OpenVINO™

CVPR 2025

Cognitive AI for the Future: Multimodal Models and RAG in Vision Language Applications, from Training to Deployment

Module 3 :

Optimize and Deploy the Multimodal RAG Pipeline

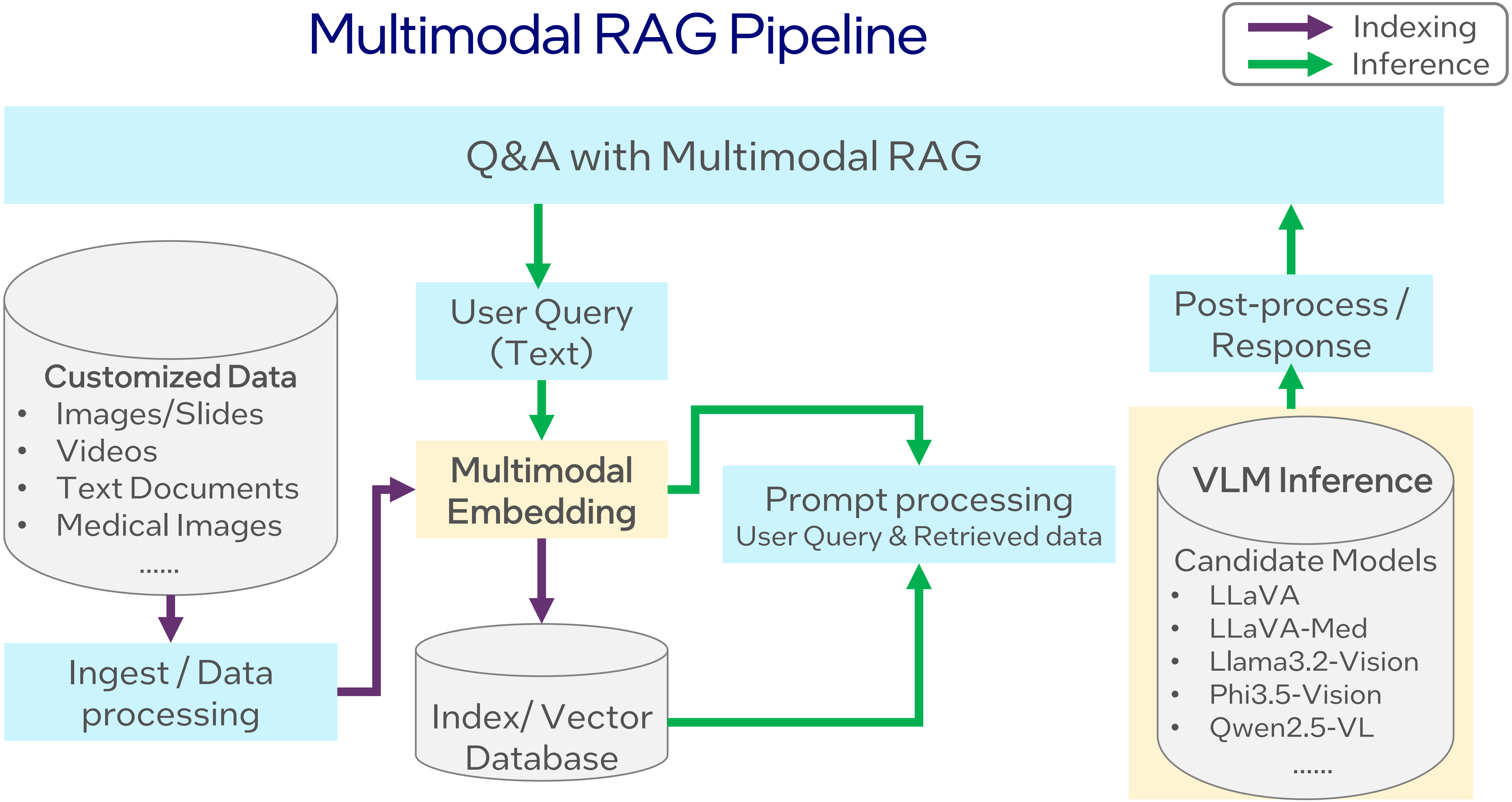
Speaker: Zhuo Wu

Job title: AI Software Evangelist

Outline (30min)

Overview the multimodal RAG pipeline	3m
Convert & optimize multimodal embedding	7m
RAG with LlamaIndex & OpenVINO	10m
Multimodal RAG based video search	10m

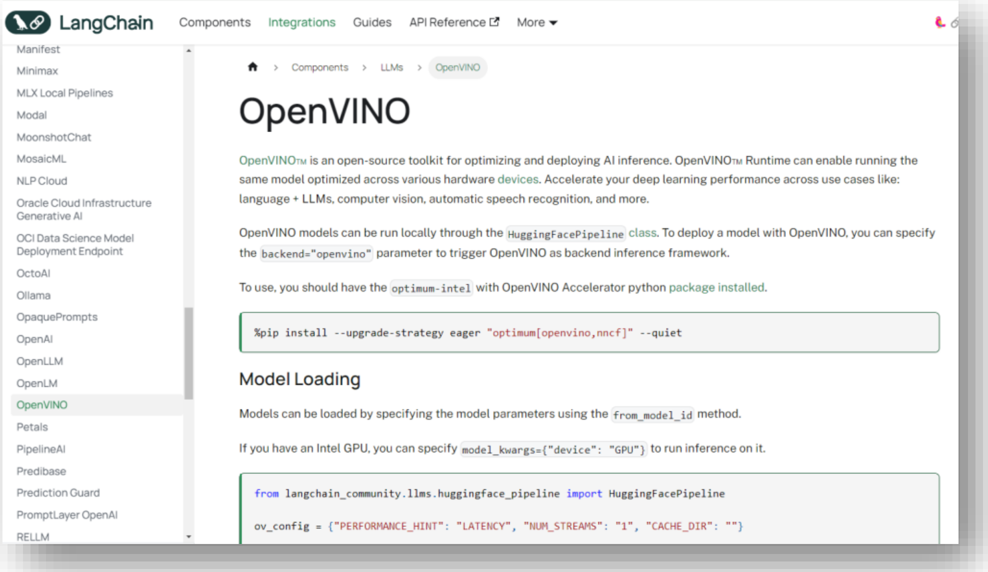
Multimodal RAG Pipeline



Retrieval-Augmented Generation (RAG)

LLM Ecosystem Integration

LangChain x OpenVINO™ x LlamaIndex



OpenVINO

OpenVINO™ is an open-source toolkit for optimizing and deploying AI inference. OpenVINO™ Runtime can enable running the same model optimized across various hardware devices. Accelerate your deep learning performance across use cases like: language + LLMs, computer vision, automatic speech recognition, and more.

OpenVINO models can be run locally through the `HuggingFacePipeline` class. To deploy a model with OpenVINO, you can specify the `backend="openvino"` parameter to trigger OpenVINO as backend inference framework.

To use, you should have the `optimum-intel` with OpenVINO Accelerator python package installed.

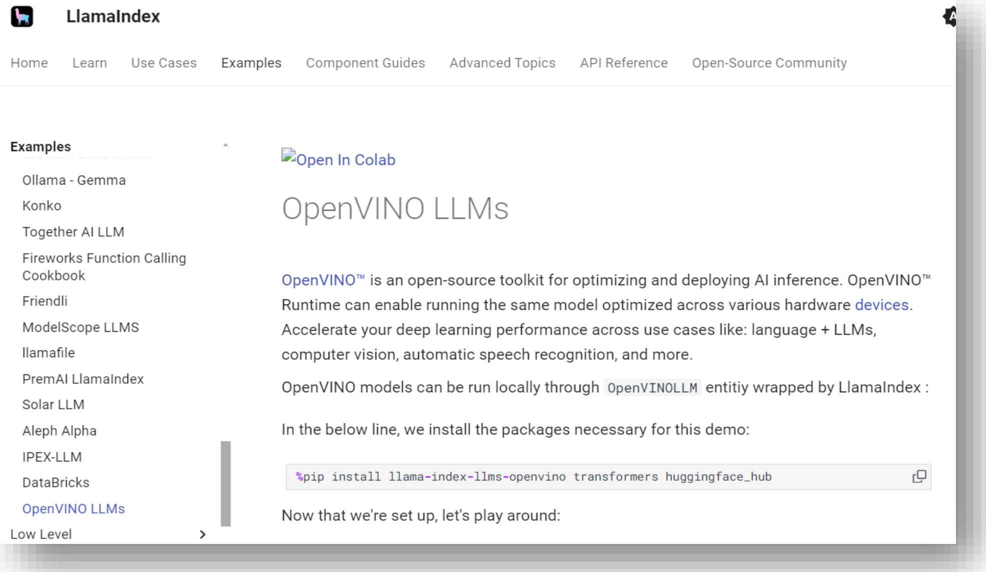
```
%pip install --upgrade-strategy eager "optimum[openvino,nncf]" --quiet
```

Model Loading

Models can be loaded by specifying the model parameters using the `from_model_id` method.

If you have an Intel GPU, you can specify `model_kwargs={"device": "GPU"}` to run inference on it.

```
from langchain_community.llms.huggingface_pipeline import HuggingFacePipeline

ov_config = {"PERFORMANCE_HINT": "LATENCY", "NUM_STREAMS": "1", "CACHE_DIR": ""}
```



OpenVINO LLMs

OpenVINO™ is an open-source toolkit for optimizing and deploying AI inference. OpenVINO™ Runtime can enable running the same model optimized across various hardware devices. Accelerate your deep learning performance across use cases like: language + LLMs, computer vision, automatic speech recognition, and more.

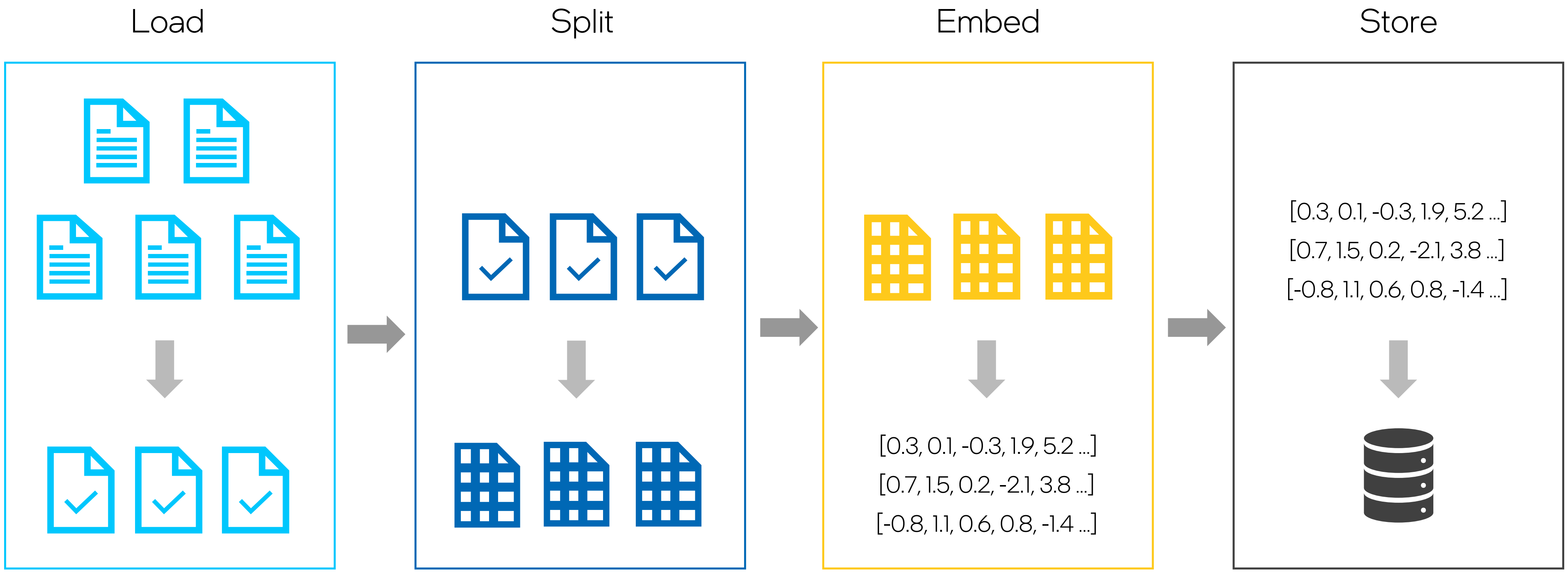
OpenVINO models can be run locally through `OpenVINOLLM` entitiy wrapped by LlamaIndex :

In the below line, we install the packages necessary for this demo:

```
%pip install llama-index-llms-openvino transformers huggingface_hub
```

Now that we're set up, let's play around:

Indexing



Load

Split

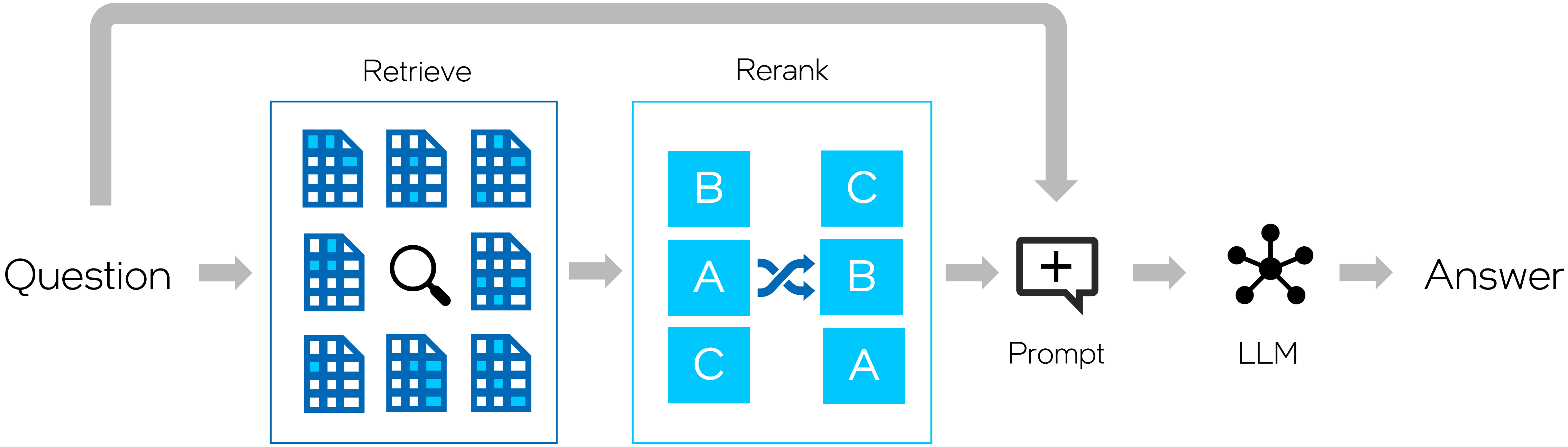
Embed

[0.3, 0.1, -0.3, 1.9, 5.2 ...]
[0.7, 1.5, 0.2, -2.1, 3.8 ...]
[-0.8, 1.1, 0.6, 0.8, -1.4 ...]

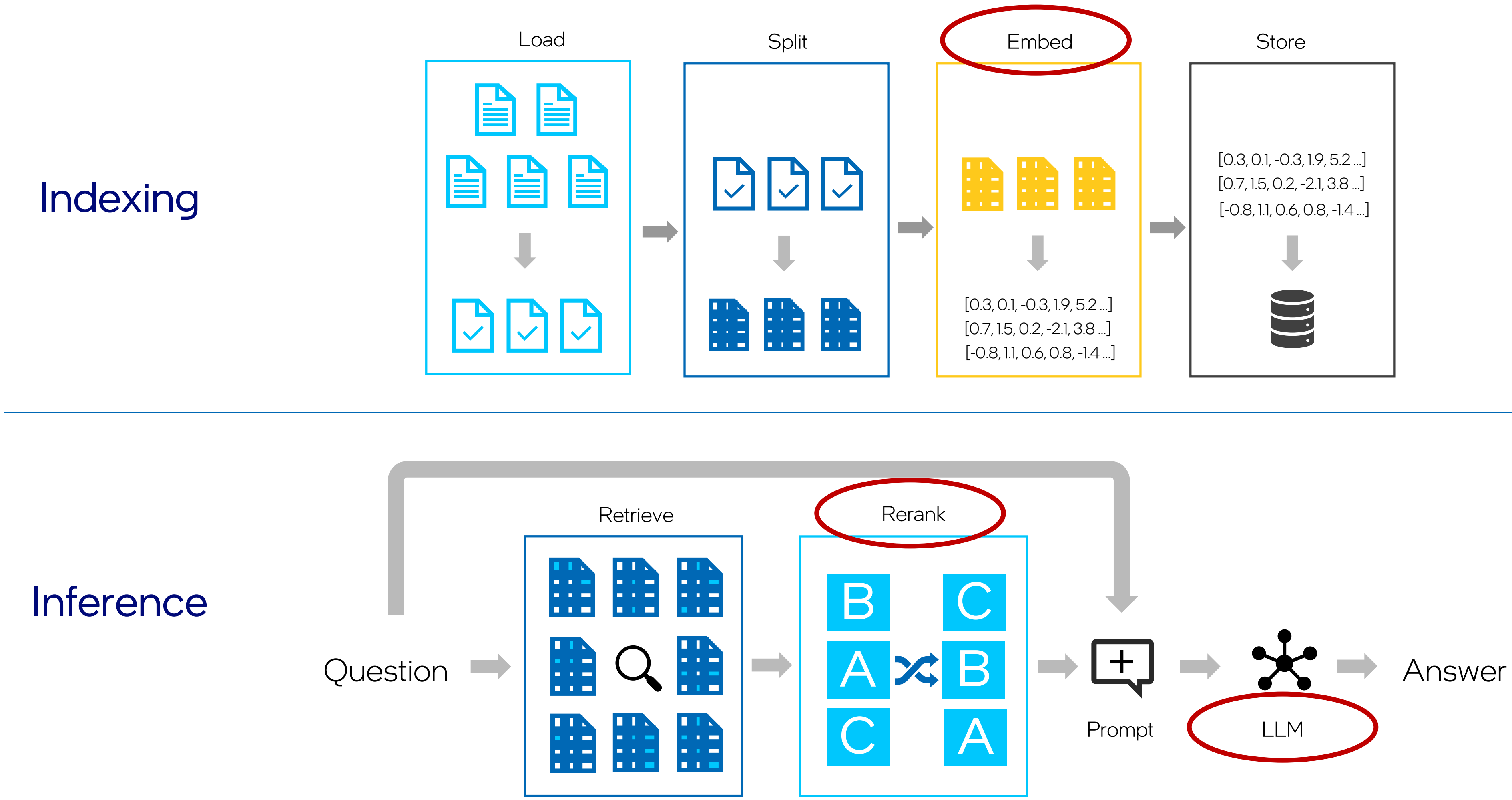
Store

[0.3, 0.1, -0.3, 1.9, 5.2 ...]
[0.7, 1.5, 0.2, -2.1, 3.8 ...]
[-0.8, 1.1, 0.6, 0.8, -1.4 ...]

Inference



Necessary models



Embeddings

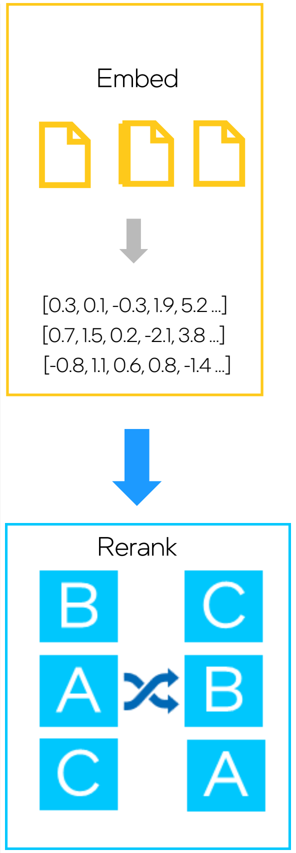
```
pip install llama-index-embeddings-openvino
```

```python
from llama_index.embeddings.huggingface_openvino import OpenVINOEmbedding
from optimum.intel import OVModelForFeatureExtraction
from transformers import AutoTokenizer

embedding_tokenizer = AutoTokenizer.from_pretrained(model_name)
embedding_tokenizer.save_pretrained(model_path)

embedding_model = OVModelForFeatureExtraction.from_pretrained(model_name, export=True,
                                                              compile=False)
embedding_model.save_pretrained(model_path)

embedding_model = OpenVINOEmbedding(str(model_path), device="CPU", embed_batch_size=1,
                                        model_kwargs={"dynamic_shapes": False})
```

Reranker

```
pip install llama-index-postprocessor-openvino-rerank
```

```
from optimum.intel import OVModelForSequenceClassification
from llama_index.postprocessor.openvino_rerank import OpenVINORerank
from transformers import AutoTokenizer

reranker_tokenizer = AutoTokenizer.from_pretrained(model_name)
reranker_tokenizer.save_pretrained(model_path)

reranker_model = OVModelForSequenceClassification.from_pretrained(model_name,
export=True, compile=False)
reranker_model.save_pretrained(model_path)

reranker_model = OpenVINORerank(model_id_or_path=str(model_path),
device="CPU", top_n=3)
```

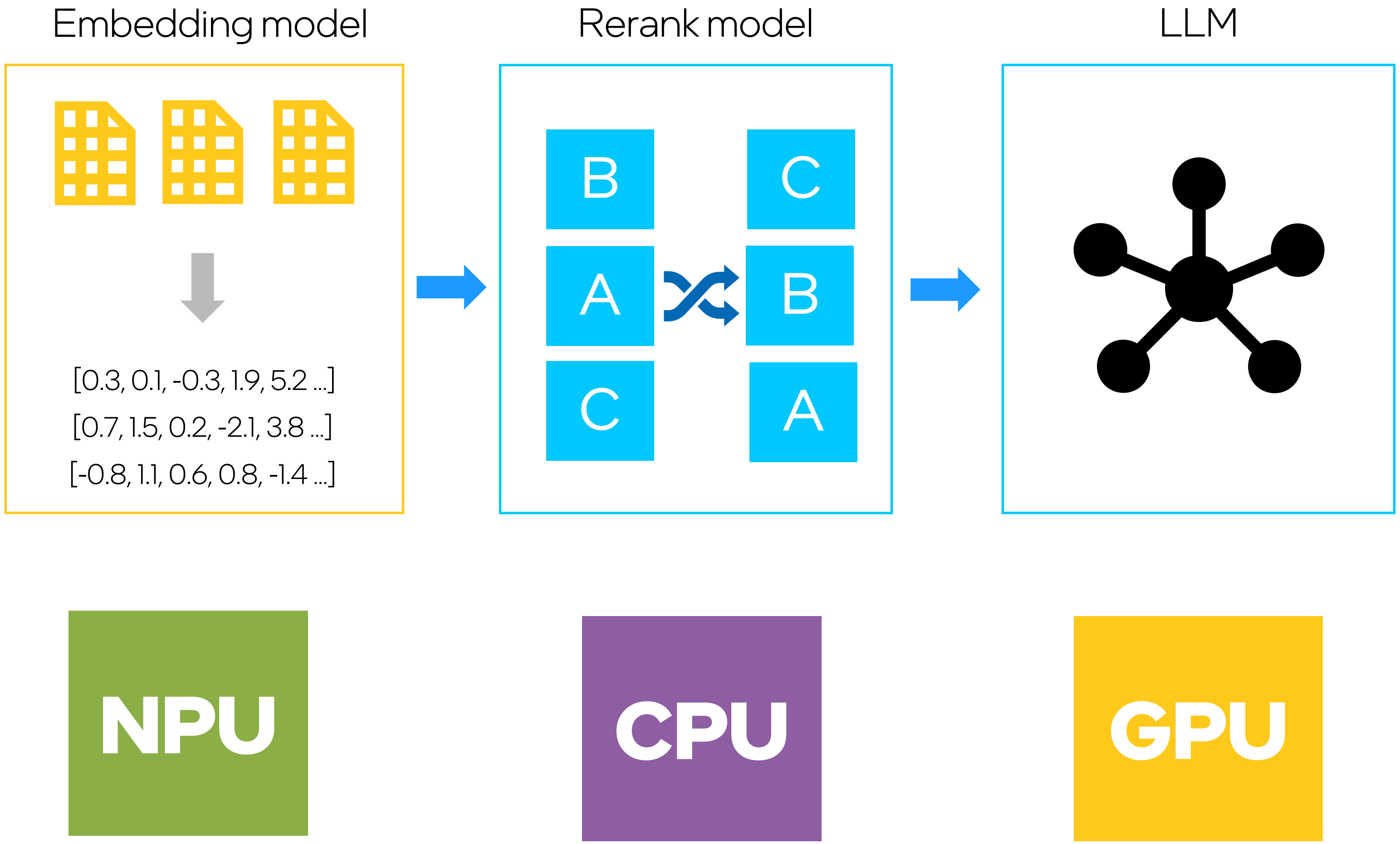


LLM

```
pip install llama-index-llms-openvino-genai
```

```python
from llama_index.llms.openvino_genai import OpenVINOGenAILLM
from optimum.intel import OVModelForCausalLM, OVWeightQuantizationConfig
from transformers import AutoTokenizer

chat_tokenizer = AutoTokenizer.from_pretrained(model_name)
export_tokenizer(chat_tokenizer, model_path)

quant_config = OVWeightQuantizationConfig(bits=4, sym=False, ratio=0.8, quant_method="awq", group_size=128,
                                          dataset="wikitext2")
chat_model = OVModelForCausalLM.from_pretrained(model_name, export=True, compile=False, trust_remote_code=True,
                                                quantization_config=quant_config, library_name="transformers")
chat_model.save_pretrained(model_path)

ov_config = {"PERFORMANCE_HINT": "LATENCY", "CACHE_DIR": ""}
llm = OpenVINOGenAILLM(model_path=str(model_path), device="GPU", config=ov_config)

llm.config.max_new_tokens = 1024
llm.config.temperature = 0.7
llm.config.top_k = 50
llm.config.top_p = 0.95
```

Deploy RAG Pipeline



The chat engine

```python
# limit chat history to 1024 tokens
memory = ChatMemoryBuffer.from_defaults(token_limit=2048)

documents = [Document(text=content, metadata={"file_name": file_path.name})]

# a splitter to divide document into chunks
splitter = LangchainNodeParser(RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=100))

dim = ov_embedding._model.request.outputs[0].get_partial_shape()[2].get_length()
# a memory database to store chunks
faiss_index = faiss.IndexFlatL2(dim)
vector_store = FaissVectorStore(faiss_index=faiss_index)
storage_context = StorageContext.from_defaults(vector_store=vector_store)

# set embedding model
Settings.embed_model = ov_embedding
index = VectorStoreIndex.from_documents(documents, storage_context, transformations=[splitter])

# create a RAG pipeline
ov_chat_engine = index.as_chat_engine(llm=ov_llm, chat_mode=ChatMode.CONTEXT, memory=memory,
                                      system_prompt=chatbot_config["system_configuration"],
                                      node_postprocessors=[ov_reranker])
```

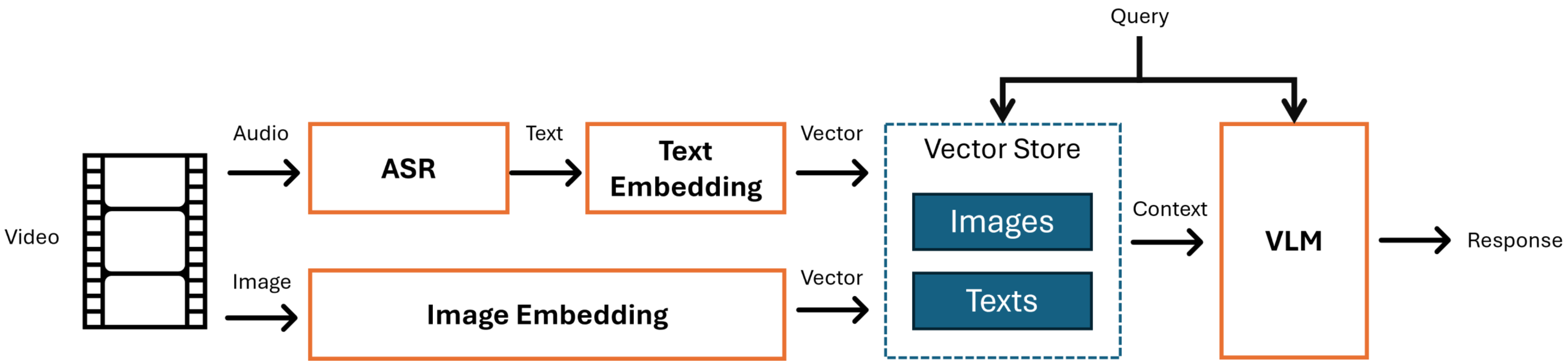
Inference

```
chat_streamer = ov_chat_engine.stream_chat(question).response_gen

answer = ""
for partial_text in chat_streamer:
      answer += partial_text
```

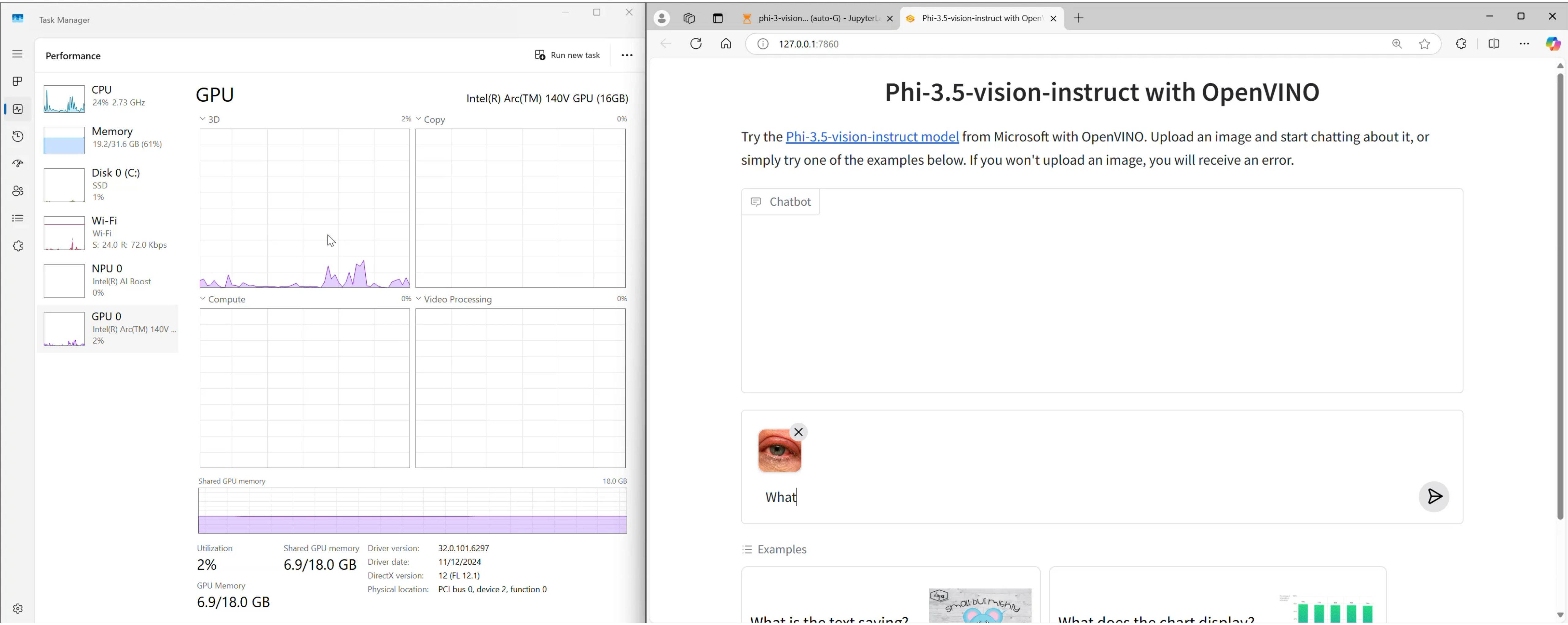
Let's run the demo!

Demo of LLM with RAG & Agent (AI assistant)

Multimodal RAG & VLM with OpenVINO for Videos



Convert, Optimize and Deploy VLM with OpenVINO



CPU
24% 2.73 GHz

Memory
19.2/31.6 GB (61%)

Disk 0 (C:)
SSD
1%

Wi-Fi
Wi-Fi
S: 24.0 R: 72.0 Kbps

NPU 0
Intel(R) AI Boost
0%

GPU 0
Intel(R) Arc(TM) 140V ...
2%

GPU
Intel(R) Arc(TM) 140V GPU (16GB)

Utilization
2%

Shared GPU memory
6.9/18.0 GB

GPU Memory
6.9/18.0 GB

Driver version: 32.0.101.6297
Driver date: 11/12/2024
DirectX version: 12 (FL 12.1)
Physical location: PCI bus 0, device 2, function 0

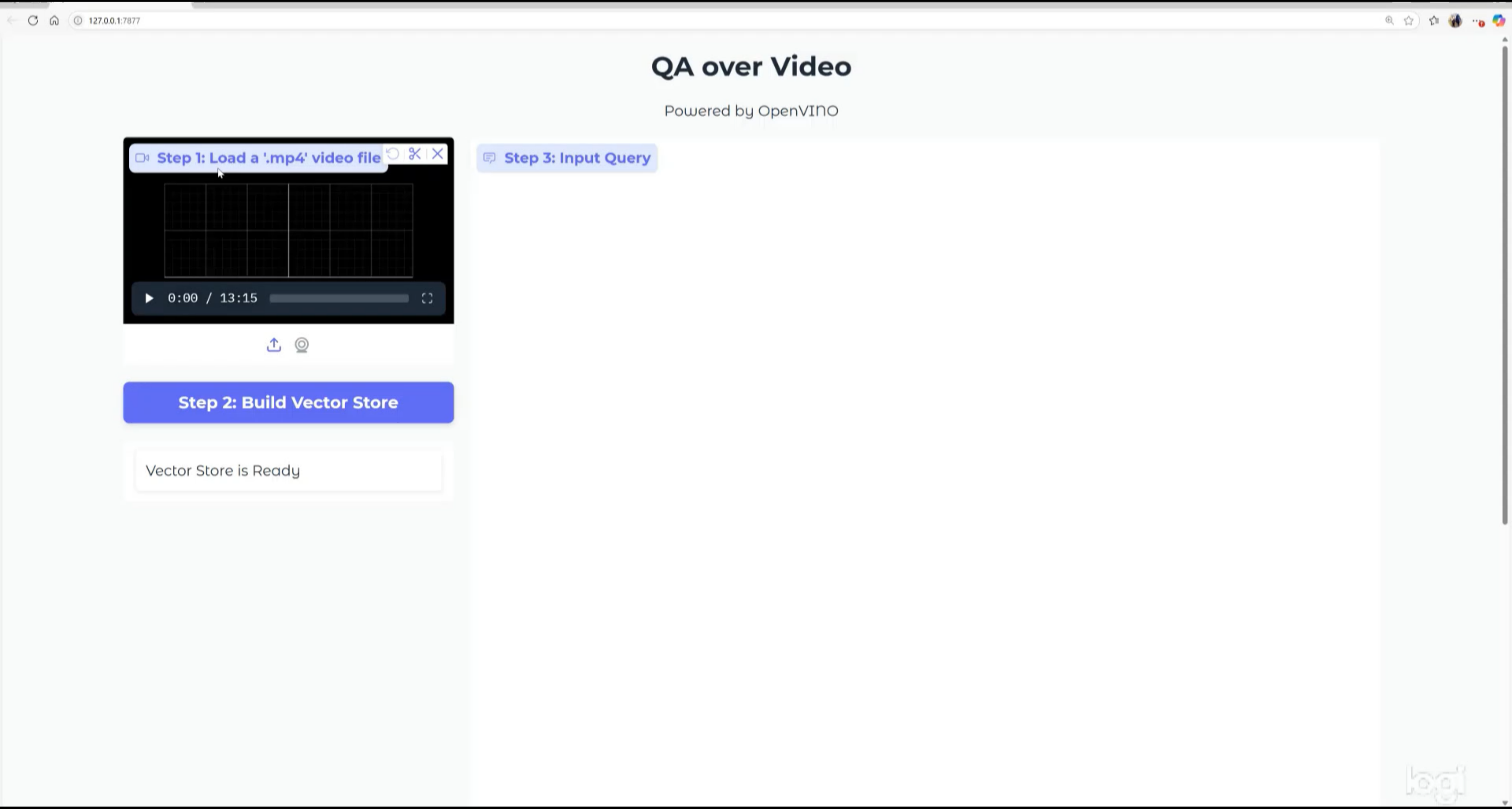
Phi-3.5-vision-instruct with OpenVINO

Try the Phi-3.5-vision-instruct model from Microsoft with OpenVINO. Upload an image and start chatting about it, or simply try one of the examples below. If you won't upload an image, you will receive an error.

What

What is the text saying?

What does the chart display?

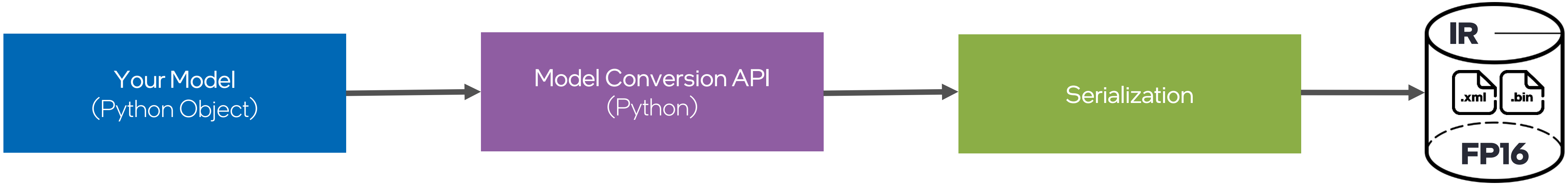


QA over Video

Powered by OpenVINO

Vector Store is Ready

Convert & Optimize Multimodal Embedding Model (BridgeTower) with OpenVINO™ Model Converter

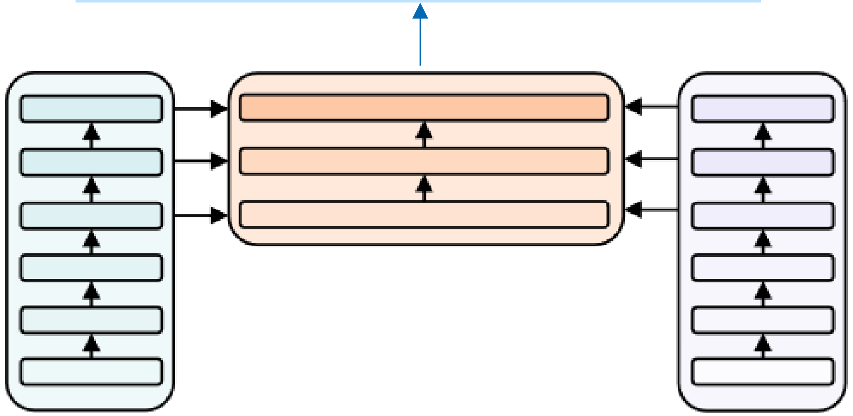


```
# Convert BridgeTower model to IR format and save
MODEL_NAME = "BridgeTower/bridgetower-large-itm-mlm-itc"
model = BridgeTowerForITC.from_pretrained(MODEL_NAME)
ov_model = ov.convert_model(model, example_input={**encoded_input})
ov.save_model(ov_model, "converted_model.xml")

# Convert BridgeTower text model to IR format and save
ov_textmodel_name = f"{models_dir}/custombridgetower_text_large_itc.xml"
text_model = BridgeTowerTextFeatureExtractor.from_pretrained(MODEL_NAME)
ov_textmodel = ov.convert_model(text_model, example_input={**text_encoding})
ov.save_model(ov_textmodel, ov_textmodel_name)
```

Convert & Optimize Multimodal Embedding Model (BridgeTower) with OpenVINO™ Model Converter

```
def restore_frompickle(filename:str)-> any:
    """ restore cookies from cache file"""
    with open(filename, 'rb') as handle:
        local_object = pickle.load(handle)
        return local_object


FILE_NAME=f"{models_dir}/input_example.pkl"
ENCODING_FILE_NAME =
f"{models_dir}/input_text_encoding.pkl‟

encoded_input = restore_frompickle(filename=FILE_NAME)
text_encoding =
restore_frompickle(filename=ENCODING_FILE_NAME)
```

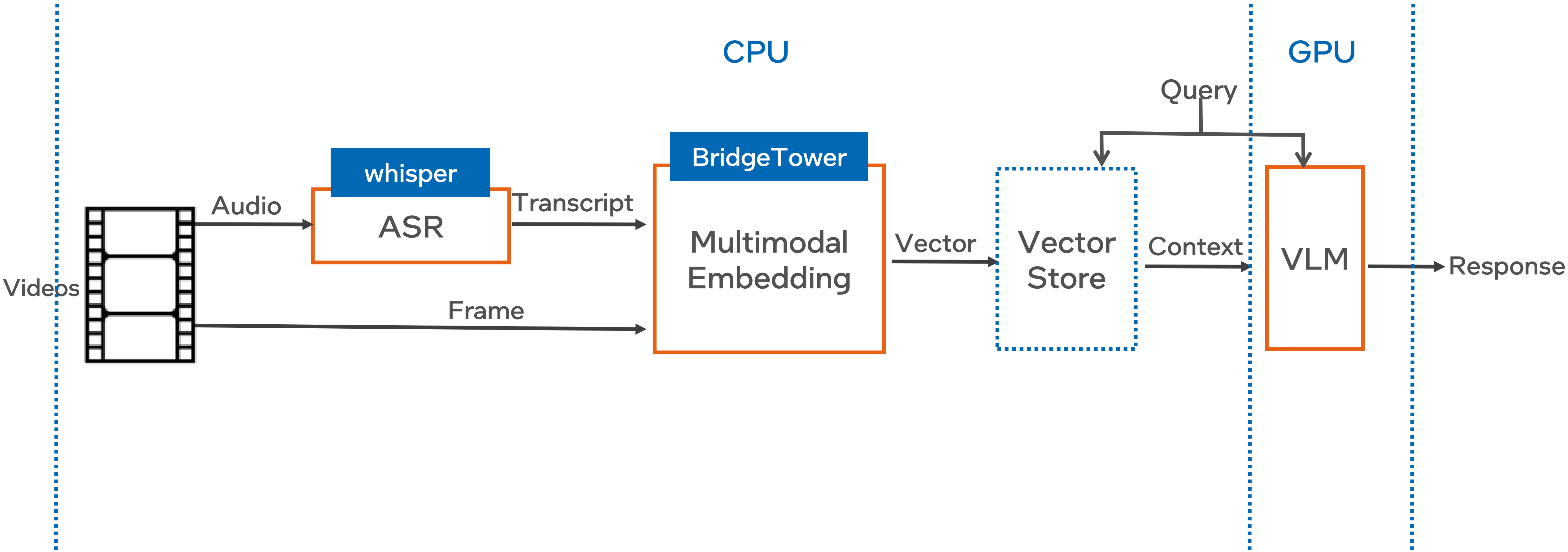
Cross modal feature / joint embeddings



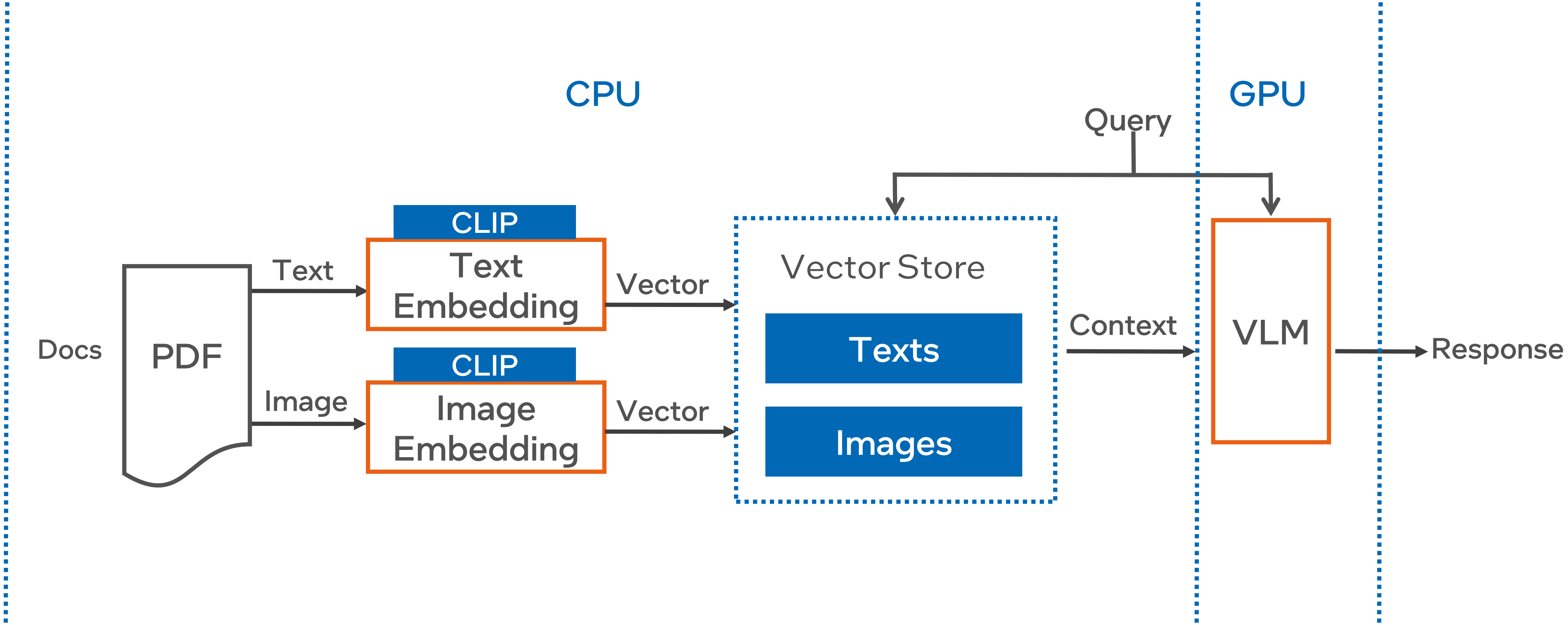
Text encoder

Image encoder

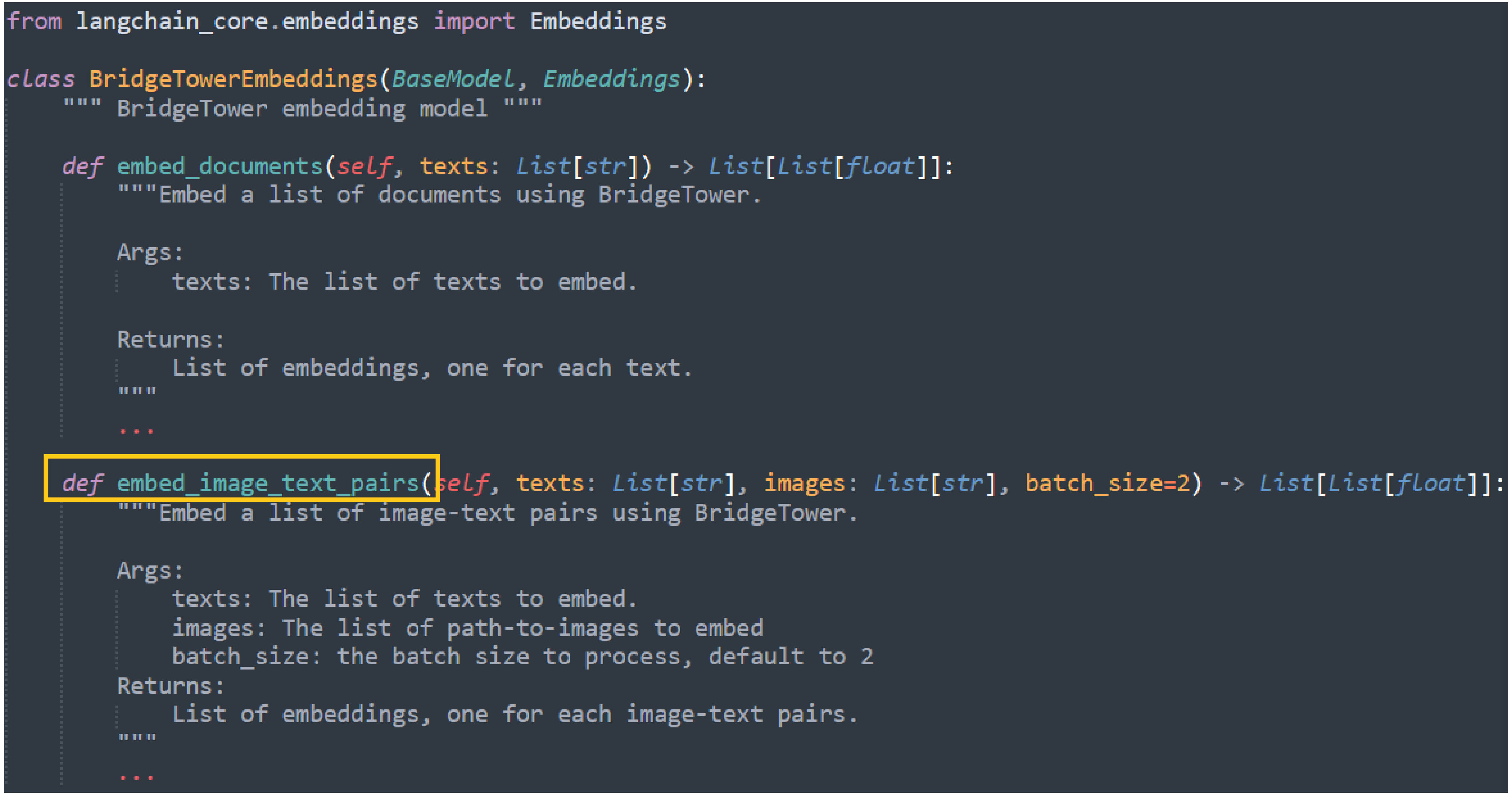
Multimodal RAG & VLM with OpenVINO for Videos



Multimodal RAG & VLM with OpenVINO for Text/Image Documents

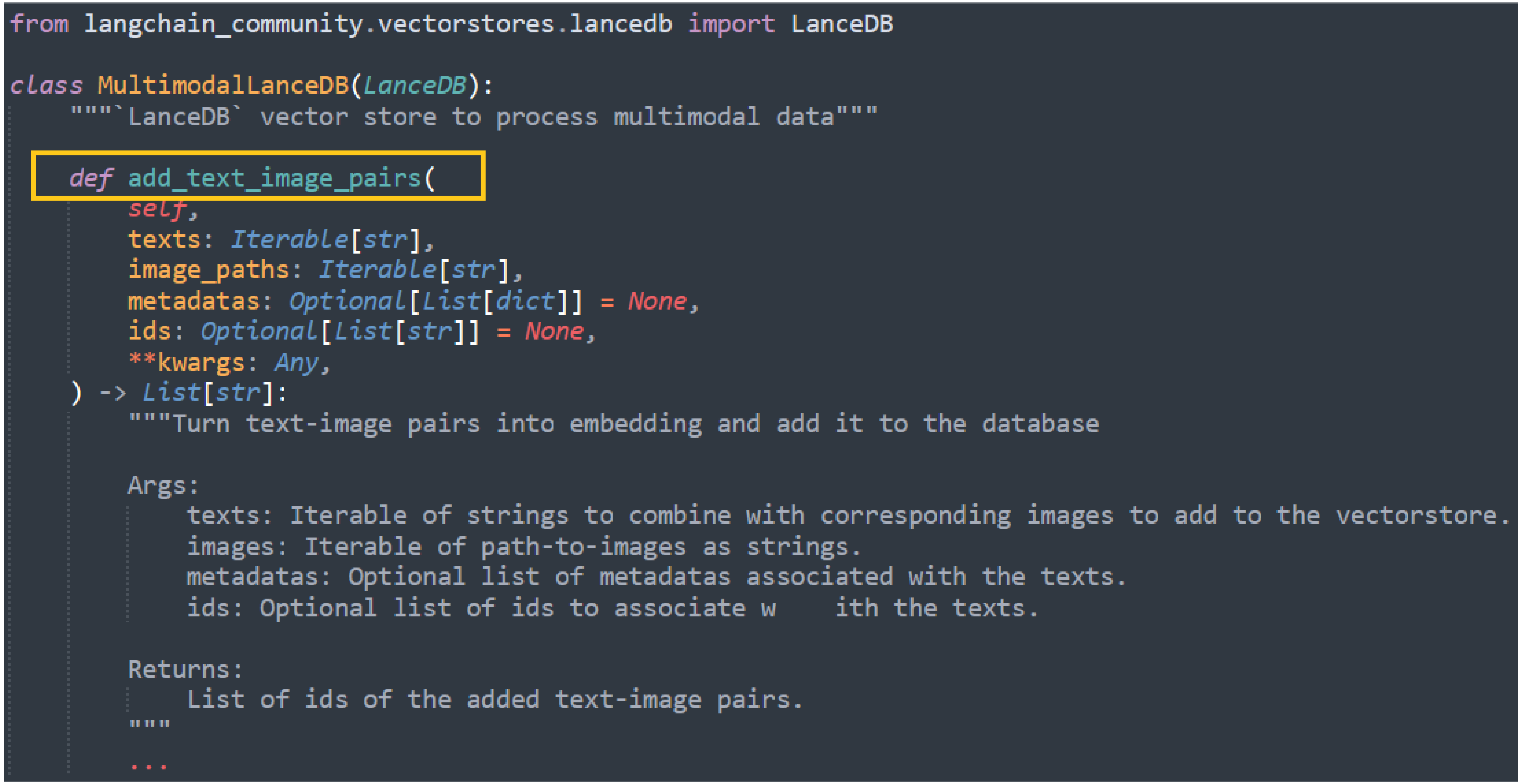


Multimodal RAG Implementations with Langchain



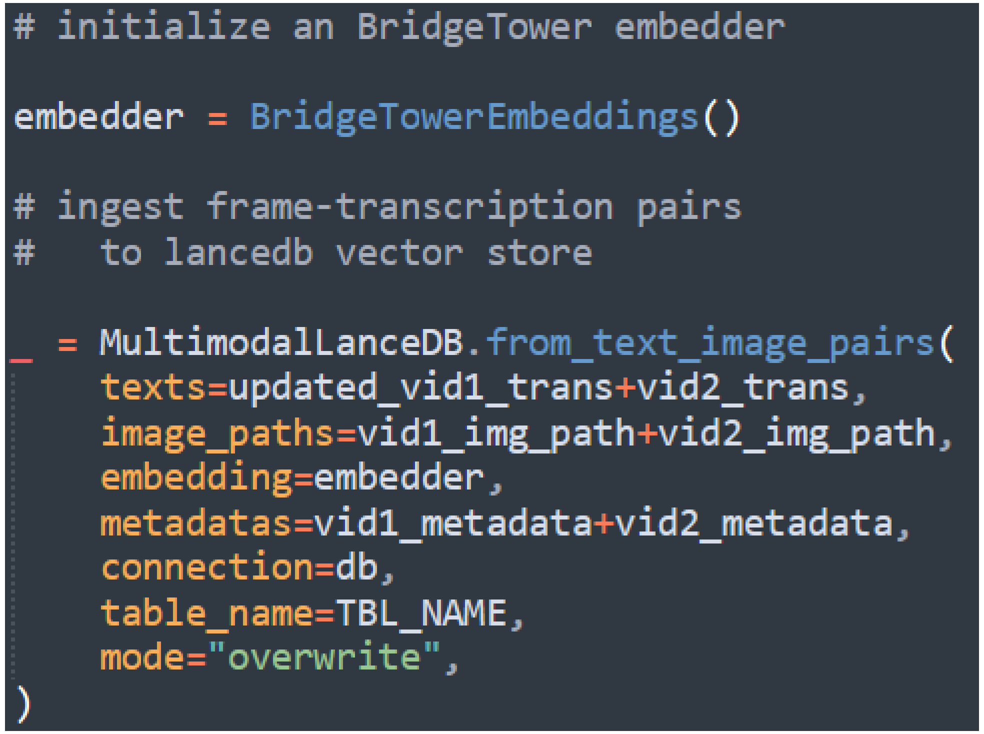
```python
from langchain_core.embeddings import Embeddings

class BridgeTowerEmbeddings(BaseModel, Embeddings):
    """ BridgeTower embedding model """

    def embed_documents(self, texts: List[str]) -> List[List[float]]:
        """Embed a list of documents using BridgeTower.

        Args:
            texts: The list of texts to embed.

        Returns:
            List of embeddings, one for each text.
        """
        ...

    def embed_image_text_pairs(self, texts: List[str], images: List[str], batch_size=2) -> List[List[float]]:
        """Embed a list of image-text pairs using BridgeTower.

        Args:
            texts: The list of texts to embed.
            images: The list of path-to-images to embed
            batch_size: the batch size to process, default to 2
        Returns:
            List of embeddings, one for each image-text pairs.
        """
        ...
```

Multimodal RAG Implementations with Langchain



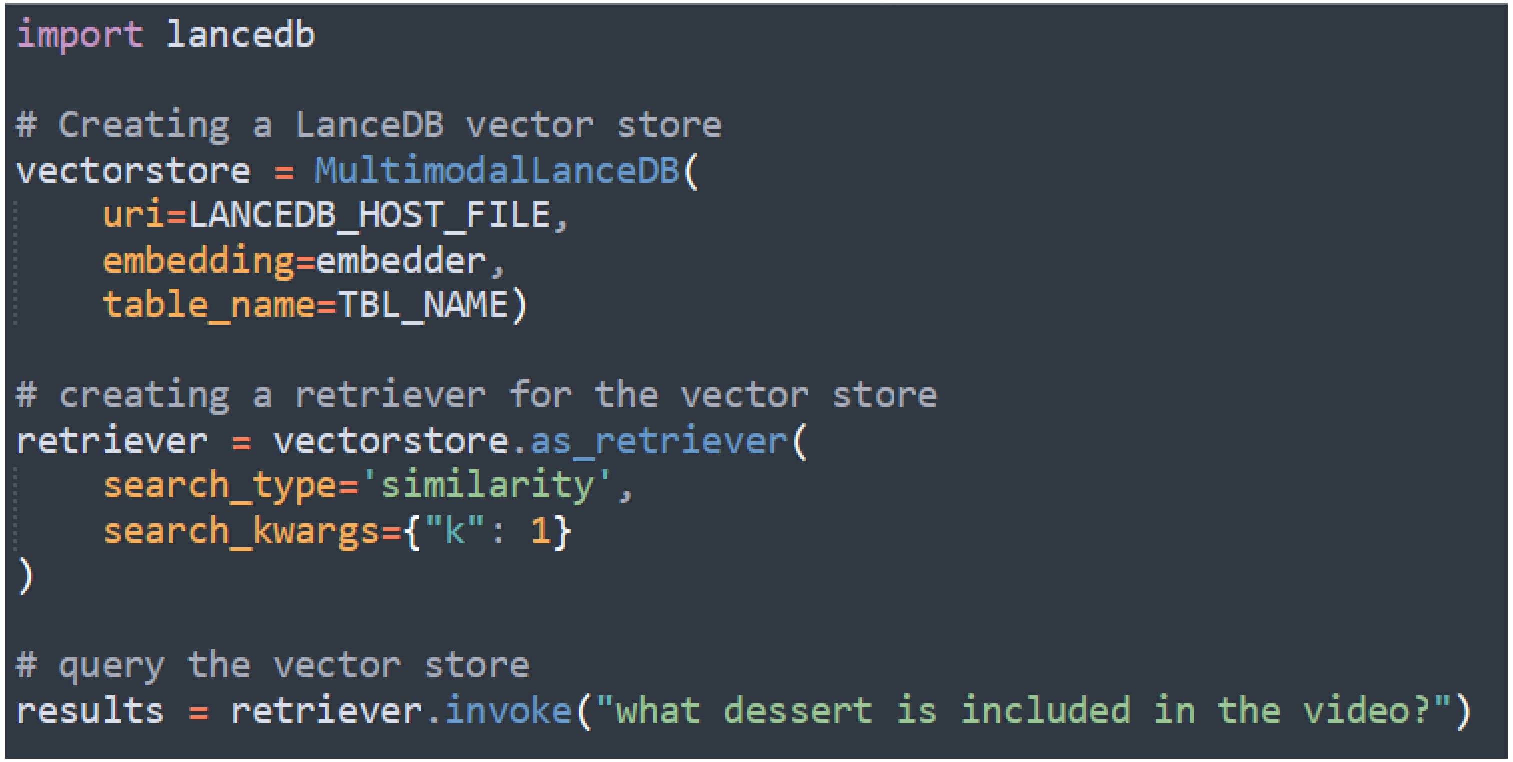
```
from langchain_community.vectorstores.lancedb import LanceDB

class MultimodalLanceDB(LanceDB):
    """`LanceDB` vector store to process multimodal data"""

    def add_text_image_pairs(
        self,
        texts: Iterable[str],
        image_paths: Iterable[str],
        metadatas: Optional[List[dict]] = None,
        ids: Optional[List[str]] = None,
        **kwargs: Any,
    ) -> List[str]:
        """Turn text-image pairs into embedding and add it to the database

        Args:
            texts: Iterable of strings to combine with corresponding images to add to the vectorstore.
            images: Iterable of path-to-images as strings.
            metadatas: Optional list of metadatas associated with the texts.
            ids: Optional list of ids to associate w    ith the texts.

        Returns:
            List of ids of the added text-image pairs.
        """
        ...
```

Multimodal RAG Implementations with Langchain



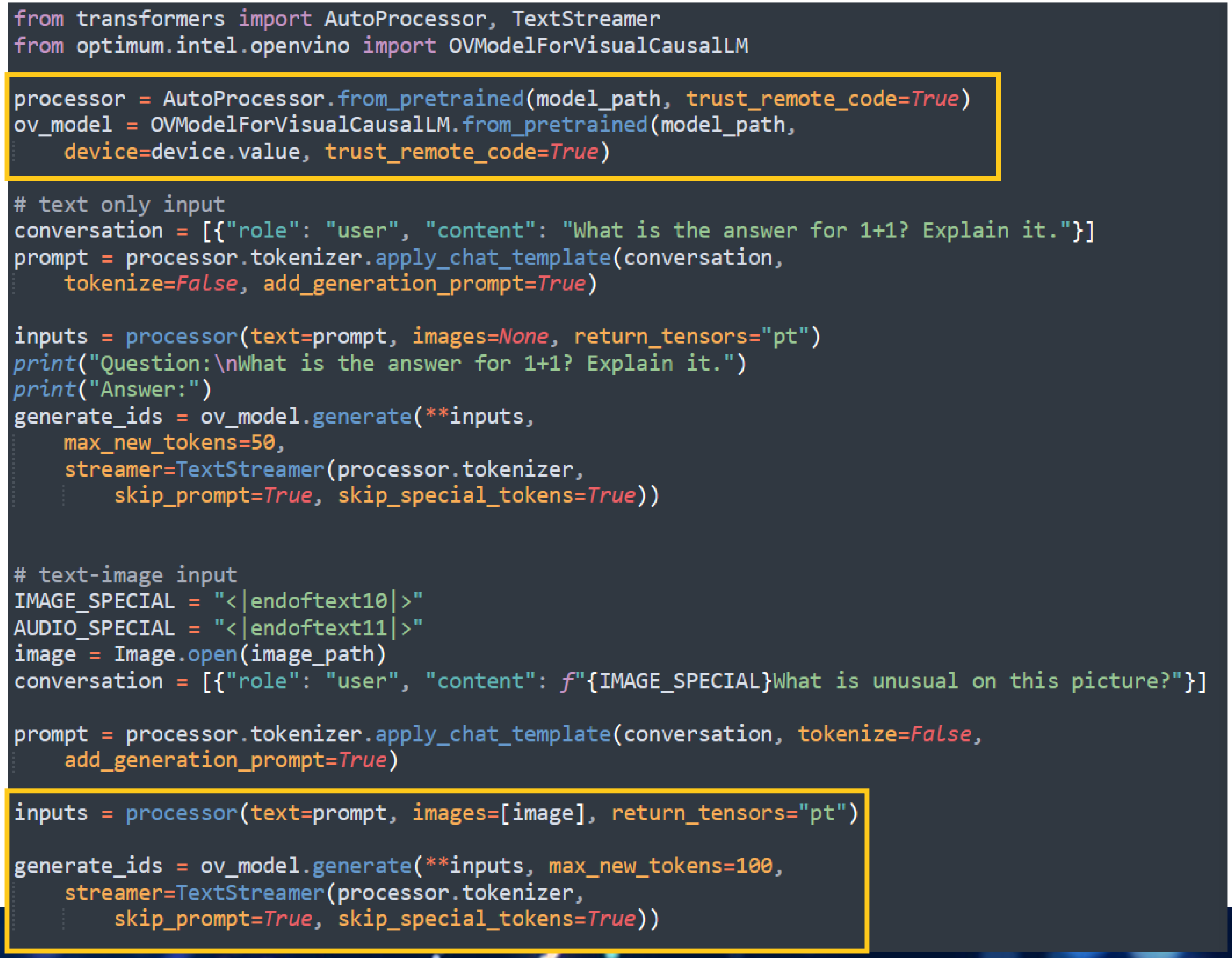
```
# initialize an BridgeTower embedder

embedder = BridgeTowerEmbeddings()

# ingest frame-transcription pairs
#   to lancedb vector store

_ = MultimodalLanceDB.from_text_image_pairs(
    texts=updated_vid1_trans+vid2_trans,
    image_paths=vid1_img_path+vid2_img_path,
    embedding=embedder,
    metadatas=vid1_metadata+vid2_metadata,
    connection=db,
    table_name=TBL_NAME,
    mode="overwrite",
)
```

Multimodal RAG Implementations with Langchain

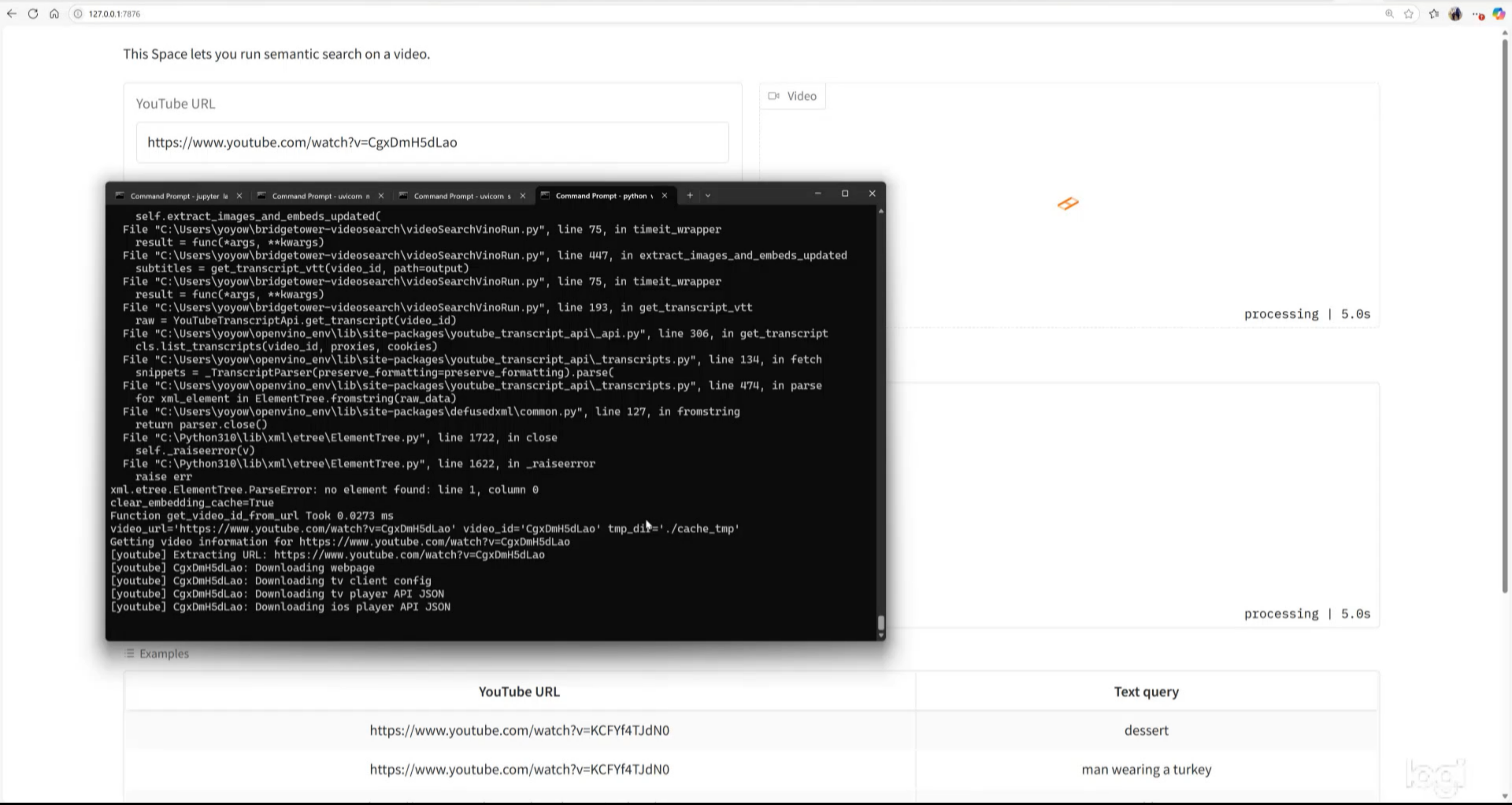


```python
import lancedb

# Creating a LanceDB vector store
vectorstore = MultimodalLanceDB(
    uri=LANCEDB_HOST_FILE,
    embedding=embedder,
    table_name=TBL_NAME)

# creating a retriever for the vector store
retriever = vectorstore.as_retriever(
    search_type='similarity',
    search_kwargs={"k": 1}
)

# query the vector store
results = retriever.invoke("what dessert is included in the video?")
```

VLM Pipeline



```python
from transformers import AutoProcessor, TextStreamer
from optimum.intel.openvino import OVModelForVisualCausalLM

processor = AutoProcessor.from_pretrained(model_path, trust_remote_code=True)
ov_model = OVModelForVisualCausalLM.from_pretrained(model_path,
    device=device.value, trust_remote_code=True)

# text only input
conversation = [{"role": "user", "content": "What is the answer for 1+1? Explain it."}]
prompt = processor.tokenizer.apply_chat_template(conversation,
    tokenize=False, add_generation_prompt=True)

inputs = processor(text=prompt, images=None, return_tensors="pt")
print("Question:\nWhat is the answer for 1+1? Explain it.")
print("Answer:")
generate_ids = ov_model.generate(**inputs,
    max_new_tokens=50,
    streamer=TextStreamer(processor.tokenizer,
        skip_prompt=True, skip_special_tokens=True))


# text-image input
IMAGE_SPECIAL = "<|endoftext10|>"
AUDIO_SPECIAL = "<|endoftext11|>"
image = Image.open(image_path)
conversation = [{"role": "user", "content": f"{IMAGE_SPECIAL}What is unusual on this picture?"}]

prompt = processor.tokenizer.apply_chat_template(conversation, tokenize=False,
    add_generation_prompt=True)

inputs = processor(text=prompt, images=[image], return_tensors="pt")

generate_ids = ov_model.generate(**inputs, max_new_tokens=100,
    streamer=TextStreamer(processor.tokenizer,
        skip_prompt=True, skip_special_tokens=True))
```



This Space lets you run semantic search on a video.

YouTube URL: https://www.youtube.com/watch?v=CgxDmH5dLao

Video
Command Prompt - jupyter la
Command Prompt - uvicorn n
Command Prompt - uvicorn s
Command Prompt - python v

```
    self.extract_images_and_embeds_updated(
  File "C:\Users\yoyow\bridgetower-videosearch\videoSearchVinoRun.py", line 75, in timeit_wrapper
    result = func(*args, **kwargs)
  File "C:\Users\yoyow\bridgetower-videosearch\videoSearchVinoRun.py", line 447, in extract_images_and_embeds_updated
    subtitles = get_transcript_vtt(video_id, path=output)
  File "C:\Users\yoyow\bridgetower-videosearch\videoSearchVinoRun.py", line 75, in timeit_wrapper
    result = func(*args, **kwargs)
  File "C:\Users\yoyow\bridgetower-videosearch\videoSearchVinoRun.py", line 193, in get_transcript_vtt
    raw = YouTubeTranscriptApi.get_transcript(video_id)
  File "C:\Users\yoyow\openvino_env\lib\site-packages\youtube_transcript_api\_api.py", line 306, in get_transcript
    cls.list_transcripts(video_id, proxies, cookies)
  File "C:\Users\yoyow\openvino_env\lib\site-packages\youtube_transcript_api\_transcripts.py", line 134, in fetch
    snippets = _TranscriptParser(preserve_formatting=preserve_formatting).parse(
  File "C:\Users\yoyow\openvino_env\lib\site-packages\youtube_transcript_api\_transcripts.py", line 474, in parse
    for xml_element in ElementTree.fromstring(raw_data)
  File "C:\Users\yoyow\openvino_env\lib\site-packages\defusedxml\common.py", line 127, in fromstring
    return parser.close()
  File "C:\Python310\lib\xml\etree\ElementTree.py", line 1722, in close
    self._raiseerror(v)
  File "C:\Python310\lib\xml\etree\ElementTree.py", line 1622, in _raiseerror
    raise err
xml.etree.ElementTree.ParseError: no element found: line 1, column 0
clear_embedding_cache=True
Function get_video_id_from_url Took 0.0273 ms
video_url='https://www.youtube.com/watch?v=CgxDmH5dLao' video_id='CgxDmH5dLao' tmp_dir='./cache_tmp'
Getting video information for https://www.youtube.com/watch?v=CgxDmH5dLao
[youtube] Extracting URL: https://www.youtube.com/watch?v=CgxDmH5dLao
[youtube] CgxDmH5dLao: Downloading webpage
[youtube] CgxDmH5dLao: Downloading tv client config
[youtube] CgxDmH5dLao: Downloading tv player API JSON
[youtube] CgxDmH5dLao: Downloading ios player API JSON
```

processing | 5.0s

processing | 5.0s

Examples

YouTube URL	Text query
https://www.youtube.com/watch?v=KCFYf4TJdN0	dessert
https://www.youtube.com/watch?v=KCFYf4TJdN0	man wearing a turkey

Conclusion

Building, Optimizing and Deploying Multimodal RAG Pipeline with OpenVINO™

Convert, optimize and deploy multimodal embedding models & VLMs in simple steps

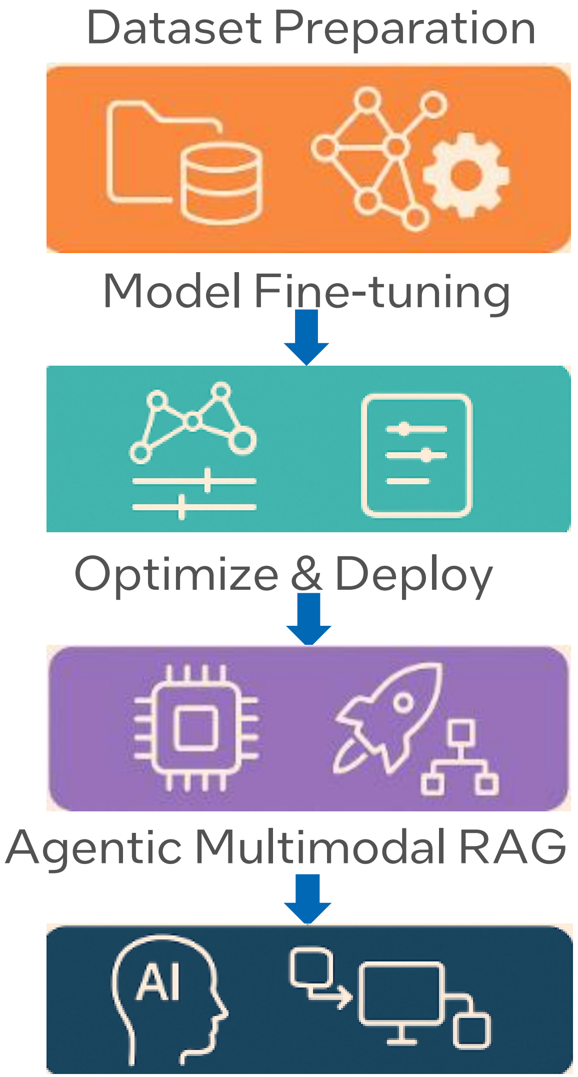
Optimization of embedding models & VLMs brings smaller binary size & reduces memory footprint

Build RAG easily with OpenVINO's integration with LangChain, LlamaIndex, *etc.*

More accurate and intuitive retrieval by combining visual and textual understanding, empowering cognitive AI

What's next?

What You Will Learn



Dataset Preparation

Model Fine-tuning

Optimize & Deploy

Agentic Multimodal RAG

- Module 1: Tool for accelerating your dataset preparation locally
- Module 2: How to fine-tune the multimodal embedding model and VLM
- Module 3: How to optimize and deploy the multimodal RAG pipeline
- Module 4: How to build Agentic multimodal RAG