LLM Fundamentals Cheat Sheet

A comprehensive guide to Large Language Models

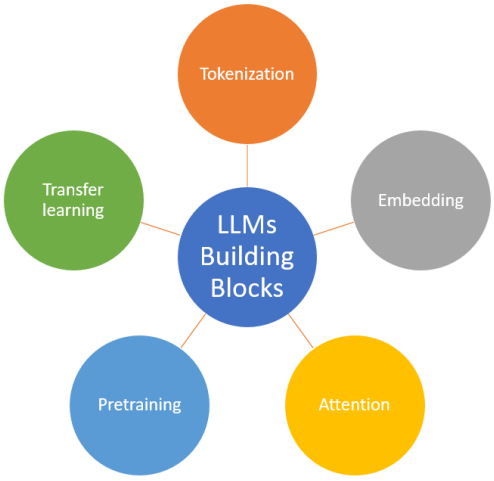
1. What is an LLM?

Large Language Models (LLMs) are advanced AI systems trained on massive text datasets to understand and generate human language.

> **Real-world example:**
>
> When you ask a question to ChatGPT or Claude, you're interacting with an LLM that processes your input and generates a contextually relevant response.

Python interaction example:

```
import openai

# Configure API key
openai.api_key = "your-api-key"

# Making a request to an LLM (GPT-4)
response = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[
        {"role": "user", "content": "Explain quantum computing in simple terms"}
    ]
)

print(response.choices[0].message.content)
```



2. Key Concepts

Tokens vs. Words

Tokens are the basic units LLMs use to process text. A token can be a word, part of a word, or even a single character.

Example:

- The sentence "I love machine learning" might be tokenized as:
  - Words: ["I", "love", "machine", "learning"] (4 words)
  - Tokens: ["I", "love", "machine", "learning"] (4 tokens)
- But "Supercalifragilisticexpialidocious" might be:
  - Words: ["Supercalifragilisticexpialidocious"] (1 word)
  - Tokens: ["Super", "calif", "ragi", "listic", "expial", "idocious"] (6 tokens)

Python example using HuggingFace tokenizers:

```
from transformers import GPT2Tokenizer

# Load a pre-trained tokenizer
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# Tokenize some text
text = "I love machine learning"
tokens = tokenizer.encode(text)

# Display tokens and token count
print(f"Tokens: {tokens}")
print(f"Token count: {len(tokens)}")
print(f"Decoded tokens: {[tokenizer.decode([token]) for token in tokens]}")
```

Context Window

The **context window** defines how much text an LLM can "see" and consider when generating responses.

Real-world example:

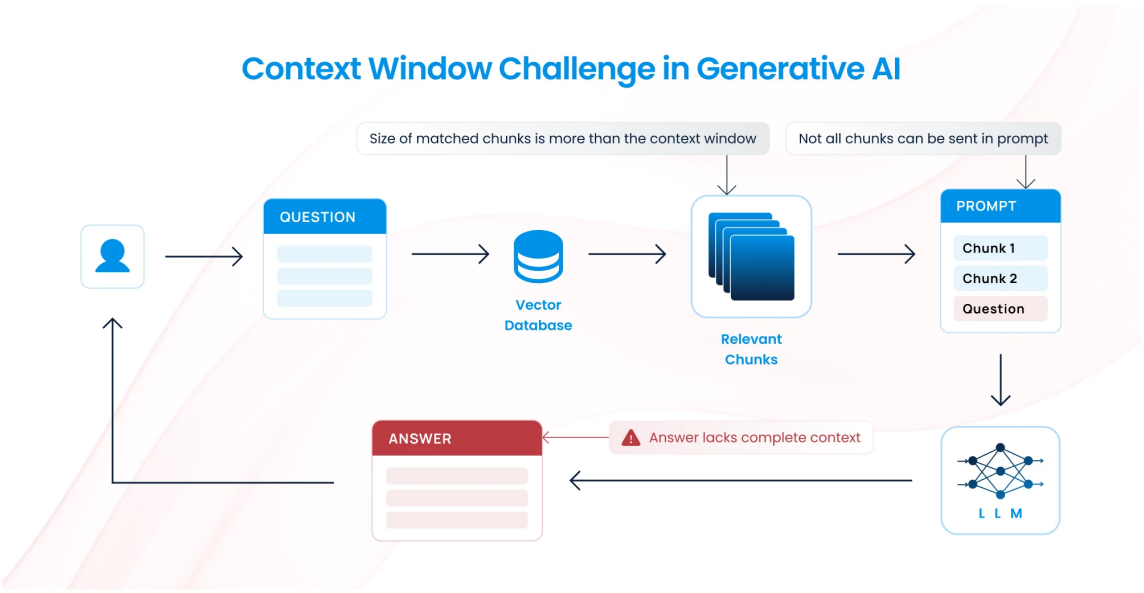
If an LLM has an 8K context window (~8,000 tokens), it can process about 16 pages of text at once. This means it can refer back to information from those 16 pages when creating its response.

Practical implications of context window size:

- **Small context window (4K):** Can process a few pages, suitable for question-answering tasks
- **Medium context window (16K):** Can process a research paper, better for summarization
- **Large context window (100K+):** Can process entire books, enables document analysis

Python example showing context limitation:

```
import anthropic

# Initialize Anthropic client
client = anthropic.Anthropic(api_key="your-api-key")

# Create a message with a large context
try:
    response = client.messages.create(
        model="claude-3-sonnet-20240229",
        max_tokens=1000,
        messages=[
            {
                "role": "user",
                "content": "Here's the full text of War and Peace: [extremely long
text]... Please summarize it."
            }
        ]
    )
    print(response.content)
except Exception as e:
    print(f"Error: {e}")  # Will likely error due to context window limits
```



3. Training Approaches

Pretraining vs. Finetuning

Pretraining

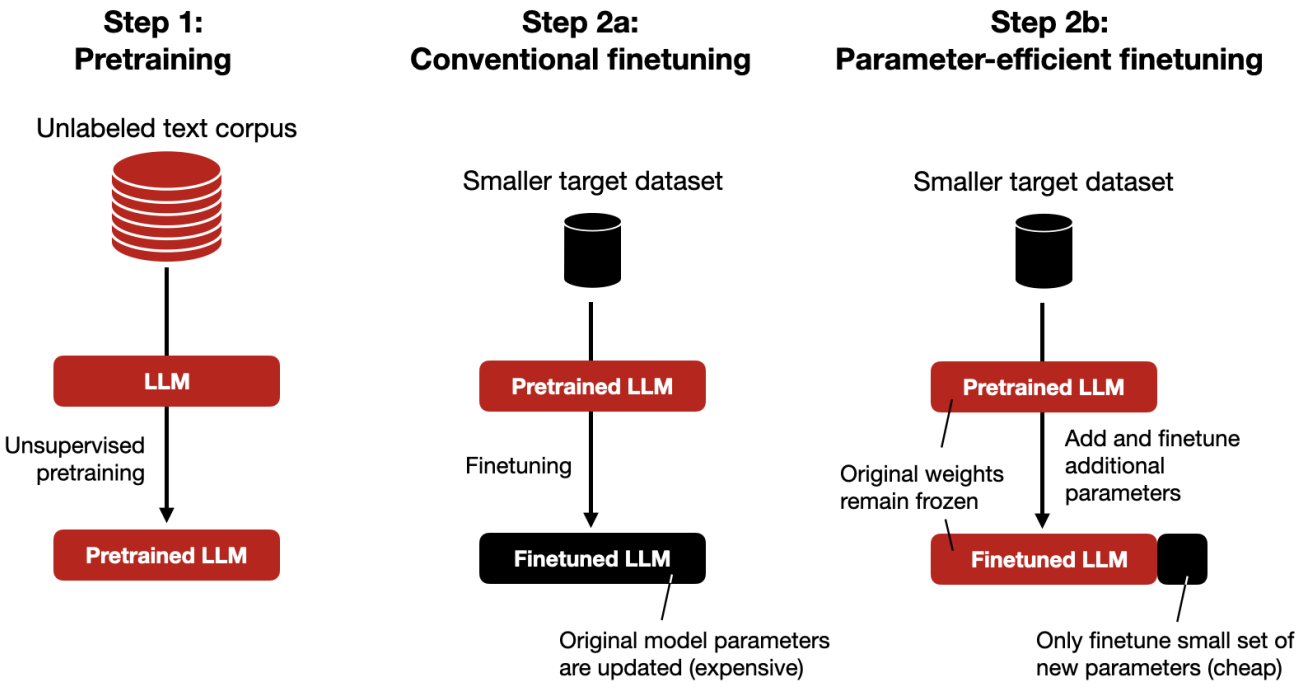
The initial training of an LLM on vast, diverse datasets to learn language patterns.

Example: GPT-4 learns general language understanding from the internet, books, etc.

Finetuning

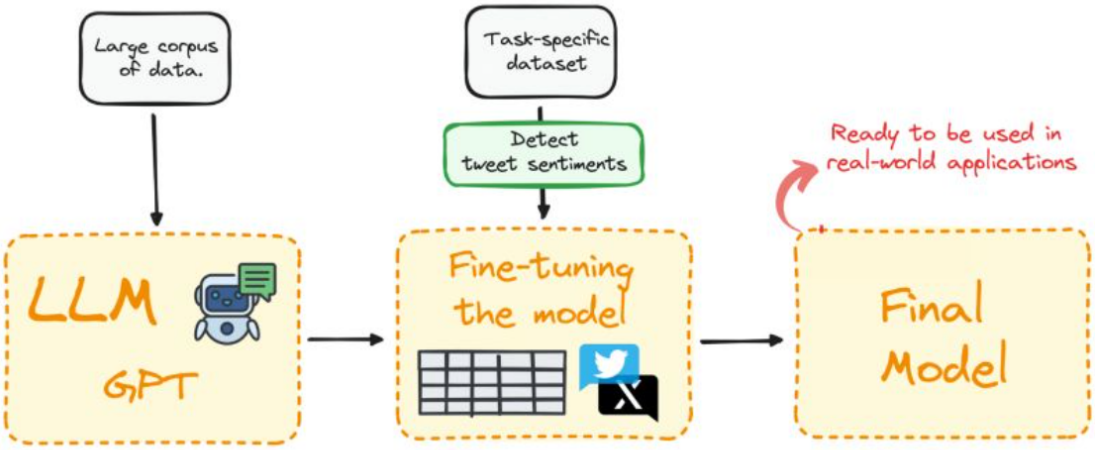
Additional training on specific datasets to adapt the model for particular tasks or styles.

Example: A company finetunes GPT-4 on their customer service conversations to create a specialized customer support assistant.



Python example of finetuning using Hugging Face:

```
from transformers import AutoModelForCausalLM, AutoTokenizer, Trainer, TrainingArguments
import torch

# Load pre-trained model
model_name = "gpt2"  # A smaller model for demonstration
model = AutoModelForCausalLM.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Prepare your training data
train_dataset = YourCustomDataset()  # This would be your custom dataset class

# Define training arguments
training_args = TrainingArguments(
    output_dir="./finetuned-model",
    num_train_epochs=3,
    per_device_train_batch_size=4,
    save_steps=10_000,
    save_total_limit=2,
)

# Create trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
)

# Start finetuning
trainer.train()

# Save the finetuned model
model.save_pretrained("./my_finetuned_model")
tokenizer.save_pretrained("./my_finetuned_model")
```



4. Prompting Techniques

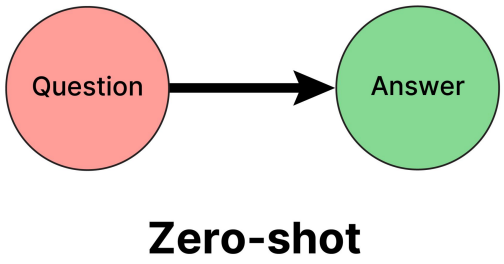
Prompt Engineering

Prompting is the art of crafting input text to get desired outputs from an LLM.

Zero-shot Prompt

No examples provided

```
Classify this review as positive or negative:
"The food was cold and the service was terrible."
```



Zero-shot

Few-shot Prompt

Including examples in the prompt

```
Classify reviews as positive or negative:

Review: "The movie was fantastic!"
Classification: Positive

Review: "Terrible experience, would not recommend."
Classification: Negative

Review: "The food was cold and the service was terrible."
Classification:
```

Chain-of-thought Prompt

Encouraging step-by-step reasoning

```
If John has 5 apples, gives 2 to Mary, and then buys 3 more, how many apples does John have?

Let's think step by step:
```

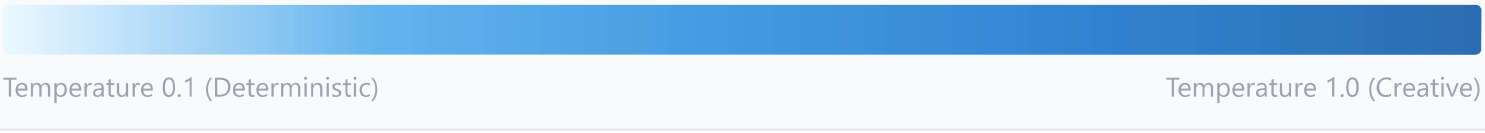
Python example with different prompt techniques:

```
import openai

openai.api_key = "your-api-key"

# Zero-shot prompting
zero_shot_response = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[
        {"role": "user", "content": "Classify this review as positive or negative: 'The
food was cold and the service was terrible.'"}
    ]
)

# Few-shot prompting
few_shot_response = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[
        {"role": "user", "content": """Classify reviews as positive or negative:

Review: "The movie was fantastic!"
Classification: Positive

Review: "Terrible experience, would not recommend."
Classification: Negative

Review: "The food was cold and the service was terrible."
Classification:"""}
    ]
)

# Chain-of-thought prompting
cot_response = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[
        {"role": "user", "content": "If John has 5 apples, gives 2 to Mary, and then buys
3 more, how many apples does John have? Let's think step by step:"}
    ]
)

print("Zero-shot:", zero_shot_response.choices[0].message.content)
print("Few-shot:", few_shot_response.choices[0].message.content)
print("Chain-of-thought:", cot_response.choices[0].message.content)
```

5. Generation Parameters

Temperature

Temperature controls randomness in output generation. Lower values (near 0) make responses more deterministic and focused, while higher values (near 1 or 2) make them more creative but potentially less coherent.



Low temperature (0.1-0.3)

Best for:

- Factual Q&A
- Coding
- Logical reasoning

Medium temperature (0.4-0.7)

Best for:

- Balanced responses
- Explanations
- General conversation

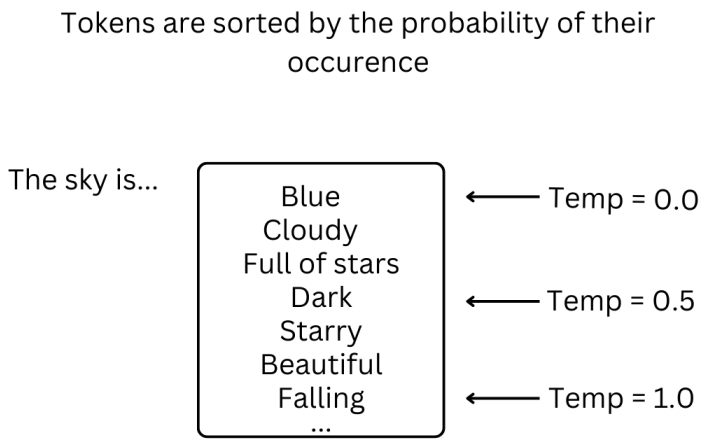
High temperature (0.8-1.2)

Best for:

- Creative writing
- Brainstorming
- Storytelling

Python example:

```
import anthropic

client = anthropic.Anthropic(api_key="your-api-key")

# Low temperature for factual content
factual_response = client.messages.create(
    model="claude-3-sonnet-20240229",
    max_tokens=150,
    temperature=0.1,
    messages=[
        {"role": "user", "content": "What are the planets in our solar system?"}
    ]
)

# High temperature for creative content
creative_response = client.messages.create(
    model="claude-3-sonnet-20240229",
    max_tokens=150,
    temperature=1.0,
    messages=[
        {"role": "user", "content": "Write a short poem about space exploration"}
    ]
)

print("Factual (T=0.1):", factual_response.content)
print("Creative (T=1.0):", creative_response.content)
```



6. Sampling Methods

Top-k Sampling

Top-k sampling restricts token selection to only the k most likely next tokens.

> **Real-world example:**
>
> If k=50, the model will only consider the 50 most probable next words when generating each word in a response.

Python example:

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Load model and tokenizer
model = GPT2LMHeadModel.from_pretrained("gpt2")
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# Input text
input_text = "The future of artificial intelligence will"
input_ids = tokenizer.encode(input_text, return_tensors="pt")

# Generate with top-k sampling
output = model.generate(
    input_ids,
    max_length=50,
    do_sample=True,
    top_k=50,
    temperature=0.7,
    pad_token_id=tokenizer.eos_token_id
)

print(tokenizer.decode(output[0], skip_special_tokens=True))
```

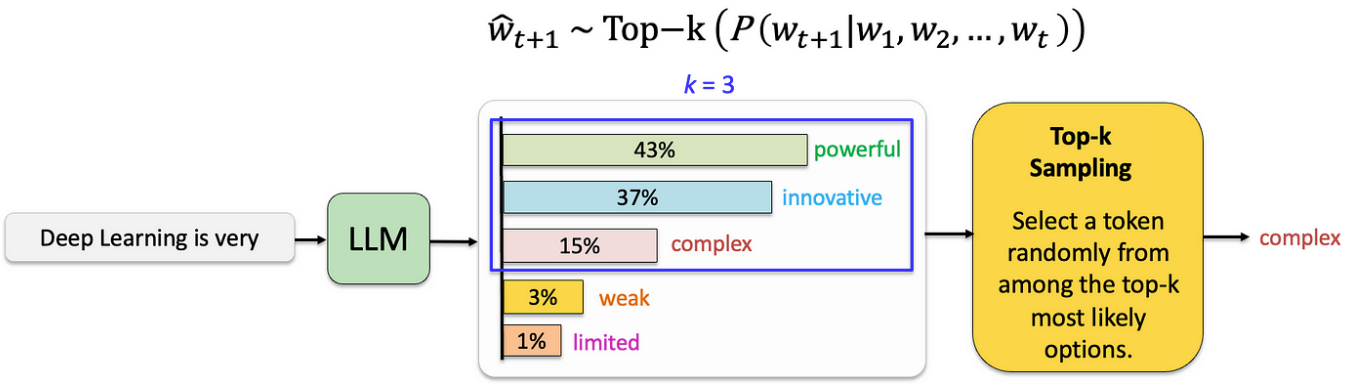
Top-p (Nucleus) Sampling

Top-p sampling selects from the smallest set of tokens whose cumulative probability exceeds probability p.

> **Real-world example:**
>
> If p=0.9, the model will consider only those tokens whose combined probability sums to 90%, discarding less likely options.

Python example:

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Load model and tokenizer
model = GPT2LMHeadModel.from_pretrained("gpt2")
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# Input text
input_text = "In the year 2050, the world"
input_ids = tokenizer.encode(input_text, return_tensors="pt")

# Generate with top-p (nucleus) sampling
output = model.generate(
    input_ids,
    max_length=50,
    do_sample=True,
    top_p=0.92,  # Using nucleus sampling
    temperature=0.7,
    pad_token_id=tokenizer.eos_token_id
)

print(tokenizer.decode(output[0], skip_special_tokens=True))
```



7. Popular Models

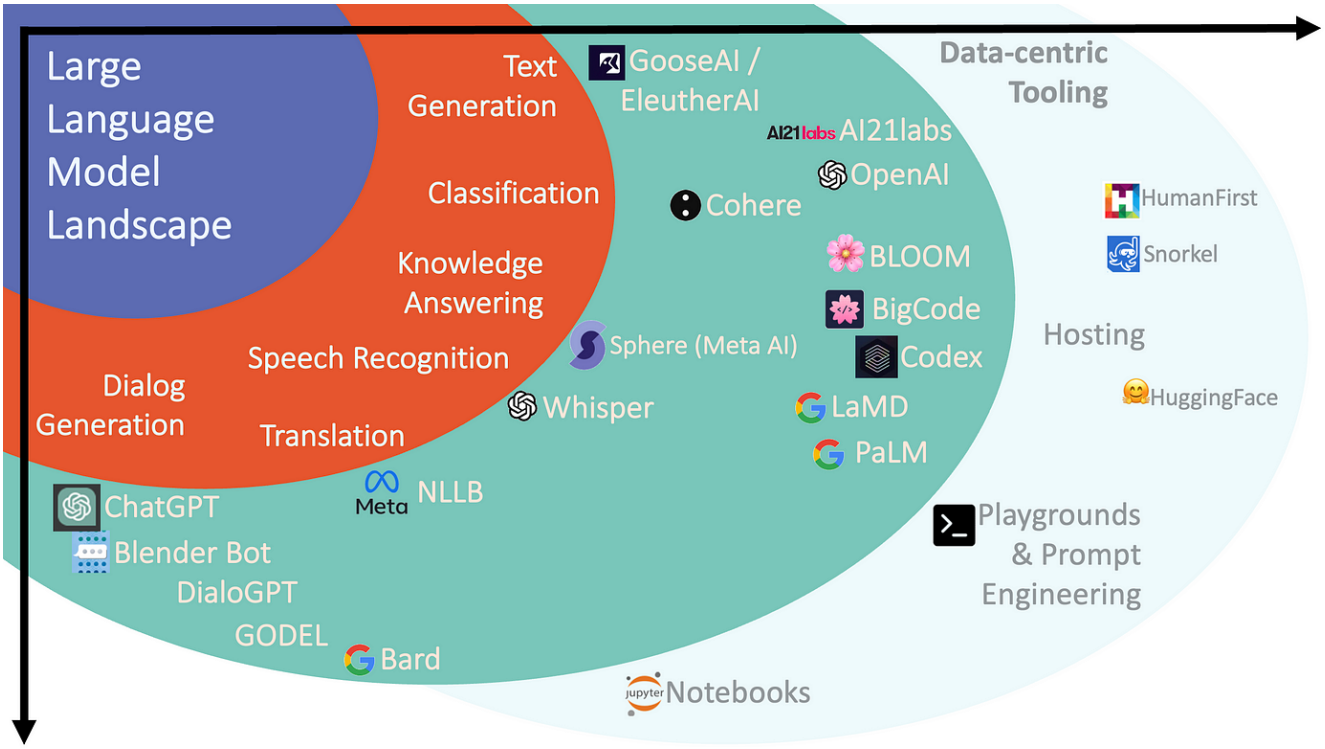
Model Family	Key Models	Notable Features
GPT (OpenAI)	GPT-3, GPT-3.5, GPT-4	• GPT-3: Released in 2020, 175B parameters • GPT-3.5: Powers earlier versions of ChatGPT • GPT-4: Released in 2023, multimodal capabilities
Claude (Anthropic)	Claude 1, Claude 2, Claude 3 (Opus, Sonnet, Haiku)	• Claude 1: Anthropic's first major LLM • Claude 2: Improved reasoning and longer context • Claude 3: Released in 2024, varying capabilities
Open Source	LLaMA, Mistral, Gemini	• LLaMA: Meta's open-weight models (7B to 70B) • Mistral: Efficient models with strong performance • Gemini: Google's multimodal LLM family

Python example using GPT-4:

```
import openai

openai.api_key = "your-api-key"

response = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Write a function in Python to find prime numbers up
to n."}
    ]
)

print(response.choices[0].message.content)
```

Python example using Claude:

```
import anthropic

client = anthropic.Anthropic(api_key="your-api-key")

response = client.messages.create(
    model="claude-3-sonnet-20240229",
    max_tokens=1000,
    messages=[
        {"role": "user", "content": "Explain how transformers work in deep learning"}
    ]
)

print(response.content)
```



8. Transformer Architecture

Overview

Transformers revolutionized NLP with their attention mechanism, allowing models to focus on relevant parts of input regardless of position.

Encoder-only (BERT)

Good for understanding text

Examples: BERT, RoBERTa

Use cases: Classification, named entity recognition

Decoder-only (GPT)

Good for generating text

Examples: GPT-3, GPT-4, Claude

Use cases: Text generation, creative writing

Encoder-decoder (T5)

Good for translation, summarization

Examples: T5, BART, Pegasus

Use cases: Translation, summarization, question-answering

Attention Mechanism

The **attention mechanism** allows the model to weight the importance of different words in relation to each other.

Real-world example:

In the sentence "The bank by the river was eroding," attention helps the model recognize that "bank" refers to a riverbank (not a financial institution) by paying attention to the contextual word "river."

Practical Transformer Example:

```
from transformers import pipeline

# Load a text generation pipeline
generator = pipeline('text-generation', model='gpt2')

# Generate text
prompt = "In the future, artificial intelligence will"
outputs = generator(prompt, max_length=50, num_return_sequences=3)

# Print generated texts
for i, output in enumerate(outputs):
    print(f"Output {i+1}:")
    print(output['generated_text'])
    print("-" * 50)
```

9. End-to-End LLM Application

Building a simple chatbot

```
import gradio as gr
import openai

# Configure your API key
openai.api_key = "your-api-key"  # Replace with your actual API key

# Maintain conversation history
messages = [{"role": "system", "content": "You are a helpful assistant."}]

def chatbot(input_text):
    # Add user message to history
    messages.append({"role": "user", "content": input_text})

    # Get response from LLM
    response = openai.ChatCompletion.create(
        model="gpt-4",
        messages=messages,
        temperature=0.7,
        max_tokens=150
    )

    # Extract assistant's response
    assistant_response = response.choices[0].message.content

    # Add assistant response to history
    messages.append({"role": "assistant", "content": assistant_response})

    return assistant_response

# Create a simple web interface
demo = gr.Interface(
    fn=chatbot,
    inputs="text",
    outputs="text",
    title="Simple LLM Chatbot",
    description="Ask me anything!"
)

# Launch the interface
demo.launch()
```

10. Resources

Documentation & Tutorials

- Hugging Face Transformers
- OpenAI API Documentation
- Anthropic Claude Documentation
- LangChain Documentation

Research Papers

- "Attention Is All You Need" Paper - The original Transformer paper
- "Language Models are Few-Shot Learners" - GPT-3 paper

Books & Courses

- "Natural Language Processing with Transformers" - Lewis Tunstall, Leandro von Werra, and Thomas Wolf
- ChatGPT Prompt Engineering for Developers - deeplearning.ai