Introduction to Generative AI and LLMs

1. Learning objectives

- Students should be able to explain the difference between AI, ML, Deep Learning, and Generative AI in simple terms with examples.
- Students should understand at a high level what Large Language Models are, why the Transformer architecture was a breakthrough, and where such models are used in the real world.
- Students should also be comfortable setting up a Python 3.10+ virtual environment that will be reused for the rest of the course.
- Students should understand what an API is, how HTTP request–response cycles work, and how JSON is used to send prompts and receive LLM outputs.
- Students should be able to obtain free-tier API keys (Google AI Studio, Groq), store them safely via environment variables, and call at least two LLM APIs from Python.
- Students should also be able to compare basic latency and output quality across providers using simple CLI scripts.

---

2. The AI landscape: AI, ML, DL, GenAI

- Artificial Intelligence is the broad field focused on building systems that can perform tasks requiring human-like intelligence such as reasoning, learning, decision-making, perception, and language understanding.
- Machine Learning is a subfield of AI where systems learn patterns from data so they can make predictions or decisions without being explicitly programmed for every rule.
- Deep Learning is a subset of ML that uses multi-layer neural networks to model very complex patterns in data such as images, audio, and natural language.
- Generative AI is a subset of Deep Learning focused on models that can create new content such as text, images, code, music, and video rather than only classifying or scoring existing data.

---

3. Traditional (discriminative) vs generative AI

- Traditional or discriminative models learn to distinguish between classes or predict labels given input data, for example spam vs not-spam or fraudulent vs legitimate transactions.
- These models output a limited set of labels or numeric scores based on learned decision boundaries in the training data.
- Generative models instead learn the underlying probability distribution of the data so they can sample or generate new examples that look similar to what they were trained on, such as new sentences, images, or code snippets.

- A simple way to remember the difference is that discriminative models answer “Is this X or Y?” while generative models answer “Create something that looks like X.”

---

4. A brief history of generative models

- Early generative text models such as n-gram language models and Markov chains could produce sequences of words by predicting the next token from the previous few, but their outputs were often short and incoherent.
- In 2014, Generative Adversarial Networks introduced a game between a generator network that tries to create realistic samples and a discriminator network that tries to detect fakes, which led to huge progress in realistic image generation.
- In 2017, the “Attention Is All You Need” paper introduced the Transformer architecture and showed that self-attention could replace recurrent networks for sequence modeling, enabling much better handling of long-range dependencies.
- From 2018 onwards, scaling Transformers to billions of parameters and training them on internet-scale text led to modern Large Language Models such as GPT, Gemini, Llama, and Phi-3.

---

5. What is a Large Language Model (LLM)?

- A Large Language Model is a neural network trained to predict the next token in a sequence of text given all previous tokens, which surprisingly gives it broad abilities in generation, reasoning, and language understanding.
- The “large” aspect refers to both the number of parameters which can range from millions to tens of billions and the massive training datasets that often contain a significant portion of the public internet.
- Once trained, a single LLM can perform many tasks such as question answering, translation, summarization, coding assistance, and more without separate task-specific models.

---

6. Capabilities and limitations of LLMs

- LLMs can generate fluent, context-aware text, follow complex instructions, adapt to examples in the prompt few-shot learning, and act as general assistants for research and coding.
- However they have limitations such as hallucinations producing confident but incorrect statements, sensitivity to prompt wording, lack of true up-to-date world knowledge without external tools, and potential biases inherited from training data.

- Because LLMs do not “understand” in a human sense and are trained to predict tokens, careful prompt design and external verification are essential for serious applications.

---

7. The Transformer architecture – high-level view

- Transformers process sequences by first converting tokens into numerical vectors called embeddings that capture semantic information about words or subwords.
- Because a pure feed-forward network does not know the order of tokens, positional encodings are added to embeddings so that the model can distinguish “dog bites man” from “man bites dog.”
- The key innovation is self-attention, where each token looks at all other tokens in the sequence and learns how strongly to attend to them when computing its new representation.
- In practice this means the model can capture long-range dependencies such as resolving pronouns and understanding that “bank” may mean a river bank or a financial institution based on context.
- Many modern LLMs use decoder-only Transformers which repeatedly apply self-attention and feed-forward layers to generate one token at a time conditioned on everything generated so far.

---

8. Real-world applications of Generative AI

- Customer support chatbots built on LLMs can answer FAQs, route users, and draft responses, often using Retrieval-Augmented Generation so that answers refer to internal documentation.
- In content creation, LLMs assist with drafting emails, articles, social media posts, marketing copy, and technical documentation, accelerating human workflows rather than fully replacing them.
- In software development, code-focused LLMs help with scaffolding functions, explaining errors, writing tests, and refactoring, though humans still need to review correctness and security.
- Education, healthcare documentation, legal drafting, and creative arts such as story and script generation are additional domains where generative models are already deployed.

---

9. Python 3.10+ environment setup (high-level)

- Each student will create a dedicated course folder and a virtual environment so that all required packages are installed in an isolated space without affecting other projects.
- The typical steps are to verify that Python 3.10+ is installed, create a folder such as `GenAI-Training`, run `python -m venv venv` inside it, and then activate the environment using the platform-specific command.
- Once the environment is active, students will install commonly used libraries for this course such as `notebook`, `requests`, `python-dotenv`, `langchain`, `torch`, `sentence-transformers`, `faiss-cpu`, `selenium`, `streamlit`, and `gradio`.

---

10. What is an API?

- An API (Application Programming Interface) is a well-defined contract that lets one piece of software talk to another by sending requests and receiving responses.
- The restaurant analogy helps:
  - your Python script is the customer,
  - the API is the waiter carrying requests and responses,
  - and the provider's servers are the kitchen preparing results.
- In this context, the LLM API is the interface that accepts prompts and returns generated text in a machine-readable format such as JSON.

---

11. Why use APIs for LLMs?

- State-of-the-art LLMs like Gemini, GPT are extremely large and require specialized hardware, which most dev machines cannot provide; APIs allow access to these models without owning GPUs or clusters.
- Providers handle scaling, security, uptime, and model updates, so developers focus on building features instead of managing infrastructure.
- APIs also give a consistent integration surface: the same HTTP + JSON pattern works for chatbots, back-end services, mobile apps, and CLI tools.

---

12. How LLM APIs work: request–response

- Most LLM APIs are simple HTTP endpoints that accept POST requests containing JSON, and respond with JSON that includes the model's output.
- Key elements are the endpoint URL, headers (especially `Authorization` and `Content-Type`), and a JSON body specifying the model and the prompt.
- The response structure typically contains metadata and at least one choice whose content is the generated text to display to the user.

---

13. Core concepts: JSON, endpoints, API keys, rate limits

- JSON (JavaScript Object Notation) is a lightweight format based on key–value pairs and arrays, which maps directly to Python dictionaries and lists.
- An endpoint is the specific URL for the operation, for example Gemini's `...:generateContent` endpoint or Groq's `/chat/completions` endpoint.
- API keys are secret tokens sent with each request (usually in the `Authorization` header or query string) so the server can authenticate and attribute usage to an account.
- Rate limits constrain how many requests can be sent per minute or per day on free tiers, protecting providers from abuse and ensuring fair resource sharing.

---

14. Overview of free-tier LLM API providers

- Google AI Studio exposes Gemini models like `gemini-1.5-flash` and `gemini-2.0-flash` with generous free quotas and clear documentation plus a web playground.
- Groq provides extremely fast hosting of popular open-source models such as Llama 3 variants and Gemma, powered by custom LPU hardware.
- OpenRouter acts as an aggregator that exposes many providers through a common API format, making model switching easier for experimentation.

---

15. Raw API call with curl

Showing a `curl` command helps students see the underlying HTTP structure independent of Python libraries.

Example `curl` call to Groq (template without real key):

```
curl.exe -X POST https://api.groq.com/openai/v1/chat/completions `
    -H "Authorization: Bearer $Env:GROQ_API_KEY" `
    -H "Content-Type: application/json" `
    -d '{ ""model"" : ""llama-3.3-70b-versatile"", ""messages"" : [ { ""role"" : ""user"", ""content"" : ""Who is president
of India?"" } ] }'
```

---

16. Python demo: calling Gemini API with requests

This script asks the user for a prompt, sends it to Gemini's `generateContent` endpoint, and prints the response text.

```
import os
import json
import requests
from dotenv import load_dotenv

load_dotenv()

api_key = os.getenv("GOOGLE_API_KEY")
if not api_key:
    raise ValueError("Google API Key not found. Set GOOGLE_API_KEY in .env")

url = f"https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-flash:generateContent?key={api_key}"

headers = {
    "Content-Type": "application/json"
}

user_prompt = input("Enter your prompt for Gemini: ")

data = {
    "contents": [{
        "parts": [{
            "text": user_prompt
        }]
    }]
}

print("\n--- Sending request to Gemini... ---")
response = requests.post(url, headers=headers, data=json.dumps(data))

if response.status_code == 200:
    resp_json = response.json()
    try:
```

```
        text = resp_json["candidates"][0]["content"]["parts"][0]["text"]
        print("\n--- Gemini's response ---")
        print(text)
    except (KeyError, IndexError):
        print("Error parsing Gemini response:")
        print(resp_json)
else:
    print(f"Error status: {response.status_code}")
    print(response.text)
```

This example reinforces reading keys from `.env`, building JSON payloads, and handling non-200 responses.

---

17. Python demo: calling Groq chat completions API

This script sends the same user prompt to Groq's chat completions endpoint and measures response time.

```
import os
import json
import time
import requests
from dotenv import load_dotenv

load_dotenv()

api_key = os.getenv("GROQ_API_KEY")
if not api_key:
    raise ValueError("Groq API Key not found. Set GROQ_API_KEY in .env")

url = "https://api.groq.com/openai/v1/chat/completions"

headers = {
    "Authorization": f"Bearer {api_key}",
    "Content-Type": "application/json"
```

```
}

user_prompt = input("Enter your prompt for Groq: ")

data = {
    "model": "llama-3.3-70b-versatile",
    "messages": [
        {"role": "user", "content": user_prompt}
    ]
}

print("\n--- Sending request to Groq... ---")
start = time.time()
response = requests.post(url, headers=headers, data=json.dumps(data))
end = time.time()

if response.status_code == 200:
    resp_json = response.json()
    try:
        text = resp_json["choices"][0]["message"]["content"]
        print("\n--- Groq's response ---")
        print(text)
        print(f"\nTime taken: {end - start:.2f} seconds")
    except (KeyError, IndexError):
        print("Error parsing Groq response:")
        print(resp_json)
else:
    print(f"Error status: {response.status_code}")
    print(response.text)
```

Students can run both scripts with the same prompt and compare style and latency across providers.