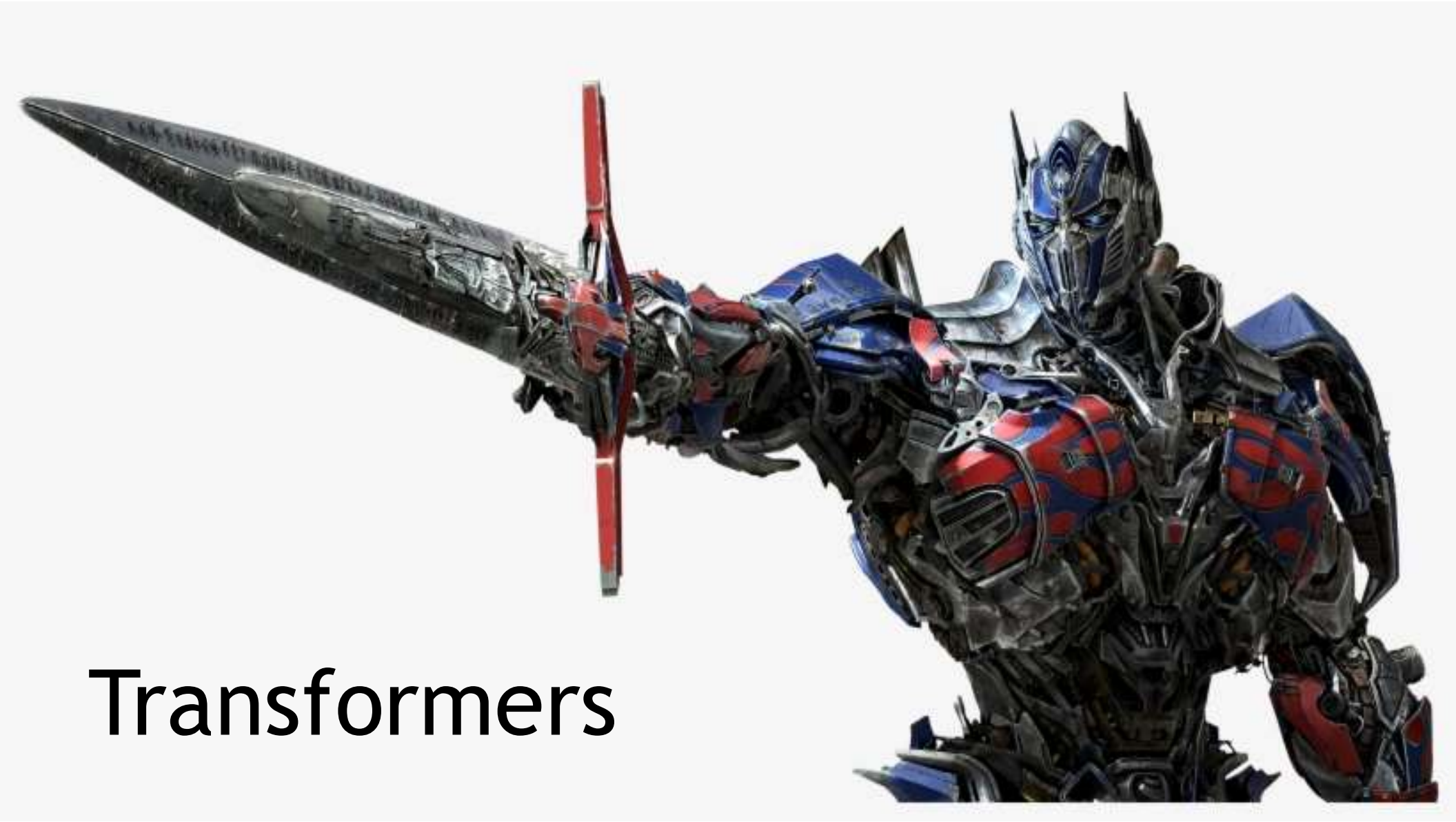
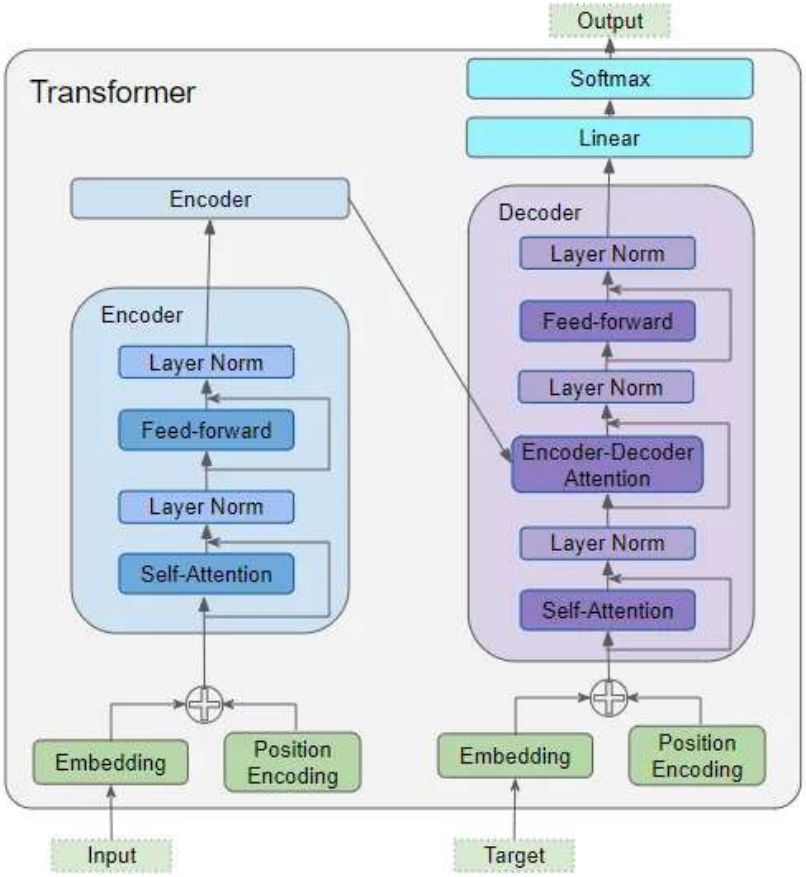
Generative AI


Transformers

Transformers

- The **transformer architecture** is a powerful framework that underpins large language models (LLMs) like GPT, BERT, and others.
- It revolutionized natural language processing (NLP) by allowing parallelization, making it faster and more scalable compared to earlier models such as RNNs and LSTMs.
- It was popularized by a technical paper published in 2017
  - https://arxiv.org/abs/1706.03762
- Transformers contain encoders and decoders

Transformer Architecture



- The Encoder contains the all-important Self-attention layer that computes the relationship between different words in the sequence, as well as a Feed-forward layer.
- The Decoder contains the Self-attention layer and the Feed-forward layer, as well as a second Encoder-Decoder attention layer.
- Each Encoder and Decoder has its own set of weights.

Attention

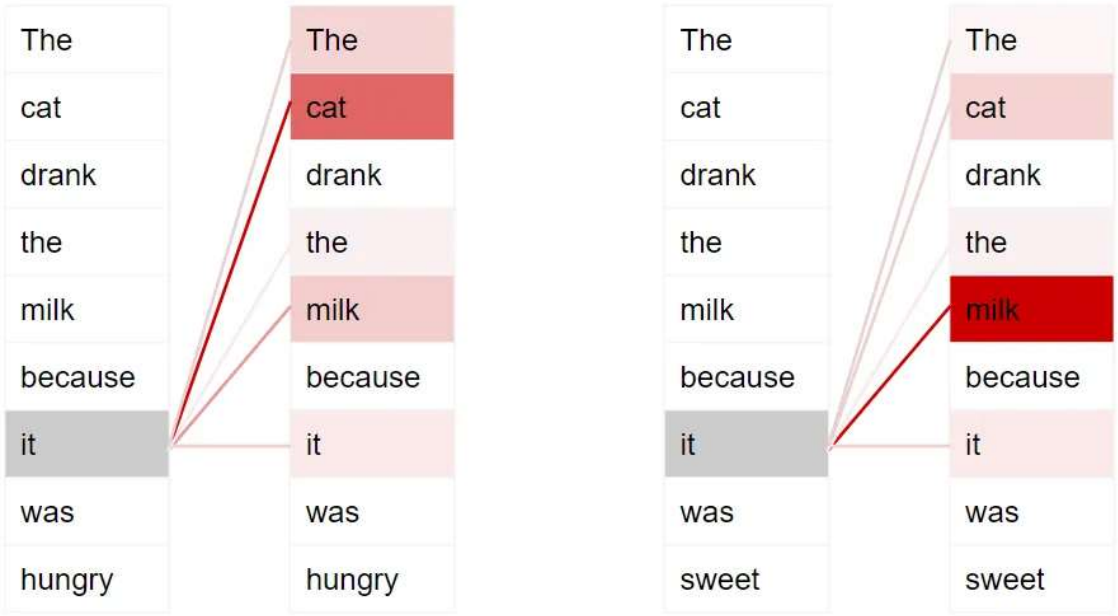
- The key to the Transformer's ground-breaking performance is its use of Attention.
- While processing a word, Attention enables the model to focus on other words in the input that are closely related to that word.
- The Transformer architecture uses self-attention by relating every word in the input sequence to every other word.
- Consider two sentences:
  - The *cat* drank the milk because **it** was hungry.
  - The cat drank the *milk* because **it** was sweet.

it => cat

it => milk

- When the model processes the word 'it', self-attention gives the model more information about its meaning so that it can associate 'it' with the correct word.

Attention

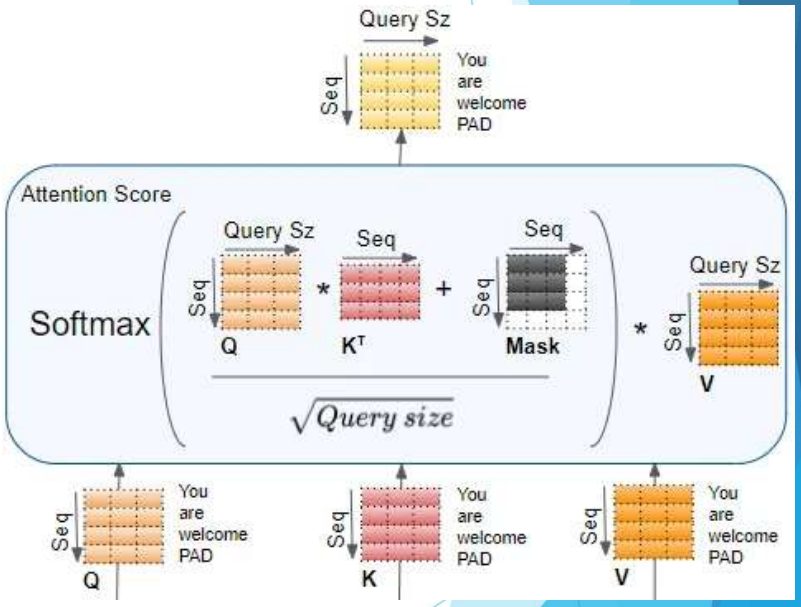

The
cat
drank
the
milk
because
it
was
hungry
The
cat
drank
the
milk
because
it
was
hungry
The
cat
drank
the
milk
because
it
was
sweet
The
cat
drank
the
milk
because
it
was
sweet

Attention

- In the Transformer, Attention is used in three places:
  - Self-attention in the Encoder — the input sequence pays attention to itself
  - Self-attention in the Decoder — the target sequence pays attention to itself
  - Encoder-Decoder-attention in the Decoder — the target sequence pays attention to the input sequence

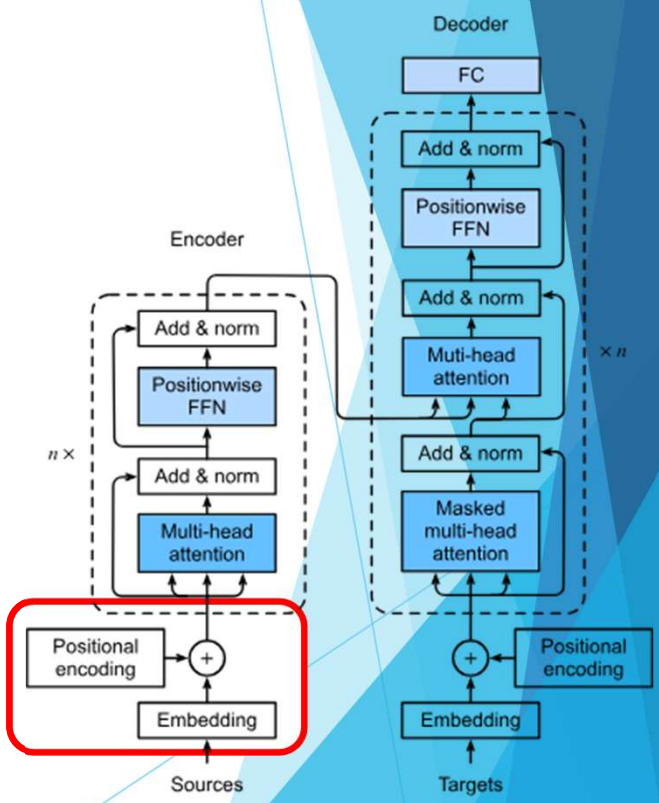
Similarity Score

- In transformers, the **similarity score** plays a key role in the **self-attention mechanism**, where it helps the model decide how much each word (or token) in a sequence should focus on other words. The similarity score measures how related or connected two words are based on their meanings.
  - Similarity scores measure how closely related two words are in the context of a sentence.
  - These scores are calculated by comparing the queries and keys of words.
  - The scores help decide attention weights, determining which words the model should focus on.
  - Higher similarity scores mean more attention, leading to better contextual understanding of the sentence.



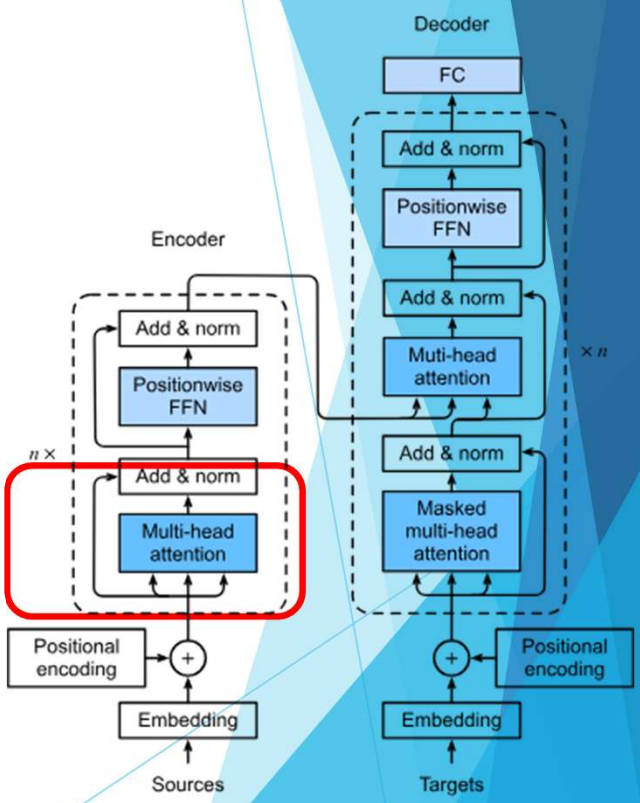
Functionality of the Encoder

- ***Input Embedding and Positional Encoding***
- Input Embedding: The input tokens (words or sub-words) are converted into dense vectors (embeddings) that capture their meanings.
- Positional Encoding: Since the transformer doesn't have an inherent sense of word order, positional encodings are added to the embeddings to introduce sequence order.
- Example: For the sentence "The cat sat," embeddings for each token are generated:
  - Embeddings:
    "The" → [0.2, 0.4, -0.1, ...]
    "cat" → [0.3, 0.6, -0.2, ...]
  - Positional encoding adds information about token positions: 1st, 2nd, 3rd.



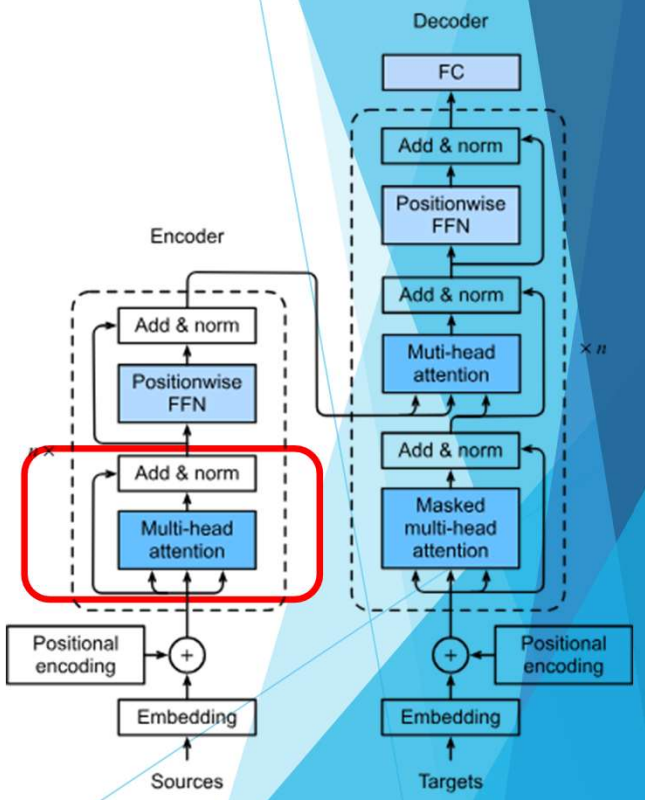
Functionality of the Encoder

- ***Self-Attention Mechanism***
- Self-attention: This mechanism allows each token to focus on other tokens in the input sequence, determining their relative importance.
- How it works: The model calculates attention scores for each token pair using the queries, keys, and values derived from the embeddings. Higher attention scores mean stronger relationships between tokens.
- Example: While processing "cat," the model might focus more on "sat" (action) than "The" (subject), depending on context.



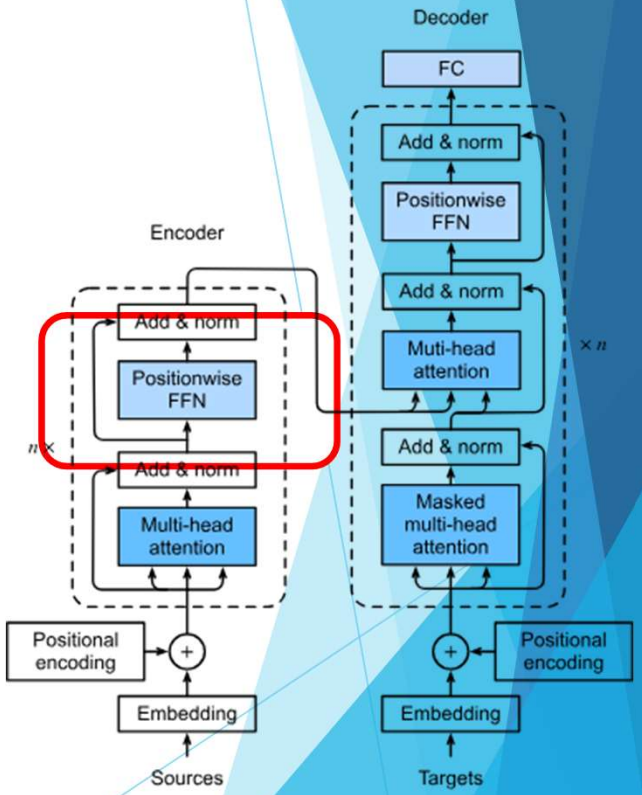
Functionality of the Encoder

- ***Multi-Head Attention***
- Instead of a single attention calculation, multiple attention heads are used in parallel to capture different types of relationships in the data (e.g., grammatical or semantic connections).
- Each head performs its own attention and produces outputs, which are then combined.
- Example: One head might focus on the relationship between "cat" and "sat" (subject-verb), while another head focuses on the relationship between "cat" and "The" (subject-article).



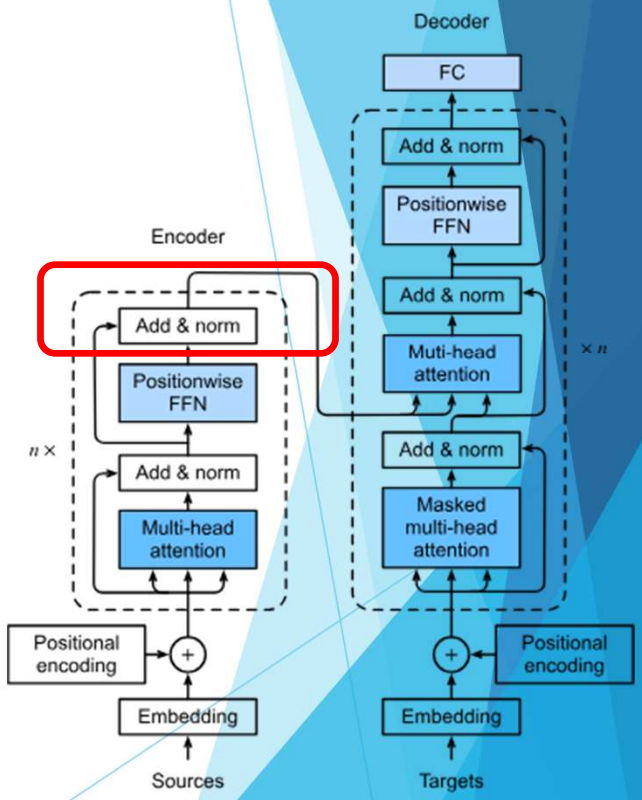
Functionality of the Encoder

- ***Feed-Forward Neural Network (FFNN)***
- After the multi-head attention step, each token's representation is passed through a feed-forward neural network. This typically consists of two linear layers with a ReLU activation.
- This step refines each token's representation, making it more abstract and context-aware.
- Example: After attending to other tokens, the representation for "cat" is transformed by the FFNN into a more informative vector based on both the word itself and its context.



Functionality of the Encoder

- ***Residual Connections and Layer Normalization***
- Residual Connection: The original input to each layer is added to the output of the attention and feed-forward layers (a form of shortcut), helping to prevent information loss and making it easier to train deep networks.
- Layer Normalization: After the addition, layer normalization is applied to stabilize and speed up training.
- Example: If the attention output for "cat" is [0.5, -0.3, 0.2], and the original embedding was [0.3, 0.6, -0.2], the residual connection adds them and normalizes the result to ensure a balanced and informative representation.

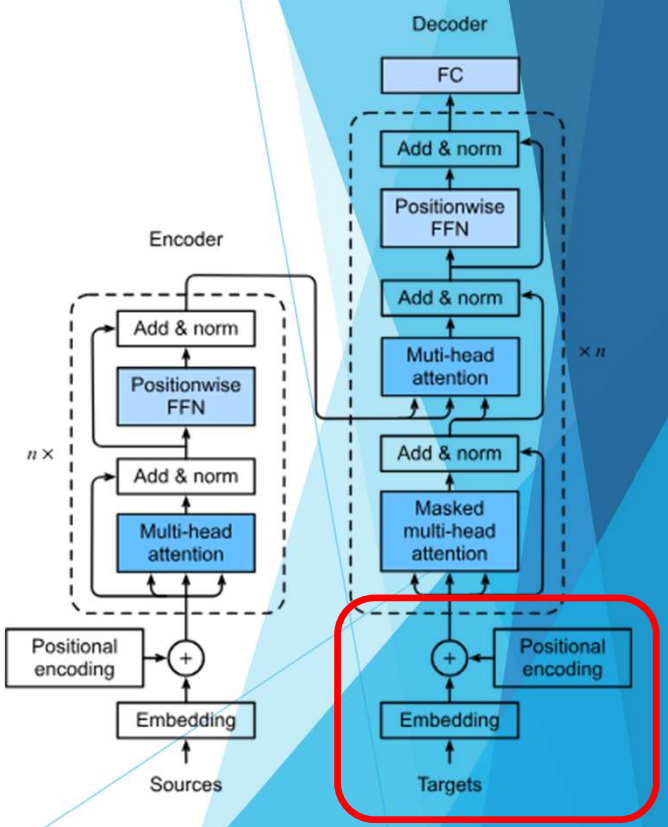


Functionality of Decoder

- ***Input Embedding and Positional Encoding***
- Input Embedding: Similar to the encoder, the input tokens (from the target sequence) are first converted into embeddings, capturing their meaning.
- Positional Encoding: Positional encodings are added to the embeddings to preserve information about the order of tokens in the target sequence.
- Example: For generating a sentence like "The dog barked," token embeddings are created for each word:

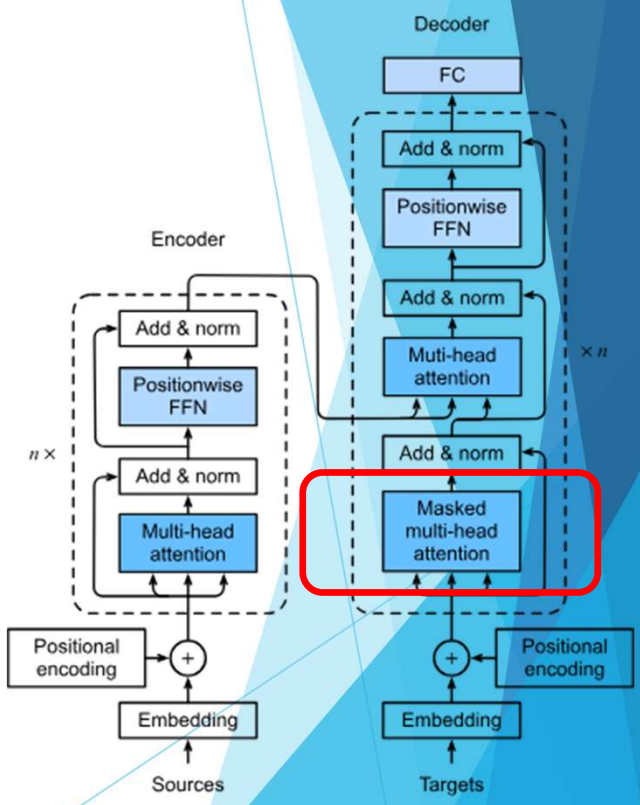
  "The" → [0.3, -0.2, 0.5, ...],

  "dog" → [0.4, 0.1, -0.3, ...]
- Positional encodings are added to these embeddings.



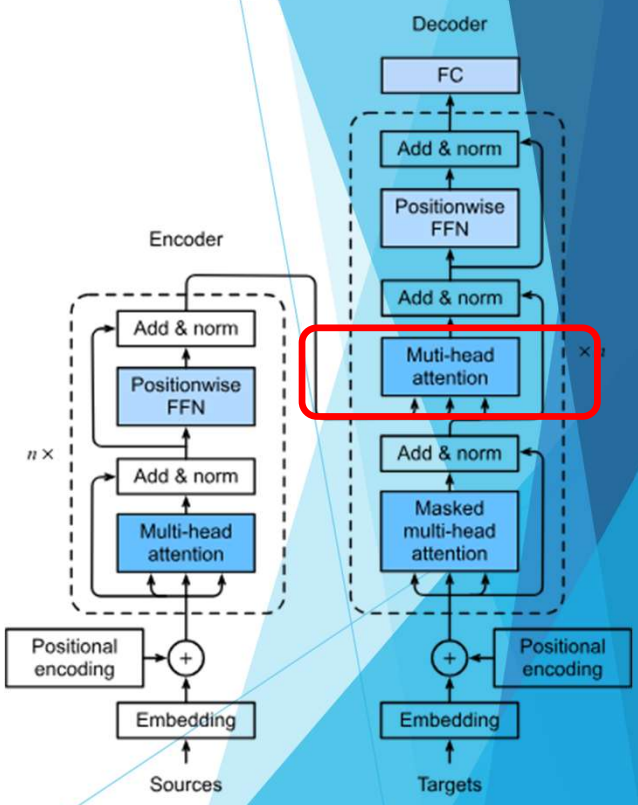
Functionality of Decoder

- ***Masked Multi-Head Self-Attention***
- Masked Attention: The decoder operates in an auto-regressive manner, meaning it can only attend to past tokens in the output sequence, not future ones. A mask is applied to prevent attention to future tokens.
- Multi-Head Attention: Multiple attention heads are used to capture different aspects of the relationships between tokens in the partially generated sequence.
- Example: While generating the word "barked," the model can attend to "The" and "dog," but not to future words like "loudly," ensuring it generates the output step by step.



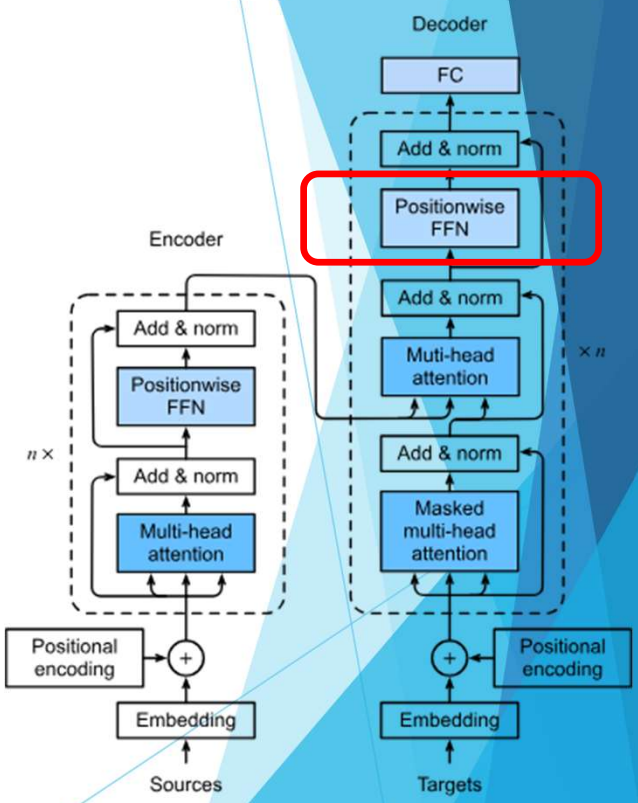
Functionality of Decoder

- **Encoder-Decoder Multi-Head Attention**
- After processing the target tokens with self-attention, the decoder attends to the output of the encoder. This allows the decoder to focus on the source sequence (input) while generating the target sequence (output).
- Multi-Head Attention: Multiple attention heads attend to different parts of the source sequence, using the encoded representations.
- Example: In a translation task, when generating the target word "barked," the model can attend to the source word "chien" (French for "dog") to ensure accurate translation.



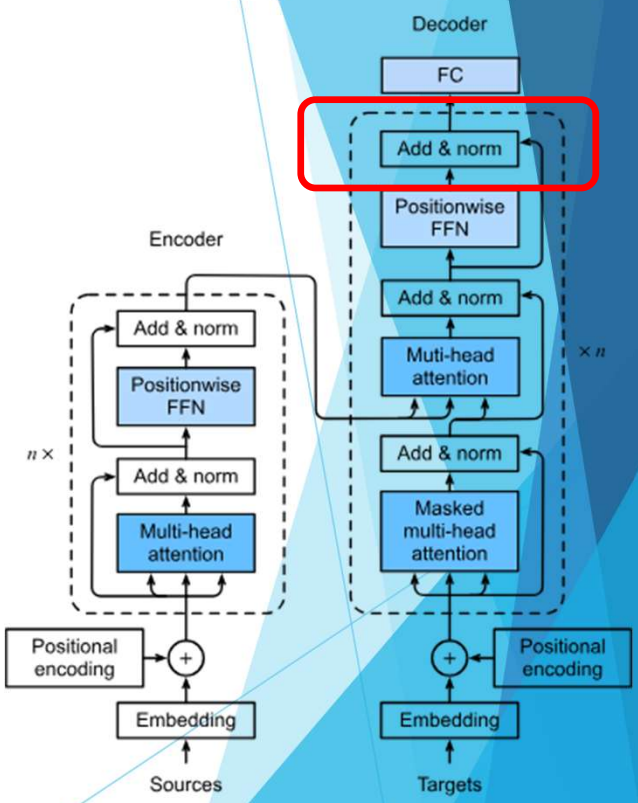
Functionality of Decoder

- **Feed-Forward Neural Network (FFNN)**
- After the attention steps, the output of the attention mechanism is passed through a feed-forward neural network, consisting of two linear layers with a ReLU activation in between.
- This network further refines the representation of each token.
- Example: The token "barked" might have an updated representation after attending to both the previous target tokens ("The dog") and the source sequence (from the encoder).



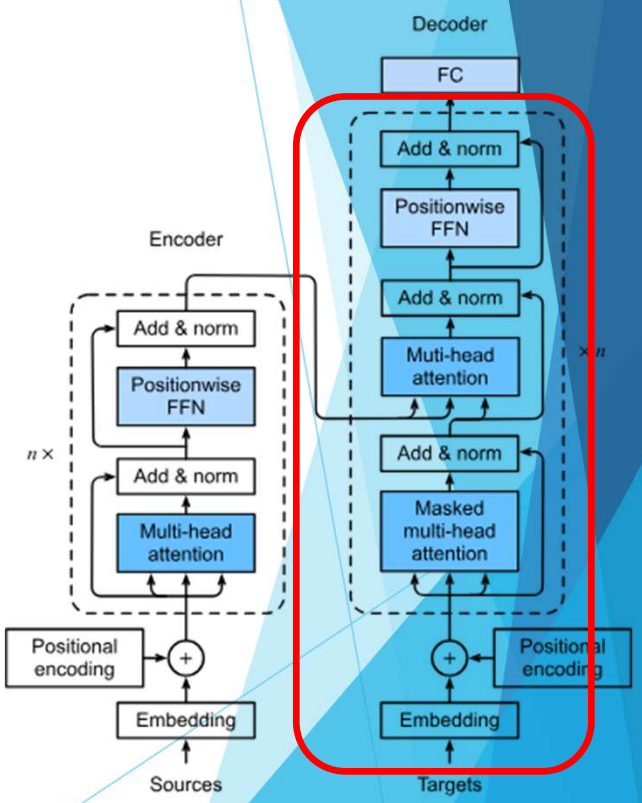
Functionality of Decoder

- **Residual Connections and Layer Normalization**
- Residual Connection: The original input to each sublayer is added back to the output of the sublayer, helping to stabilize and improve the flow of information. This applies to both attention and FFNN sublayers.
- Layer Normalization: After adding the residual connection, layer normalization is applied to ensure stable and efficient training.
- Example: The output for "barked" is normalized after each attention and feed-forward step, making the model more robust during training.



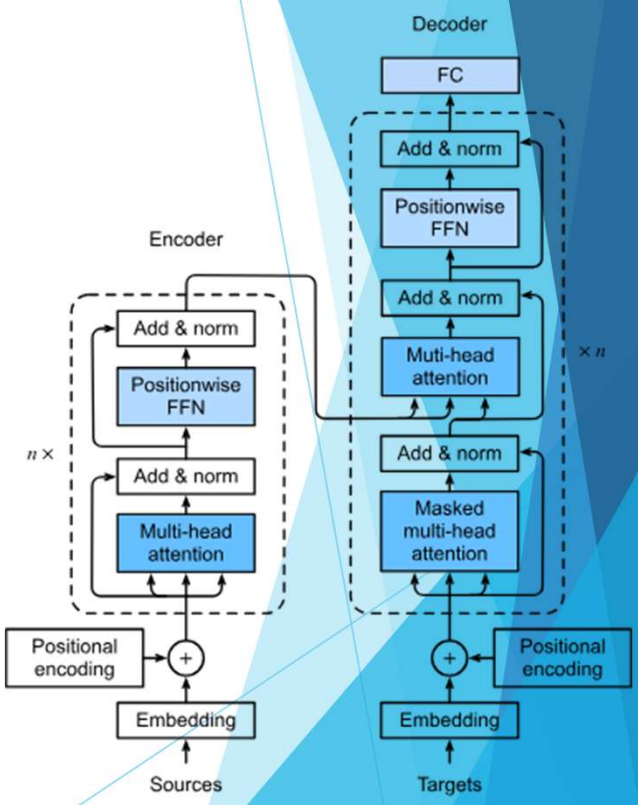
Functionality of Decoder

- **Stacking Decoder Layers**
- The decoder contains multiple layers (typically 6, 12, or more), and each layer has the same architecture (self-attention, encoder-decoder attention, FFNN). Each layer progressively refines the representation of the tokens in the target sequence.
- Example: As the token "barked" passes through several decoder layers, its representation is continually refined, becoming more contextually appropriate based on both the target and source sequences.



Functionality of Decoder

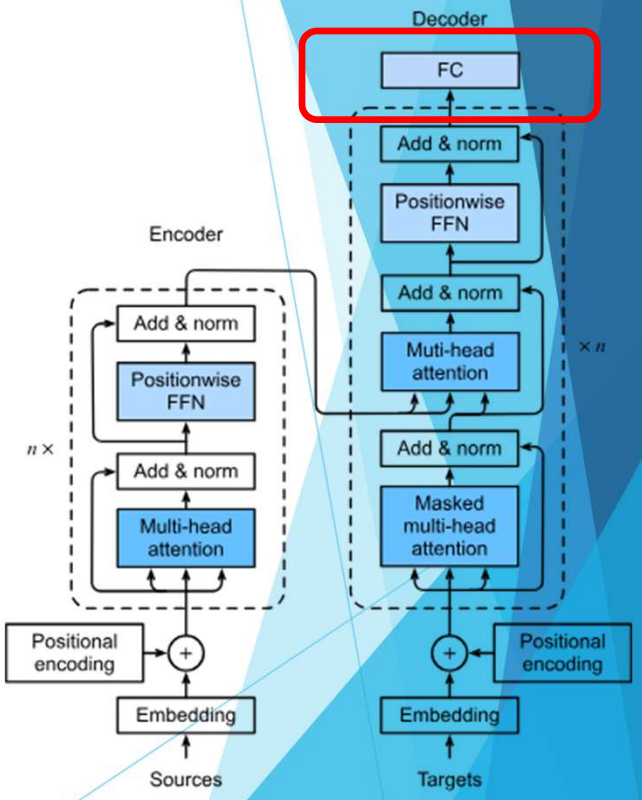
- **Masked Language Modeling (Next Token Prediction)**
- In text generation tasks, the decoder operates autoregressively by predicting the next token in the sequence based on the previous tokens and the encoder’s output. It uses the final decoder layer's output to predict the next token.
- Example: After generating "The dog barked," the decoder predicts the next token (e.g., "loudly") based on both the already generated tokens and the input sentence from the encoder.



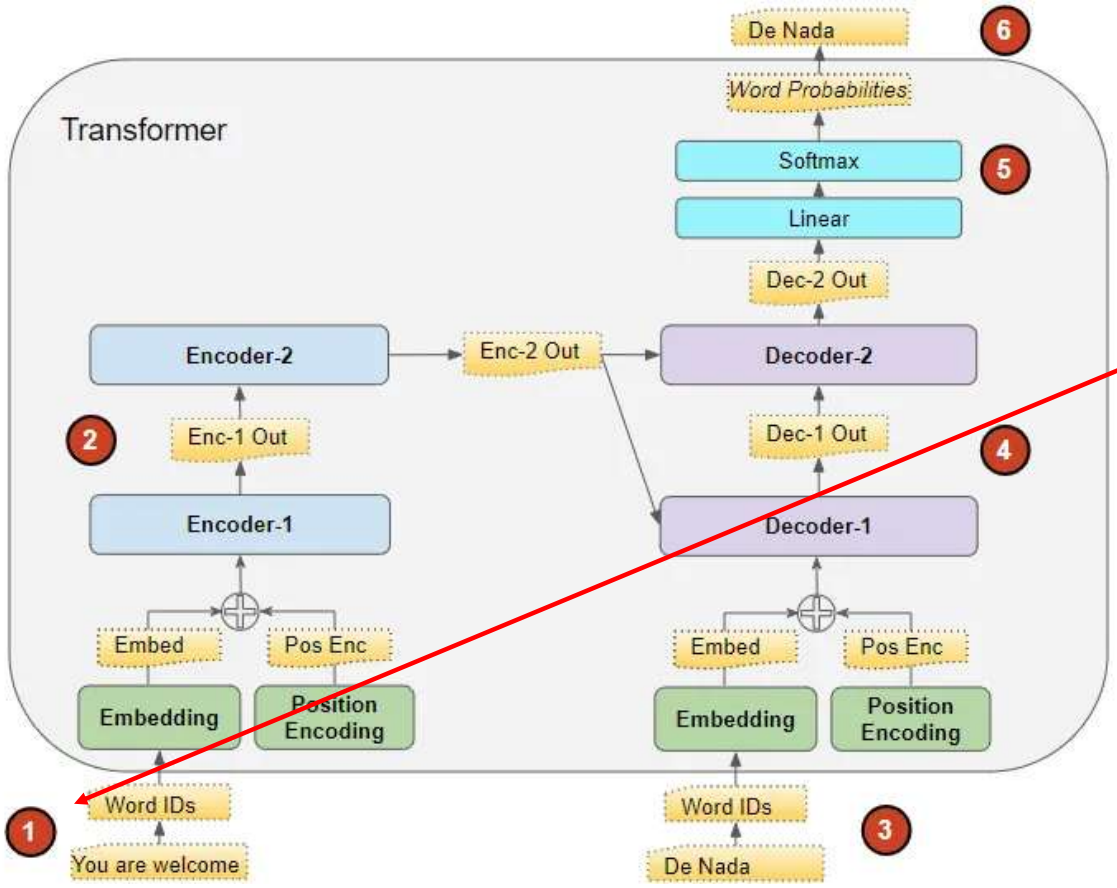
Functionality of Decoder

- Output Layer and Softmax
- The final step involves passing the decoder output through a linear layer followed by a softmax function to convert the token representations into a probability distribution over the vocabulary. The token with the highest probability is selected as the next word.
- Example: The softmax output might give probabilities like ["barked": 0.8, "meowed": 0.1, "ran": 0.05], and the model selects "barked" as the next token.

The decoder processes the target sequence step-by-step, using both the previous target tokens and the encoder output to generate context-aware tokens.

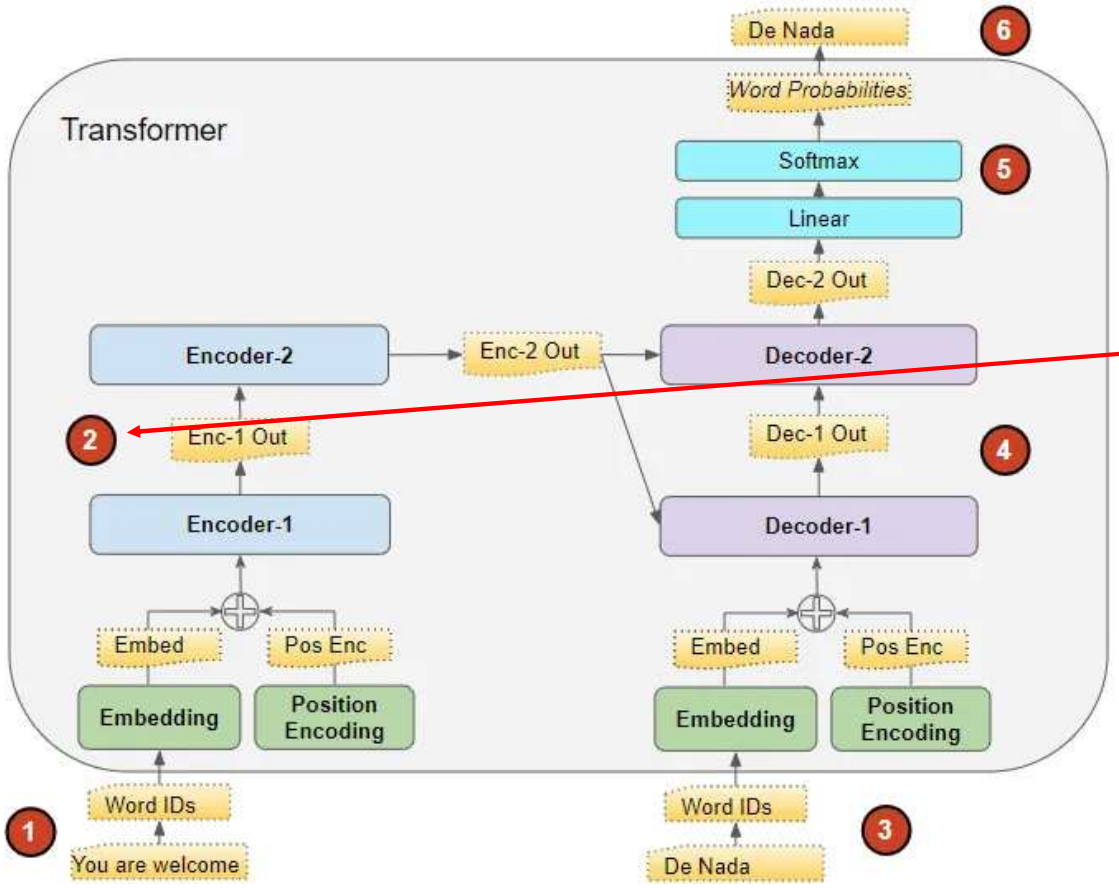


Working: Training



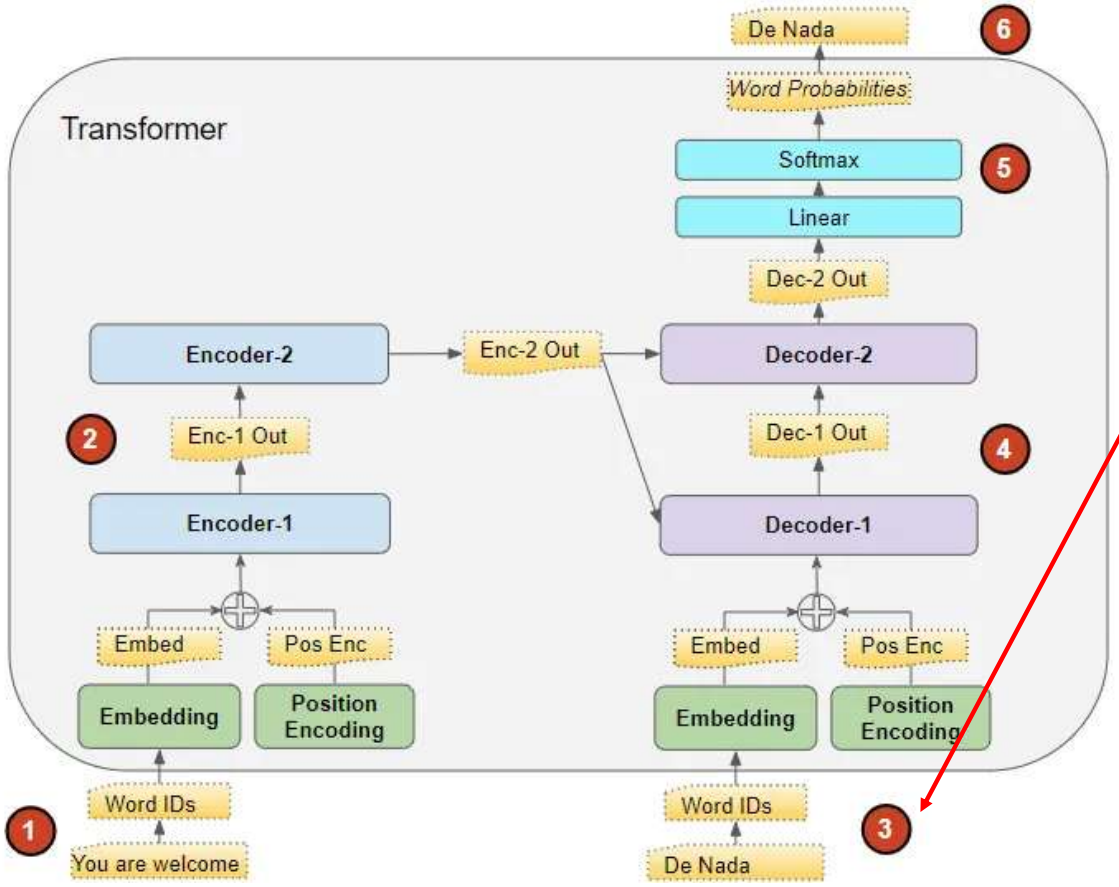
The input sequence is converted into Embeddings (with Position Encoding) and fed to the Encoder.

Working: Training



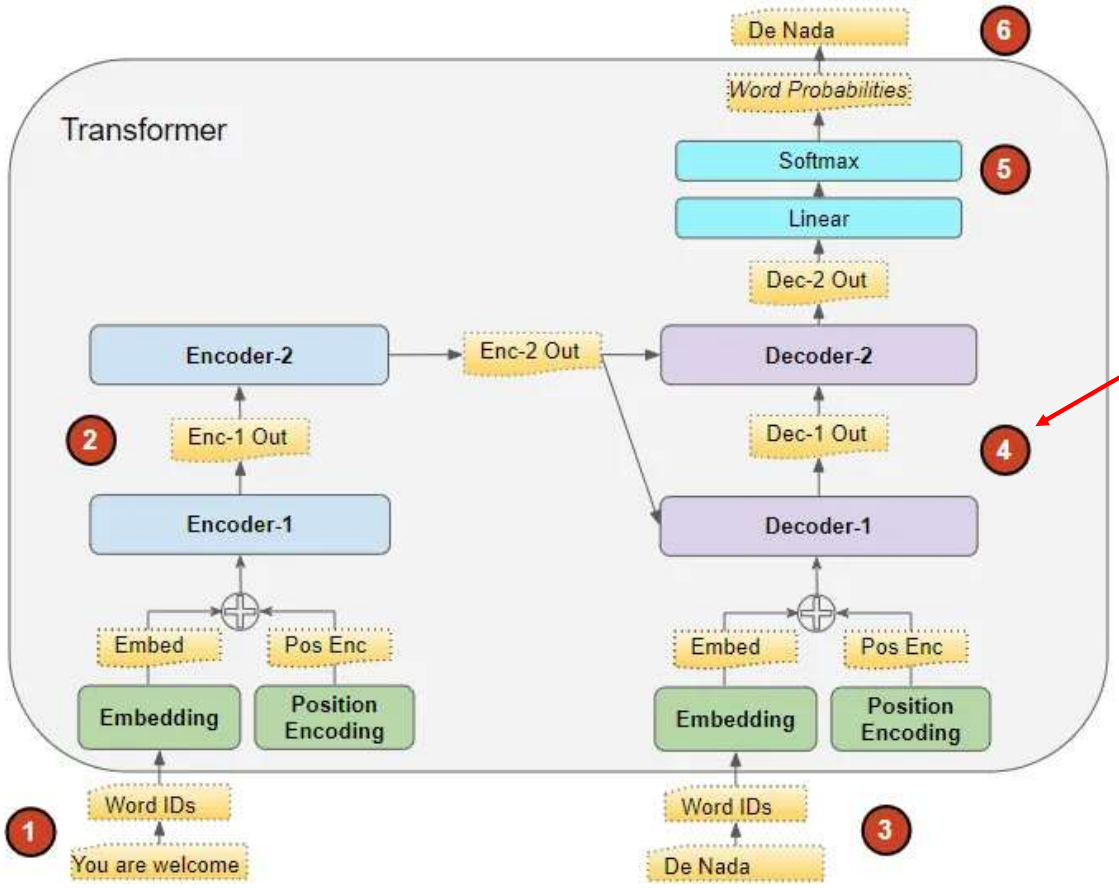
The stack of Encoders processes this and produces an encoded representation of the input sequence.

Working: Training



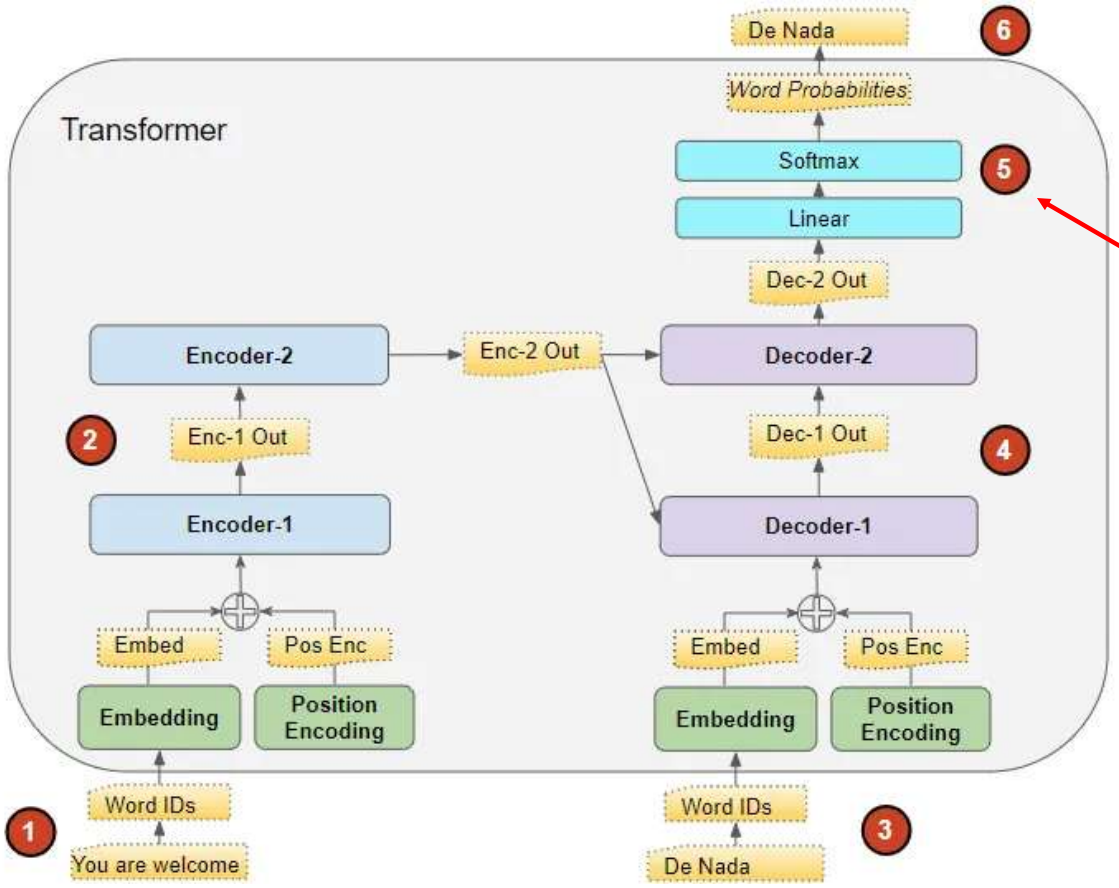
The target sequence is prepended with a start-of-sentence token, converted into Embeddings (with Position Encoding), and fed to the Decoder.

Working: Training



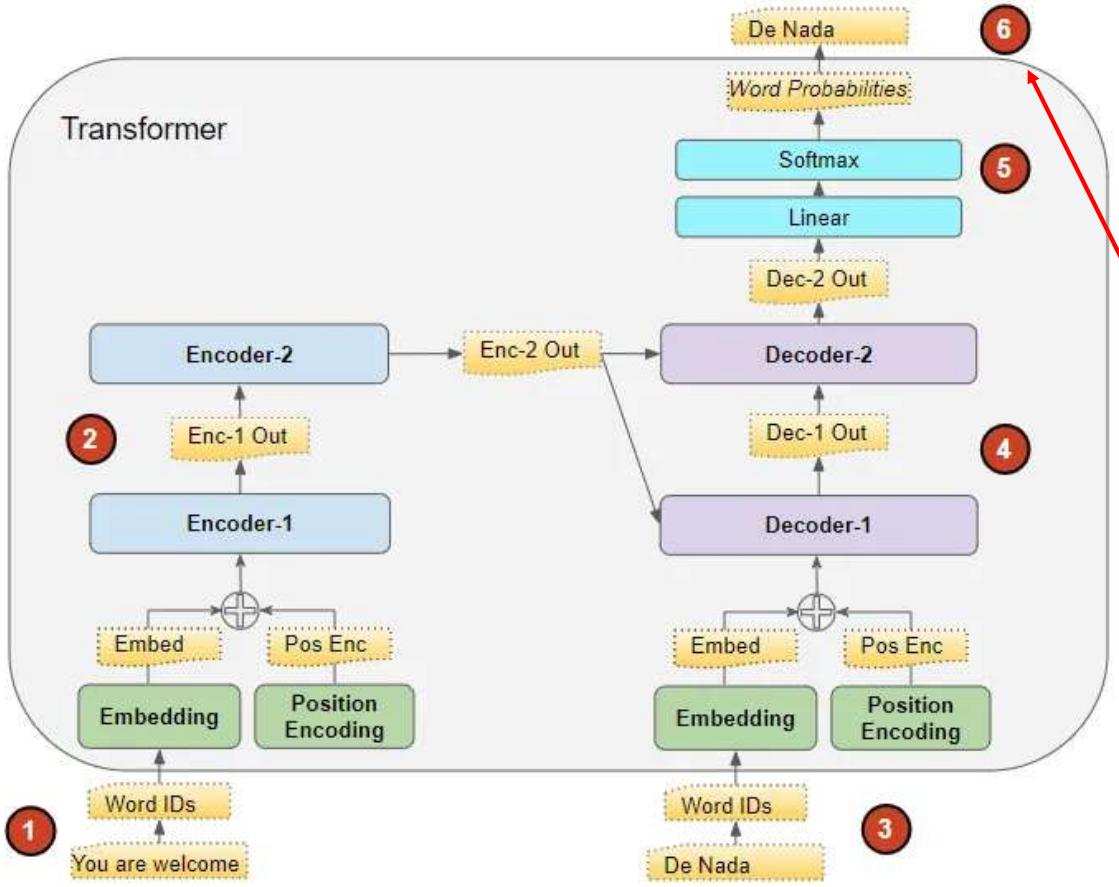
The stack of Decoders processes this along with the Encoder stack's encoded representation to produce an encoded representation of the target sequence.

Working: Training



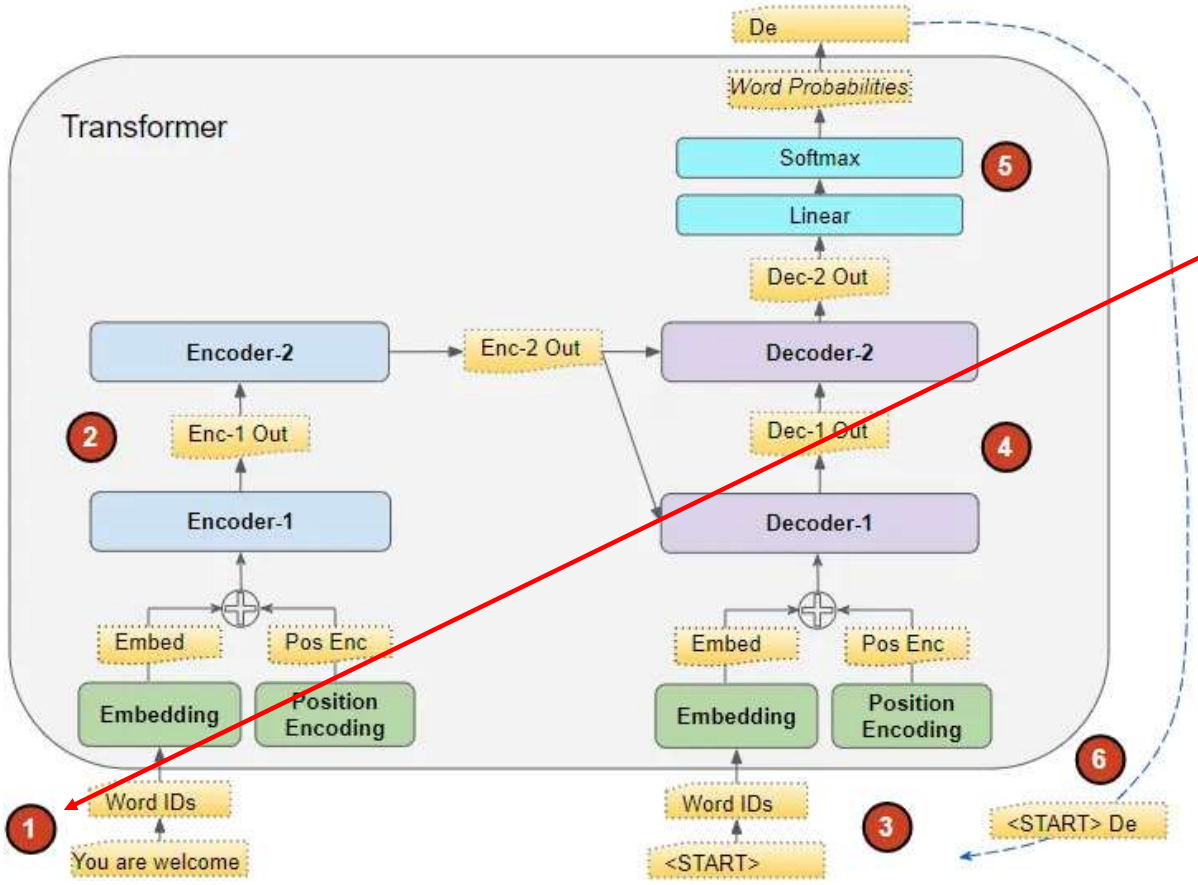
The Output layer converts it into word probabilities and the final output sequence.

Working: Training



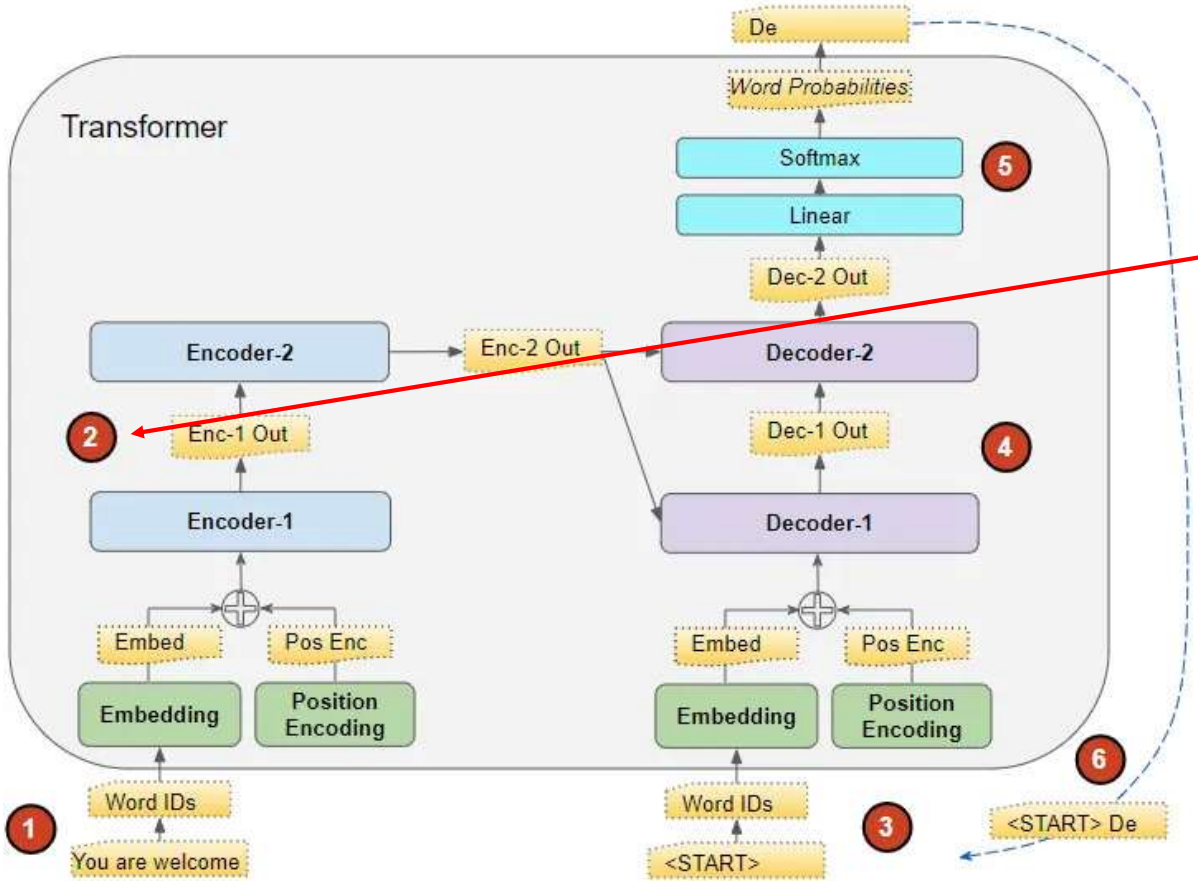
The Transformer's Loss function compares this output sequence with the target sequence from the training data. This loss is used to generate gradients to train the Transformer during back-propagation.

Working: Inference



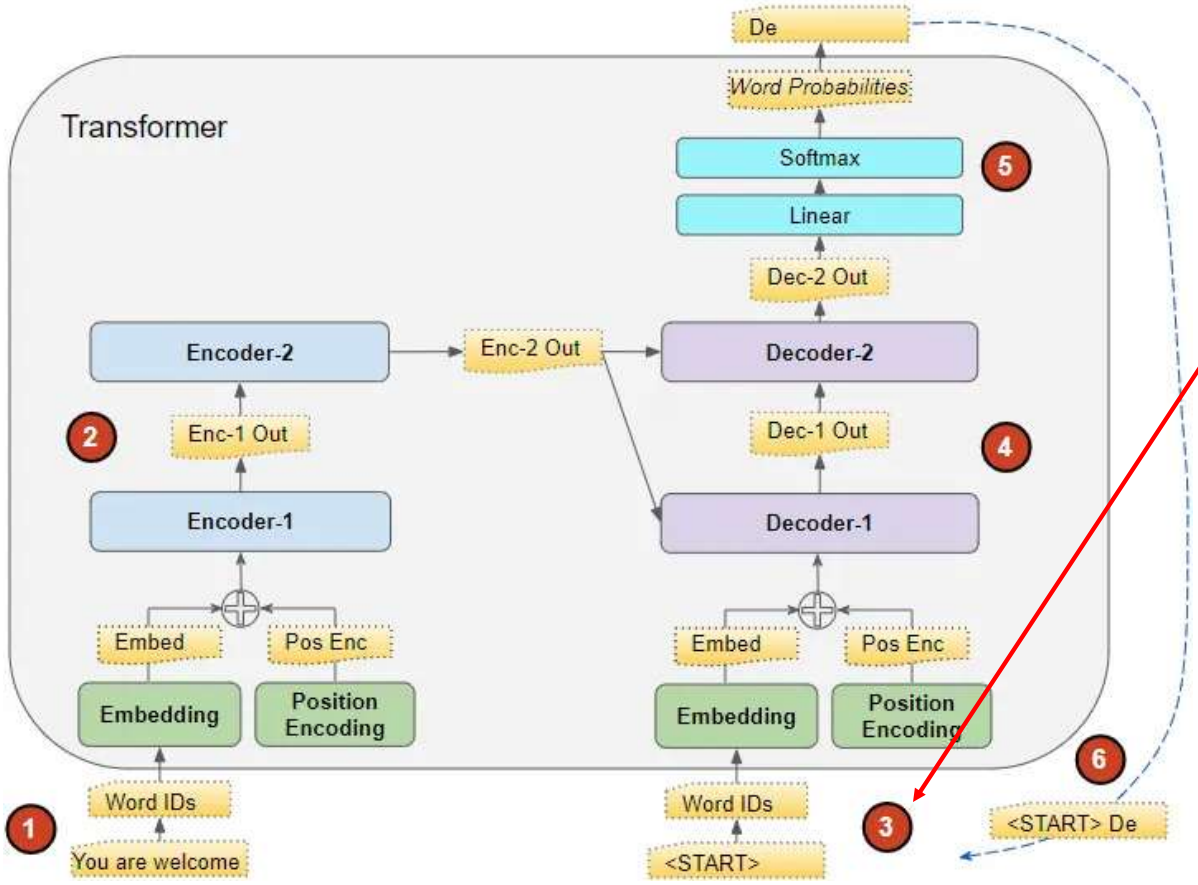
The input sequence is converted into Embeddings (with Position Encoding) and fed to the Encoder

Working: Inference



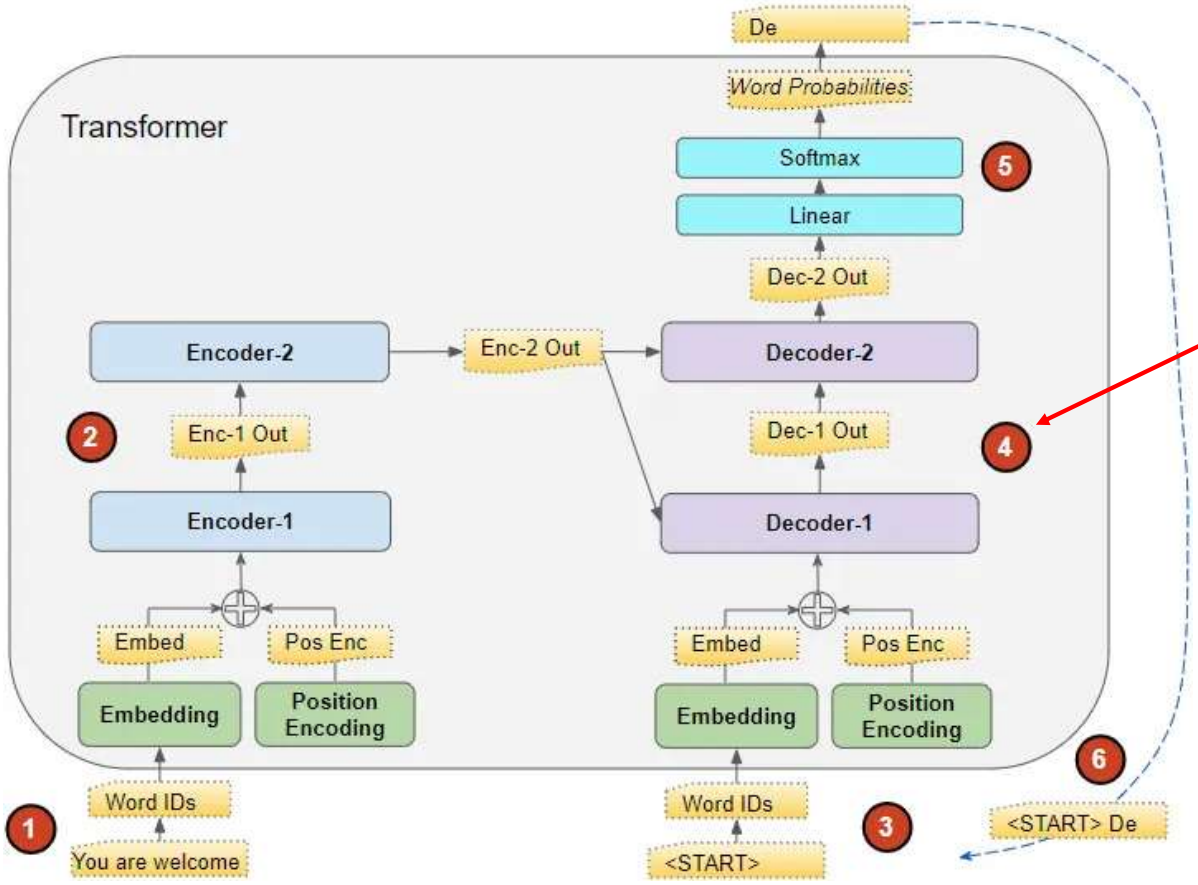
The stack of Encoders processes this and produces an encoded representation of the input sequence.

Working: Inference



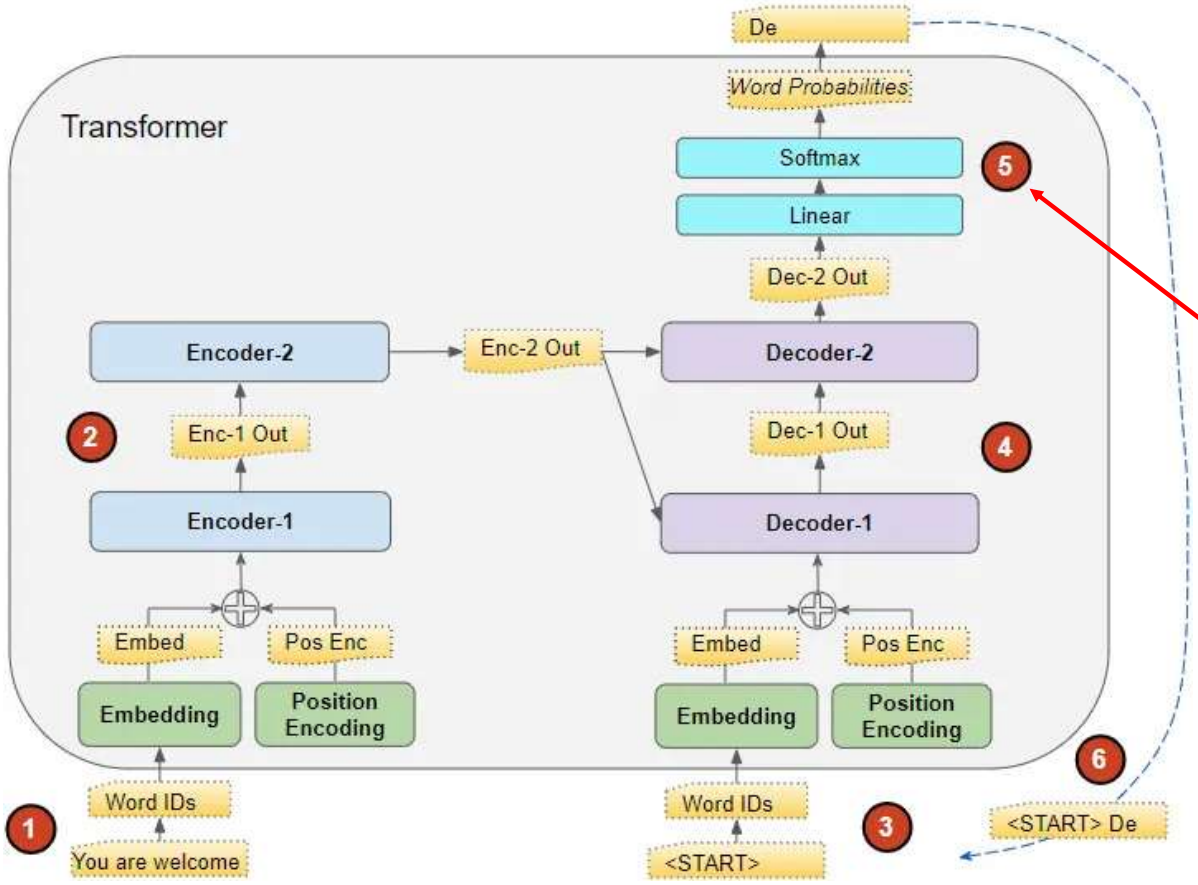
Instead of the target sequence, we use an empty sequence with only a start-of-sentence token. This is converted into Embeddings (with Position Encoding) and fed to the Decoder.

Working: Inference



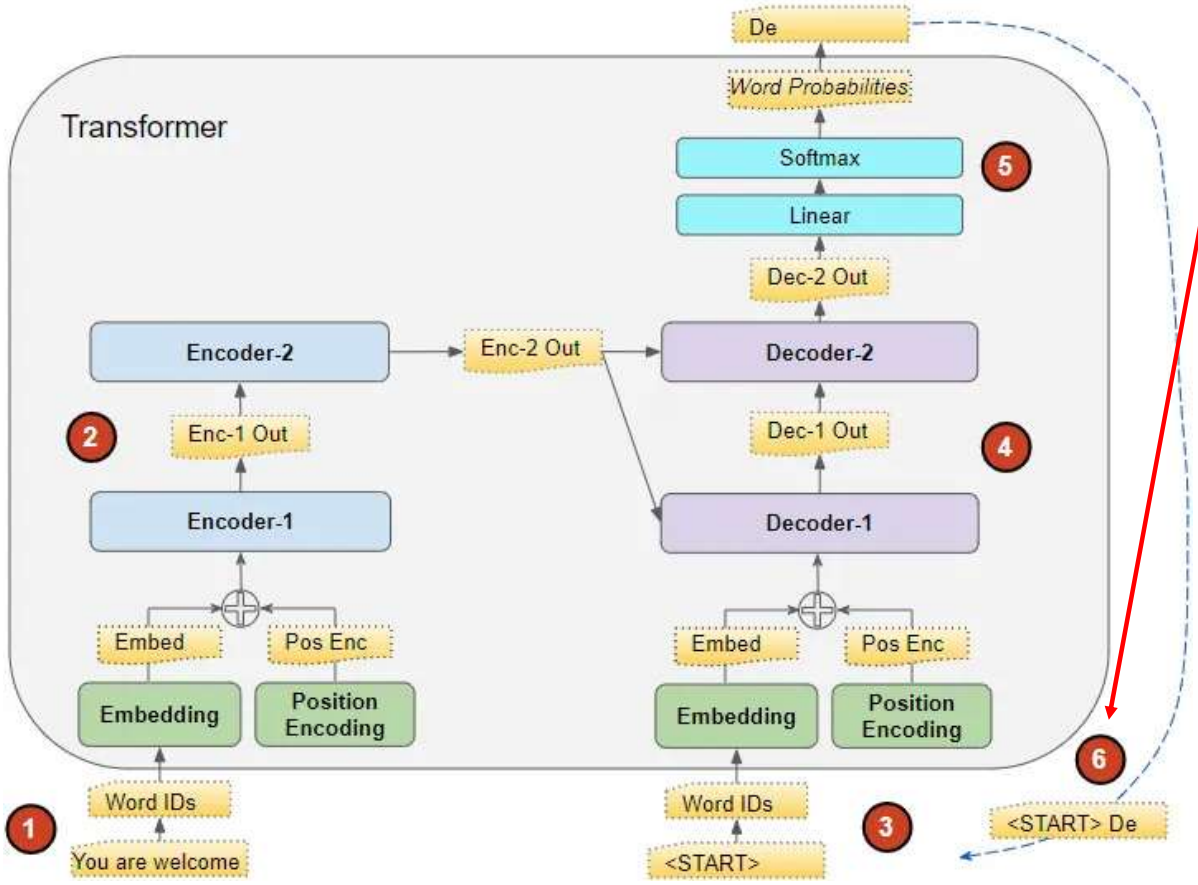
The stack of Decoders processes this along with the Encoder stack's encoded representation to produce an encoded representation of the target sequence.

Working: Inference



The stack of Decoders processes this along with the Encoder stack's encoded representation to produce an encoded representation of the target sequence.

Working: Inference



We take the last word of the output sequence as the predicted word. That word is now filled into the second position of our Decoder input sequence, which now contains a start-of-sentence token and the first word.

Go back to step #3. As before, feed the new Decoder sequence into the model. Then take the second word of the output and append it to the Decoder sequence. Repeat this until it predicts an end-of-sentence token. Note that since the Encoder sequence does not change for each iteration, we do not have to repeat steps #1 and #2 each time

Transformer Explainer

https://poloclub.github.io/transformer-explainer/

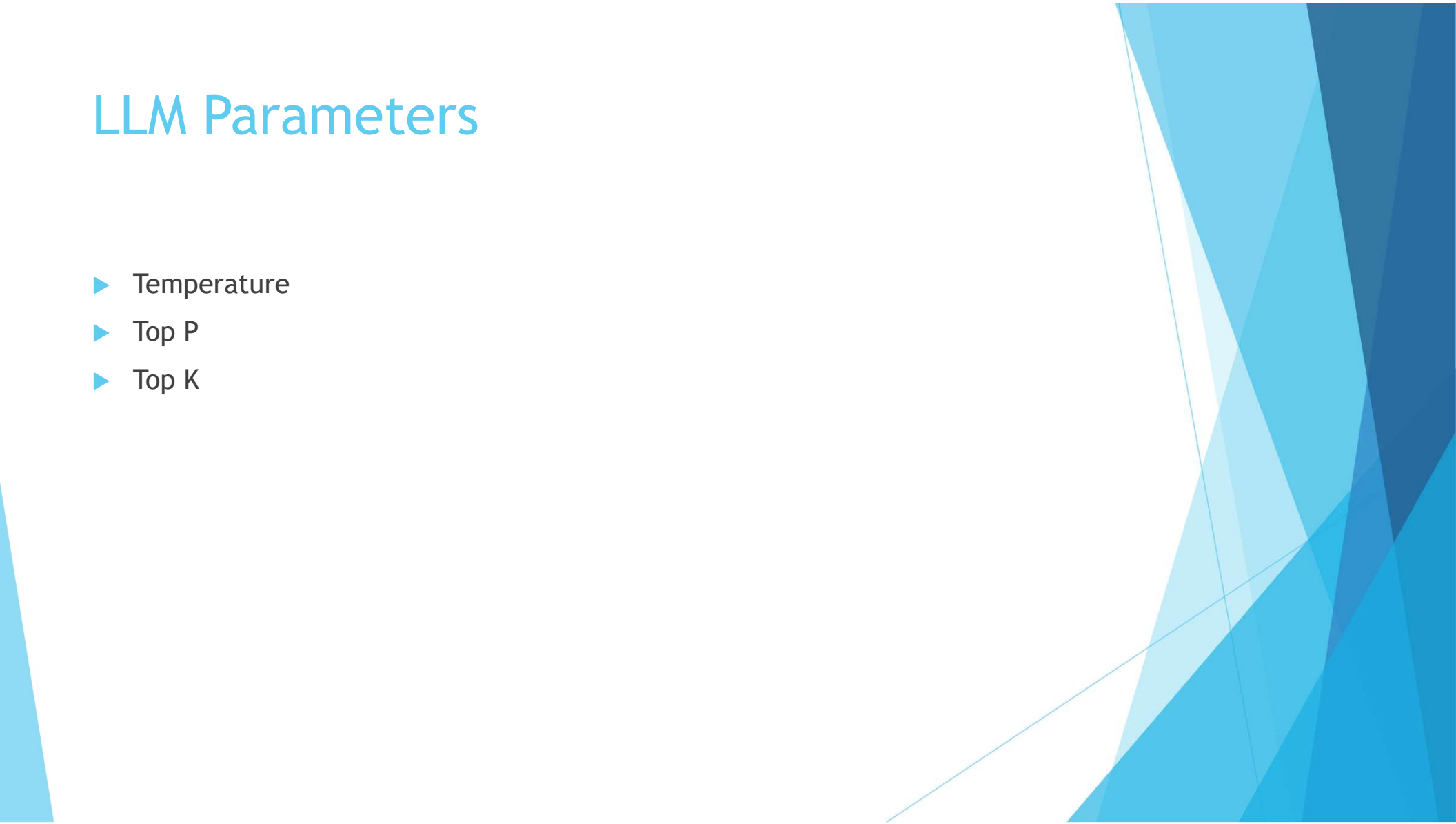
https://bbycroft.net/llm

Large Language Models

- **Definition:** LLMs are neural network models trained on vast amounts of text data to understand, generate, and manipulate human language.
- **Examples:** GPT-3, GPT-4, BERT, T5.
- **Applications:** Chatbots, translation, summarization, text generation, and more.
- **Pre-training**: LLMs are first trained on massive datasets to predict words, fill in blanks, or generate text.
- **Fine-tuning**: LLMs are fine-tuned for specific tasks like translation, summarization, etc.
- **Data**: They are trained on datasets like Wikipedia, Common Crawl, and specialized datasets.

LLM Parameters

- Temperature
- Top P
- Top K



Temperature

- **Definition**: Temperature is a hyperparameter that influences the randomness of predictions made by the model. It is applied to the logits (raw output scores) before converting them into probabilities using the softmax function.
- How it works:
  - A low temperature (e.g., 0.2) makes the model more confident, producing outputs that are more deterministic and likely to choose the highest probability words. This can result in more repetitive and less creative text.
  - A high temperature (e.g., 1.0 or above) increases randomness, allowing for more diverse outputs. However, it may also produce nonsensical or less coherent text.
- **Usage**:
  - Adjusting the temperature can help control the balance between creativity and coherence in generated text. For example:
    - Temperature = 0.5: Produces coherent and somewhat varied text.
    - Temperature = 1.0: Generates creative but potentially less coherent text.

Top K

- **Definition**: Top-k sampling is a technique where, at each step of generation, the model selects from the top k most probable words. All other words are excluded from consideration.
- How it works:
  - After obtaining the probabilities of the next word, the model sorts the words and only keeps the top k candidates. The final selection is made by sampling from this restricted set.
- **Usage**:
  - Setting a low k (e.g., 10) will limit the choices to a small number of likely candidates, leading to more coherent but less diverse outputs.
  - Setting a high k (e.g., 1000) allows for more creativity but may introduce less coherence, as many less likely words are considered.

Top P

- **Definition**: Top-p sampling, or nucleus sampling, selects from the smallest set of words whose cumulative probability exceeds a threshold p. This means that it adapts the number of words considered based on their probabilities.
- How it works:
  - After obtaining the probabilities, the model sorts them and calculates the cumulative probability. It selects words until the cumulative probability surpasses the threshold p.
- **Usage**:
  - A lower p (e.g., 0.7) results in fewer options, similar to top-k sampling but more adaptive to the actual distribution of probabilities.
  - A higher p (e.g., 0.9) retains more options, allowing for more diversity while still maintaining a level of coherence based on the model's predictions.

Summary

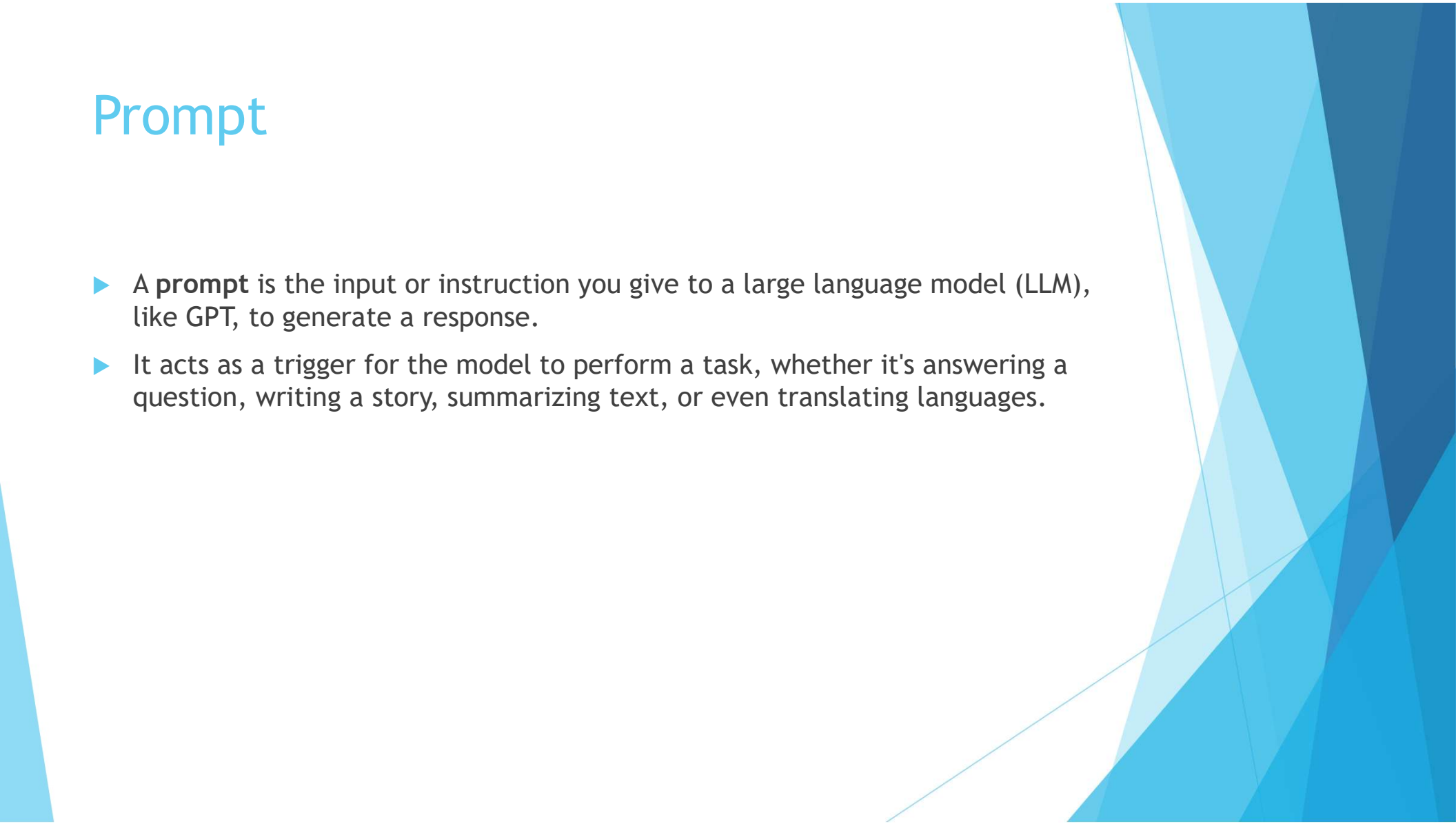
- Temperature: Controls the randomness of predictions. Lower values lead to more conservative outputs, while higher values encourage creativity.
- Top-K Sampling: Limits the choice to the top k words. Affects the balance between coherence and diversity based on the selected k.
- Top-P Sampling: Adapts the number of candidates based on their cumulative probability. Balances the output's diversity and coherence more dynamically than top-k.

Example

- If you're generating text about cats:
- **Temperature**: At 0.5, you might get a straightforward sentence like "Cats are popular pets." At 1.0, you might get "Cats, with their whimsical tails and mischievous antics, often reign supreme in cozy homes!"
- **Top-K**: If k=10, the output will be limited to the ten most likely next words, maintaining coherence. If k=1000, it might choose unexpected or less relevant words.
- **Top-P**: With p=0.9, the model might still focus on likely words but will consider a broader set than top-k, allowing for more creative variations.

Prompt

- A **prompt** is the input or instruction you give to a large language model (LLM), like GPT, to generate a response.
- It acts as a trigger for the model to perform a task, whether it's answering a question, writing a story, summarizing text, or even translating languages.



Key Points

- **Input to the Model:**
  - A prompt is typically a sentence, question, or phrase you type into the system to guide the model in generating the desired response.
  - Example: If you ask, "What is the capital of France?" the prompt is the question itself, and the model will respond with "Paris."
- **Prompt Shapes the Response:**
  - How you phrase the prompt can influence the quality and type of response. Clearer or more specific prompts often lead to better or more accurate answers.
  - Example: Instead of "Tell me about cars," you can specify, "What are the advantages of electric cars over gasoline cars?"

Key Points

- **Task Specification:**
  - A prompt can also specify the task the model should perform. For example, you can ask the model to "summarize this text" or "translate this sentence to French."
- **Natural Language:**
  - Prompts are typically written in natural language, meaning you can ask the model questions or give commands as if you were talking to a human.

Types of Prompts

- **Direct Question**: "What is the tallest mountain in the world?"
- **Instructional Prompt**: "Write a poem about the ocean."
- **Completion Prompt**: "The story begins on a stormy night, and then..."
- **Summarization**: "Summarize the key points of this article."

Why Prompts Matter?

- **Precision**: The more specific and clear the prompt, the better the output.
- **Creativity**: Open-ended prompts can encourage more creative responses, while specific ones lead to focused answers.
- **Customization**: Prompts can be designed to make the model focus on a particular style, tone, or format of response.

Prompt Engineering

- **Prompt engineering** involves designing and phrasing prompts to get the most accurate, detailed, or creative responses from large language models (LLMs).
- The following are some of the common types of prompt engineering techniques:

Zero-Shot Prompting

- **Definition**: In **zero-shot prompting**, you ask the model to complete a task without providing any examples or prior information. The model must generate a response based solely on the prompt and its pre-trained knowledge.
- **Use Case**: Useful when you want the model to handle new tasks without training it on specific examples.
- **Example**:
  - **Prompt**: "Translate this sentence into Spanish: 'I love learning new things.'"
  - **Response**: "Me encanta aprender cosas nuevas."
- **Effect**: This approach relies on the model's ability to generalize from its training data, but it can sometimes lead to less accurate or detailed responses.

One-Shot Prompting

- **Definition**: In **one-shot prompting**, you provide the model with one example of the task before asking it to generate a response. The model uses this example as a reference for its output.
- **Use Case**: When you want the model to understand the format or tone of the task but don't want to provide too many examples.
- **Example**:
  - **Prompt**: "Translate this sentence into French: 'I am happy.' Example: 'I am tired' becomes 'Je suis fatigué.' Now translate: 'I am happy.'"
  - **Response**: "Je suis heureux."
- **Effect**: Improves the accuracy of responses compared to zero-shot prompting by giving the model a single reference point.

Few-Shot Prompting

- **Definition**: In **few-shot prompting**, you give the model a few examples of the task to help it understand the pattern before generating a response. This is especially useful for tasks that require a specific structure or style.
- **Use Case**: Ideal for tasks where context or style is important, such as classification, text completion, or creative writing.
- **Example**:
  - **Prompt**: "Translate the following sentences into French:
    - 'I am tired.' -> 'Je suis fatigué.'
    - 'I am hungry.' -> 'J'ai faim.' Now, translate: 'I am excited.'"
  - **Response**: "Je suis excité."
- **Effect**: With more examples, the model can better understand and perform the task, leading to more accurate results.

Chain-of-Thought Prompting (CoT)

- **Definition**: **Chain-of-thought prompting** encourages the model to break down complex problems into intermediate steps before providing a final answer. It helps the model reason through each step of a task, improving its ability to solve multi-step problems.
- **Use Case**: Great for reasoning tasks like math problems, logic puzzles, or tasks that require a detailed thought process.
- **Example**:
  - **Prompt**: "If I have 5 apples and I give 2 apples to Sarah, and then I buy 3 more apples, how many apples do I have? Let's think step by step."
  - **Response**: "I start with 5 apples. I give 2 apples to Sarah, so now I have 3 apples. Then I buy 3 more apples, so now I have 6 apples."
- **Effect**: CoT prompting leads to more accurate responses for complex reasoning tasks because the model is forced to think through the steps logically.

Instruction Prompting

- **Definition**: **Instruction prompting** gives the model clear, explicit instructions on how to perform a task, usually in a step-by-step format. The prompt acts like a direct command or guide for the model to follow.
- **Use Case**: Best for tasks where you want very specific outputs or a particular structure, such as report generation, email writing, or summarization.
- **Example**:
  - **Prompt**: "Summarize the following article in 3 bullet points: 'Artificial intelligence is transforming industries by improving efficiency, automating tasks, and generating insights from large datasets.'"
  - **Response**:
    - AI improves efficiency in various industries.
    - It automates repetitive tasks.
    - AI helps generate insights from big data.
- **Effect**: Clear instructions help the model produce structured, concise, and specific outputs, reducing the chance of ambiguous or incorrect responses.

Reverse Prompting

- **Definition**: In reverse prompting, instead of providing an instruction directly, you give the model the desired outcome or result and let it infer what instructions would generate that result.
- **Use Case**: This is useful in cases where the model is asked to guess the rules or steps needed to get a result.
- **Example**:
  - **Desired Result**: "Paris is the capital of France."
  - **Reverse Prompt**: "Give the capital of a major European country starting with 'F'."
- **Effect**: This can help in generating more creative responses by letting the model "reverse engineer" the steps.

Contrastive Prompting

- **Definition**: Contrastive prompting asks the model to compare two or more options and provide reasoning for why one is better or different from the other.
- **Use Case**: Useful for tasks involving decision-making, evaluations, or comparisons.
- **Example**:
  - **Prompt**: "What are the differences between electric cars and gasoline cars, and which is better for the environment?"
- **Effect**: Helps the model create a balanced, reasoned answer by forcing it to contrast different elements.

Meta-Prompting

- **Definition**: Meta-prompting is when the model is asked to generate its own prompts based on a higher-level task. It's like asking the model to think of a good question to solve a problem or achieve a task.
- **Use Case**: Used when you need the model to explore different approaches or generate creative solutions.
- **Example**:
  - **Prompt**: "What questions should I ask to understand climate change better?"
- **Effect**: Generates multiple useful prompts or directions that a user may not have thought of.

Persona Prompting

- **Definition**: In persona prompting, you ask the model to assume a specific character, personality, or role, which affects the tone and style of the output.
- **Use Case**: Ideal for writing in a particular voice, like an expert, a historical figure, or a character.
- **Example**:
  - **Prompt**: "Explain quantum physics as if you are Albert Einstein."
- **Effect**: Adapts the output style and tone according to the persona, making it feel more personalized or aligned with the desired perspective.

Role-Specific Prompting

- **Definition**: Similar to persona prompting, but more focused on a specific professional or situational role (e.g., a teacher, doctor, or consultant).
- **Use Case**: Useful for generating professional responses in fields like law, medicine, or education.
- **Example**:
  - **Prompt**: "As a doctor, explain the symptoms of flu to a patient."
- **Effect**: Helps tailor responses to suit specific professional scenarios, improving relevance and usefulness.

Self-Consistency Prompting

- **Definition**: This technique involves querying the model multiple times on the same prompt, and then aggregating or analyzing the responses to select the most consistent or common answer.
- **Use Case**: Helps improve the reliability of the answer in uncertain tasks by looking for consistent reasoning.
- **Example**:
  - **Prompt**: "What is the most effective way to reduce carbon emissions?"
  - Ask multiple times, then check for consistency across responses.
- **Effect**: Reduces randomness and helps in obtaining a more reliable, stable answer by considering multiple outputs.

Contextual Expansion Prompting

- **Definition**: In this approach, you iteratively expand the context by giving the model more information in stages to guide it toward more accurate or detailed responses.
- **Use Case**: Useful for complex tasks where more background is needed to generate a precise response.
- **Example**:
  - **Prompt 1**: "Summarize this article in one sentence."
  - **Prompt 2**: "Now explain the main argument of the article in detail."
- **Effect**: This progressive approach allows the model to build up its understanding, leading to richer and more complete responses.

Exploration Prompting

- **Definition**: This involves asking the model to generate multiple possible solutions or answers to a problem, encouraging divergent thinking.
- **Use Case**: Useful in brainstorming, creative writing, or problem-solving scenarios where multiple ideas are needed.
- **Example**:
  - **Prompt**: "What are five different ways to improve productivity in remote work environments?"
- **Effect**: Expands the range of possibilities, making the model more creative by exploring various solutions or approaches.

Multi-Task Prompting

- **Definition**: In multi-task prompting, you ask the model to handle multiple different tasks within a single prompt. This helps generate complex responses that combine different actions like summarization and sentiment analysis.
- **Use Case**: Ideal when you need the model to perform more than one task on the same data or content.
- **Example**:
  - **Prompt**: "Summarize this text and then tell me whether it's positive, negative, or neutral in tone."
- **Effect**: This technique allows the model to handle multi-step, multi-task problems efficiently in one go.

Deliberation Prompting

- **Definition**: In deliberation prompting, you ask the model to evaluate its own answers and improve them. It involves asking the model to “rethink” or “revise” a given response.
- **Use Case**: Useful for generating refined or improved answers after an initial response, especially in creative writing, reasoning tasks, or when accuracy is critical.
- **Example**:
  - **Prompt**: "Generate a response to this question. Now, evaluate your response and improve it."
- **Effect**: Helps in refining the quality of answers by pushing the model to reanalyze and improve its previous outputs.

Summary

Type	Description	Effect
Zero-Shot Prompting	No examples are given, the model relies on its general knowledge.	May be less accurate, but faster to generate.
One-Shot Prompting	Provides one example for the model to follow.	Improves output accuracy with minimal example.
Few-Shot Prompting	Several examples are provided to help the model understand the task.	More examples lead to better task understanding.
Chain-of-Thought Prompting	Breaks down complex problems into smaller reasoning steps.	Improves multi-step reasoning and accuracy.
Instruction Prompting	Gives direct, step-by-step instructions for the model to follow.	Clear, structured, and predictable output.

Summary

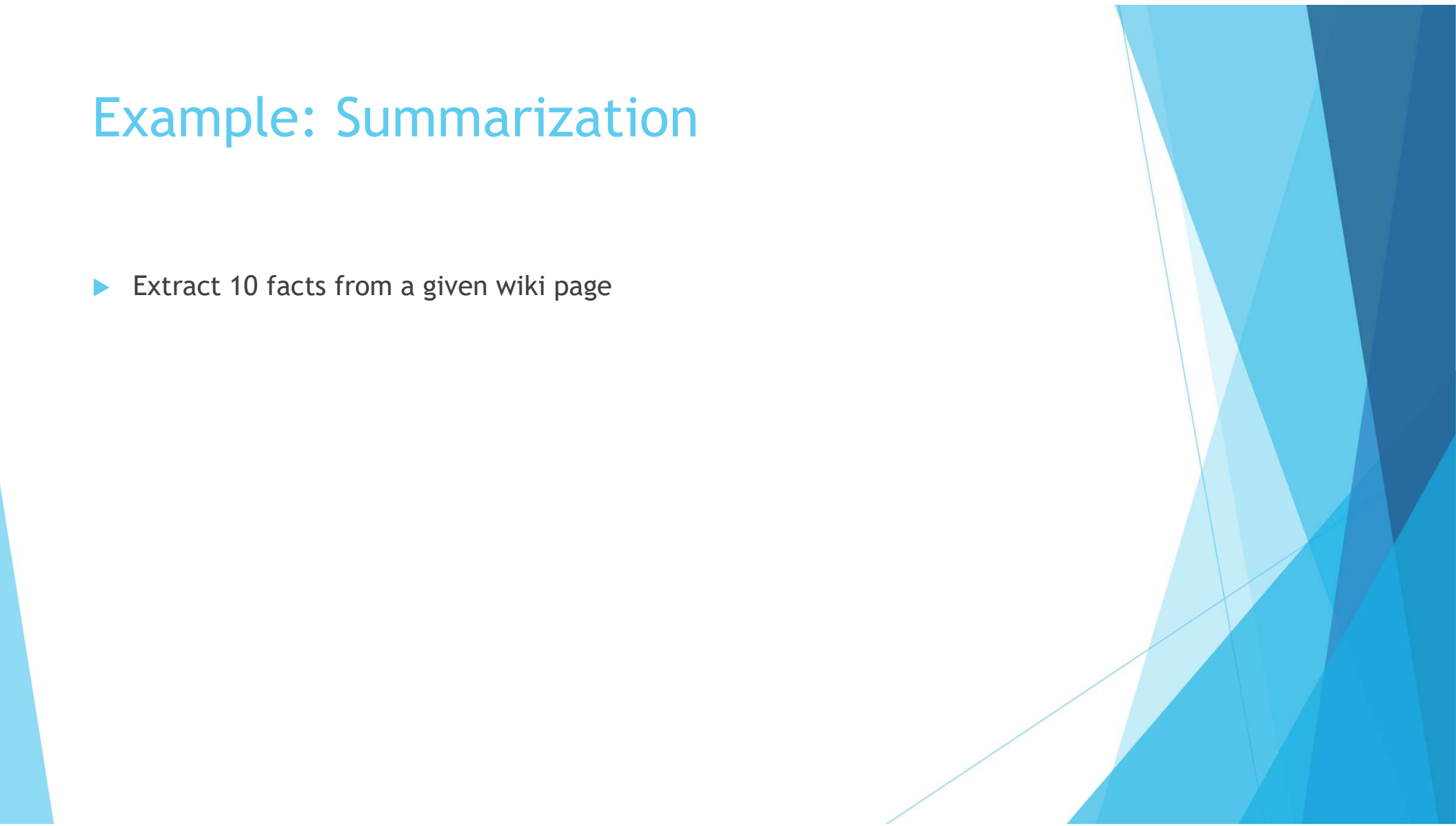
Technique	Description	Effect
Reverse Prompting	Model infers the instruction from the desired outcome.	Generates creative, indirect responses.
Contrastive Prompting	Compare two or more items and generate reasoning.	Produces balanced, reasoned evaluations.
Meta-Prompting	Model generates its own prompts to explore new questions.	Encourages diverse questions or approaches.
Persona Prompting	Model takes on a specific personality or character.	Personalizes the response tone and style.
Role-Specific Prompting	Model assumes a professional role for task completion.	Tailors responses for specific scenarios.

Summary

Technique	Description	Effect
Self-Consistency Prompting	Query model multiple times to find the most consistent answer.	Improves reliability and reduces randomness.
Contextual Expansion	Gradually expand context to guide the model to better responses.	Results in detailed, accurate outputs.
Exploration Prompting	Model generates multiple solutions or ideas.	Boosts creativity and divergent thinking.
Multi-Task Prompting	Multiple tasks are handled in a single prompt.	Combines complex tasks efficiently.
Deliberation Prompting	Model evaluates and improves its own output.	Refines and enhances answer quality.

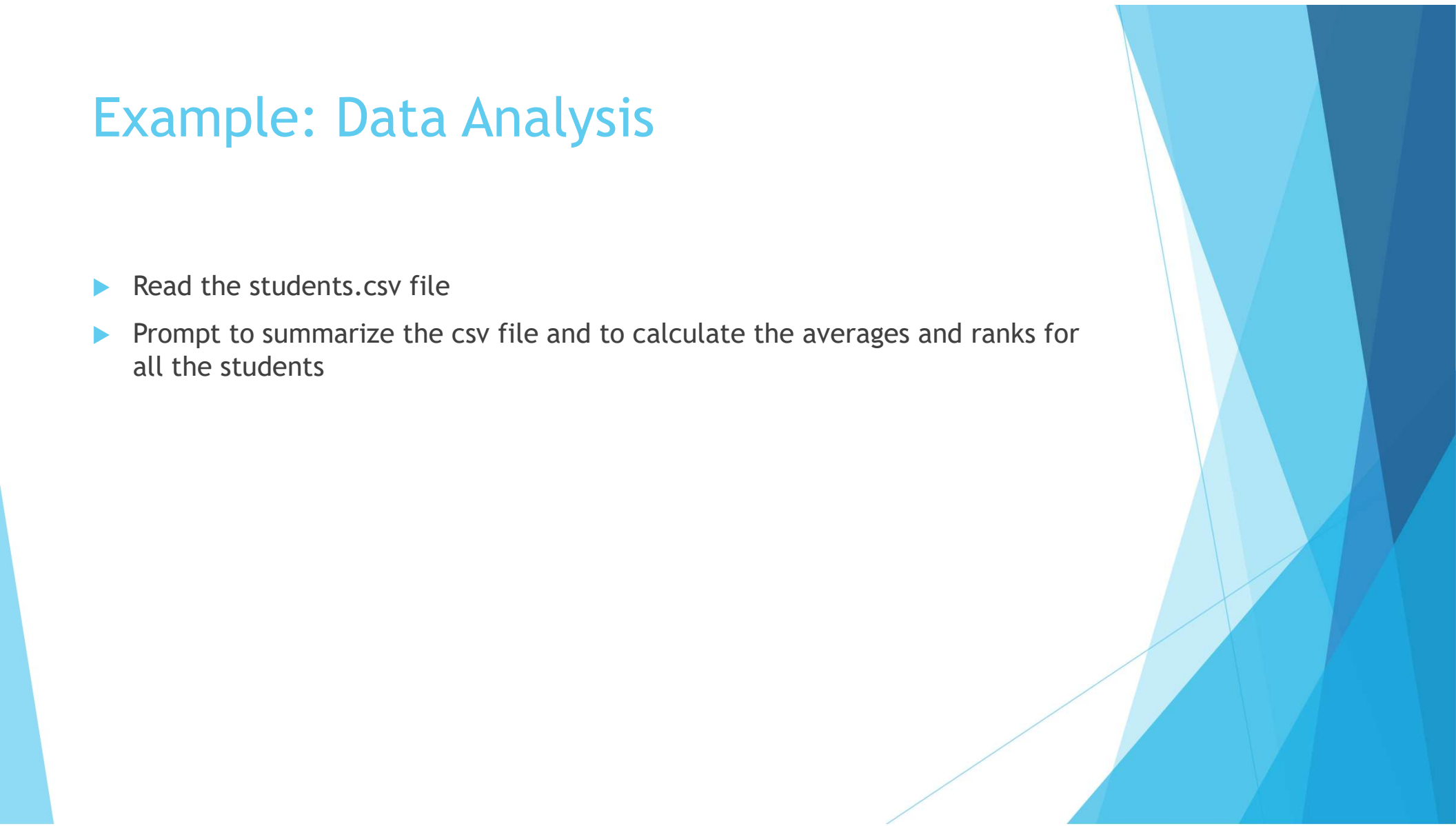
Example: Summarization

- Extract 10 facts from a given wiki page



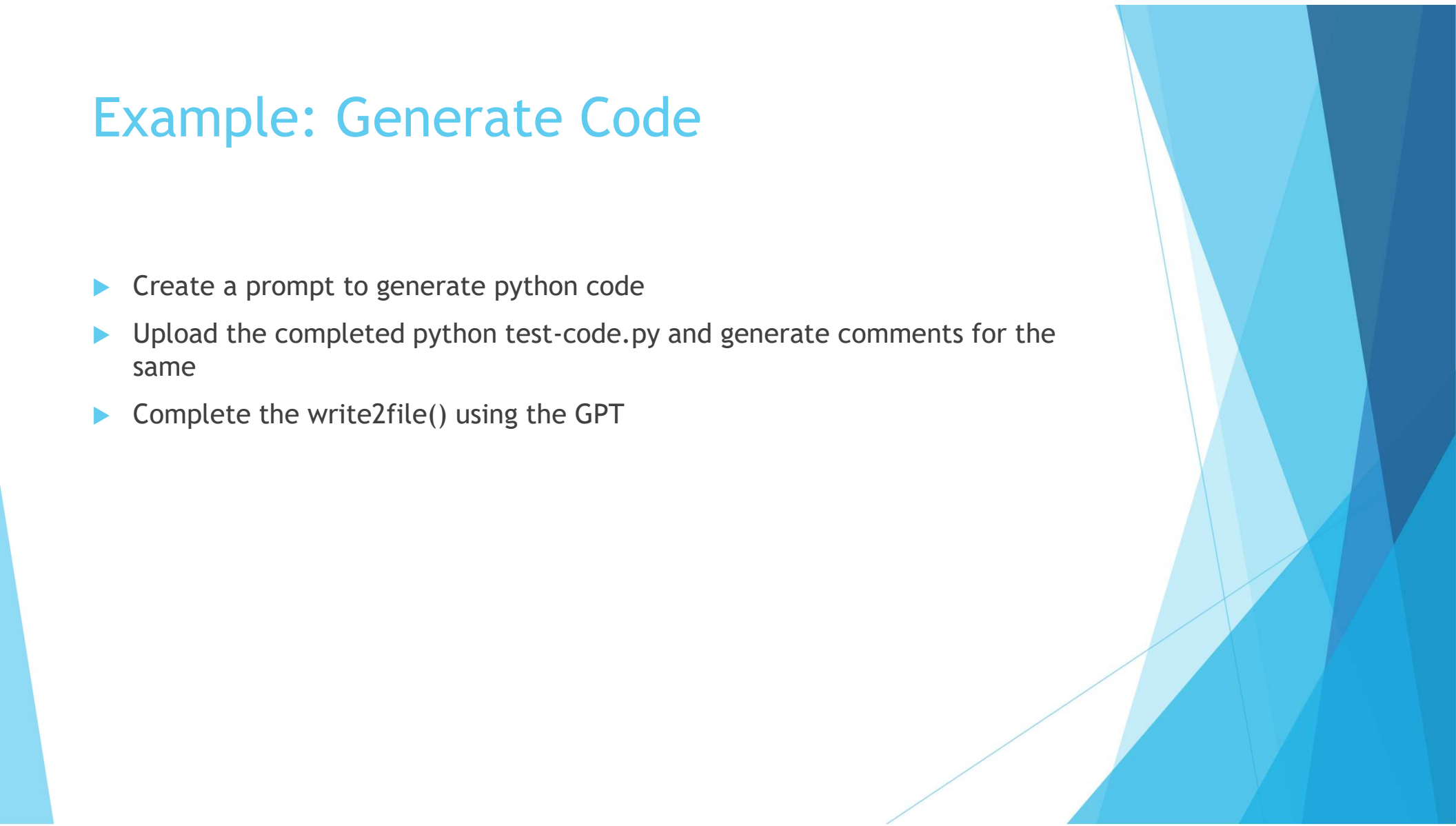
Example: Data Analysis

- Read the students.csv file
- Prompt to summarize the csv file and to calculate the averages and ranks for all the students



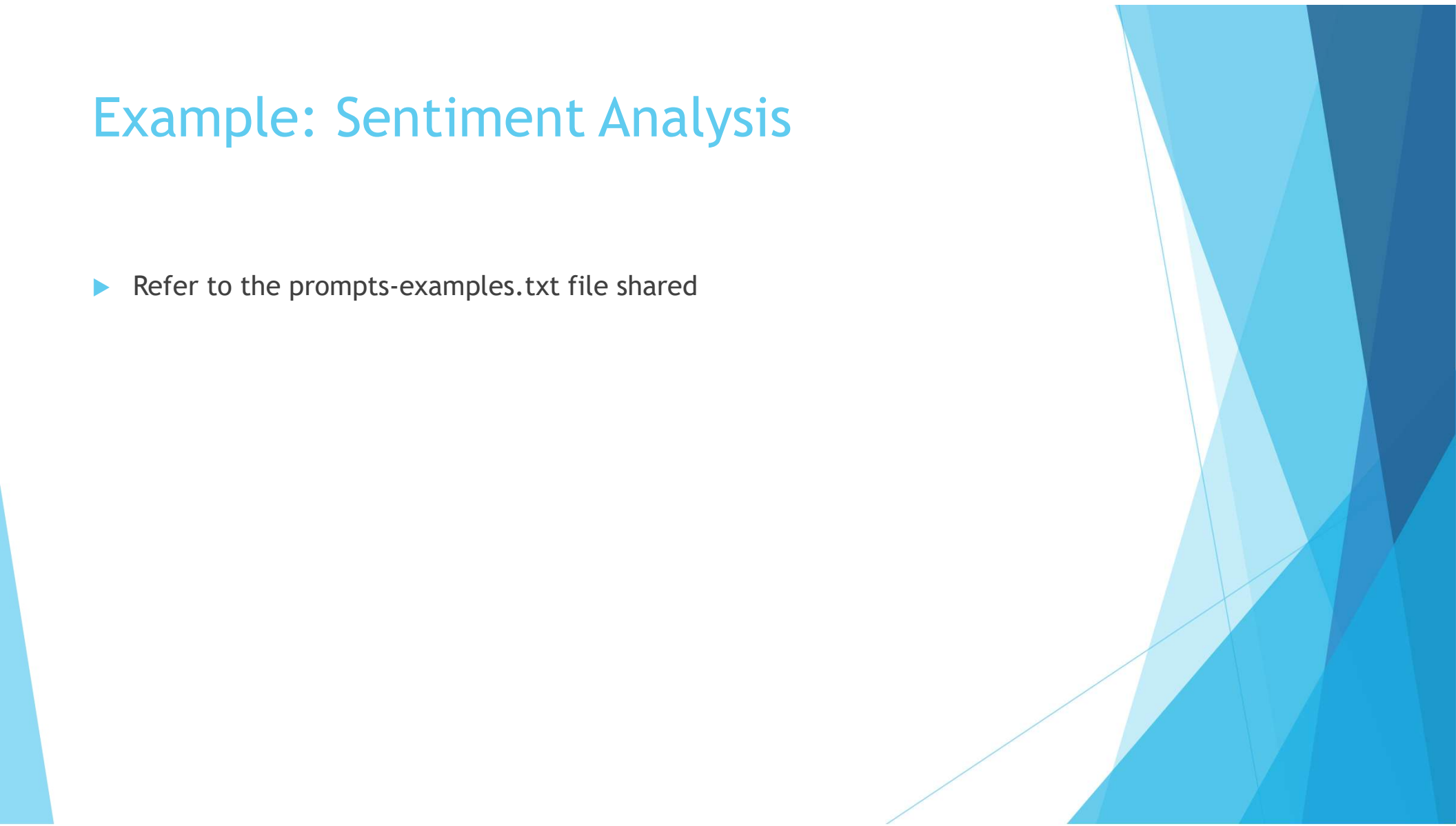
Example: Generate Code

- Create a prompt to generate python code
- Upload the completed python test-code.py and generate comments for the same
- Complete the write2file() using the GPT



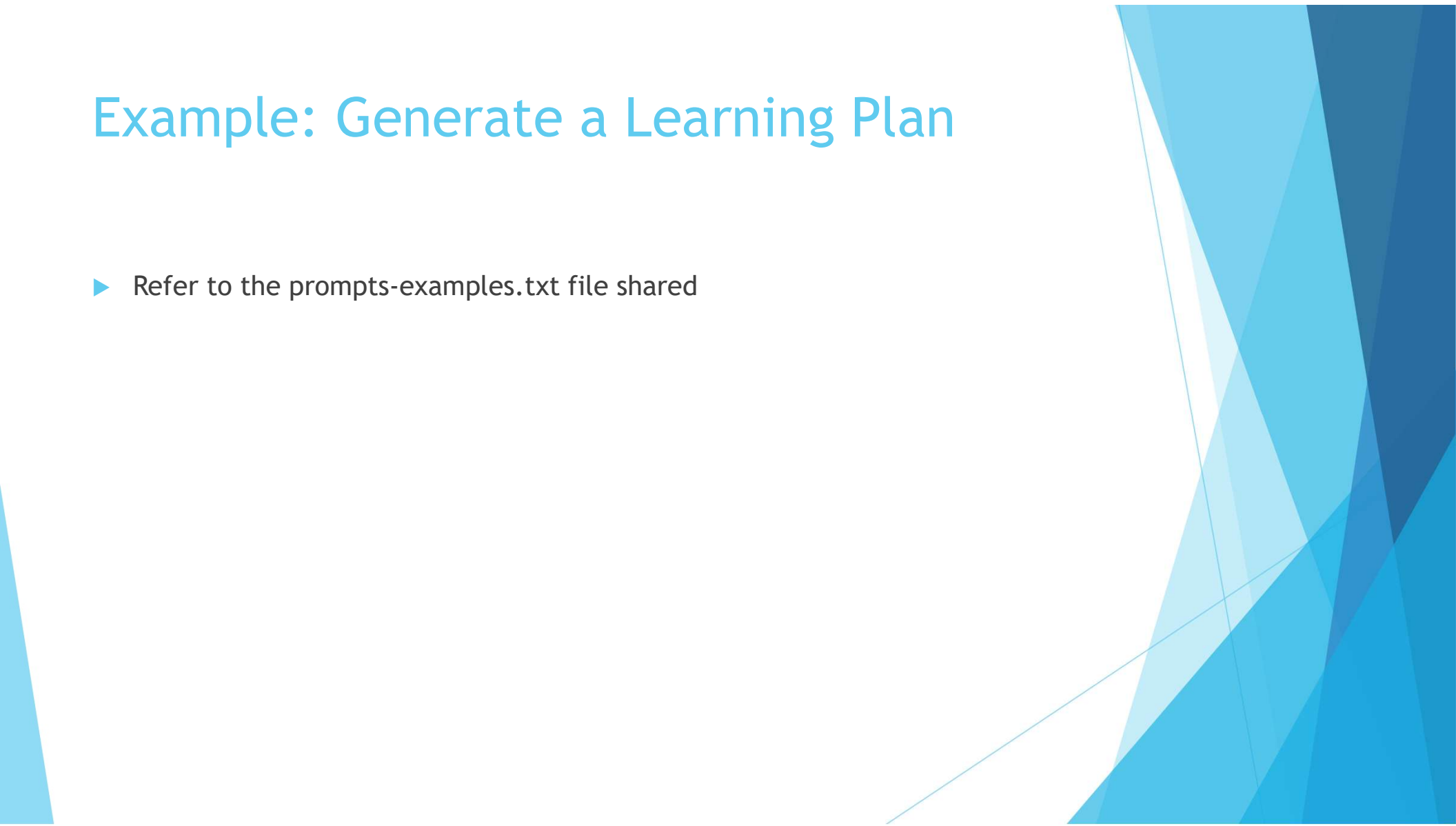
Example: Sentiment Analysis

- Refer to the prompts-examples.txt file shared



Example: Generate a Learning Plan

- Refer to the prompts-examples.txt file shared



Controlling Hallucinations

- In the context of large language models (LLMs), hallucination refers to the generation of confident but factually incorrect or nonsensical information. This occurs because the model predicts text based on patterns in the training data rather than verifying facts.
- Controlling Hallucinations:
  - **Fine-tuning**: Train the model on high-quality, domain-specific data.
  - **Retrieval-Augmented Generation (RAG)**: Integrate external knowledge bases or APIs to provide factual references during generation.
  - **Fact-checking**: Use post-generation verification techniques.
  - **Prompt Engineering**: Design clear, specific, and well-structured prompts to guide the model.
  - **Human-in-the-Loop**: Employ humans to review and validate outputs in critical applications.

Transfer Learning

- **Transfer Learning** is a machine learning technique where a model developed for one task is reused as the starting point for a model on a second, related task.
- This approach leverages the knowledge learned from the first task to improve the performance and reduce the training time for the second task.
- **Pre-trained Models**:
  - A model is first trained on a large dataset (often on a broad task) and then fine-tuned for a specific task with a smaller, domain-specific dataset.
  - **Example**: Using a model pre-trained on a large image dataset like ImageNet and fine-tuning it for a specific task like medical image classification.

Fine Tuning LLMs

- Fine-tuning involves training a pre-trained LLM on a specific dataset to tailor its performance to a particular domain or task. It adjusts the model's weights slightly while retaining its general capabilities, making it more accurate and reliable for the desired use case.
- **Steps in Fine-Tuning:**
  - Prepare domain-specific labeled data.
  - Use transfer learning on the pre-trained model.
  - Validate and test to ensure improvements in performance.
- **Example:** Fine-tuning an LLM with medical literature to enhance its accuracy in answering healthcare-related queries.

Fine Tuning Types

- **Full Fine-Tuning**
  - The entire model is retrained on task-specific data.
  - **Pros**: High accuracy and customization.
  - **Cons**: Computationally expensive and requires a large dataset.
- **Parameter-Efficient Fine-Tuning (PEFT)**
  - Adjusts only a subset of model parameters, such as adapters or embeddings, while keeping the majority of the model frozen.
  - **Examples**:
    - **LoRA (Low-Rank Adaptation)**: Inserts trainable low-rank matrices into the model layers.
    - **Prefix Tuning**: Trains small task-specific prefix vectors added to the input embeddings.
    - **Adapter Layers**: Introduces small, trainable modules into the transformer layers.
  - **Pros**: Reduces computational cost and memory usage.

Fine Tuning Types

- **Instruction Tuning**
  - Fine-tunes the model on a collection of prompts and responses to align it better with user instructions.
  - **Example**: OpenAI's GPT models fine-tuned for ChatGPT-like behavior.
  - **Use Case**: Conversational agents or chatbots.
- **Few-Shot Fine-Tuning**
  - Trains the model on a small number of examples per task.
  - **Pros**: Useful for tasks with limited labeled data.
  - **Cons**: May result in overfitting if not carefully managed.

Fine Tuning Types

- **Continual Learning**
  - Incrementally fine-tunes the model on new data over time to keep it updated without forgetting previous knowledge.
  - **Use Case**: Applications requiring regular updates (e.g., news summarization).
- **Contrastive Learning-Based Fine-Tuning**
  - Trains the model to better understand distinctions between similar inputs by using contrastive loss.
  - **Use Case**: Improving understanding of context-sensitive tasks.
- **Multitask Fine-Tuning**
  - Fine-tunes the model on multiple tasks simultaneously to improve generalization.
  - **Example**: Training a model to perform both sentiment analysis and text summarization.

RLHF

- **Reinforcement Learning from Human Feedback** is a technique where the model is trained based on feedback from humans, typically in the form of preferences or ratings, rather than explicit labeled data.
- It uses human-generated signals (positive or negative feedback) to shape the model's behavior, optimizing it to better align with human values or desired outputs.

- **How Does RLHF Work?**
  - **Step 1**: The model generates responses to tasks or prompts.
  - **Step 2**: Human annotators rank or provide feedback on these responses (e.g., which answer is better or more aligned with expectations).
  - **Step 3**: A reward model is trained on this feedback to quantify how well the model's responses align with human preferences.
  - **Step 4**: Reinforcement learning algorithms (like Proximal Policy Optimization, PPO) are applied to fine-tune the model by maximizing the reward signal from human feedback.

RLHF: Advantages

- RLHF stands out by directly incorporating human preferences in the training process, making it highly effective in improving **ethical alignment (behavior and preferences) , reducing hallucination**, and **refining conversational abilities**.
- Unlike traditional fine-tuning methods, RLHF is particularly suited for tasks that require **subjective evaluation**, such as **dialogue generation** or **creative content generation**, where there's no clear-cut "correct" answer.

- **Use Cases:**
  - **Chatbots and Conversational Models:** Fine-tuning LLMs to behave more naturally and be more helpful, based on human conversation feedback.
  - **Content Moderation:** Aligning the model to avoid generating harmful, biased, or inappropriate content.
  - **Instruction Following:** Enhancing models to better follow complex instructions, understand nuances, and be more aligned with user intentions.

RLHF

- **Challenges:**
  - High Cost: Collecting human feedback and using it effectively in reinforcement learning is resource-intensive.
  - Bias in Feedback: Human feedback can be subjective or inconsistent, which could introduce biases in the model's learning.
  - Scalability: Gathering sufficient feedback for large-scale models and tasks can be difficult.

RAG

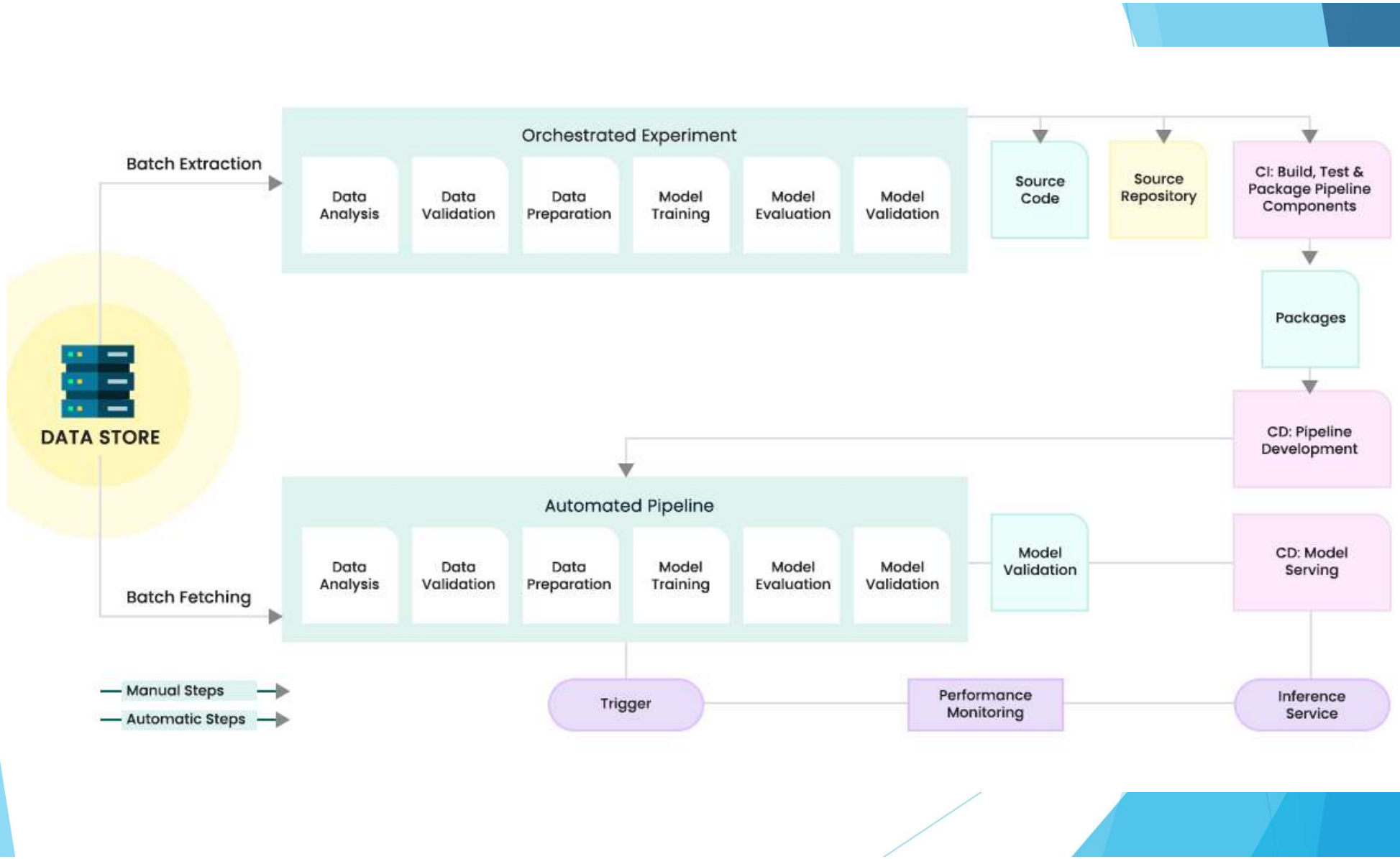
- RAG combines LLMs with external knowledge sources to ensure factual accuracy. The model retrieves relevant information from a database, search engine, or knowledge graph and uses it to generate responses.
- **How it works:**
  - The input query is used to search the external knowledge base.
  - The retrieved data is combined with the query.
  - The LLM generates an answer based on this augmented input.
- **Example:** A chatbot using RAG to fetch real-time financial data to answer questions about stock prices.

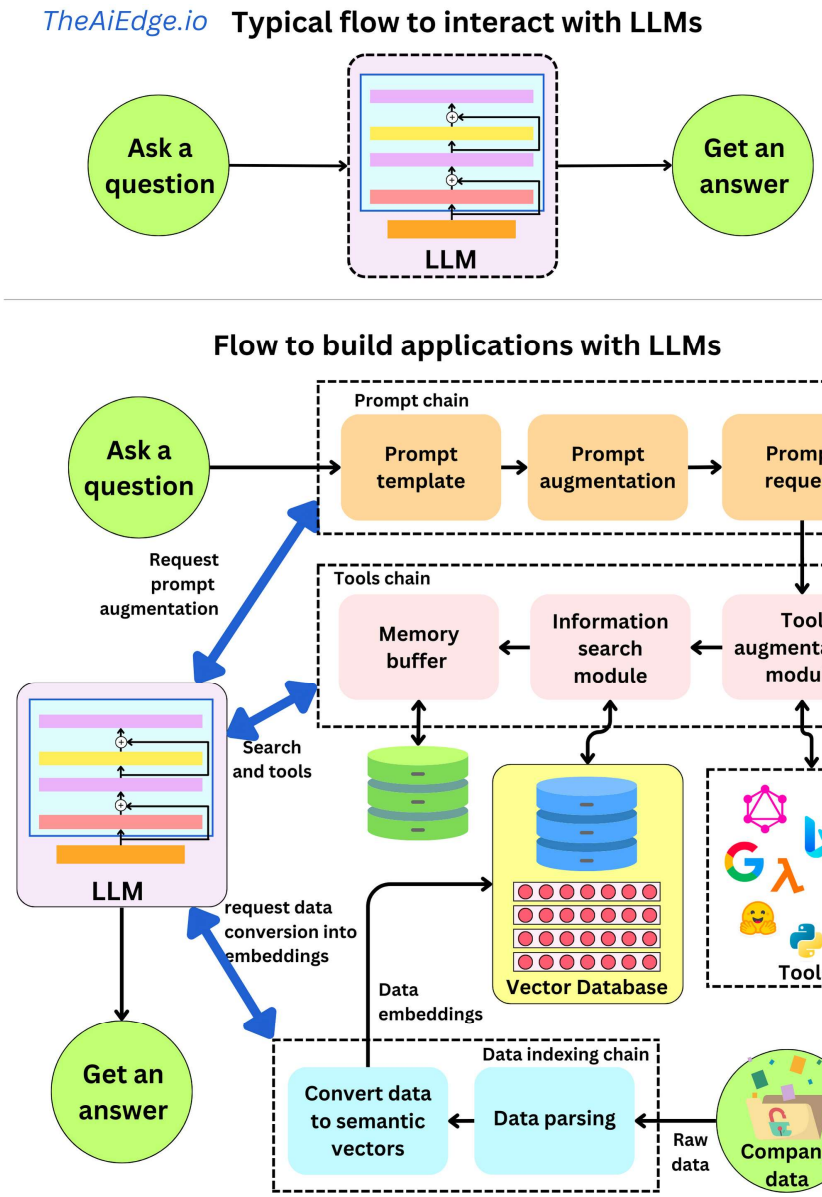
Human In The Loop

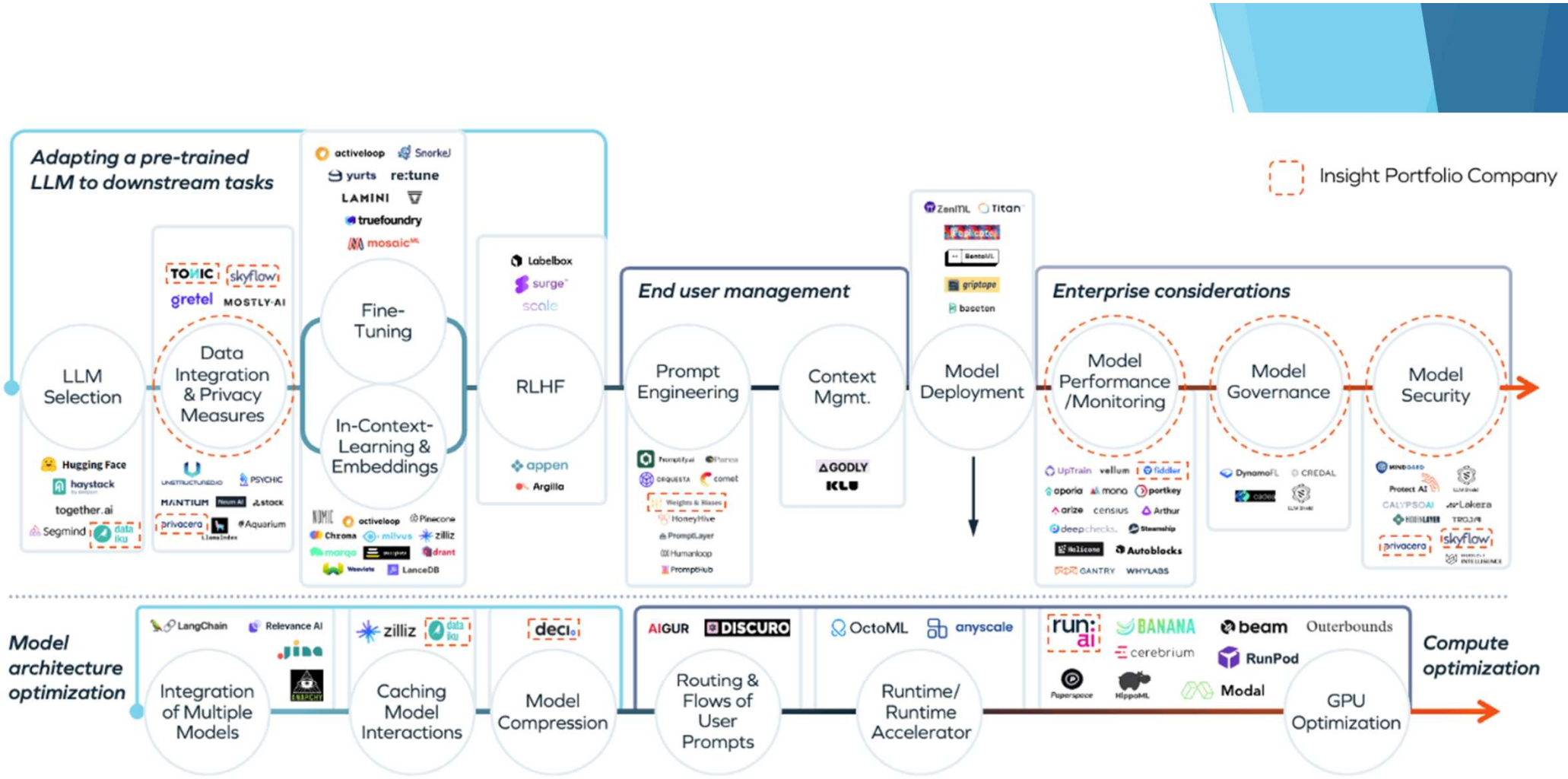
- HITL involves human oversight during model training, deployment, or response validation to ensure quality, accuracy, and ethical compliance.
- **Applications of HITL**:
  - **Training**: Humans curate and label data for better learning.
  - **Monitoring**: Humans review outputs to identify and correct issues like biases or inaccuracies.
  - **Feedback Loops**: Continuous human feedback helps improve the model over time.
- **Example**: A content moderation system where humans review flagged outputs to refine the model's filtering criteria.

LLM-OPS

- **LLMOps** (Large Language Model Operations) is a set of practices and tools for managing the lifecycle of large language models (LLMs), from development to deployment and monitoring.
- It aims to streamline the deployment, monitoring, and maintenance of LLMs in production environments.
- Difference between LLMOPS and MLOPS:
  - LLMOps is a specialized subset of MLOps (machine learning operations), which focuses specifically on the challenges and requirements of managing LLMs.
  - While MLOps covers the general principles and practices of managing machine learning models, LLMOps addresses the unique characteristics of LLMs, such as their large size, complex training requirements, and high computational demands.


Orchestrated Experiment
Data Analysis
Data Validation
Data Preparation
Model Training
Model Evaluation
Model Validation
Batch Extraction
Source Code
Source Repository
CI: Build, Test & Package Pipeline Components
Packages
CD: Pipeline Development
DATA STORE
Automated Pipeline
Data Analysis
Data Validation
Data Preparation
Model Training
Model Evaluation
Model Validation
Batch Fetching
Model Validation
CD: Model Serving
Manual Steps
Automatic Steps
Trigger
Performance Monitoring
Inference Service


TheAiEdge.io
Typical flow to interact with LLMs
Ask a question
LLM
Get an answer
Flow to build applications with LLMs
Ask a question
Prompt chain
Prompt template
Prompt augmentation
Prompt request
Request prompt augmentation
Tools chain
Memory buffer
Information search module
Tool augmentation module
Search and tools
LLM
Vector Database
Tools
request data conversion into embeddings
Data embeddings
Get an answer
Data indexing chain
Convert data to semantic vectors
Data parsing
Raw data
Company data


Insight Portfolio Company
Adapting a pre-trained LLM to downstream tasks
LLM Selection
Data Integration & Privacy Measures
Fine-Tuning
In-Context-Learning & Embeddings
RLHF
End user management
Prompt Engineering
Context Mgmt.
Model Deployment
Enterprise considerations
Model Performance /Monitoring
Model Governance
Model Security
Model architecture optimization
Integration of Multiple Models
Caching Model Interactions
Model Compression
Routing & Flows of User Prompts
Runtime/ Runtime Accelerator
GPU Optimization
Compute optimization

Ethical AI

- Ethical AI refers to the practice of designing, developing, and deploying artificial intelligence (AI) systems in a manner that aligns with ethical principles, values, and societal norms.
- The goal is to ensure that AI technologies are used responsibly and fairly, minimizing harm and promoting positive outcomes for individuals and society.

Ethical AI

- **Fairness**:
- AI should avoid discrimination and bias, ensuring equal treatment for all users. Bias can emerge from skewed training data or algorithms, which may unfairly impact certain groups (e.g., based on race, gender, or socioeconomic status).
- **Example**: Ensuring a hiring algorithm does not favor one demographic over another.
- **Transparency**:
- AI systems should be transparent and explainable, so users and stakeholders can understand how decisions are made. This is especially important in high-stakes areas like healthcare and finance.
- **Example**: Providing clear explanations for decisions made by AI systems

Ethical AI

- **Accountability**:
- There should be mechanisms to hold AI systems and their creators accountable for their actions and outcomes. This includes ensuring that AI systems are designed and monitored to avoid harm.
- **Example**: Clear responsibility for AI's decisions in automated processes, with human oversight.
- **Privacy and Data Protection**:
- AI systems must respect user privacy and comply with data protection regulations (e.g., GDPR). They should handle data securely and ensure that sensitive personal data is protected.
- **Example**: Ensuring AI models don't store or misuse personal data.

Ethical AI

- **Safety and Security**:
- AI systems should be designed to be safe, reliable, and secure, minimizing risks of unintended harmful behavior. This includes building safeguards against malicious attacks or system failures.
- **Example**: Preventing AI systems from being exploited for harmful purposes, such as creating deepfakes or autonomous weapons.
- **Inclusivity**:
- AI systems should be inclusive and accessible to all, ensuring that their benefits are distributed fairly across different demographics and communities.
- **Example**: Designing AI that works for people with disabilities or marginalized groups.

Ethical AI

- **Human Control**:
- AI systems should be designed to augment human decision-making, not replace it. Human oversight should be maintained, especially in critical domains like healthcare and justice.
- **Example**: Ensuring that AI assists doctors in diagnosing diseases without replacing their judgment.
- Challenges:
  - **Bias and Discrimination**: AI models can inherit biases from the data they are trained on, leading to unfair outcomes.
  - **Lack of Explainability**: Many AI models, particularly deep learning models, are often considered "black boxes," making it difficult to understand how they make decisions.
  - **Global and Cultural Differences**: Ethical standards can vary across cultures, making it challenging to create universally acceptable AI systems.
  - **Autonomy and Control**: Balancing AI's autonomy with appropriate human control in decision-making processes.

Promoting Ethical AI

- **Ethical Guidelines**: Many organizations and governments have developed frameworks and guidelines (e.g., AI Ethics Guidelines by the European Commission).
- **Ethical AI Research**: Increased focus on developing algorithms that are more transparent, fair, and accountable.
- **Regulation and Legislation**: Governments are introducing laws and regulations to ensure ethical AI practices (e.g., GDPR, AI Act in Europe).

Bias and Fairness

- **Bias:**
  - Occurs when AI systems produce unfair, prejudiced, or inaccurate outcomes due to skewed training data, flawed algorithms, or human biases.
  - **Example**: A hiring algorithm favoring one gender or race due to biased training data.
- **Types of Bias:**
  - **Data Bias**: Bias in training data that reflects historical inequalities (e.g., gender or racial bias).
  - **Algorithmic Bias**: Bias introduced by the model's design or learning process.
  - **Sampling Bias**: Bias from using unrepresentative datasets that don't cover all groups adequately.

Bias and Fairness

- **Fairness:**
  - Ensuring that AI decisions do not favor one group over others and treat all individuals equally.
  - **Example**: A credit scoring model that equally assesses individuals from all demographics.
- **Approaches to Fairness:**
  - **Equal Opportunity**: Ensuring equal chances for all groups.
  - **Demographic Parity**: Ensuring outcomes are distributed equally across groups.
  - **Individual Fairness**: Similar individuals should be treated similarly by the model.

Bias and Fairness

- **Challenges:**
  - **Trade-offs**: Addressing fairness might reduce model accuracy or performance.
  - **Context Sensitivity**: Fairness definitions may vary across different applications and cultures.
- **Mitigation Techniques:**
  - **Bias Detection**: Identifying and quantifying bias in models.
  - **Re-sampling or Re-weighting**: Adjusting the training data to ensure diverse representation.
  - **Fairness Constraints**: Incorporating fairness objectives into the model's training process.

Security in AI

- Security in AI is crucial for building resilient systems that can operate safely, reliably, and ethically in diverse environments. Here are some points:
- **Model Robustness:**
  - AI systems should be resistant to adversarial attacks, where small, intentional perturbations to input data cause the model to make incorrect predictions.
  - **Example**: Small changes in an image that mislead an image classifier into making a wrong classification.
- **Data Privacy:**
  - Ensuring AI models respect user privacy by handling sensitive data securely and in compliance with regulations (e.g., GDPR).
  - **Example**: AI systems should not store or misuse personal information without consent.

Security in AI

- **Adversarial Machine Learning:**
- AI models can be vulnerable to attacks where malicious actors deliberately manipulate input data to exploit model weaknesses.
- **Example:** Attacking a self-driving car's AI by manipulating road signs to cause it to misinterpret them.
- **Model Theft and Reverse Engineering:**
- Protecting proprietary AI models from being stolen or reverse-engineered by malicious actors who could exploit them or steal intellectual property.
- **Example:** Preventing unauthorized access to a company’s trained model via model extraction attacks.

Security in AI

- **Security of Deployment Infrastructure:**
  - Ensuring that the infrastructure hosting AI models (e.g., cloud platforms, APIs) is secure from cyberattacks and unauthorized access.
  - **Example**: Protecting AI APIs from denial-of-service (DoS) attacks or data breaches.
- **Explainability and Trust:**
  - Security also involves making AI decisions interpretable and transparent to ensure users trust and understand the system, especially in critical areas like healthcare or finance.
- **Ethical and Legal Compliance:**
  - Ensuring AI systems follow legal and ethical standards to prevent misuse or harmful applications.
  - **Example**: Using AI in surveillance must comply with privacy laws and prevent misuse.

Future of AI

- **Increased Autonomy and Automation:**
  - AI will continue to enhance automation across industries, from manufacturing to healthcare, enabling more efficient processes and reducing human intervention in repetitive tasks.
  - **Example**: Fully autonomous vehicles, smart factories, and automated medical diagnoses.
- **AI-Powered Personalization:**
  - AI will become more adept at delivering personalized experiences in areas like entertainment, shopping, education, and healthcare, tailoring services to individual needs and preferences.
  - **Example**: AI-driven recommendations in streaming platforms or personalized learning paths in education.

Future of AI

- **Ethical and Regulatory Development:**
  - As AI becomes more integrated into society, there will be increased focus on ensuring ethical practices, addressing biases, and creating regulations to ensure AI is safe, fair, and accountable.
  - **Example**: Governments and organizations developing global AI governance frameworks and laws.
- **Human-AI Collaboration:**
  - Rather than replacing humans, AI will augment human capabilities, enabling people to make better decisions and be more productive across a wide range of domains.
  - **Example**: AI assistants helping doctors with diagnoses or enhancing creative processes in design and art.

Future of AI

- **Advancements in General AI (AGI):**
  - Research in artificial general intelligence (AGI) may eventually lead to machines capable of performing any intellectual task that a human can, opening up new possibilities for problem-solving and innovation.
  - **Example**: AI systems that can autonomously learn and adapt to new tasks without being specifically trained for each one.