1. **What is Pickle and why it is used ?**  
   In Python, **pickle** is a module used for **serializing** and **deserializing** Python objects. Serialization (also called "pickling") means converting a Python object into a byte stream, and deserialization ("unpickling") is converting that byte stream back into a Python object.

**🔧 Use Case:**

pickle is useful when you want to save Python objects (like lists, dictionaries, or even custom classes) to a file or transmit them over a network.

sample code :  
  
import pickle

# Sample data

data = {'name': 'Alice', 'age': 30, 'city': 'New York'}

# Serialize (pickle) the object to a file

with open('data.pkl', 'wb') as f:

pickle.dump(data, f)

# Deserialize (unpickle) the object from the file

with open('data.pkl', 'rb') as f:

loaded\_data = pickle.load(f)

print(loaded\_data)

1. **What is faiss in Python?**

faiss stands for **Facebook AI Similarity Search**. It is a **library developed by Facebook AI Research** to **perform efficient similarity search and clustering of dense vectors** (typically high-dimensional embeddings, like those from NLP or image models).

**📦 What faiss Does:**

faiss is mainly used for:

* **Fast nearest neighbor search** in large datasets.
* **Similarity search** between vectors (e.g., finding the most similar sentences, images, or documents).
* **Clustering** of high-dimensional data.
* **Indexing** vectors in a memory-efficient and query-optimized way.

**✅ Why Use faiss?**

If you’re working with:

* **Embeddings from machine learning models** (e.g., BERT, CLIP, OpenAI Embeddings)
* **Recommendation systems**
* **Semantic search engines**
* **Vector databases**
* **Large-scale retrieval problems**

Then faiss provides tools that are:

* **Very fast** (especially with GPU support)
* **Memory-efficient**
* Scalable to **millions of vectors**

**🔧 Basic Example:**

python

CopyEdit

import faiss

import numpy as np

# Create some 128-dimensional vectors

d = 128

nb = 1000

query\_vector = 1

# Generate random vectors

data = np.random.random((nb, d)).astype('float32')

query = np.random.random((query\_vector, d)).astype('float32')

# Build index

index = faiss.IndexFlatL2(d) # L2 = Euclidean distance

index.add(data) # Add data to index

# Search the nearest 5 neighbors

D, I = index.search(query, 5) # D = distances, I = indices

print("Nearest indices:", I)

print("Distances:", D)

**🧠 Summary:**

* faiss is used for **fast similarity search** over large-scale vector data.
* It’s common in **AI**, **NLP**, **image search**, and **retrieval-based applications**.
* It can work on **CPU or GPU** and supports multiple indexing strategies (flat, IVF, PQ, HNSW, etc.).

Need to run : pip install faiss-cpu   
  
See github : Practice->GPT4 faiss folder

**FAISS\_INDEX = "text\_index.faiss"  
What is a .faiss file?**

A .faiss file (like text\_index.faiss) is a serialized FAISS index — meaning it stores a pre-built vector index on disk so you can reuse it later without rebuilding it from scratch.

**What is a .faiss file?**

A .faiss file (like text\_index.faiss) is a **serialized FAISS index** — meaning it stores a pre-built vector index on disk so you can **reuse it later without rebuilding** it from scratch.

**🔧 Why Use a .faiss File?**

When you're working with **large datasets** or **precomputed vector embeddings**, it can take time to build the FAISS index. Saving it to a file lets you:

* ✅ Avoid recomputing the index every time you run your code.
* ✅ Reuse the same index across different Python scripts or sessions.
* ✅ Share or deploy the index with other systems

**What is fitz?**

* fitz is the original name of the PyMuPDF binding.
* When you import fitz, you're actually importing PyMuPDF, but using its legacy name.

import fitz # This is PyMuPDF

**🔹 What Can You Do with fitz / PyMuPDF?**

**PyMuPDF allows you to:**

* Read and write PDF files.
* Extract text, images, and metadata.
* Search and highlight text.
* Modify PDFs (add annotations, pages, etc.).
* Work with other formats like EPUB, XPS, OpenXPS, and CBZ.

**🔹 Example Usage**

import fitz # PyMuPDF

# Open a PDF file

doc = fitz.open("example.pdf")

# Print number of pages

print("Total pages:", len(doc))

# Extract text from the first page

page = doc[0]

text = page.get\_text()

print(text)

# Close the document

doc.close()

**🔹 Installation**

**You can install it using pip:**

pip install PyMuPDF

**Then use it in your code via:**

**import fitz.**

**What is Pillow?(Used for image processing )**

**Pillow** is a free and open-source library for the Python programming language that is used for image processing and manipulation. It is a fork of the original Python Imaging Library (PIL), which was discontinued. Pillow provides enhanced features, bug fixes, and crucially, support for Python 3.

Key functionalities of Pillow include:

* Opening, manipulating, and saving various image file formats: This includes common formats like JPEG, PNG, GIF, TIFF, BMP, and more.
* Basic image operations: Resizing, cropping, rotating, flipping, and adjusting color.
* Image enhancements: Applying filters (e.g., blur, sharpen), adjusting contrast, brightness, and color balance.
* Drawing and text operations: Adding text, shapes, and lines to images.
* Image analysis: Retrieving image properties like size, mode, and pixel data.  
    
  cmd : pip install pillow

**from urllib.parse import urlparse**

**What is urllib.parse?**

Urllib.parse is an standard library module in python.

It provides functions to split up,combine,encode,decode URLs.

Useful for manipulating URL query strings,path and parameters.

Urlparse : is a function from the urlib.parse module in python.

It splits a URL string into its components.

* Scheme (http,https,ftp etc)
* netloc (domain name + optional port)
* path (the path after the domain)
* params(optional parameters for the last path element)
* query (the query string ., ? id=1&name=test)
* fragment(the part after #)

**Why it is used ?**  
to safely extract parts of url.

To modify some parts of url(like query parameters) and rebuild url.

Very useful in web scraping,APIs ot building dynamic URLs.

**tqdm** is a popular Python library used to display smart progress meters for loops and iterables. Its name comes from the Arabic word "taqaddum" (تقدّم), meaning "progress."

Key features and usage:

* **Simple Integration:** You can easily add a progress bar by wrapping any iterable with tqdm().

Python

from tqdm import tqdm  
  
 for i in tqdm(range(100)):  
 *# Your code here*  
 pass

* trange() shortcut: For loops over a range, trange(N) can be used as a convenient shortcut for tqdm(range(N)).

Python

from tqdm import trange  
  
 for i in trange(100):  
 *# Your code here*  
 pass

**What is shutil?**

shutil is a built in python standard library module for high-level file operations.

* Copy files and directories .
* Move files and directories.
* Delete entire folders.
* Archive files.

It’s basically your go -to tool for “shell utilities” – that’s why shutil means Shell UTILities.

* Shutil.copyfile(str,dst)
* Shutil.copytree(str,dst) – copies an entire folder and it s contents.
* Shutil.rmtree(path) - Deletes an entire folder and its contnetns
* Shutil.move(src,dst) – Moves a file or folder from one place to another.

**What are Sentence Transformers?**

Sentence Transformers is a Python library for generating dense vector representations (embeddings) for whole sentences, paragraphs, or documents.

👉 Unlike standard BERT, which gives you embeddings for each token (word piece), Sentence Transformers combines those into a single fixed-size vector for the entire input.

👉 These embeddings capture meaning, so that:

Similar sentences have embeddings that are close together in vector space.

Dissimilar ones are far apart.

It’s perfect for:

✅ Semantic similarity (e.g., “How similar are these two texts?”)

✅ Semantic search (finding text that matches a query by meaning)

✅ Clustering

✅ Paraphrase mining

✅ Information retrieval (like matching resumes to JDs!)

📚 How does it work?

It uses a Transformer model (like BERT, RoBERTa, or MiniLM) trained to create embeddings that make semantic similarity tasks easy.

You don’t need to build or train it from scratch — just use a pre-trained model!

Common models: 'all-MiniLM-L6-v2' (small, fast, accurate) or 'paraphrase-MiniLM-L6-v2'.

⚙️ **How to install**

bash

Copy code

pip install sentence-transformers

✅ Basic Example

👉 Example 1: Compare two sentences

python

Copy code

from sentence\_transformers import SentenceTransformer, util

# Load a pre-trained sentence transformer model

model = SentenceTransformer('all-MiniLM-L6-v2')

# Two example sentences

sentence1 = "I love machine learning and AI."

sentence2 = "Artificial intelligence and machine learning are my passion."

# Encode sentences to get embeddings

embedding1 = model.encode(sentence1)

embedding2 = model.encode(sentence2)

# Compute cosine similarity

cosine\_score = util.cos\_sim(embedding1, embedding2)

print(f"Cosine Similarity: {cosine\_score.item():.4f}")

Output:

sql

Copy code

Cosine Similarity: 0.87  (values near 1 mean very similar)

👉 Example 2: Find most similar sentence in a list

python

Copy code

from sentence\_transformers import SentenceTransformer, util

model = SentenceTransformer('all-MiniLM-L6-v2')

# Your query

query = "Looking for a developer with Python and Flask skills."

# Candidate sentences (like resumes, snippets, or job postings)

candidates = [

    "Experienced Python developer with Flask and Django.",

    "Expert in JavaScript and React frontend.",

    "Data analyst with SQL and Tableau experience."

]

# Encode all sentences

query\_embedding = model.encode(query)

candidate\_embeddings = model.encode(candidates)

# Compute cosine similarities

cosine\_scores = util.cos\_sim(query\_embedding, candidate\_embeddings)

# Show results

for idx, score in enumerate(cosine\_scores[0]):

    print(f"Candidate {idx+1}: Score: {score.item():.4f} -> {candidates[idx]}")

👉 Example 3: Batch encode for a corpus

python

Copy code

from sentence\_transformers import SentenceTransformer

model = SentenceTransformer('all-MiniLM-L6-v2')

# A batch of texts (e.g., resumes, documents)

documents = [

    "Senior backend developer with Python and Django.",

    "Frontend engineer specializing in React and Vue.",

    "DevOps engineer with AWS and Docker experience."

]

# Get embeddings for all

embeddings = model.encode(documents)

print(f"Shape of embeddings: {embeddings.shape}")  # (3, 384) -> 3 documents, 384-dim vectors

✅ Key Takeaways

📌 Sentence Transformers = one vector per sentence / paragraph that encodes meaning.

📌 They’re super useful for semantic search, ranking, clustering, and matching text by meaning.

📌 Hugging Face hosts many pre-trained models — you can just plug them in!

🧠 *Common models*

Model name Description

'all-MiniLM-L6-v2' Fast, general-purpose, 384 dimensions

'paraphrase-MiniLM-L6-v2' Optimized for paraphrase similarity

'all-mpnet-base-v2' Larger, more accurate, but slower

✅ Where you’re using it

In your resume shortlisting app, you:

1️⃣ Embed the Job Description

2️⃣ Embed the Resume Text

3️⃣ Use cosine similarity to see how close they are!

That’s the magic! 🔍✨

If you want, I can also show:

✅ How to fine-tune your own sentence transformer

✅ How to store embeddings in FAISS or a vector DB

✅ How to visualize embedding clusters

Just say “Show me more!” and I’ll expand! 🚀

**What is PyPDF2?**

PyPDF2 is a popular Python library used for reading, manipulating, and writing PDF files.

📄 What can you do with PyPDF2?

Here are the core things PyPDF2 can do:

✅ Extract text from PDF pages (like resumes)

✅ Merge multiple PDFs into one file

✅ Split a PDF into individual pages

✅ Rotate or reorder pages

✅ Encrypt or decrypt PDF files

📄 In your project

You use PyPDF2 to:

Open the resume PDF

Extract the text from each page

Combine all the text into a single string

Pass that text into your embedding model for similarity scoring.

🗂️ How does it work? (Your code example)

python

Copy code

import PyPDF2

def extract\_text\_from\_pdf(pdf\_path):

    text = ""

    with open(pdf\_path, 'rb') as file:

        reader = PyPDF2.PdfReader(file)  # Create a PDF reader object

        for page in reader.pages:

            text += page.extract\_text()  # Extract text from each page

    return text

Line-by-line:

with open(pdf\_path, 'rb') → Open the PDF in binary mode (PDFs are not plain text files).

PyPDF2.PdfReader(file) → Loads the PDF and lets you work with its pages.

reader.pages → A list of all pages in the PDF.

page.extract\_text() → Grabs the raw text from that page.

Combine it all: text += ... → Builds the full resume text.

✅ Why use PyPDF2?

Advantages:

Simple and free for basic PDF tasks.

Works well for extracting text from mostly text-based PDFs (like resumes).

No external tools needed.

⚡ Limitations

It struggles with PDFs that are scanned images — because those aren’t real text, they’re images.

For scanned PDFs, you’d need an OCR tool like pytesseract to convert images to text.

📦 Install it

bash

Copy code

pip install PyPDF2

📝 Summary

Aspect PyPDF2 Does

Extract text ✅ Good for text-based PDFs

Merge/Split ✅ Combine or split PDF files

Rotate/Reorder ✅ Fix page order or orientation

Secure files ✅ Add/remove passwords

OCR ❌ Not built-in (need pytesseract for images)

If you’d like, I can show you:

✅ How to use OCR for scanned resumes

✅ How to merge or split PDF files with PyPDF2

✅ Or a better library for advanced PDF tasks

Just say “Show me OCR” or “Show me split/merge!” and I’ll write you an example! 🚀

You said:

Give some sample code

ChatGPT said:

Absolutely! Here’s a handy collection of PyPDF2 examples — short and clear — so you can see how it works for different tasks.

✅ 1️⃣ Extract text from a PDF (like a resume)

python

Copy code

import PyPDF2

def extract\_text\_from\_pdf(pdf\_path):

    text = ""

    with open(pdf\_path, 'rb') as file:

        reader = PyPDF2.PdfReader(file)

        for page in reader.pages:

            page\_text = page.extract\_text()

            if page\_text:

                text += page\_text

    return text

# Example usage:

pdf\_path = "example\_resume.pdf"

resume\_text = extract\_text\_from\_pdf(pdf\_path)

print(resume\_text)

✅ 2️⃣ Merge multiple PDFs into one

python

Copy code

from PyPDF2 import PdfMerger

def merge\_pdfs(pdf\_list, output\_name):

    merger = PdfMerger()

    for pdf in pdf\_list:

        merger.append(pdf)

    merger.write(output\_name)

    merger.close()

# Example usage:

pdf\_files = ["resume\_part1.pdf", "resume\_part2.pdf"]

merge\_pdfs(pdf\_files, "merged\_resume.pdf")

✅ 3️⃣ Split a PDF into separate pages

python

Copy code

from PyPDF2 import PdfReader, PdfWriter

def split\_pdf(pdf\_path):

    reader = PdfReader(pdf\_path)

    for i, page in enumerate(reader.pages):

        writer = PdfWriter()

        writer.add\_page(page)

        output\_filename = f"page\_{i+1}.pdf"

        with open(output\_filename, "wb") as output\_pdf:

            writer.write(output\_pdf)

# Example usage:

split\_pdf("example\_resume.pdf")

✅ 4️⃣ Rotate pages (e.g., fix upside-down scans)

python

Copy code

from PyPDF2 import PdfReader, PdfWriter

def rotate\_pdf(pdf\_path, output\_path, rotation\_angle=90):

    reader = PdfReader(pdf\_path)

    writer = PdfWriter()

    for page in reader.pages:

        page.rotate(90)  # Rotate by 90 degrees clockwise

        writer.add\_page(page)

    with open(output\_path, "wb") as f:

        writer.write(f)

# Example usage:

rotate\_pdf("example\_resume.pdf", "rotated\_resume.pdf")

✅ 5️⃣ Add a password to a PDF

python

Copy code

from PyPDF2 import PdfReader, PdfWriter

def encrypt\_pdf(input\_pdf, output\_pdf, password):

    reader = PdfReader(input\_pdf)

    writer = PdfWriter()

    for page in reader.pages:

        writer.add\_page(page)

    writer.encrypt(password)

    with open(output\_pdf, "wb") as f:

        writer.write(f)

# Example usage:

encrypt\_pdf("example\_resume.pdf", "protected\_resume.pdf", "mysecretpassword")

⚡ Quick Notes

📄 PdfReader → Loads an existing PDF.

🖊️ PdfWriter → Writes new PDFs.

🪢 PdfMerger → Combines multiple PDFs.

🔑 Encryption → Adds simple password protection.

🚫 OCR → PyPDF2 can’t read scanned images — you’d need pytesseract for that.

If you’d like, I can show:

✅ An example for OCR with scanned PDFs

✅ Or wrap any of these in a console app like your resume shortlisting!

Just say “Show me OCR version!” and I’ll build it for you. 📚✨

📘 DSPy (Declarative Self-improving Python) – Notes

🧠 What is DSPy?

DSPy is a Python framework to build modular, declarative, and self-improving workflows using large language models (LLMs).

It allows you to define prompts and logic in a structured way, much like defining functions.

Think of it as a bridge between traditional Python programming and LLM prompt engineering.

---

🔧 Key Concepts

Concept Description

dspy.Predict - A base class to define a module that takes inputs and generates outputs using an LLM.

Signature - Declares what the model should expect as inputs and what it should return.

forward() - method Implements how the output is generated using the inputs.

Compiler Optimizes and compiles multiple modules into a pipeline.

---

📦 Installation

pip install dspy

for open ai :

# pip install dspy-ai

# pip install "dspy-ai[openai]"

📘 LangChain

🔹 What is LangChain?

LangChain is an open-source Python framework designed to build applications with Large Language Models (LLMs). It allows developers to create powerful, context-aware, and data-connected AI apps using components like prompts, chains, agents, memory, and retrievers.

---

🔹 Why Use LangChain?

LangChain simplifies the process of:

Creating multi-step LLM workflows

Connecting LLMs with external data (files, web, DBs)

Managing conversational memory

Enabling agents to make decisions and use tools

Building scalable, production-ready LLM-powered applications

---

🔹 Core Components

Component Description

🔸 PromptTemplate Used to format dynamic prompts for LLMs

🔸 LLMChain A simple chain that takes a prompt and passes it to the LLM

🔸 Memory Stores and retrieves past interactions

🔸 Agent Chooses tools dynamically based on input

🔸 Tool External functions like web search, calculator, DB access

🔸 Retriever Fetches relevant info from documents for Q&A

🔸 Document Loaders Load data from PDFs, web pages, etc.

---

🔹 Installation

pip install langchain

You may also need:

pip install openai chromadb faiss-cpu