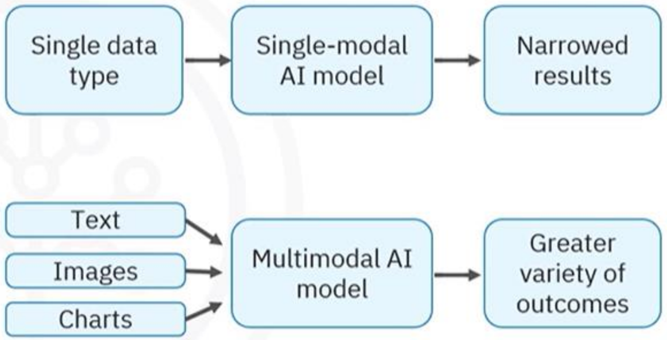
# Build Multimodal Generative AI Applications

## What is multimodal AI

- Artificial intelligence systems that:
- Process multiple types of data simultaneously
  - Text
  - Images
  - Audio
  - Video
- Closely mimics humans
  - Constant integration of information using different senses




## The evolution of AI

**Past**
- Specialized models excelling at one task
  - Text processing (BERT)
  - Image recognition (ResNet)
- No interaction between modalities

**Present**
- Integrated models handling multiple types of data

Examples:
- IBM Granite 3.2 Vision
- Meta's Llama 3.2 and Llama 4
- OpenAI's GPT-4.1

**Future**
- Unified, general-purpose AI models
- Seamless multimodal understanding

## Key functionalities of multimodal AI

**Text processing**
- Natural language understanding

**Computer vision**
- Image and video analysis

**Speech processing**
- Converts audio into text and extracts meaning

**Text-to-speech**
- Converts written text into natural-sounding speech

**Multimodal fusion**
- Combines data from multiple modalities to build a unified representation or response

## How multimodal AI works

**1. Input processing:**
- Separate processing of each data type

**2. Feature extraction:**
- Key features extracted from each modality

**3. Alignment:**
- All data types synchronized

**4. Fusion:**
- Combining information from all modalities

**5. Generation**
- Unified response unifying all modalities

# Industry leaders in multimodal AI

**IBM**
- Granite 3.2 Vision

**Open AI**
- GPT-4o
- CLIP
- DALL-E

**Meta**
- Llama 4
- Llama 3.2 Vision

**Google**
- Gemini
- Gemma

**Anthropic**
- Claude 3.7 Sonnet

# Challenges in multimodal AI

- Difficult to ensure synchronization of different types of data
- Imbalance in the data availability and quality across modalities
- Need for substantial computational resources
- Challenge in interpreting why models make certain decisions
- Inherited biases from training data
- Privacy concerns with sensitive information across different modalities

# Future trends in multimodal AI

**Unified models**

Shift to single models capable of handling all modalities from specialized systems

**Edge computing**

Running models directly on devices to improve **privacy**, **speed**, and **accessibility**

**Self-supervised learning techniques**

Reduces reliance on large labeled datasets by learning from unlabeled data

**Personalization**

Adapts to individual user preferences

**Ethical AI**

Emphasizes on fairness, transparency, and responsible use in multimodal systems

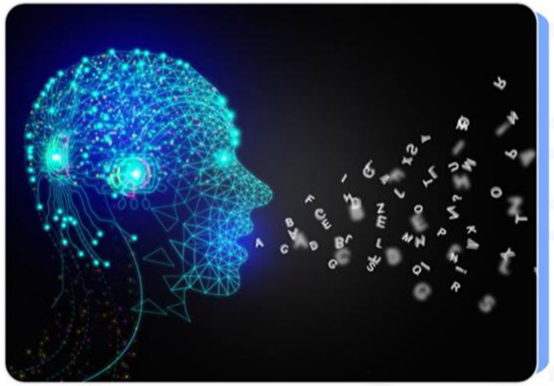
# Getting started with multimodal AI

**Join communities**
- Forums
- Workshops
- Hackathons

**Follow industry leaders**
- Stay updated on the latest developments

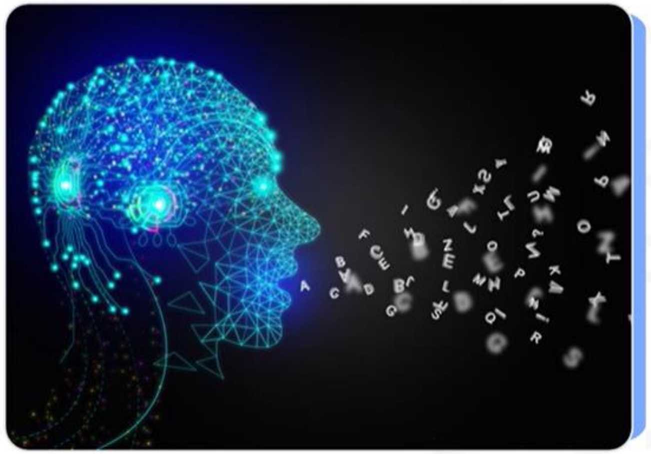
**Experiment with pre-built APIs**
- Understand how the systems work

## Text-to-speech (TTS)



Core of TTS → Linguistics + Speech synthesis

## Text-to-speech (TTS)



**Modern systems**

- Use advanced AI
- Generate human-like voices
- Work across multiple languages
- Adapt to diverse speaking styles

## Evolution of TTS

**Early days**

- Reliance on rule-based systems and robotic voices

**Breakthrough**

- Introduction of concatenative synthesis
- Prerecorded sounds to generate natural voice

**Modern era**

- Deep learning transformed TTS
- Google WaveNet and Tacotron generate speech from text

**Current state**

- End-to-end AI speech synthesis
- Intermediate representations not required

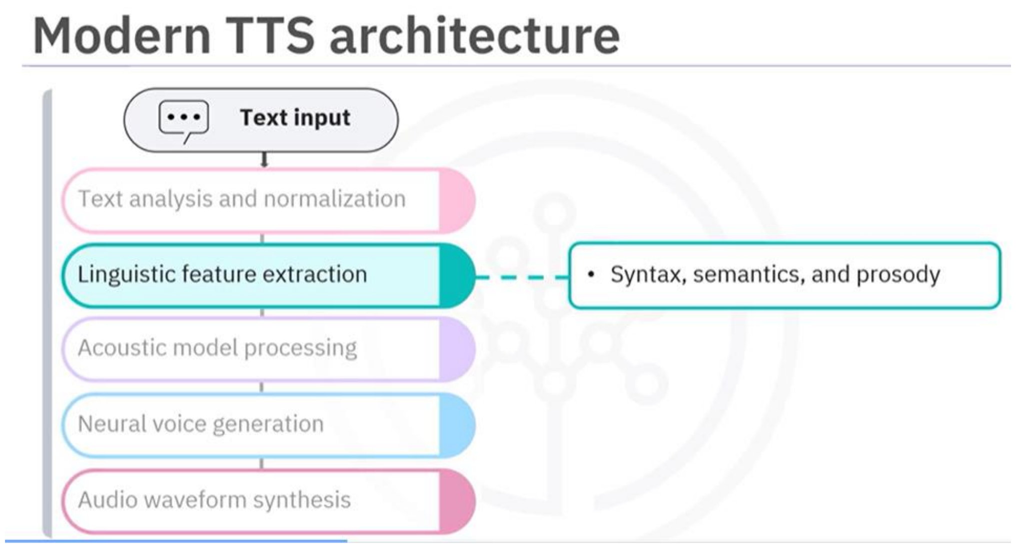
**Next generation**

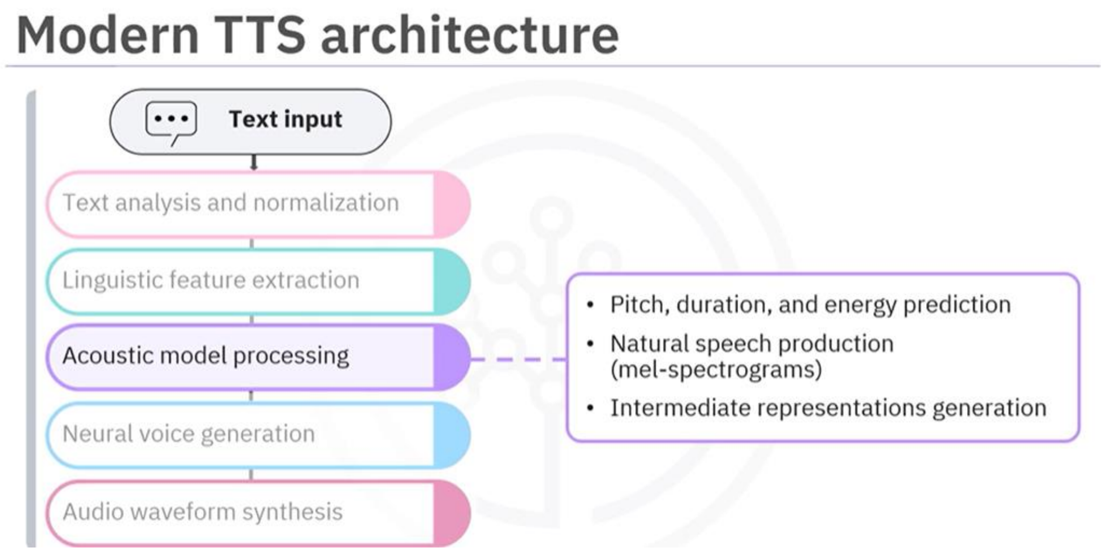
- Emotional and contextual awareness
- Human-like and engaging interactions

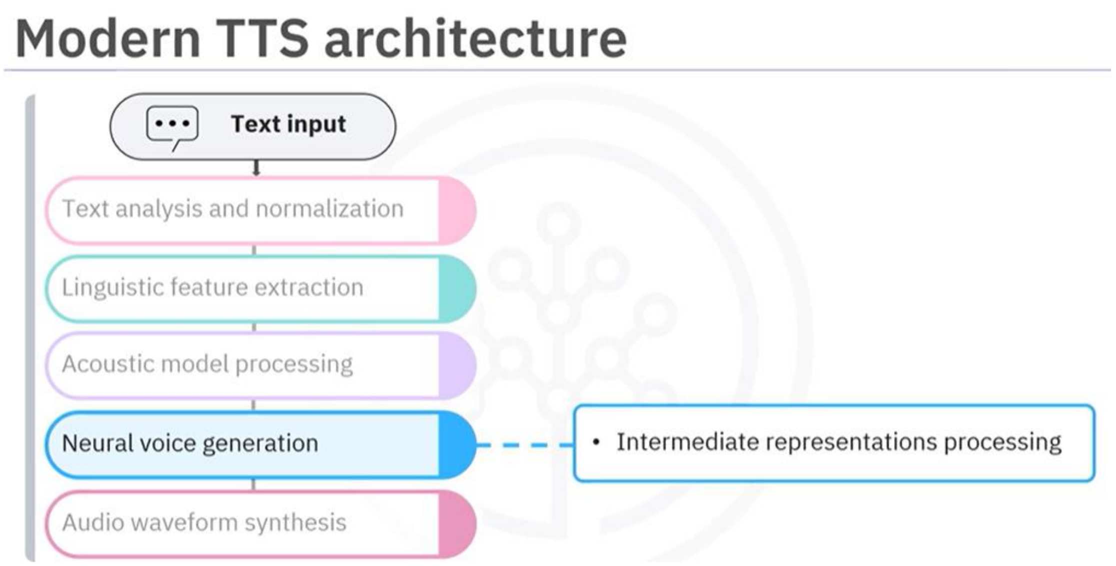
## Modern TTS architecture

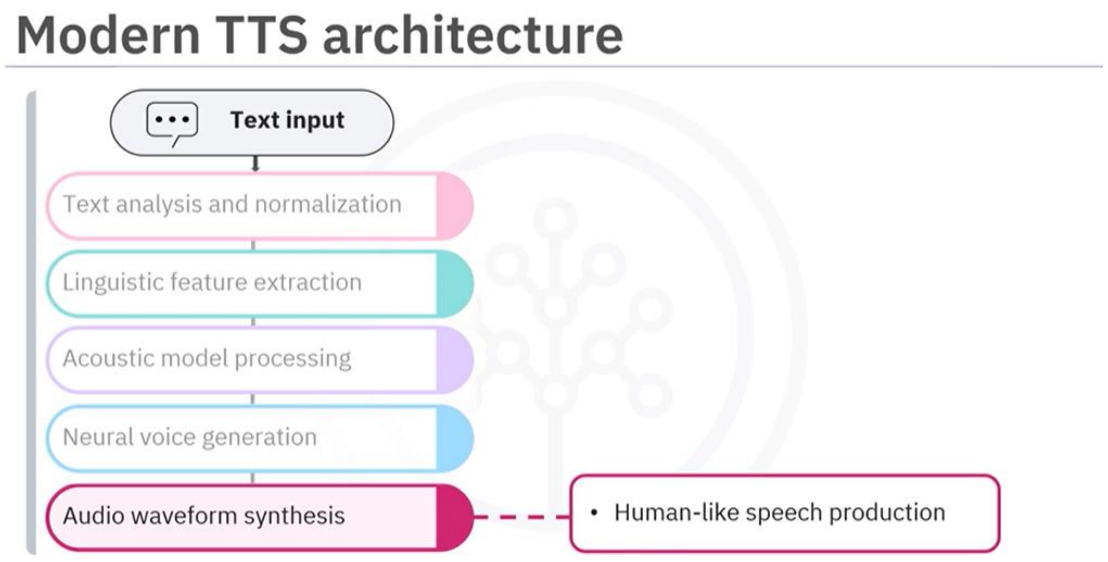
**Text input**

- Text analysis and normalization
  - Abbreviations expansion
  - Numbers to words conversion
  - Grapheme-to-phoneme conversion
- Linguistic feature extraction
- Acoustic model processing
- Neural voice generation
- Audio waveform synthesis

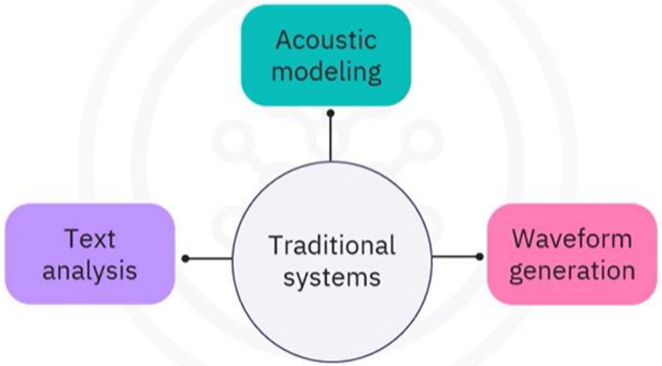

Modern TTS architecture
Text input
Text analysis and normalization
Linguistic feature extraction
Syntax, semantics, and prosody
Acoustic model processing
Neural voice generation
Audio waveform synthesis


Modern TTS architecture
Text input
Text analysis and normalization
Linguistic feature extraction
Acoustic model processing
Pitch, duration, and energy prediction
Natural speech production (mel-spectrograms)
Intermediate representations generation
Neural voice generation
Audio waveform synthesis


Modern TTS architecture
Text input
Text analysis and normalization
Linguistic feature extraction
Acoustic model processing
Neural voice generation
Intermediate representations processing
Audio waveform synthesis

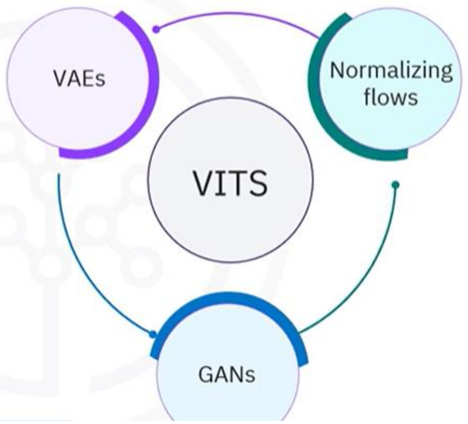

Modern TTS architecture
Text input
Text analysis and normalization
Linguistic feature extraction
Acoustic model processing
Neural voice generation
Audio waveform synthesis
Human-like speech production

## Technical deep dive: End-to-end systems

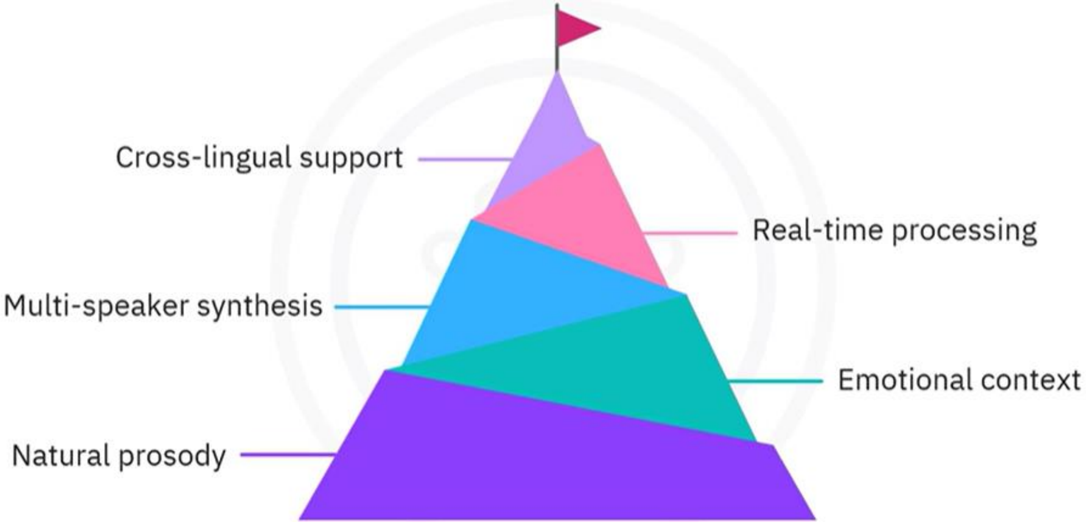



## Technical deep dive: End-to-end systems

- Directly maps text to audio
- Generates natural, high-quality speech




## Current challenges




## Future of TTS

## What is speech-to-text

Raw audio → Preprocessing/ Feature extraction → Acoustic model → Phonetic recognition model / Decoder → Text output

**Transforming spoken language into written text**

**Speech-to-text technologies:**

- Combine
  - Audio processing
  - Natural language understanding
- Recognize speech patterns and phonemes
- Work across multiple languages
- Adapt to different accents and speaking styles

Referred to as **automatic speech recognition (ASR)**

## The evolution of STT

**Early days**
- Template matching
- Rule-based systems

**Breakthrough**
- Hidden Markov models
- Statistical approaches

**Modern era**
- Deep learning
- Neural networks

**Current state**
- Self-supervised learning
- Transformer models

## Technical deep dive

```
    model(inputs.input_values).logits

    # Decode the logits to text
    predicted_ids =
torch.argmax(logits, dim=-1)
    transcription =
processor.batch_decode(predicted_ids)

    return transcription[0]

# Example usage
text = transcribe_audio("output.wav")
print(f"Transcription: {text}")
```

**End-to-end speech recognition system**

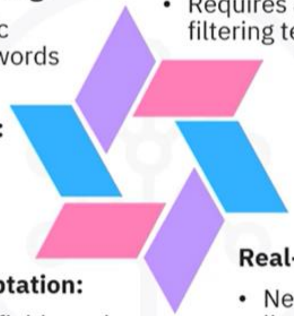
**Model:** Wav2Vec2

- Directly maps audio to text
- Pretrained with hours of audio data
- Fine-tuned for speech recognition
- Load the audio
- Preprocess it to the right format
- Pass it through the model
- Decode the output

## Challenges with respect to STT

**Context understanding:**
- Captures semantic meaning beyond words

**Background noise:**
- Requires advanced filtering techniques



**Low-resource languages:**
- Lack sufficient training data

**Speaker variability:**
- Must handle different voices and accents

**Domain adaptation:**
- Crucial for fields such as medicine or law

**Real-time processing:**
- Needs optimization for live applications

# Future and current trends in STT

**Contextual understanding**

- Capturing meaning beyond just words

**Multilingual models**

- Supporting multiple languages with a single

**Personalization**

- Adapting to individual speakers' voices

**Self-supervised learning**

- Training on unlabeled audio data

**Edge computing**

- Enabling private, low-latency on-device processing

## What is image captioning?

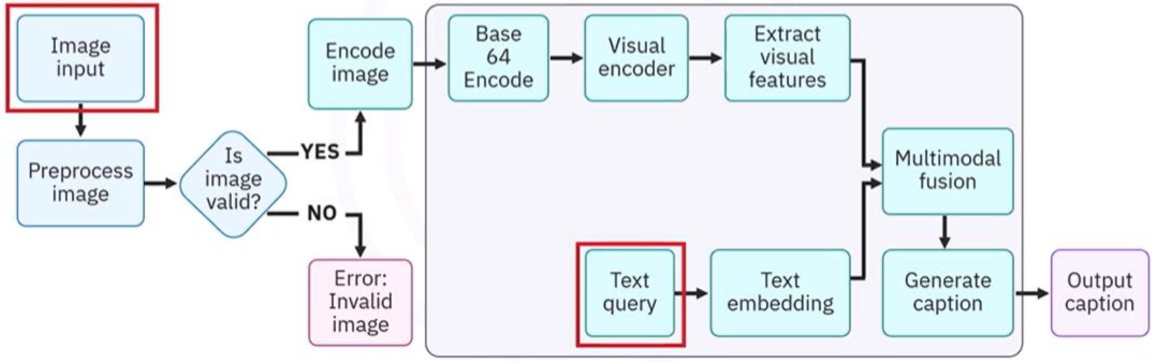

## How do multimodal models work?

Image captioning using LLM comprises three stages:


## Input processing

**Image captioning system accepts two input types:**

- Image that needs to be captioned
- Text query that guides captioning process




## Implementing image captioning model in Python


Implementing an image captioning model in Python involves combining the following:

- CNN to encode image
- RNN to generate caption

## Steps for implementing image captioning model in Python

Model generating caption:


## Recap

- Combining computer vision with natural language processing creates powerful tools for understanding visual content
- Image captioning process with an LLM comprises input processing, image validation and encoding, and multimodal LLM processing
- The core of the image captioning process uses visual encoders, text embedding, fusion layers, and language generation tools

## Recap

- To implement an image captioning system in Llama 4 Maverick model via IBM watsonx you have to:
  - Import libraries and authenticate access
  - Encode images and prepare prompts
  - Send combined image-text messages to the model
  - Extract descriptive text from the model's response

## What is MM-RAG?

**Multimodal**

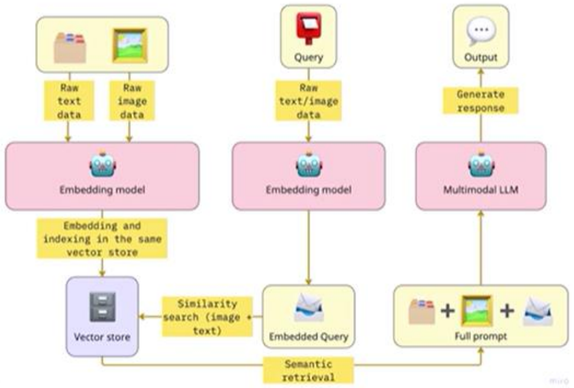
- Systems working with multiple data types or "modalities"

**Retrieval-augmented**

- Enhancing LLM responses by retrieving relevant information

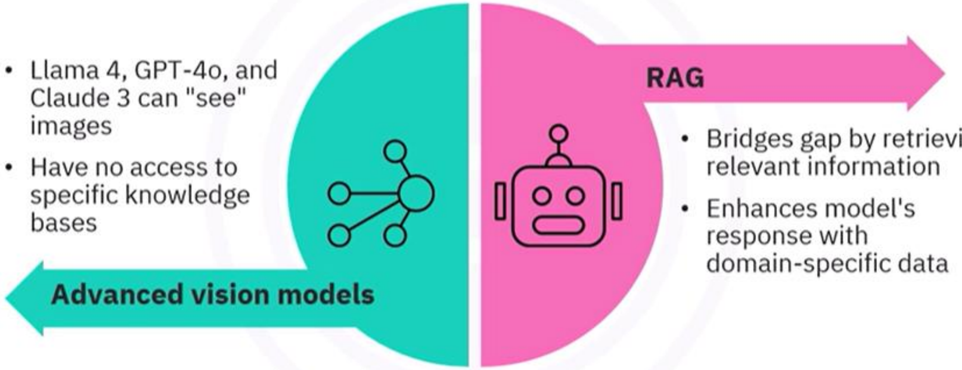
**Generation**

- Generate detailed, accurate responses combining modalities




## Advanced vision models vs. RAG

- Llama 4, GPT-4o, and Claude 3 can "see" images
- Have no access to specific knowledge bases




- Bridges gap by retrieving relevant information
- Enhances model's response with domain-specific data

## MM-RAG pattern

**Step 1:**

**Multimodal data retrieval**

- Use specialized retrievers
- Process text documents, images, audio recordings, and videos

**Step 2:**

**Contrastive learning**

- Train models that create related data
- Help the system connect images and text easily

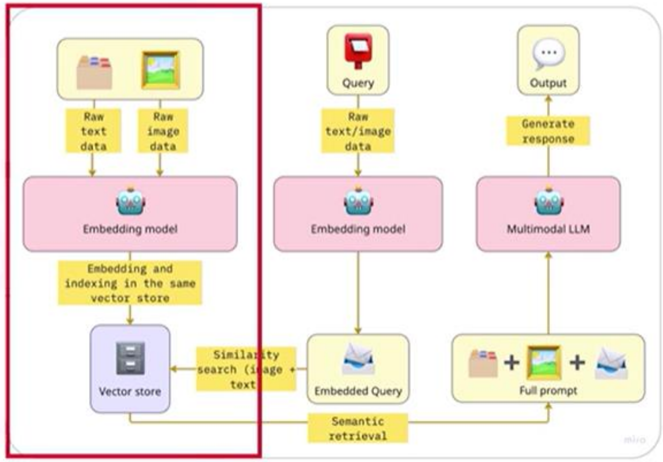
**Step 3:**

**Generative models informed by multimodal context**

## MM-RAG pipeline

**Data indexing**

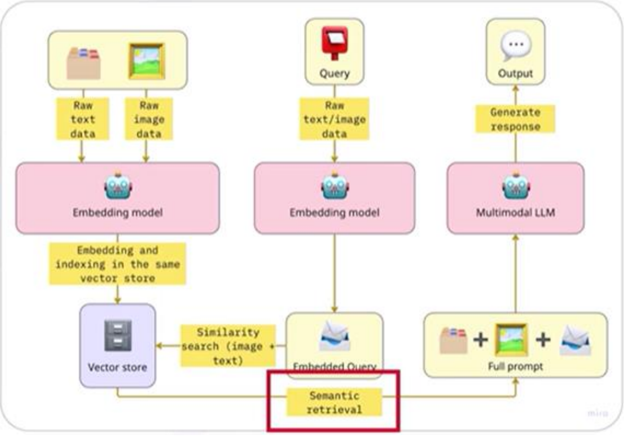
- Data types
  - Converted into embeddings
  - Indexed in a vector database




## MM-RAG pipeline

**Data retrieval**

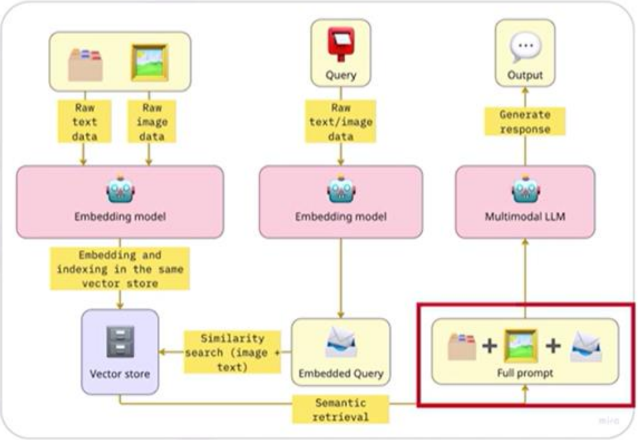
- User query (text, image, or both)
  - Received
  - Converted to embedding
  - Searched in vector database




## MM-RAG pipeline

**Augmentation**

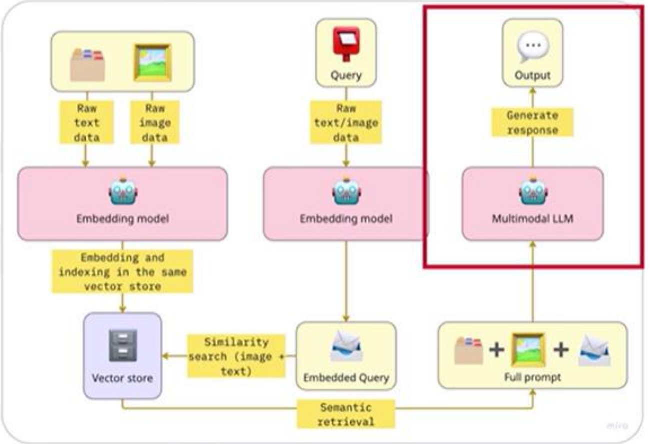
- Combine retrieved multimodal data with original user query



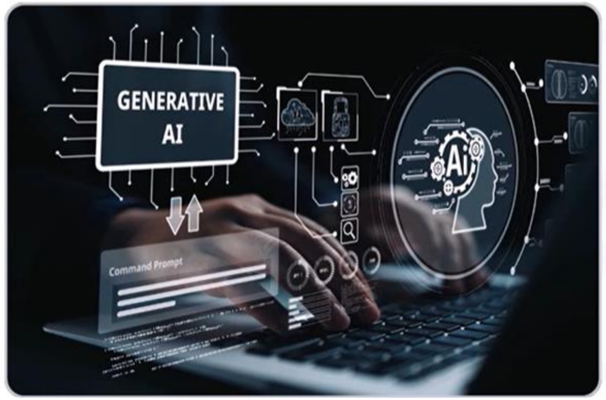

## MM-RAG pipeline

**Response generation**

- Input augmented query into multimodal generative model
- Produce a response that integrates information



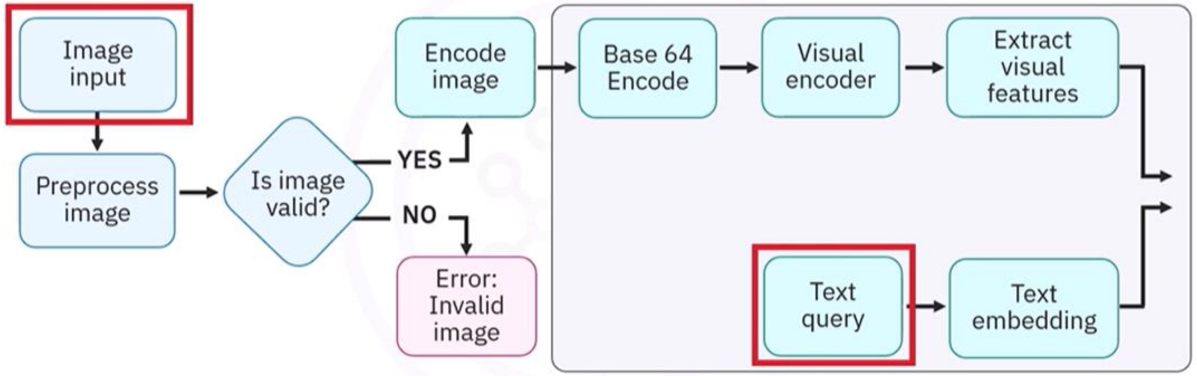
## What are multimodal chatbots and QA systems?




- Advanced AI applications
- Can process, understand, and generate responses
- Work with multiple data inputs:
  - Text
  - Images
  - Audio
  - Video
- Can **see**, **read**, and **understand** the world like humans

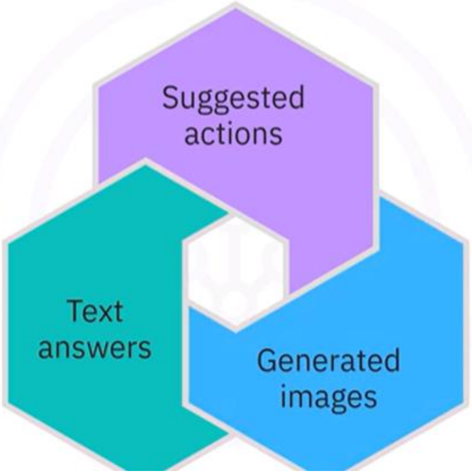
## Multimodal chatbots and QA systems: Architecture


## Multimodal chatbots and QA systems: Architecture




## Multimodal chatbots and QA systems: Architecture

**Contextual response generation**



## Implementation example: Image-based QA system

**Step 1** Setting up the environment and importing libraries

```
import requests
import base64
import os
from ibm_watsonx_ai import Credentials
from ibm_watsonx_ai import APIClient
from ibm_watsonx_ai.foundation_models import Model, ModelInference
from ibm_watsonx_ai.foundation_models.schema import TextChatParameters
from ibm_watsonx_ai.metanames import GenTextParamsMetaNames

from PIL import Image
```

## Implementation example: Image-based QA system

**Step 2** Initializing the model

```
credentials = Credentials(
    url = "https://us-south.ml.cloud.ibm.com",
    api_key = "<YOUR_API_KEY>"
)
client = APIClient(credentials)
```

```
model_id = "meta-llama/llama-3-2-90b-vision-instruct"
project_id = "your-project-id"
params = TextChatParameters()
```

**Step 2** Initializing the model

```
model = ModelInference(
    model_id=model_id,
    credentials=credentials,
    project_id=project_id,
    params=params
  )
```

## Implementation example: Image-based QA system

**Step 3** Preparing an image for processing

- Convert images into AI-readable format
  - Numerical or text-based representations
- Create two functions for image conversion

```
def prepare_image(image_path):
    """Convert an image to base64 encoding for the model"""
    with open(image_path, "rb") as image_file:
        encoded_image = base64.b64encode(image_file.read()).decode("utf-8")
    return encoded_image
```

## Implementation example: Image-based QA system

**Step 4** Creating the multimodal query function

```
def query_multimodal_model(encoded_image, user_question,
system_prompt=""):
    """
    Send a multimodal query to the model and get a response
    """
    messages = [
        {
            "role": "user",
            "content": [
                {
```

**Step 4** Creating the multimodal query

```
# Send the request to the model
response = model.chat(messages=messages)

# Return the model's response
return response['choices'][0]['message']['content']
```

## Implementation example: Image-based QA system

**Step 5** Using the multimodal QA function

- Prepare an image using encoding functions (Step 3)

```
# Example usage
image = prepare_image("sample_image.jpg")
```

## Implementation example: Image-based QA system

**Step 5** Using the multimodal QA function

- Further shape how the model approaches the question

```
# For a more specific domain, we can add a system prompt
system_prompt = """You are an expert assistant that helps analyze images.
Please provide detailed observations and answer the user's question
based on what you see in the image.
"""
```

## Implementation example: Image-based QA system

**Step 5** Using the multimodal QA function

- Craft a question about the image

```
question = "What can you see in this image? Is there anything unusual?"
```

## Implementation example: Image-based QA system

**Step 5** Using the multimodal QA function

- Call the function with components, process the input, and return

```
response = query_multimodal_model(image, question, system_prompt)
print("Model response:", response)
```

**Result:** Contextually relevant answer blending visual and textual query