

AI Guild | GenAI Practicum

Diving into Multimodal
Large Language Models

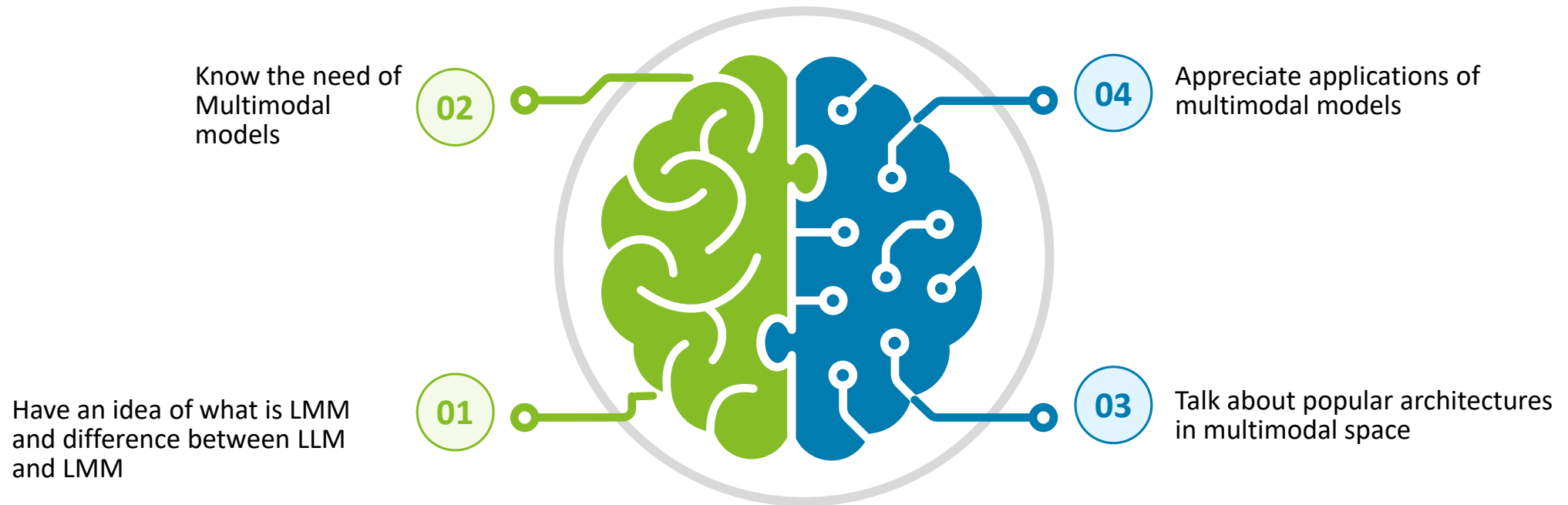
Topic



- 01** Introduction to Multimodal LLMs
- 02** LLM aided Visual reasoning
- 03** Diffusion Models, CLIP, BLI
- 04** AI Ethics

Learning objectives

By the end of session, you should be able to



Prerequisites

- Completion of AI Academy individual tracks or Pathway 0

Section 0a – LangChain Catch up

Callbacks system allows you to hook into the various stages of your LLM application. This is useful for logging, monitoring, streaming, and other tasks. You can subscribe to these events by using the callbacks argument available throughout the API. The callbacks argument is available on most objects throughout the API (Chains, Models, Tools, Agents, etc.) in two different forms:

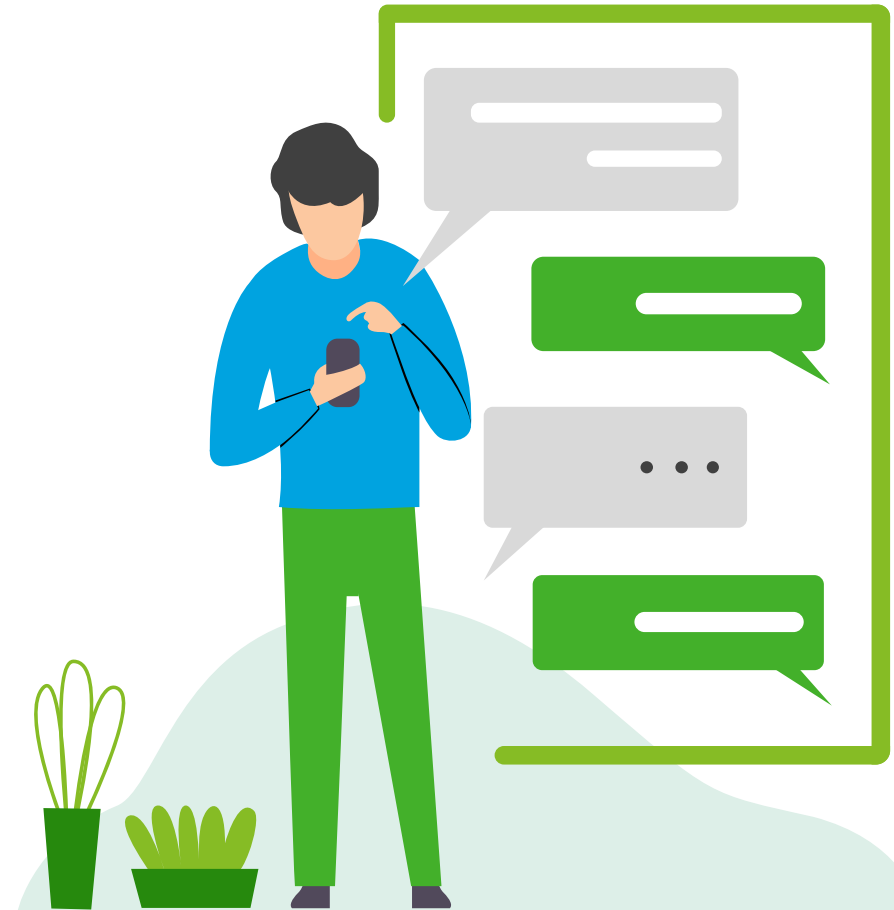
Constructor callbacks

- In the constructor, e.g. `LLMChain (callbacks=[handler], tags=['a-tag'])`, which will be used for all calls made on that object, and will be scoped to that object only, e.g. if you pass a handler to the `LLMChain` constructor, will not be used by the Model attached to that chain..
- For use cases such as logging, monitoring, etc., which are not specific to a single request, but rather to the entire chain. For example, if you want to log all the requests made to an `LLMChain`, you would pass a handler to the constructor.

Request callbacks

- In the `run()/apply()` methods used for issuing a request, e.g. `chain.run (input, callbacks=[handler])`, which will be used for that specific request only, and all sub-requests that it contains (e.g. a call to an `LLMChain` triggers a call to a Model, which uses the same handler passed in the `call()` method)
- For use cases such as streaming, where you want to stream the output of a single request to a specific websocket connection, or other similar use cases. For example, if you want to stream the output of a single request to a websocket, you would pass a handler to the `call()` method

The verbose argument is available on most objects throughout the API (Chains, Models, Tools, Agents, etc.) as a constructor argument, e.g. `LLMChain(verbose=True)`, and it is equivalent to passing a `ConsoleCallbackHandler` to the `callbacks` argument of that object and all child objects. This is useful for debugging, as it will log all events to the console.



LangChain Expression Language



LangChain Expression Language, or LCEL, is a declarative way to easily compose chains together. LCEL was designed from day 1 to support putting prototypes in production, with no code changes, from the simplest “prompt + LLM” chain to the most complex chains (we’ve seen folks successfully run LCEL chains with 100s of steps in production). To highlight a few of the reasons you might want to use LCEL:

Streaming support

When you build your chains with LCEL you get the best possible time-to-first-token (time elapsed until the first chunk of output comes out). For some chains this means e.g., we stream tokens straight from an LLM to a streaming output parser, and you get back parsed, incremental chunks of output at the same rate as the LLM provider outputs the raw tokens.

Async support

Any chain built with LCEL can be called both with the synchronous API (e.g., in your Jupyter notebook while prototyping) as well as with the asynchronous API (e.g., in a LangServe server). This enables using the same code for prototypes and in production, with great performance, and the ability to handle many concurrent requests in the same server.

Optimized parallel execution

Whenever your LCEL chains have steps that can be executed in parallel (eg if you fetch documents from multiple retrievers) we automatically do it, both in the sync and the async interfaces, for the smallest possible latency.

Seamless LangSmith tracing integration: As your chains get more and more complex, it becomes increasingly important to understand what exactly is happening at every step. With LCEL, all steps are automatically logged to LangSmith for maximum observability and debuggability.

Seamless LangServe deployment integration: Any chain created with LCEL can be easily deployed using LangServe.



Retries and fallbacks

Configure retries and fallbacks for any part of your LCEL chain. This is a great way to make your chains more reliable at scale. We're currently working on adding streaming support for retries/fallbacks, so you can get the added reliability without any latency cost.



Access intermediate results

For more complex chains it's often very useful to access the results of intermediate steps even before the final output is produced. This can be used let end-users know something is happening, or even just to debug your chain. You can stream intermediate results, and it's available on every LangServe server.



Input and output schemas

Input and output schemas give every LCEL chain Pydantic and JSONSchema schemas inferred from the structure of your chain. This can be used for validation of inputs and outputs, and is an integral part of LangServe.

LangChain Lab 8 – A Simple RAG Chatbot with Memory

Section 0b – RAG in the Real World

Be prepared to have a lot of moving pieces in your solution. Be prepared for the tweaking and tuning needed over the life of the solution.



Simple RAG Use Case



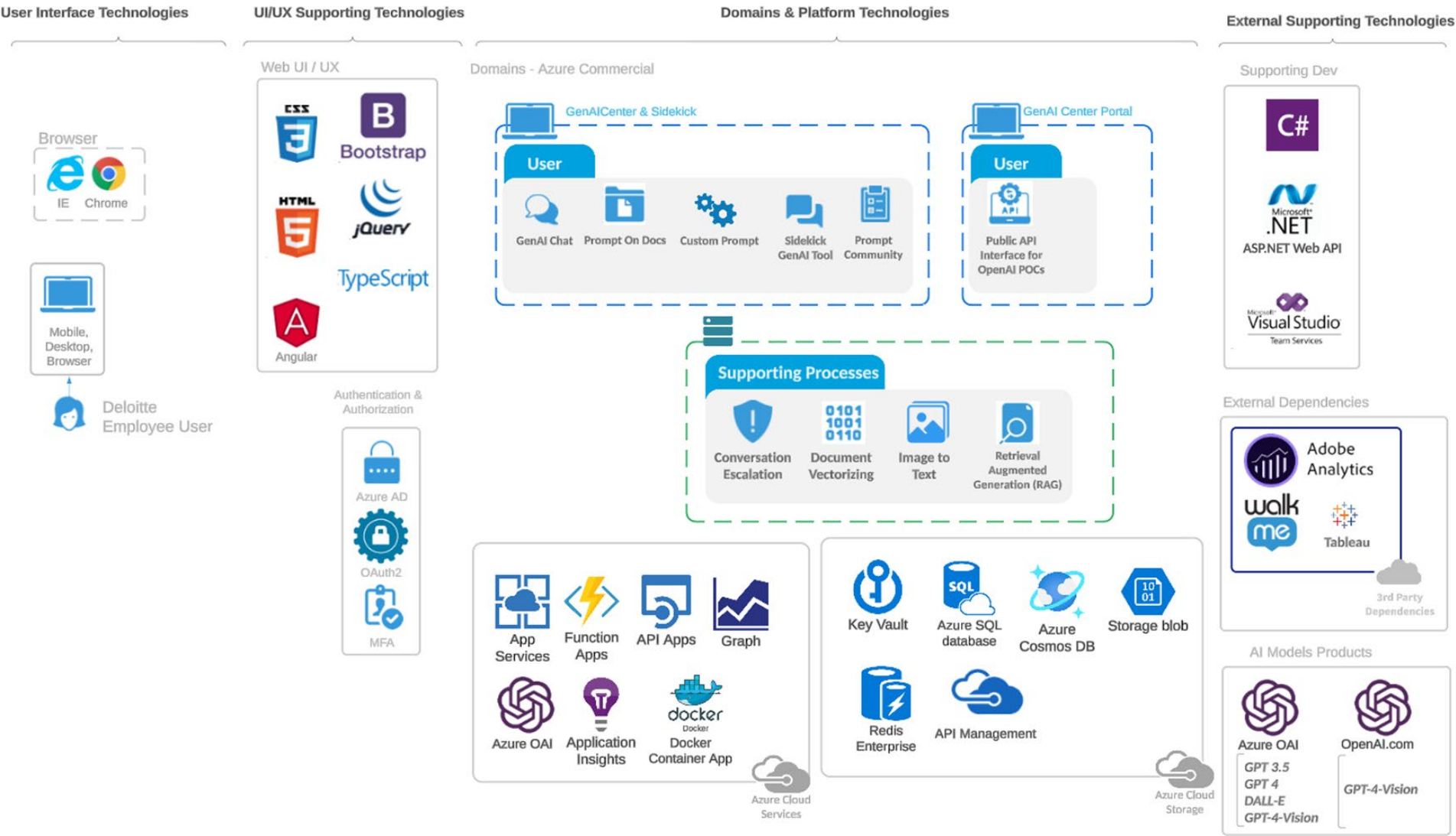
- A department wants to put all of its collected knowledge in a RAG solution. This will consist of PDFs, Word docs, Excel sheets, PowerPoint decks, and more. Some of this will be usable, some won't. It all needs to be curated.

An Advanced RAG Use Case



- A client organization wants to put all of its collected knowledge in a RAG solution. This will have all the issues inherent in the Simple case, just at scale.
- Complexity
- Security
- Cost
- Multidisciplinary
- Sys Admins
- Cloud Services

Sidekick Architecture



Part 1 – Multimodal Models

Introduction to Multimodal models



Multi Modal means Incorporating **additional modalities to LLMs (Large Language Models) creates LMMs (Large Multimodal Models).**

Multimodal can mean one or more of the following:



- Input and output are of different modalities
 - text-to-image, image-to-text
- Inputs are multimodal
 - a system that can process both text and images
- Outputs are multimodal
 - a system that can generate both text and images

“A teddy bear on a skateboard in Times Square”





Different data modes are text, image, audio, tabular data, etc. One data mode can be represented or approximated in another data mode. For example:

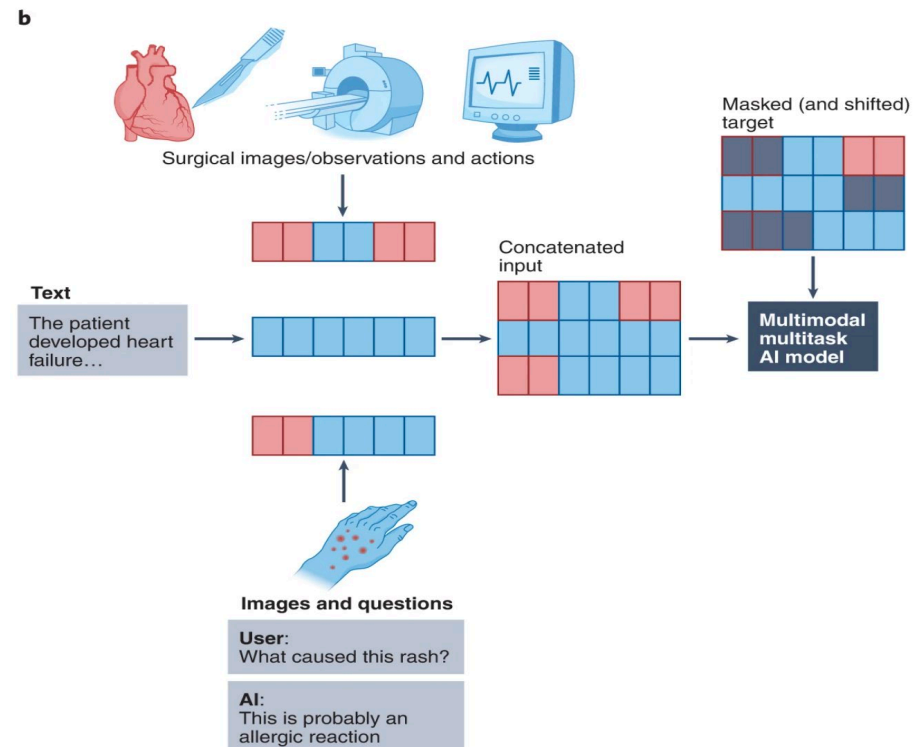
1. Audio can be represented as images (mel spectrograms). 
2. Speech can be transcribed into text, though its text-only representation loses information such as volume, intonation, pauses, etc.
3. An image can be represented as a vector, which, in turn, can be flattened and represented as a sequence of text tokens.
4. A video is a sequence of images plus audio. ML models today mostly treat videos as sequences of images. This is a severe limitation, as sounds have proved to be just as important as visuals for videos. 
5. A text can be represented as an image if you simply take a picture of it.
6. A data table can be converted into a chart, which is an image.

Need of Multimodal Models



Incorporating data from other modalities can help boost model performance.

A model that can learn from both text and images will perform better than a model that can learn from only text or only image.



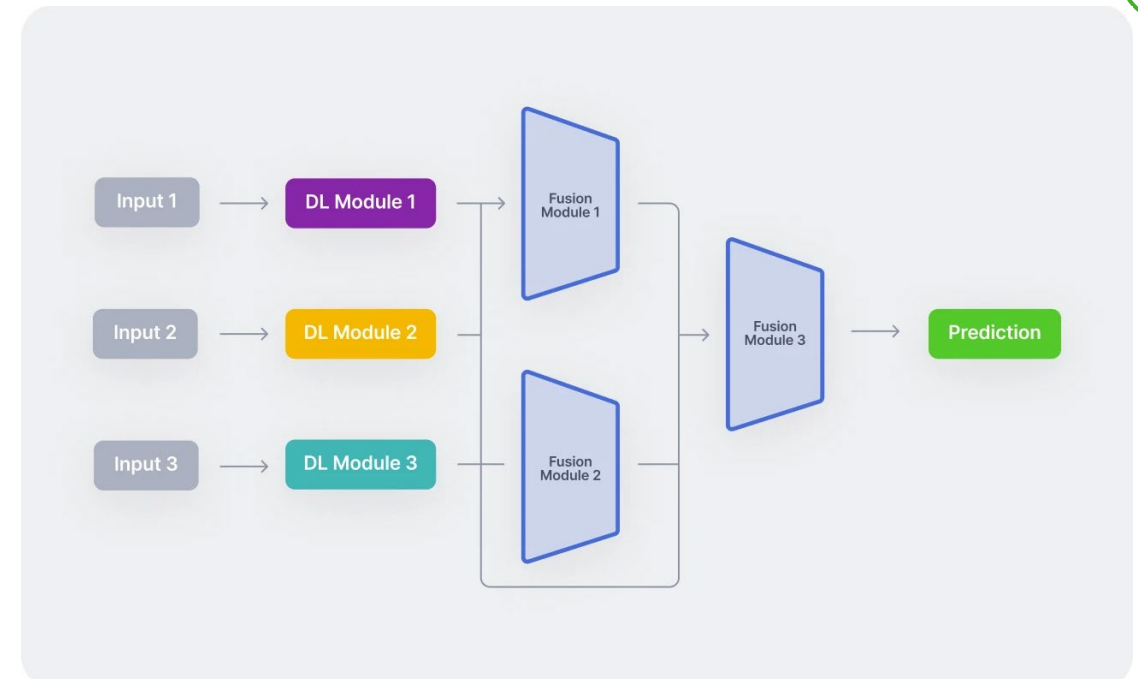
An example of how multimodality can be used in healthcare. Image from Multimodal biomedical AI (Acosta et al., Nature Medicine 2022)

Components of Multimodal systems



At a high level, a multimodal system consists of the following components:

1. An **encoder** for each data modality to generate the embeddings for data of that modality.
2. A way to **align embeddings** of different modalities into the same **multimodal embedding space**.
3. A model that accepts the multimodal embedding space and makes predictions.



Workflow of a typical multimodal. Three unimodal neural networks encode the different input modalities independently. After feature extraction, fusion modules combine the different modalities (optionally in pairs), and finally, the fused features are inserted into a classification network.

CLIP: Contrastive Language-Image Pre-training



- CLIP's key contribution is its ability to map data of different modalities, text and images, into a shared embedding space.
- This shared multimodal embedding space makes text-to-image and image-to-text tasks so much easier.

Food101

guacamole (90.1%) Ranked 1 out of 101 labels



✓ a photo of **guacamole**, a type of food.

✗ a photo of **ceviche**, a type of food.

✗ a photo of **edamame**, a type of food.

✗ a photo of **tuna tartare**, a type of food.

✗ a photo of **hummus**, a type of food.

Youtube-BB

airplane, person (89.0%) Ranked 1 out of 23 labels



✓ a photo of a **airplane**.

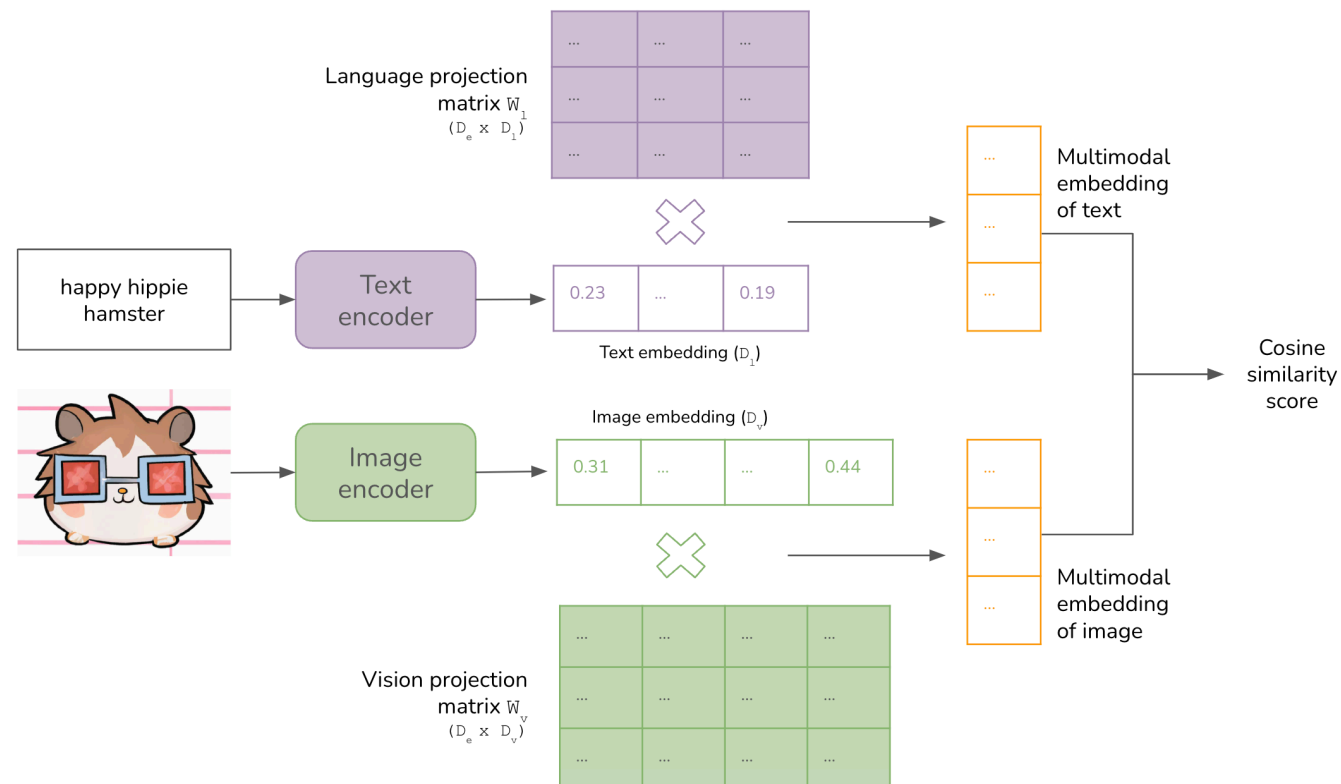
✗ a photo of a **bird**.

✗ a photo of a **bear**.

✗ a photo of a **giraffe**.

✗ a photo of a **car**.

- Text Encoder
- Image Encoder
- Multimodal Embedding is generated for both image and text
- Training goal is to maximize the similarity scores of the right (image, text) pairings while minimizing the similarity scores of the wrong pairings (contrastive learning).



Model Overview:

1. **Text-to-Image Generation:** Creates detailed images from textual descriptions.
2. **Enhanced Capabilities:** Improved image quality, resolution, and coherence.
3. **Diverse Outputs:** Generates a wide variety of realistic and abstract images.
4. **Applications:** Useful in marketing, content creation, and design prototyping.



Model Overview:

- 1. Advanced Text-to-Image Generation:** Produces highly detailed and accurate images from textual prompts.
- 2. Superior Image Quality:** Enhanced realism, resolution, and coherence over previous versions.
- 3. Versatile Outputs:** Capable of generating complex and diverse visual content.
- 4. Applications:** Ideal for creative industries, marketing, and rapid design prototyping.



Model Overview:

- 1. Text to Video:** Designed to generate videos out of text.
- 2. Optimized Performance:** Enhances computational speed and accuracy for complex tasks.
- 3. Robust Architecture:** Built with a resilient framework to ensure reliability and stability.
- 4. Versatile Applications:** Suitable for various industries including finance, healthcare, and logistics.



The background is a solid green color. It features abstract geometric patterns. On the left, there are several overlapping concentric circles. On the right, there is a large, dense grid of small squares. In the bottom right corner, there is a pattern of small dots arranged in a grid-like fashion.

Part 2 - Code Generation with GitHub Copilot

Our objectives:

Code Faster

Code Better

Code Smarter

If you are a programmer and you are not using Gen AI in your day-to-day coding, you are missing out.

Learn to use Gen AI to help you code better, faster, and smarter.

No more wading through pages of Stack Overflow or fruitless Google searches.

Let Gen AI become your pair programmer.-

GitHub Copilot Demonstration

Many ways to use Gen AI

Code creation from comments and context

Solving weird errors

Code migration from one language to another

Mimicry and pattern matching

Code refactoring

Parsing data with optional items

Complex test data generation

See also: [GitHub Copilot - Snack Size Training](#) on Deloitte Media Portal



LAB

Programming with GitHub Copilot

Open VS Code and work through the examples:

Code creation from comments and context

Solving weird errors

Code migration from one language to another

Mimicry and pattern matching

Code refactoring

Parsing data with optional items

Complex test data generation

Part 3 - Discussion: AI Ethics

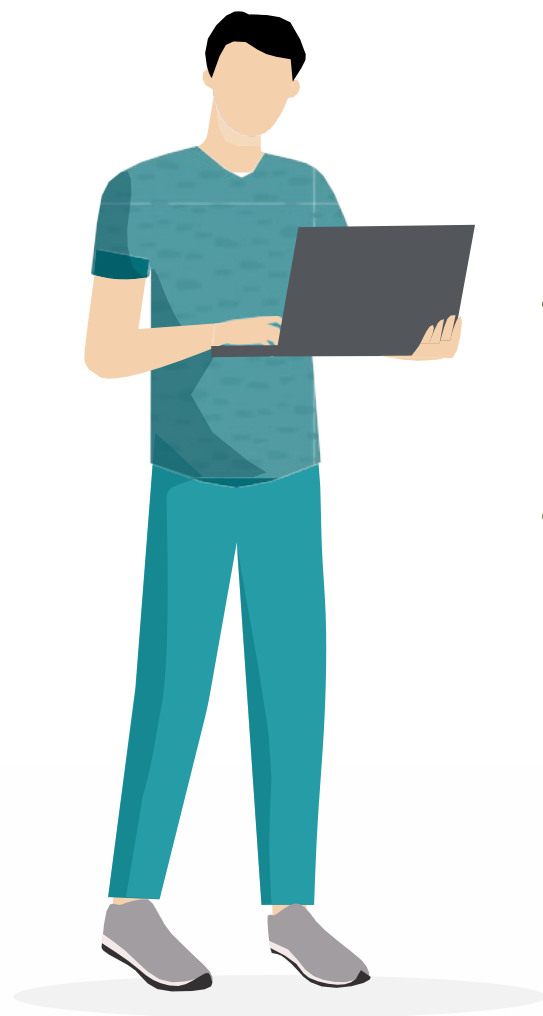
Just because you can do a thing, doesn't mean you should.



- Who defines responsible use of generative AI, especially as cultural norms evolve and social engineering approaches vary across geographies? Who ensures compliance? What are the consequences for irresponsible use?
- In the event something goes wrong, how can individuals take action?
- How do users give and remove consent (opt in or opt out)? What can be learned from the privacy debate?
- Will using generative AI help or hurt trust in your organization — and institutions overall?
- How can we ensure that content creators and owners keep control of their IP and are compensated fairly? What should new economic models look like?
- Who will ensure proper functioning throughout the entire life cycle, and how will they do so? Do boards need an AI ethics lead, for example?

Q & A

Share key Takeaways



01

Multimodal LLMs can see, and hear, changing the way we can interact with them.

02

Multimodal LLMs can generate images, sounds, music, text, and video

03

Multimodal comes with a host of legal and ethical implications

04

**By combining vision (image interpretation) with text analyses,
We can supplement RAG for some of the most complex documents**

05

Combining multimodalities creates new opportunities for a more accessible world.



About Deloitte

Deloitte refers to one or more of Deloitte Touche Tohmatsu Limited, a UK private company limited by guarantee (“DTTL”), its network of member firms, and their related entities. DTTL and each of its member firms are legally separate and independent entities. DTTL (also referred to as “Deloitte Global”) does not provide services to clients. In the United States, Deloitte refers to one or more of the US member firms of DTTL, their related entities that operate using the “Deloitte” name in the United States and their respective affiliates. Certain services may not be available to attest clients under the rules and regulations of public accounting. Please see www.deloitte.com/about to learn more about our global network of member firms.

Copyright © 2025 Deloitte Development LLC. All rights reserved.