Build an Advanced AI Research Assistant (Chatbot) that integrates LLM, RAG, and Agent-based architecture.

Import Useful Library

In [1]:

```python
import os
import re
import json
import random
import numpy as np
import pandas as pd

from nltk.tokenize import sent_tokenize
from datasets import Dataset
import torch
import faiss

import transformers
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
from sentence_transformers import SentenceTransformer, CrossEncoder
transformers.logging.set_verbosity_error()
```

File Path and Device Configuration

In [2]:

```python
INPUT_FILE = os.path.join(os.getcwd(), "s2orc Dataset", "ComputerScience,2022-2022_test.jsonl")
DEVICE = "cpu"  # 🔒 For CPU only
```

Preprocessing

In [3]:

```python
def extract_cleaned_papers(input_file, max_docs, target_field):
    documents = []
    with open(input_file, 'r', encoding='utf-8') as infile:
        count = 0
        for line in infile:
            paper = json.loads(line)
            if paper.get("metadata", {}).get("s2fieldsofstudy", [None])[0] != target_field:
                continue
            raw_text = paper.get("text", "")
            title_match = re.match(r"^(.*?)\n", raw_text.strip())
            title = title_match.group(1).strip() if title_match else "Untitled"
            content = raw_text.replace(title, "").strip()

            documents.append({
                "id": paper["id"],
                "title": title,
                "content": content,
                "year": paper["metadata"].get("year", None),
                "field": paper["metadata"].get("s2fieldsofstudy", [])
            })

            count += 1
            if count >= max_docs:
                break
    return documents

documents = extract_cleaned_papers(INPUT_FILE, max_docs=5000, target_field="Computer Science")
print(f"✅ Loaded {len(documents)} documents")
df = pd.DataFrame(documents)
df.head()
```

```
✅ Loaded 326 documents
```

Out[3]:

	id	title	content	year	field
0	247411061	DUNE Software and High Performance Computing	DUNE, like other HEP experiments, faces a chal...	2022	[Computer Science]
1	247597192	A Perspective on Neural Capacity Estimation: V...	Recently, several methods have been proposed f...	2022	[Computer Science]
2	245668726	Adaptive Template Enhancement for Improved Per...	A novel instance-based method for the classifi...	2022	[Computer Science]
3	248300244	Re-Examining System-Level Correlations of Auto...	How reliably an automatic summarization evalua...	2022	[Computer Science]
4	254854674	Iso-Dream: Isolating and Leveraging Noncontrol...	World models learn the consequences of actions...	2022	[Computer Science]

Chunking: Chunk documents into passages

Chunks the document into overlapping chunks using a sliding window over sentences. This avoids splitting mid-sentence and keeps semantic coherence.

In [4]:

```python
def chunk_document(doc, max_tokens=300, stride=150):
    sentences = sent_tokenize(doc)
    chunks = []
    current_chunk = []
    current_len = 0
    i = 0
    while i < len(sentences):
        current_chunk = []
        current_len = 0
        j = i
        while j < len(sentences) and current_len + len(sentences[j].split()) <= max_tokens:
            current_chunk.append(sentences[j])
            current_len += len(sentences[j].split())
            j += 1
        chunks.append(" ".join(current_chunk))
        i += stride if stride > 0 else j  # Slide forward by 'stride' or jump to next non-overlapping
    return chunks

print("🔄 Chunking documents...")
chunked_docs = []
titles = []
for doc in documents:
    chunks = chunk_document(doc["content"], max_tokens=300, stride=150)
    chunked_docs.extend(chunks)
    titles.extend([doc["title"]] * len(chunks))
print(f"Total passages after chunking: {len(chunked_docs)}")

# Print sample chunks with the title
for i in range(2):
    print(f"\n📄 Chunk {i+1} (Title: {titles[i]}):")
    print(chunked_docs[i])
    print(f"\n🔢 Tokens: {len(chunked_docs[i].split())}")
```

```
🔄 Chunking documents...
Total passages after chunking: 752

📄 Chunk 1 (Title: DUNE Software and High Performance Computing):
DUNE, like other HEP experiments, faces a challenge related to matching execution patterns of our production simulation and data processing
software to the limitations imposed by modern high-performance computing facilities. In order to efficiently exploit these new architecture
s, particularly those with high CPU core counts and GPU accelerators, our existing software execution models require adaptation. In additio
n, the large size of individual units of raw data from the far detector modules pose an additional challenge somewhat unique to DUNE. Here
we describe some of these problems and how we begin to solve them today with existing software frameworks and toolkits. We also describe wa
ys we may leverage these existing software architectures to attack remaining problems going forward. This whitepaper is a contribution to t
he Computational Frontier of Snowmass21. Introduction
DUNE [1] software faces many software development and data processing challenges shared with other large HEP experiments. Moore's law has m
orphed from delivering ever faster CPUs to delivering ever more numerous and sometimes even slower cores. This is epitomized by GPUs which
provide O(1k) cores per card but which are clocked far slower than even low-end CPUs. We must develop software that can efficiently exploit
the highly parallel architectures of high-performance computers (HPC) while still making efficient use of the more familiar high-throughput
computing (HTC) facilities that offer modest parallelism of O(10) CPU cores per compute node. Contemporaneously, the available RAM capacity
associated with CPU and GPU is not growing as fast as their core counts and RAM/core ratio has become a limiting parameter for many types o
f jobs. Where possible we must reimplement and rearchitect to reduce memory waste. On the other hand, many classes of jobs have an unavoida
ble memory overhead.

🔢 Tokens: 282

📄 Chunk 2 (Title: DUNE Software and High Performance Computing):
It works by allowing only a limited number of GPU tasks to be submitted at any given time. Once the limit is reached, a subsequent task mus
t wait for "its turn". Currently, this waiting is implemented by allowing the thread and thus the CPU core that issued the task to go idle.
This idleness will tend to strike and remove cores from making progress precisely when CPU tasks are most needed. To break this bottleneck
we plan to implement a so far ignored type of node provided by the TBB flow_graph library called async_node. Such a node is allowed a free
thread under the assumption it will be lightly used. This allows an asynchronous query protocol with the graph execution engine. Essentiall
y, the node may be periodically checked for completion. This can cause delay in the GPU task return but does not lead to CPU nor GPU idlene
ss. Another imperfection with the current semaphore based GPU load balancing is that the semaphore is implemented at the C++ language leve
l. As such it produces a tight code binding between all flow-graph node implementations that wish to issue GPU tasks. This goes against the
general wellfactored, interface-based code architecture of the toolkit. As the semaphore is process-local it is useless for moderation of G
PU access which is shared across processes. To solve this we expect to develop a distributed application composed of multiple processes bas
ed on art or WCT or in some cases both. The flow-graphs will have nodes that provide communication between processes which potentially may
be executing on different compute nodes. Initial plans for this distributed architecture were based on the ZIO [9] project. More recently,
investigation into ADIOS2 [10] show it as a promising and likely preferred alternative.

🔢 Tokens: 289
```

Prepare dataset for embedding

In [5]:
```
data = {"title": titles, "text": chunked_docs}
dataset = Dataset.from_dict(data)

ef = dataset.to_pandas()
ef.insert(0, "chunk_id", range(1, len(ef)+1))

random.sample([dict(i) for i in dataset], k=1)
```

Out[5]:
```
[{'title': 'Modeling and Estimation of CO2 Emissions in China Based on Artificial Intelligence',
  'text': "Since China's reform and opening up, the social economy has achieved rapid development, followed by a sharp increase in carbon d
ioxide (CO2) emissions. Therefore, at the 75th United Nations General Assembly, China proposed to achieve carbon peaking by 2030 and carbon
neutrality by 2060. The research work on advance forecasting of CO2 emissions is essential to achieve the above-mentioned carbon peaking an
d carbon neutrality goals in China. In order to achieve accurate prediction of CO2 emissions, this study establishes a hybrid intelligent a
lgorithm model suitable for CO2 emissions prediction based on China's CO2 emissions and related socioeconomic indicator data from 1971 to 2
017. The hyperparameters of Least Squares Support Vector Regression (LSSVR) are optimized by the Adaptive Artificial Bee Colony (AABC) algo
rithm to build a high-performance hybrid intelligence model. The research results show that the hybrid intelligent algorithm model designed
in this paper has stronger robustness and accuracy with relative error almost within ±5% in the advance prediction of CO2 emissions. The mo
deling scheme proposed in this study can not only provide strong support for the Chinese government and industry departments to formulate p
olicies related to the carbon peaking and carbon neutrality goals, but also can be extended to the research of other socioeconomic-related
issues. Introduction\nGlobal warming has become a fact generally accepted by the international community. Climate warming has seriously aff
ected the living environment and social development of humankind. Although the cyclical changes in the natural environment itself affect gl
obal climate change, more and more studies have shown that human activities have accelerated the global warming process largely. Since the
industrial revolution, with the development of social economy and the increasing intensity of human activities, the concentration of carbon
dioxide (CO 2 ) in the atmosphere has risen sharply [1]."}]
```

Embedding Models

For our project, we go with:

1) **multiqa_mini:** Multi-domain QA MiniLM model

2) **scibert:** SciBERT fine-tuned on scientific vocab

1) **allmini_v2:** General-purpose MiniLM v2 embeddings
2) **mpnet_base:** High-accuracy general-purpose MPNet model

Each document is encoded into a high-dimensional vector using SentenceTransformer.

In [6]:
```python
def embed_chunk(model_name, dataset, batch_size=32):
    model = SentenceTransformer(model_name)
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    text_chunks = [chunk["text"] for chunk in dataset]

    if "scibert" in model_name:      # Preprocess and truncate input for scibert model
        final_chunks = []
        for text in text_chunks:
            input_ids = tokenizer.encode(text, max_length=512, truncation=True, add_special_tokens=True)
            decoded_text = tokenizer.decode(input_ids, skip_special_tokens=True)
            final_chunks.append(decoded_text)
    else:
        final_chunks = text_chunks

    embeddings_tensor = model.encode(final_chunks, batch_size=batch_size, convert_to_tensor=True, show_progress_bar=True)
    embeddings_list = embeddings_tensor.cpu().numpy().tolist()
    return [embeddings_tensor, embeddings_list]


embedding_models = {
    "multiqa": "sentence-transformers/multi-qa-MiniLM-L6-cos-v1",
    "scibert": "allenai/scibert_scivocab_uncased",
    "allmini": "sentence-transformers/all-MiniLM-L6-v2",
    "mpnet": "sentence-transformers/all-mpnet-base-v2"
}

model_embed_list = []
for key, name in embedding_models.items():
    print(f"\n🔄 Processing with model: {key}")
    e = embed_chunk(name, dataset)
    model_embed_list.append(e[0])
    ef["embedding_"+key] = e[1]

# ef.to_excel('Embeddings.xlsx', index=False)
ef.head()
```

```
🔄 Processing with model: multiqa
Batches:   0%|          | 0/24 [00:00<?, ?it/s]
🔄 Processing with model: scibert
Batches:   0%|          | 0/24 [00:00<?, ?it/s]
🔄 Processing with model: allmini
Batches:   0%|          | 0/24 [00:00<?, ?it/s]
🔄 Processing with model: mpnet
Batches:   0%|          | 0/24 [00:00<?, ?it/s]
```

Out[6]:

	chunk_id	title	text	embedding_multiqa	embedding_scibert	embedding_allmini	embedding_mpnet
0	1	DUNE Software and High Performance Computing	DUNE, like other HEP experiments, faces a chal...	[0.07001133263111115, -0.09193330258131027, -0...	[0.17008523643016815, -0.41661253571510315, 0....	[-0.19705259799957275, 0.0813499465584755, -0....	[0.035186517983675, 0.04750188812613487, -0.03...
1	2	DUNE Software and High Performance Computing	It works by allowing only a limited number of ...	[-0.11874090135097504, -0.09827910363674164, 0...	[0.038926806300878525, -0.09388677775859833, -...	[-0.1930292844772339, 0.05765102058649063, -0....	[-0.024255581200122833, -0.10105226188898087, ...
2	3	A Perspective on Neural Capacity Estimation: V...	Recently, several methods have been proposed f...	[-0.060939595103263855, -0.14694444835186005, ...	[0.5142149925231934, -0.32449233531951904, -0....	[-0.06463883817195892, -0.13580818474292755, -...	[-0.07576467096805573, 0.08724159747362137, 0....
3	4	A Perspective on Neural Capacity Estimation: V...	In estimating MI, the samples for the product ...	[-0.08345496654510498, -0.16691753268241882, -...	[0.31848376989364624, -0.11525746434926987, -0...	[-0.1146080270409584, -0.10533967614173889, 0....	[-0.09548977017402649, 0.05486003309488296 5, -...
4	5	A Perspective on Neural Capacity Estimation: V...	When comparing the numerical results with the ...	[0.01412767730653286, -0.14586158096790314, 0....	[0.16452282667160034, -0.18880560994148254, -0...	[-0.037659913301467896, -0.08678082376718521, ...	[-0.12054596096277237, -0.013146807439625263, ...

FAISS index

The embeddings are stored in a FAISS index for efficient similarity search. The FAISS index allows us to retrieve the most semantically similar documents to a given query in real-time.

In [8]:
```python
tensors_multiqa = model_embed_list[0]
tensors_scibert = model_embed_list[1]
tensors_allmini = model_embed_list[2]
tensors_mpnet   = model_embed_list[3]

# Convert PyTorch tensors to NumPy
np_multiqa = tensors_multiqa.detach().cpu().numpy().astype('float32')
np_scibert = tensors_scibert.detach().cpu().numpy().astype('float32')
np_allmini = tensors_allmini.detach().cpu().numpy().astype('float32')
np_mpnet   = tensors_mpnet.detach().cpu().numpy().astype('float32')

# Normalize embeddings and create a cosine similarity FAISS index
def build_faiss_cosine_index(embeddings_np):
    faiss.normalize_L2(embeddings_np)
    dim = embeddings_np.shape[1]
    index = faiss.IndexFlatIP(dim)
    index.add(embeddings_np)
    return index

# Create indexes
faiss_indexes = {
    "multiqa": build_faiss_cosine_index(np_multiqa),
    "scibert": build_faiss_cosine_index(np_scibert),
    "allmini": build_faiss_cosine_index(np_allmini),
    "mpnet":   build_faiss_cosine_index(np_mpnet),
}

# To find top-k similar documents to a query embedding
def search_faiss(index, query_tensor, top_k=3):
    query_np = query_tensor.detach().cpu().numpy().astype('float32').reshape(1, -1)
    faiss.normalize_L2(query_np)
    distances, indices = index.search(query_np, top_k)
    return distances[0], indices[0]

# Example: search similar documents using a Scibert embedding
query_tensor = tensors_scibert[0]
dists, idxs = search_faiss(faiss_indexes["scibert"], query_tensor, top_k=3)
for rank, (i, score) in enumerate(zip(idxs, dists), 1):
    print(f"{rank}. Doc Index: {i}, Similarity Score: {score:.4f}")
```

```
1. Doc Index: 0, Similarity Score: 1.0000
2. Doc Index: 293, Similarity Score: 0.9455
3. Doc Index: 336, Similarity Score: 0.9325
```

Implement a Retrieval Function

A function that takes a user query, embeds it using the same model, and retrieves the most relevant documents from the FAISS index.

In [9]:
```
def retrieve_top_docs_with_scores_and_rerank(query_text, dataset, faiss_indexes, top_k=3):
    embedding_models = {
        "multiqa": "sentence-transformers/multi-qa-MiniLM-L6-cos-v1",
        "scibert": "allenai/scibert_scivocab_uncased",
        "allmini": "sentence-transformers/all-MiniLM-L6-v2",
        "mpnet": "sentence-transformers/all-mpnet-base-v2"
    }
    reranker = CrossEncoder("cross-encoder/ms-marco-MiniLM-L6-v2")

    results = {}
    for model_name, model_path in embedding_models.items():
        model = SentenceTransformer(model_path)
        query_embedding = model.encode(query_text, convert_to_tensor=True)
        distances, indices = search_faiss(faiss_indexes[model_name], query_embedding, top_k)

        # Get top docs with FAISS scores
        top_docs = [(dataset[int(i)]["text"], float(distances[idx])) for idx, i in enumerate(indices)]

        # Get top docs with scores after reranking
        rerank_inputs = [(query_text, doc_text) for doc_text, _ in top_docs]
        rerank_scores = reranker.predict(rerank_inputs)
        combined = list(zip(top_docs, rerank_scores))  # [ ((doc_text, faiss_score), rerank_score), ... ]
        combined_sorted = sorted(combined, key=lambda x: x[1], reverse=True)   # Sort by reranker score descending
        top_docs_with_scores = [(doc_score[0], doc_score[1], rerank_score) for (doc_score, rerank_score) in combined_sorted]
        results[model_name] = top_docs_with_scores

        # Get results in tabular format
        rows = []
        for model_name, docs in results.items():
            for rank, (doc_text, faiss_score, rerank_score) in enumerate(docs, 1):
                rows.append({
                    "Model": model_name,
                    "Rank": rank,
                    "FAISS Score": faiss_score,
                    "Reranker Score": rerank_score,
                    "Doc": doc_text
                })
        df_results = pd.DataFrame(rows)

    return df_results

# Example:
query = "Can you tell me something about deep learning?"
results = retrieve_top_docs_with_scores_and_rerank(query, dataset, faiss_indexes, top_k=1)
results.head(100)
```

Out[9]:

	Model	Rank	FAISS Score	Reranker Score	Doc
0	multiqa	1	0.532506	0.702883	Several researchers have employed deep learnin...
1	scibert	1	0.640150	-11.057257	We sat in the outdoor patio area next to a few...
2	allmini	1	0.562497	0.702883	Several researchers have employed deep learnin...
3	mpnet	1	0.540243	0.702883	Several researchers have employed deep learnin...

Integrate a Language Model (LLM)

Generate a response using the LLM with retrieved document context.

In [10]:
```python
# Load LLM model and tokenizer
llm_model_name = "google/flan-t5-large"
llm_tokenizer = AutoTokenizer.from_pretrained(llm_model_name)
llm_model = AutoModelForSeq2SeqLM.from_pretrained(llm_model_name)

# Answer generation
def generate_answer(query, context_docs, model, tokenizer, device, max_gen_length=150):
    context = context_docs if isinstance(context_docs, str) else "\n---\n".join(context_docs)
    prompt = f"Answer the question based on the following documents.\n\n{context}\n\nQuestion: {query}\n\nAnswer:"

    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=min(512, tokenizer.model_max_length))
    inputs = {k: v.to(device) for k, v in inputs.items()}
    model = model.to(device)

    with torch.no_grad():
        output_ids = model.generate(
            input_ids=inputs["input_ids"],
            max_length=min(1024, inputs["input_ids"].shape[1] + max_gen_length),
            pad_token_id=tokenizer.pad_token_id,
            eos_token_id=tokenizer.eos_token_id,
            num_beams=4,
            do_sample=False
        )
    answer = tokenizer.decode(output_ids[0], skip_special_tokens=True).strip()
    if len(answer.split()) < 3 or any(term in answer for term in ["BVI", "Q-Task", "BTC", "coding gain"]):
        return "I'm not sure. Can you rephrase or ask a different question?"
    return answer

# CLI Chat Loop
def chat_loop(dataset, faiss_indexes, llm_model, llm_tokenizer, DEVICE):
    print("\n💬 Start chatting with your AI assistant (type 'exit' to stop)...\n")
    while True:
        query = input("🧑 You: ").strip()
        if query.lower() in ["exit", "quit"]:
            print("👋 Goodbye!")
            break
        df_results = retrieve_top_docs_with_scores_and_rerank(query, dataset, faiss_indexes, top_k=1)
        best_doc_text = df_results.loc[df_results["Reranker Score"].idxmax()]["Doc"]
        if best_doc_text:
            print("📄 Top doc snippet:", best_doc_text[:100].replace('\n', ' ') + "...")
        else:
            print("📄 No relevant documents found.\n")
        answer = generate_answer(query, best_doc_text, llm_model, llm_tokenizer, DEVICE)
        print(f"🤖 AI: {answer}\n")

chat_loop(dataset, faiss_indexes, llm_model, llm_tokenizer, DEVICE)
```

```
💬 Start chatting with your AI assistant (type 'exit' to stop)...

🧑 You: What challenge do modern high-performance computing (HPC) facilities pose for HEP experiments like DUNE?
📄 Top doc snippet: DUNE, like other HEP experiments, faces a challenge related to matching execution patterns of our pr...
🤖 AI: matching execution patterns of our production simulation and data processing software to the limitations imposed by modern high-per
formance computing facilities

🧑 You: Why does the large size of individual raw data units from the far detector modules pose a challenge for DUNE?
📄 Top doc snippet: DUNE, like other HEP experiments, faces a challenge related to matching execution patterns of our pr...
🤖 AI: DUNE, like other HEP experiments, faces a challenge related to matching execution patterns of our production simulation and data pr
ocessing software to the limitations imposed by modern high-performance computing facilities

🧑 You: How has Moore’s law changed in recent years, and how does this impact computing architectures?
📄 Top doc snippet: DUNE, like other HEP experiments, faces a challenge related to matching execution patterns of our pr...
🤖 AI: Moore's law has morphed from delivering ever faster CPUs to delivering ever more numerous and sometimes even slower cores

🧑 You: How does the current GPU task submission system work?
📄 Top doc snippet: It works by allowing only a limited number of GPU tasks to be submitted at any given time. Once the ...
🤖 AI: allowing only a limited number of GPU tasks to be submitted at any given time

🧑 You: What is the main drawback of the current waiting mechanism for GPU tasks?
📄 Top doc snippet: It works by allowing only a limited number of GPU tasks to be submitted at any given time. Once the ...
🤖 AI: idleness will tend to strike and remove cores from making progress precisely when CPU tasks are most needed

🧑 You: What solution is proposed to break the bottleneck caused by CPU idleness?
📄 Top doc snippet: It works by allowing only a limited number of GPU tasks to be submitted at any given time. Once the ...
🤖 AI: I'm not sure. Can you rephrase or ask a different question?

🧑 You: How does the use of an async_node improve GPU task handling?
📄 Top doc snippet: It works by allowing only a limited number of GPU tasks to be submitted at any given time. Once the ...
🤖 AI: This can cause delay in the GPU task return but does not lead to CPU nor GPU idleness

🧑 You: What approach is proposed to handle GPU access across multiple processes?
📄 Top doc snippet: It works by allowing only a limited number of GPU tasks to be submitted at any given time. Once the ...
🤖 AI: a distributed application composed of multiple processes based on art or WCT or in some cases both

🧑 You: exit
👋 Goodbye!
```