NEWSLETTERS

Deep Dive: How I taught ChatGPT to Draw Diagrams with LangChain

Building Machine Learning Solutions

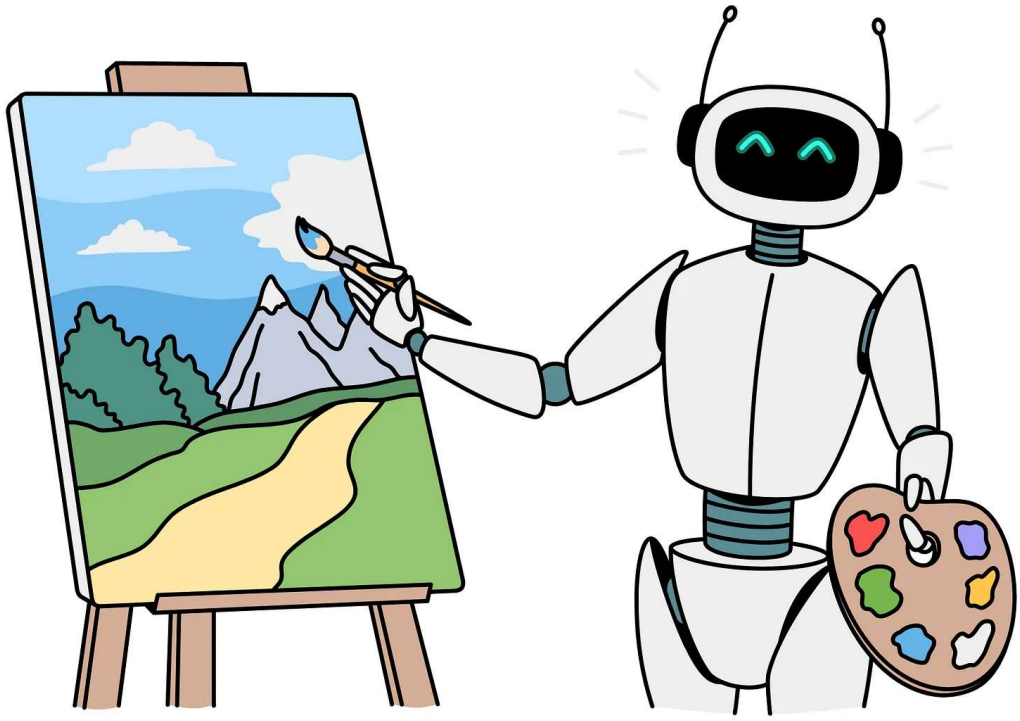
DAMIEN BENVENISTE

16 JUN 2023 · PAID

I have been working my way through generating automated Machine Learning content using ChatGPT and LangChain. I still have a long way to go but let me share my experience attempting to generate explanatory diagrams using those tools. We cover:

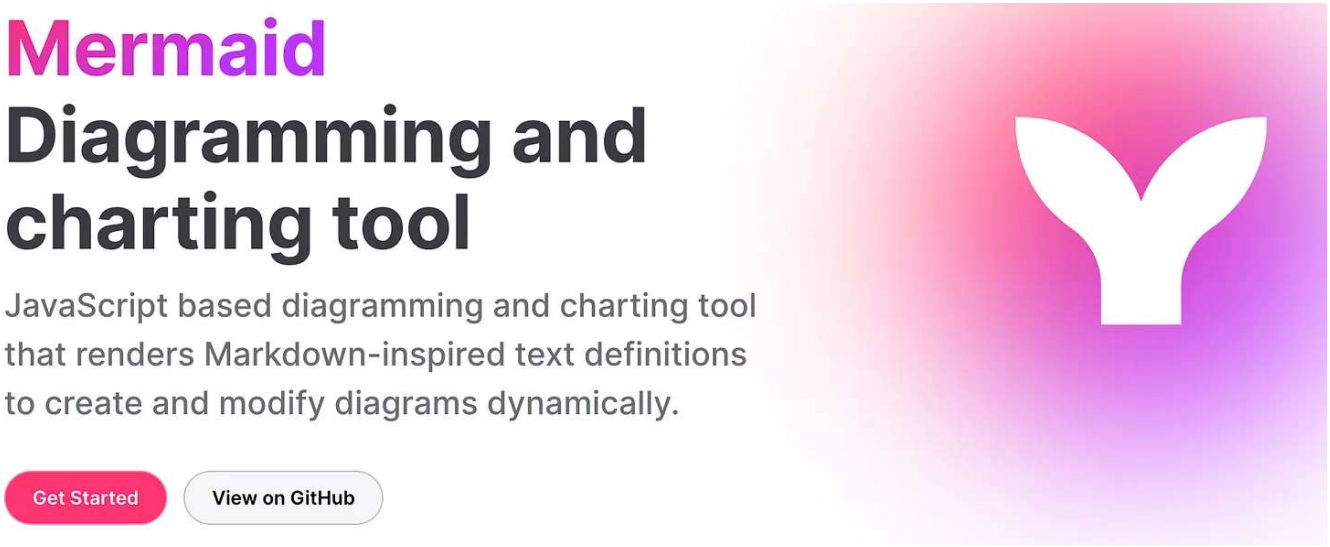
- ***Drawing diagrams with Mermaid***
- ***Teaching ChatGPT to draw diagrams***
  - ***Scraping the Mermaid documentation***
  - ***Loading the data into a vector database***
  - ***Generating diagrams with ChatGPT***
- ***Explaining an article with diagrams***
  - ***Getting the article***
  - ***Extracting the concepts to explain***
  - ***Explaining concepts***
  - ***Describing diagrams***
  - ***Translating into Mermaid code***
- ***Example of an article explained with ChatGPT***
  - ***Transformer***
  - ***Self-attention***

- ***Attention mechanisms***
- ***Sequence transduction models***
- ***Parallelizable models***



Drawing diagrams with Mermaid

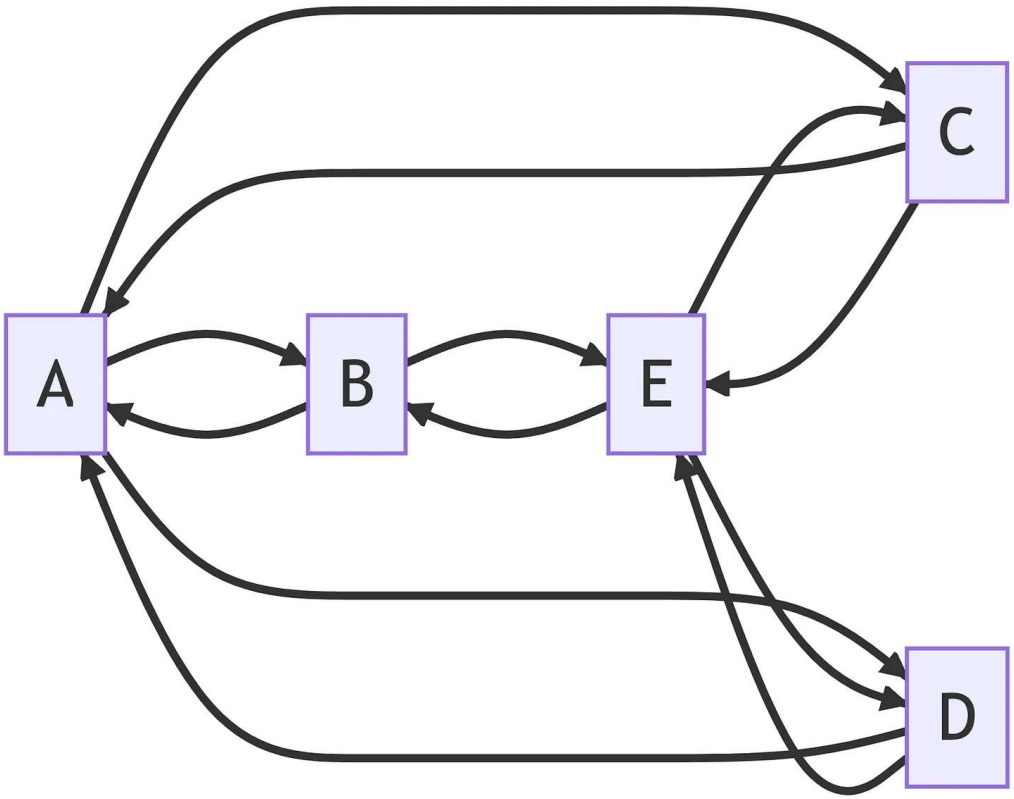
I wanted to find an easy charting tool that ChatGPT could use to visually explain complex concepts. Mermaid is a simple Javascript library:



It uses simple syntax to generate diagrams. For example the following code:

```
graph LR;
    A--> B & C & D;
    B--> A & E;
    C--> A & E;
    D--> A & E;
    E--> B & C & D;
```

generates this diagram:



To render those diagrams, you can use these online editors:

- Mermaid editor 1
- Mermaid editor 2

Additionally you can visualize those images in Python using the following script

```
import base64
from IPython.display import Image, display
import matplotlib.pyplot as plt

def visualize(graph):
    graphbytes = graph.encode('ascii')
    base64_bytes = base64.b64encode(graphbytes)
    base64_string = base64_bytes.decode('ascii')
    display(Image(url='https://mermaid.ink/img/' + base64_string))

visualize("""
graph LR;
    A--> B & C & D;
    B--> A & E;
```

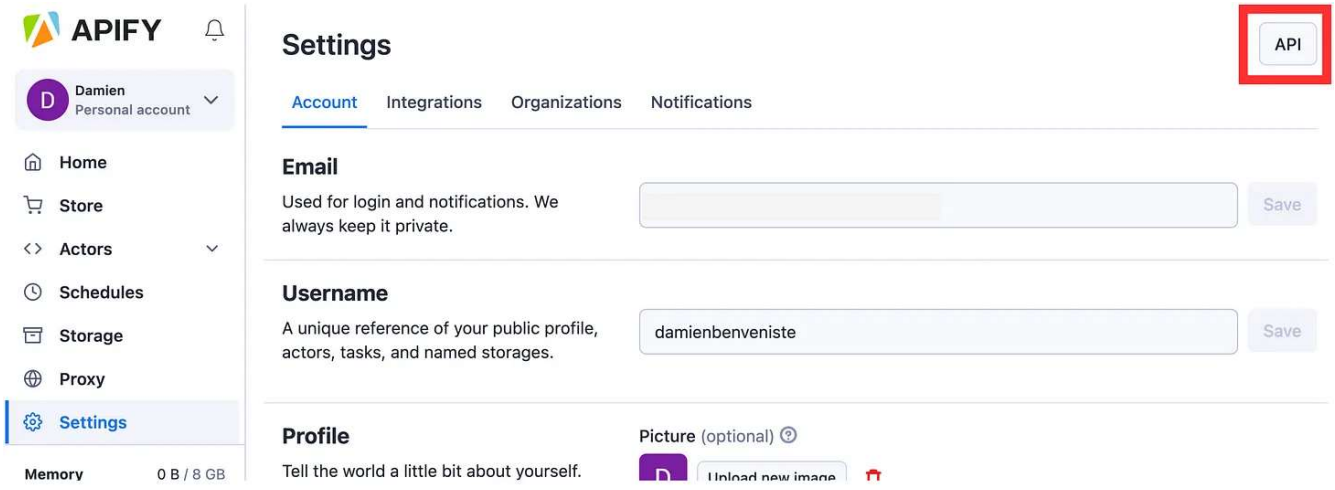
```
    C--> A & E;
    D--> A & E;
    E--> B & C & D;
""")
```

Teaching ChatGPT to draw diagrams

Scraping the Mermaid documentation

To be honest, ChatGPT was already trained on a previous version of the Mermaid documentation, so it could already generate Mermaid code, but I wanted to showcase how I would do it if it didn't. Here, I will scrape the Mermaid documentation to make it available to ChatGPT to help it generate diagrams. The main goal is to demonstrate LangChain's scraping capability, but I found it helped the LLM in making less mistakes in generating the code and it could help in the case the syntax evolved since the LLM got trained.

To scrape the Mermaid website we use Apify. After signing up, you can get your API key, by clicking the "API" button in the settings section



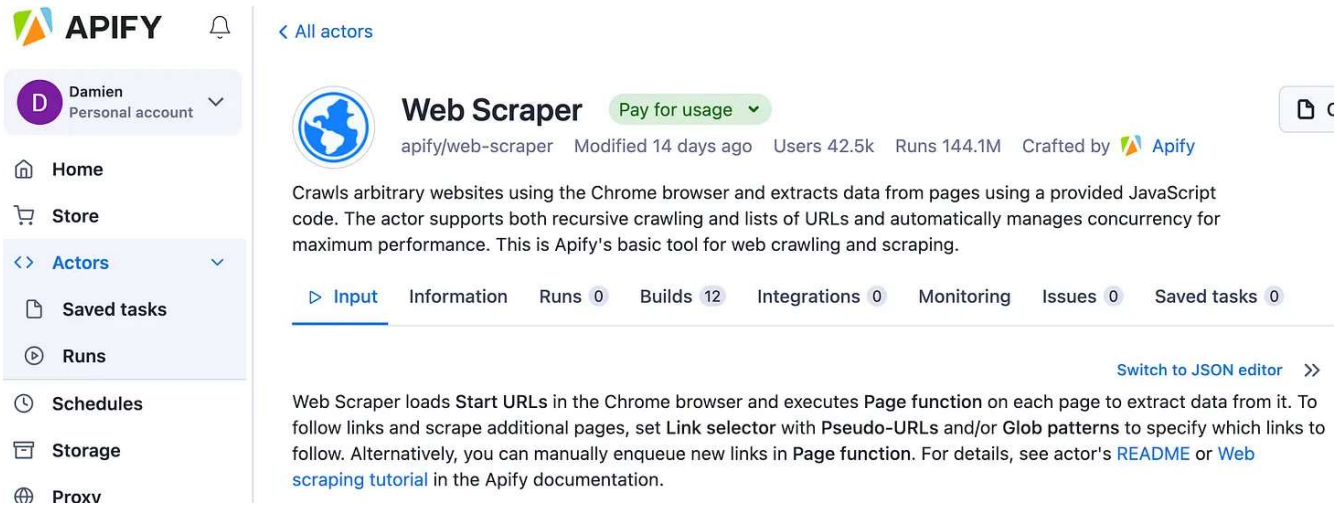
Let's make sure to capture the Apify API key as an environment variable

```
import os
os.environ['APIFY_API_TOKEN'] = ...
```

Scraping the whole website with the Apify wrapper in LangChain is quite easy. We just need to run the following code

```
from langchain.document_loaders.base import Document
from langchain.utilities import ApifyWrapper
from langchain.indexes import VectorstoreIndexCreator

apify = ApifyWrapper()

url = 'https://mermaid.js.org/'

loader = apify.call_actor(
    actor_id='apify/website-content-crawler',
    run_input={'startUrls': [{'url': url}]},
    dataset_mapping_function=lambda item: Document(
        page_content=item['text'] or '',
        metadata={'source': item['url']}
    ),
)
```

Here we used the `'apify/website-content-crawler'` actor to automatically scrape the whole website recursively starting from the URL https://mermaid.js.org/. You can find more information about this scraper in the Apify console

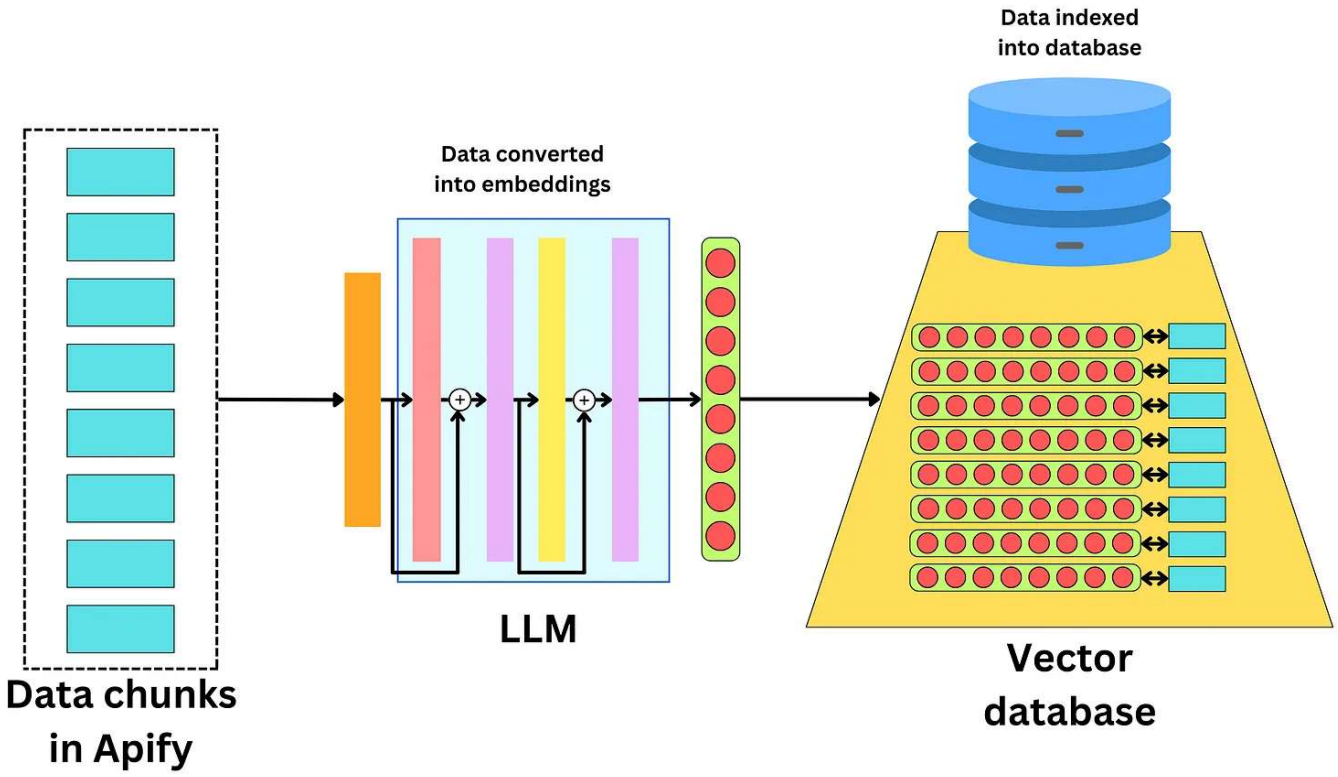


You can find the scraping results in the “runs” section

#	Webpage URL url	Extracted text text
1	https://mermaid.js.org/	Mermaid Diagramming and charting tool JavaScript based diagramming and charting tool that renders Markdown-inspired text definitions to create and modify diagrams dynamically.
2	https://mermaid.js.org/config...	Tutorials This is a list of publicly available Tutorials for using Mermaid.JS and is intended as a basic introduction for the use of the Live Editor for generating diagrams, and deploying...
3	https://mermaid.js.org/ecosy...	Integrations The following list is a compilation of different integrations and plugins that allow the rendering of mermaid definitions within other applications. They also serve as proof of...
4	https://mermaid.js.org/intro/	About Mermaid Mermaid lets you create diagrams and visualizations using text and code. It is a JavaScript based diagramming and charting tool that renders Markdown-inspired...
5	https://mermaid.js.org/config...	mermaid CLI mermaid CLI has been moved to mermaid-cli. Please read its documentation instead.
6	https://mermaid.js.org/config	Skip to content Mermaid Appearance Appearance On this page Frequently Asked

Loading the data into a vector database

We now index that data into a local vector database to make it searchable. We use the OpenAI embedding text encoding to convert text data into vectors



Before continuing, make sure to get your OpenAI API key by signing up on the OpenAI platform and capturing the API key as an environment variable

```
os.environ['OPENAI_API_KEY'] = ...
```

To move the data into a vector database, we simply run

```
from langchain.indexes import VectorstoreIndexCreator

index = VectorstoreIndexCreator().from_loaders([loader])
```

The current default vector store is ChromaDB. We can now query the database

```
query = 'What is the syntax for flowcharts?'
result = index.query_with_sources(query)
result
```

```
> 'question': 'What is the syntax for flowcharts?'

'answer': ' The syntax for flowcharts includes nodes (geometric shapes)
and edges (arrows or lines). Special characters can be escaped using
quotes or entity codes, and subgraphs can be defined using the
"subgraph" keyword.'

'sources': 'https://mermaid.js.org/syntax/flowchart.html'
```

Generating diagrams with ChatGPT

Now that we have the data, let's ask ChatGPT to generate some diagrams. We can turn the index into a retriever for the LLM to use

```
retriever = index.vectorstore.as_retriever()
# we change the number of document to return
retriever.search_kwargs['k'] = 10
```

'k' here is the number of documents returned when the LLM queries the database. We can create a chain to augment ChatGPT with that database

```
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI

llm = ChatOpenAI()

mermaid_qa = RetrievalQA.from_chain_type(
    llm=llm,
    retriever=retriever,
)
```

Let's see if ChatGPT can transform the following text (from Wikipedia) into a diagram:

> *Machine learning (ML) is a field devoted to understanding and building methods that let machines "learn" – that is, methods that leverage data to improve computer*

> *performance on some set of tasks.[1] Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so.[2] Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, agriculture, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.[3][4] A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers, but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning.[6][7] Some implementations of machine learning use data and neural networks in a way that mimics the working of a biological brain.[8][9] In its application across business problems, machine learning is also referred to as predictive analytics.*

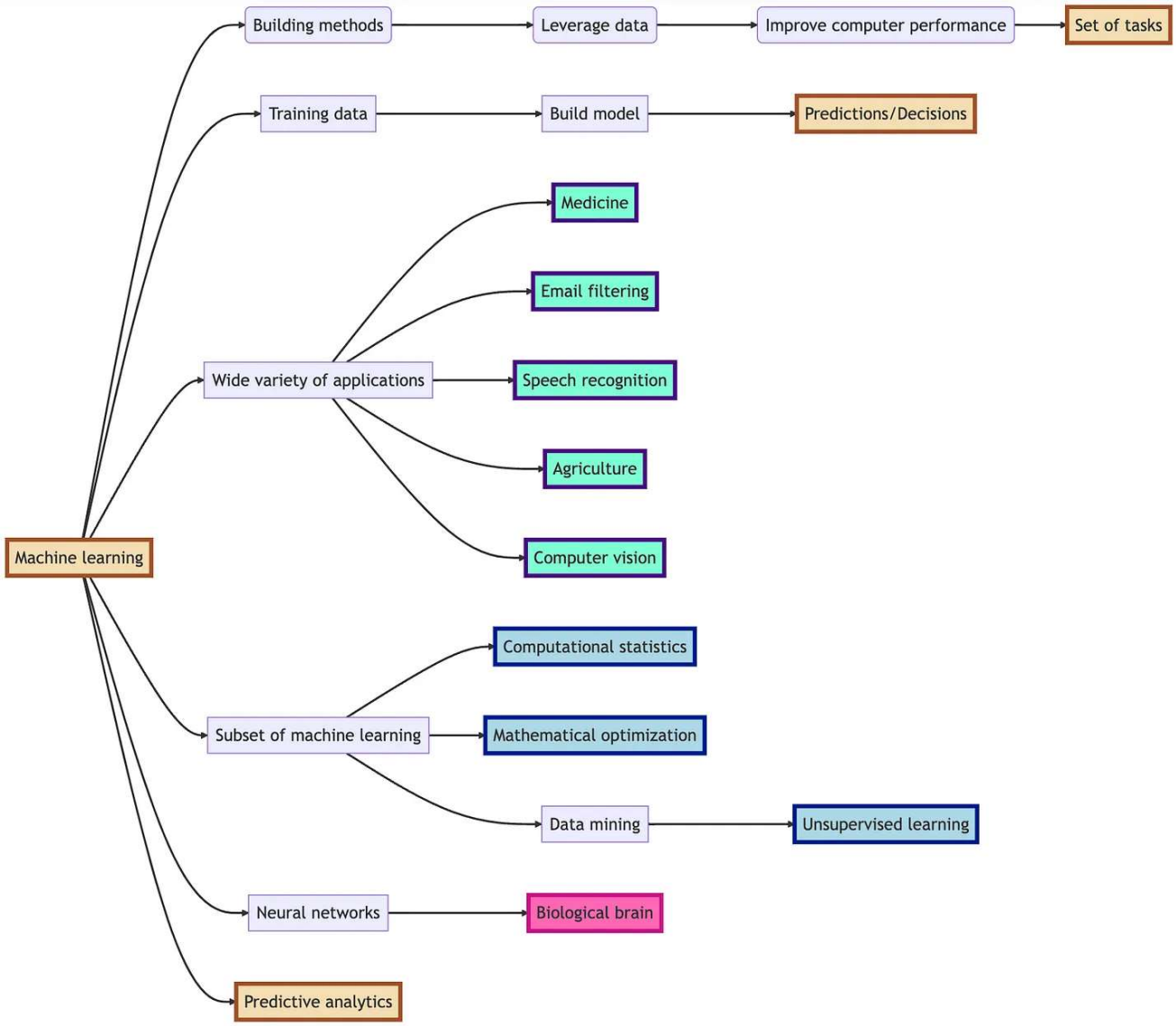
```
query = """
Your job is to write the code to generate a colorful mermaid diagram
describing the following text.
Return only the code and make sure it has multiple colors

TEXT: {text}
"""
result = mermaid_qa.run(query.format(text=text))
```

Here is the resulting code

```
graph LR
A[Machine learning] --> B(Building methods)
B --> C(Leverage data)
C --> D(Improve computer performance)
D --> E[Set of tasks]

A --> F[Training data]
F --> G[Build model]
G --> H[Predictions/Decisions]
```

```
A --> I[Wide variety of applications]
I --> J[Medicine]
I --> K[Email filtering]
I --> L[Speech recognition]
I --> M[Agriculture]
I --> N[Computer vision]

A --> O[Subset of machine learning]
O --> P[Computational statistics]
O --> Q[Mathematical optimization]
O --> R[Data mining]
R --> S[Unsupervised learning]

A --> T[Neural networks]
T --> U[Biological brain]

A --> V[Predictive analytics]

style A fill:#F5DEB3,stroke:#A0522D,stroke-width:4px;
style E fill:#F5DEB3,stroke:#A0522D,stroke-width:4px;
style H fill:#F5DEB3,stroke:#A0522D,stroke-width:4px;
style J fill:#7FFFD4,stroke:#4B0082,stroke-width:4px;
style K fill:#7FFFD4,stroke:#4B0082,stroke-width:4px;
style L fill:#7FFFD4,stroke:#4B0082,stroke-width:4px;
style M fill:#7FFFD4,stroke:#4B0082,stroke-width:4px;
style N fill:#7FFFD4,stroke:#4B0082,stroke-width:4px;
style P fill:#ADD8E6,stroke:#00008B,stroke-width:4px;
style Q fill:#ADD8E6,stroke:#00008B,stroke-width:4px;
style S fill:#ADD8E6,stroke:#00008B,stroke-width:4px;
style U fill:#FF69B4,stroke:#C71585,stroke-width:4px;
style V fill:#F5DEB3,stroke:#A0522D,stroke-width:4px;
```

and the related diagram



Explaining an article with diagrams

Getting the article

I wanted to see if I could use LangChain to automate the process of explaining articles. Let's see if we can explain the famous article "Attention is all you need!"

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Let's use the ArXiv Python package (`pip install arxiv`) to download the paper

```
import arxiv

paper = next(arxiv.Search(id_list=['1706.03762']).results())
paper.download_pdf(filename='attention.pdf')
```

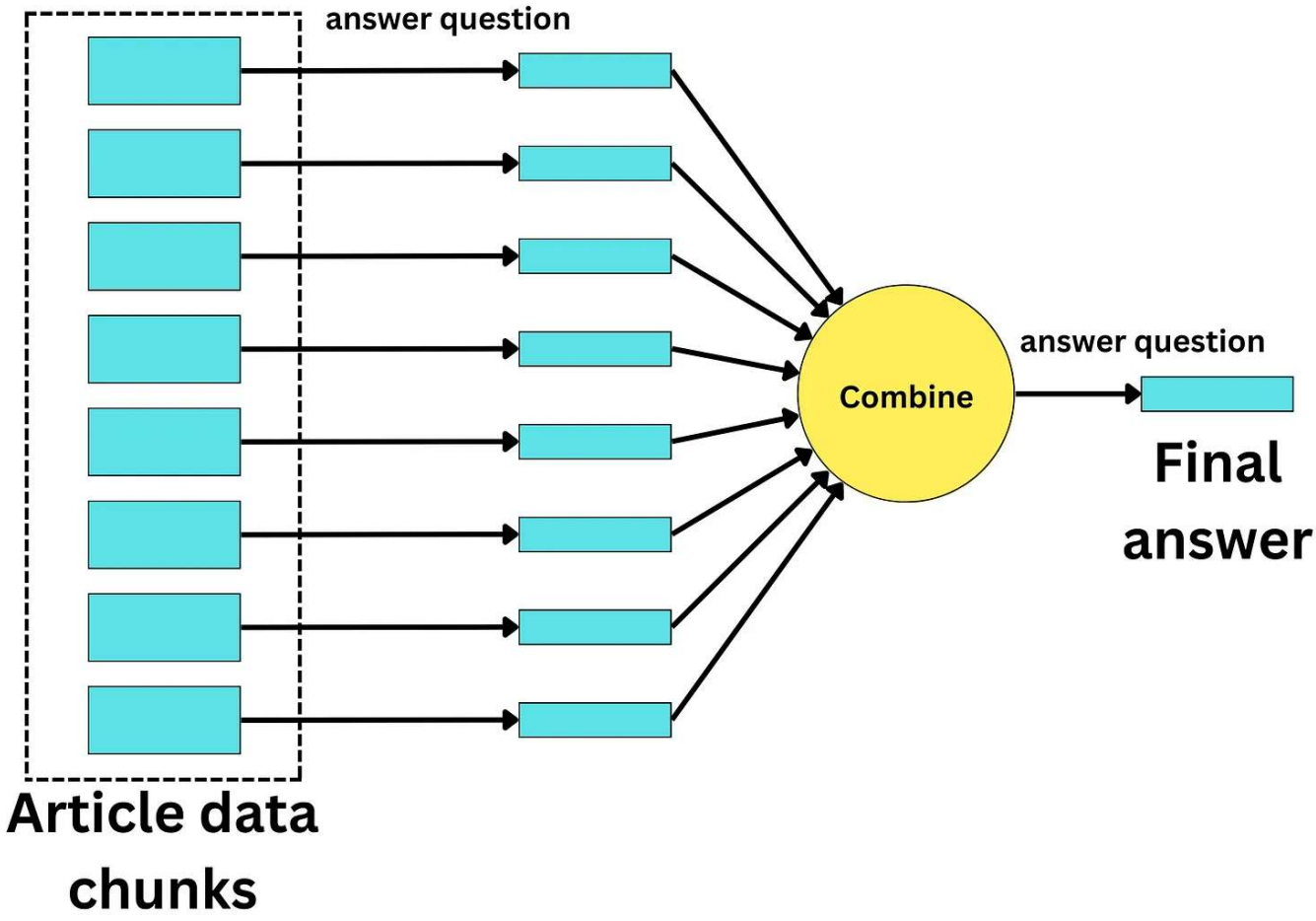
and the LangChain PDF loader to load and split into documents

```
from langchain.document_loaders import PyPDFLoader

loader = PyPDFLoader('attention.pdf')
pages = loader.load_and_split()

pages[0]
```

```
Document(page_content='Attention Is All You Need\nAshish Vaswani\x03\nGoo
gle Brain\navaswani@google.comNoam Shazeer\x03\nGoogle Brain\nnoam@googl
e.comNiki Parmar\x03\nGoogle Research\nnikip@google.comJakob Uszkoreit\x0
3\nGoogle Research\nusz@google.com\nLlion Jones\x03\nGoogle Research\nlli
on@google.comAidan N. Gomez\x03y\nUniversity of Toronto\naidan@cs.toront
o.eduŁukasz Kaiser\x03\nGoogle Brain\nlukaszkaiser@google.com\nIllia Polo
sukhin\x03z\nillia.polosukhin@gmail.com\nAbstract\nThe dominant sequence
transduction models are based on complex recurrent or\nconvolutional neur
al networks that include an encoder and a decoder. The best\nperforming m
odels also connect the encoder and decoder through an attention\nmechanis
m. We propose a new simple network architecture, the Transformer,\nbased
solely on attention mechanisms, dispensing with recurrence and convolutio
ns\nentirely. Experiments on two machine translation tasks show these mod
els to\nbe superior in quality while being more parallelizable and requir
ing significantly\nless time to train. Our model achieves 28.4 BLEU on the
WMT 2014 English-\nto-German translation task, improving over the existin
g best results, including\nensembles, by over 2 BLEU. On the WMT 2014 Eng
lish-to-French translation task,\nour model establishes a new single-mode
l state-of-the-art BLEU score of 41.8 after\ntraining for 3.5 days on eig
```

Extracting the concepts to explain

Let's have ChatGPT scan the paper to extract the concepts explained. We use the "map-reduce" question answering strategy. The idea is we apply the same question to all the data chunks of the article and apply a combine step to combine the answers:



Let's load the chain

```
from langchain.chains.question_answering import load_qa_chain

concept_chain = load_qa_chain(
    llm=llm,
    chain_type='map_reduce'
)
```

and ask the question

```
query = """
The text provided as context is an extract of an academic paper in the
field of machine learning.
Your job is to find the main concepts that are explained in this paper
and return those concepts as a list, all on one line separated by the
```

```
characters '||'. Order that list from most essential concept for the
paper to least essential. Limit the list to the 5 most important
concepts

Example: concept 1||concept 2||concept 3
"""

result = concept_chain({'input_documents': pages, 'question': query})

result['output_text']

> Transformer||self-attention||attention mechanisms||sequence
transduction models||parallelizable models
```

Here I asked ChatGPT to return the concepts separated by the characters '||' so I can process the resulting string

```
concept_list = result['output_text'].split('||')
concept_list

> ['Transformer',
 'self-attention',
 'attention mechanisms',
 'sequence transduction models',
 'parallelizable models']
```

Explaining concepts

Now that we have the concepts we can ask ChatGPT to explain those concepts. Let's load the chain

```
explain_chain = load_qa_chain(
    llm=llm,
    chain_type='map_reduce'
)
```

Let's write a query to request an explanation of a concept

```
query = query = """
The text provided as context is an extract from an academic paper in
machine learning.

Your job is to use that context to explain the following concept.
The answer must be self-contained so you cannot refer to the article in
the answer.
Imagine you are a Computer Science Professor teaching at the
university.
Respond only with an explanation of the concept:


CONCEPT: {concept}
"""
```

We can now iterate through the concepts:

```
explanations = []

for concept in concept_list:

    result = explain_chain({
        'input_documents': pages,
        'question': query.format(concept=concept)
    })

    explanations.append({
        'concept': concept,
        'explanation': result['output_text']
    })

explanations[0]['explanation']
```

> *The Transformer is a type of neural network architecture that is commonly used in natural language processing tasks, such as language translation and text generation.*

> *Unlike traditional recurrent neural networks, which process sequences one word at a time, the Transformer is based on the self-attention mechanism, allowing it to consider all words in the sequence simultaneously. The Transformer consists of an encoder and a decoder, each composed of multiple layers of self-attention and feedforward neural networks. The encoder processes the input text and generates a sequence of hidden representations, while the decoder takes these representations as input and generates the output text. By allowing each position in the input and output sequences to attend to all positions in the sequence, rather than just the previous positions, the Transformer is able to better capture the dependencies between words in the input and output sequences. Additionally, the Transformer is highly parallelizable, allowing for faster training and inference on modern hardware. Overall, the Transformer has proven to be a highly effective model architecture for sequence transduction tasks, achieving state-of-the-art results on various natural language processing benchmarks, and has become a foundational technology in the field of NLP.*

Describing diagrams

Before asking for the Mermaid code, I will request a description of a possible visual diagram that explains the concepts. The reason is twofold: it forces ChatGPT to focus on describing a possible visual without mermaid code. Second, we can feed only the description instead of the whole article when using the Mermaid documentation database. Let's load the chain and write a query:

```
describe_chain = load_qa_chain(
    llm=llm,
    chain_type='map_reduce'
)

query = """
The text provided as context is an extract of an academic paper in the
field of machine learning.
Your job is to use that context to describe a visual diagram that could
explain in details that concept.
Be precise and keep the description symbolic:
```

```
CONCEPT: {concept}
"""
```

Let's run through the different concepts:

```
for expl_dict in explanations:

    result = describe_chain({
        'input_documents': pages,
        'question': query.format(concept=expl_dict['concept'])
    })

    expl_dict['description'] = result['output_text']

explanations[0]['description']
```

> *The visual diagram of the Transformer architecture consists of an input sequence of symbols and an output sequence of symbols. The input layer is connected to the first encoder layer, which consists of multiple boxes arranged horizontally, each box representing a different attention head. The self-attention mechanism is represented by a set of queries, keys, and values, with arrows connecting them to the output of the previous layer. The encoder layer also includes a position-wise fully connected feed-forward network that applies a non-linear activation function to the output of the self-attention mechanism. The output of the feed-forward network is added to the input, resulting in the final output of the sub-layer. The output of the last encoder layer is then fed into a decoder, which also consists of multiple boxes arranged horizontally, each box representing a different attention head. The attention mechanism is represented by arrows connecting the encoder and decoder layers. The decoder layer also includes a self-attention mechanism, a multi-head encoder-decoder attention mechanism, and a position-wise fully connected feed-forward network. The output of the decoder layer is passed through an additional output block for token probabilities and a softmax activation block. The diagram also includes two additional layers: "Input-Input Layer5" and "Attention-Attention Layer6," which show boxes representing the attention heads*

> *involved in anaphora resolution, following long-distance dependencies in the encoder self-attention.*

Translating into Mermaid code

Let's use again the `mermaid_qa` we defined earlier. Here is the query I am using:

```
query = """
Your job is to write the code to generate a colorful mermaid diagram
using the following diagram description and context.
Use subgraphs if the diagram becomes too complex. Make sure the diagram
has a greater height than width.
Return only the code and make sure the diagram has multiple colors.

DESCRIPTION:
-----------
{description}
-----------

CONTEXT
----------
{context}
----------

CODE:
"""
```

We run through the different concepts:

```
for expl_dict in explanations:

    result = mermaid_qa.run(
        query.format(
            description=expl_dict['description'],
            context=expl_dict['explanation']
        )
    )
```

```
    expl_dict['code'] = result
```

Example of an article explained with ChatGPT

The following texts and diagrams are ChatGPT's explanation of the "Attention is all you need!" article. It is not perfect nor all the constraints are respected but we are getting there!

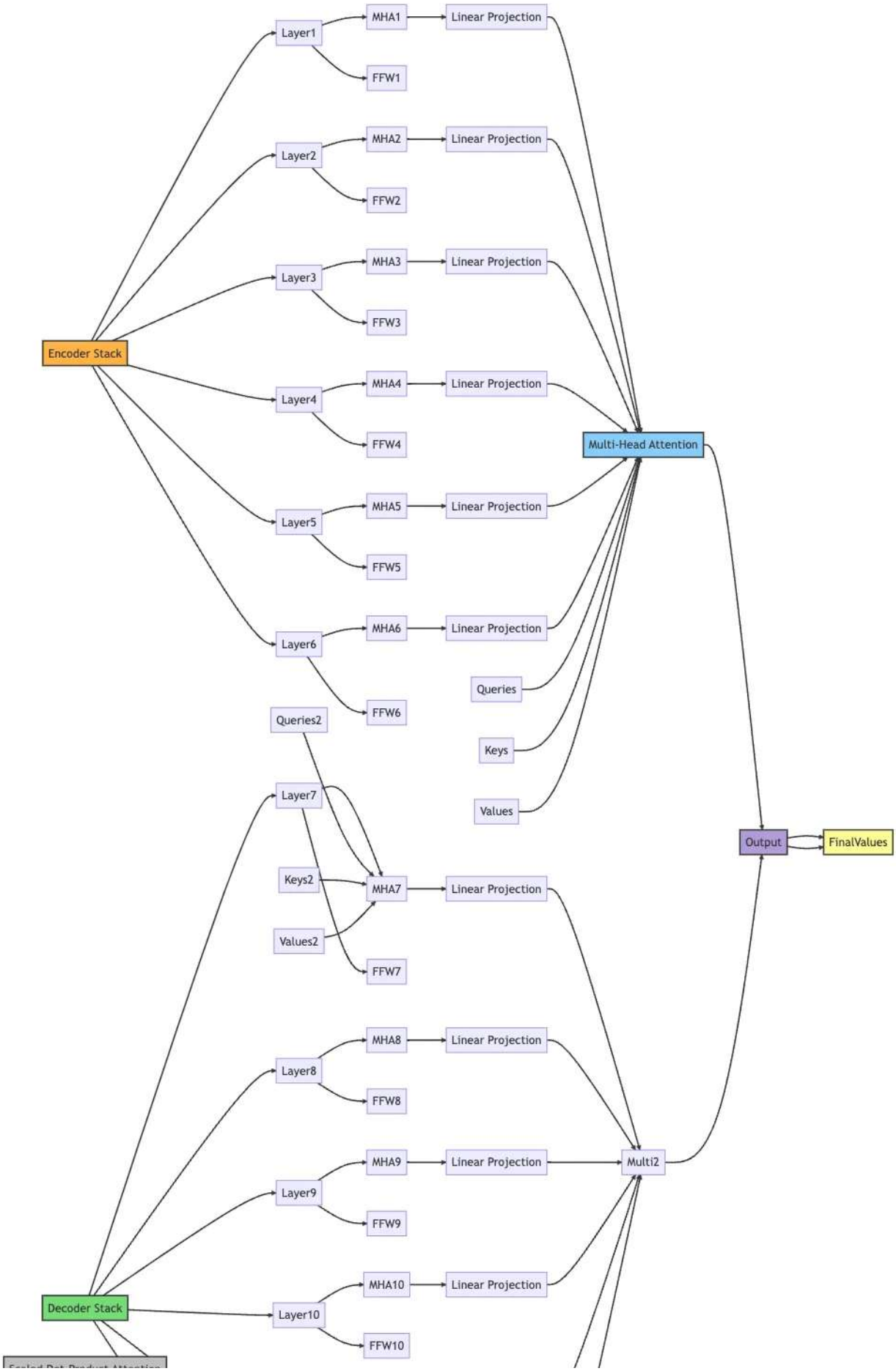
Transformer

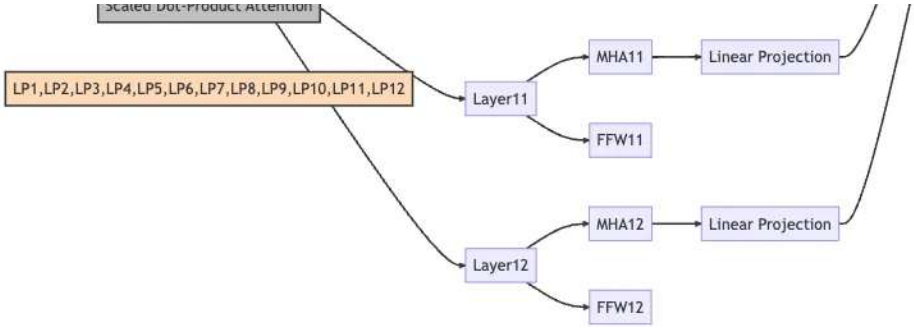
The Transformer is a type of neural network architecture that is commonly used in natural language processing tasks, such as language translation and text generation. Unlike traditional recurrent neural networks, which process sequences one word at a time, the Transformer is based on the self-attention mechanism, allowing it to consider all words in the sequence simultaneously.

The Transformer consists of an encoder and a decoder, each composed of multiple layers of self-attention and feedforward neural networks. The encoder processes the input text and generates a sequence of hidden representations, while the decoder takes these representations as input and generates the output text.

By allowing each position in the input and output sequences to attend to all positions in the sequence, rather than just the previous positions, the Transformer is able to better capture the dependencies between words in the input and output sequences. Additionally, the Transformer is highly parallelizable, allowing for faster training and inference on modern hardware.

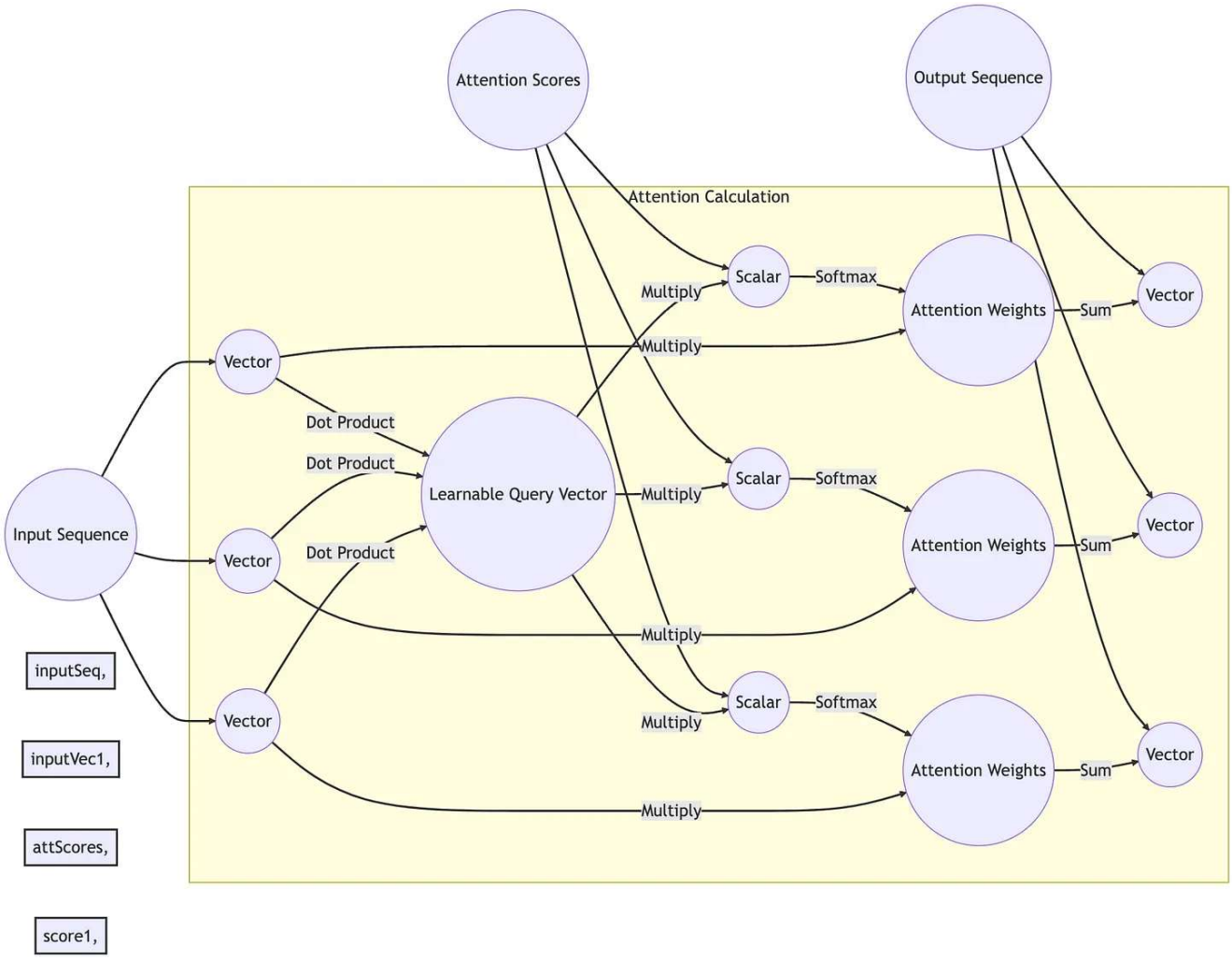
Overall, the Transformer has proven to be a highly effective model architecture for sequence transduction tasks, achieving state-of-the-art results on various natural language processing benchmarks, and has become a foundational technology in the field of NLP.


Encoder Stack
Layer1
MHA1
Linear Projection
FFW1
Layer2
MHA2
Linear Projection
FFW2
Layer3
MHA3
Linear Projection
FFW3
Layer4
MHA4
Linear Projection
FFW4
Layer5
MHA5
Linear Projection
FFW5
Layer6
MHA6
Linear Projection
FFW6
Multi-Head Attention
Queries
Keys
Values
Queries2
Layer7
Keys2
Values2
MHA7
Linear Projection
FFW7
Output
FinalValues
Layer8
MHA8
Linear Projection
FFW8
Layer9
MHA9
Linear Projection
FFW9
Multi2
Decoder Stack
Layer10
MHA10
Linear Projection
FFW10



Self-attention

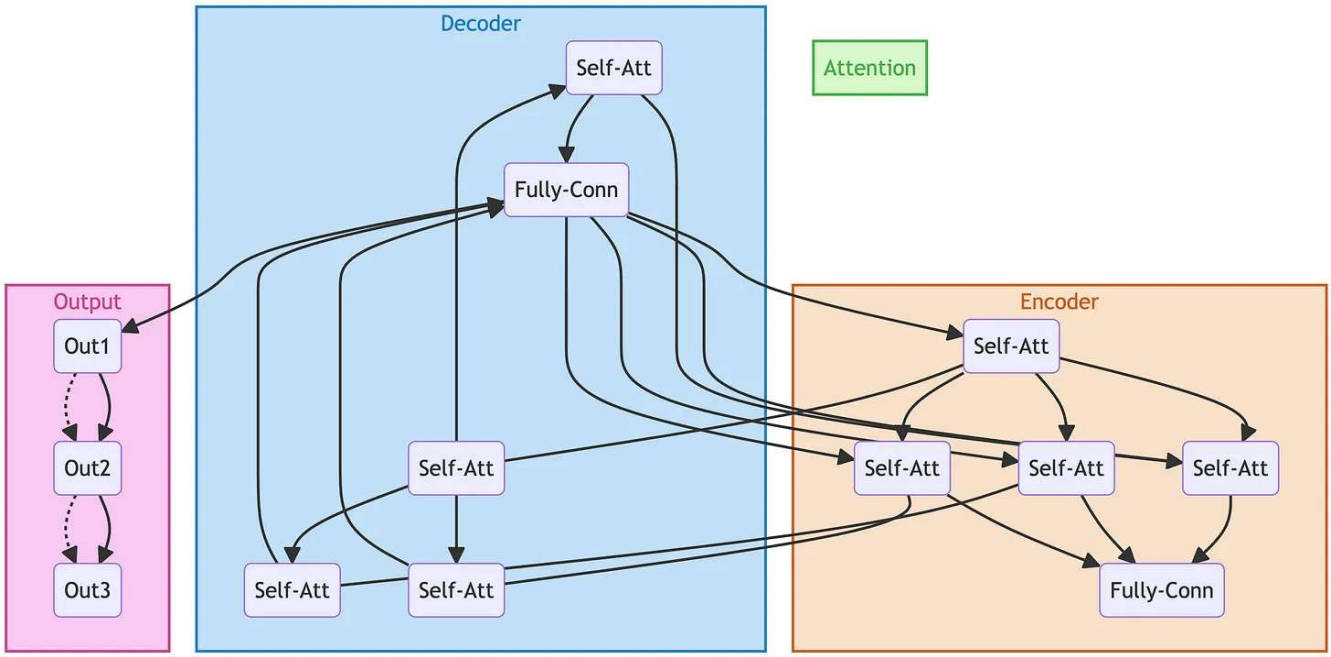
Self-attention is a mechanism used in machine learning, particularly in natural language processing tasks, that allows a model to weigh the importance of different parts of the input sequence when producing a certain output. It works by calculating a set of attention scores that measure how much each input element contributes to the generation of the output element. These scores are then used to compute a weighted sum of the input elements, which is used as the input to the next layer of the model. This allows the model to focus on the most relevant parts of the input, creating a representation that captures the relationships between different elements of the sequence. Self-attention has been shown to be particularly effective in capturing long-term dependencies in input sequences and improving performance in a variety of natural language processing tasks. It is a key component of the Transformer model, which is a state-of-the-art neural network architecture used for tasks such as language translation and text generation.



Attention mechanisms

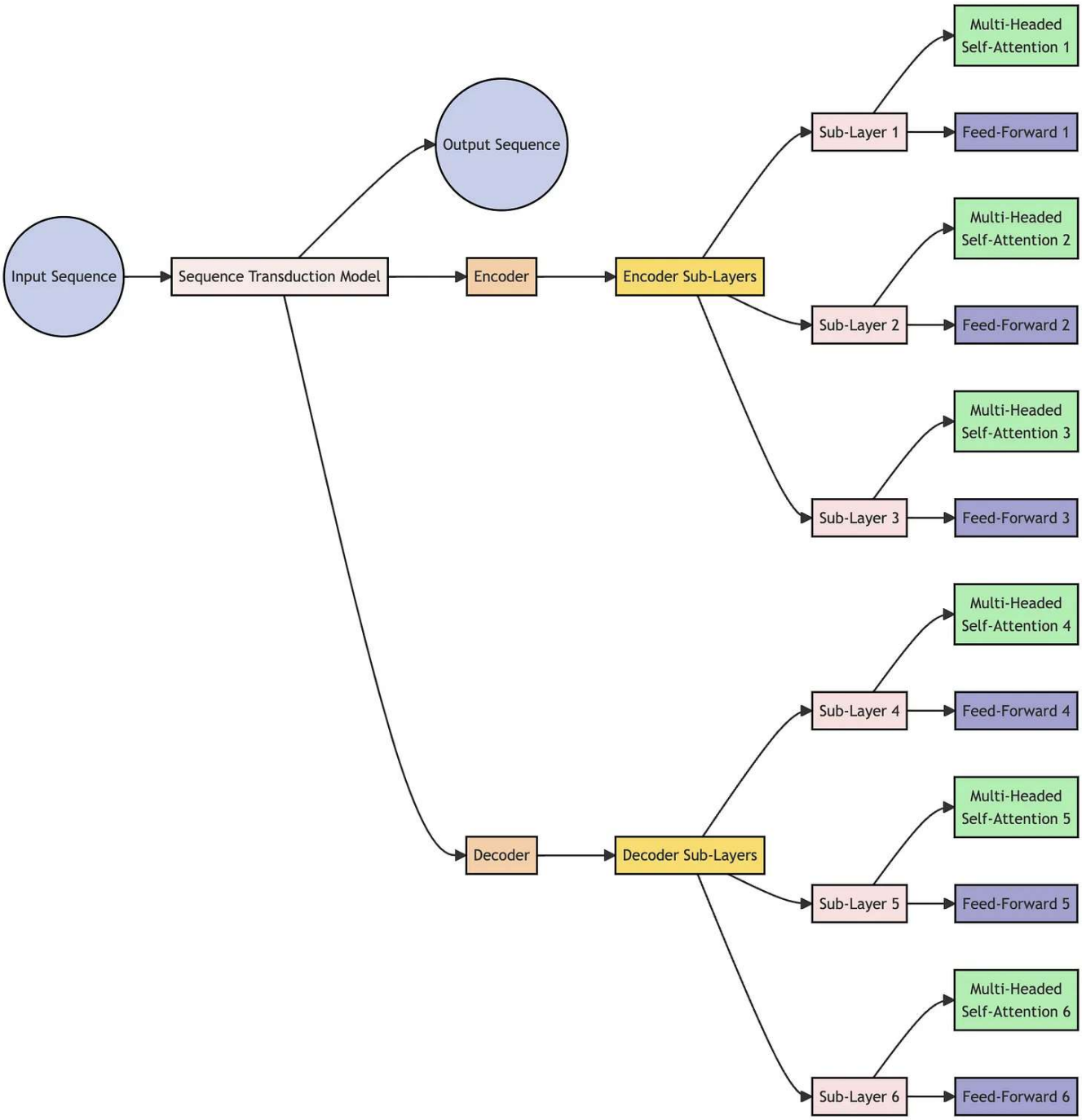
Attention Mechanisms are a type of neural network architecture that enable machine learning models to focus on specific parts of the input data during processing. They allow the model to selectively attend to different parts of the input, based on their relevance to the task at hand, instead of treating all input elements equally. Attention mechanisms assign different levels of importance or weights to different input elements, allowing the model to selectively attend to the most relevant information. This is particularly useful in tasks like natural language processing, where the model needs to pay attention to different words or phrases in the input sentence to generate accurate output. Attention mechanisms have been shown to be effective in a wide range of tasks including machine translation, speech recognition, image captioning, and more. They can significantly improve the accuracy and efficiency of machine learning models,

especially in cases where long-range dependencies need to be captured, or where the input or output sequences are very long.



Sequence transduction models

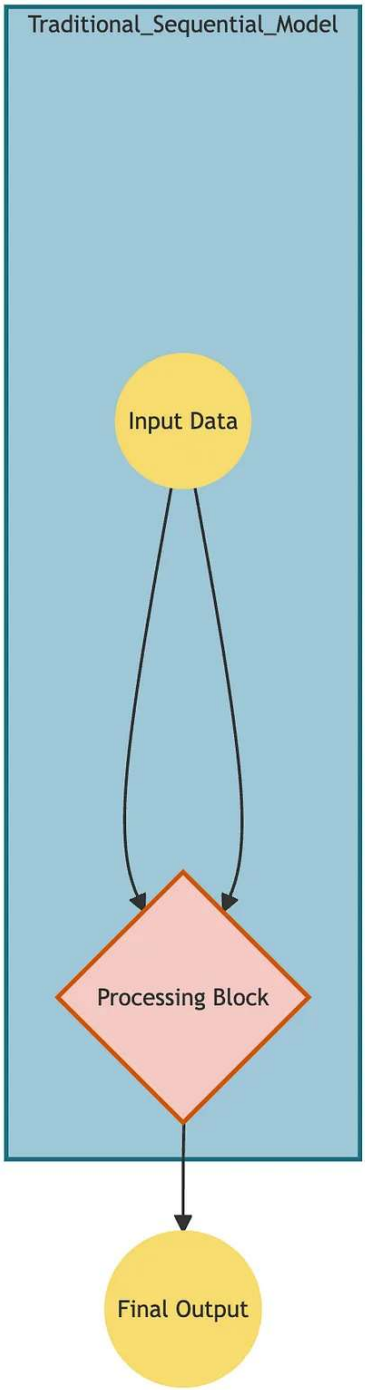
Sequence transduction models are a type of machine learning model used to transform one sequence of data into another sequence of data. They are commonly used in natural language processing tasks such as language translation, where an input sequence in one language is transduced into an output sequence in another language. These models typically consist of an encoder that processes the input sequence and a decoder that generates the output sequence. The encoder and decoder can be based on different architectures, such as recurrent neural networks or transformers, and can be trained using various techniques, such as maximum likelihood estimation or reinforcement learning. Sequence transduction models have shown significant improvements in computational efficiency and translation quality.

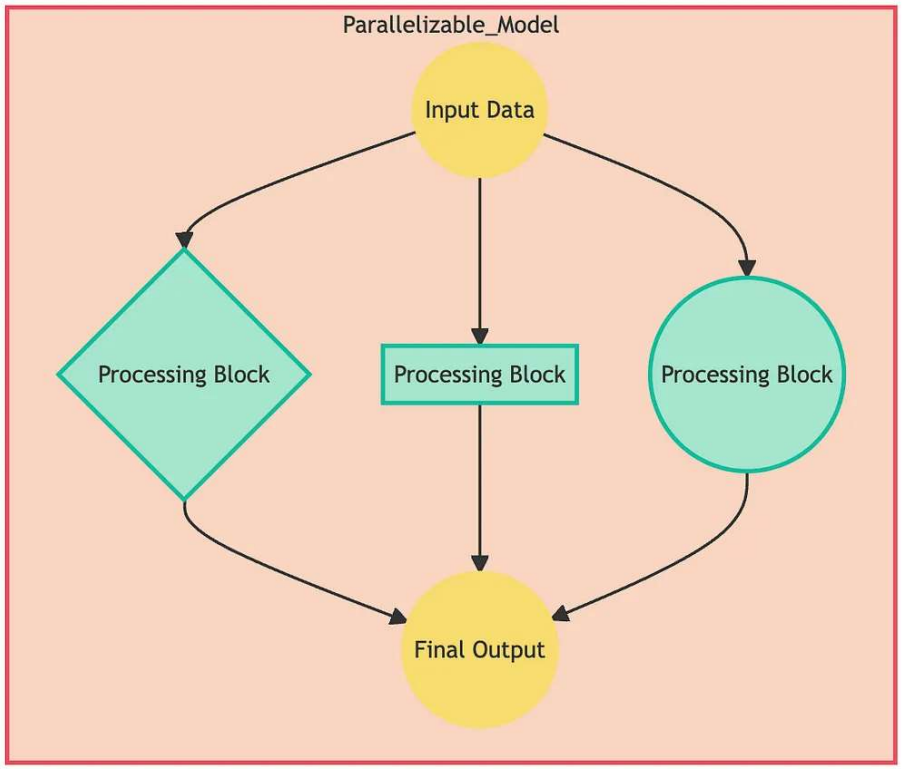


Parallelizable models

In machine learning, parallelizable models are those that can be trained using parallel processing techniques. This involves breaking down the training process into smaller tasks that can be executed simultaneously on different processors or machines, allowing for faster training times and more efficient use of computing resources. This is particularly important for large datasets or complex models that would otherwise require a significant amount of time and resources to train on a single processor.

Examples of parallelizable models include neural networks with multiple layers, decision tree algorithms, and models that rely on attention mechanisms to draw global dependencies between input and output. By using parallel computing, we can significantly reduce the time required to train these models and make them more scalable.





That's all Folks!

14 Likes · 1 Restack

Comments

Write a comment...