An Introduction to Large Language Models (LLMs)

Brent McPherson, PhD

2024-05-17

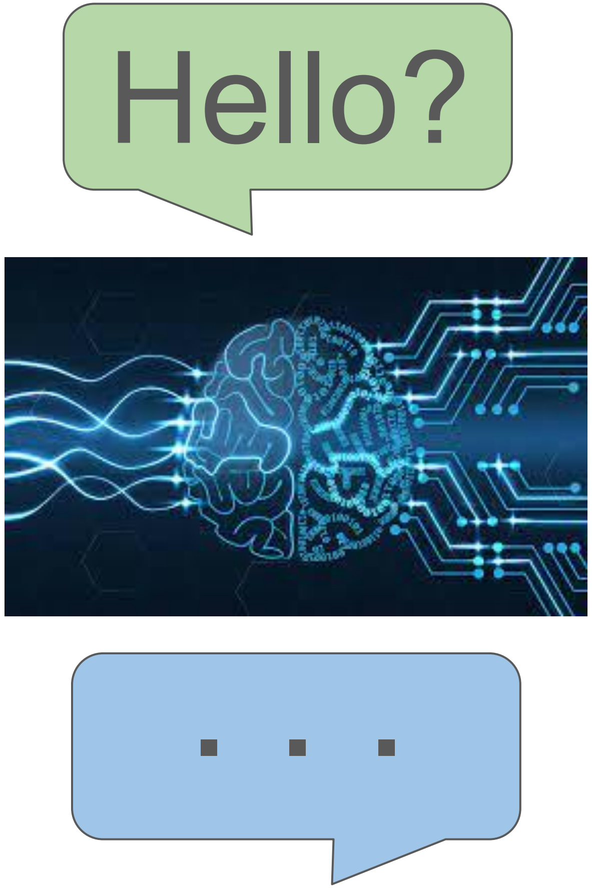
neuro Montreal Neurological Institute-Hospital

ORIGAMI Lab

McGill UNIVERSITY

Topics

- What are Large Language Models?
- The Impact of LLMs
- Using an LLM Programmatically
    - Some different tools
    - Retrieval Augmented Generation
- What models should we use?

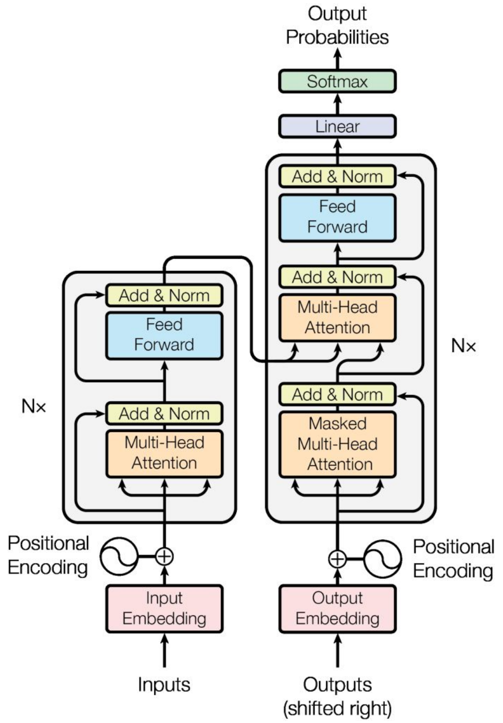


https://www.forbes.com/sites/garydrenik/2023/01/11/large-language-models-will-define-artificial-intelligence/

What are Large Language Models?

What are Large Language Models (LLMs)?

- Massive (billions of parameters) deep learning models.
    - Mostly transformers (encoder / decoder)
- They are trained on a massive corpus (trillions of words) to probabilistically generate responses.
- Conventionally, they provide a plain language interface to interact with them.
    - No “coding” required



https://www.wisecube.ai/blog/a-comprehensive-overview-of-large-language-models/

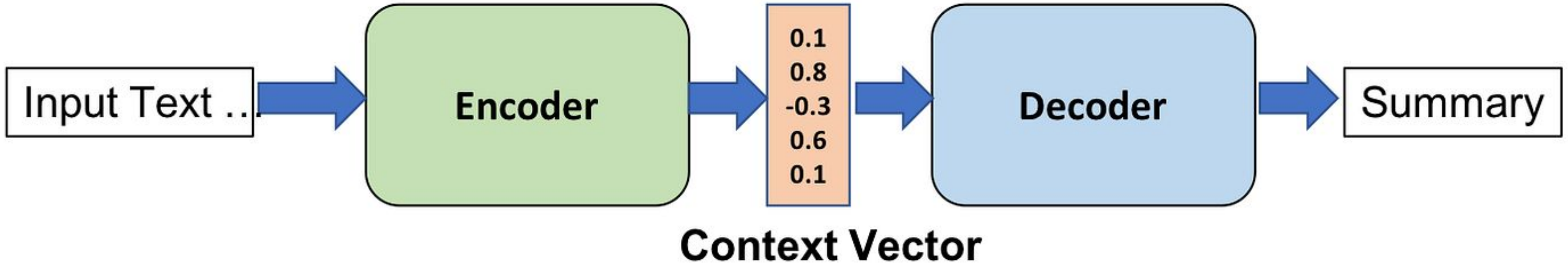
The Parts of an LLM

- Tokenization
  - splitting the input / output texts into smaller units that can be processed by the LLMs.
- Embedding
  - a high dimensional encoding of tokens that represents their semantic meaning.
- Attention
  - LLMs selectively weight the importance of words within the context of their embeddings.
- Pre-training
  - unsupervised training stage of an LLM on large amounts of text to establish word embeddings.
- Transfer Learning
  - utilize the pre-trained weights to provide context to more specialized or fine-tuned data.

What are Large Language Models (LLMs)?

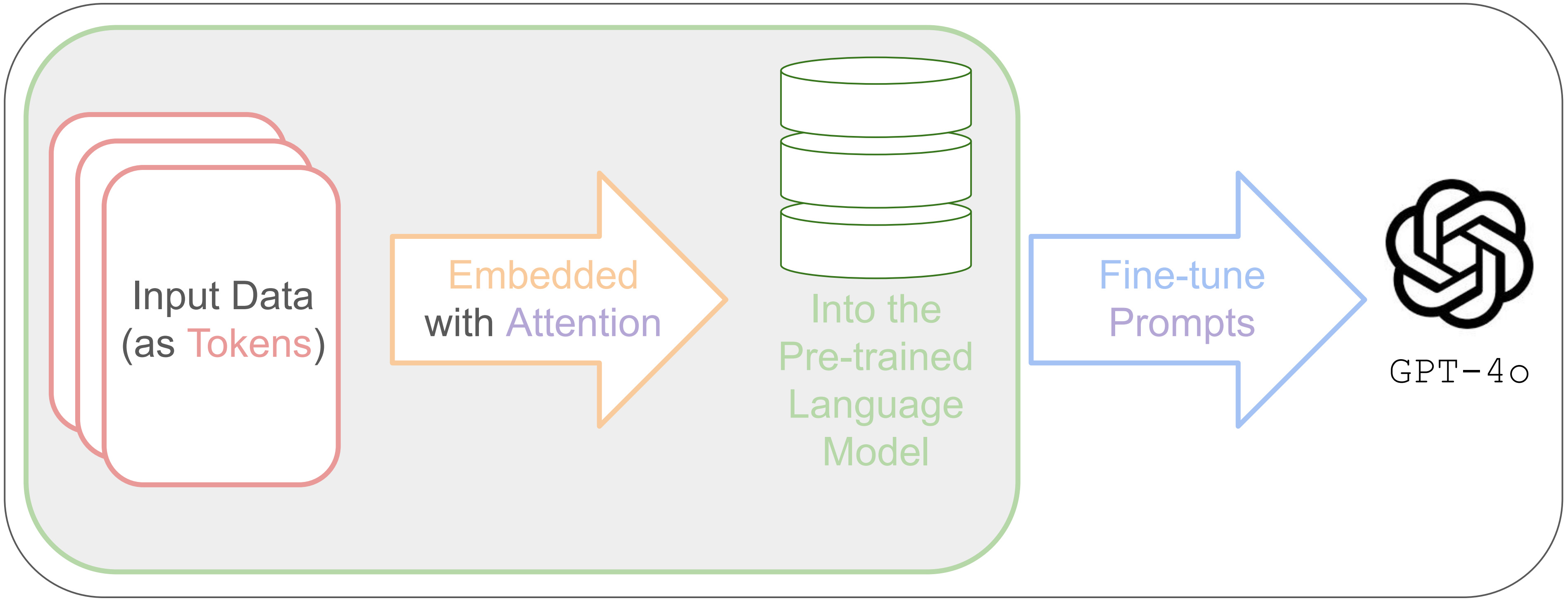
- A transformer model embeds (encodes) the data in a high dimensional space.
- To generate responses, the prompt is parsed into that embedding space to identify relevant information that is based on weights from the training data.
- The kinds of responses are fine-tuned with more targeted kinds of training inputs.
- The response is generated by providing the next most likely segment (token) of text.

Tokenization - Embedding - Attention - Pre-Training - Transfer Learning



https://medium.com/@anishnama20/exploring-the-power-of-encoder-decoder-models-pros-cons-and-applications-8bfbe2e66e76

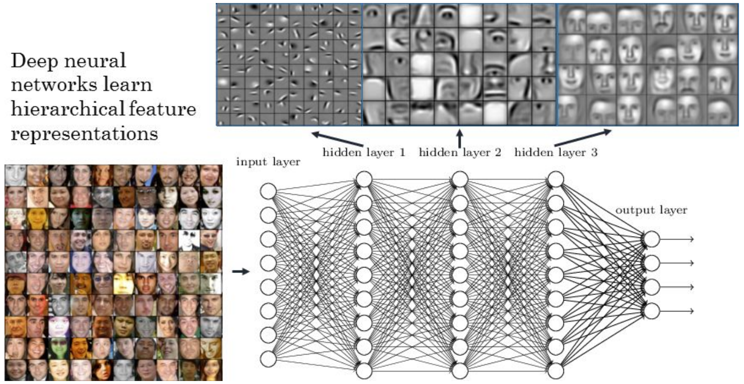
Building an LLM



Tokenization - Embedding - Attention - Pre-Training - Transfer Learning

What are Large Language Models (LLMs)?

- During model training, all the text is encoded into the model space as tokens.
  - A token can be letters and punctuation, but it can also be whole words or parts of speech.
- The layers of the LLM weight the tokens together into more complex topics.
  - Similar to the layers in vision networks, segmenting image facets of increasing complexity.
- The further apart phrases or topics are in the trained semantic space, the less related they are.



https://julien-vitay.net/lecturenotes-neurocomputing/3-deeplearning/3-CNN.html

What are Large Language Models (LLMs)?

- The prompt is evaluated in the embedding space to determine what parts are most important to attend to.
- The model can be augmented or specialized for different tasks or domains.
  - This is expensive in every sense of the word.
- The pre-trained set of embeddings can be transferred to perform better for different kinds of specialized prompts.
  - Again, this is similar to the vision parsing networks.

Tokenization - Embedding - Attention - Pre-Training - Transfer Learning

Questions?

https://www.youtube.com/watch?v=5hfYJsQAhl0

The Impact of Large Language Models

How are Large Language Models Used?

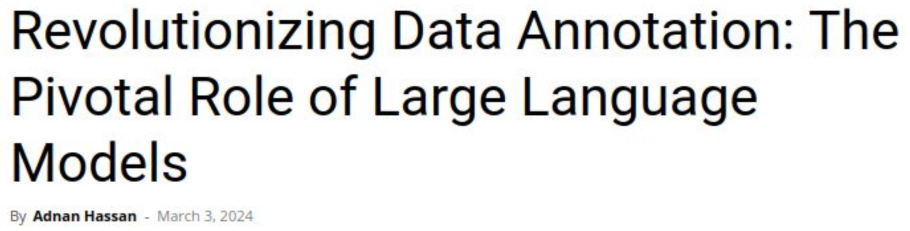
- Text Generation (Chatbots)
  - Customer Service
  - Talk with your documents
- Coding assistants
  - Github Copilot
  - Coding teams
- News Reports - Summaries
  - AI generated news stories
  - Academic Work
- Search Engines
  - Google called a code red
  - Databases - Search Replacement?

https://www.forbes.com/sites/bernardmarr/2023/01/10/the-11-best-and-most-innovative-new-gadgets-and-devices-in-2023/

And everyone is hyped!

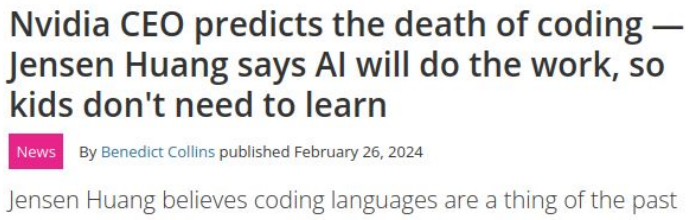
GitHub Copilot

Code 55% Faster!



Revolutionizing Data Annotation: The Pivotal Role of Large Language Models

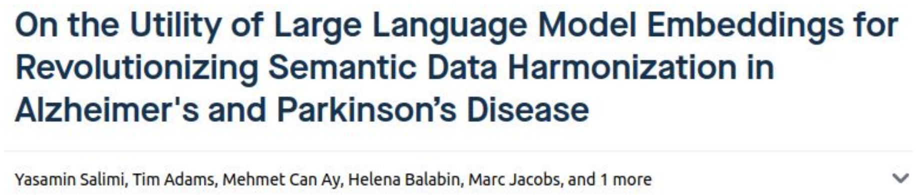
By **Adnan Hassan** - March 3, 2024



Nvidia CEO predicts the death of coding — Jensen Huang says AI will do the work, so kids don't need to learn

News By Benedict Collins published February 26, 2024

Jensen Huang believes coding languages are a thing of the past



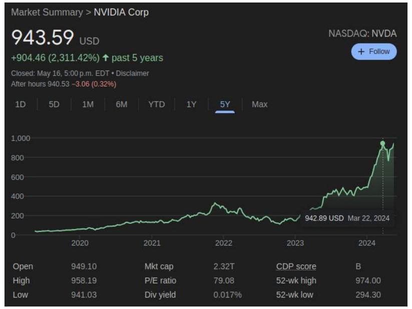
On the Utility of Large Language Model Embeddings for Revolutionizing Semantic Data Harmonization in Alzheimer's and Parkinson's Disease

Yasamin Salimi, Tim Adams, Mehmet Can Ay, Helena Balabin, Marc Jacobs, and 1 more

And everyone is hyped!… right?

GitHub Copilot

"Downward Pressure on Code Quality"


Market Summary > NVIDIA Corp

943.59 USD

+904.46 (2,311.42%) ↑ past 5 years

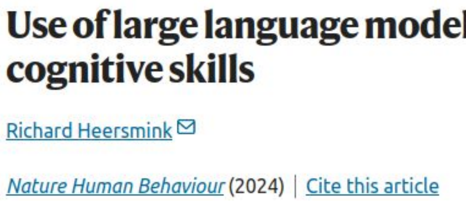
NASDAQ: NVDA

+ Follow

Closed: May 16, 5:00 p.m. EDT • Disclaimer
After hours 940.53 −3.06 (0.32%)

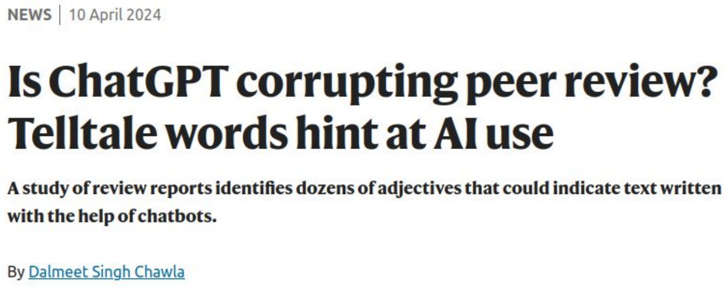
1D | 5D | 1M | 6M | YTD | 1Y | 5Y | Max

Open	949.10	Mkt cap	2.32T	CDP score	B
High	958.19	P/E ratio	79.08	52-wk high	974.00
Low	941.03	Div yield	0.017%	52-wk low	294.30


Use of large language models might affect our cognitive skills

Richard Heersmink

Nature Human Behaviour (2024) | Cite this article


NEWS | 10 April 2024

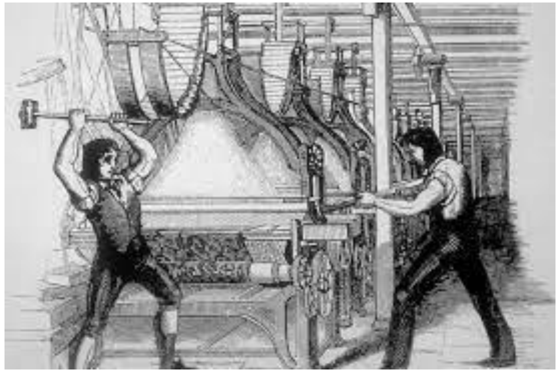
Is ChatGPT corrupting peer review? Telltale words hint at AI use

A study of review reports identifies dozens of adjectives that could indicate text written with the help of chatbots.

By Dalmeet Singh Chawla

This isn’t the first time…

- The industrial revolution greatly reduced needs of common types of skilled labor.
- A period of upheaval and consolidation, but eventually a new normal.
    - Improved quality of life (arguably).
- However, not everybody was on board.
    - Luddites



https://en.wikipedia.org/wiki/Luddite; https://www.historyextra.com/period/industrial-revolution/who-were-luddites-facts-what-happened/; https://www.linkedin.com/pulse/article-2-steam-steel-strategy-industrial-revolution-thomas-petrou

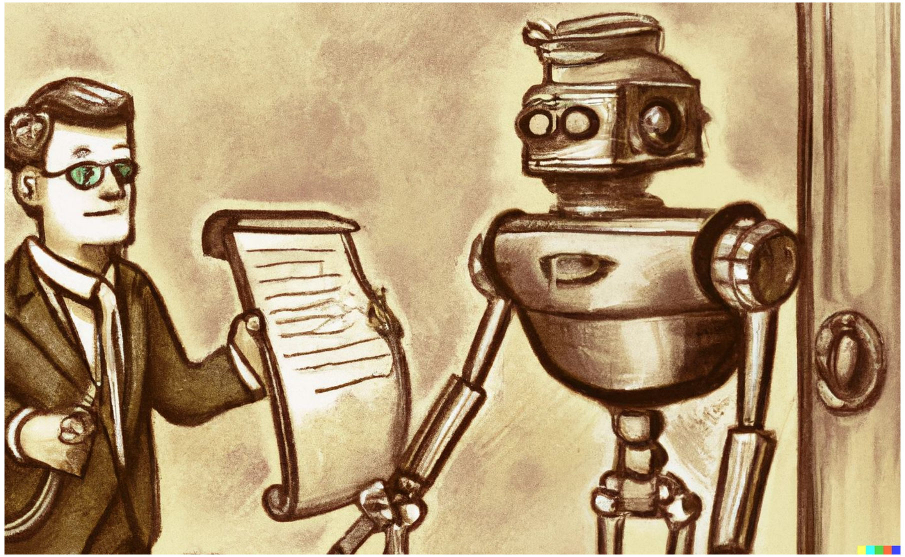
How do we do this right?

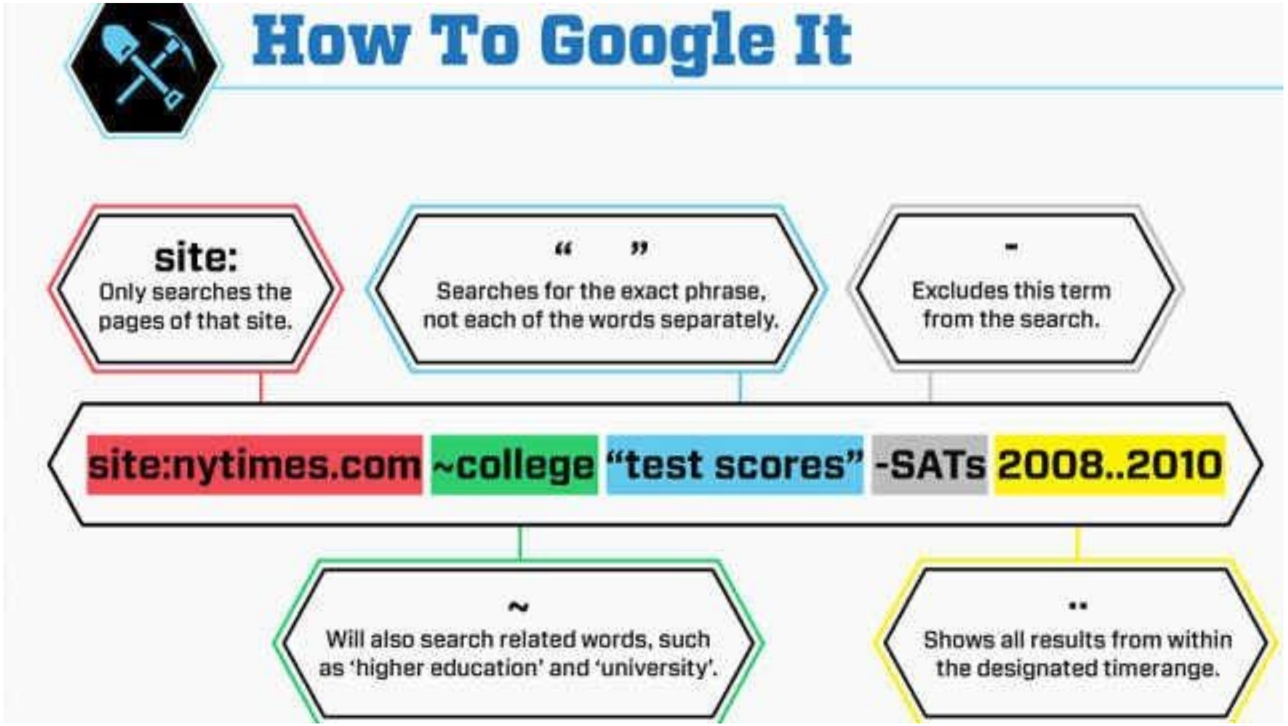
- There is a lot of potential for LLMs to greatly improve our lives.
  - Automate away many more tedious, detail oriented tasks.
  - Provide easier access to information.
- There is a lot of problems that may arise from their unchecked use.
  - They are still very unreliable - hallucinations.
  - Corporate control of the largest models is driven by profit, not benefit.
- It is critical that we proactively engage with and understand how to best use this technology so it doesn’t cause more harm than good.

Using an LLM Programmatically

Prompt Engineering

- A recent buzzword for the “job of the future” thanks to LLMs.
- It is about understanding how to interact with LLMs in the most effective way.
  - Without having to retrain them.
- There are many tools to interacting with LLMs to broaden their general utility.





How To Google It

site:
Only searches the pages of that site.

“ ”
Searches for the exact phrase, not each of the words separately.

-
Excludes this term from the search.

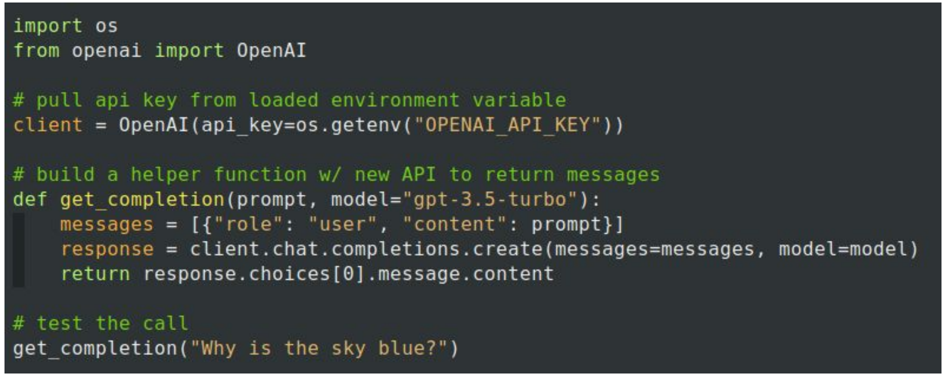
site:nytimes.com ~college “test scores” -SATs 2008..2010

~
Will also search related words, such as ‘higher education’ and ‘university’.

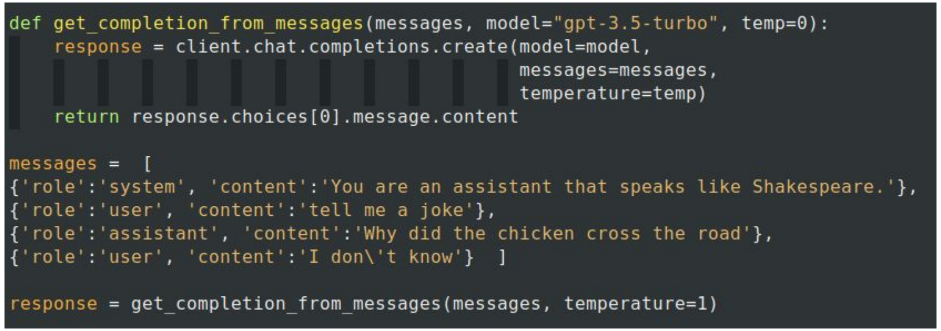
..
Shows all results from within the designated timerange.

Anatomy of a Prompt

- There are 3 speakers who can “talk” in a prompt.
  - The user
  - The model
  - The system
- Prompts are typically passed as JSON that hold the history of the interaction.



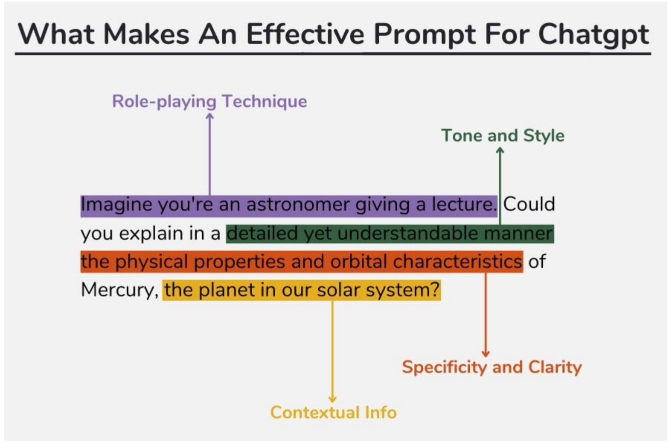
```python
import os
from openai import OpenAI

# pull api key from loaded environment variable
client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# build a helper function w/ new API to return messages
def get_completion(prompt, model="gpt-3.5-turbo"):
    messages = [{"role": "user", "content": prompt}]
    response = client.chat.completions.create(messages=messages, model=model)
    return response.choices[0].message.content

# test the call
get_completion("Why is the sky blue?")
```



```python
def get_completion_from_messages(messages, model="gpt-3.5-turbo", temp=0):
    response = client.chat.completions.create(model=model,
                                              messages=messages,
                                              temperature=temp)
    return response.choices[0].message.content

messages =  [
{'role':'system', 'content':'You are an assistant that speaks like Shakespeare.'},
{'role':'user', 'content':'tell me a joke'},
{'role':'assistant', 'content':'Why did the chicken cross the road'},
{'role':'user', 'content':'I don\'t know'}  ]

response = get_completion_from_messages(messages, temperature=1)
```

Anatomy of a Prompt

- The **user** is what you (the user) are asking the model to produce.
- The **model** is the LLMs responses to any input.
- The **system** is a silent setting that can modify how the LLM responds.


What Makes An Effective Prompt For Chatgpt
Role-playing Technique
Tone and Style
Imagine you're an astronomer giving a lecture. Could you explain in a detailed yet understandable manner the physical properties and orbital characteristics of Mercury, the planet in our solar system?
Specificity and Clarity
Contextual Info

Prompt Engineering - General Guidelines

1. Be clear and specific in what you ask.

   - This does not mean be brief.

2. Use delimiters to separate out specific parts or the prompt.

   - i.e. summarize the text in triple quotes.
   - Helps to sanitize the inputs of the LLMs.

3. Ask for structured output in the response

   - JSON formatting, HTML tables, etc.

4. Ask the LLM to check input conditions as part of the prompt.

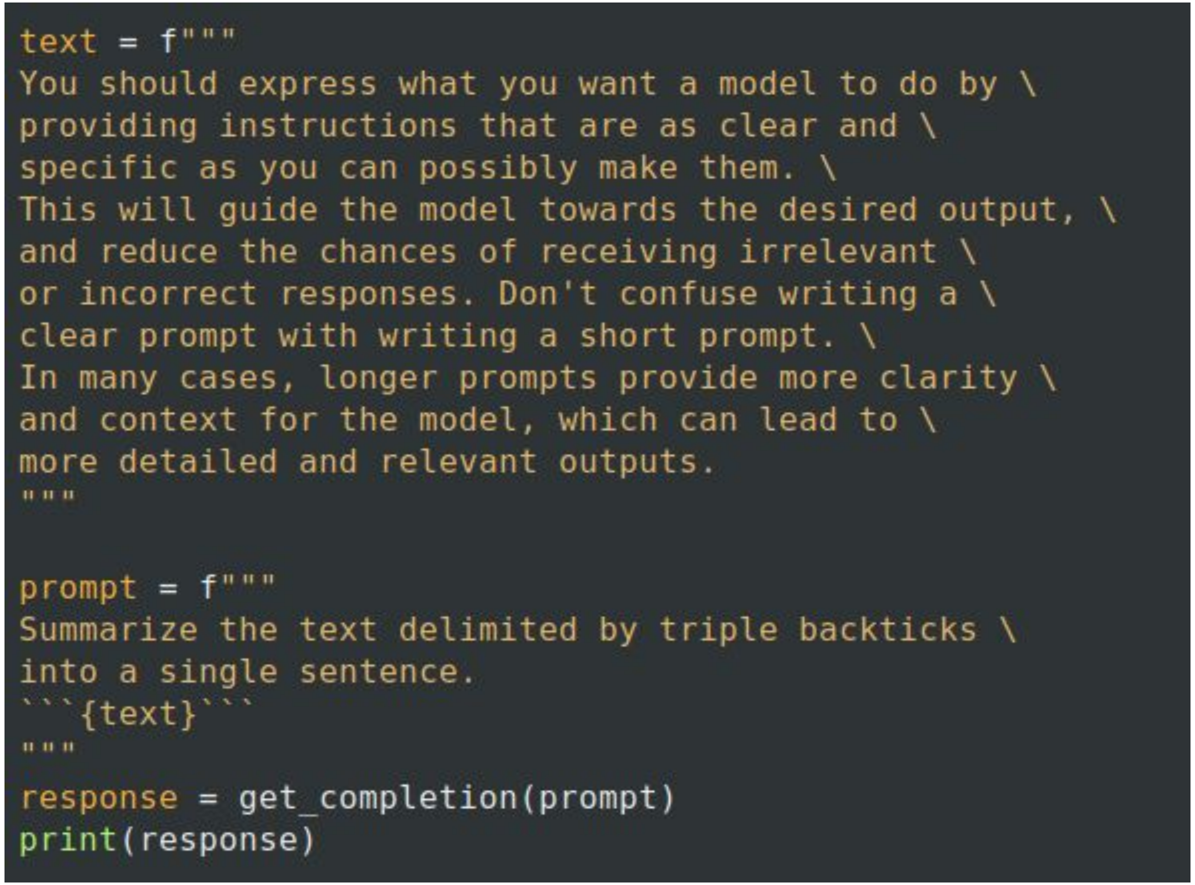
5. Provide successful input / output pairs for it to mimic.

6. Give the model time to “think”.

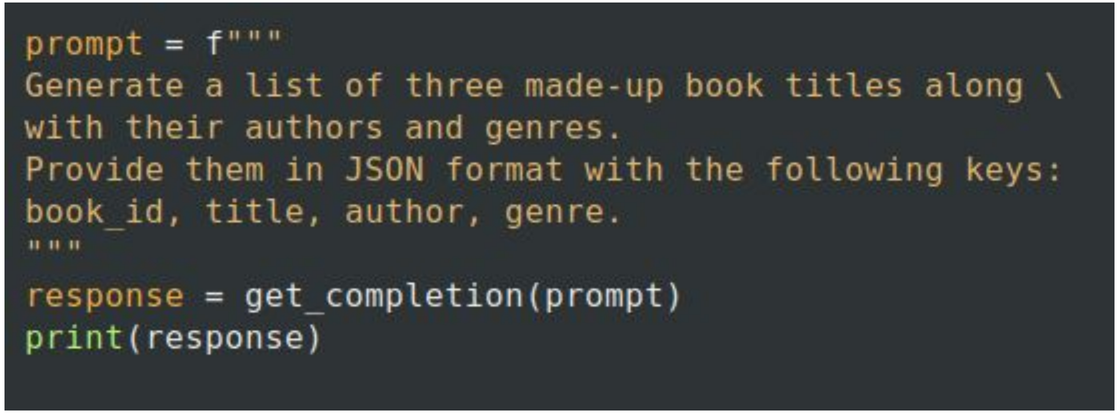
   - Give the LLM instructions for incremental steps toward the desired outcome
   - Have the LLM show its work while hiding its reasoning from the user.

7. Tell the model to make its own solution and compare it to the input

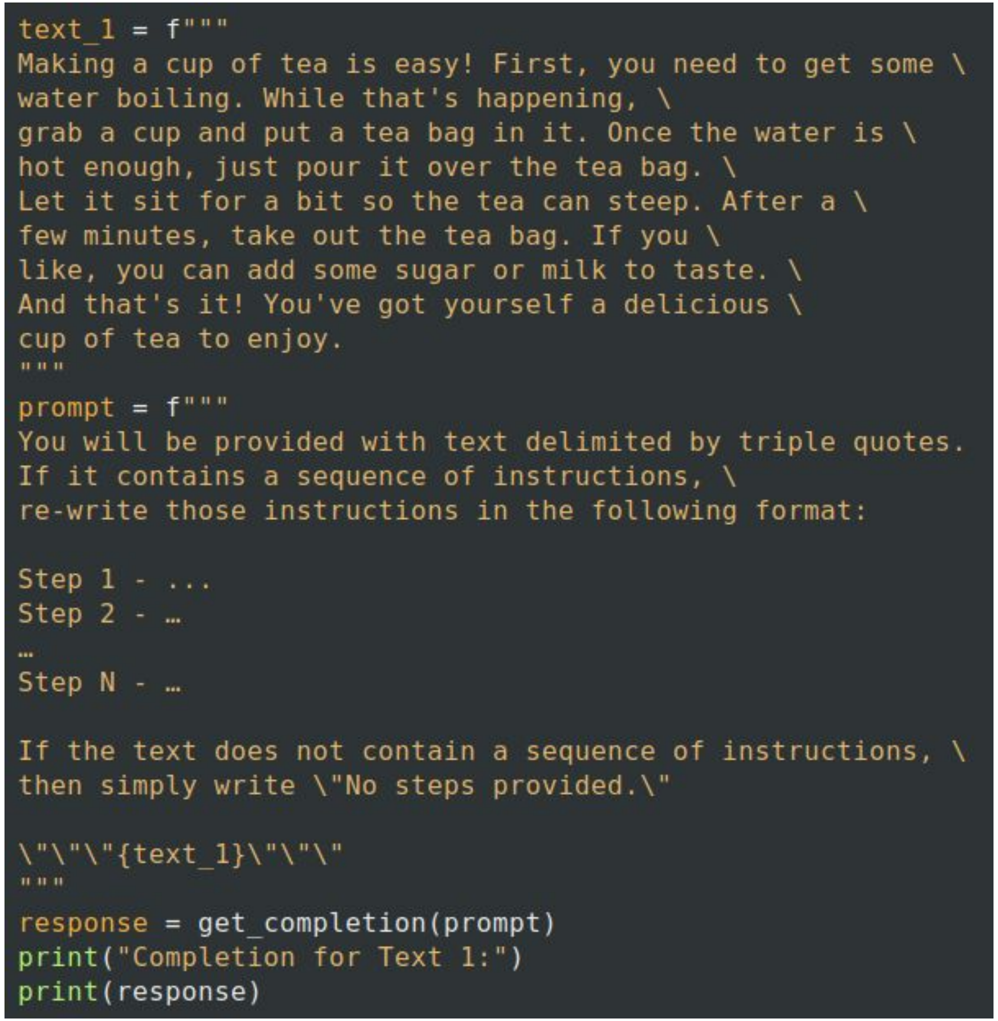
   - Again, step by step helps.



~~~python
text = f"""
You should express what you want a model to do by \
providing instructions that are as clear and \
specific as you can possibly make them. \
This will guide the model towards the desired output, \
and reduce the chances of receiving irrelevant \
or incorrect responses. Don't confuse writing a \
clear prompt with writing a short prompt. \
In many cases, longer prompts provide more clarity \
and context for the model, which can lead to \
more detailed and relevant outputs.
"""


prompt = f"""
Summarize the text delimited by triple backticks \
into a single sentence.
```{text}```
"""
response = get_completion(prompt)
print(response)
~~~



```python
prompt = f"""
Generate a list of three made-up book titles along \
with their authors and genres.
Provide them in JSON format with the following keys:
book_id, title, author, genre.
"""
response = get_completion(prompt)
print(response)
```

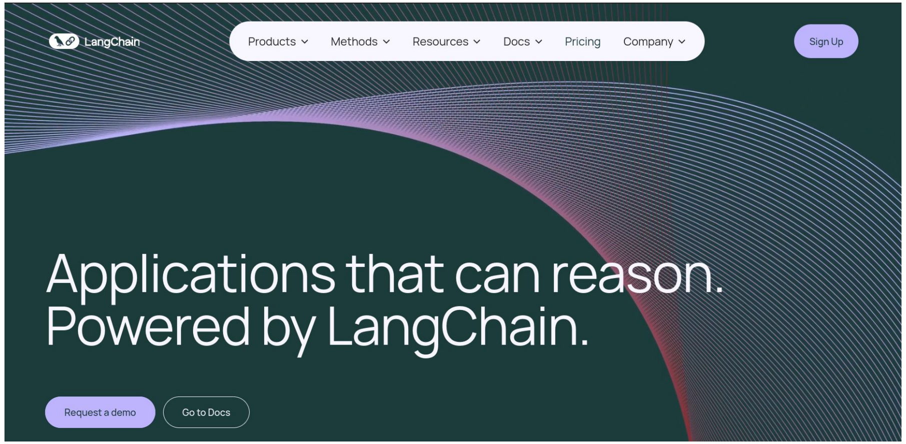


```
text_1 = f"""
Making a cup of tea is easy! First, you need to get some \
water boiling. While that's happening, \
grab a cup and put a tea bag in it. Once the water is \
hot enough, just pour it over the tea bag. \
Let it sit for a bit so the tea can steep. After a \
few minutes, take out the tea bag. If you \
like, you can add some sugar or milk to taste. \
And that's it! You've got yourself a delicious \
cup of tea to enjoy.
"""
prompt = f"""
You will be provided with text delimited by triple quotes.
If it contains a sequence of instructions, \
re-write those instructions in the following format:

Step 1 - ...
Step 2 - …
…
Step N - …

If the text does not contain a sequence of instructions, \
then simply write \"No steps provided.\"

\"\"\"{text_1}\"\"\"
"""
response = get_completion(prompt)
print("Completion for Text 1:")
print(response)
```

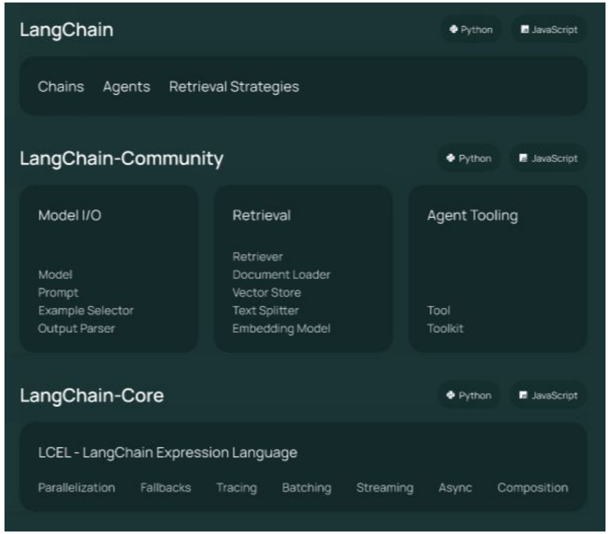
Specific Tools

LangChain

- Python / JS tools for interacting with LLMs programmatically.
- Facilitates prompt engineering and multiple calls to answer a question.
- Many utilities for parsing data for vector embedding in **RAG**s.
- Allows for (relatively) painless swapping between different LLMs.



Applications that can reason.
Powered by LangChain.



LangChain

Chains Agents Retrieval Strategies

LangChain-Community

Model I/O	Retrieval	Agent Tooling
Model	Retriever	Tool
Prompt	Document Loader	Toolkit
Example Selector	Vector Store	
Output Parser	Text Splitter	
	Embedding Model	

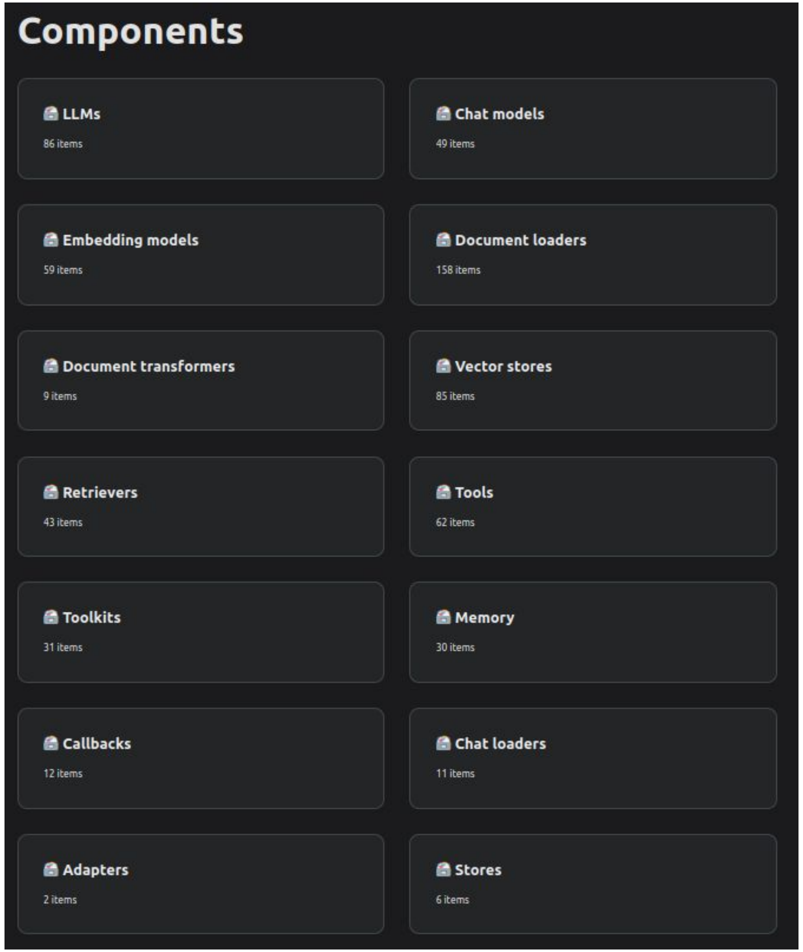
LangChain-Core

LCEL - LangChain Expression Language

Parallelization Fallbacks Tracing Batching Streaming Async Composition

LangChain Components

- Classes / Modules for working with standard components.
- Prompt Templates can be created with many standard strategies and logic.
    - LCEL (LangChain Expression Language)
- Because LangChain prompts are modular, they can be easily passed to multiple models.



Components

LLMs
86 items

Chat models
49 items

Embedding models
59 items

Document loaders
158 items

Document transformers
9 items

Vector stores
85 items

Retrievers
43 items

Tools
62 items

Toolkits
31 items

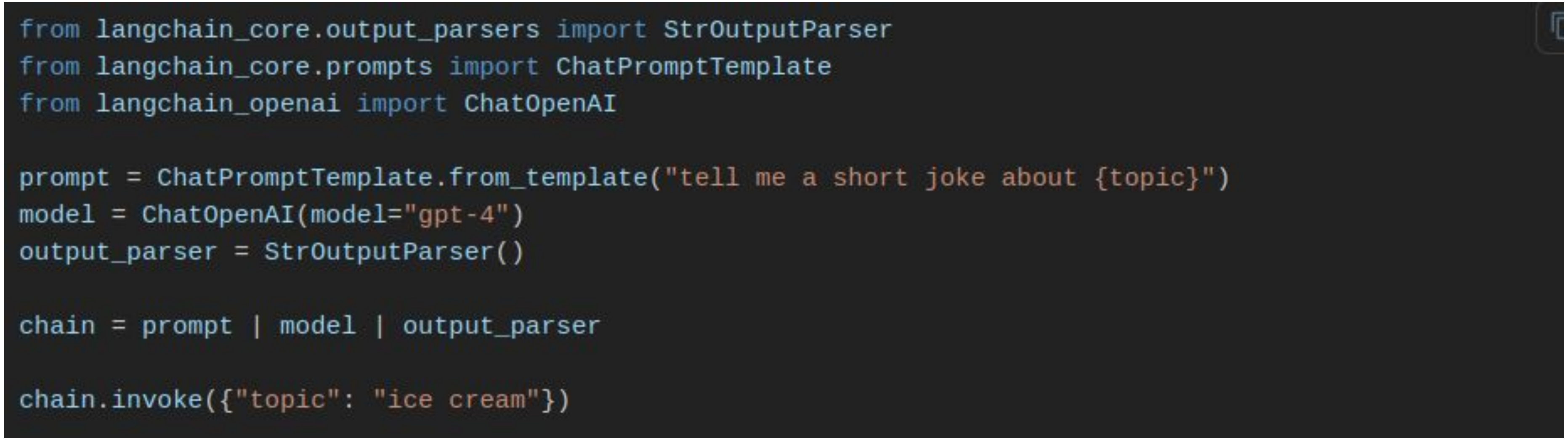
Memory
30 items

Callbacks
12 items

Chat loaders
11 items

Adapters
2 items

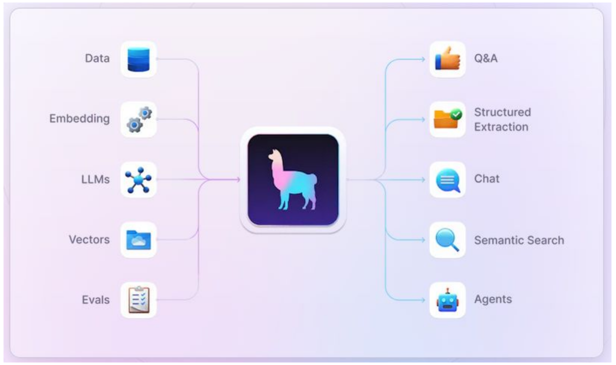
Stores
6 items

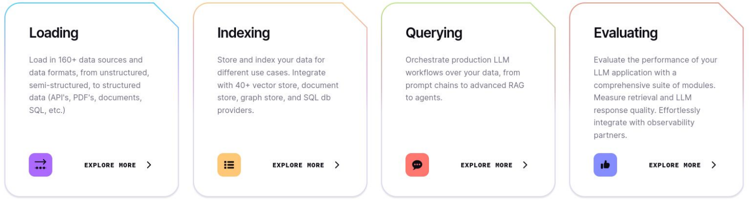


```python
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate
from langchain_openai import ChatOpenAI

prompt = ChatPromptTemplate.from_template("tell me a short joke about {topic}")
model = ChatOpenAI(model="gpt-4")
output_parser = StrOutputParser()

chain = prompt | model | output_parser

chain.invoke({"topic": "ice cream"})
```

LlamaIndex

- Many overlapping features with LangChain.
- A bit more focus on building [Agents](#).
- These tools are all still rapidly being developed.
  - Some features / models may be further developed in one ecosystem over another.





Loading

Load in 160+ data sources and data formats, from unstructured, semi-structured, to structured data (API's, PDF's, documents, SQL, etc.)

EXPLORE MORE >

Indexing

Store and index your data for different use cases. Integrate with 40+ vector store, document store, graph store, and SQL db providers.

EXPLORE MORE >

Querying

Orchestrate production LLM workflows over your data, from prompt chains to advanced RAG to agents.

EXPLORE MORE >

Evaluating

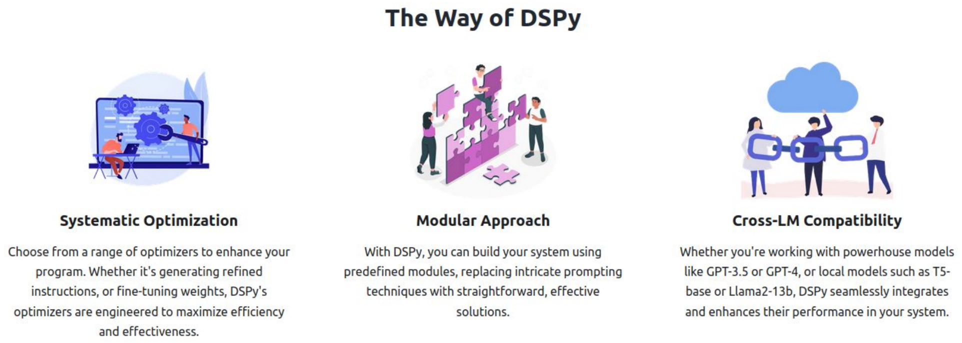
Evaluate the performance of your LLM application with a comprehensive suite of modules. Measure retrieval and LLM response quality. Effortlessly integrate with observability partners.

EXPLORE MORE >

DSPy

Programming—not prompting—Language Models

- Newer framework / philosophy.
- Priority on building Agents.
- Optimize your prompts on an LLM without manually adjusting inputs.
  - Rewrite prompts to optimize a cost function to get closer to your desire. output.



The Way of DSPy

Systematic Optimization

Choose from a range of optimizers to enhance your program. Whether it's generating refined instructions, or fine-tuning weights, DSPy's optimizers are engineered to maximize efficiency and effectiveness.

Modular Approach

With DSPy, you can build your system using predefined modules, replacing intricate prompting techniques with straightforward, effective solutions.

Cross-LM Compatibility

Whether you're working with powerhouse models like GPT-3.5 or GPT-4, or local models such as T5-base or Llama2-13b, DSPy seamlessly integrates and enhances their performance in your system.

Scrapegraph.ai

- Combines LLM prompting with web scraping for structured data extraction.
- Web crawlers + LLMs
- Use the flexibility of LLMs parsing to ease the ingestion of data for processing.

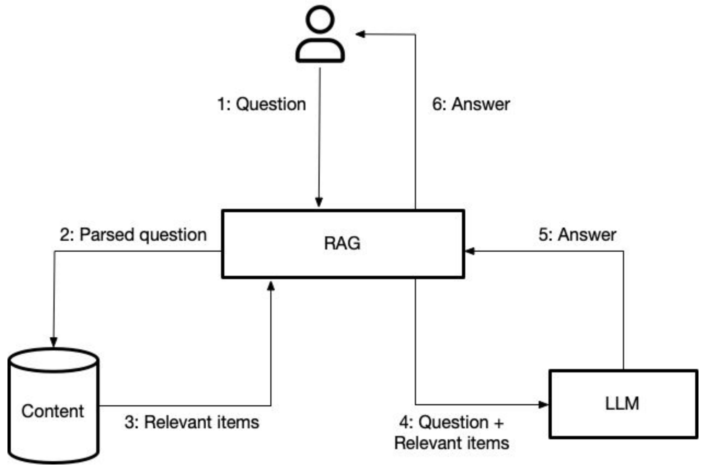
Ollama

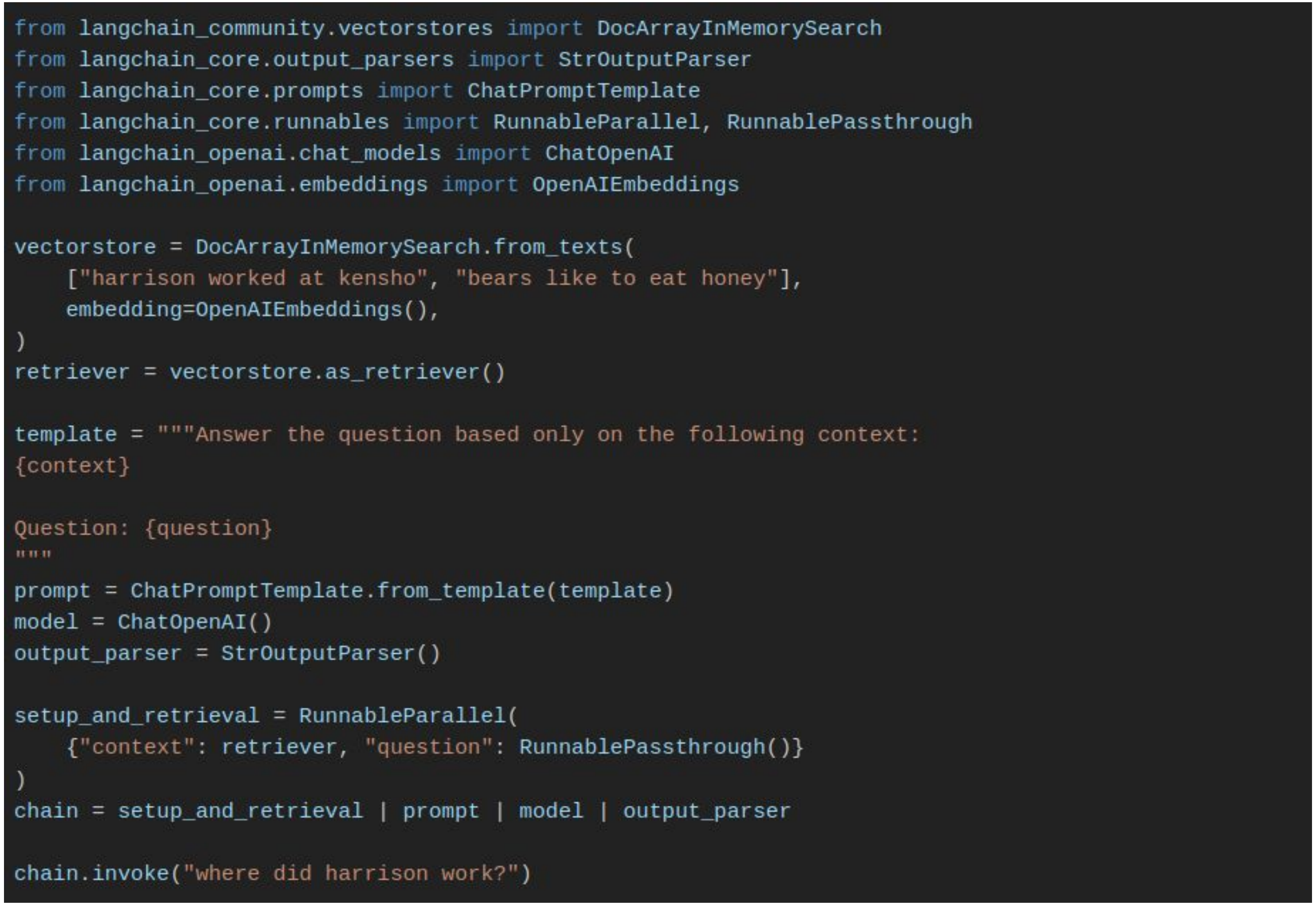
- Locally host and run any number of LLMs.
- Integrates with the previously described tools.
- Designed to be similar to running Docker containers.

What are they all using?

Retrieval Augmented Generation (RAG)

- It is a standard way of augmenting a model to specialize in a task without having to retrain the model.
- This involves a multistep process.
  - LangChain, LlamaIndex, and many other tools can facilitate this.
- The database (vector store) of information is ***<u>very</u>*** important.

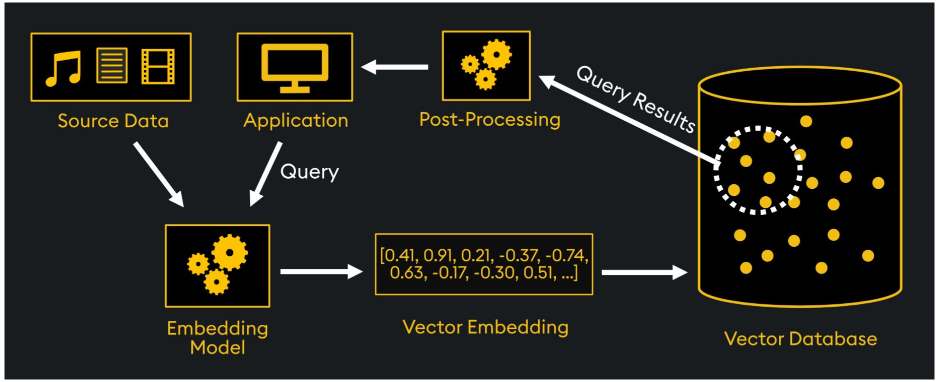




```python
from langchain_community.vectorstores import DocArrayInMemorySearch
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables import RunnableParallel, RunnablePassthrough
from langchain_openai.chat_models import ChatOpenAI
from langchain_openai.embeddings import OpenAIEmbeddings

vectorstore = DocArrayInMemorySearch.from_texts(
    ["harrison worked at kensho", "bears like to eat honey"],
    embedding=OpenAIEmbeddings(),
)
retriever = vectorstore.as_retriever()

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

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

setup_and_retrieval = RunnableParallel(
    {"context": retriever, "question": RunnablePassthrough()}
)
chain = setup_and_retrieval | prompt | model | output_parser

chain.invoke("where did harrison work?")
```

Vector Databases

- It is a database of information that is embedded in the model's space.
- The prompt is passed to a vector database to identify other relevant information based on the semantic distance between the prompt and the encoded information.
- The identified information is used to augment the prompt and improve the performance of the model.

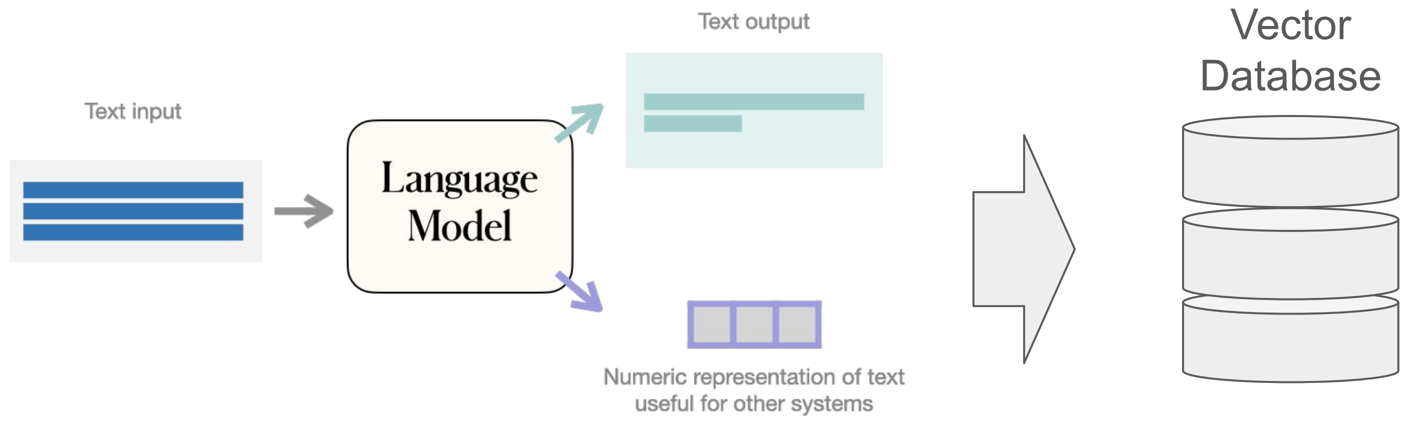


All kinds of data can be embedded into the vector store

- Pubmed / Arxiv
- Git / Github
- Slack / Gmail
- Project Gutenberg
- Obsidian / Notion
- YouTube transcripts
- Google Services
- AWS
- Microsoft Office Suite
- Most common databases

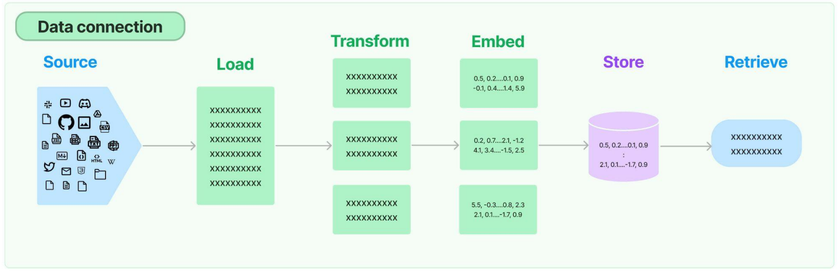
So what do we need?

- A trained large language model (LLM)
- A way to encode text to the LLMs embedding dimension (Embedding)
- A way to store the text (Vector Database)
- A way to manage the data going between components (LangChain, etc.)



Preparing Data for a RAG

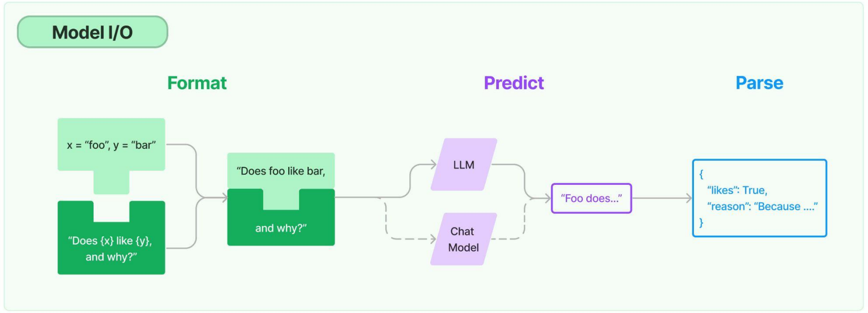
- Extract the text from its source
  - There are many parsers to ease the extraction.
    - Plain text, PDF, YouTube video transcript, GitHub, Notion, Slack, Databases, …
- Split the text into chunks
  - The chunks have a size and an overlap.
  - The effectiveness with which the data will be encoded can be tuned with these settings.
- Encode the chunks in the model space.
  - Build a vector database of contexts you want available when using the model.



https://python.langchain.com/docs/get_started/introduction

Pass a Prompt for Context

- Embed the prompt in the vector store.
- Determine the nearby entries based on their distance from each other in the high dimensional space.
  - This can be based on absolute distance or probabilistic variations.
    - Maximum Marginal Variance
- Return that information as text to augment the LLM prompt.
  - Prime the model with specific contexts.


Model I/O
Format
Predict
Parse
x = "foo", y = "bar"
"Does {x} like {y}, and why?"
"Does foo like bar, and why?"
LLM
Chat Model
"Foo does..."
{ "likes": True, "reason": "Because ...." }

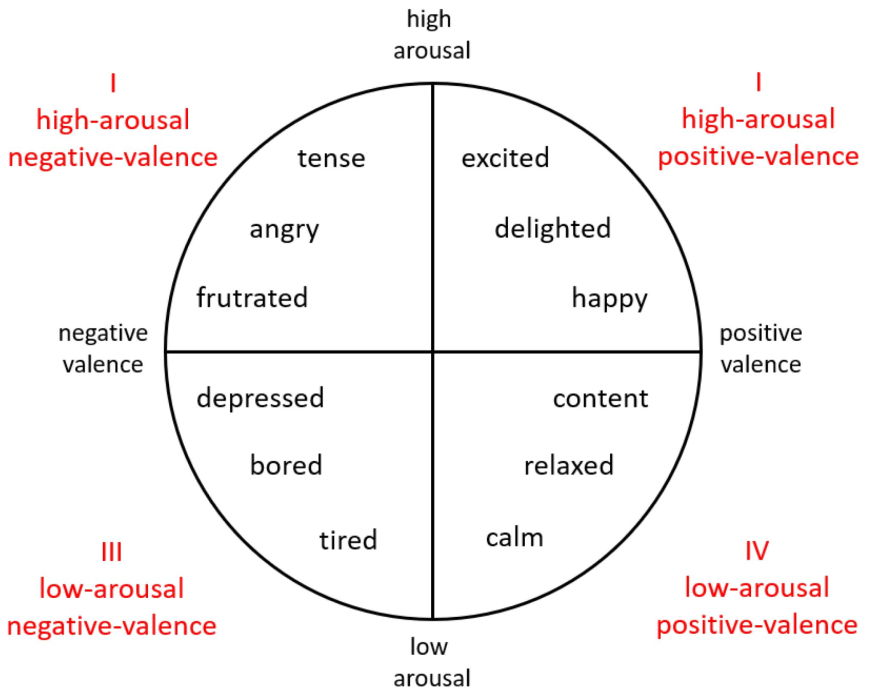
https://python.langchain.com/docs/get_started/introduction

Remember, at any point along the way you could be passing (chaining) your input through another LLM call!

Specialized LLM Improvements for Intermediary Prompts

- Content Filters / Censors
- Emotional Valence
- Metadata Taggers

These can provide their own metadata about part of a prompt.



https://www.mdpi.com/1424-8220/22/23/9102?type=check_update&version=1

What models should we use?

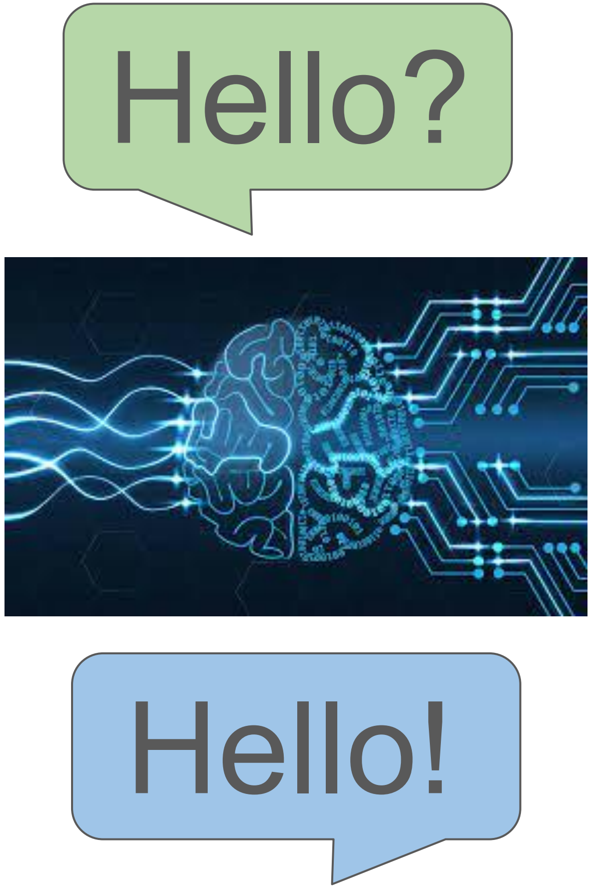
Open Models. Obviously.

- OpenAI is the most popular
  - Its name is also a lie.
- Google ~~Bard~~ / PaLM / Gemini
  - Costs money.
- Meta / Facebook
  - ~~Llama2~~ Llama3
  - Open sourced.
  - Many specialized derivatives.
- HuggingFace
  - A full repository of open models.

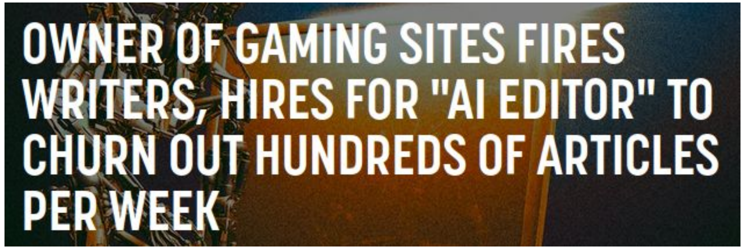
https://the-decoder.com/openllama-is-a-fully-open-source-llm-now-ready-for-business/

Conclusion

- LLMs are a new technology that can fundamentally change how we approach our work.
- It is important to proactively engage and understand their functionality and limits.
- There are many emerging (and open) tools to programmatically work with them.
- Start getting creative solving your problems with LLMs.



https://www.forbes.com/sites/garydrenik/2023/01/11/large-language-models-will-define-artificial-intelligence/


OWNER OF GAMING SITES FIRES WRITERS, HIRES FOR "AI EDITOR" TO CHURN OUT HUNDREDS OF ARTICLES PER WEEK

… culled the jobs of at least 50 humans — annihilating, by some estimates, around 40 percent of its workforce.

… would use AIs like ChatGPT to output up to an astounding — if not outright impossible — 200 to 250 articles of questionable quality *per week*.

Assuming a five-day work week, that's a lot to ask of one person to do with up to 50 articles per day, or less than *ten minutes* per article…

The pay isn't remarkable, either. At the listed salary range of $40,000 to $55,000 per year, that works out to roughly $4.23 per article, **at best**.