LA TECHNOLOGY TRANSFER
presenta

IA Generativa liberare il potenziale di Business usando Agentic AI con LangChain e LangGraph

di Ivan Reznikov

Online live streaming
15-16 Maggio 2025

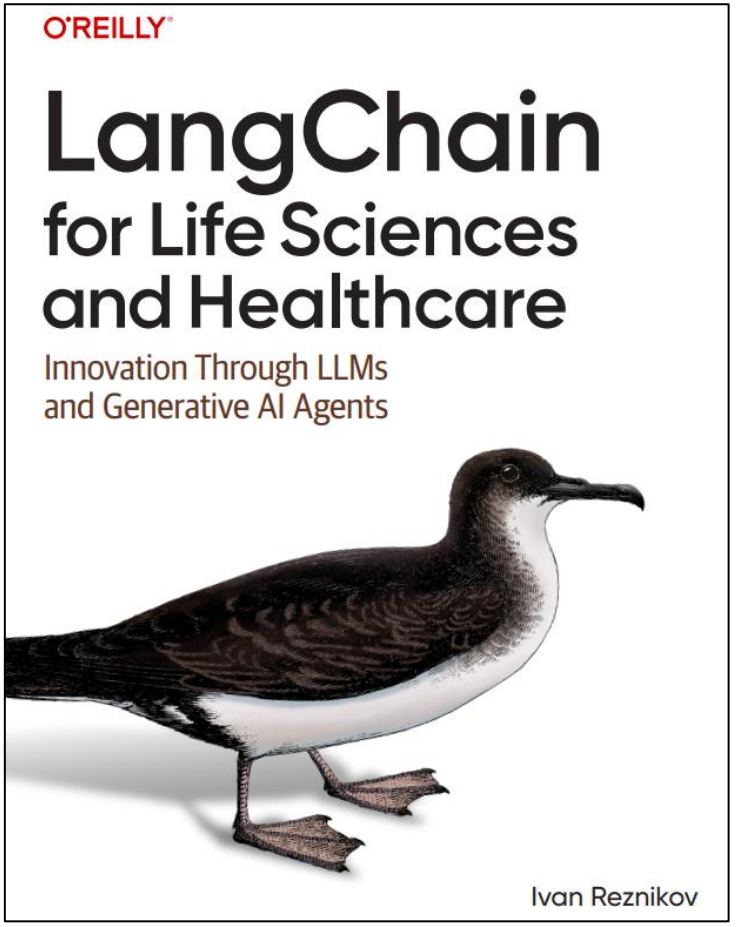
Technology Transfer
Piazza Cavour, 3 - 00193 Roma
Tel. 06-6832227 **Fax** 06-6871102
e-mail: info@technologytransfer.it **Web:** www.technologytransfer.it

Shameless Self-Promotion

- ✓PhD in Computational Sciences
- ✓12+ years of Python and Data Science experience
- ✓Worked for small/large companies, startups, government
- ✓Principal Data Scientist
- ✓Kaggle Competition Expert
- ✓TEDx Speaker (2017), GITEX/PyCON(2021, 2023), CodersHQ(2022, 2023), O'Reilly AI (2024)
- ✓Author of "LangChain for Life Sciences and Healthcare"
- ✓Visiting Lecturer @ Middlesex University Dubai, MS in Data Science

https://www.linkedin.com/in/reznikovivan/
https://github.com/IvanReznikov

Shameless Self-Promotion (last slide, I promise)

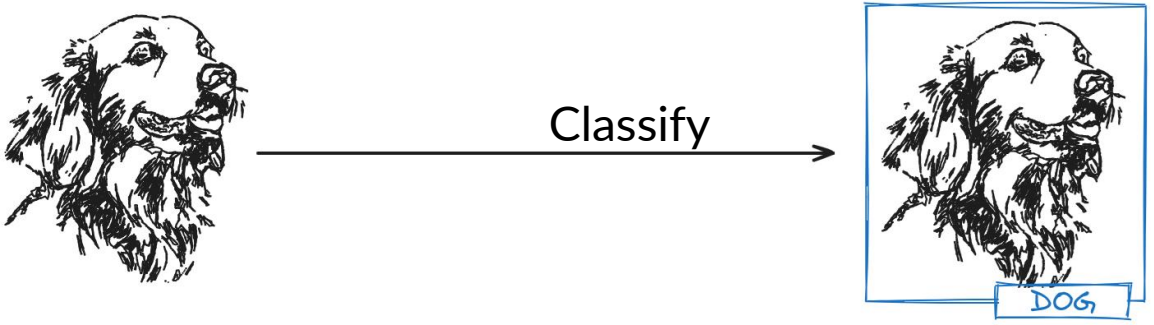


Plan

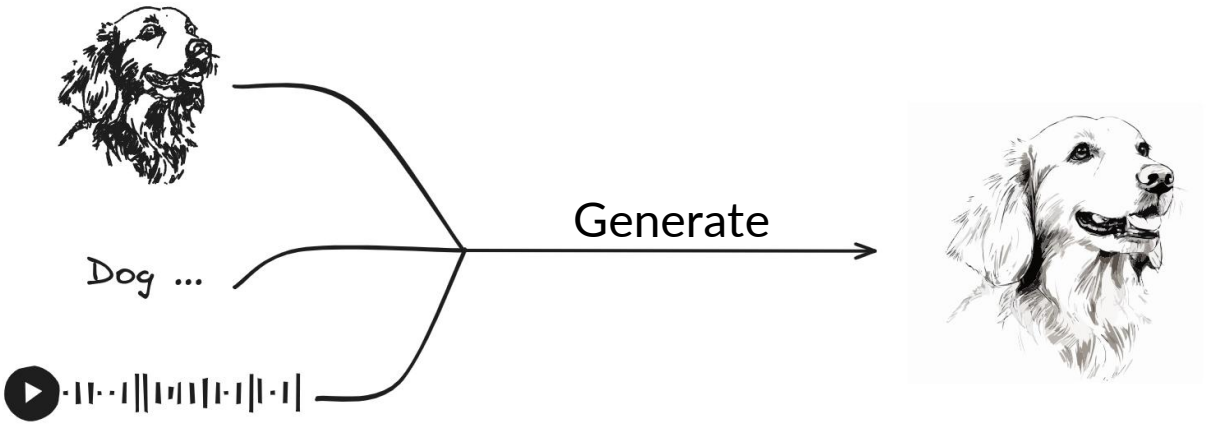
Day 1	Day 2
What is Generative AI?	Practice with Agentic AI
What are Large Language Models and how they work?	Practice with No/Low Code frameworks
Basics of LangChain and RAG Pipelines	Practice with Multi Agent
Practice with LangChain	Practice with Guardrails
Basics of Agentic AI	

Generative AI Basics

Deep Learning Model Types



Discriminative model: classify as a dog or cat



Generative model: generate an image of a dog

Deep Learning Model Types

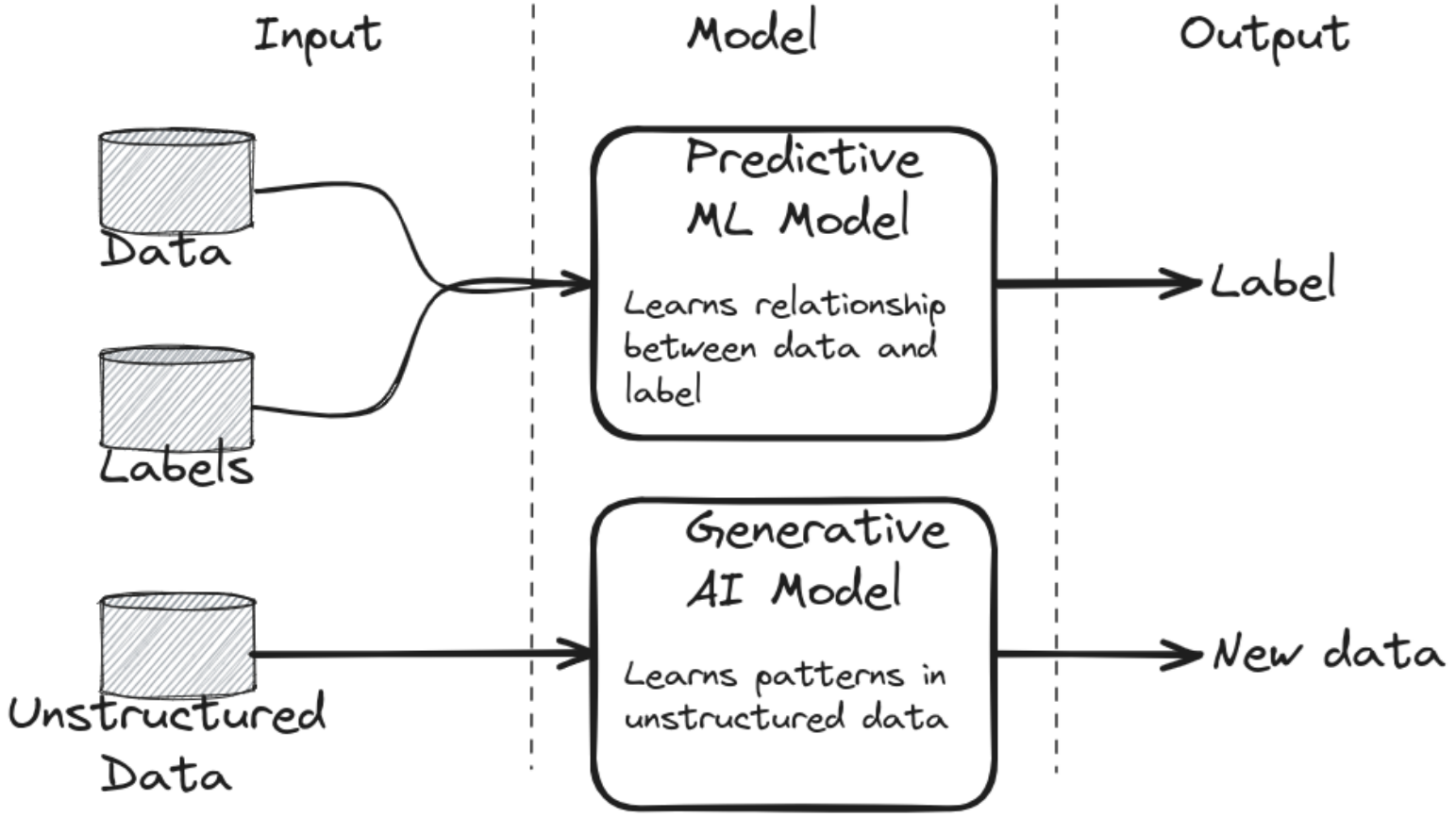
Discriminative AI

- Used to classify or predict
- Typically trained on a dataset of labeled data
- Learns the relationship between the features of the data points and the labels

Generative AI

- Generates new data that is similar to data it was trained on
- Understands distribution of data and how likely a given example is
- Predict next word in a sequence

Deep Learning Model Types



Applications?

Business / Tech

Fake Biden robocall linked to Texas-based companies, New Hampshire attorney general announces

By David Wright, Yahya Abou-Ghazala and Brian Fung, CNN

6 minute read · Updated 3:58 PM EST, Tue February 6, 2024

TECHNOLOGY EXECUTIVE COUNCIL

Generative AI financial scammers are getting very good at duping work email

PUBLISHED WED, FEB 14 2024•11:54 AM EST UPDATED 3 HOURS AGO

Ellen Sheng
@ELLENSHENG

WATCH LIVE

KEY POINTS

- Even companies that ban employees from using generative artificial intelligence are falling prey to financial scams that deploy the technology and amplify traditional phishing techniques used by hackers.
- Armed with tools like ChatGPT or its dark web equivalent FraudGPT, criminals can easily create realistic videos of profit and loss statements, fake IDs, false identities or even convincing deepfakes of company executives using their voice and image.
- A recent scam that cost a Hong Kong-based company over $25 million shows how convincing the crimes have become and how difficult it is to detect them.

How Large?

10 TBs of Data from web

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

How Large??

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

LLAMA Model

> 5000 GPUs
weeks, months
~ $10M

How Much did DeepSeek Spend??

China's DeepSeek became the biggest topic in tech this week, with many in the industry and on Wall Street focused on a single number: $6 million.

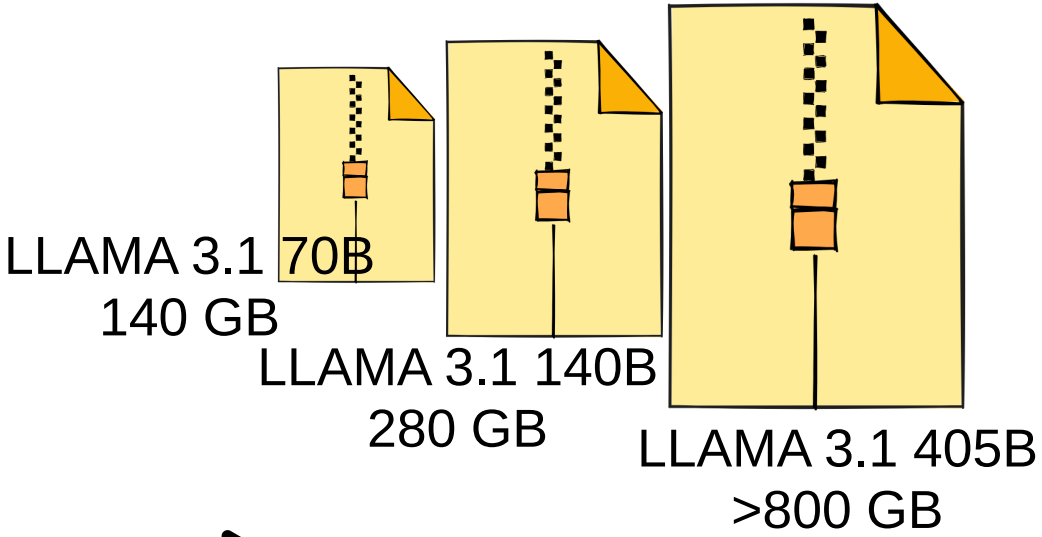
In DeepSeek's paper about its newest artificial intelligence model, the company said that its total training costs amounted to $5.576 million, based on the rental price of Nvidia's graphics processing units. DeepSeek included a clear caveat, saying that the number included only the model's "official training" and excluded the costs tied to "prior research and ablation experiments on architectures, algorithms, or data."

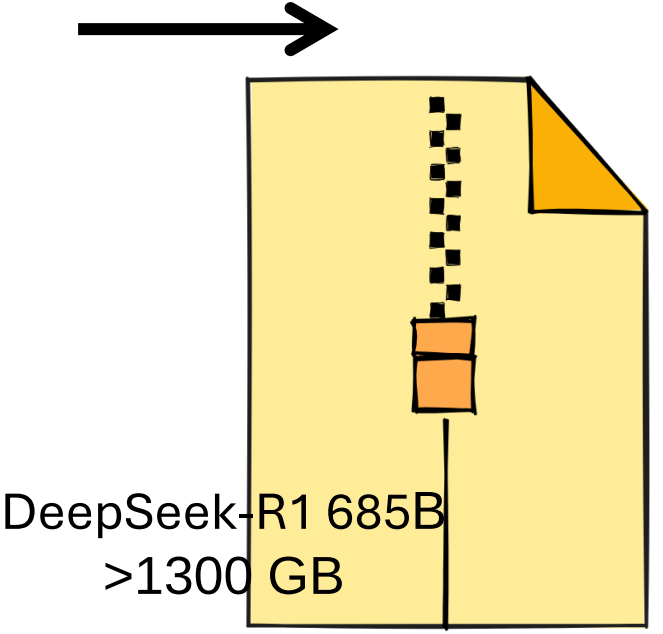
Early in the week, DeepSeek's AI Assistant took the coveted spot for most-downloaded free app in the U.S. on Apple's App Store, dethroning OpenAI's ChatGPT. Global tech stocks sold off, with chipmakers Nvidia and Broadcom losing a combined $800 billion in market cap on Monday.

A new report from SemiAnalysis, a semiconductor research and consulting firm, added more context to DeepSeek's expenses. The firm estimated that DeepSeek's hardware spend is "well higher than $500M over the company history," adding that R&D costs and total cost of ownership are significant. Generating "synthetic data" for the model to train on would require "considerable amount of compute," SemiAnalysis wrote.

How Large???

> 5000 GPUs
weeks, months
~ $10M





Large Language Models

Tokenizers

Tokenization is breaking down human language into smaller units like words or subwords to convert text into machine-readable numerical representations.

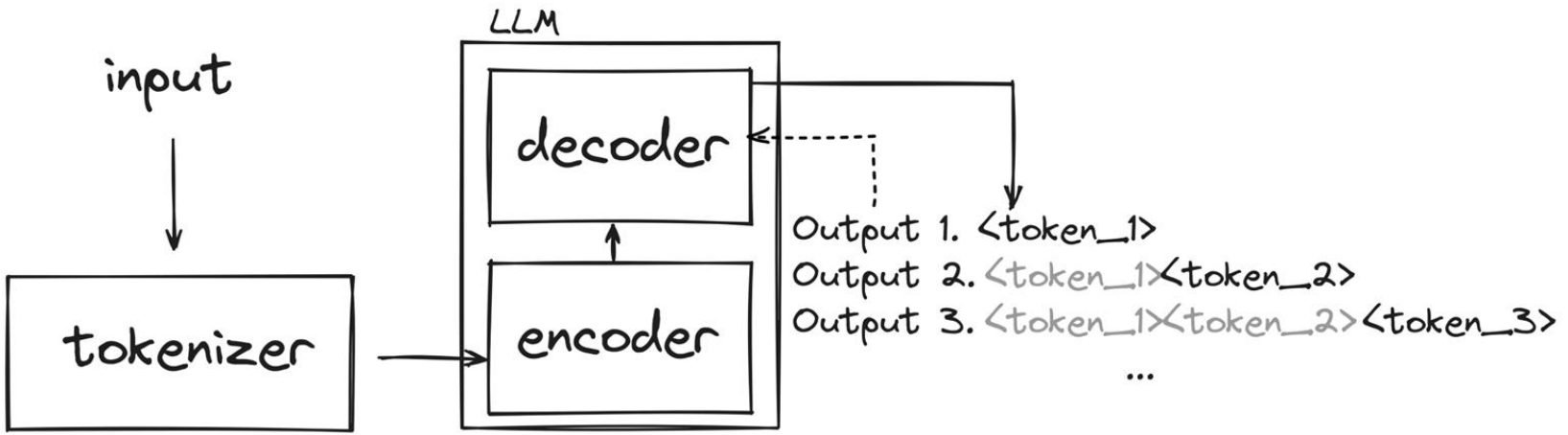
Ah, the siren call of cosmopolitan metropolises! One could scarcely conceive of a more tapestried itinerary than a peregrination commencing in the inimitable Paris, a city redolent with artistic efflorescence and gourmand delights, thence proceeding to the venerable Rome, a veritable palimpsest of ancient grandeur and baroque magnificence, culminating in the multifarious London, a sprawling agglomeration pulsating with contemporary verve and historical gravitas.

Ah, il richiamo ammaliatore delle metropoli cosmopolite! Si potrebbe a malapena concepire un itinerario più variopinto di un pellegrinaggio che inizi nell' inimitabile Parigi, una città ricca di efflorescenza artistica e delizie gourmand, procedendo poi alla venerabile Roma, un vero e proprio palinsesto di antica grandezza e barocca magnificenza, culminando nella multiforme Londra, un vasto agglomerato pulsante di vivacità contemporanea e gravità storica.

https://platform.openai.com/tokenizer

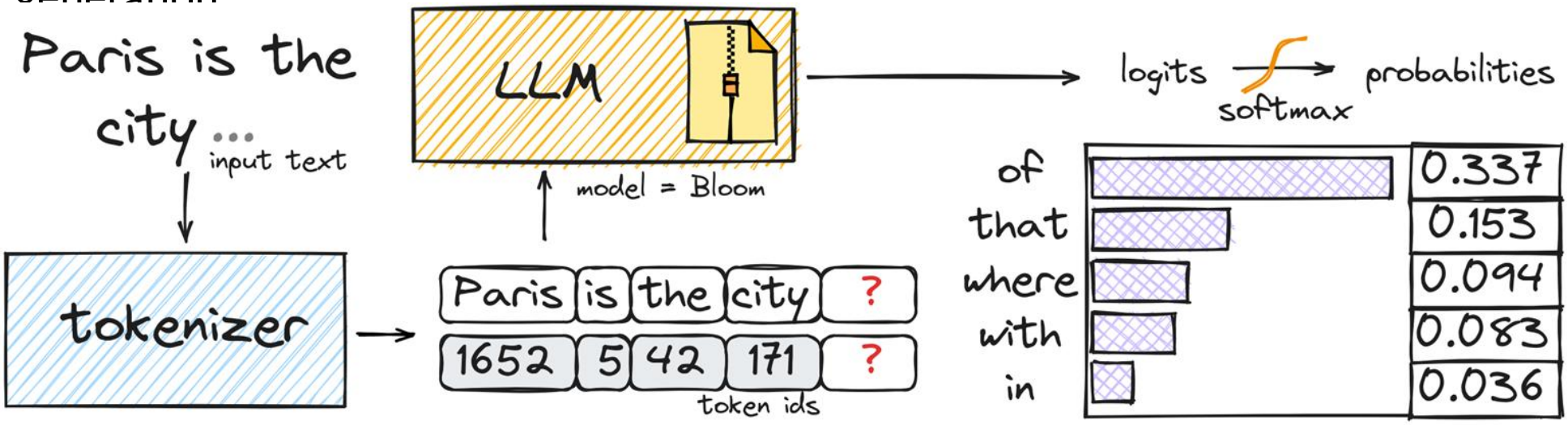
Three main steps for text generation

- The input text is passed to a tokenizer that generates unique numerical representation of every token.
- The tokenized input text is passed to the Encoder part of the pre-trained model. The Encoder processes the input and generates a feature representation that encodes inputs meaning and context.
- The Decoder takes the feature representation from the Encoder and starts generating new text based on that context token by token.



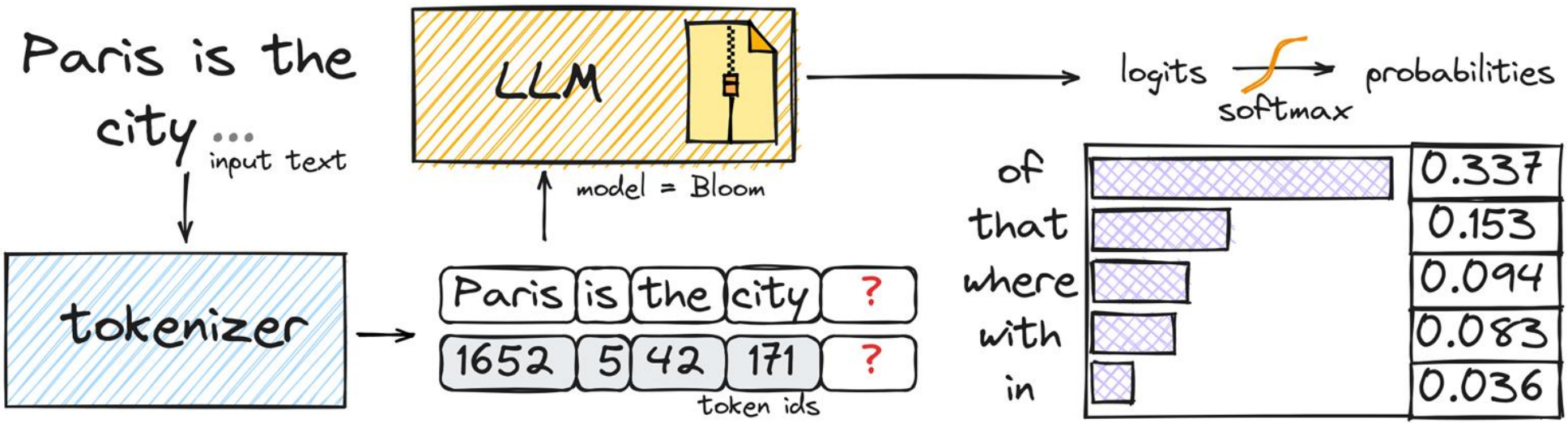
Decoding the outputs

Say, we want to generate the continuation of the phrase "*Paris is the city ...*".
The Encoder (we'll be using Bloom-560m model) sends logits for all the tokens we have (if you don't know what logits are — consider them as scores) that can be converted, using softmax function, to probabilities of the token being selected for generation



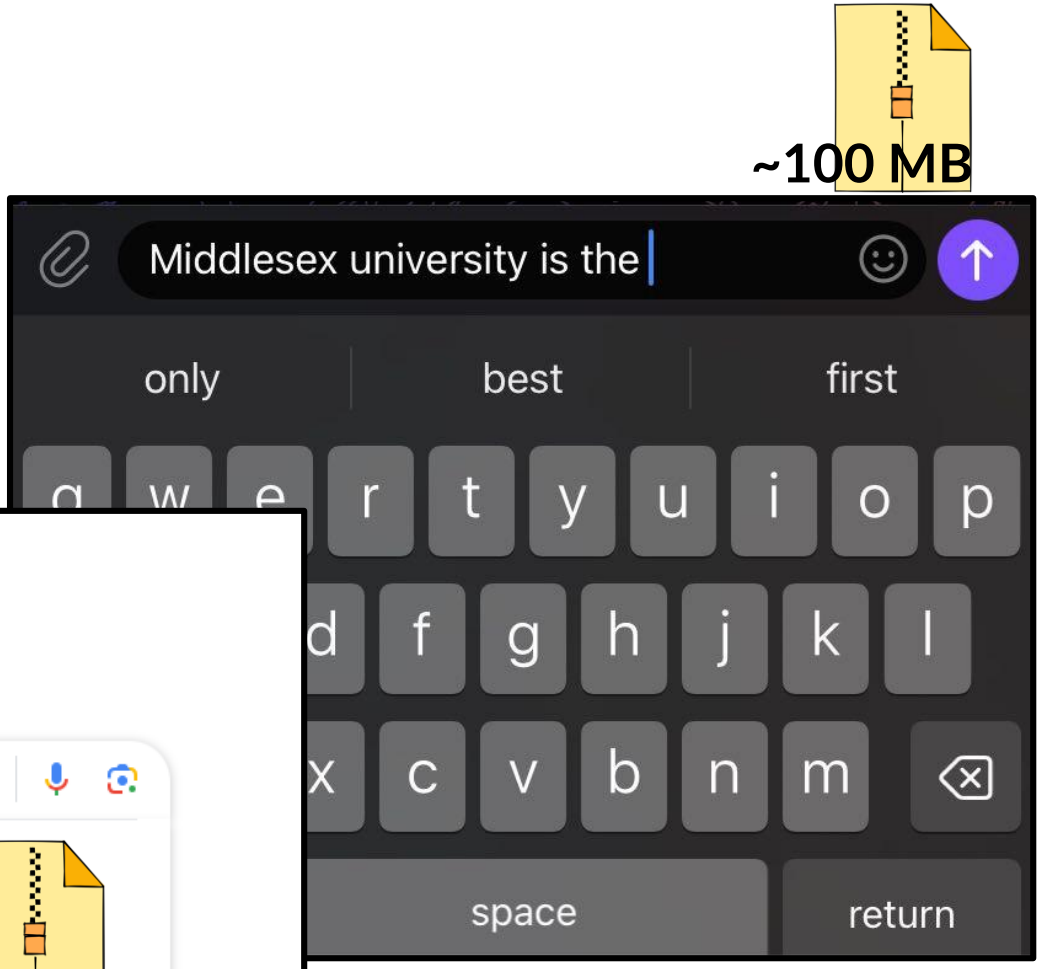
Possible following tokens

- Paris is the city **of** love.
- Paris is the city **that** never sleeps.
- Paris is the city **where** art and culture flourish.
- Paris is the city **with** iconic landmarks.
- Paris is the city **in** which history has a unique charm.

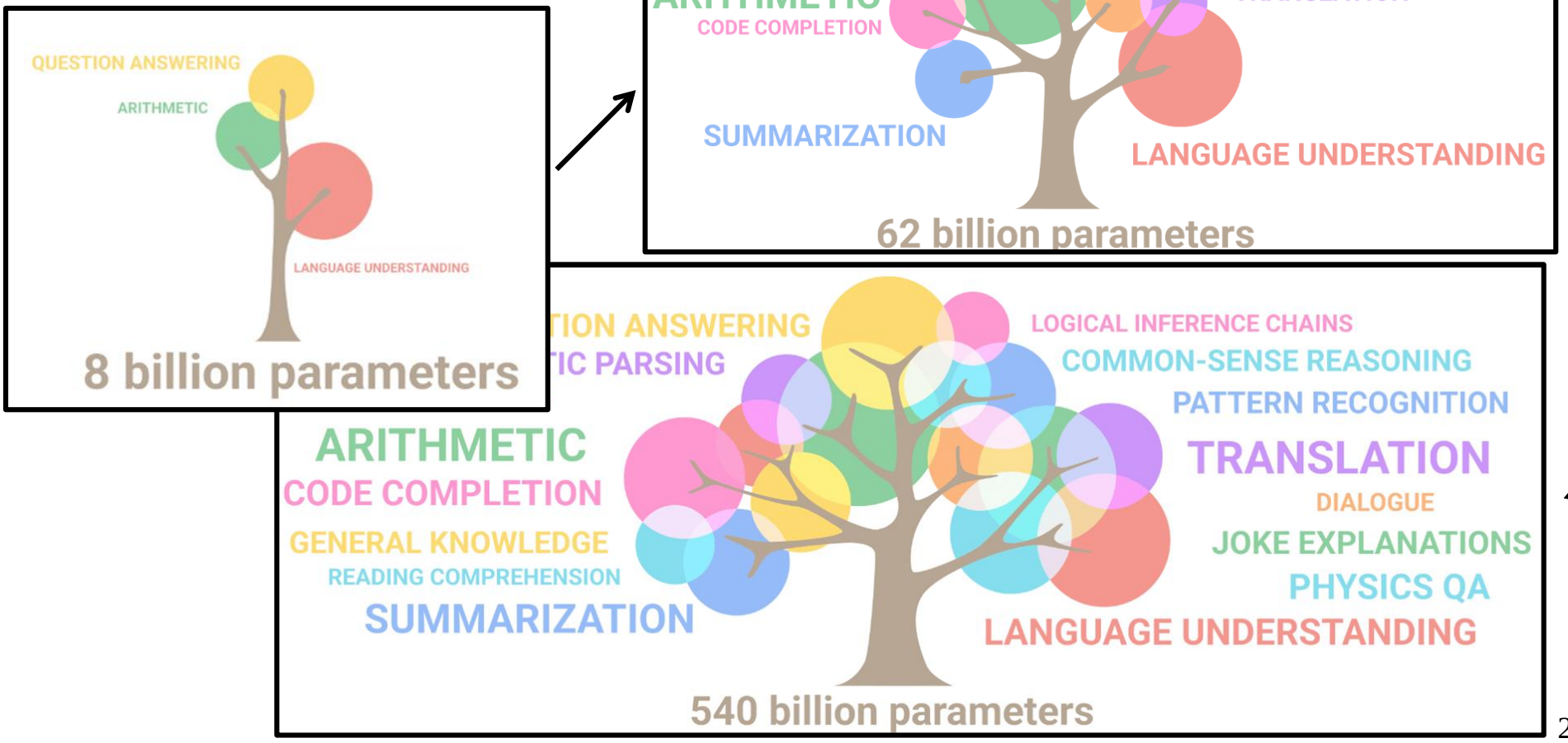


Language models b.c.

Language models existed before ChatGPT. The difference is the size. We're talking about a shift from hundreds of megabytes to hundreds of gigabytes!



Model skills



https://blog.research.google/2022/04/pathways-language-model-palm-scaling-to.html

Playground OpenAI

gpt-3.5-turbo-instruct

Knowledge of a Large Language Model

Paris

Article Talk

From Wikipedia, the free encyclopedia

This article is about the capital of France. For other uses, see Paris (disambiguation).

Paris (French pronunciation: [paʁi]) is the capital and most populous city of France. With an official estimated population of 2,102,650 residents as of 1 January 2023[2] in an area of more than 105 km^2 (41 sq mi),[5] Paris is the fourth-most populated city in the European Union and the 30th most densely populated city in the world in 2022.[6] Since the 17th century, Paris has been one of the world's major centres of finance, diplomacy, commerce, culture, fashion, and gastronomy. For its leading role in the arts and sciences, as well as its early and extensive system of street lighting, in the 19th century, it became known as the City of Light.[7]

The City of Paris is the centre of the Île-de-France region, or Paris Region, with an official estimated population of 12,271,794 inhabitants on 1 January 2023, or about 19% of the population of France.[2] The Paris Region had a GDP of €765 billion (US$1.064 trillion, PPP)[8] in 2021, the highest in the European Union.[9] According to the Economist Intelligence Unit Worldwide Cost of Living Survey, in 2022, Paris was the city with the ninth-highest cost of living in the world.[10]

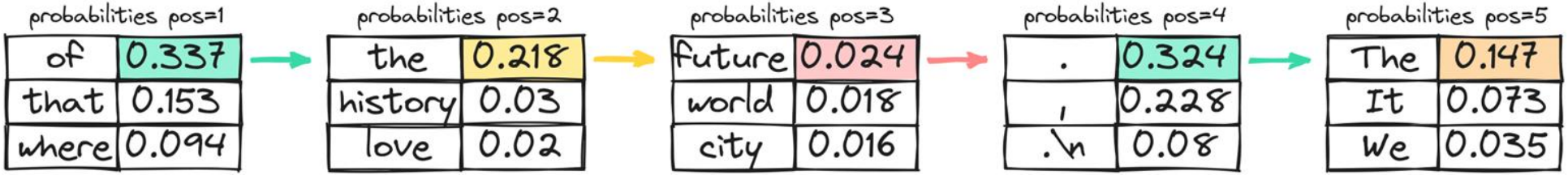
- Facts
- Dates
- Entity Relations
- Punctuation
- Synonyms
- Semantics
- Translation

- <u>Complex:</u>
- Reasoning
- Math

1. Greedy Sampling

In the greedy strategy, the model always chooses the token it believes is the most probable at each step — it doesn't consider other possibilities or explore different options. The model selects the token with the highest probability and continues generating text based on the selected choice.

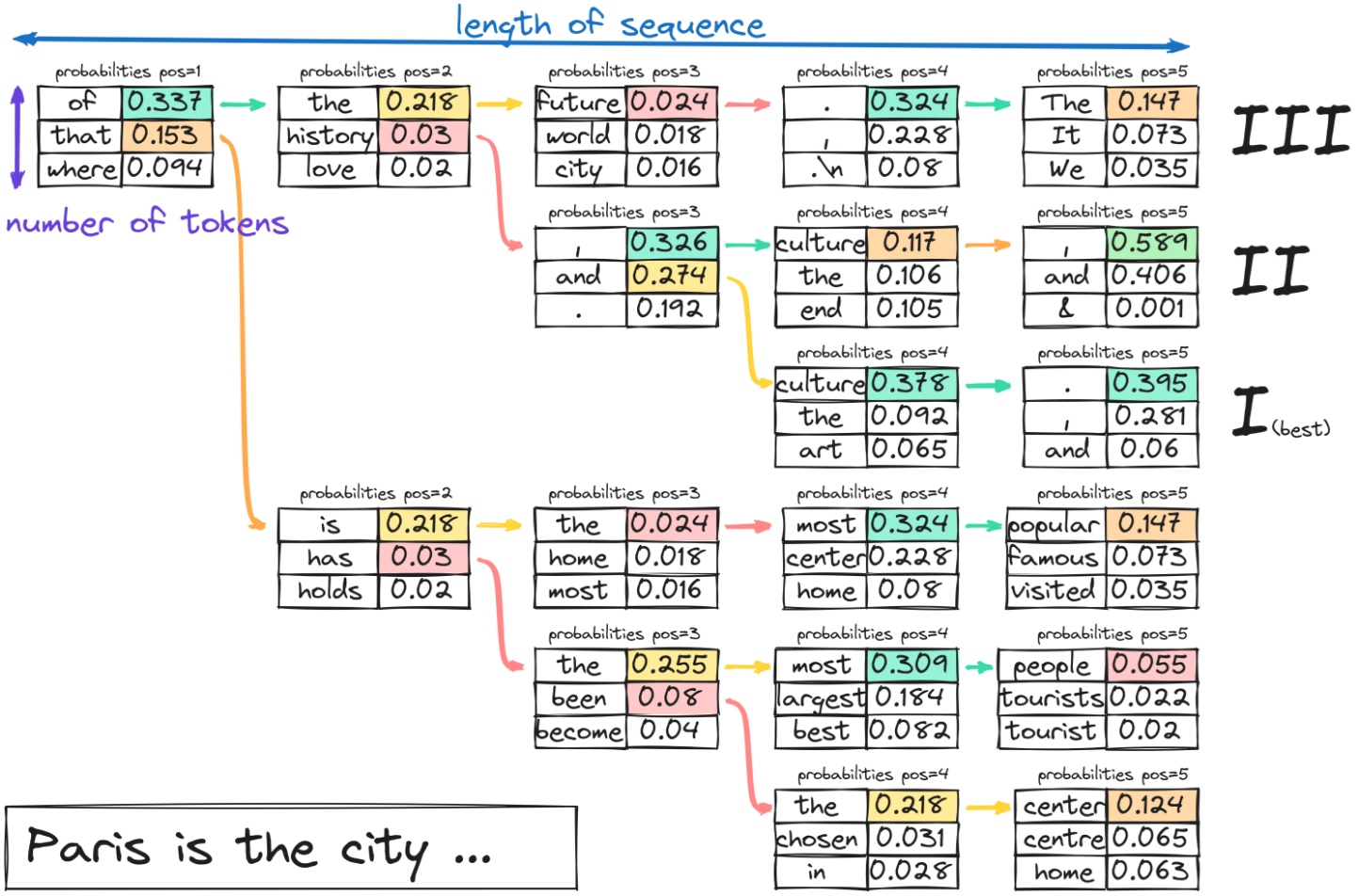
Generated output: Paris is the city of the future. The



2. Beam search

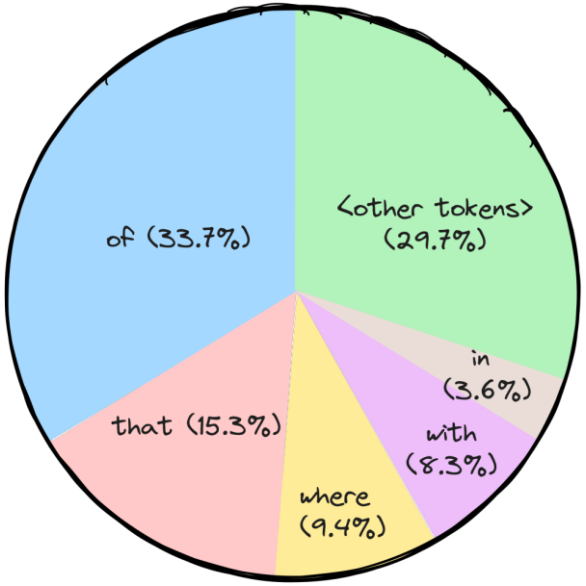
In beam search, instead of just considering the most likely token at each step, the model considers a set of the top "k" most probable tokens. This set of k tokens is called a "beam."

Generated output: Paris is the city of history and culture.



3. Normal Random Sampling

In normal random sampling you select the next word by choosing a random value and mapping it to the token got picket. Imagine it as spinning a wheel, where the area of each token is defined by its probability. The higher the probability — the more chances the token would get selected.

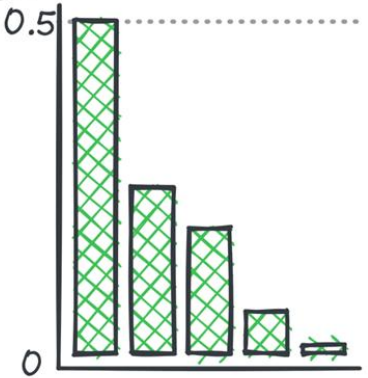


	logits
token A	-1.806
token B	-2.499
token C	-2.786
token D	-3.885
token E	-5.494

softmax →

probabilities
0.494
0.247
0.185
0.062
0.012

$$softmax(x_i) = \frac{e^{x_i}}{\sum_i e^{x_i}}$$



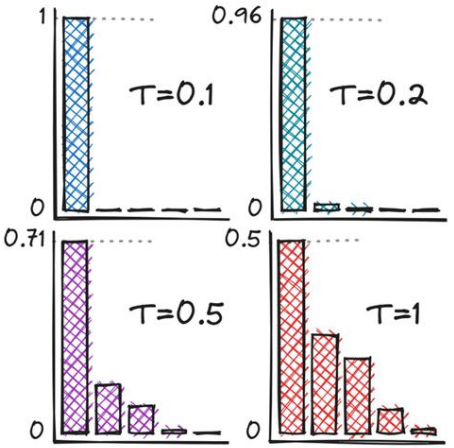
4. Random Sampling + Temperature

We've been using the softmax function to convert logits to probabilities.
We now introduce temperature — a hyperparameter that affects the randomness of the text generation. The difference with the classic softmax is in the denominator - we divide by T. Higher values of temperature (e.g., 1.0) make the output more diverse, while lower values (e.g., 0.1) make it more focused and deterministic. In fact, T = 1 will lead to the softmax function we used initially.

	logits
token A	-1.806
token B	-2.499
token C	-2.786
token D	-3.885
token E	-5.494

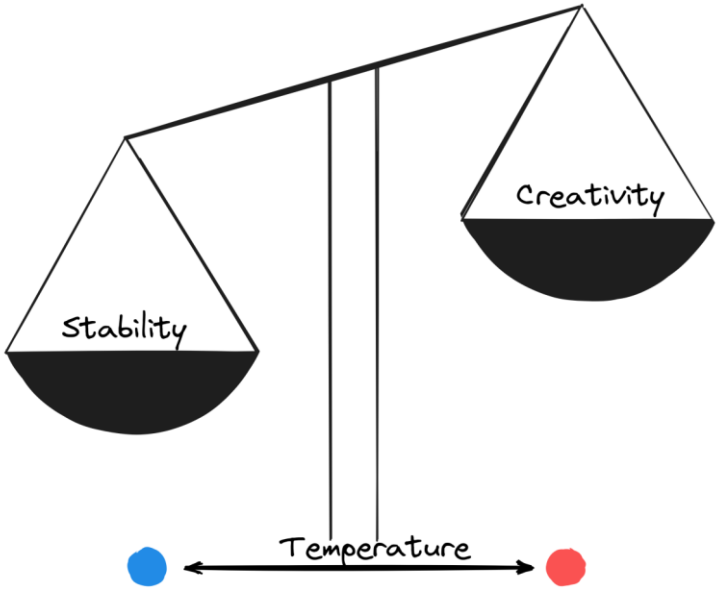
softmax

$$softmax(x_i) = \frac{e^{x_i/T}}{\sum_i e^{x_i/T}}$$



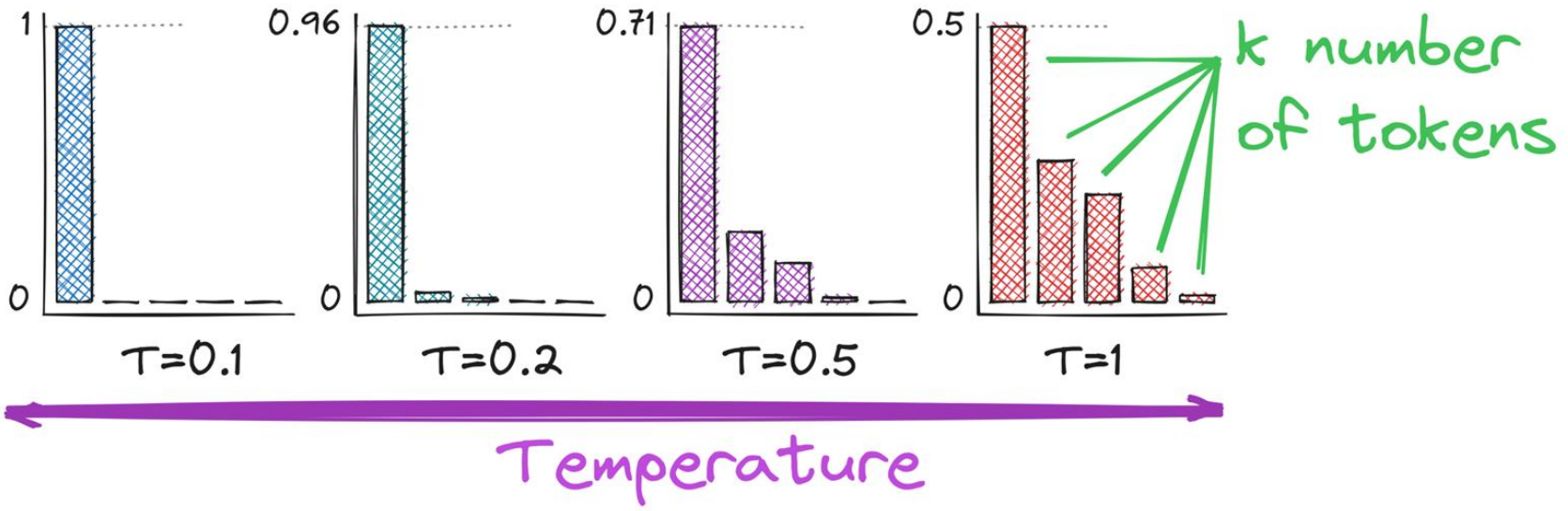
Temperature

Temperature controls the randomness of token selection during decodingHigher temperature boosts creativity, whereas lower temperature is more about coherence and structure. While embracing creativity allows for fascinating linguistic adventures, tempering it with stability ensures the elegance of the generated text.



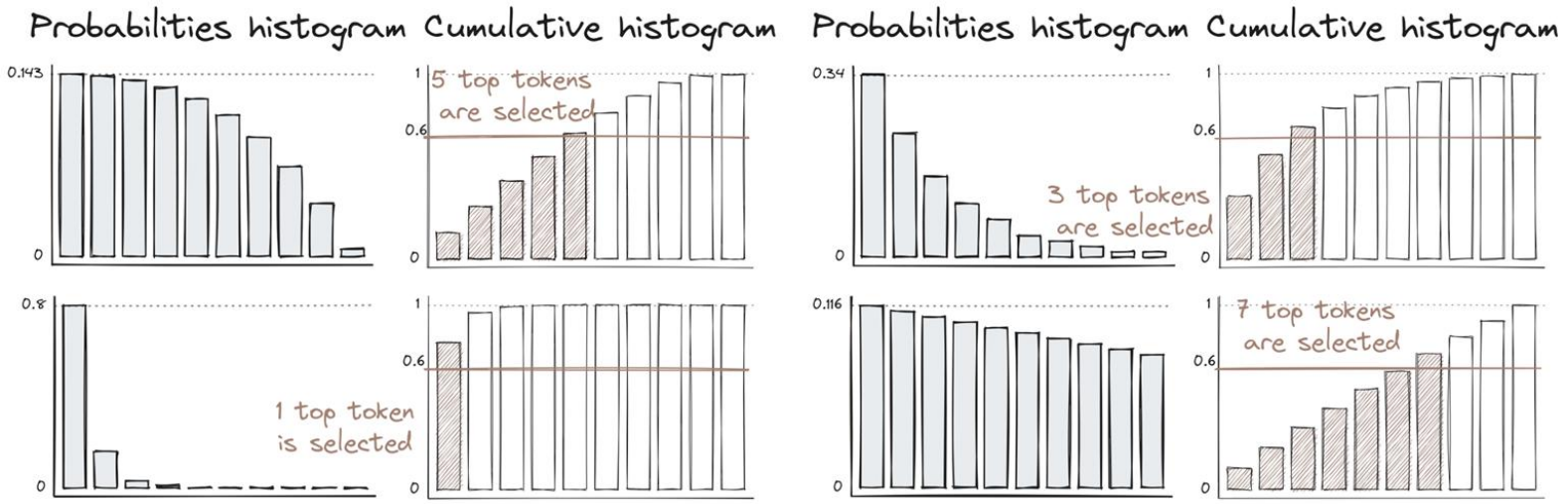
5. Top-k sampling

We can now shift probabilities with temperature. Another enhancement is to use top-k tokens rather than all of them. This will increase the stability of the text generation, not decreasing creativity too much. Basically, it's now random sampling with temperature for only top k tokens.



6. Top-p or nucleus sampling

Nucleus sampling is designed to address some limitations of different sampling techniques. Instead of specifying a fixed number of "k" tokens to consider, a probability threshold "p" is used. This threshold represents the cumulative probability that you want to include in the sampling.



Summary

Method	Description	Diversity	Performance	Quality of Output
Greedy Search	Selects the most probable token at each step.	Low	High	May lack diversity
Beam Search	Considers top-K probable tokens in parallel.	Moderate	Medium-High	Improved diversity
Random Sampling	Randomly selects tokens based on probabilities.	High	Medium	May lack coherence
Random Sampling with Temperature	Adds randomness based on temperature.	High	Medium	More controlled randomness
Top-K Sampling	Selects top-K probable tokens randomly.	High	Medium	Improved diversity
Nucleus Sampling	Selects tokens until cumulative probability ≤ p.	High	Medium-High	Improved coherence

.from_pretrained(<model>)

In many pre-trained language models, the tokenizer and the model architecture are designed and trained together. The reason for this coupling is that the tokenizer and the model architecture need to be compatible with each other to ensure consistent tokenization and decoding.

If the tokenizer and the model architecture were different or not synchronized, it could lead to tokenization errors, mismatched embeddings, and incorrect predictions.

```
1 tokenizer = AutoTokenizer.from_pretrained("some_model")
2 model = BloomForCausalLM.from_pretrained("some_model")
```

API keys

You will first need an API key for the LLM provider you want to use. Currently, we must choose between proprietary or open-source foundation models based on a trade-off mainly between performance and cost.

Many open-source models are organized and hosted on Hugging Face as a community hub

```
1 os.environ["OPENAI_API_KEY"] = ... # API_TOKEN
2 os.environ["HUGGINGFACEHUB_API_TOKEN"] = ... # API_TOKEN
```

API keys

Proprietary models are closed-source foundation models owned by companies. They usually are larger than open-source models and thus have better performance, but they may have expensive APIs.
Examples: OpenAI, co:here, AI21 Labs, Anthropic, etc.

Open-source models are usually smaller models with lower capabilities than proprietary models, but they are more cost-effective than proprietary ones.
Examples: BLOOM (BigScience), LLaMA (Meta), Flan-T5 (Google), etc.

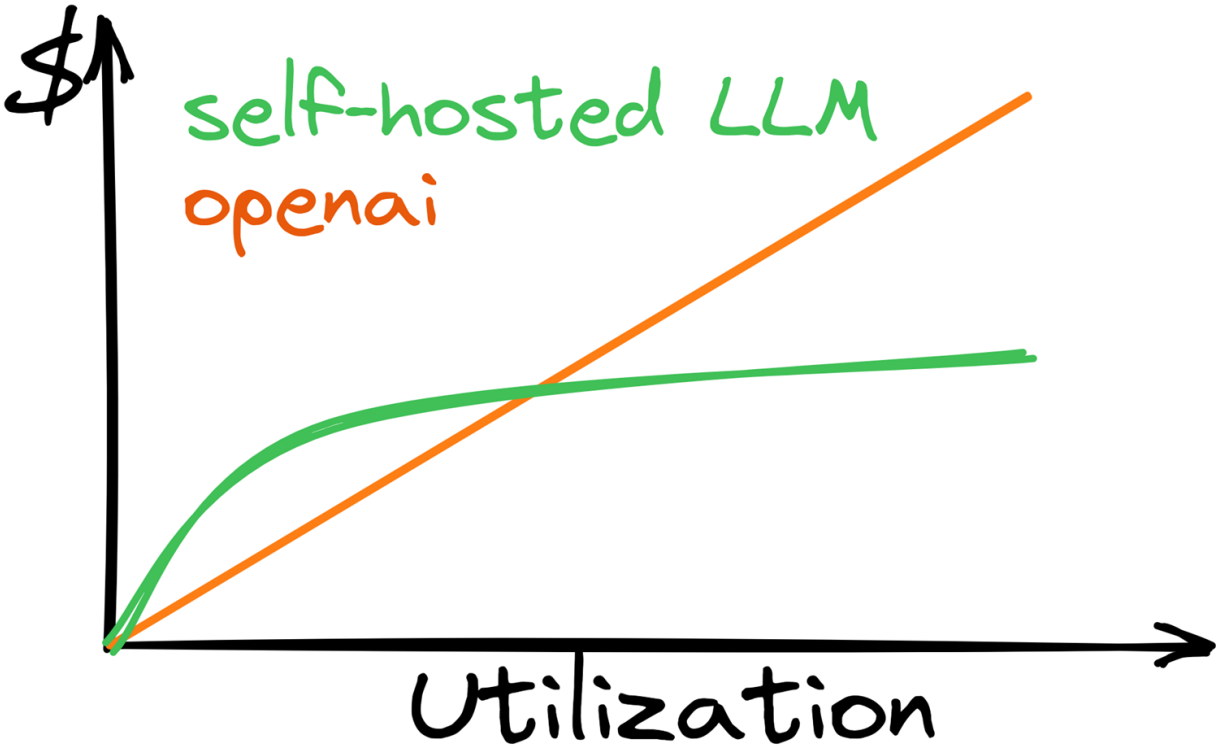
```
# Proprietary LLM from e.g. OpenAI
from langchain.llms import OpenAI
llm = OpenAI(model_name="text-davinci-003")

# Alternatively, open-source LLM hosted on Hugging Face
from langchain import HuggingFaceHub
llm = HuggingFaceHub(repo_id = "google/flan-t5-xl")
```

Cost per token

Model Name	Provider	Context Length	Input Cost ($/1K tokens)	Output Cost ($/1K tokens)
OpenAI GPT-4o	OpenAI	128,000	$0.005	$0.015
Anthropic Claude 3.5 Sonnet	Anthropic	200,000	$0.003	$0.015
Google Gemini 1.5 Pro	Google	1,000,000	$0.007	$0.021
Cohere Command R	Cohere	128,000	$0.15	$0.60
Mistral Mixtral 8x7B	Mistral	32,768	$0.0007	$0.0007
Meta Llama 3 70B	Meta	8,192	$0.00059	$0.00079

April 2025

Self-hosted or API?



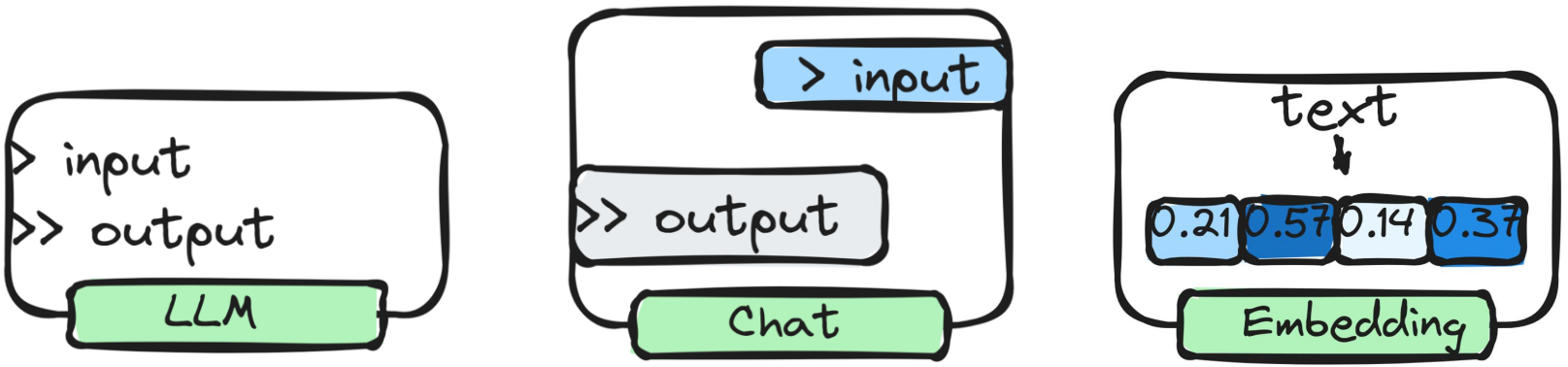
What to choose?

You want	To choose
private data usage	self-hosted LLM
control over model	self-hosted LLM
quick development	openai
low costs	openai (probably)
minimal infra	openai
top quality	openai (currently)

Models

LangChain supports a variety of different models, so you can choose the one that suits your needs best:

- **Language Models** are used **for generating text**
    - LLMs utilize APIs that take input text and generate text outputs
    - ChatModels employ models that process **chat messages** and produce responses
- **Text Embedding Models** convert text into **numerical representations**



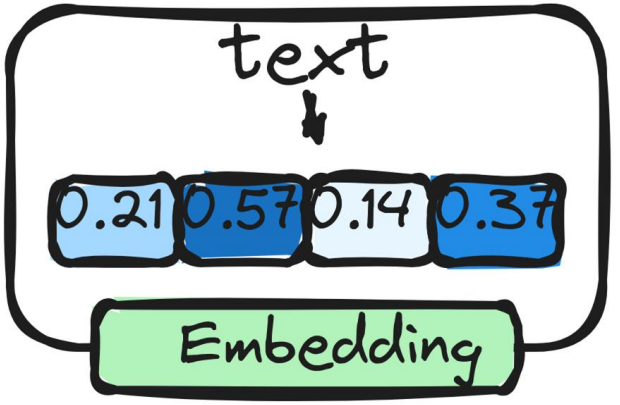
Embeddings

Embeddings are compact numerical representations of words or entities that help computers understand and process language more effectively.
These representations encode the meaning and context of words, allowing machines to work with language in a more meaningful and efficient way.

- OpenAIEmbeddings
- HuggingFaceEmbeddings
- GPT4AllEmbeddings
- etc

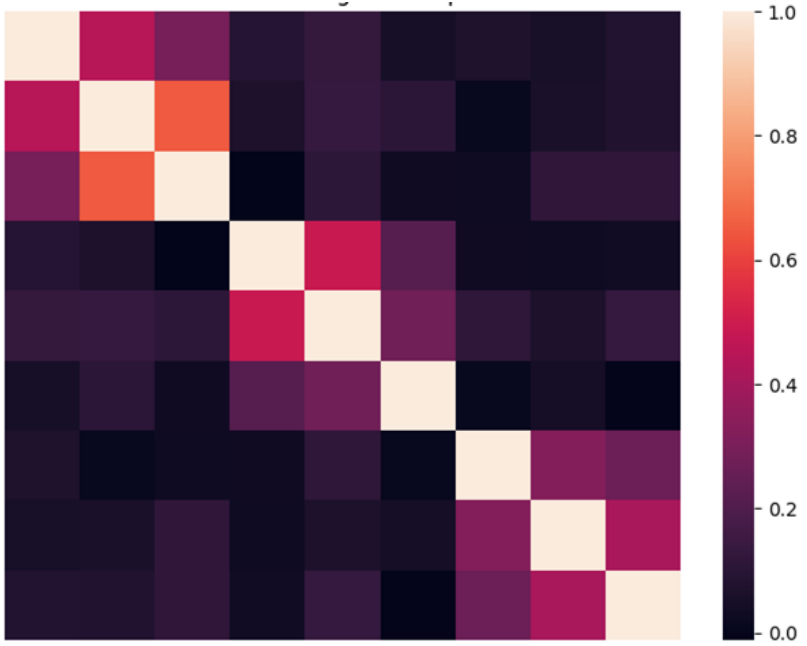
- SpacyEmbeddings

- FakeEmbeddings

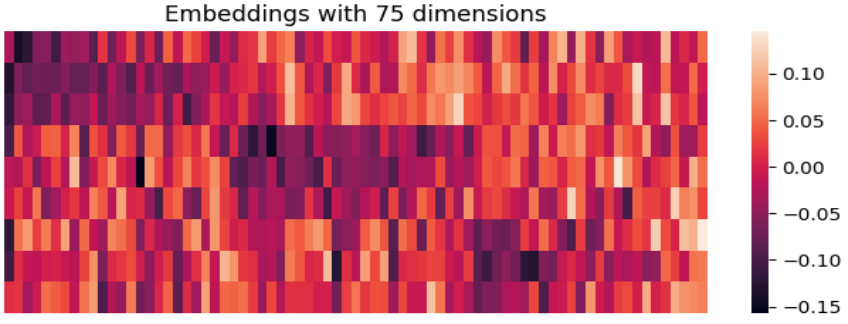


Embeddings example

1)Best travel neck pillow for long flights
2)Lightweight backpack for hiking and travel
3)Waterproof duffel bag for outdoor adventures
4)Stainless steel cookware set for cooktops
5)High-quality chef's knife set
6)High-performance stand mixer for baking
7)New releases in fiction literature
8)Inspirational biographies and memoirs
9)Top self-help books for personal growth



sentence-transformers/all-mpnet-base-v2



Large Language and Chat Models

A **large language model** is an advanced type of language model that is trained using deep learning techniques on **massive amounts of text data.**

Chat models are usually backed by a language model, but their APIs are more structured. These models take a list of Chat Messages as input, and return a Chat Message.

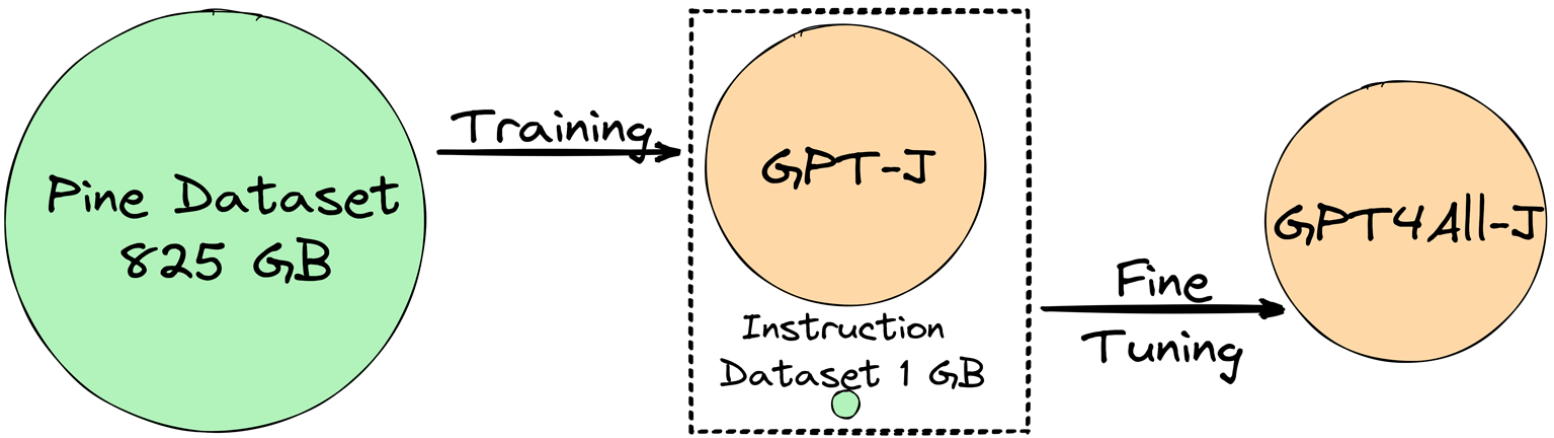
Text
<any form of text>
<article X>
<book Y>
<history Z>
etc

Prompt	Response
<stackoverflow question X>	<response X>
<Q&A platform question Y>	<Q&A platform answer Y>
<Human generated request Z>	<Human generated response Z>
<Chat history Person A>	<Chat history Person B>

Large Language and Chat Models

text (string)
"I've learned the nitrogen vacancies used in Memristors are for "switching", between excited…
"Volunteer Services Volunteer Services As Charleston Area Medical Center volunteers, our…
"Q: Python: My return variable is always None So I found a strange thing that happens in python…

prompt (string)	response (string)
" In Angular if I had a few panels, similar to tabs how do you open the panel on click and animat…	"To accomplish this in Angular, you can use ngAnimate and ng-show/ng-hide directives. Here's…
"Given the following scientific paper: This paper presents a novel approach for temporal and semanti…	"This scientific paper describes three different approaches for segmenting and analyzing edited…
" i am using the following code: <pre> <code>package Presentacion; import…	"The error is caused by a typo in the code. The correct class name is "PrintWriter" and not…



Asking GPT a question

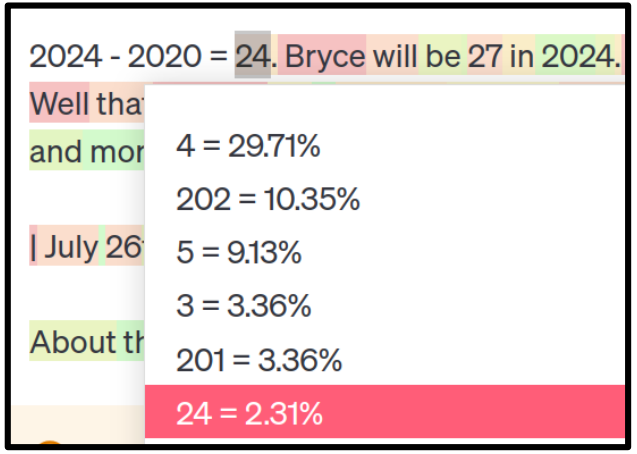
In order to ask model a question there is even no need to connect LangChain.

In this example, we're preparing the prompt, connecting to *openai* package and returning extracted response.

```python
import openai

def get_completion(prompt, model="gpt-3.5-turbo"):
    """
    This function takes a user prompt, sends it to the model, and returns the generated response.

    Parameters:
        prompt (str): The user prompt to be sent to the model.
        model (str): The name of the language model to use. Defaults to "gpt-3.5-turbo".

    Returns:
        str: The generated response from the language model.
    """

    # Prepare the user prompt in a format suitable for the OpenAI Chat API
    messages = [{"role": "user", "content": prompt}]

    # Use the OpenAI Chat API to get a completion (response) from the language model
    response = openai.ChatCompletion.create(
        model=model,
        messages=messages,
        temperature=0.0,
    )

    # Extract the generated response from the API response
    generated_response = response.choices[0].message["content"]

    return generated_response
```

Asking GPT a question

As one can see, simple questions can be answered by the model correctly.
But for complicated requests LangChain may be the way to go.




```
get_completion("What do they call a Big Mac in France?")

>> 'In France, a Big Mac is called "Le Big Mac."'
```

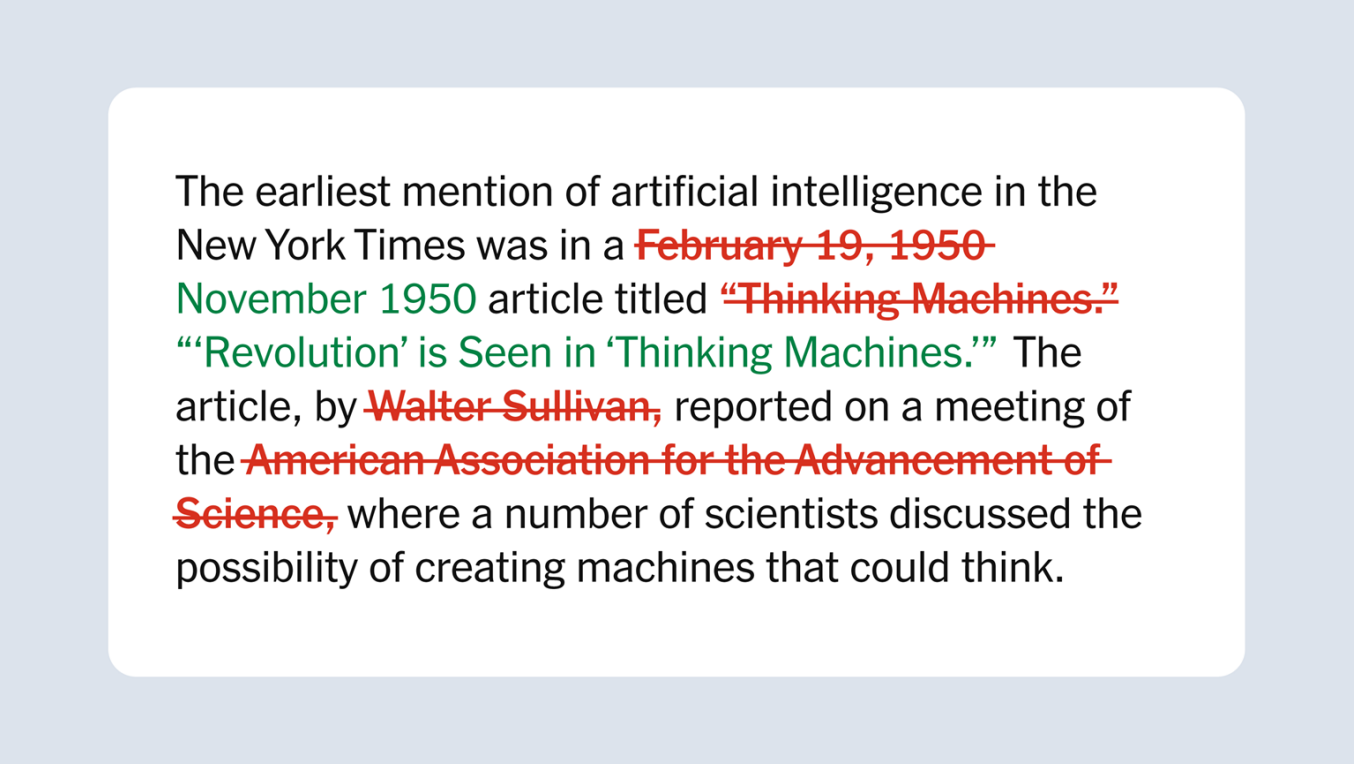
```
get_completion("What is the average age of retirement of the last 5 US presidents?")

>> The average age of retirement of the last 5 US presidents is approximately 77 years old. Here are the retirement ages of the last 5 presidents:

1. Donald Trump - Retired at the age of 74 in 2021.
2. Barack Obama - Retired at the age of 55 in 2017.
3. George W. Bush - Retired at the age of 62 in 2009.
4. Bill Clinton - Retired at the age of 54 in 2001.
5. George H. W. Bush - Retired at the age of 68 in 1993.

Adding up these retirement ages and dividing by 5 gives an average retirement age of 77.
```

Basics of LangChain

Hallucinations

Hallucinations can be understood as answering confidently when asked for the weather forecast in a fictional city or providing fabricated references in an academic context

But what are the reasons of that?

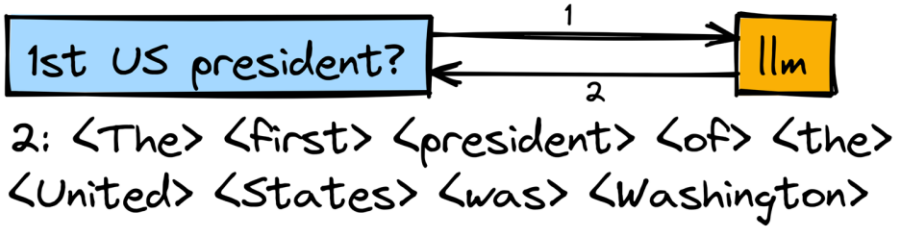


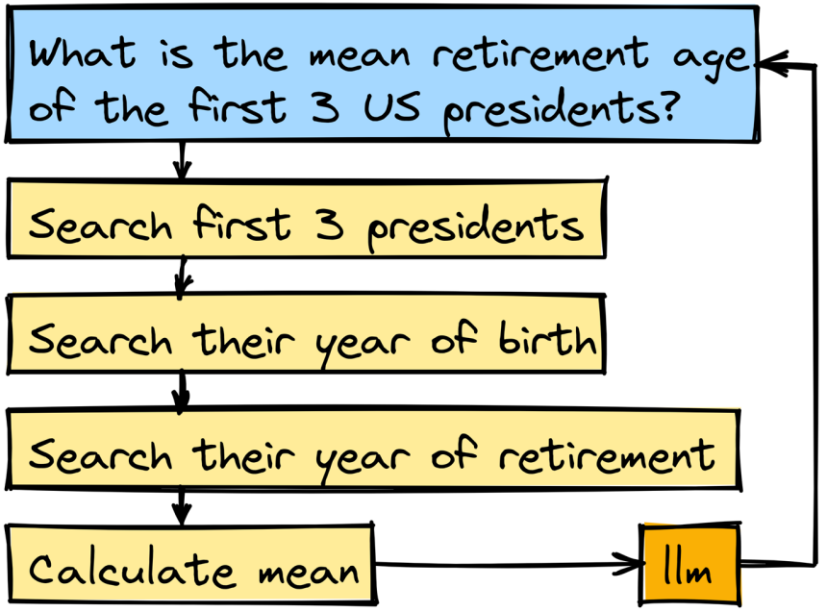
LLM LangChain Application

LLMs are used in LangChain applications in several purposes:

- language model to answer user response

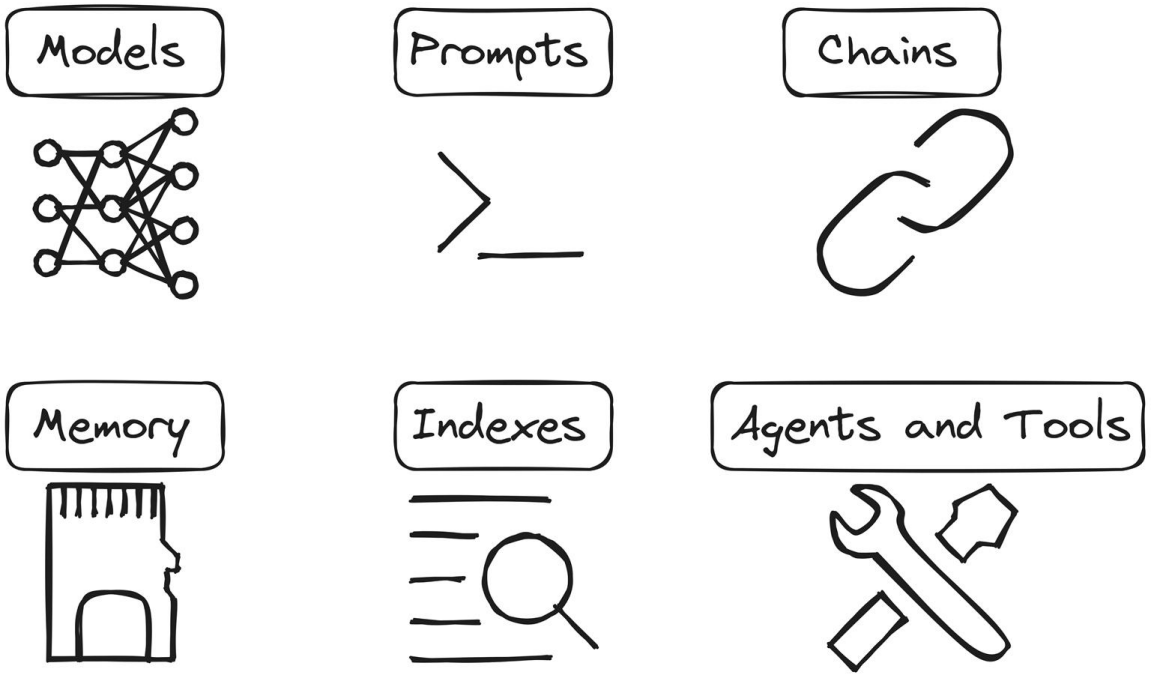
- “llm butter” for LangChain components





What is LangChain? Quick glance

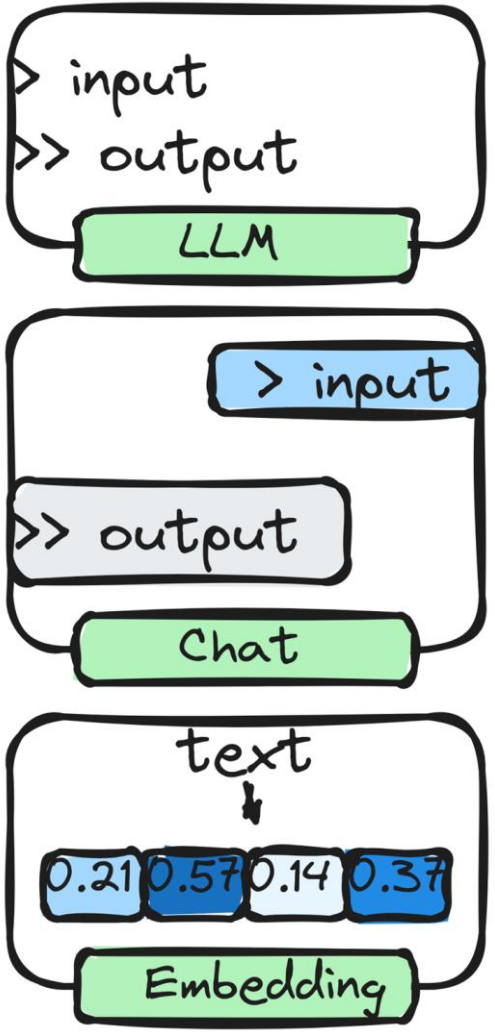
LangChain is a framework that makes it easier to create applications using large language models (LLMs).



Models: reminder

LangChain supports a variety of different models, so you can choose the one that suits your needs best

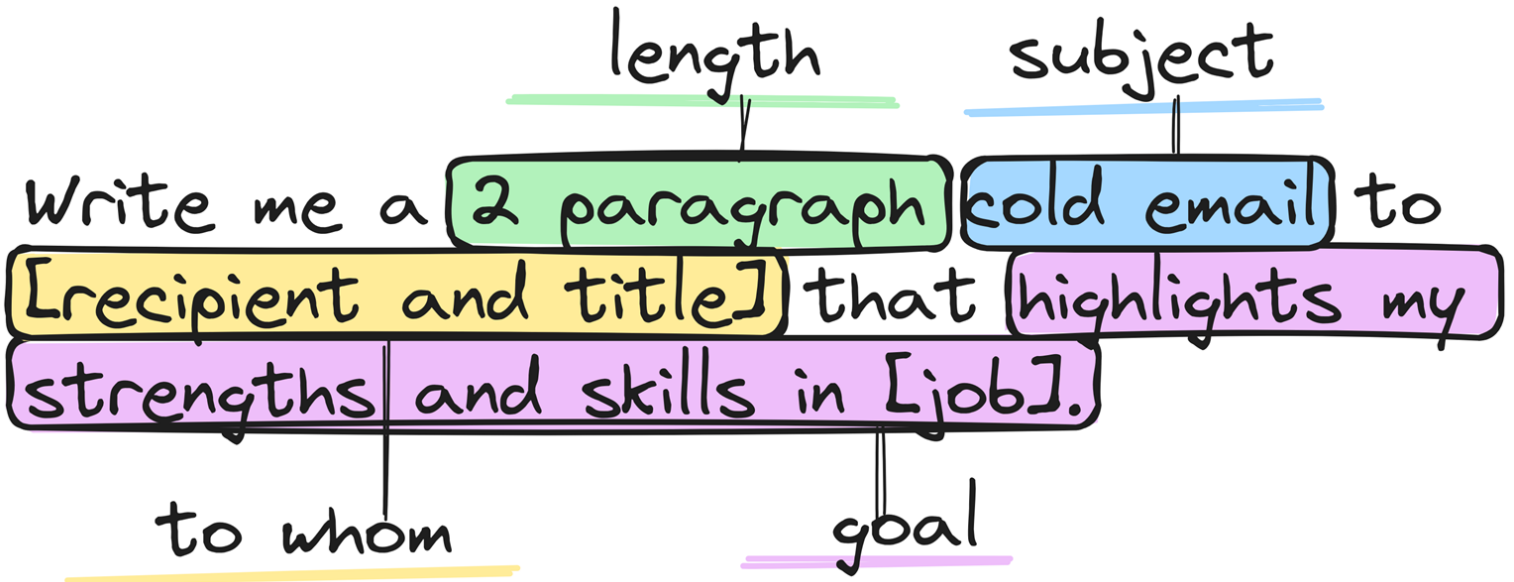
- **Language Models** are used **for generating text**
    - LLMs utilize APIs that take input text and generate text outputs
    - ChatModels employ models that process chat messages and produce responses.
- **Text Embedding Models** convert text into **numerical representations**



Prompts

Prompts are the instructions that you give to the LLM. They tell the LLM what you want it to do, and how you want it to do it.

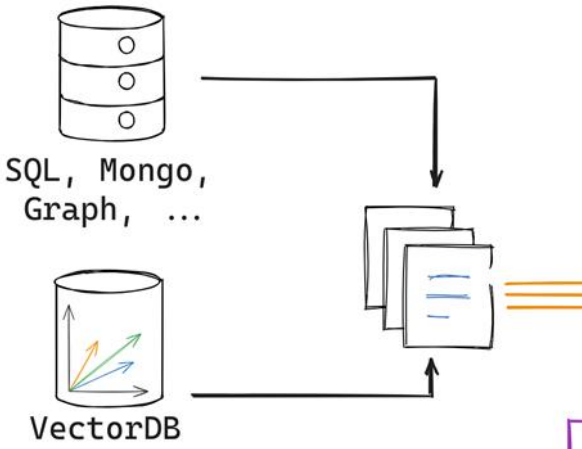
Prompts can take the form of a string (for Language Models) or a list of messages (for Chat Models).

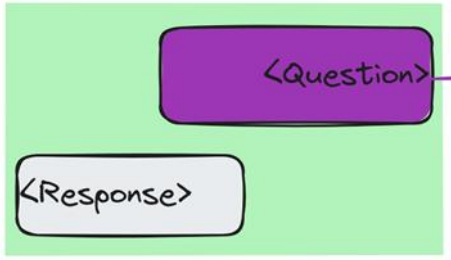


Prompts

```python
def setup_rag(**kwargs):
    retriever = chroma_db.as_retriever(**kwargs)

    template = """Answer the question based only on the following context:
    {context}

    Question: {question}
    """
    prompt = ChatPromptTemplate.from_template(template)
    output_parser = StrOutputParser()

    setup_and_retrieval = RunnableParallel(
        {"context": retriever, "question": RunnablePassthrough()}
    )
    rag_chain = setup_and_retrieval | prompt | llm | output_parser
    return rag_chain
```

Prompts

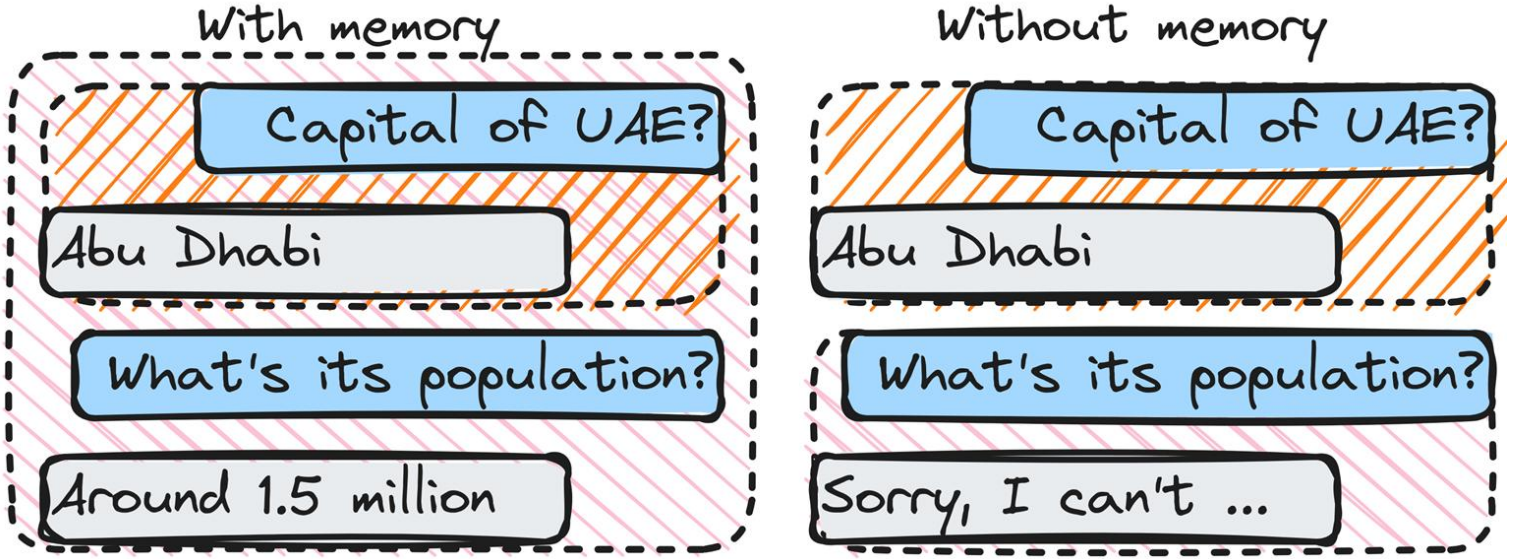





```python
def setup_rag(**kwargs):
    retriever = chroma_db.as_retriever(**kwargs)
    template = """ You are an expert in ...
    Answer the question based only on the following context:
    {context}

    Question: {question}
    """
    prompt = ChatPromptTemplate.from_template(template)

    output_parser = StrOutputParser()
    setup_and_retrieval = RunnableParallel(
        {"context": retriever,"question": RunnablePassthrough()}
    )
    rag_chain = setup_and_retrieval | prompt | llm | output_parser
    return rag_chain
```

Memory

LangChain introduces memory components that enable the retention and utilization of past interactions. By default, LangChain components are stateless, treating each query independently. Memory in LangChain allows for the storage and management of previous chat messages.



Vectorstores

```python
from langchain.vectorstores import DocArrayInMemorySearch

db = DocArrayInMemorySearch.from_documents(
    documents=documents,
    embedding=embedding
)

docs = db.similarity_search(query)

#or

docs = db.similarity_search_with_score(query)
```

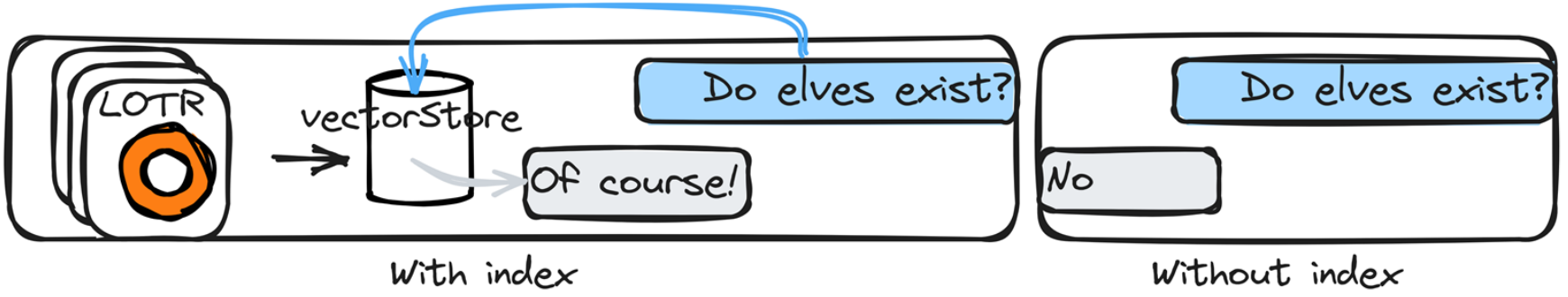
Chroma

```python
from langchain.vectorstores import Chroma

chroma_db = Chroma.from_documents(
    documents=documents,
    embedding=openai_embedding,
    persist_directory=persist_directory
)

docs = chroma_db.similarity_search_with_score(query,k=3)
```

Indexes

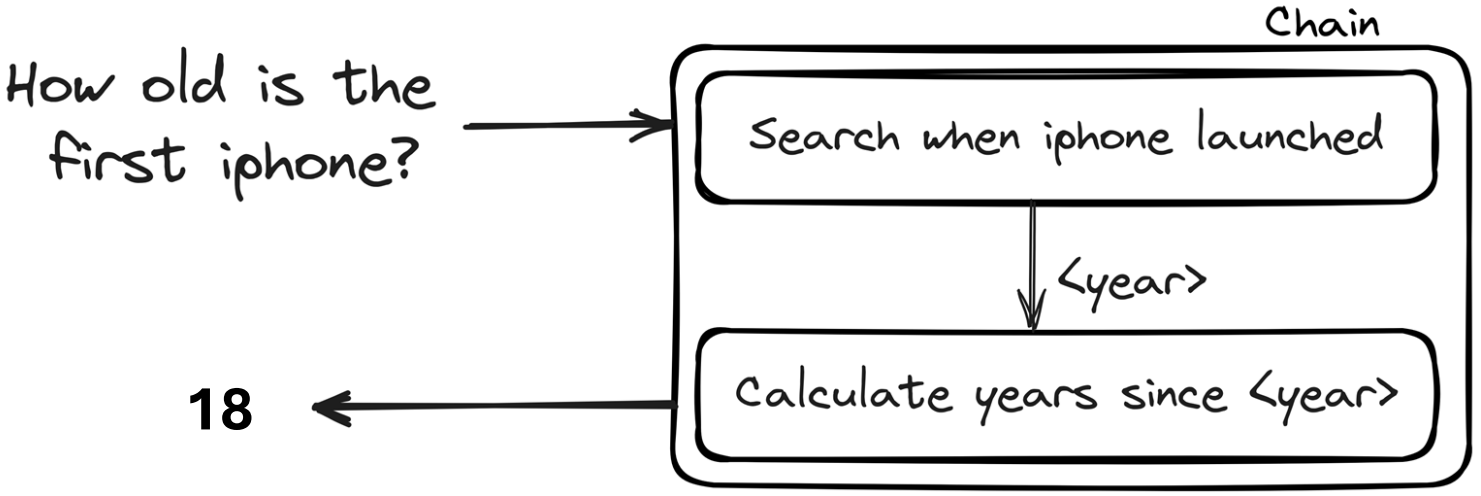
Indexes are databases of information that can be used to provide context to the LLM.

For example, if you are asking the LLM to answer a question about a particular topic, you could use an index to provide the LLM with a list of relevant articles or websites.



Chains

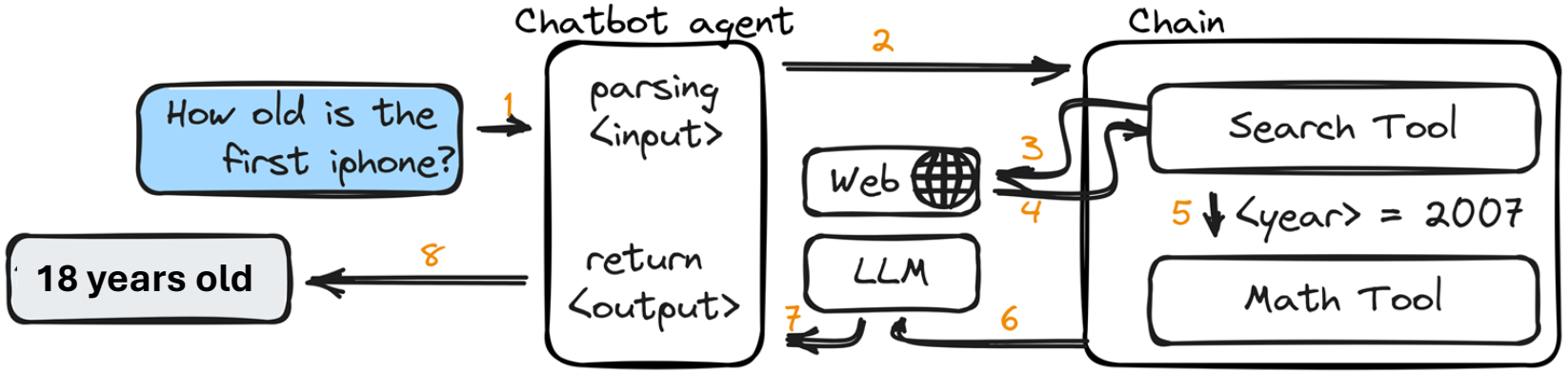
Chains are sequences of calls to the LLM. They allow you to perform complex tasks by chaining together multiple calls to the LLM. There are multiple types of chains to solve different cases.



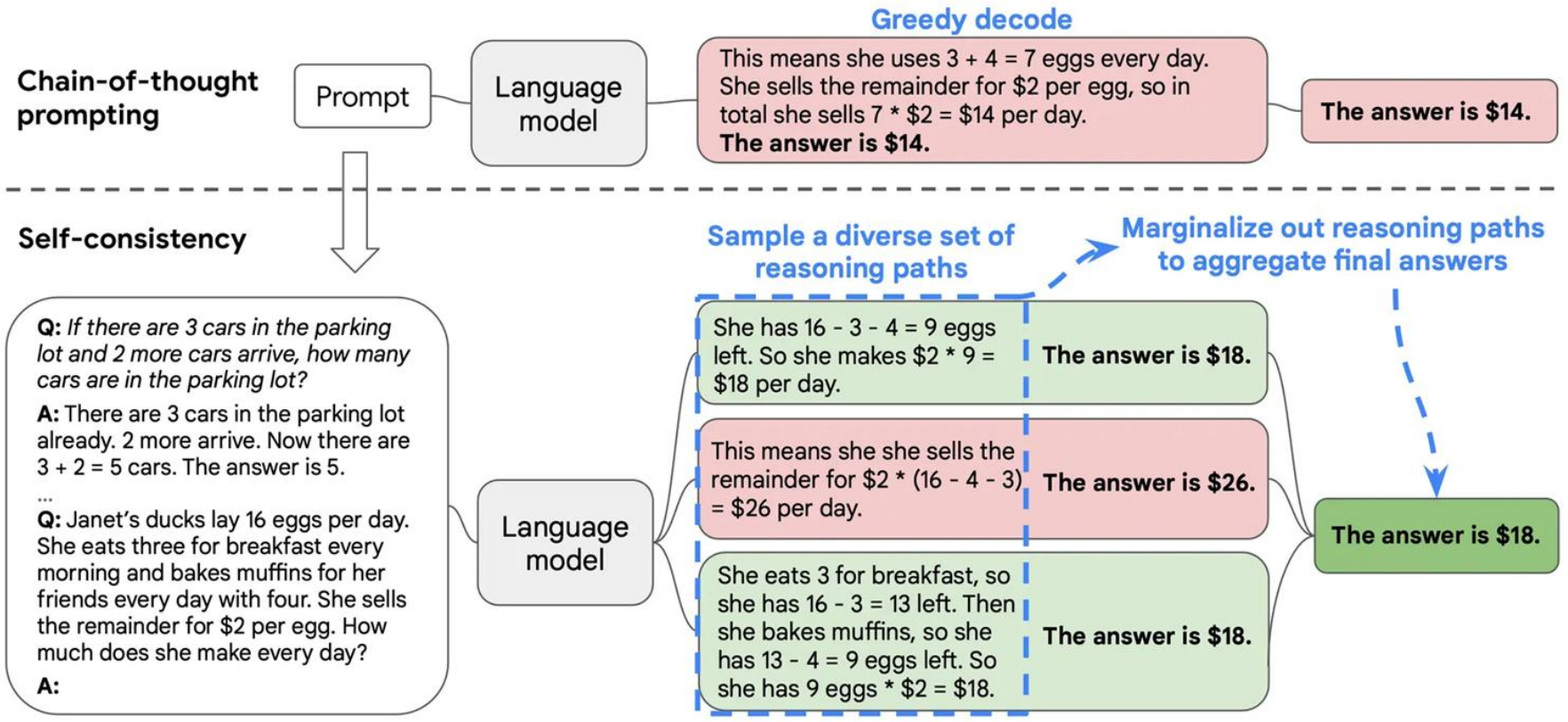
Agents and Tools

Agents are objects that manage interaction between the user and the LLM. They handle things like *parsing input*, *generating output* and determining the sequence of actions to follow and tools to use.

A tool is a function designed to perform a specific task. Examples of tools can range from *Google Search*, *DB lookups*, *API calls*, to other chains. The standard interface for a tool is a function that accepts a string as input and returns a string as output.



Agents and Tools: Prompt Alternative



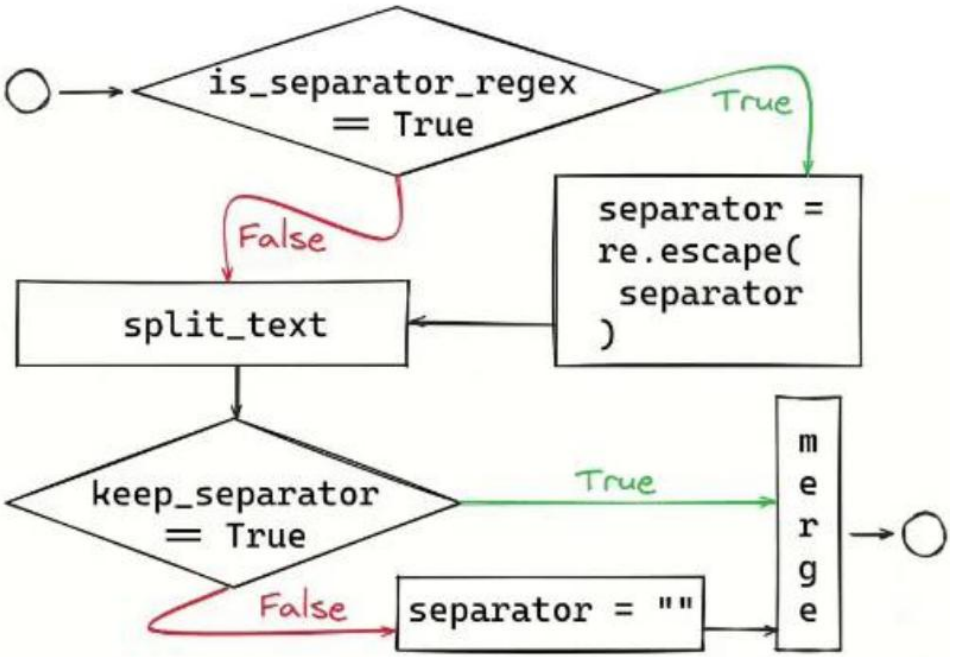
Tools: Data Loaders

Tools: Data Loaders

```python
# Example of loading and using document loaders
def load_document_from_url(url: str, filename: str, loader_class: Any) -> Any:
    """Download a document from a URL, save it, and load it using a specified loader class.

    Args:
        url: URL of the document to download.
        filename: Filename to save the downloaded document.
        loader_class: The class of loader to use for loading the document.

    Returns:
        The loaded document.
    """
    download_and_save_file(url, filename)
    loader = loader_class(filename)
    return loader.load()

# Load and process documents
text_document = load_document_from_url(text_url, "planets.txt", TextLoader)
csv_document = load_document_from_url(csv_url, "insurance.csv", CSVLoader)
pdf_document = load_document_from_url(pdf_url, "dubai_law.pdf", PyPDFLoader)
```

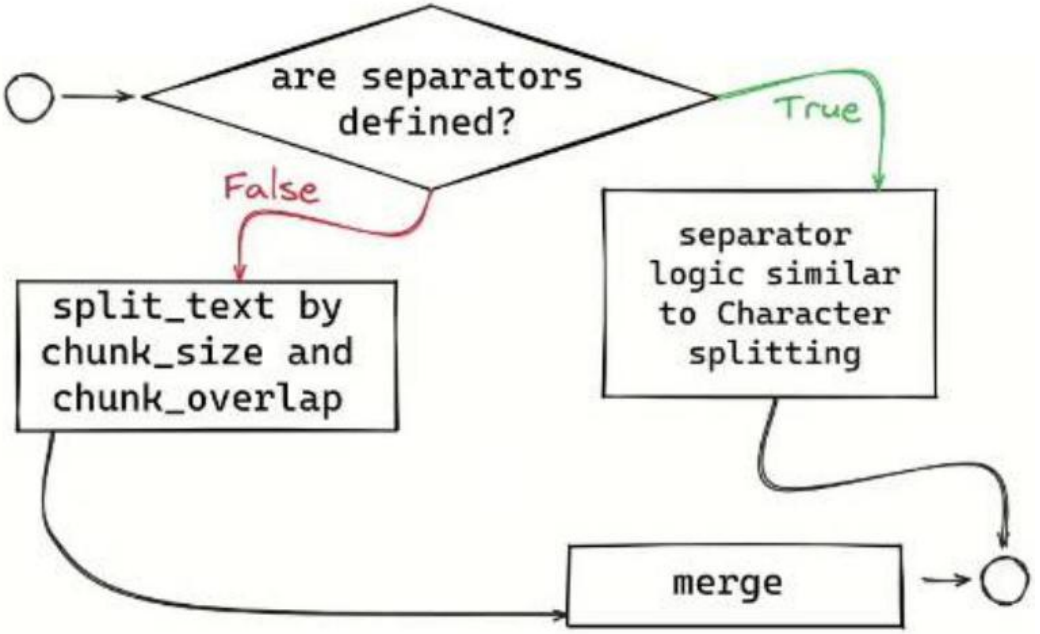
Tools: Data Splitters

```
c_splitter = CharacterTextSplitter(
    separator = ".",
    keep_separator = True
)
```



Tools: Data Splitters

```
r_splitter = RecursiveCharacterTextSplitter(
    chunk_size = 5
    chunk_overlap = 2
)
```



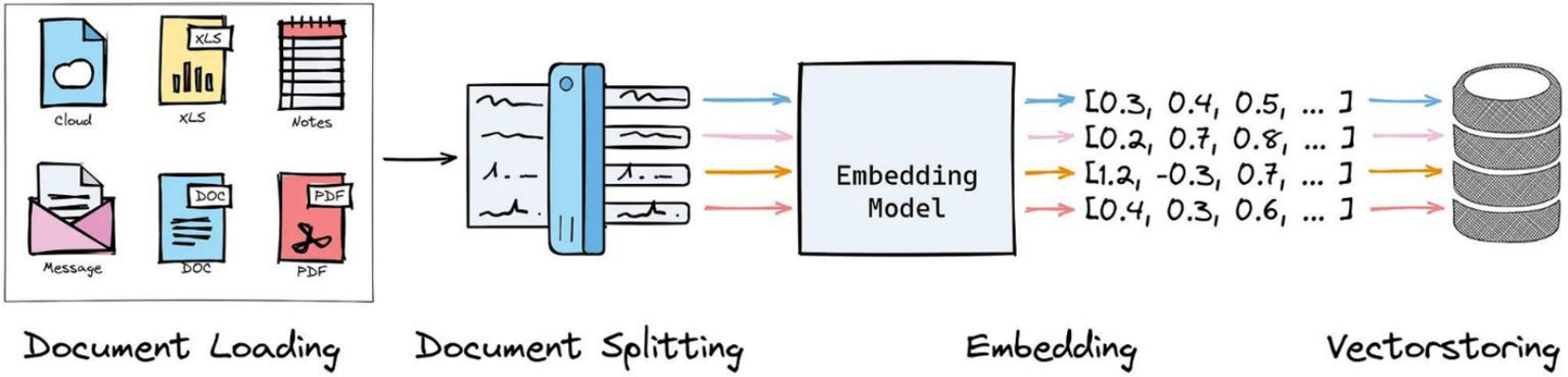

```python
text_splitter = RecursiveCharacterTextSplitter(
    separators=["\n"], chunk_size=250, chunk_overlap=0, keep_separator=False
)
# split_text
chunks = text_splitter.split_text(text_document[0].page_content)

#split_documents
chunks = text_splitter.split_documents(text_document)
```

Tools: Data Splitters

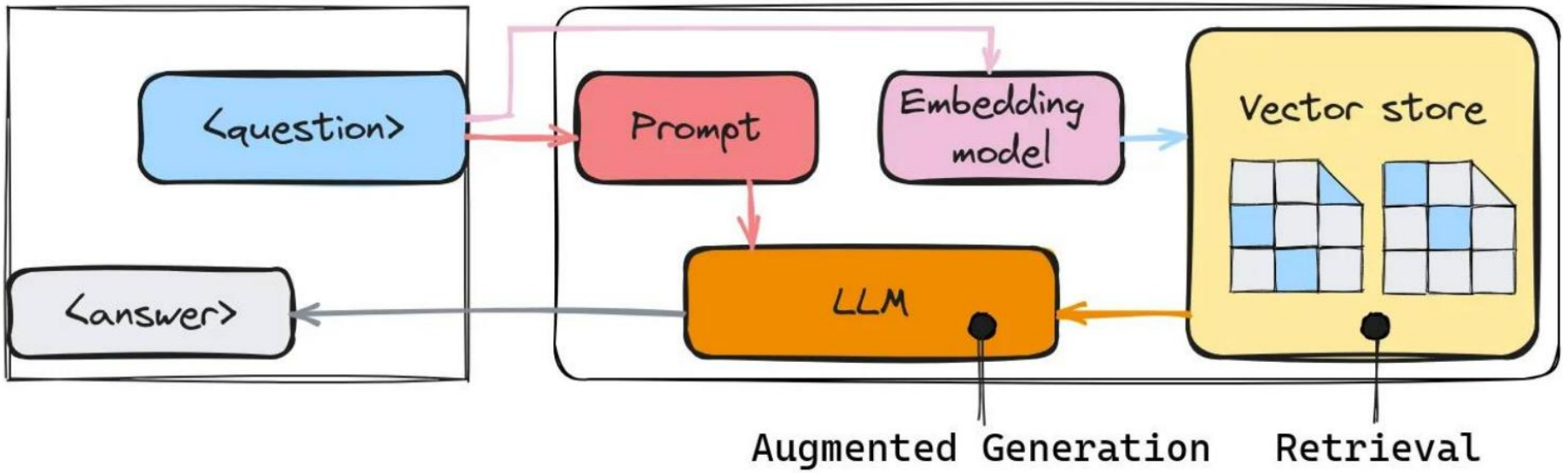
Recursive	A list of user defined characters	Recursively splits text. Splitting text recursively serves the purpose of trying to keep related pieces of text next to each other. This is the recommended way to start splitting text.
HTML	HTML tags	Splits text based on HTML tags. Notably, this adds in relevant information about where that chunk came from (based on the HTML)
Markdown	Markdown specific characters	Splits text based on Markdown-specific characters. Notably, this adds in relevant information about where that chunk came from (based on the Markdown)
Code	Code specific characters.	Splits text based on characters specific to coding languages. 15 different languages are available to choose from.
Token	Tokens	Splits text on tokens count with a specified method to measure tokens.
Character	A user defined character	Splits text based on a user defined character. One of the simpler methods.
Semantic Chunker	Sentences	First splits on sentences. Then combines ones next to each other if they are semantically similar enough

Data Preprocessing Pipeline

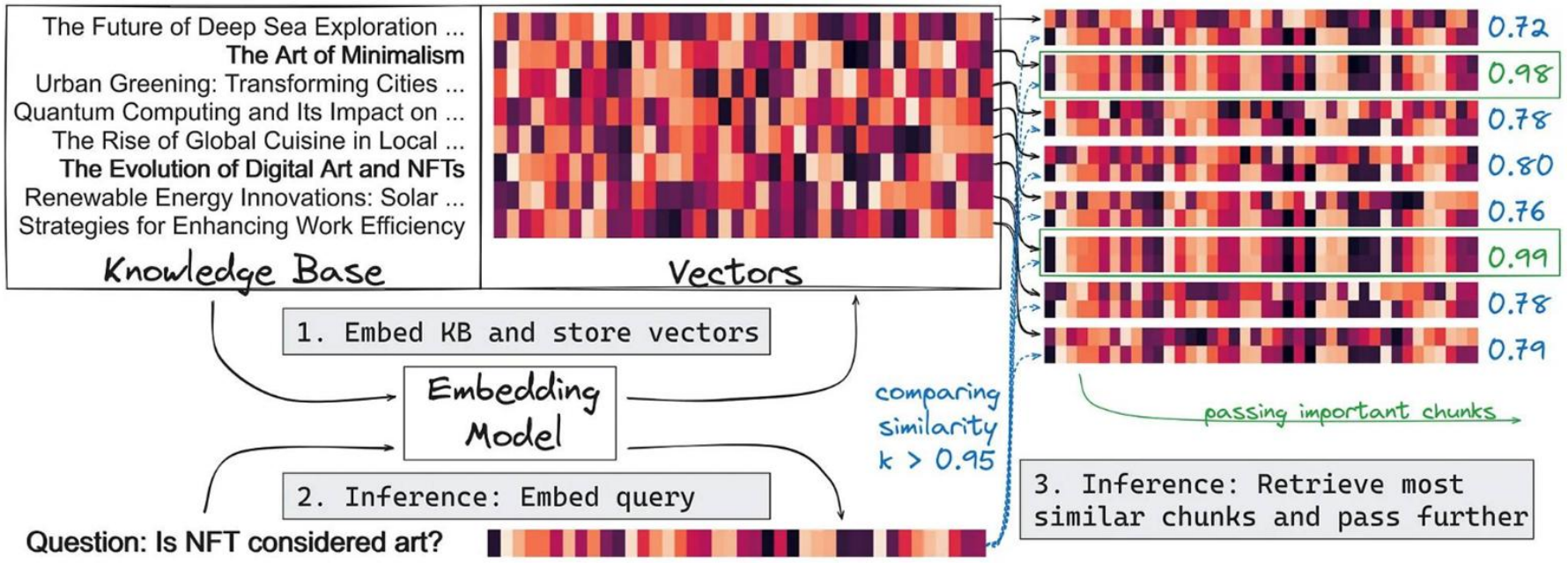
1) Load a document
2) Split the document
3) Embed the split chunks
4) Store the chunks



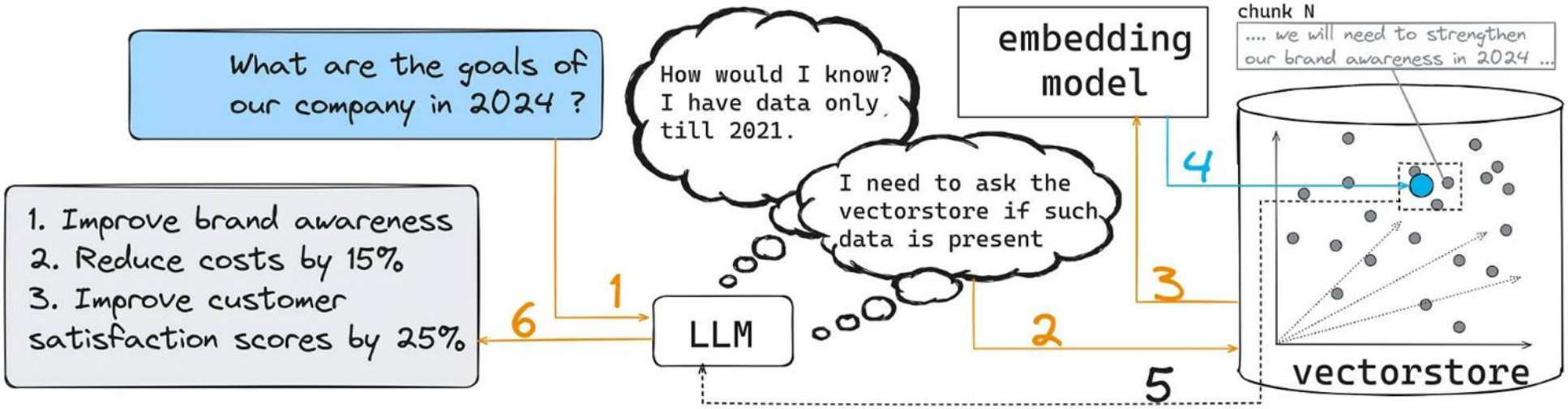
Retrieval-Augemented Generation (RAG)



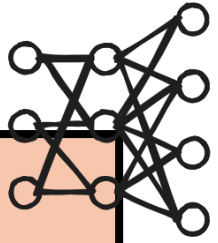
RAG Pipeline



RAG Pipeline

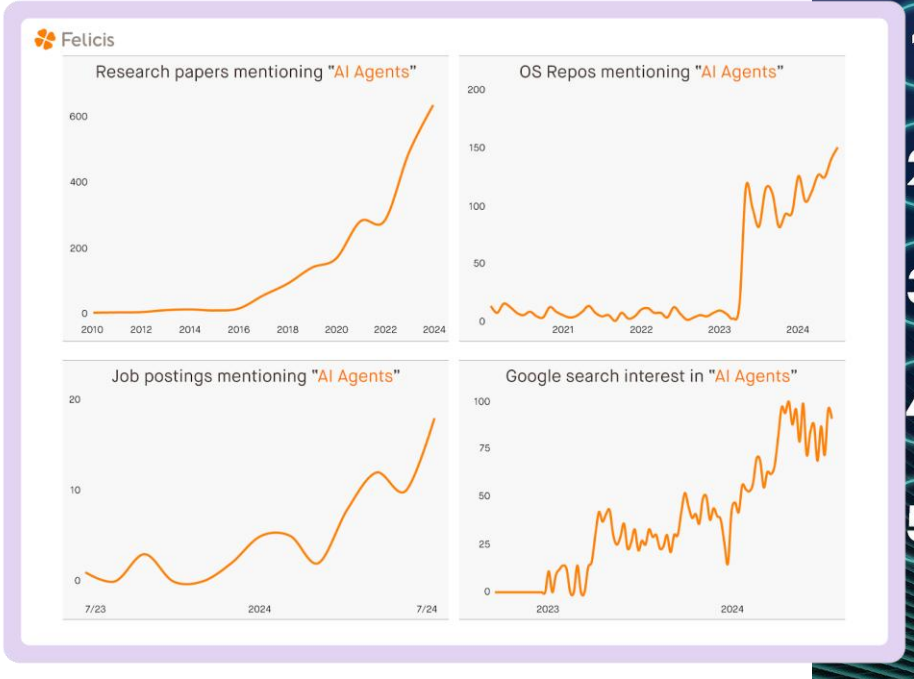


Practice with LangChain



Agentic AI

Trends


Felicis
Research papers mentioning "AI Agents"
600
400
200
0
2010
2012
2014
2016
2018
2020
2022
2024
OS Repos mentioning "AI Agents"
200
150
100
50
0
2021
2022
2023
2024
Job postings mentioning "AI Agents"
20
10
0
7/23
2024
7/24
Google search interest in "AI Agents"
100
75
50
25
0
2023
2024


2025 Top 10 Strategic Technology Trends
1 Agentic AI
2 AI Governance Platforms
3 Disinformation Security
4 Post-Quantum Cryptography
5 Ambient Invisible Intelligence
6 Energy-Efficient Computing
7 Hybrid Computing
8 Spatial Computing
9 Polyfunctional Robots
10 Neurological Enhancement
Source: Gartner
© 2024 Gartner, Inc. and/or its affiliates. All rights reserved. 3125012
Gartner

The Three Waves of AI

Predictive AI

Focus on analyzing data to predict outcomes.
Analyze past data to predict future outcomes.
Why It Matters: Enabled data-driven decision-making.

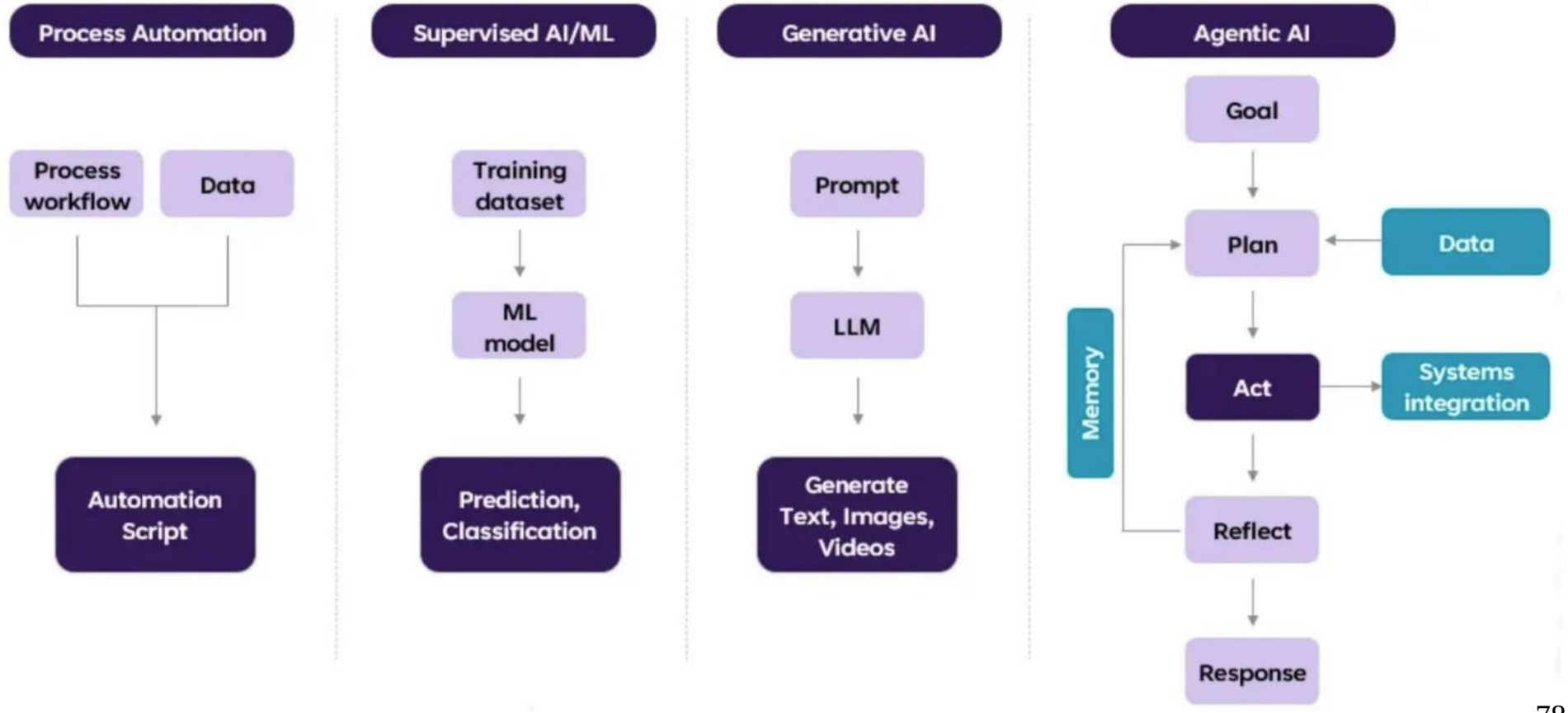
Generative AI

Focus on creating content from data.
Creating content (text, images, code, videos).
Why It Matters: Empowered creativity and productivity.

Agentic AI

Focus on autonomous actions and learning iteratively.
Autonomous actions, environment interaction, iterative learning.
Why It Matters: AI becomes proactive, managing complex tasks.

The Three Waves of AI



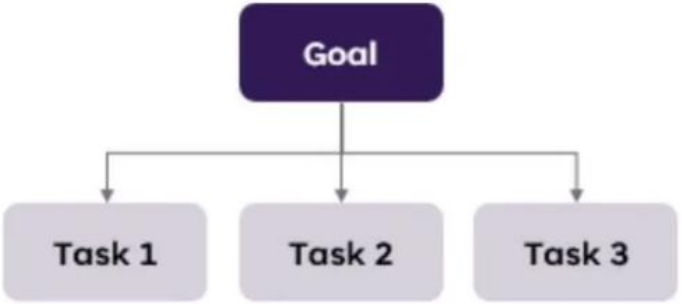
Agentic AI capabilities: Task Decomposition

Task decomposition

Given a complex user task, the system generates a plan to fulfill the request depending on the capabilities of available agents at run-time.

Chain-of-Thought (CoT)

CoT is the most widely used decomposition framework today to transform complex tasks into multiple manageable tasks and shed light into an interpretation of the model's thinking process.



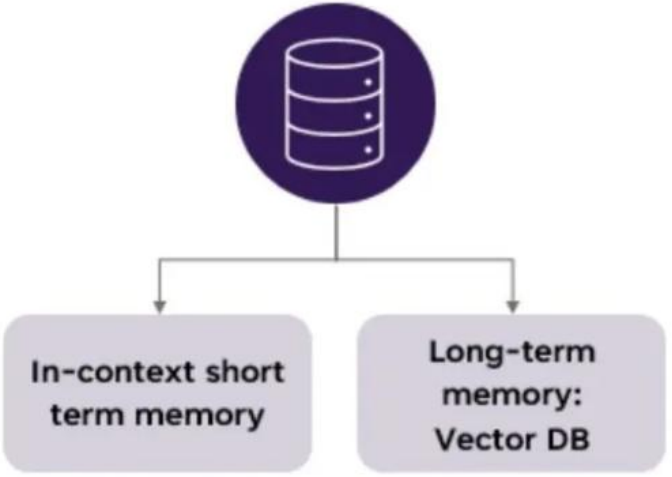
Agentic AI capabilities: Memory Management

Memory management

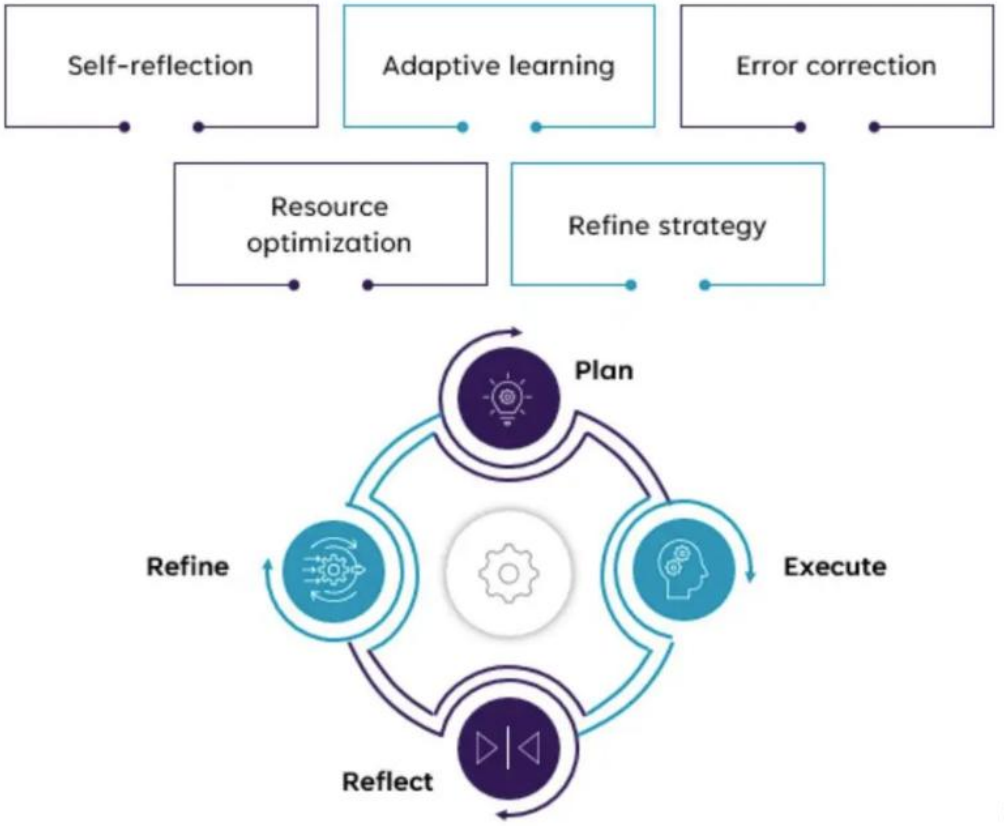
Memory management is key for Agentic AI systems for context sharing between tasks and maintaining execution context over long periods.

Long term memory

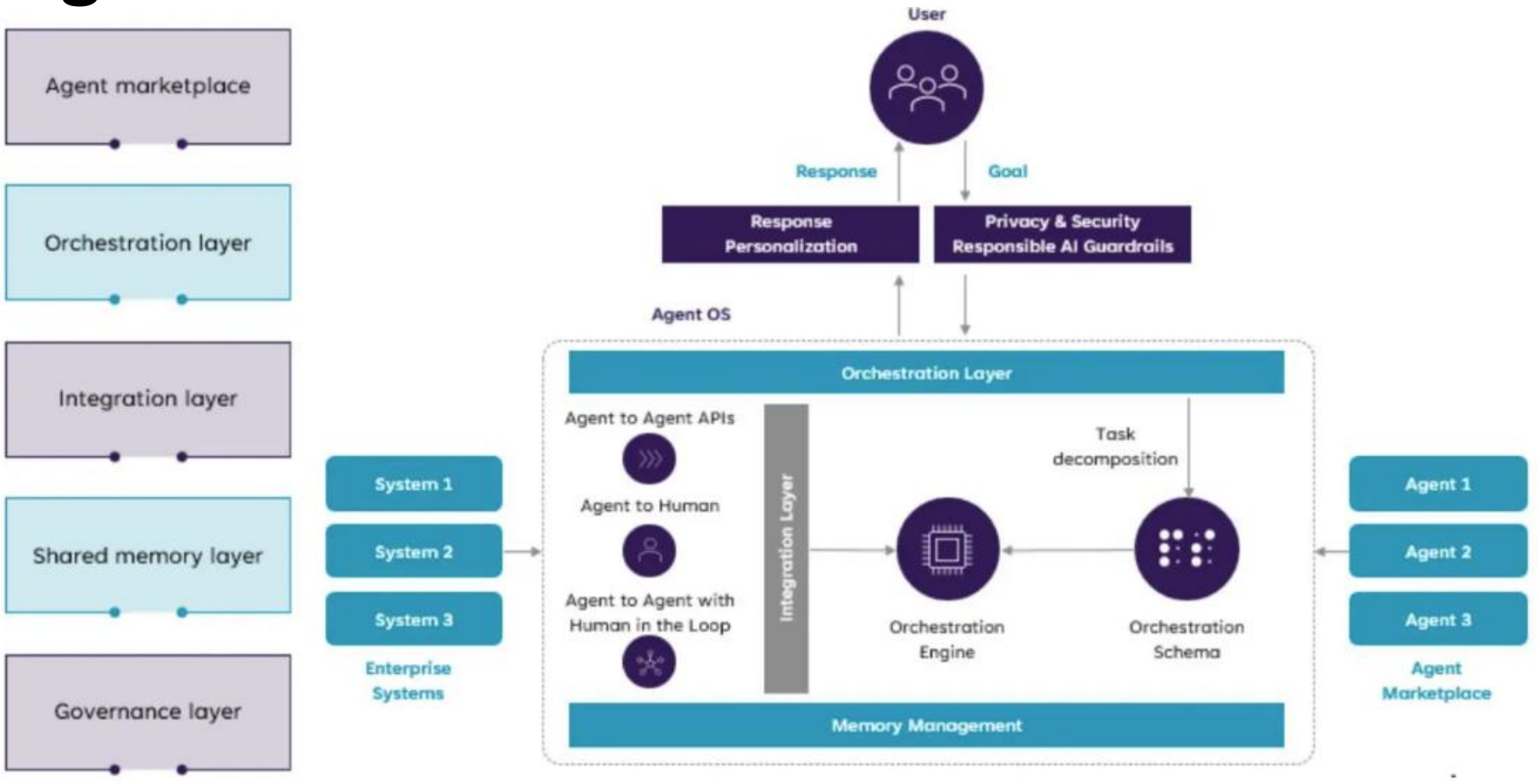
Memory management becomes challenging when agents execute for extended period as their memory is typically limited to their context. The solution is to use Vector DBs to store agent memory externally, retrieving it as needed.



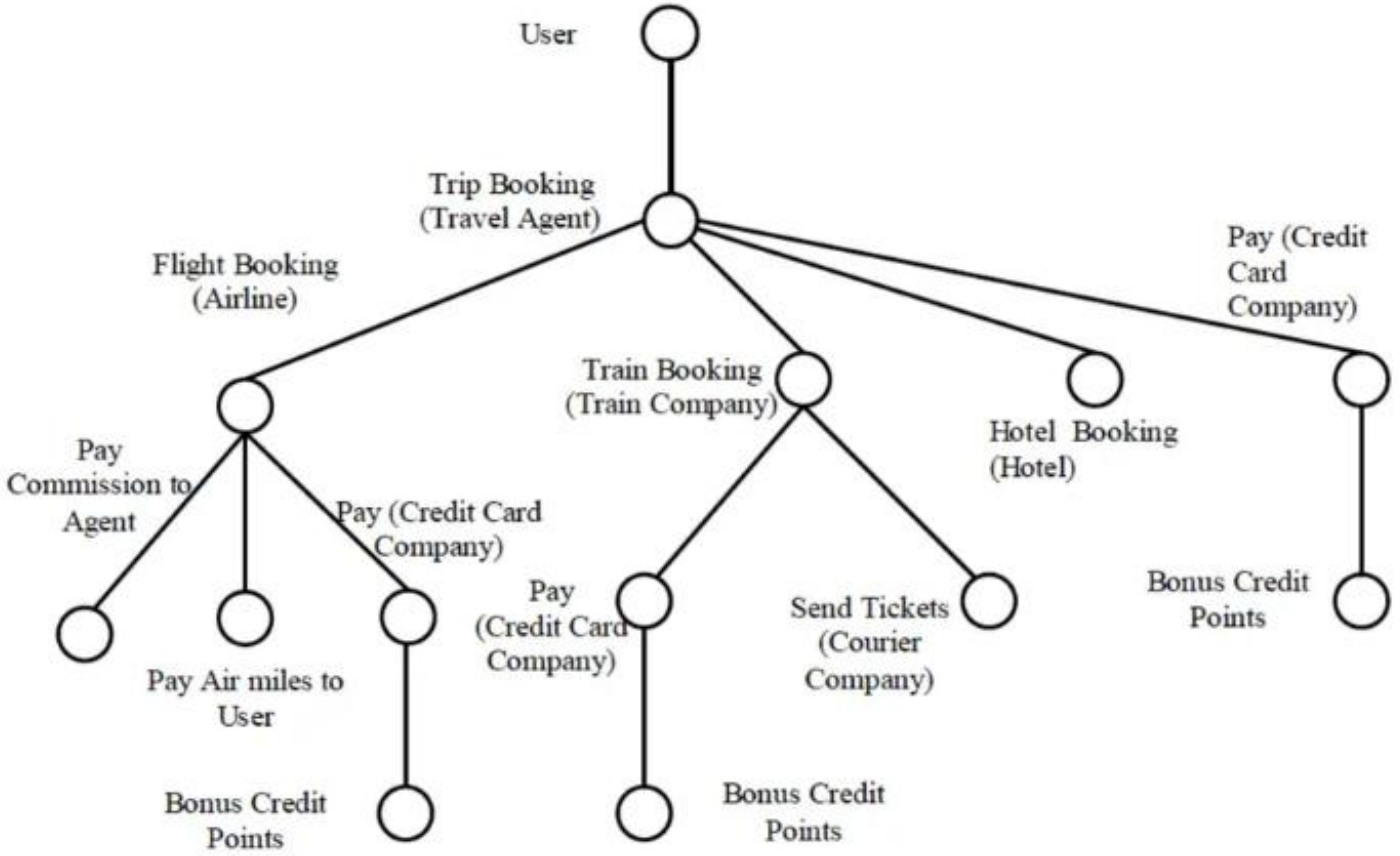
Agentic AI capabilities: Reflect and Adapt



Agentic AI Infrastructure

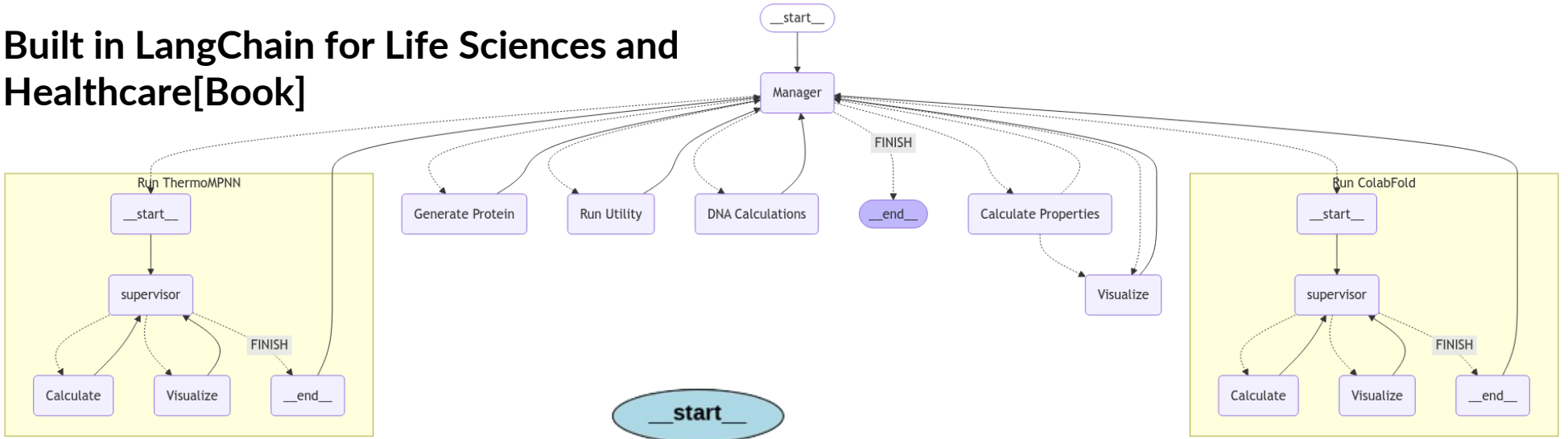


Decomposing a Comple Agentic AI Task

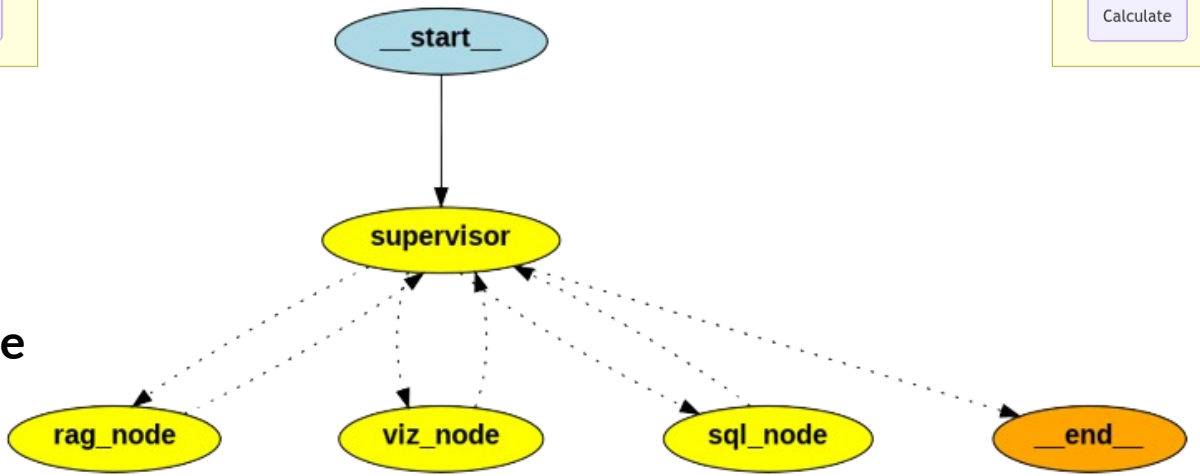


Decomposing a Comple Agentic AI Task

Built in LangChain for Life Sciences and Healthcare[Book]



Built this course



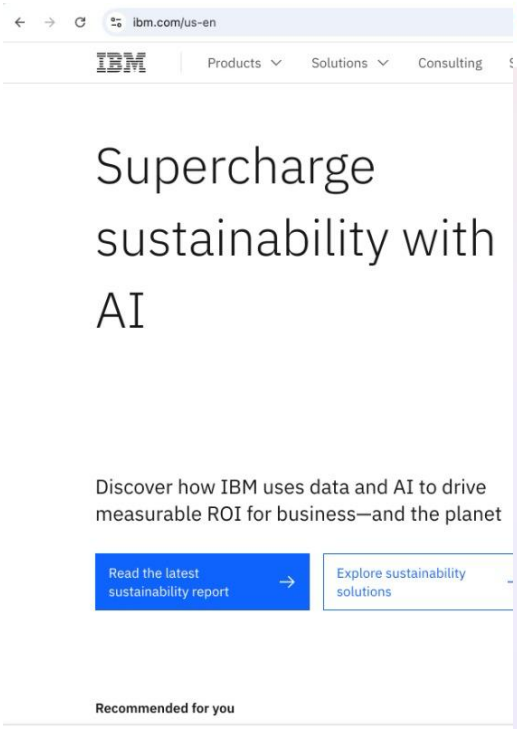
Tool Calling: A Core Feature of Agentic AI

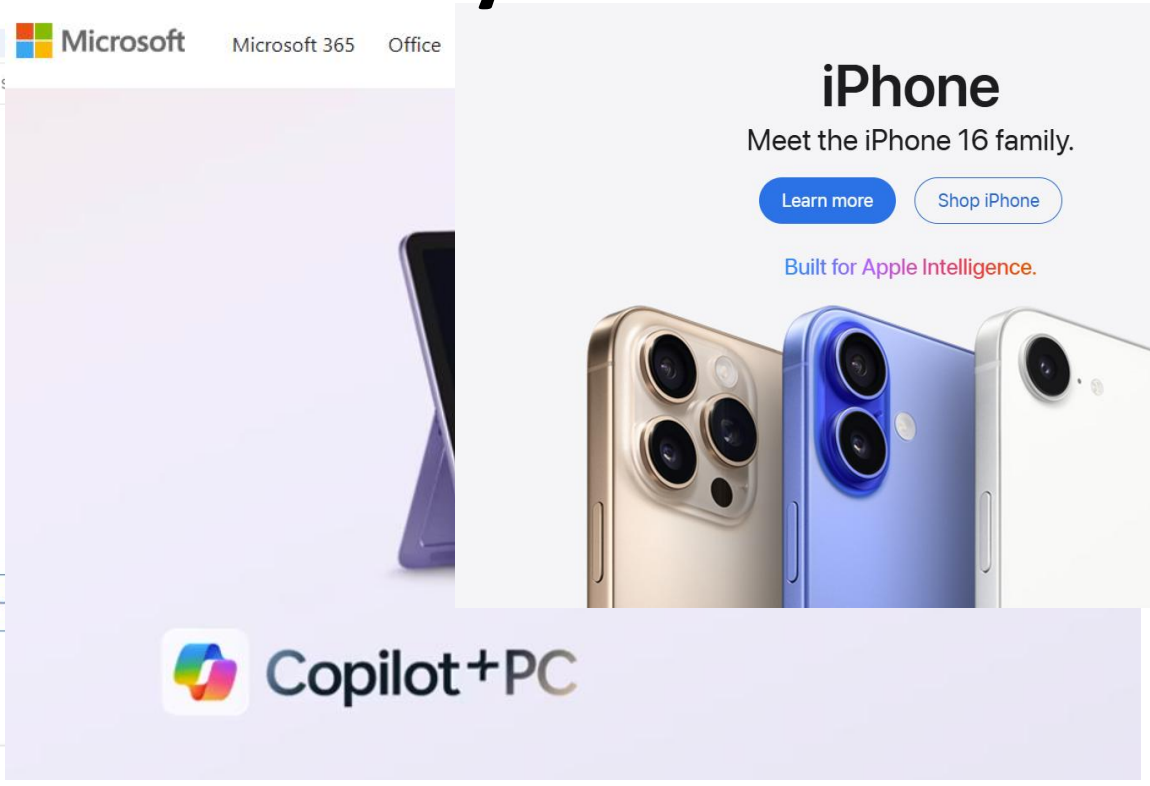
Tool calling enables AI agents to autonomously select and utilize external tools or APIs to accomplish tasks beyond their inherent capabilities.

Significance in Agentic AI:
•Enhances problem-solving by allowing AI to access specialized resources.
•Facilitates dynamic decision-making through real-time data retrieval and processing.
•Expands the functional scope of AI agents, enabling them to perform complex, multi-step operations.

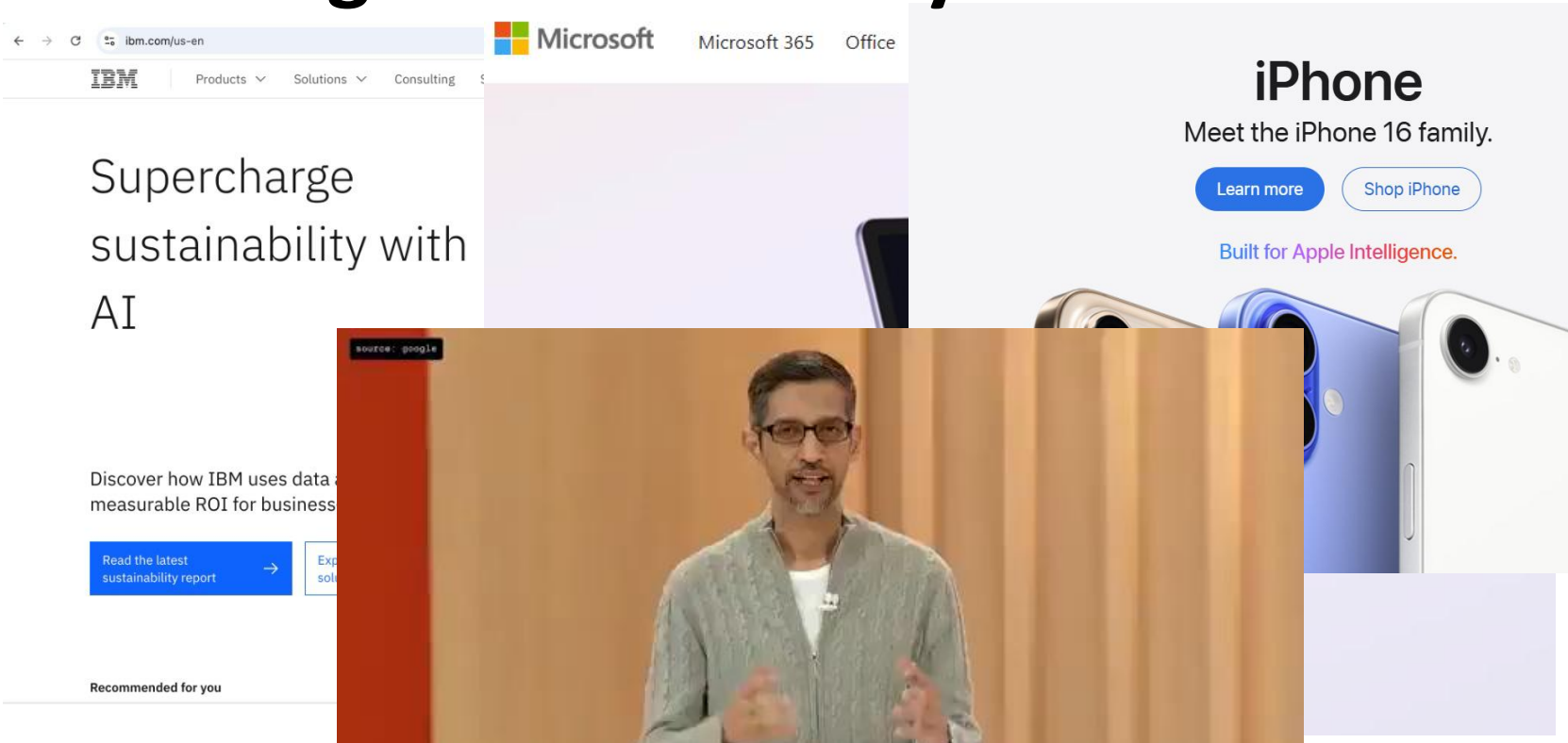
Example:
An AI agent tasked with planning a trip can autonomously access flight booking APIs, hotel reservation systems, and local event databases to create a comprehensive itinerary.

AI and Agentic AI is everywhere





AI and Agentic AI is everywhere



L4. Innovators. What will come after Agentic AI?

Description: At this level, AI systems are capable of aiding in invention and discovery. They can generate new ideas, propose novel solutions, and assist in creative processes.

Example: An AI that helps scientists discover new materials by analyzing vast amounts of data and suggesting potential compounds for experimentation.

L5. Organizations. What will come after Agentic AI?

Description: AI systems at this level can perform the work of an organization. They can manage operations, make strategic decisions, and coordinate complex tasks autonomously.

Example: An AI that runs a company, handling everything from supply chain management to employee scheduling and financial planning.

LangGraph - Build Agents with Control

- **Orchestration framework for controllable agentic workflows**
- **Gain precision and control to build agents that reliably handle complex tasks.**
- **Controllable cognitive architecture for any task**
- **Autonomous, but well-behaved multi agent workflows**

Practice with Agentic AI