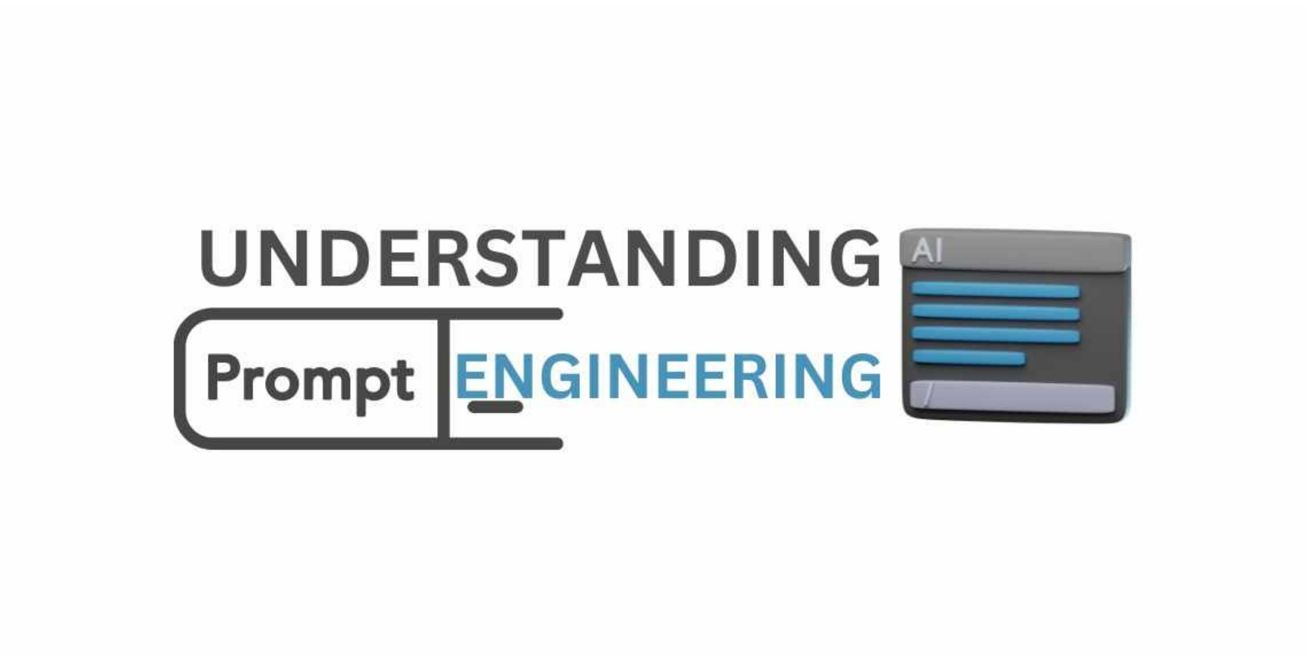
Basics of Prompt Engineering



In [34]:
```python
import warnings
warnings.filterwarnings('ignore')
```

In [35]:
```python
import nltk
nltk.download('punkt')

from nltk.tokenize import word_tokenize

sentence = "Hello, how are you?"
tokens = word_tokenize(sentence)
print(tokens)
```

```
['Hello', ',', 'how', 'are', 'you', '?']
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\sajid\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [36]:
```python
import nltk
nltk.download('averaged_perceptron_tagger')
from nltk import pos_tag
from nltk.tokenize import word_tokenize

sentence = "I love playing tennis"
tokens = word_tokenize(sentence)
tags = pos_tag(tokens)
print(tags)
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\sajid\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[('I', 'PRP'), ('love', 'VBP'), ('playing', 'VBG'), ('tennis', 'NN')]
```

In [37]:
```python
import nltk
nltk.download('maxent_ne_chunker')
nltk.download('words')

from nltk import ne_chunk
from nltk.tokenize import word_tokenize

sentence = "Barack Obama was born in Hawaii"
tokens = word_tokenize(sentence)
entities = ne_chunk(pos_tag(tokens))
print(entities)
```

```
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     C:\Users\sajid\AppData\Roaming\nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\sajid\AppData\Roaming\nltk_data...
[nltk_data]   Package words is already up-to-date!
```

```
(S
  (PERSON Barack/NNP)
  (PERSON Obama/NNP)
  was/VBD
  born/VBN
  in/IN
  (GPE Hawaii/NNP))
```

In [38]:

```
import nltk
nltk.download('vader_lexicon')

from nltk.sentiment import SentimentIntensityAnalyzer

sentence = "I enjoyed the movie. It was great!"
analyzer = SentimentIntensityAnalyzer()
sentiment_scores = analyzer.polarity_scores(sentence)
print(sentiment_scores)
```

```
{'neg': 0.0, 'neu': 0.342, 'pos': 0.658, 'compound': 0.8268}
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\sajid\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

In [39]:

```
import nltk
nltk.download('wordnet')

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
word = "running"
lemma = lemmatizer.lemmatize(word, pos='v')
print(lemma)
```

```
run
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\sajid\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [40]:

```
from textblob import TextBlob

# Create a TextBlob object
text = "I love this movie, it's fantastic!"
blob = TextBlob(text)

# Perform sentiment analysis
sentiment = blob.sentiment
print(sentiment.polarity)  # Polarity ranges from -1 to 1, indicating negative to positive sentiment
print(sentiment.subjectivity)  # Subjectivity ranges from 0 to 1, indicating objective to subjective sentiment
```

```
0.5
0.75
```

In [41]:

```
from textblob import TextBlob

# Create a TextBlob object
text = "The cat is sitting on the mat."
blob = TextBlob(text)

# Perform part-of-speech tagging
pos_tags = blob.tags
print(pos_tags)
```

```
[('The', 'DT'), ('cat', 'NN'), ('is', 'VBZ'), ('sitting', 'VBG'), ('on', 'IN'), ('the', 'DT'), ('mat', 'NN')]
```

In [42]:

```
from textblob import TextBlob

# Create a TextBlob object
text = "The cat is sitting on the mat."
blob = TextBlob(text)

# Extract noun phrases
noun_phrases = blob.noun_phrases
print(noun_phrases)
```

```
[]
```

In [43]:

```
from textblob import TextBlob
from textblob.classifiers import NaiveBayesClassifier

# Training data
train_data = [
    ('I love this product!', 'positive'),
    ('This is terrible.', 'negative'),
    ('The service was great.', 'positive'),
    ('I did not like the movie.', 'negative')
]

# Create a TextBlob classifier
classifier = NaiveBayesClassifier(train_data)
```

```
# Test data
test_data = [
    'I had a wonderful experience.',
    'The food was delicious.',
    'I dislike the new update.'
]

# Classify the test data
for text in test_data:
    result = classifier.classify(text)
    print(f'Text: {text}\nClassification: {result}\n')
```

```
Text: I had a wonderful experience.
Classification: positive

Text: The food was delicious.
Classification: positive

Text: I dislike the new update.
Classification: negative
```

In [44]:

```
import spacy

nlp = spacy.load("en_core_web_sm")
sentence = "I love spaCy"
doc = nlp(sentence)

for token in doc:
    print(token.text, token.pos_)
```

```
I PRON
love VERB
spaCy VERB
```

In [45]:

```
import spacy

nlp = spacy.load("en_core_web_sm")
sentence = "Apple Inc. is looking to buy a startup in India."
doc = nlp(sentence)

for ent in doc.ents:
    print(ent.text, ent.label_)
```

```
Apple Inc. ORG
India GPE
```

In [46]:

```
import spacy

nlp = spacy.load("en_core_web_sm")
sentence = "I want to eat pizza"
doc = nlp(sentence)

for token in doc:
    print(token.text, token.dep_, token.head.text)
```

```
I nsubj want
want ROOT want
to aux eat
eat xcomp want
pizza dobj eat
```

In [47]:

```
import spacy

nlp = spacy.load("en_core_web_sm")
text = "This is the first sentence. This is the second sentence."
doc = nlp(text)

for sent in doc.sents:
    print(sent.text)
```

```
This is the first sentence.
This is the second sentence.
```

In [48]:

```
import spacy

nlp = spacy.load("en_core_web_sm")
token = nlp("running")[0]

print(token.lemma_)
```

```
run
```

In [49]:

```
from gensim import corpora

documents = ["I love coding",
             "Python is my favorite language",
             "Machine learning is interesting"]
```

```
# Tokenize the documents
tokenized_docs = [doc.split() for doc in documents]

# Create a dictionary from the tokenized documents
dictionary = corpora.Dictionary(tokenized_docs)

# Create a document-term matrix
doc_term_matrix = [dictionary.doc2bow(doc) for doc in tokenized_docs]

print(doc_term_matrix)
```

```
[[(0, 1), (1, 1), (2, 1)], [(3, 1), (4, 1), (5, 1), (6, 1), (7, 1)], [(5, 1), (8, 1), (9, 1), (10, 1)]]
```

In [50]:

```
from gensim import corpora, models

documents = ["I love coding",
             "Python is my favorite language",
             "Machine learning is interesting"]

# Tokenize the documents
tokenized_docs = [doc.split() for doc in documents]

# Create a dictionary from the tokenized documents
dictionary = corpora.Dictionary(tokenized_docs)

# Create a document-term matrix
doc_term_matrix = [dictionary.doc2bow(doc) for doc in tokenized_docs]

# Apply Latent Semantic Analysis
lsa_model = models.LsiModel(doc_term_matrix, num_topics=2)
lsa_topics = lsa_model.show_topics(num_topics=2, num_words=3)

for topic in lsa_topics:
    print(topic)
```

```
(0, '-0.581*"5" + -0.359*"7" + -0.359*"6"')
(1, '-0.463*"10" + -0.463*"8" + -0.463*"9"')
```

In [51]:

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased')

sentence = "This is a positive sentence."
input_ids = tokenizer.encode(sentence, add_special_tokens=True)
outputs = model(torch.tensor([input_ids]))[0]

predicted_label = torch.argmax(outputs).item()
print(predicted_label)
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncas
ed and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
0
```

In [52]:

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer

tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2')

input_text = "Once upon a time"
input_ids = tokenizer.encode(input_text, return_tensors='pt')
outputs = model.generate(input_ids, max_length=50, num_return_sequences=1)

generated_text = tokenizer.decode(outputs[0])
print(generated_text)
```

```
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Pl
ease pass your input's `attention_mask` to obtain reliable results.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Once upon a time, the world was a place of great beauty and great danger. The world was a place of great danger
, and the world was a place of great danger. The world was a place of great danger, and the world was a
```

In [53]:

```
from nltk import ngrams

# Sentence for n-gram modeling
sentence = "I love to explore natural language processing."

# Create a trigram model
n = 3
trigrams = ngrams(sentence.split(), n)

# Print the trigrams
for gram in trigrams:
    print(gram)
```

```
('I', 'love', 'to')
('love', 'to', 'explore')
('to', 'explore', 'natural')
('explore', 'natural', 'language')
('natural', 'language', 'processing.')
```

In [54]:

```
from nltk import ngrams

# Sentence for n-gram modeling
sentence = "The quick brown fox jumps over the lazy dog."

# Create a bigram model
n = 2
bigrams = ngrams(sentence.split(), n)

# Print the bigrams
for gram in bigrams:
    print(gram)
```

```
('The', 'quick')
('quick', 'brown')
('brown', 'fox')
('fox', 'jumps')
('jumps', 'over')
('over', 'the')
('the', 'lazy')
('lazy', 'dog.')
```

In [55]:

```
import nltk
from nltk import HiddenMarkovModelTagger

# Training data for HMM
train_data = [
    [('I', 'PRON'), ('love', 'VERB'), ('to', 'PART'), ('explore', 'VERB')],
    [('Natural', 'ADJ'), ('language', 'NOUN'), ('processing', 'NOUN')]
]

# Test sentence for tagging
test_sentence = ['I', 'like', 'to', 'read', 'books']

# Train the HMM model
hmm_tagger = HiddenMarkovModelTagger.train(train_data)

# Tag the test sentence
tagged_sentence = hmm_tagger.tag(test_sentence)

# Print the tagged sentence
print(tagged_sentence)
```

```
[('I', 'PRON'), ('like', 'VERB'), ('to', 'PART'), ('read', 'VERB'), ('books', 'PART')]
```

In [56]:

```
import nltk
from nltk.corpus import treebank
from nltk.tag import HiddenMarkovModelTrainer

# Training data for HMM (using Treebank corpus)
train_data = treebank.tagged_sents()[:500]  # Using first 500 sentences

# Test sentence for tagging
test_sentence = "The cat is sitting on the mat."

# Train the HMM model
hmm_trainer = HiddenMarkovModelTrainer()
hmm_tagger = hmm_trainer.train_supervised(train_data)

# Tag the test sentence
tagged_sentence = hmm_tagger.tag(test_sentence.split())

# Print the tagged sentence
print(tagged_sentence)
```

```
[('The', 'DT'), ('cat', 'NNP'), ('is', 'NNP'), ('sitting', 'NNP'), ('on', 'NNP'), ('the', 'NNP'), ('mat.', 'NNP
')]
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js