingual data, hence giving them multilingual capabilities.

Since the training data is sourced from diverse sources, it is preprocessed to ensure only high-quality data is used. This involves cleaning the data, removing duplicates, filtering harmful content, and removing low-quality data.

Tokens and Tokenization

Before an LLM can understand text, it needs to break it down into smaller pieces called **tokens**. **Tokenization** is the process of splitting text into these tokens. The tokens may represent whole words, subwords, or even individual characters. In general, frequent and short words may be represented by a single token. Long words or those that occur less frequently may be split into multiple tokens.
For example, the sentence “Python is fun!” might be split into tokens like [“Python”, “is”, “fun”, “!”].

In LLMs, tokens are the building blocks of language. These are the words/sub-words that the LLM is trained to recognize and generate. They are the model’s “vocabulary”. Since natural language is diverse (many words, different languages, slang etc), it would be very inefficient to train the LLM to recognize all words that are used in all languages. If new words are made, the model would not recognize them. Instead, by having a fixed set of words, subwords, characters, and digits as tokens, every other word or sequence of text can be created by combining these unique tokens. Since computers process digits and not words, the tokens are assigned unique numerical IDs, called token IDs, which are used for further processing.

Tokenization in action

To visualize how tokenization works, we can use OpenAI's online tokenizer tool using the link: https://platform.openai.com/tokenizer

a. Enter some text in the text input area.
b. Switch between "Text" and "Token IDs" to see how the text is split and the resulting Token IDs
c. Change the text to something different, while still maintaining a few common words.

GPT-4o & GPT-4o mini | GPT-3.5 & GPT-4 | GPT-3 (Legacy)

I love geoinformatics and programming

Clear | Show example

Tokens 8 | Characters 37

I love geoinformatics and programming

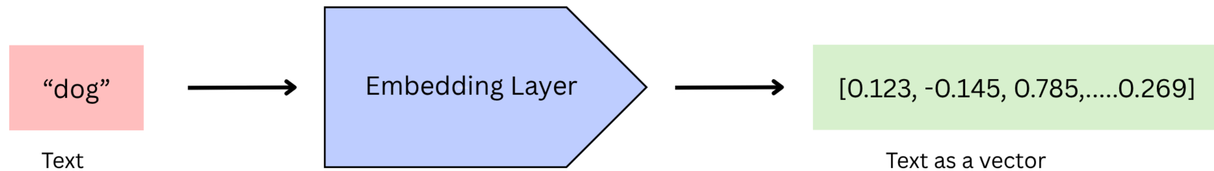
Text | Token IDs

Example of tokenization of text using OpenAI's tokenizer

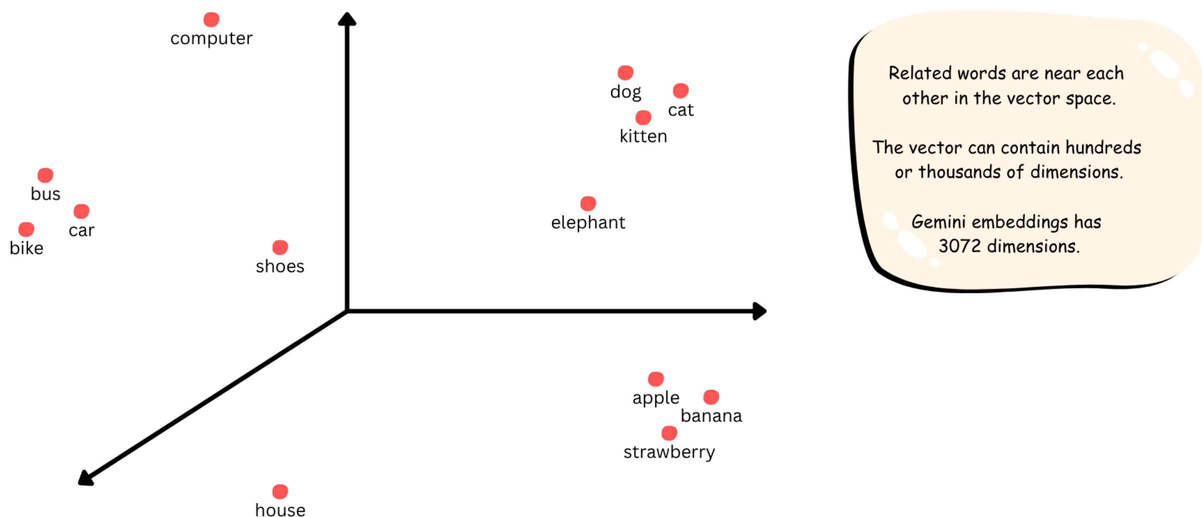
Notice how some common short words are tokenized the same way, and the tokens have the same IDs? Short words like "and", "the", "is", and "or" occur frequently in the training data, so they are stored as single tokens in the vocabulary. Long words are usually composed of multiple subword tokens. Their tokenization can vary depending on the context. For example, a word may be split differently when it appears on its own versus inside a sentence. When tokenizing text, the tokenizer tries to match the longest known tokens from its vocabulary, which is why token boundaries can sometimes change.

Embeddings

After tokenization, each token is represented by its ID. On their own, numbers do not contain any semantic meaning. To help the model understand the relationship between different tokens, each token ID is mapped to a numerical vector representation known as an embedding.



Embeddings represent the semantic properties in a continuous, high-dimension space. Tokens with similar meaning are located closer together in the embedding space. For example, the words ‘dog’ and ‘cat’ will be located closer together, since the two refer to pets, than the words “dog” and “university”. In this way, embeddings provide the model with a numerical representation of meaning.



Embedding is not just for text, it can be applied to images, audio or even graph data. In general, embedding is the process of converting data (of any type) into vectors. Of course, the embedding methods of each modality is different and unique. When talking about “embeddings” in this tutorial, we are referring to text embeddings. The deeper layers of an LLM also produce embeddings, which represent the meaning of sentences or even longer sequences of text, such as paragraphs.

Self-Supervised Language Modeling

Pretraining is performed using self-supervised learning. In this type of ML, the labels are generated from the unlabeled training data. The unsupervised problem is transformed into a supervised problem by auto-generating the labels.

In language modeling, the model is trained to predict tokens based on their surrounding context. Some tokens are hidden, and the model is trained to predict them. Because the answer is already contained in the text, no external data annotation is required. This makes it possible to train LLMs on large and very diverse data.

There are two types of self-supervised learning that can be used: masked language modeling and next-token prediction. In masked language modeling, some tokens in a sequence are hidden, and the model is trained to predict the missing tokens based on the surrounding context.

In next token prediction, the model is trained to predict the next token in a sequence based on the preceding tokens. Text generation is performed one token at a time. Each generated token is added back to the sequence and the updated sequence is used to predict the next token, hence making the process autoregressive in nature. The GPT family of LLMs uses this technique.

Through repeated exposure to large amounts of text and continuous prediction of tokens, the model learns grammar, common language patterns, and even facts. These abilities are not explicitly taught. Instead, they arise from learning to make increasingly accurate predictions of tokens based on the context.

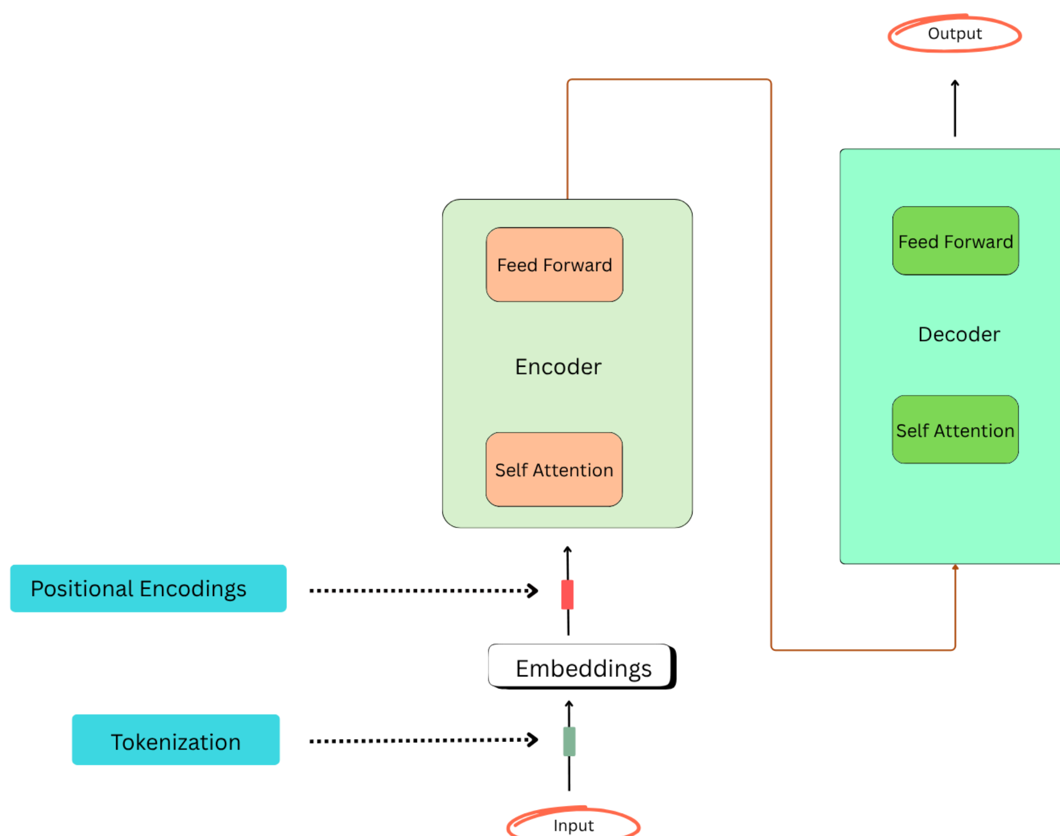
Transformers

The transformer architecture is recognized as the core architecture that brought a breakthrough in the development of LLMs. Earlier Natural Language Processing (NLP) methods, such as neural network architectures, processed text sequentially, that is, one word after the other. This made it difficult to model long-range dependencies between words and limited how efficiently models could be trained on large datasets.

The Transformer overcomes these limitations by processing all tokens in a sequence in parallel, allowing it to scale to very large amounts of text. It has a self-attention mechanism that allows the model to assess the relevance of each token in the input data relative to every other token, rather than being limited to only the neighboring tokens. This enables the model to focus on the important words and contextual relationships between tokens that are far apart in the input sequence.

In the original design, the transformer architecture is composed of two main elements: an encoder and a decoder. The encoder processes the input text and produces representations for all tokens. The decoder then takes this encoded representation as input and generates the output sequence. Both encoder and decoder have multiple layers inside, with each layer transforming the information for the next layer. Before the token embeddings are processed by

the transformer, positional encodings are added to them. This helps the deeper transformer layers to understand the order of the words.



In practice however, modern LLMs use variations of this architecture. Some models such as BERT, only use the encoder. They are trained using masked language modeling. Because encoder-only models process the entire sequence at once, they are particularly suited for tasks that require understanding and analyzing text, e.g text classification, rather than generating new text.

Decoder-only models are trained using next-token prediction. They are effective for text generation, and they are behind most chat-based LLMs. This includes the GPT, LLaMA, and Mistral series of LLMs. Encoder-decoder models such as T5, are suited for tasks where an input sequence needs to be transformed into a different output sequence, such as translation and text summarization. However, decoder-only models have dominated the LLM landscape and can perform a wide range of tasks, even those suited to encoder-only or encoder-decoder models.

2.1.2 Fine-Tuning and Instruction Tuning

At this point, we have a pre-trained LLM, also known as a foundation model. This is a general-purpose model with advanced language understanding and generation capabilities.

However, it cannot follow user instructions, and its output is not aligned with human expectations. It just generates anything based on what it has seen in the training data. You can ask it to summarize some text, and it may generate output that exceeds the length of the input. To make the model useful, it must undergo additional training steps to align its behavior with human expectations. This is where fine-tuning and instruction tuning come in.

Fine-tuning involves taking a pre-trained foundation model and further training it on a specific dataset to improve its performance on a particular task or domain. This helps to customize the LLM's behavior, inject or enhance its knowledge, and improve its performance for specific tasks and domains.

Using supervised fine-tuning, the model is further trained on a smaller dataset containing examples of inputs with the desired outputs. This can include question-answer pairs, task descriptions with correct solutions, or instructions followed by appropriate responses. Unlike pre-training, fine-tuning uses explicitly labeled data. This allows the model to learn how to perform specific tasks, generalize to new tasks, and to respond in a way that is useful to humans. It leverages the language understanding and generation capabilities acquired during pre-training.

> ChatGPT is a fine-tuned LLM derived from foundation models such as GPT-3.5 or GPT-4. It has been fine-tuned as a conversational chatbot.

2.1.3 Reinforcement Learning from Human Feedback (RLHF)

This is the final step of training an LLM. Even after fine-tuning and instruction tuning, a model may still produce outputs that are unhelpful or misaligned with human expectations. RLHF is performed to further refine the model behavior.

RLHF begins by collecting feedback from human evaluators. For a given prompt, the model generates multiple responses. Human reviewers then compare these responses and rank them according to quality, usefulness, correctness etc. It is not about a response being right or wrong, but rather about which response is most helpful.

The human preference rankings are then used to train a reward model, which learns how well a given response matches the expectations of humans. It assigns high scores to helpful responses and lower scores to unhelpful ones. During the RLHF process, the LLM generates responses to prompts and receives feedback in the form of reward scores from the reward model. The objective is to generate outputs that receive the highest reward score. This helps the model to generate responses that align with human expectations and values, while avoiding biased and toxic outputs.

RLHF is a continuous process. You have probably received two responses from ChatGPT for the same prompt and asked to select the response that you prefer. You are basically providing feedback that will train a reward model, which will be used to improve ChatGPT's responses.

2.2 How Large Language Models Generate Text

Once an LLM has been pre-trained, fine-tuned, and aligned, it can generate text in response to a prompt (user input). Prompts generally consist of instructions, questions, input data, and examples.

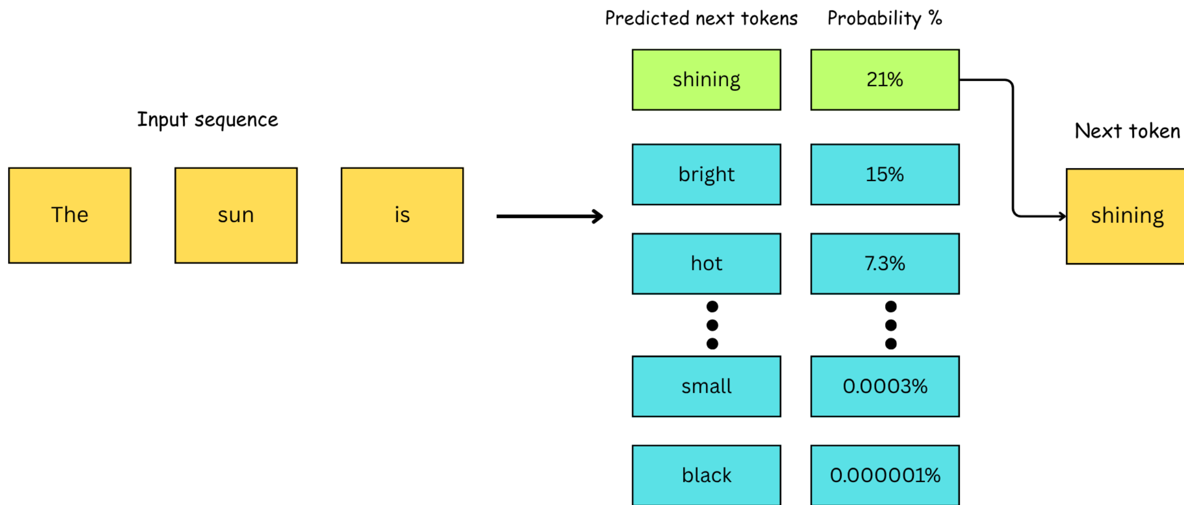
2.2.1 Prompt to Tokens

When a prompt is submitted to a model, it is first tokenized and the tokens converted to embeddings. In chat-based systems like ChatGPT, the system may include previous messages in the conversation, which are combined into a single input sequence. That is how the LLM is able to maintain context and "remember" what a user asked or how it responded in the previous interaction. This information is then passed to the transformer for processing.

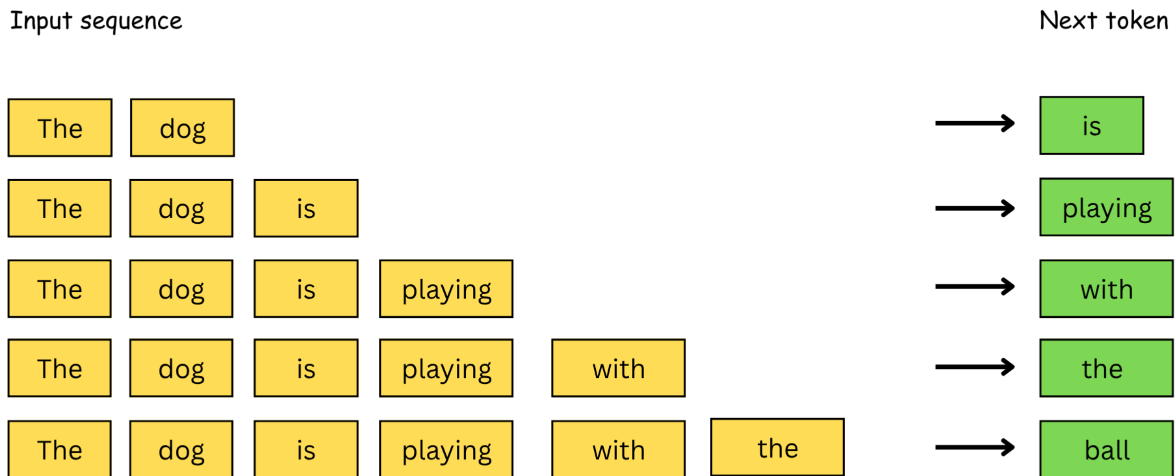
2.2.2 Predicting the Next Token

Text generation in LLMs is based on next-token prediction. Given the input tokens, the model uses the transformer architecture to compute the probability of each possible next token. The probability of the next token is based on the patterns that the model has seen in the training data. The higher the probability, the more likely the token fits the context. It then selects a token (either the most likely one or a random one from the top choices) and adds it to the sequence.

Think of it as a super-smart version of your phone's autocorrect. When you type "The sun is", the model predicts the next likely token, such as "shining" or "bright", based on patterns that it has seen during training.



The process repeats, with the new token becoming part of the input for the next prediction. The transformer's attention mechanism helps the model to focus on relevant parts of the input and previously generated tokens to maintain context.



By learning to predict the next token across diverse texts, LLMs can handle a wide range of tasks without specific programming. This includes coding, where the LLM predicts tokens like "print", "(" , ")", "class", "def" etc to form valid code.

As you can already tell, an LLM does not really have an understanding or intelligence in the human sense. The text generation is based on statistics.

2.2.3 Controlling the Text Generation Process

Have you ever wondered why an LLM does not always output the exact same text even when you give it the exact same input? Let us see why.

Even though predicting the next token is effective, it is not always perfect. Predicting tokens based on the highest probabilities can lead to outputs that are too predictable and repetitive.

To improve generation, models use techniques like:

a. **Temperature settings**
   Temperature is a parameter that controls the randomness of the model's generated text. For most LLMs, the temperature value ranges between 0 and 2.
   - **Temperature = 0**: The model always picks the token with the highest probability. This ensures consistency but can lead to repetitive or robotic text, since it ignores alternatives.
   - **Low temperature (e.g., 0.1 - 0.5):** The model favors high probability tokens. This reduces randomness. This is great for tasks that require precise and factual outputs, like answering questions or generating code.

   - **High Temperature (e.g., 1.1 - 2.0):** This elevates tokens with lower probabilities and gives them a better chance of being selected. The model becomes creative and diverse since it does not always select the most probable tokens. This is useful for creative applications such as generating varied ideas, storytelling, creative writing, etc.

     Most LLM APIs offer an option for setting the temperature value, depending on your use case.

b. **Top-k sampling**
   This techniques limits the model's choices to the top k most probable tokens. The model then samples the next token from only those. This prevents the model from picking very unlikely tokens while still allowing some variety.

   For example, if k=5, the model only considers the 5 tokens with the highest probabilities, and picks one randomly from them.

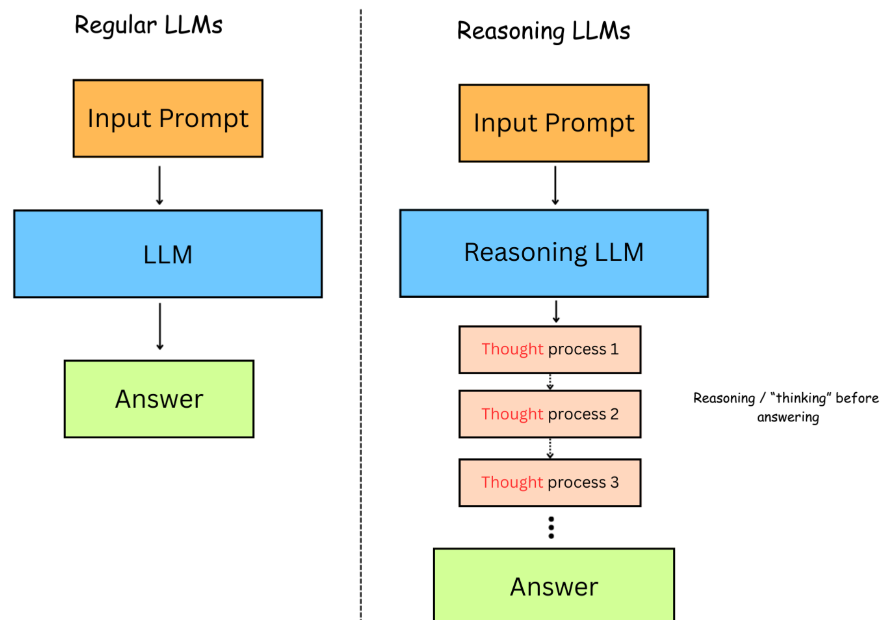
2.3 Reasoning Models

Before reasoning models were introduced, normal LLMs struggled with complex questions or reasoning problems such as advanced math, programming or logical puzzles. Normal LLMs are pattern-matching models, specialized in predicting the next token based on patterns that they have seen. This can be a challenge when the LLMs are presented with a complex query, which they have not "seen" during training. For example, when a normal LLM is asked to solve a multi-step math equation, it may simply guess based on general patterns rather than carefully working through the solution.

Reasoning models like OpenAI's GPT-5, o1, and o3 models, Google's Gemini 2.0 Flash Thinking, xAI's Grok 3, DeepSeek R1, and Anthropic's Claude 4 have shown that LLMs can approach problems more systematically, reason through multiple steps before producing a final answer.

What are reasoning LLMs?

Unlike standard LLMs, reasoning models are designed to "think" before they answer. Instead of producing an output immediately, they generate a series of intermediate reasoning steps known as Chain of Thought (CoT) before arriving at an answer. This process reflects how humans think and solve complex problems by splitting them into smaller and easier subproblems.



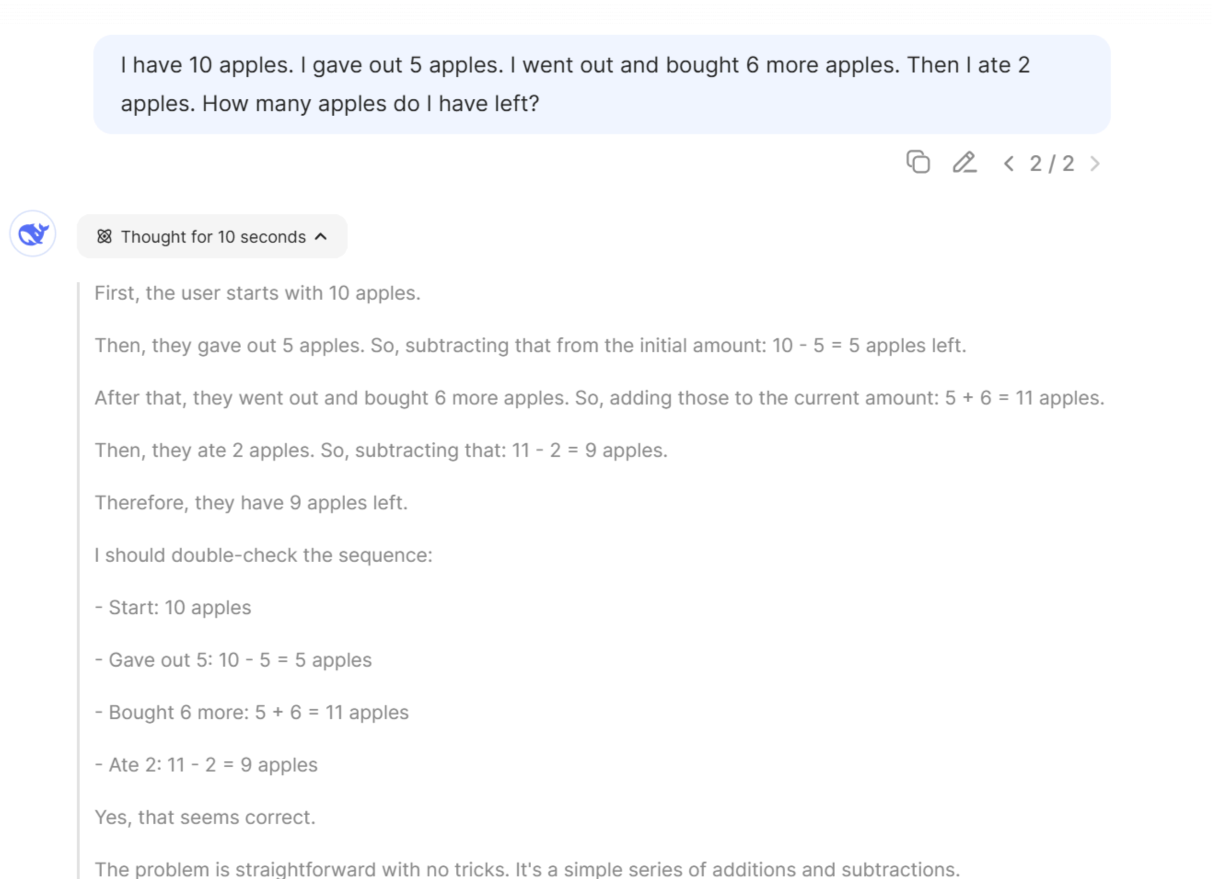
This reasoning process allows the model to:

- Break down problems into smaller, solvable parts.
- Check its work and spot potential errors in the intermediate reasoning steps.

- Explore alternative approaches before finalizing an answer.
- Solve for unfamiliar tasks that cannot be solved by normal next-token prediction or pattern matching.

This process makes reasoning models to be more accurate and reliable for multi-step reasoning tasks. However, the LLM takes longer to produce an answer or response, unlike normal LLMs.

The CoT is generated the same way as any other text sequence, but reasoning models focus more on tokens that lead to step-by-step thinking and breaking down the problem into smaller parts, instead of rushing towards the final answer.



CoT generated by DeepSeek R1 reasoning model

Since reasoning models take time to “think” through a problem, they take slightly more time to generate the final response. They are ideal for solving complex problems or problems that require multi-step planning, such as Math, coding, scientific reasoning, and project planning. For more straightforward and quick responses, non-reasoning LLMs can be used.

2.4 Model Distillation

As LLMs become increasingly powerful, their size also grows. Recent models have hundreds of billions and even trillions of parameters. These big models are expensive to run, require lots of

computing resources (GPUs), and consume a lot of energy. To make them smaller, practical, and accessible, a technique called *model distillation* is used.

Model distillation is the process of transferring knowledge from a large, complex model eg GPT-4o, to a smaller, faster, and more efficient model (eg GPT-4o mini), without losing too much performance capabilities.

The Teacher-Student Paradigm
Model distillation uses the teacher-student framework.

- **Teacher model** - A large, powerful LLM that has been trained on massive computational resources using large amounts of data eg GPT-4o (~ 200 billion parameters).

- **Student model** - A smaller, more lightweight model that learns to approximate the teacher's knowledge and behavior eg GPT-4o mini (~ 8 billion parameters) [*Source*]

During training, the student model's objective is to replicate the teacher's outputs while maintaining a small size and using fewer resources. The teacher model generates predictions, such as text completions, probabilities, or reasoning steps (chain of thought), for a dataset. The student is trained on this dataset, adjusting its parameters to match the teacher's outputs.

By minimizing the difference between the student's outputs and the teacher's outputs, the student model learns the decision patterns of the teacher while using fewer parameters.

The distilled models are smaller in size, run faster due to having fewer parameters, and need less compute.

2.5 Multimodal LLMs

Most LLMs are designed to process text. But in the real world, information doesn't only come in text form. We communicate and observe through images, audio, video etc. This is where multimodal LLMs step in.

Multimodal LLMs are language models that can process more than one type (modality) of data within a single system. They can handle inputs such as images, audio, videos, 3D Models, and sensor data, and generate outputs in one or multiple modalities. For example, Grok 4 and GPT-4o have voice mode, through which users can "talk" to the model instead of typing in the input.

How Different Modalities are Processed
Each type of input is converted into embeddings that the model can understand. The process is different for each modality:

- **Text** - Split into text tokens, as we saw in the earlier sections.

- **Images** - Divided into smaller patches. For instance, a 512 × 512 pixel image may be divided into 14 × 14 smaller patches. Each patch is encoded into a vector.

- **Audio** - Converted into a sequence of audio codes (like breaking a song into smaller chunks)

- **Video** - Treated as a sequence of frames, each frame split into patches, and then processed similarly to images.

Each modality uses a specialized encoder (eg an image encoder for images, audio encoder for audio). These encoders transform the raw input into embeddings, which are then projected into a common high-dimension space. This space allows for the direct comparison and combination of embeddings from different types of modalities. This enables the model to “understand” and combine information from the different modalities. Self-attention mechanisms are also used to help the model focus on the relevant parts of the input data.

Each modality has its own decoder or generator for producing the output.

Some examples of Multimodal LLMs include:

LLM	Modalities
GPT-5	Text, Images, Video
GPT-4o	Text, Images, Audio, Video (input)
Gemini 2.5	Text, Images, Audio, Videos
Grok 4	Text, Images, Audio, Video
Claude 4 Sonnet	Text, Images
Llama 4 Maverick	Text, Images(input)
Mistral Medium 3	Text, Image(input)

Multimodal LLMs opens up more capabilities that go beyond text:

- General tasks such as captioning images and transcribing audio clips.

- In the geoinformatics domain, multimodal LLMs can understand remote sensing images. They have also been used help to classify geospatial imagery, by generating comprehensive, human-readable descriptions from geospatial imagery (Hacheme, G. Q., Tadesse, G. A., Robinson, C., Zaytar, A., Dodhia, R., & Ferres, J. M. L. ,2025).

2.6 Limitations of LLMs

Despite their impressive capabilities, LLMs are not perfect. They have some limitations due to their architecture, training data and the underlying principle in how they work.

a. **Hallucinations**
   This is the most common limitation whereby the LLM produces an output that sounds or looks great, but is inaccurate in reality. LLMs have been shown to fabricate facts and even cite non-existent academic papers. This happens because the model is predicting the "most likely next word" based on patterns in the training data, and not verifying the truth. As such, you should not always trust what the LLM says without doing external fact-checking.

   Recent models like GPT-5 and Claude 4 try to reduce limitations by using reasoning techniques. They are also equipped with tools for searching the internet in real-time hence helping to overcome stale information from their training data.

b. **Context Window Limitations**
   LLMs have a limited context window, which is the amount of text that they can process at once. The size of the context window is measured in tokens. Current models support larger context windows (~ 1,000,000 tokens), but this still limits their ability to handle very long inputs such as books or book chapters, long XML and JSON data etc.

   Exceeding the context window can cause the model to "forget" earlier information and lead to misleading responses.

c. **Domain-specific limitations**
   LLMs excel in general tasks but struggle in specialized domains such as medicine, law, and geoinformatics, where precise and domain-specific knowledge is crucial. This is mostly because LLMs are trained on general data. To make them better at specific domains, finetuning them on that domain-specific data can be done.

d. **Computational costs**
   Training and running LLMs require powerful computational resources eg GPUs, hence making them expensive.

2.7 Extending LLM Capabilities

To overcome the LLM limitations, various techniques are used.

a. **Fine-tuning on domain data.**
   Fine-tuning takes a pretrained or general-purpose LLM and adapts it for a specific task or domain. The pretrained LLM is further trained on a smaller dataset for that domain or task using supervised learning. This makes the LLM more accurate and reliable when

used for that particular domain or task. Only a few model parameters are adjusted, hence making it faster and cheaper than pre-training.

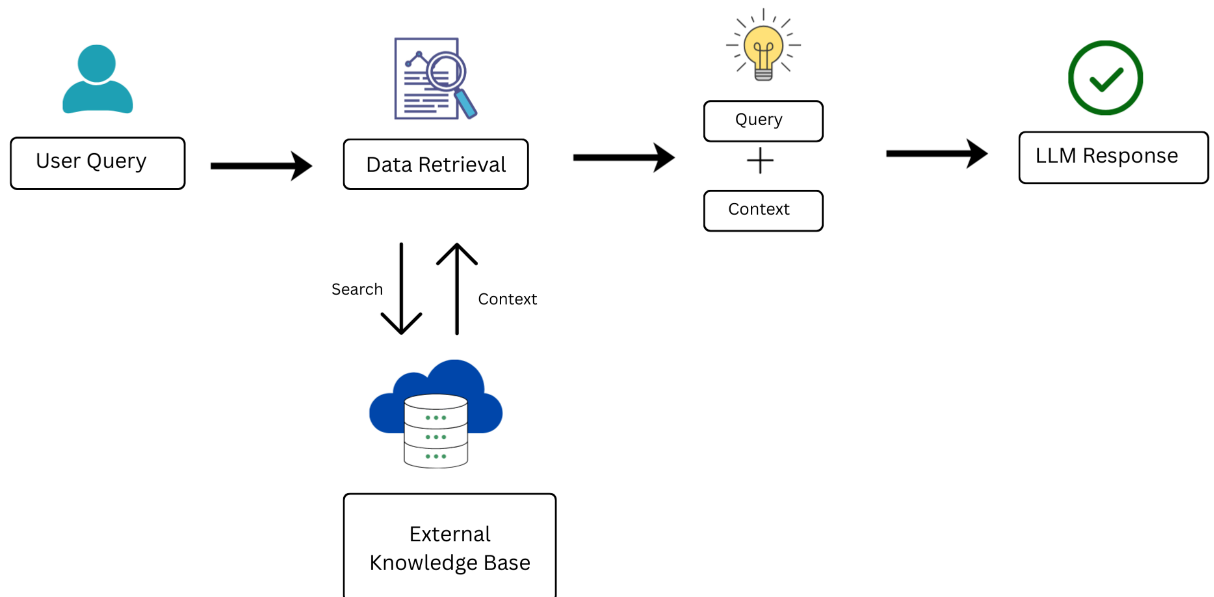
Fine-tuning uses labeled data. The LLM is “shown” various examples of input data (what the LLM will process eg text, images) and the matching labeled outputs (the desired results eg a text summary, an image label).

For example, (Sherman, Zachary, et al. 2025) finetuned OpenAI’s GPT-4o mini to improve the generation of Python code for spatial analysis tasks. 600 plus prompt-response datasets related to geospatial Python scripting were used.

b. Retrieval-Augmented Generation (RAG)

RAG extends LLMs by allowing them to retrieve relevant information from an external knowledge base before generating a response. This helps ground the LLM response in actual information and reduce hallucinations. The knowledge base can also contain private or proprietary data that the model was not trained on.

Unlike fine-tuning which injects new knowledge into the LLM by updating some model weights (parameters), RAG just extracts some relevant information from an external source when needed, and gives it to the LLM to use to generate the answer.



RAG works in 3 main steps:

- **Retrieval** - Chunks of information that are relevant to the user query are retrieved from the external knowledge base. Only a few chunks are retrieved based on the semantic similarity to the user query.

- **Context Augmentation -** The retrieved information is added to the LLM's prompt as context, along with the original query. This provides the LLM with the correct information to generate the answer.

- **Response Generation -** The LLM finally generates the final output using the original query and augmented information.

c. **AI Agents**
An AI agent is an autonomous system that uses an LLM for reasoning and decision making, and is empowered to take actions by connecting to external tools, data sources or APIs. In simple terms, while an LLM alone can reason and generate text, an AI agent enables it to act.

An AI agents uses an LLM to understand the user's request and plan the steps needed to complete it. If need be, the task can be split into smaller tasks. The agent then uses the available external tools to perform the tasks. After a tool has been called, the results are sent back to the agent. The agent then incorporates this feedback and decides the next step to take. This process repeats until the overall task or goal has been achieved.

d. **Real-Time Internet Data Access**
LLMs have a knowledge cutoff. For example, a model might only know about events until 2023. By connecting it to the internet, we give it access to the most recent information. Recent LLMs like GPT-4, GPT-5, Claude 4, and Grok 4 have this feature.

How it works:
- The LLM sends query to an API of a search engine (eg Google, Bing)
- Search results are retrieved and summarized
- The LLM incorporates this fresh information into its response.

2.8 Tools, Frameworks, and Software for Working with LLMs

The LLM ecosystem is rapidly evolving. Every day, new tools, frameworks, and libraries are released that aim to make using LLMs and building LLM-based applications easier. Let us have a look at the most popular ones.

Frameworks for Building with LLMs
Frameworks help developers to connect with external tools, data sources and create custom workflows.

- **Langchain**[1]

[1] https://python.langchain.com/docs/tutorials/

This is an open-source Python framework for building LLM-based applications. It supports chaining (connecting) LLMs with tools, APIs and databases. Multiple LLM prompts and tools can also be chained together to create a custom workflow.

- **LlamaIndex**[2]
  This framework specializes in connecting LLMs with external data (PDFs, databases, documents). It provides powerful indexing (organizing data for efficient retrieval) and retrieval tools for RAG.

- **Ollama**[3]
  This is a lightweight tool for running open-source LLMs locally on your laptop or desktop. It supports LLM models like Llama 3 and Mistral, with easy setup for CPU/GPU. It is useful when privacy is a concern. However, the latency can be high.

- **Hugging Face**[4]
  This is a comprehensive platform and library for downloading, fine-tuning, and deploying models. It also contains multiple datasets that can be used for testing and fine-tuning models.

- **OpenRouter**[5]
  This provides a universal API that lets you access many different LLMs through a single API endpoint. It is useful for experimenting with multiple LLM providers.

Vector Databases
Vector databases are specialized databases that store embeddings and allow fast similarity search. They are essential when building RAG applications.

Common vector databases include Qdrant[6], Pinecone[7], and ChromaDB[8].

Frameworks for Building AI Agents
To turn an LLM into an agent that can act autonomously, these frameworks can be used:

- **LangGraph**[9]
  This is an extension of LangChain designed for building agents with memory. It is highly flexible, and you can create very customized workflows for the agent. Best suited for advanced use cases where customization is key.

---

[2] https://docs.llamaindex.ai/en/stable/
[3] https://ollama.com/
[4] https://huggingface.co/
[5] https://openrouter.ai/
[6] https://qdrant.tech/documentation/overview/
[7] https://www.pinecone.io/product/
[8] https://docs.trychroma.com/docs/overview/getting-started
[9] https://www.langchain.com/langgraph

- **AutoGen (Microsoft)**[10]
  This framework focuses on multi-agent systems where different AI agents can collaborate and communicate with each other. This framework offers a higher level of abstraction than LangGraph, making it easier to prototype multi-agent workflows quickly.

- **CrewAI**[11]
  A framework for orchestrating groups of AI agents like a team, hence the name “crew”. Each agent has a specific role, skillset, and goal. It also offers high levels of abstraction and is easy to get started with.

- **n8n**[12]
  This framework is geared towards low-code or no-code. It provides pre-configured tools and modules that can be dragged and dropped into visual workflows. It is a good choice when you want to build AI agents without writing much code.

- **smolagents**[13]
  This is a lightweight framework for building AI agents with minimal code. This makes it useful for quick prototyping. It has a special type of agent called “Code Agent” that generates and directly executes Python code to accomplish a given task. It is also among the easiest frameworks to learn to use. We will use this framework in the practical section to create and run a simple AI agent.

3. Practical Section

So far, we have learned how LLMs are created and how they work under the hood. Let us now move to the interesting part. We are going to explore how LLMs can support real-world applications in Geoinformatics.

In the first part of this practical, we will use an LLM (such as Mistral AI or ChatGPT) to generate Python scripts for various spatial analysis processes. Instead of relying on the traditional graphical interface in QGIS to select the appropriate tools, we will describe the task in natural language and let the LLM generate the corresponding PyQGIS code. PyQGIS is a feature of QGIS that enables us to execute workflows using Python code. We will use it to run the code that the LLM will generate.

In the second part of this practical, we will learn how to extend the current capabilities of LLMs. We will give an LLM model the ability to act, and not just generate text. We will create an AI agent that can accurately retrieve the coordinates of any location from OpenStreetMap Nominatim API, and use the coordinates to retrieve the weather conditions from Open-Meteo

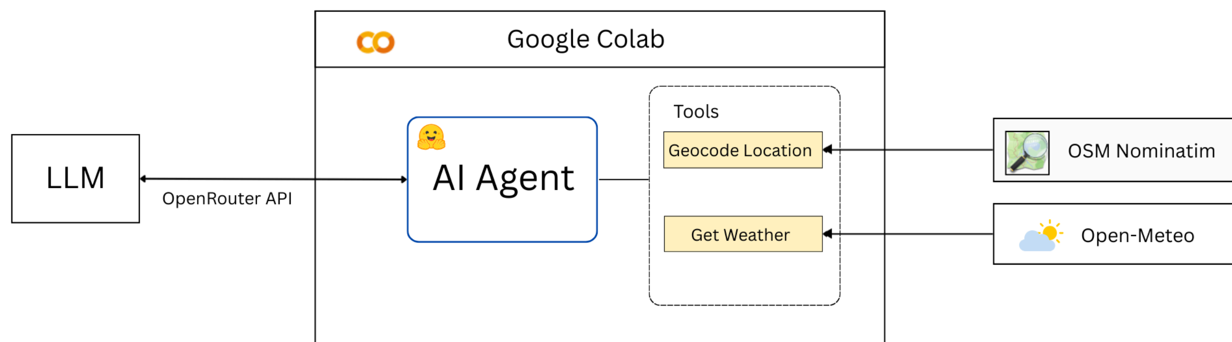
[10] https://www.microsoft.com/en-us/research/project/autogen/
[11] https://github.com/crewAIInc/crewAI
[12] https://n8n.io/
[13] https://huggingface.co/docs/smolagents/en/index

API. The AI agent will use an LLM to generate a human-readable description of the weather conditions.

Required Software and Tools.

1. LLM (eg Mistral AI, ChatGPT account)
2. QGIS software
3. Google Colab
4. Smolagent library
5. OpenRouter API

For the AI Agent, we will use Google Colab as the environment for creating and running the AI agent. Colab is a Jupyter notebook service that is hosted by Google. It requires no setup and can be accessed directly on the browser. One only needs a Google account to use the service. Google Colab offers free access to computing resources. We will use the 'smolagent' library to create the AI agent. To give the agent its 'intelligence" or "brain", we will connect it to an LLM through the OpenRouter API.



NOTE: Other Python notebook environments such as Jupyter Notebook can be used as an alternative to Google Colab. This tutorial uses Google Colab because it is easier and quicker to set up (via a browser).

3.1 Part 1: Using LLMs to perform Spatial Analysis

In this section, we will learn how to use LLMs to perform accessibility analysis. Imagine you work for Münster city's planning department, and have been tasked with identifying buildings that are more than 500m from bus stops. This information will inform the planning process for new bus stops.

A simple manual workflow for this may involve:

a. Loading various layers to QGIS - buildings, bus stops and roads.

b. Checking the projection system of the data and reprojecting it if necessary.
c. Creating a 500m buffer from each bus stop.
d. Clipping buildings within the bus stops' 500m buffer.
e. Subtracting the clipped buildings from the original buildings layer.

Let us now use an LLM to perform this task.

3.1.1 Setting up the environment and data

1. **Install QGIS**
   Visit https://qgis.org/download/ and download and install QGIS on your computer.

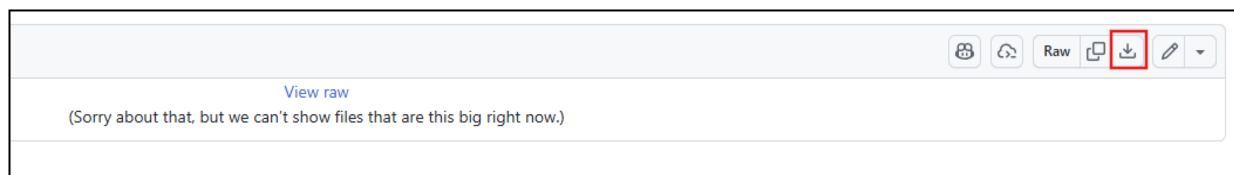
2. **Data Download**
   To run the accessibility analysis on QGIS, we need data for buildings and bus stops . This data has been downloaded from OpenStreetMap (OSM) can be found in the GitHub repository of this tutorial. After downloading the data, unzip the files.

   There are 2 ways of downloading the data:

   a. Make sure you have Git installed on your system. Navigate to the working directory that you want to use for this tutorial. Then open your terminal and run the following commands:

      ```
      git clone https://github.com/oer4sdi/OER-LLMs.git
      ```

      The required data is inside the */data* directory.
   b. Navigate to the data folder in the GitHub repository. For each file, click the zip file, then click the download button.



3.1.2 Performing Accessibility analysis using an LLM

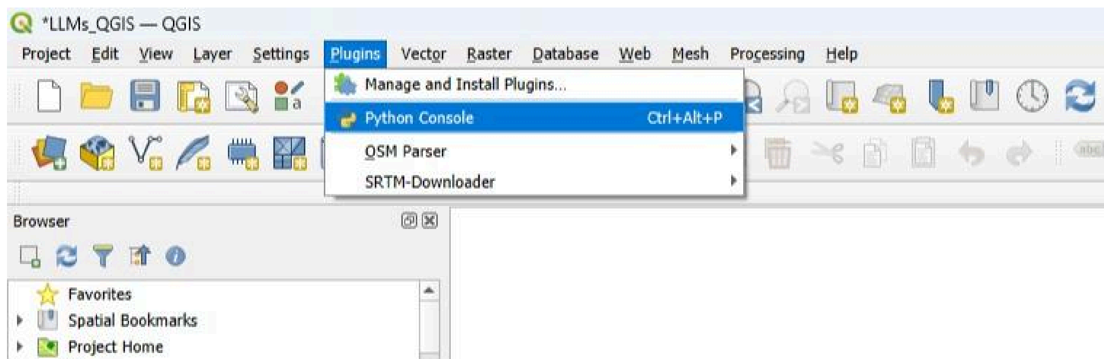
LLMs are powerful and can achieve a lot, but the quality of the results depends heavily on the information you give them, and the clarity of your prompt. Depending on the complexity of the task, a good prompt can include the task or question, along with helpful details like context, input data, or sample responses.

NOTE: LLMs do not always generate the same output, even when given the exact same prompt. This is because their text generation process involves some randomness. The next token predicted may be selected from a range of possible tokens, based on their probabilities. This sampling process introduces some variability in the LLM's output. So it is expected that running the prompts that are provided in this section will produce a slightly different or totally different output for you. For this reason, the PyQGIS script that the LLM will generate for you may not work at first. If that happens, adding more context, clarifying the instructions, or including the error messages in a follow-up prompt should help the model to correct the script.

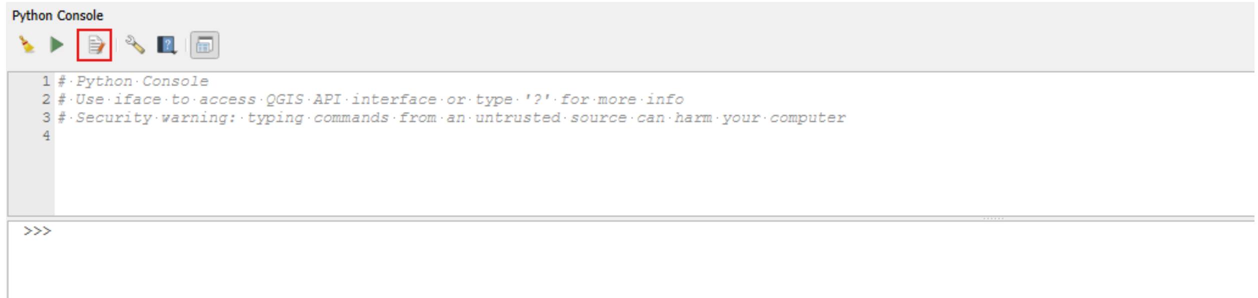
The prompts, LLM outputs, and the results of PyQGIS execution in this section are meant to be a guide of how a typical accessibility analysis with an LLM might look like. You can reuse the same prompts (with your own file paths to the data), provide more context or specific instruction, or even prompt the LLM to follow a different or alternative GIS workflow in order to achieve the intended results.

1. **Sign up or Log in to an LLM model**
   The example in this section uses ChatGPT. Sign up or login to an LLM model of your choice. Larger and recent models eg Mistral, Grok, Claude, Gemini, and DeepSeek are preferred as they are powerful and more likely to generate the correct script.

2. **Create a PyQGIS script**
   Open QGIS and create a new project. Navigate to *Plugins* > *Python* Console to activate the PyQGIS tool.



   A small tab will open at the bottom of the QGIS interface. Click the "Show editor" button to open a blank script

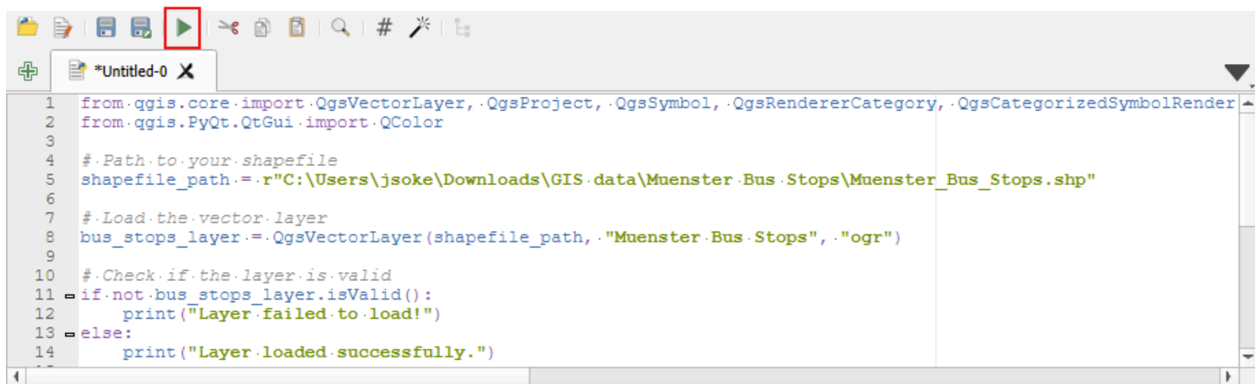


A script opens on the right side of this tab. We will paste and execute the code that the LLM will generate on this script.

3. **Load the bus stops layer**
   We navigate to the chatbox of the LLM and instruct it to generate a PyQGIS script to load the bus stops layer. Remember to adjust the file path to match the location of your data.

> I am doing an accessibility analysis on QGIS. The goal is to identify buildings that are more than 500m away from a bus stop.
>
> Write a PyQGIS script to load a vector layer from a shapefile located at 'C:\Users\jsoke\Downloads\GIS data\Muenster Bus Stops\Muenster_Bus_Stops.shp' into QGIS and style the layer with a red color

Copy-paste the generated code into the PyQGIS Script. Click the "Run" button to execute the script. The bus stops layer is loaded into QGIS, with the specified styling.



4. **Load the buildings layer**
   Next, we prompt the LLM and execute the PyQGIS code for loading the buildings layer.

> Write me another script for loading the layer at: "C:\Users\jsoke\Downloads\GIS

data\Muenster Buildings\Muenster_Buildings_poly.shp"

5. **Attempt 1: Generating one Script for performing the analysis**
   Even though the analysis can be performed in multiple steps, the LLM attempts to generate a single PyQGIS script that performs the entire workflow.

   How can I identify buildings that are more than 500m from a bus stop? Provide the PyQGIS code for that analysis

   The LLM correctly captures the workflow idea as shown below.

   ✔ **What the script does**

   1. **Loads both layers** (bus stops + buildings).
   2. **Creates a 500 m dissolved buffer** around all bus stops.
   3. Uses **Select by Location** to find buildings that *intersect* that buffer.
   4. **Inverts the selection** → the selected buildings are **more than 500 m** from any bus stop.

   You'll see the selected buildings highlighted in QGIS.

   The generated PyQGIS code may work for you on the first try. If it does not, it may be due to various reasons.

   Compressing the logic of the whole workflow into one large script is likely to run into errors. Among the issues in the generated code is using the wrong projection. The layers are in WGS 84 (a geographic coordinate system), while the script attempts to create a 500m buffer around the bus stops. Without reprojecting the data into a local projection system that uses meters, the 500 m buffer is interpreted to be 500°.

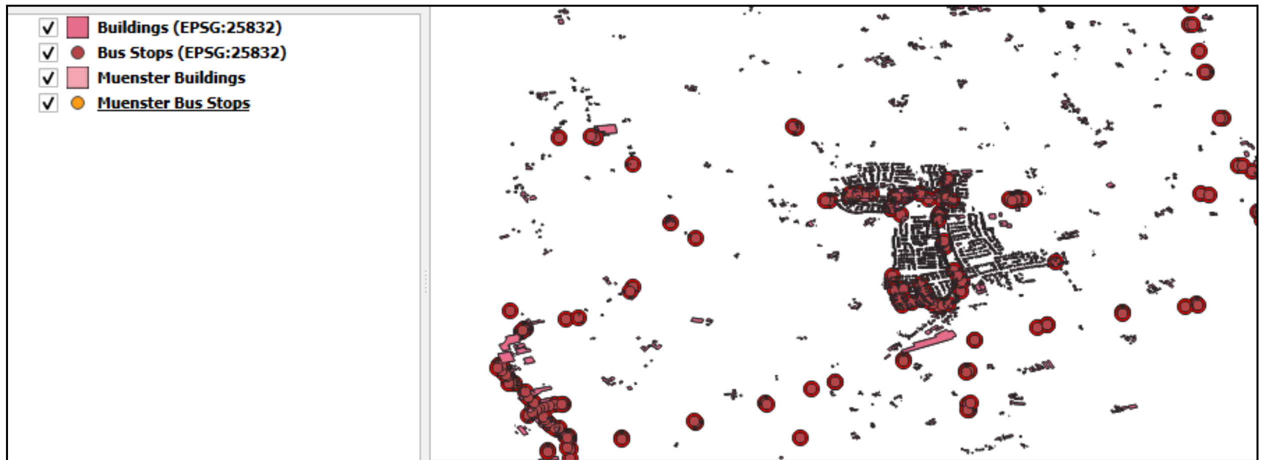
   Steps 6, 7, and 8 assume you have encountered the issues mentioned. If not, you can tailor the prompts to fit the problems you encountered.

6. **Attempt 2: Reprojecting the layers and splitting the workflow into smaller steps**
   To fix the issues, we prompt the LLM to reproject the layers and to separate the workflow logic into separate scripts.

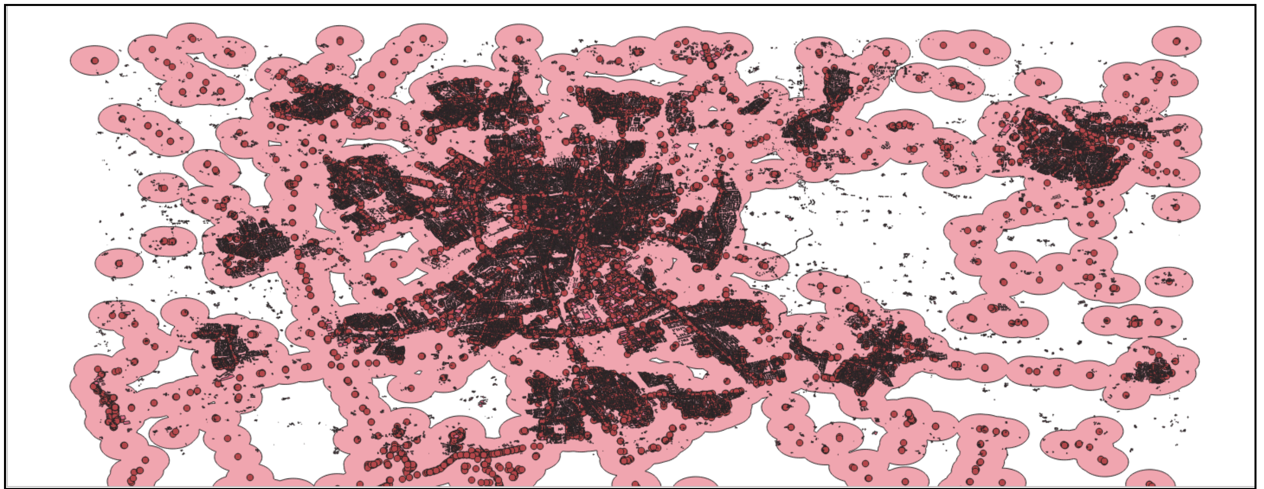
   The script doesn't work. The coordinate reference system of both layers is: EPSG:4326 - WGS 84. Generate a script for reprojecting both layers. Then separate scripts for performing the other steps

The LLM uses its internal knowledge to determine the correct projected coordinate reference system for Münster and generates a script for performing the reprojection.

Copy the PyQGIS script for the reprojection and execute it. The new reprojected layers will be loaded into QGIS.



7. **Create a 500m buffer around the bus stops**
   The next script creates a buffer of 500m around each bus stop. Where multiple buffers overlap, they are dissolved into one continuous polygon.



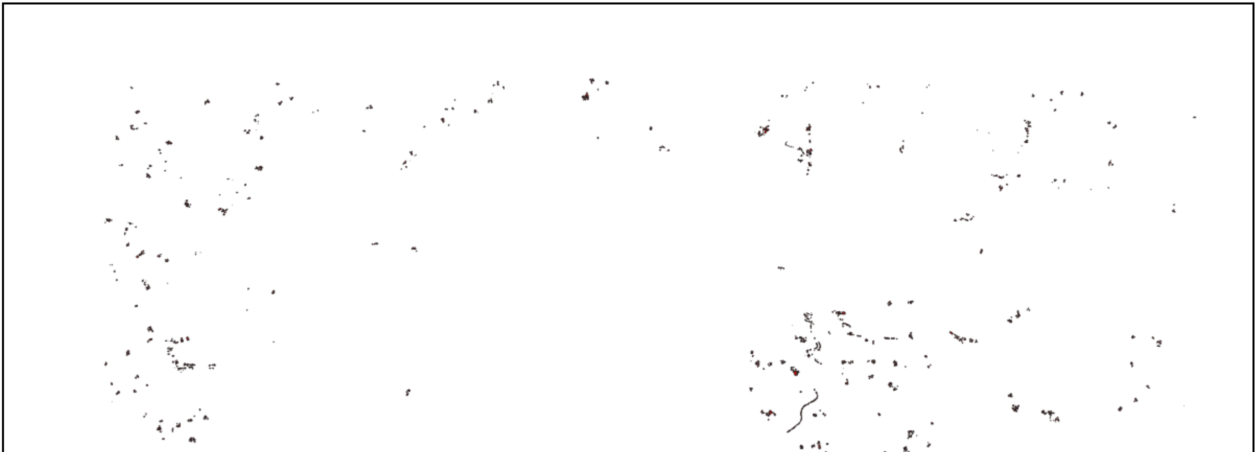
If the previous step did not produce a separate script for generating the buffer, you could prompt the LLM for the PyQGIS script.

8. **Identify buildings that are outside the 500m buffer**
   The first script generated for this step encountered errors. The script tried to calculate the distance of each individual building to the nearest bus stops, and then determine buildings that are more than 500m away from any bus stop.

We can prompt the LLM again to generate a different script that uses a simpler approach, just as it had suggested earlier.

> The third script for identifying buildings that are more than 500m from the nearest bus stop did not work. Generate a script that selects buildings that are outside the 500m buffer layer.

Since we are more specific, the new script works as expected, and we have a new layer that only has buildings that don't intersect the 500m buffer.



9. **Determine the number of inaccessible buildings**
   We can even go further and determine the number of buildings that are located far from the bus stops.

> Generate a script that calculates the number of buildings that are outside the 500m buffer. The layer with those buildings is named 'Buildings_more_than_500m'

The generated script simply counts the features on that layer. The result is printed on the Python console.

```
20 >>> exec(Path('C:/Users/jsoke/AppData/Local/Temp/tmpa91xmpcy').read_text())
21 Number of buildings outside the 500m buffer: 2688
```

The steps we have gone through show how to prompt LLMs when you have an idea of what the spatial analysis workflow. In fact, you could prompt the LLM to generate an overview of the workplan, without the code. If all steps are feasible, you could then prompt it to generate the code for each step. Reasoning LLMs perform better in such tasks, since they can break down a problem into a series of smaller steps that need to be executed to achieve the desired result.

Later on, you can further experiment on other spatial analysis workflows, using the provided data or different data.

As you can see or have already experienced, the LLM may not generate the correct script at first. The process involves clarifying the requirements and including the encountered errors in the prompts. Since the LLM knows a couple of QGIS tools and methods, it can generate a script that uses any of them. Some tools may not be very efficient with our data. The process is also not reproducible.

LLMs are useful for quick prototyping, but they have no proper understanding or human intelligence. This means that you, as the user, are responsible for the results of any data analysis you perform, not the LLM. The solutions or code they generate must be understood by the user and, if need be, critiqued and refined. The LLM may also generate code that runs without errors, but the results are methodologically/content-wise incorrect, which can mislead decision-making.

Generating PyQGIS workflows is a good use case for fine-tuning an LLM. Fine-tuning an LLM with a dataset that contains various GIS tasks or workflows, the data requirements, the QGIS tools to be used etc., could make the LLM more reliable for workflows that it has been fine-tuned on. A QGIS plugin can even be created that provides an interface for natural-language input while automatically executing the necessary commands in the background, such as loading layers, manipulating them, and visualizing the results. However, that is beyond the scope of this tutorial.

3.2 Part 2: Creating an AI Agent

Let us now create our AI Agent for weather information. The Agent will be powered by an LLM, which we will access using the OpenRouter API. We will need an OpenRouter API key.

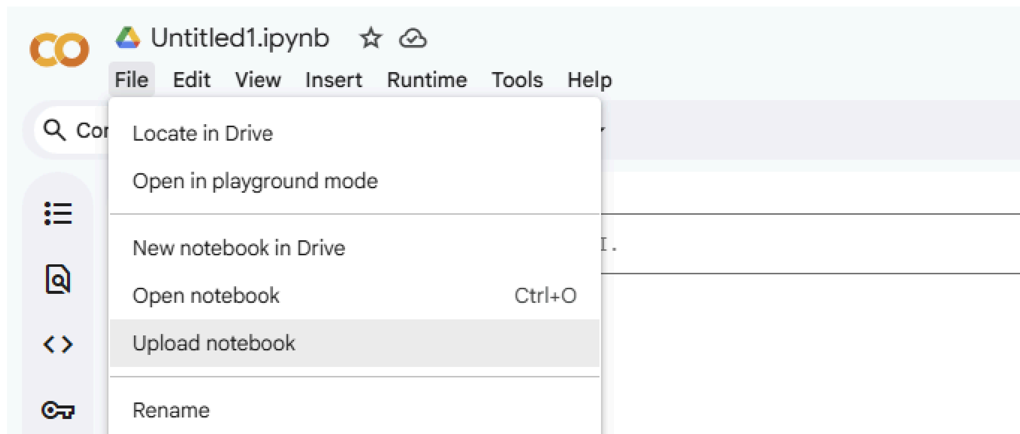
3.2.1 Setting up the Google Colab environment

1. **Download the notebook with the steps to be followed**
   The notebook required for this section is among the project files that were cloned from GitHub. If you have not cloned the Github repository of this tutorial, you will need to download the notebook directly from GitHub.

2. **Sign up or Login to Google Colab**
   Visit https://colab.google/. Click the "*New Notebook*" button. You will be prompted to sign in using your Google (gmail) account. You will then be redirected to an empty Colab notebook.

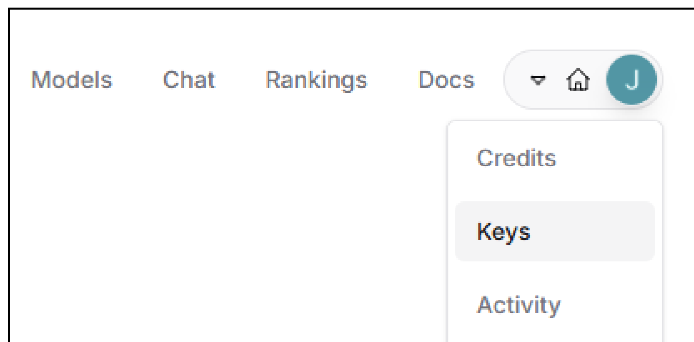
3. **Upload the Notebook to Google Colab**

Click `File` > `Upload Notebook`. Navigate to the location of the downloaded notebook, and select it for uploading to Colab.

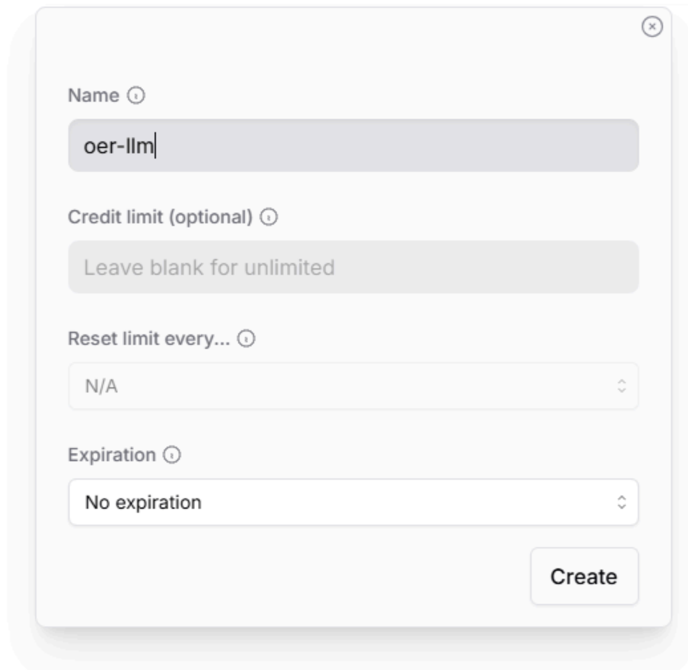


3.2.2 Generating the OpenRouter API Key

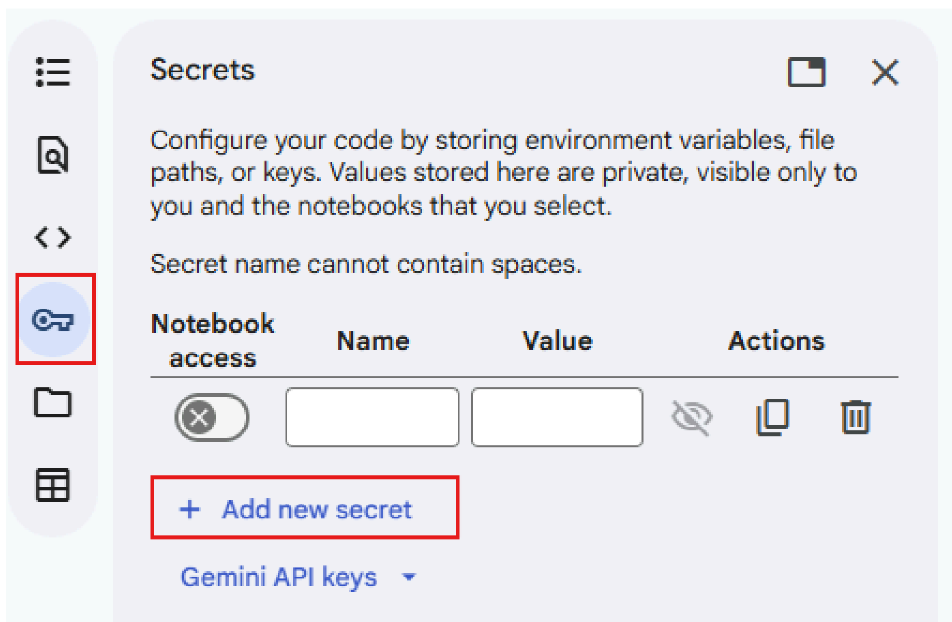
1. Open https://openrouter.ai/ and click the Sign Up button.
2. Choose a preferred method to sign up - Google account, GitHub account or a normal email and password combination.
3. After signing up, click the user avatar in the upper-right corner. Then click the '`Keys`' button.



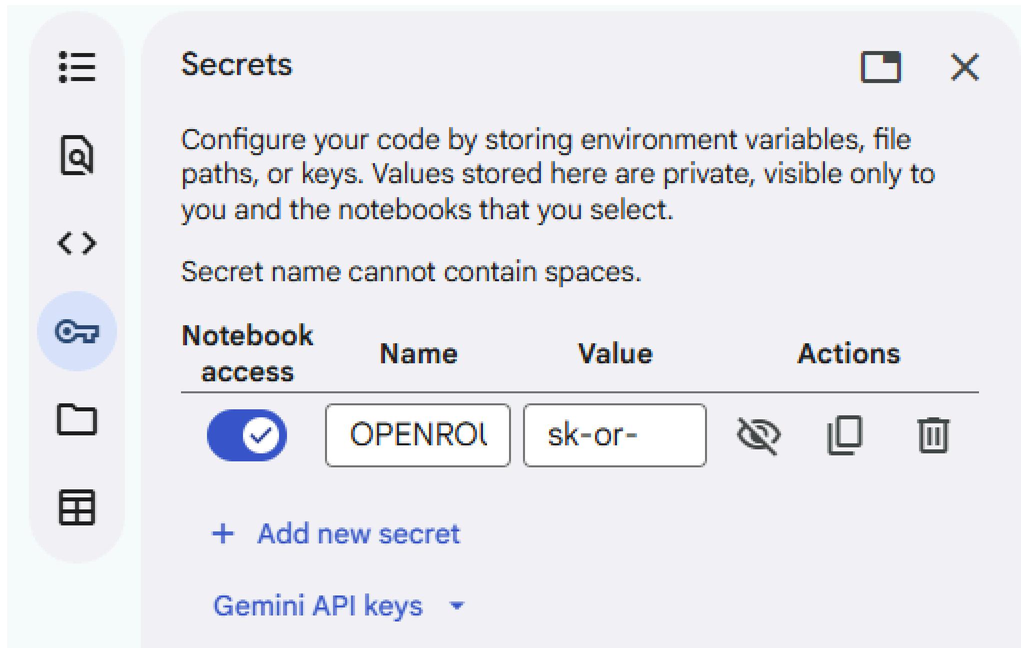
4. Click the "`Create API Key`" button. Give the API an appropriate name and click the "`Create`" button.



5. Copy the generated API key.
6. Go back to the Google Colab notebook. On the left tab, select the "`Secrets`" tab, then click "`Add new secret`".



7. Input "OPENROUTER_API_KEY" as the name of the api key, and paste the API key that you generated on OpenRouter to the value section. Also, check the "`Notebook access`" option to allow the notebook to use the API key.



8. You can now use the Python Notebook to generate and test the tools and the LLM agent step by step. Afterwards, come back to the Conclusion section where we'll reflect on what we have learned in the tutorial and get some tips on how to dive deeper into the topic.

4. Discussion and Conclusion

In this tutorial, you have learned what LLMs are, how they are created, and how they generate text. You have also seen the limitations of LLMs and techniques to overcome some of them. LLMs are more powerful when they are given the ability to act through AI agents, augmented with up-to-date external information via Retrieval-Augmented Generation (RAG), or fine-tuned for specific tasks. In the practical section, you used an LLM to generate PyQGIS code, which you executed on QGIS. You then went further by giving an LLM the ability to act using tools, by developing an AI agent that can geocode place names, fetch real-time weather information, and summarize it for the specified place. The AI Agent was simple, but it shows how LLMs can be integrated with external systems and take action beyond text generation.

LLMs are increasingly being adopted and integrated into applications across various domains. In geoinformatics, LLMs have been used to transform natural-language queries into SQL and SPARQL queries. Multi-modal LLMs have been used to generate descriptions of satellite

images. The list of possibilities is endless. Think of any task that requires some technical expertise to handle (formulating technical queries, finding information, creating and executing scripts, calling APIs etc), there is a good chance that LLMs can be used to simplify or even automate the process. At the same time, it is important to approach the LLM outputs critically and not trust them blindly. As a user or developer, you remain responsible for verifying the correctness and reliability of the outputs.

The field of LLMs is evolving rapidly. New models, tools, and frameworks for building LLM-based applications are released frequently. However, rather than focusing solely on specific tools, it is advisable to first develop a solid understanding of foundational concepts (software design principles, AI Agents, RAG). These principles remain valuable even as tools and technologies continue to evolve.

5. References

1. Medium: How Large Language Models work. https://medium.com/data-science-at-microsoft/how-large-language-models-work-91c362f5b78f

2. IBM: What is artificial intelligence (AI) https://www.ibm.com/think/topics/artificial-intelligence

3. Kulkarni, C. (2023). The evolution of large language models in natural language understanding. J. Artif. Intell. Mach. Learn. Data Sci, 1, 49-53.

4. Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., & Gao, J. (2024). Large language models: A survey. arXiv preprint arXiv:2402.06196.

5. HatchWorksAI: Open-Source LLMs vs Closed: Unbiased Guide for Innovative Companies [2025] https://hatchworks.com/blog/gen-ai/open-source-vs-closed-llms-guide/

6. ExplodingTopics: Best 44 Large Language Models (LLMs) in 2025 https://explodingtopics.com/blog/list-of-llms

7. Substack: Tokenization in large language models, explained https://seantrott.substack.com/p/tokenization-in-large-language-models

8. Multilingual Token Visualizer with different open source LLMs https://huggingface.co/spaces/aiqtech/LLM-Token-Visual

9. HuggingFace: LLM Embeddings Explained: A Visual and Intuitive Guide

https://huggingface.co/spaces/hesamation/primer-llm-embedding?section=what_are_embeddings?

10. Medium: Transformers Architecture Simplified
https://medium.com/@theaveragegal/transformer-architecture-simplified-3fb501d461c8

11. Datacamp: How Transformers Work: A Detailed Exploration of Transformer Architecture
https://www.datacamp.com/tutorial/how-transformers-work

12. Vellum: LLM Temperature: How It Works and When You Should Use It
https://www.vellum.ai/llm-parameters/temperature#:~:text=Adjusting%20the%20temperature%20changes%20how,produce%20more%20random%2C%20creative%20responses
.

13. Substack: Maarten Grootendorst: A Visual Guide to Reasoning Models
https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-reasoning-llms

14. Datacamp: LLM Distillation Explained: Applications, Implementation & More
https://www.datacamp.com/blog/distillation-llm

15. Hacheme, G. Q., Tadesse, G. A., Robinson, C., Zaytar, A., Dodhia, R., & Ferres, J. M. L. (2025). GeoVision Labeler: Zero-Shot Geospatial Classification with Vision and Language Models. arXiv preprint arXiv:2505.24340.

16. Nvidia: Multimodal Large Language Models
https://www.nvidia.com/en-us/glossary/multimodal-large-language-models/

17. IBM: What is a multimodal LLM (MLLM)?
https://www.ibm.com/think/topics/multimodal-llm

18. ML6: How LLMs access real-time data from the web
https://www.ml6.eu/blogpost/how-llms-access-real-time-data-from-the-web

19. Label Your Data: Supervised Fine Tuning. Enhancing Your LLM Accuracy
https://labelyourdata.com/articles/llm-fine-tuning/supervised-fine-tuning

20. Sherman, Zachary, et al. "Generative AI for Geospatial Analysis: Fine-Tuning ChatGPT to Convert Natural Language into Python-Based Geospatial Computations." ISPRS International Journal of Geo-Information 14.8 (2025): 314.

21. Gupta, Shailja, Rajesh Ranjan, and Surya Narayan Singh. "A comprehensive survey of retrieval-augmented generation (rag): Evolution, current landscape and future directions." arXiv preprint arXiv:2410.12837 (2024).