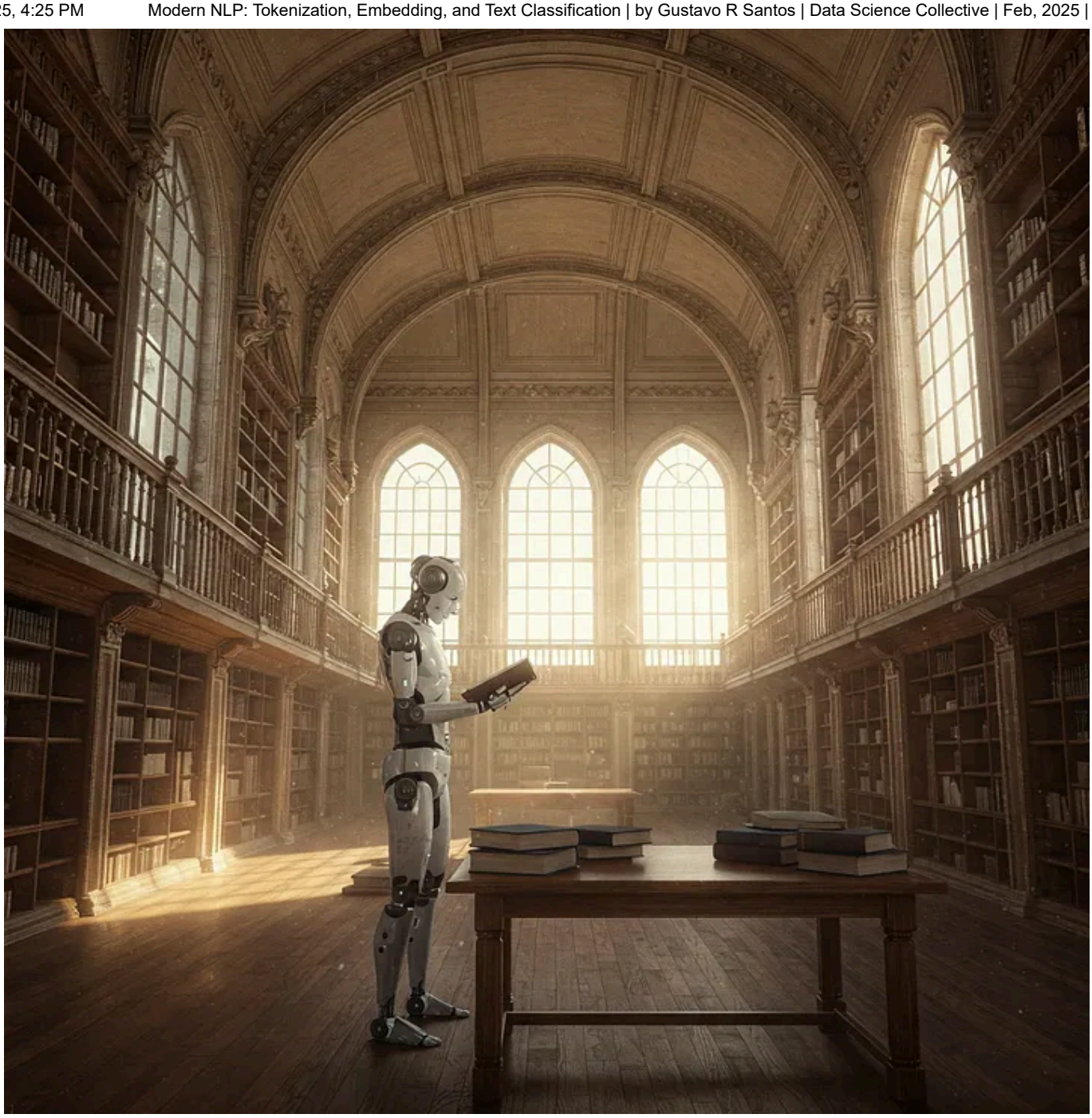
Modern NLP: Tokenization, Embedding, and Text Classification

Gustavo R Santos
Published in Data Science Collective
9 min read · Feb 28, 2025

Learning modern Natural Language Processing with Python code.



Text to Numbers | Image generated by AI. Gemini 3, 2025. https://gemini.google.com

Introduction

Natural Language Processing (NLP) has evolved so much. The Large Language Models (LLM) are the best proof of that. Thanks to modern techniques, it is possible to chat with the computer just like chatting with any human being.

I remember the *old-school* packages for NLP, such as `nltk` and `spacy`, and I miss their simplicity. Packages like that helped me with many projects where I was able to find patterns in texts, discover keywords, and even analyze some context with n-grams (patterns of n words that repeat in sequence within a text). Not to mention the beautiful word clouds, that were so popular for a while.

But those days are gone. Old-school packages still have their place in text analysis, but when we're talking about LLMs, there are modern techniques of NLP that we will be going over in this article.

Let's get to them.

Modern NLP Concepts

In this article, here is what we will cover:

- How LLMs process text using tokenization
- What are embeddings
- How the Attention Mechanism works in Transformers
- How to fine-tune a pre-trained LLM (BERT) on a text classification task

How LLMs process text using tokenization

The more I learn about computers and programming languages, the more I see how many technologies were developed from mirroring human activities. For example, the decision process that we go over in our heads inspired the Decision Tree algorithm.

The same is valid for LLMs and how they process language. Let's think about how we learn to talk: (1) we hear sounds; (2) we start associating sounds (words) with objects; (3) we start associating sounds with actions and they start to mean something; (4) we repeat the sound, then we are talking.

Every new sound we match to a meaning is incorporated into our vocabulary. Later, when we need to communicate, we access our vocabulary and use our words. After much training over the years, we develop a very complex network of phrases and meanings using our vocabulary.

Now, taking this process to the computers, and how they process text, the scientists who developed the LLMs mirrored that process with success.

1. Computers take in text.
2. Break it down into tokens — which is the smallest piece of a phrase with meaning. Usually, that's a word or a part of the word.
3. Next, the computers have to make sense of those tokens. And what do they process best? **Numbers**! So each token will be transformed into a vector of numbers.
4. Those vectors of numbers will be used for the next phase, which is the embeddings, allowing computers to understand the relationships between words.

Let's see that in action with some coding.

```
# Imports
from transformers import BertTokenizer

# Load pre-trained BERT tokenizer
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

# Sample text for tokenization
```

```
text = "Large Language Models are revolutionizing NLP."

# Step 1: Tokenize the text
tokens = tokenizer(text, return_tensors="pt", padding=True, truncation=True)
# View
tokens
```

The result looks like this, next.

```
{'input_ids': tensor([[  101,  2312,  2653,  4275,  2024,  4329,  6026, 17953,  2361,  1012,
           102]]),
'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]),
'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
```

- Notice how each word was transformed into an `id`
- The word *revolutionizing* was probably broken down into two subwords
- The ID 101 is associated with the token `[CLS]` . That token's vector is thought to capture the overall meaning or information of the entire sentence or sequence of sentences. It is like a stamp that indicates to the LLMs the meaning of that chunk.
- The ID 102 is associated with the token `[SEP]` to separate sentences.

Let's move on and understand how those tokens can be used.

What are embeddings

The computer now has its vocabulary. It knows the tokens, and they mean something by themselves. The next step in the "*learning how to talk process*" of the computer is making sense of all that database (vocabulary) and creating relationships between the words and their meanings.

There are two interesting ways of thinking about embedding.

1. Think of a map. If I just show it to you and say that NYC and Los Angeles are there, that does not help much because you don't know where or how far they are from each other. Now if I give you the coordinates, you can see they are on two opposite sides of the same country. So the *embedding* will create those coordinates for each token in a way that makes it easier for the LLM to know how close or far their meanings are.
2. You can think of an *embedding vector* as a "meaning profile" of a word. Each number in the vector tells you how much the word "scores" on a particular aspect of meaning. Imagine you have a 3-dimensional embedding vector. The vector for "**dog**" could be [0.8, 0.2, 0.1], where: 0.8 means it scores high on "animal"; 0.2 means it scores low on "food"; 0.1 means it scores very low on "action".

Let's code this.

```
# Import necessary libraries
from transformers import AutoTokenizer, AutoModel
import torch
```

```
# Load a pre-trained tokenizer and model (BERT in this case)
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
model = AutoModel.from_pretrained("bert-base-uncased")

# Sample text for tokenization
text = "Large Language Models are revolutionizing NLP."

# Step 1: Tokenize the text
tokens = tokenizer(text, return_tensors="pt", padding=True, truncation=True)

# Step 2: Generate embeddings using the model
with torch.no_grad():  # No gradient calculation needed for inference
    output = model(**tokens)

# Step 3: Extract the embeddings (CLS token represents the sentence embedding)
embedding = output.last_hidden_state[:, 0, :]

# This is what the text became. A tensor with numbers for the LLM to understand.
embedding
```

And the embedding returns a tensor that looks like this.

```
tensor([[-3.4580e-01, -1.7613e-01,  9.1574e-02, -3.1258e-01, -4.3672e-01,
         -4.6326e-01,  3.5836e-01,  6.3136e-01, -1.6741e-01, -6.0901e-01,
          1.5760e-02,  2.0739e-01, -3.9844e-01,  1.2286e-01,  2.5884e-01,
          2.2635e-01, -2.9151e-01,  5.7592e-01,  2.7204e-01,  7.5218e-02,
         -3.6411e-01, -6.5309e-01,  1.4161e-01, -5.1403e-02,  6.1204e-02,
         -1.0301e-01,  8.3905e-02, -3.2397e-01,  4.2635e-02, -2.5525e-01,
          ...
         -1.9017e-01, -1.5907e-01,  4.7907e-02,  5.5601e-01,  7.5720e-02,
         -4.8495e-01,  1.6084e-02,  3.4603e-01]])
```

Having this tensor, the LLM now can put that sentence in context with other text, making it easier to build complex text output.

Why embeddings are crucial for NLP tasks

Words to Numbers | Image generated by AI. Meta, 2025. https://meta.ai

As previously stated, computers deal with numbers, not words. Embeddings bridge this gap by converting words into numerical vectors.

Thus, embeddings are designed in a way that similar meanings have similar vectors — i.e., they're close together in the vector space.
This allows NLP models to understand semantic relationships like synonyms, antonyms, and related concepts. Modern embeddings, especially those from transformer models (like (Bidirectional Encoder Representations from Transformers — **BERT**), go even further by capturing contextual meaning.

> Embeddings transform text from words into meaningful numerical representations for machines, capturing semantic relationships, reducing dimensionality, and enabling better generalization. They are the bedrock of modern NLP, allowing models to understand and process human language with unprecedented accuracy.

How the Attention Mechanism works in Transformers

In the famous paper *Attention is All You Need,* the authors defined **attention** as *a function that maps a query and a set of key-value pairs to an output, where they are all vectors. The output is computed as a* ***weighted sum of the values****, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.*

In simple words when you're searching for something and you question the LLM (**query**). The model was trained on a bunch of labeled information (**keys and values**), and you want an answer (**output**).

Attention figures out which labels are most relevant to your question, and gives you an answer by combining the information from those relevant labels.

Next, let's finish with the code of a BERT model tuned to classify text.

How to fine-tune a pre-trained LLM (BERT) on a text classification task

We can use BERT pre-trained models to classify text, such as movie reviews. Let's see how that's done.

We will use the open dataset from IMDB.

```
pip install datasets --quiet
```

Then, we import some modules.

```
# Import necessary libraries
import torch
from torch.utils.data import DataLoader
from transformers import BertTokenizer, BertForSequenceClassification
from torch.optim import AdamW
from datasets import load_dataset
import numpy as np
from transformers import DataCollatorWithPadding, TrainingArguments, Trainer

# Check if GPU is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

If your text is in English, the tokenizer `bert-base-uncased` is a common starting point. For other languages, you'll need a multilingual BERT `bert-base-multilingual-cased` or a language-specific BERT.

If your text is from a specialized domain (e.g., medical, legal, scientific), consider using a BERT model that has been pre-trained on that domain.

```
# Load pre-trained BERT tokenizer
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

# Load a sample sentiment classification dataset (IMDB reviews)
dataset = load_dataset("imdb")
```

Next, we must tokenize our data. So we create a simple function to do that for us, breaking the text into chunks of a maximum of 128 characters. When it's more, the chunk is truncated. When it's less, it adds some pads.

```
# Tokenization function
def tokenize_function(examples):
    tokenized = tokenizer(examples["text"],
                          padding="max_length",
                          truncation=True,
                          max_length=128)

    return {key: torch.tensor(val) for key, val in tokenized.items()} #Convert to tensors here.
```

```
# Tokenize dataset
tokenized_datasets = dataset.map(tokenize_function, batched=True)
```

To make it fast, we will grab just a few observations from our data.

```
# Convert to PyTorch dataset format
train_dataset = tokenized_datasets["train"].shuffle(seed=42).select(range(20))  # Use a subset for qu
test_dataset = tokenized_datasets["test"].shuffle(seed=42).select(range(10))
```

This is a sample.

```
train_dataset['text'][1]
```

```
This movie is a great. The plot is very true to the book which is a classic
written by Mark Twain. The movie starts of with a scene where Hank sings a
song with a bunch of kids called "when you stub your toe on the moon"
It reminds me of Sinatra's song High Hopes, it is fun and inspirational.
The Music is great throughout and my favorite song is sung by the King, Hank
 (bing Crosby) and Sir "Saggy" Sagamore. OVerall a great family movie or even
a great Date movie. This is a movie you can watch over and over again.
 The princess played by Rhonda Fleming is gorgeous. I love this movie!!
 If you liked Danny Kaye in the Court Jester then you will definitely
like this movie.
```

This code freezes most of a pre-trained model to preserve its general language knowledge and saves computational resources. It then unfreezes a specific part (the “pooler” layer) to allow the model to adapt to your specific task.

A quick note here, when you run this cell, you might be asked to create a *wandb* API Key. Just go to this page and follow the instructions.

```
# Load pre-trained BERT model for classification (2 classes: Positive/Negative)
model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=2).to(device)

# freeze all base model parameters
for name, param in model.base_model.named_parameters():
    param.requires_grad = False

# unfreeze base model pooling layers
for name, param in model.base_model.named_parameters():
    if "pooler" in name:
        param.requires_grad = True
```

We set the hyperparameters.

```
# hyperparameters
lr = 2e-4
batch_size = 8
num_epochs = 10

training_args = TrainingArguments(
    output_dir="bert-classifier_movie",
    learning_rate=lr,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=num_epochs,
    logging_strategy="epoch",
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
)

# Instance of Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    tokenizer=tokenizer
)

# Train the model
trainer.train()
```

And finally, we get the results.

```
# apply model to validation dataset
predictions = trainer.predict(test_dataset)

# Check if the prediction matches with the test label
predictions.label_ids == test_dataset['label']

# --- Results---
# array([ True,  True,  True,  True,  True,  True,  True,  True,  True, True])
```

Before You Go

Well, we covered a lot in this article.

The modern NLP is on the hype lately. There's so much being done with this technology and the news keeps showing up.

In this post, I wanted to give you an easy understanding of how LLMs take input like textual queries and return such amazing results, helping us in many daily tasks.

Oh, and a final hint: take a look at this visual guide of BERT *by Jay Alammar*. It's very well-explained. A great resource to study.

If you liked this content, follow me for more about Data Science.

Gustavo R Santos - Medium

Read writing from Gustavo R Santos on Medium. Data Scientist | I solve business challenges through the power of data. |...

gustavorsantos.medium.com

Contacts & Portfolio

Gustavo R Santos

I am a Data Scientist specializing in data analysis, machine learning, and visualization using Python, R, SQL, and...

gustavorsantos.me

References

Tokenization in large language models, explained

Modern language models predict "tokens", not words-but what exactly are tokens?

seantrott.substack.com

A Visual Guide to Using BERT for the First Time

Translations: Chinese, Korean, Russian Progress has been rapidly accelerating in machine learning models that process...

jalammar.github.io

Fine-Tuning BERT for Text Classification | Towards Data Science

A hackable example with Python code

towardsdatascience.com

How Text Embeddings help suggest similar words

The world is filled with fascinating technology. It can feel overwhelming to see such extravagant machines and systems...

pathway.com

NLP Bert Data Science Python LIm

Following

Published in Data Science Collective

834K Followers · Last published 3 hours ago

Advice, insights, and ideas from the Medium data science community

Edit profile

Written by Gustavo R Santos

3K Followers · 34 Following

Data Scientist | I solve business challenges through the power of data. | Visit my site: https://gustavorsantos.me

Responses (5)

Gustavo R Santos

What are your thoughts?

Jack Marks
Mar 3

Really nice summary, thank you!

2 1 reply Reply

Andulile
Feb 28

Great; in the world of software development. I know Token, but never knew how the software developers making it the way it functions or applied.

Thanks for sharing this important thing in the world of Info Tech.

2 1 reply Reply

Andargachewalayu
Mar 9

dep learning information retrieval in amharic language python code

Reply

See all responses

More from Gustavo R Santos and Data Science Collective

In Data Science Collective by Gustavo R Santos

Vector Databases and Search By Similarity for NLP

Learn about vector databases and how they can help your data science projects.

Mar 14 100

In Data Science Collective by Paolo Perrone

Chart Smarter: How to Design Data Visualizations That Work

Our brains are built for visuals.

Mar 4 1.3K 19

In Data Science Collective by Eric Broda

Agentic Mesh: Building Highly Reliable Agents

LLMs are getting overloaded. Specialized LLMs, with deterministic orchestration & an agent architecture offer a more reliable path forward.

Mar 5 578 10

In AI Advances by Gustavo R Santos

The Next Level of Pandas read_csv()

9 underused parameters of the well-known Pandas' read_csv() method.

Nov 28, 2024 431 3

See all from Gustavo R Santos

See all from Data Science Collective

Recommended from Medium

Sebastian Carlos

Fired From Meta After 1 Week: Here's All The Dirt I Got

This is not just another story of a disgruntled ex-employee. I'm not shying away from the serious corporate espionage or the ethical...

Jan 8 18.8K 415

In Level Up Coding by Dr. Ashish Bamania

Chain-of-Draft (CoD) Is The New King Of Prompting Techniques

A deep dive into the novel Chain-of-Draft (CoD) Prompting that reducing LLM inference cost and latency like never before.

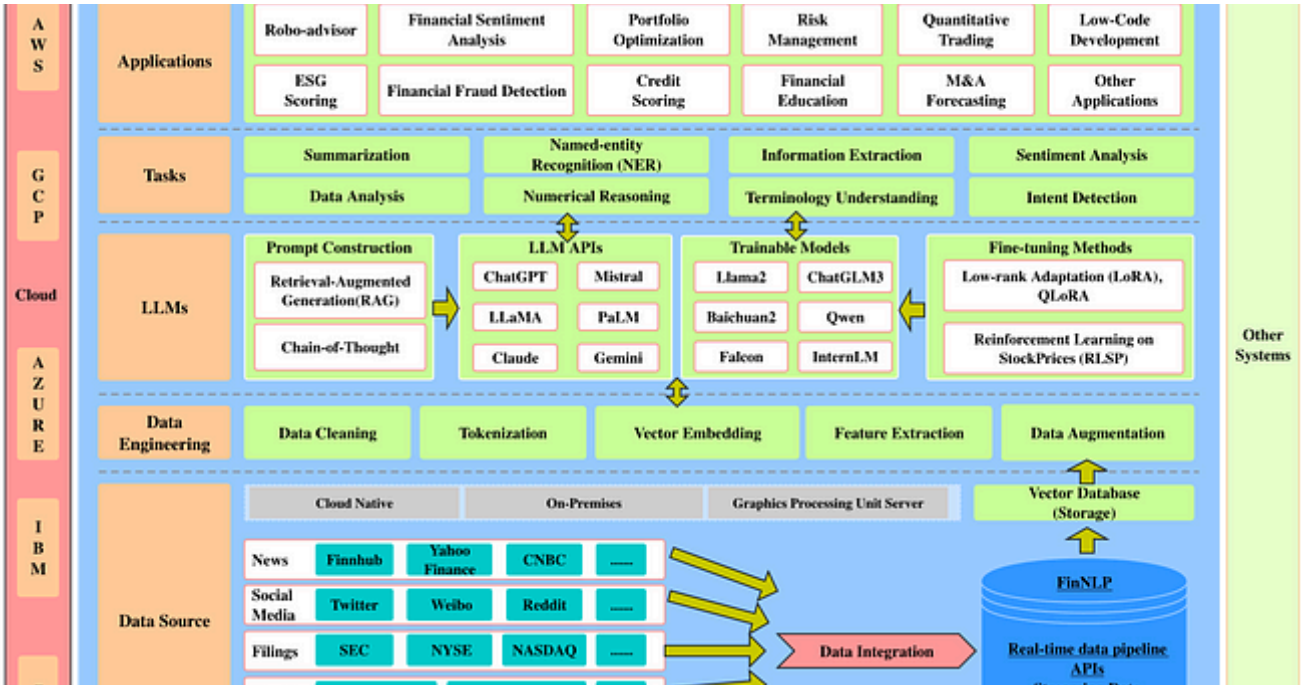
Mar 3 2.2K 32

In Everyday AI by Manpreet Singh

Craziest MCP Servers You Must Try

I remember when I first heard about MCP (Model Context Protocol). I thought

Mar 9 1K 10

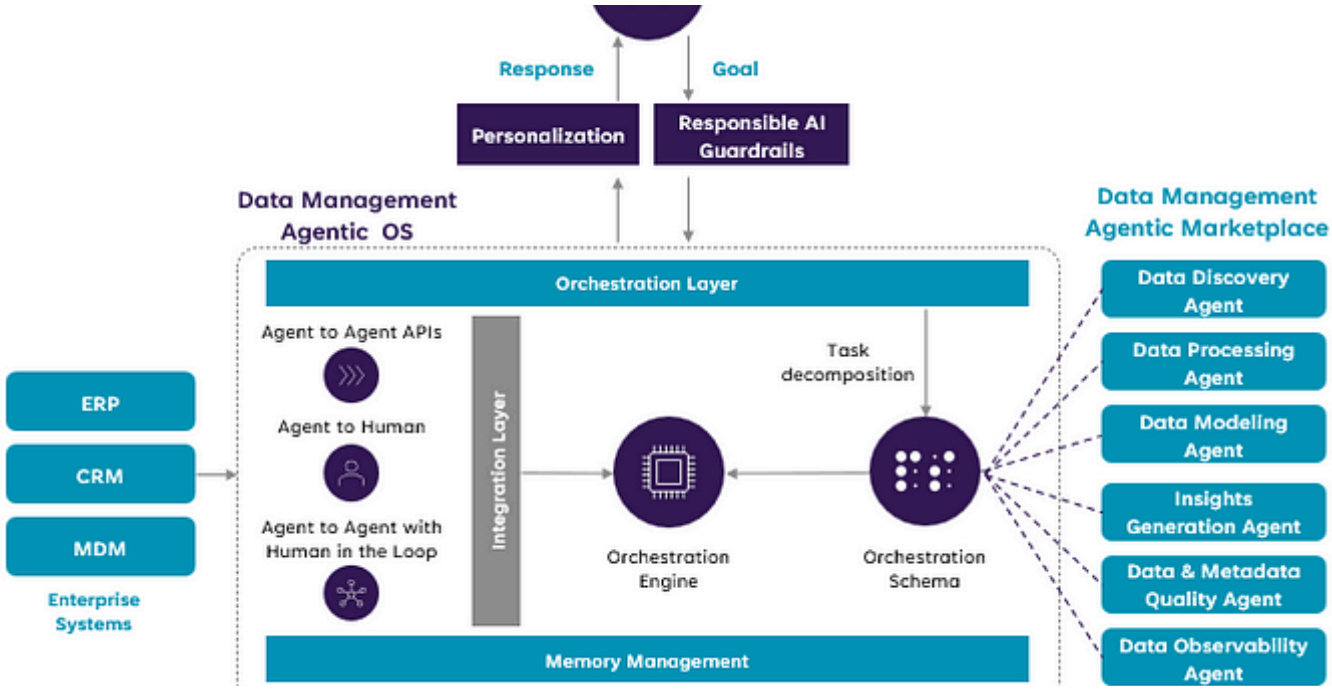


In AI monks.io by JIN

FinGPT: The Future of Financial Analysis—Revolutionizing Markets with Open-Source AI

Discover how FinGPT is disrupting traditional financial tools like Bloomberg Terminal, making powerful analytics accessible for everyone —...

Feb 16 915 14



In AI Advances by Debmalya Biswas

Agentic AI for Data Engineering

Reimagining Enterprise Data Management leveraging AI Agents

Mar 23 514 12

In Python in Plain English by Dhruv Ahuja

How I Learned to Love `__init__.py`: A Simple Guide 😊

💡 Heads Up! Click here to unlock this article for free if you're not a Medium member!

Feb 3 1.6K 14