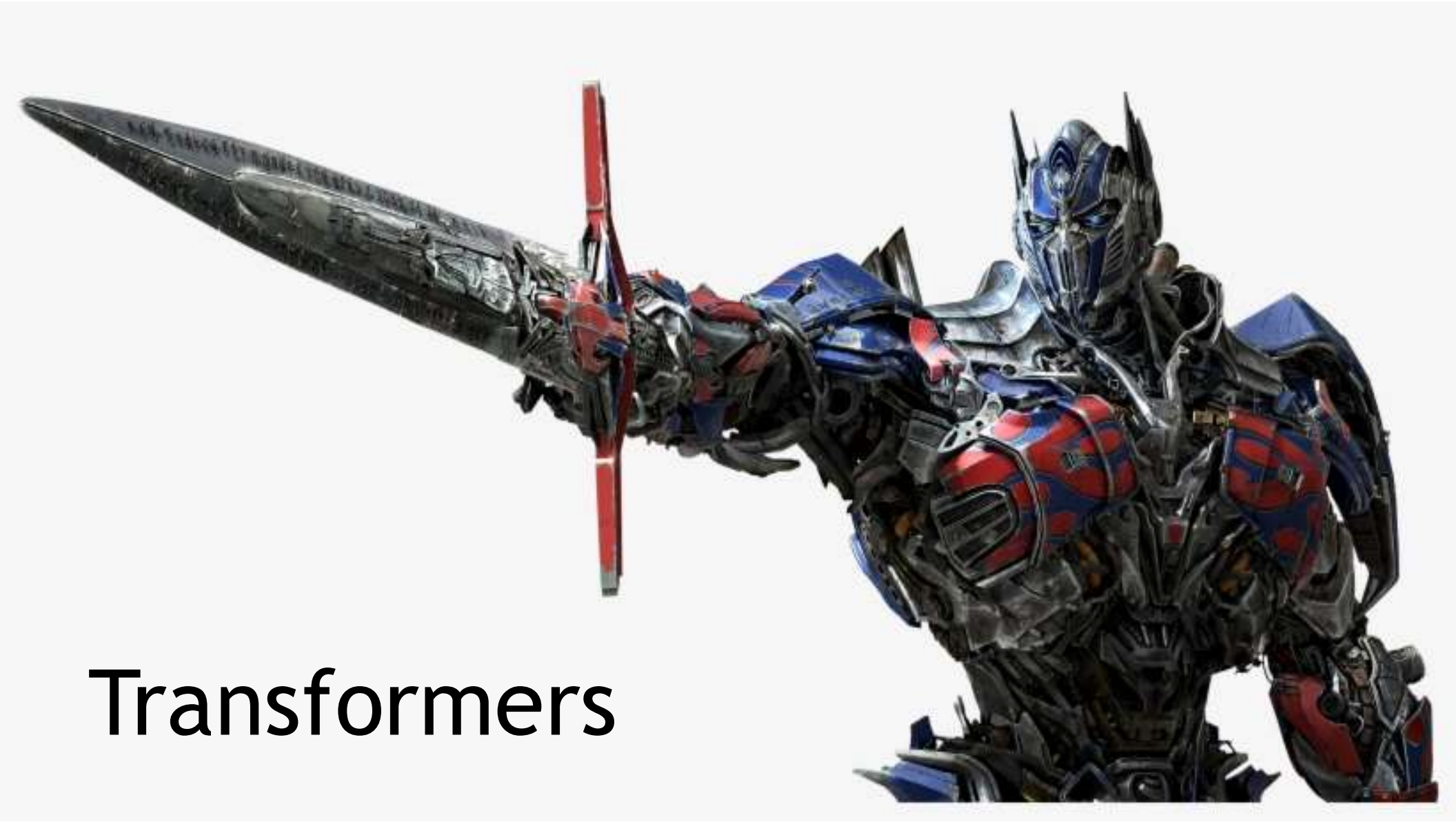
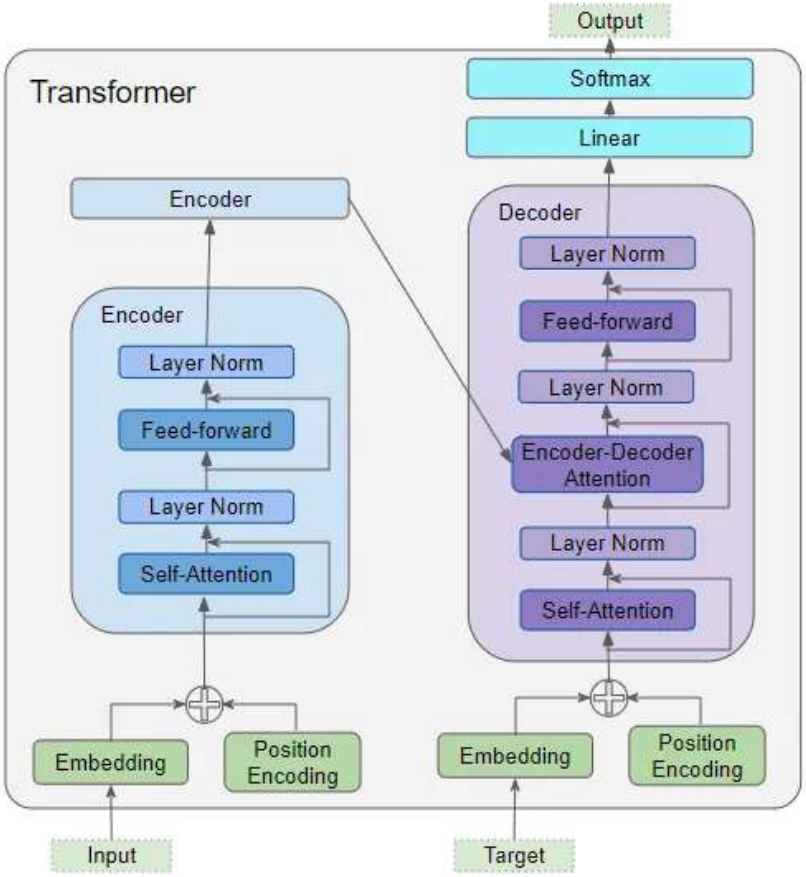
Generative AI


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

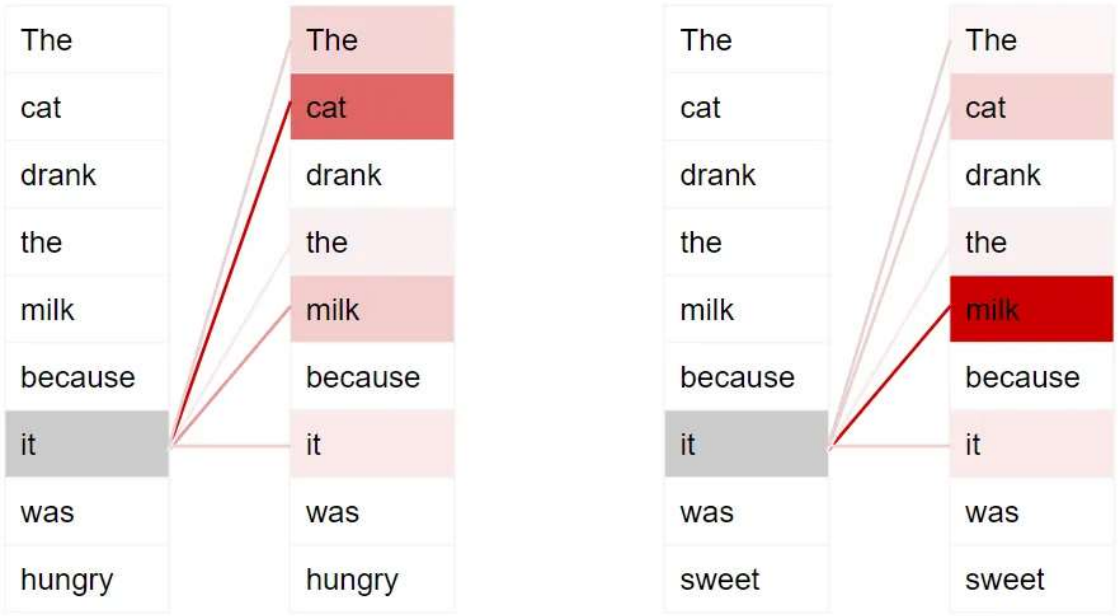
- The key to the Transformer's ground-breaking performance is its use of Attention.
- While processing a word, Attention enables the model to focus on other words in the input that are closely related to that word.
- The Transformer architecture uses self-attention by relating every word in the input 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

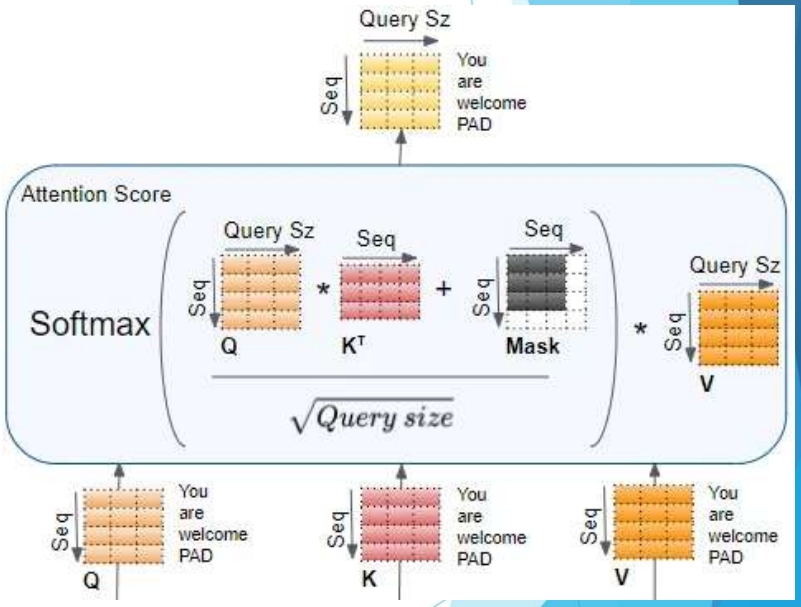

The
cat
drank
the
milk
because
it
was
hungry
The
cat
drank
the
milk
because
it
was
hungry
The
cat
drank
the
milk
because
it
was
sweet
The
cat
drank
the
milk
because
it
was
sweet

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

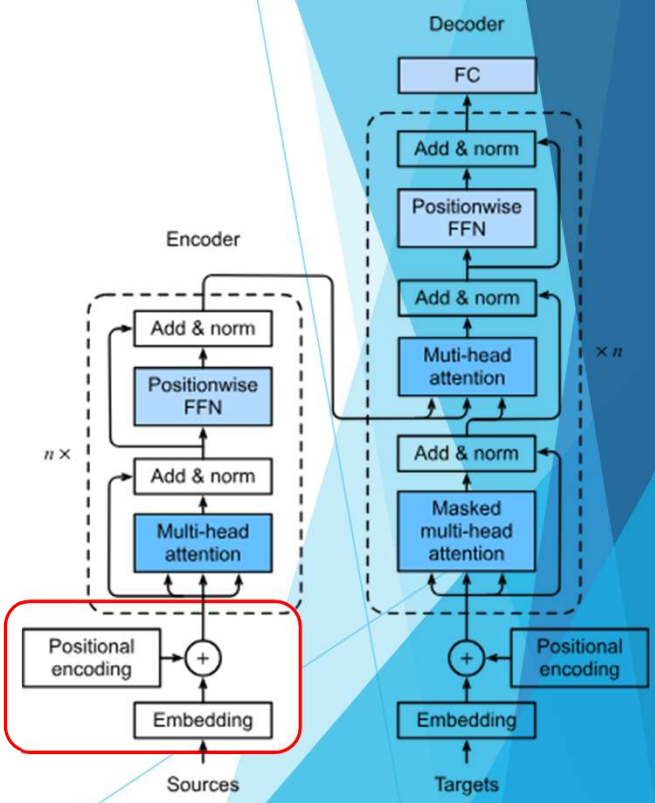
Similarity Score

- 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.



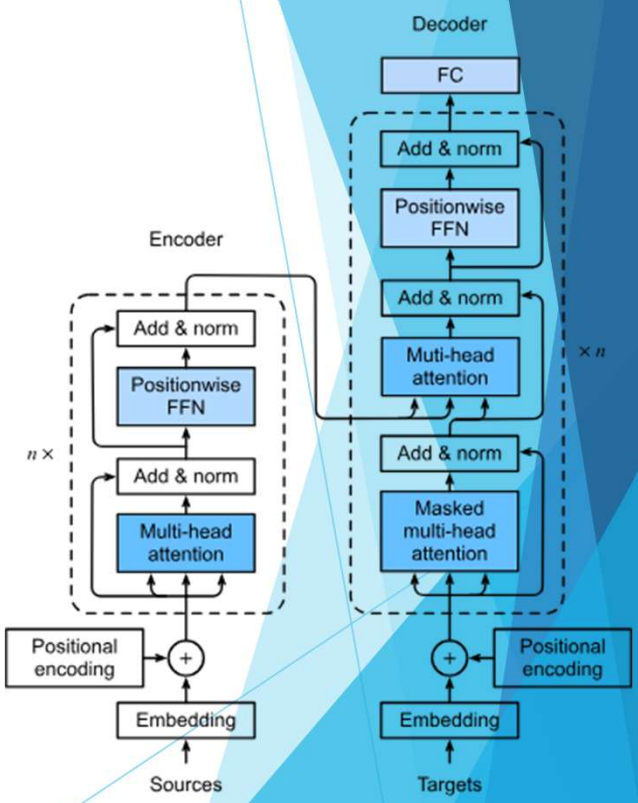
Functionality of the Encoder

- ***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.



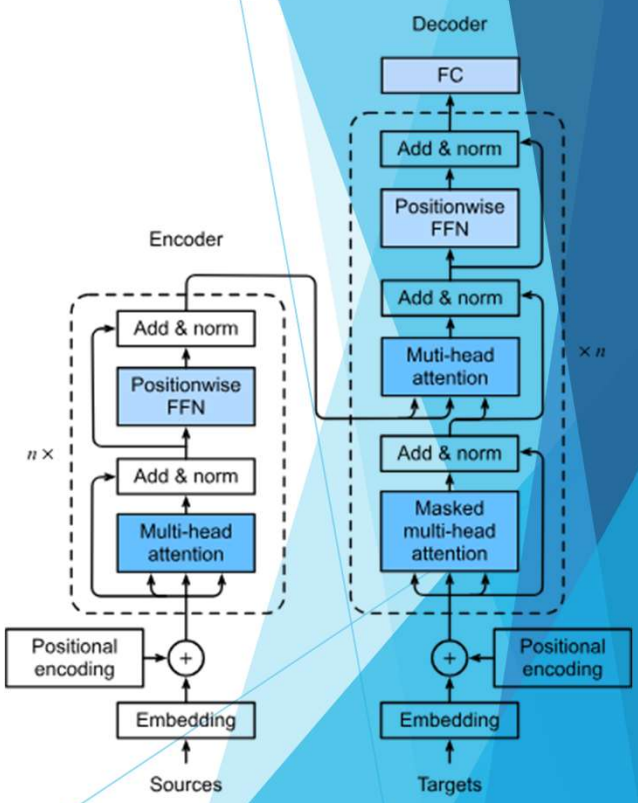
Functionality of the Encoder

- ***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.



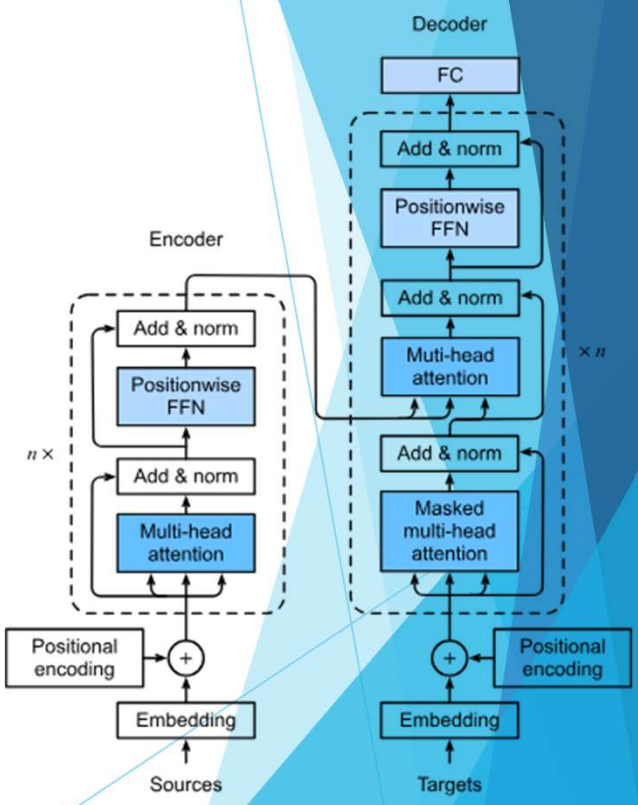
Functionality of the Encoder

- ***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).



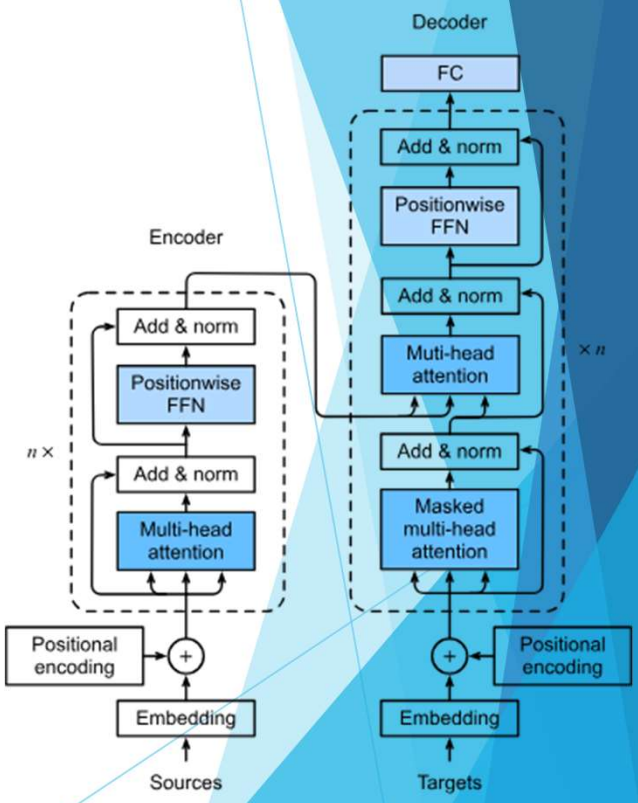
Functionality of the Encoder

- ***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.

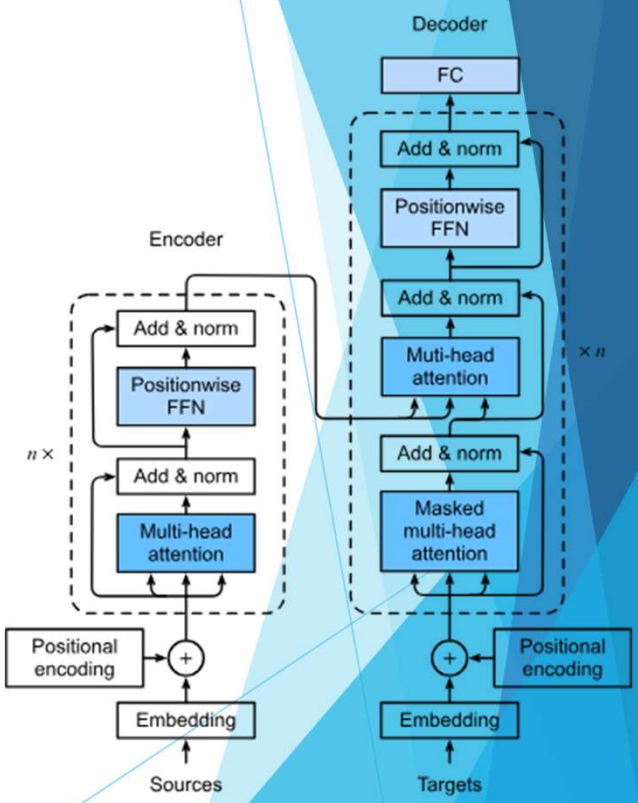


Functionality of Decoder

- ***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:

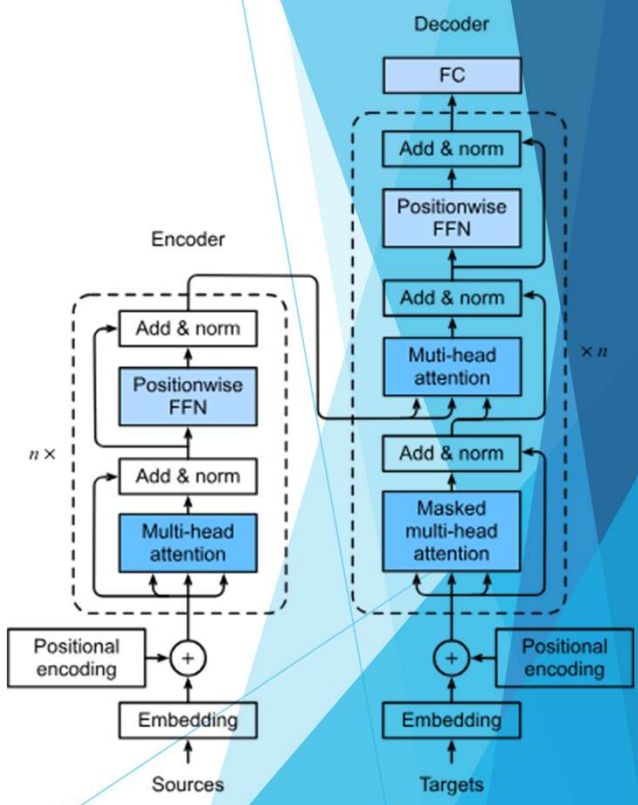
  "The" → [0.3, -0.2, 0.5, ...],

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



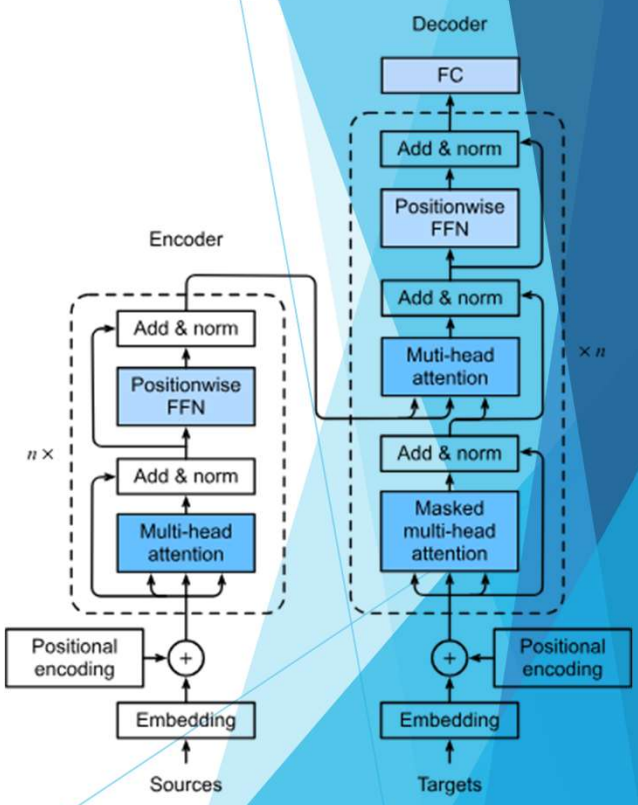
Functionality of Decoder

- ***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.



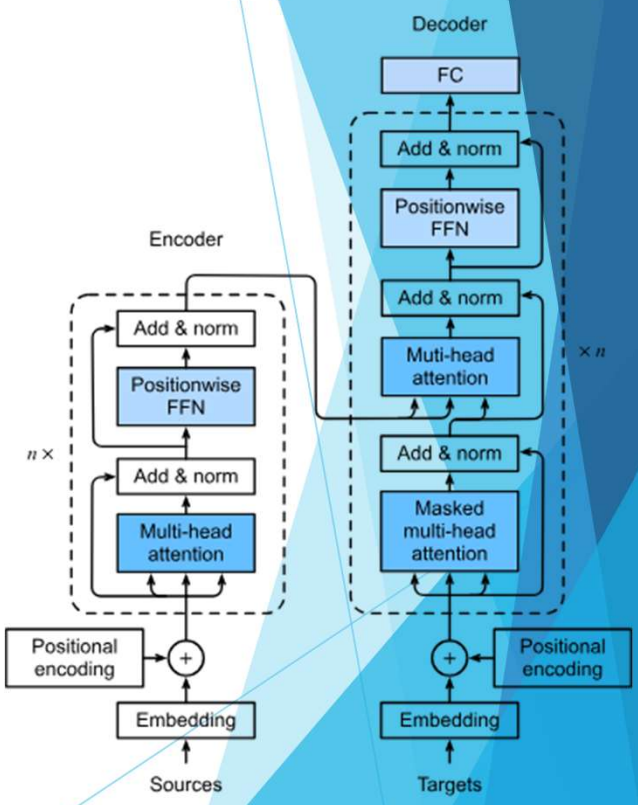
Functionality of Decoder

- **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.



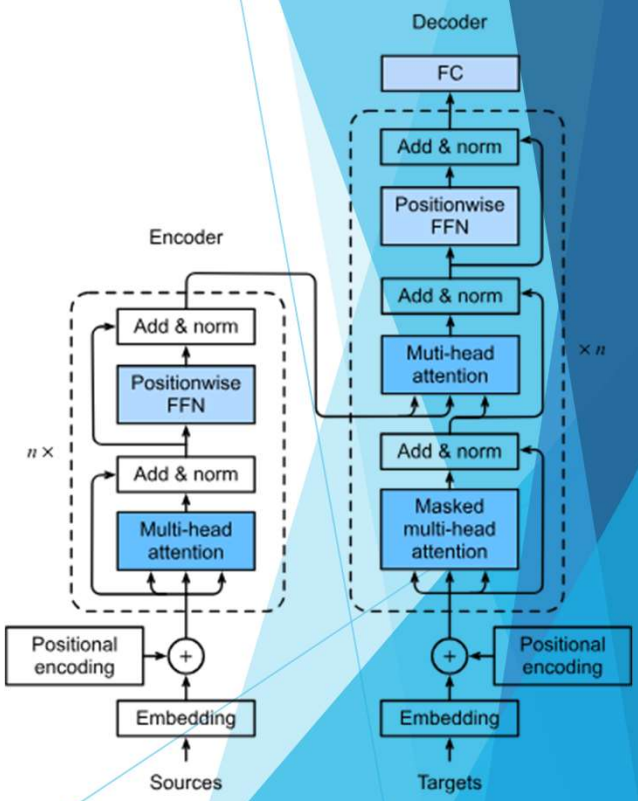
Functionality of Decoder

- **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).



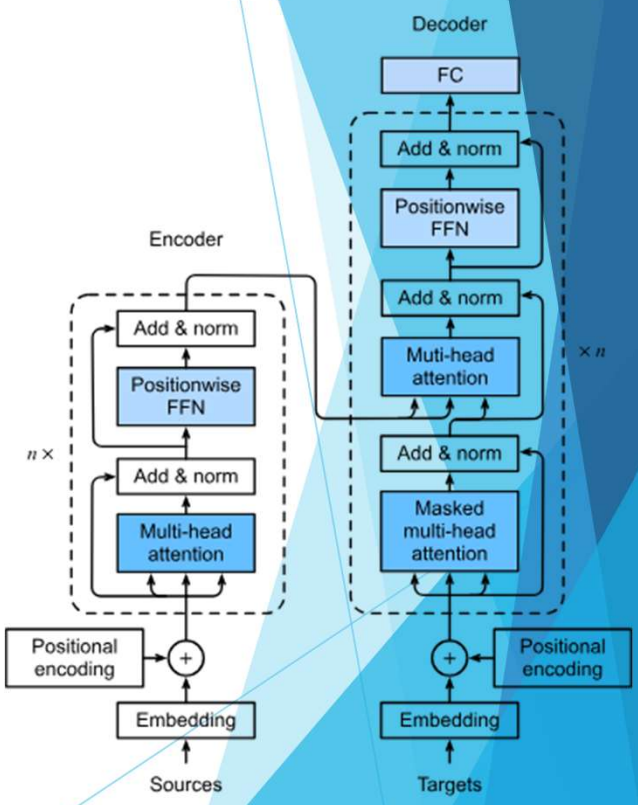
Functionality of Decoder

- **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.



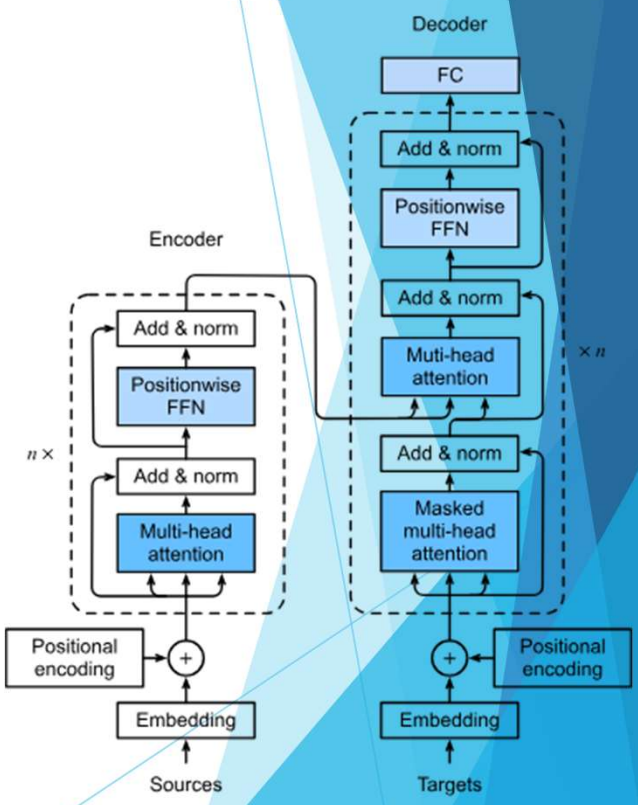
Functionality of Decoder

- **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

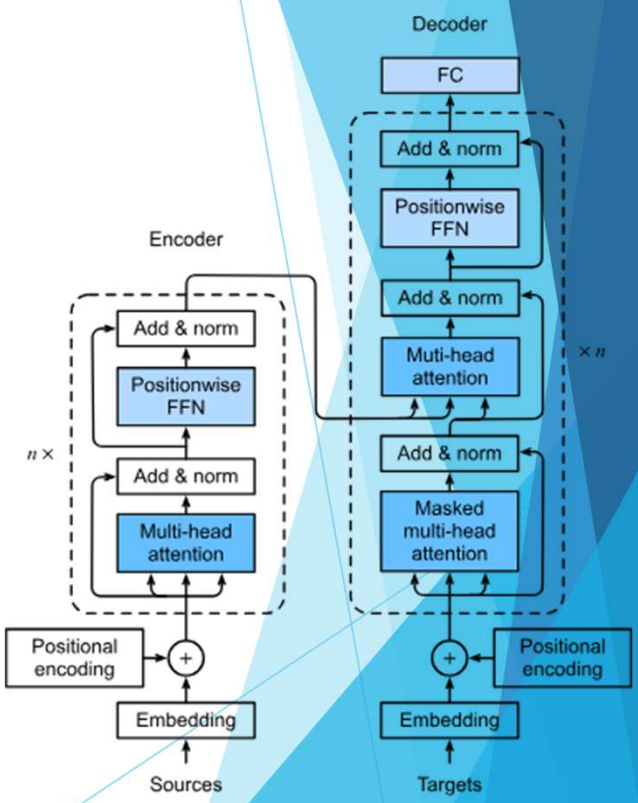
- **Masked Language Modeling (Next Token Prediction)**
- In text generation tasks, the decoder operates autoregressively by predicting the next token in the sequence based on the previous tokens and the encoder’s output. It uses the final decoder layer's output to predict the next token.
- Example: After 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.



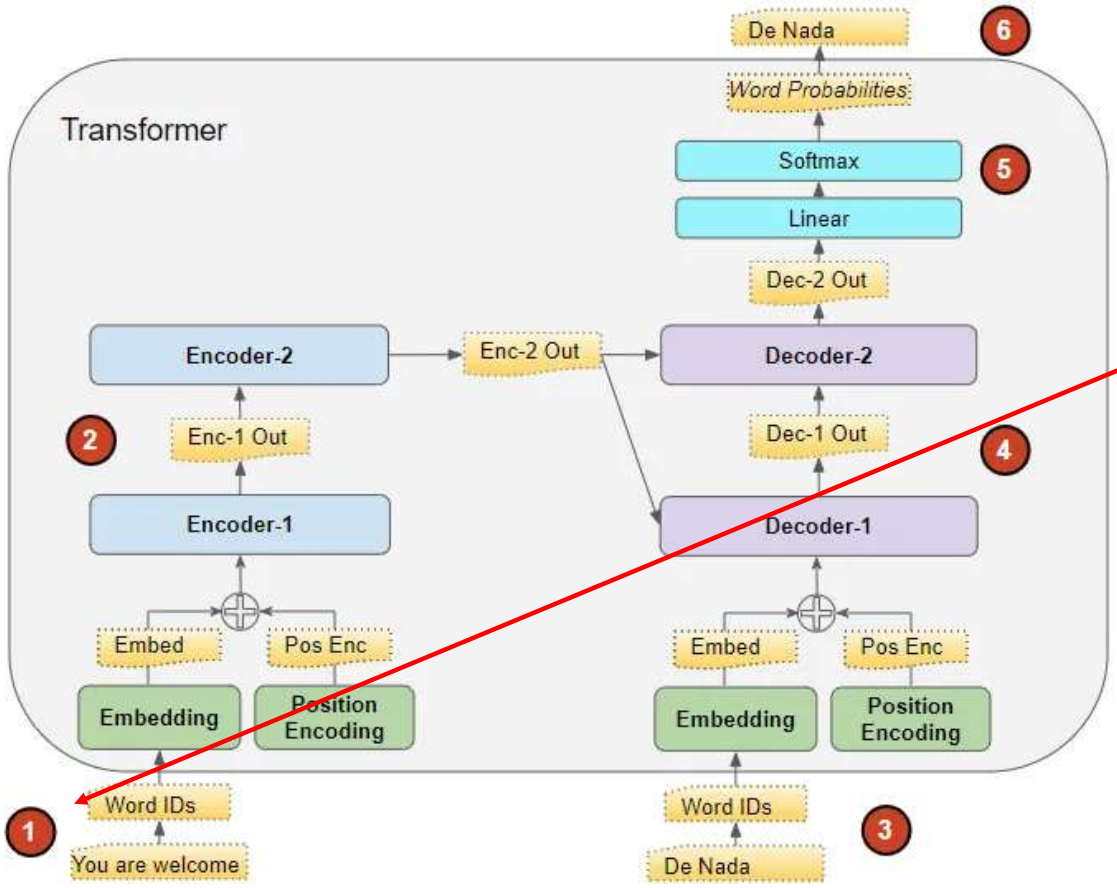
Functionality of Decoder

- 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.

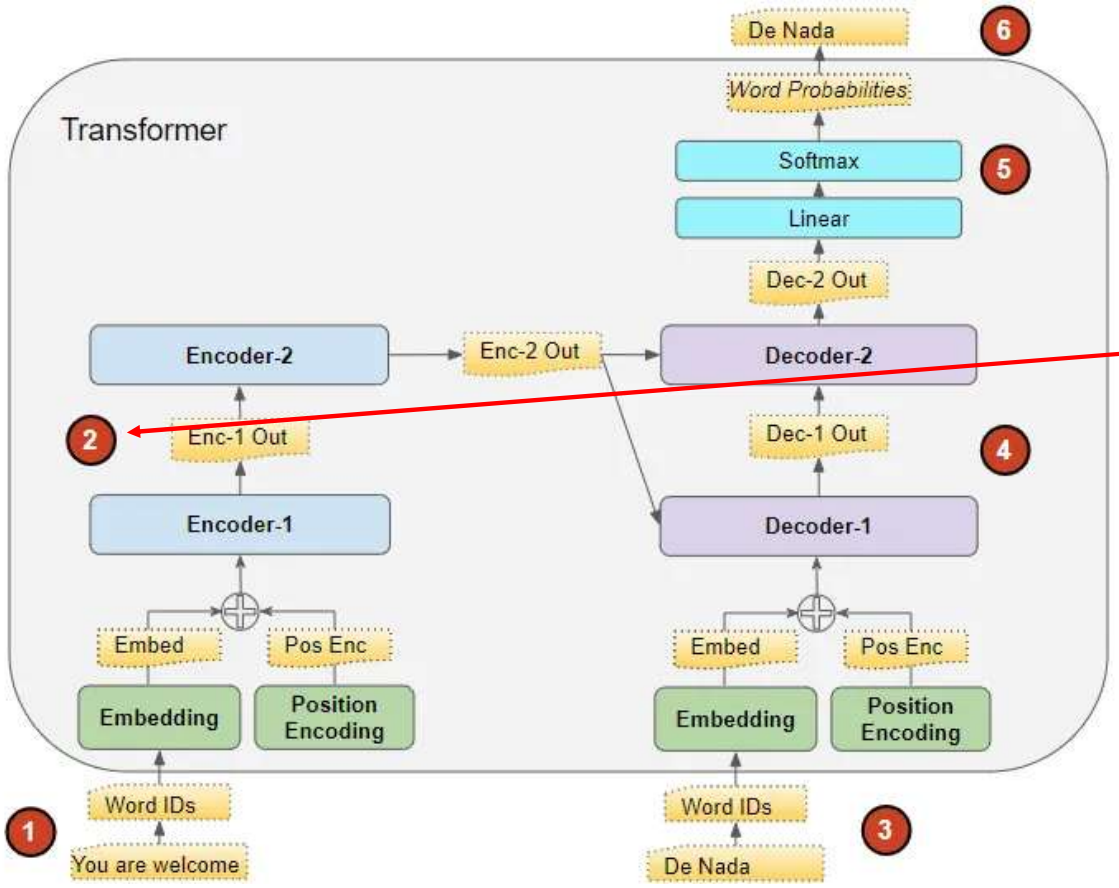


Working: Training



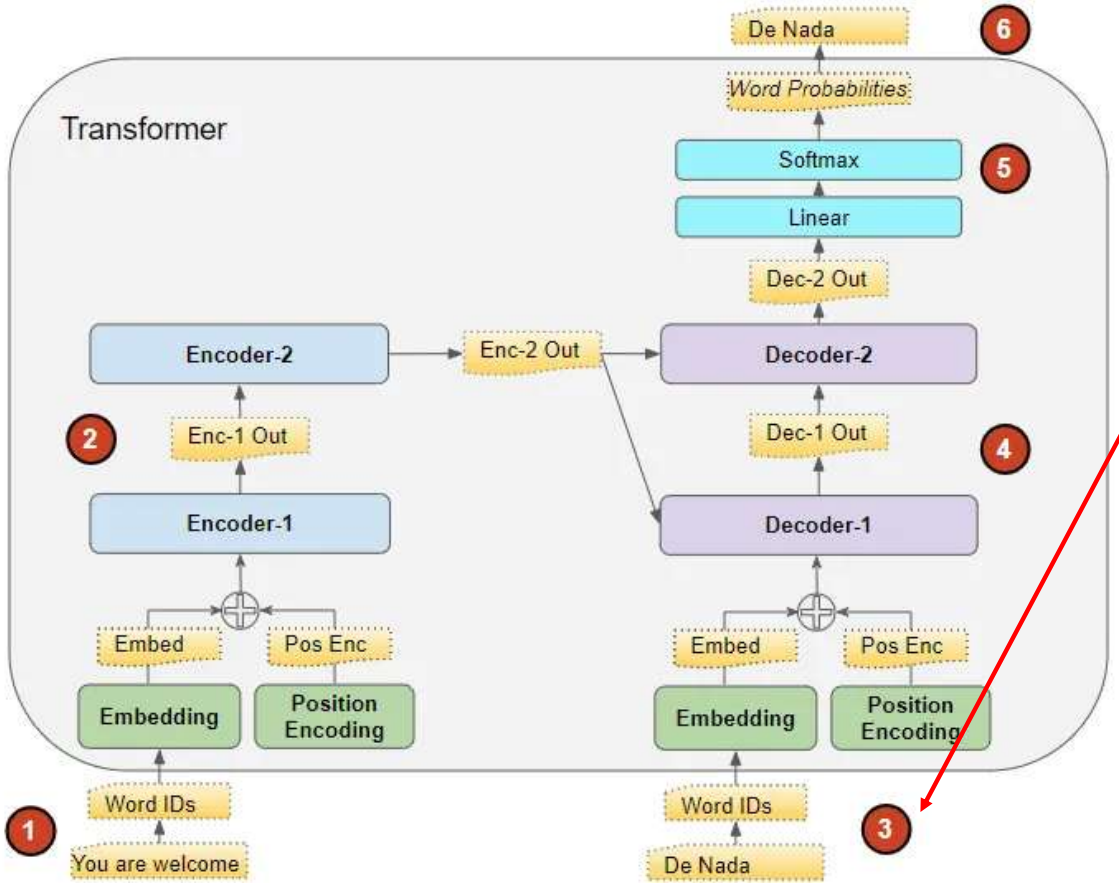
The input sequence is converted into Embeddings (with Position Encoding) and fed to the Encoder.

Working: Training



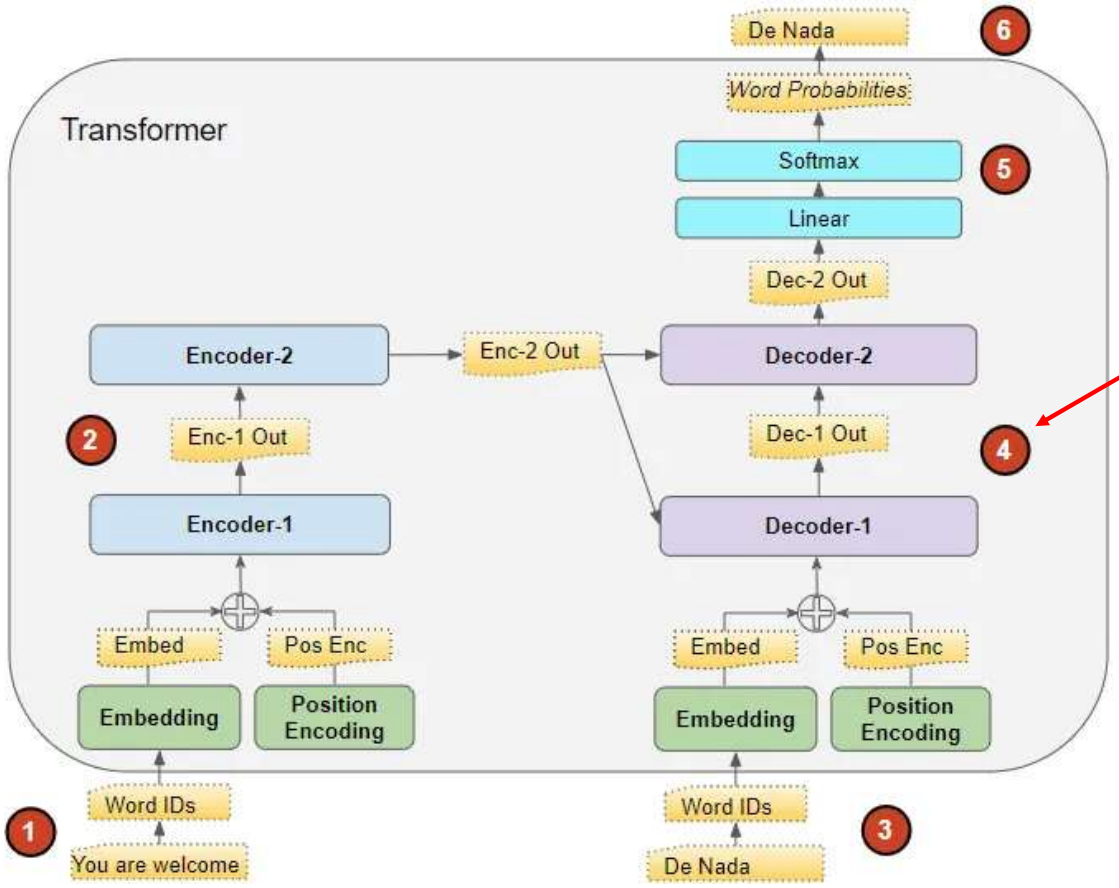
The stack of Encoders processes this and produces an encoded representation of the input sequence.

Working: Training



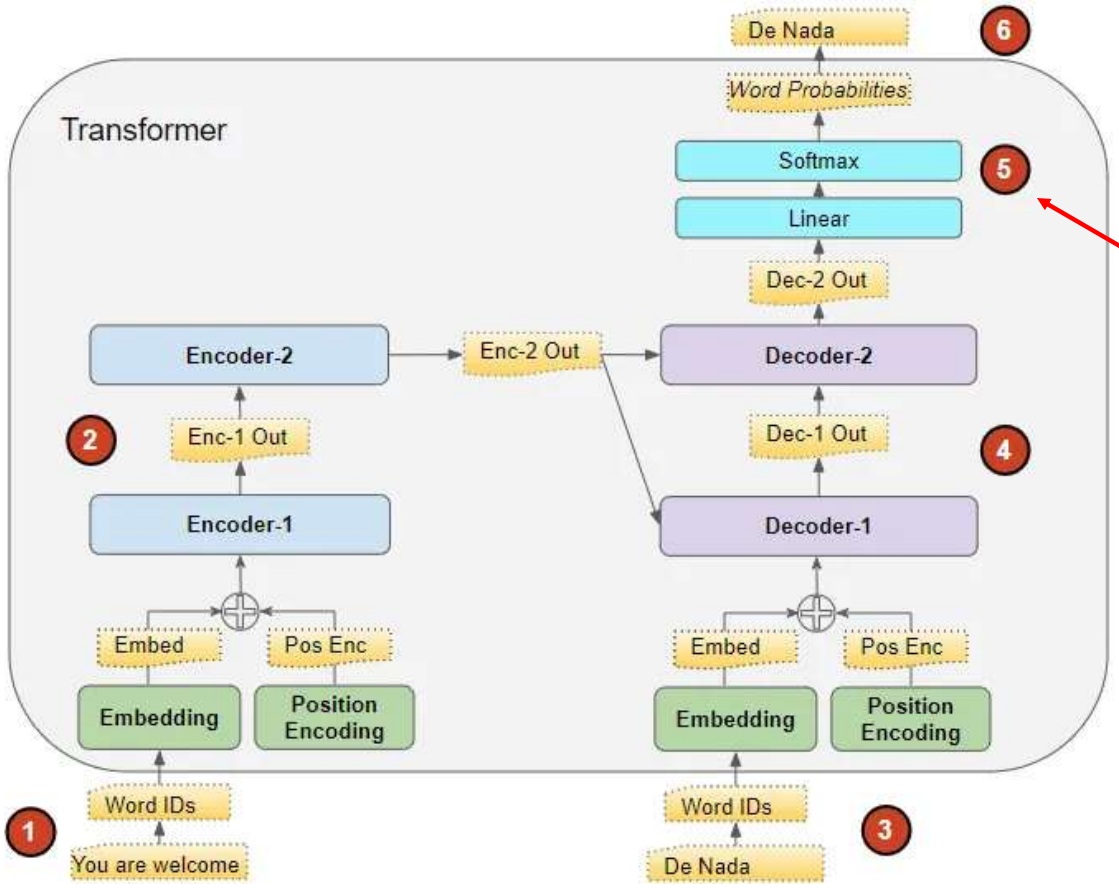
The target sequence is prepended with a start-of-sentence token, converted into Embeddings (with Position Encoding), and fed to the Decoder.

Working: Training



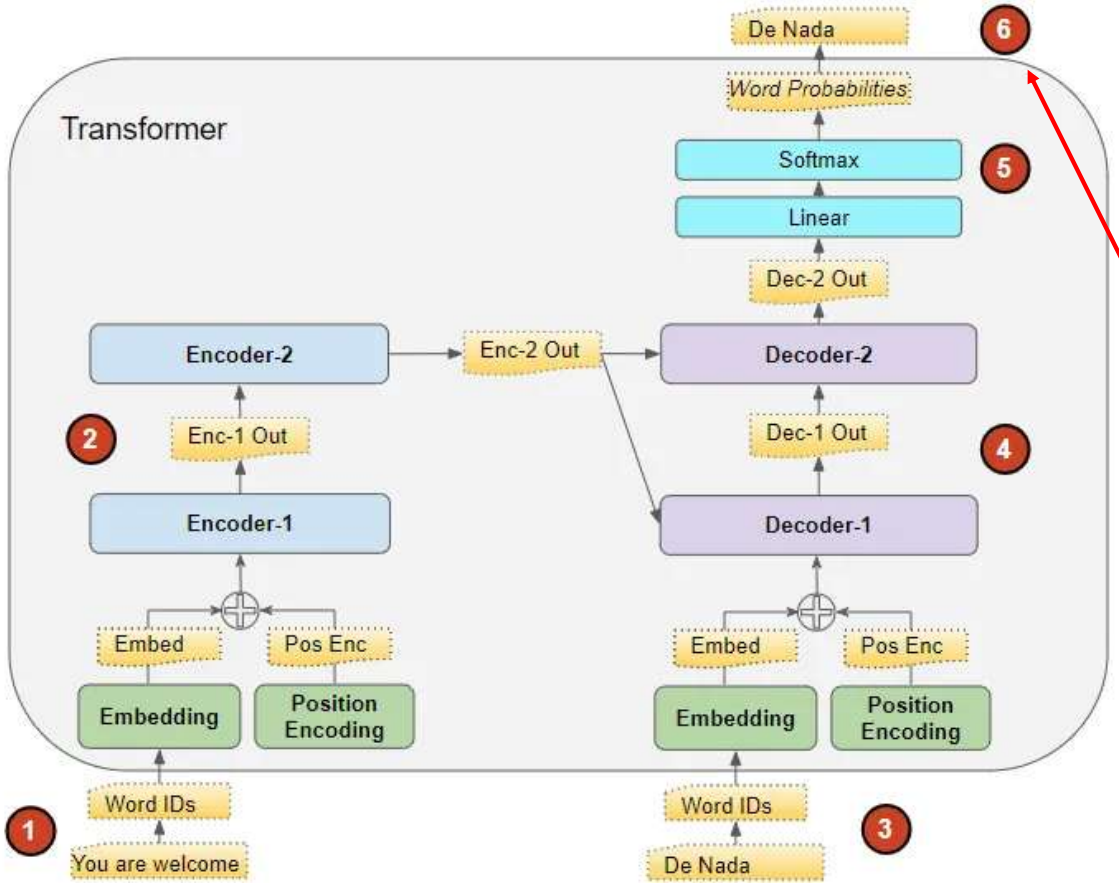
The stack of Decoders processes this along with the Encoder stack's encoded representation to produce an encoded representation of the target sequence.

Working: Training



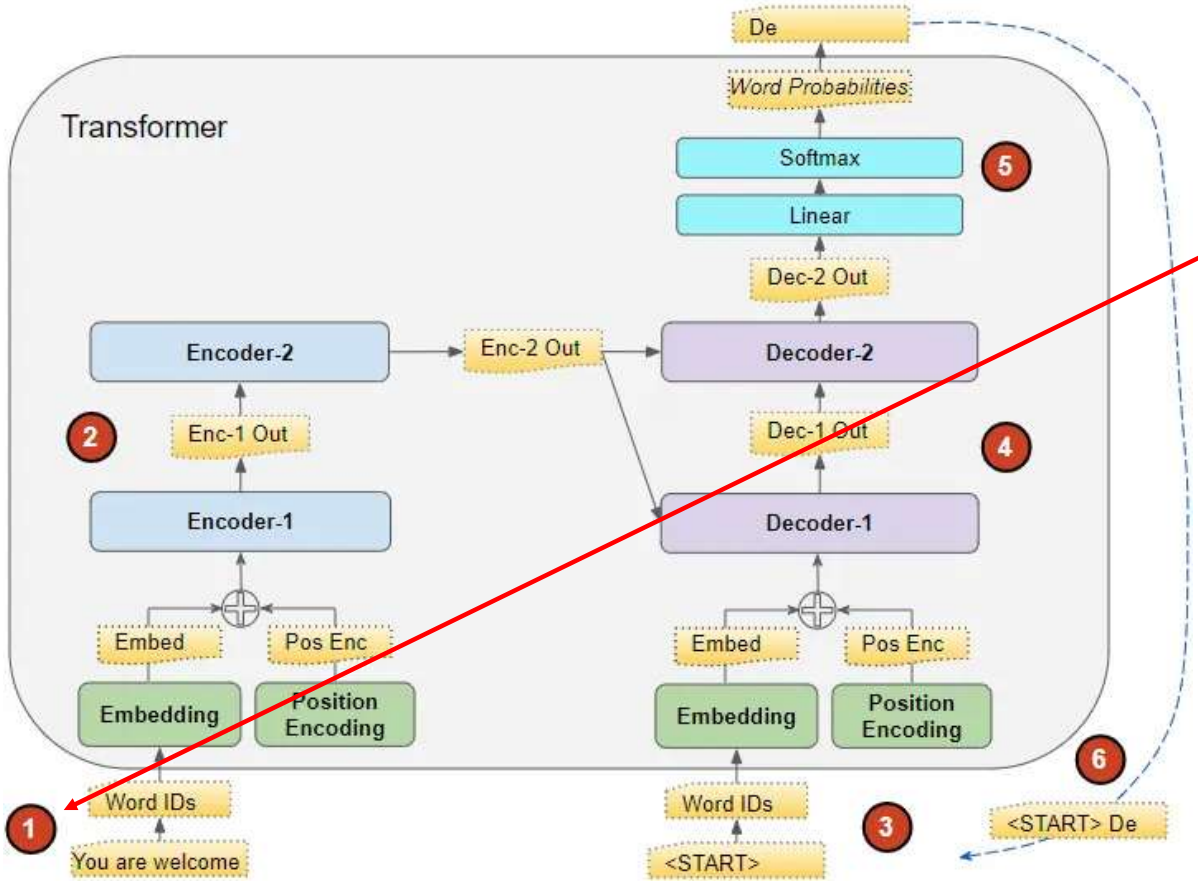
The Output layer converts it into word probabilities and the final output sequence.

Working: Training



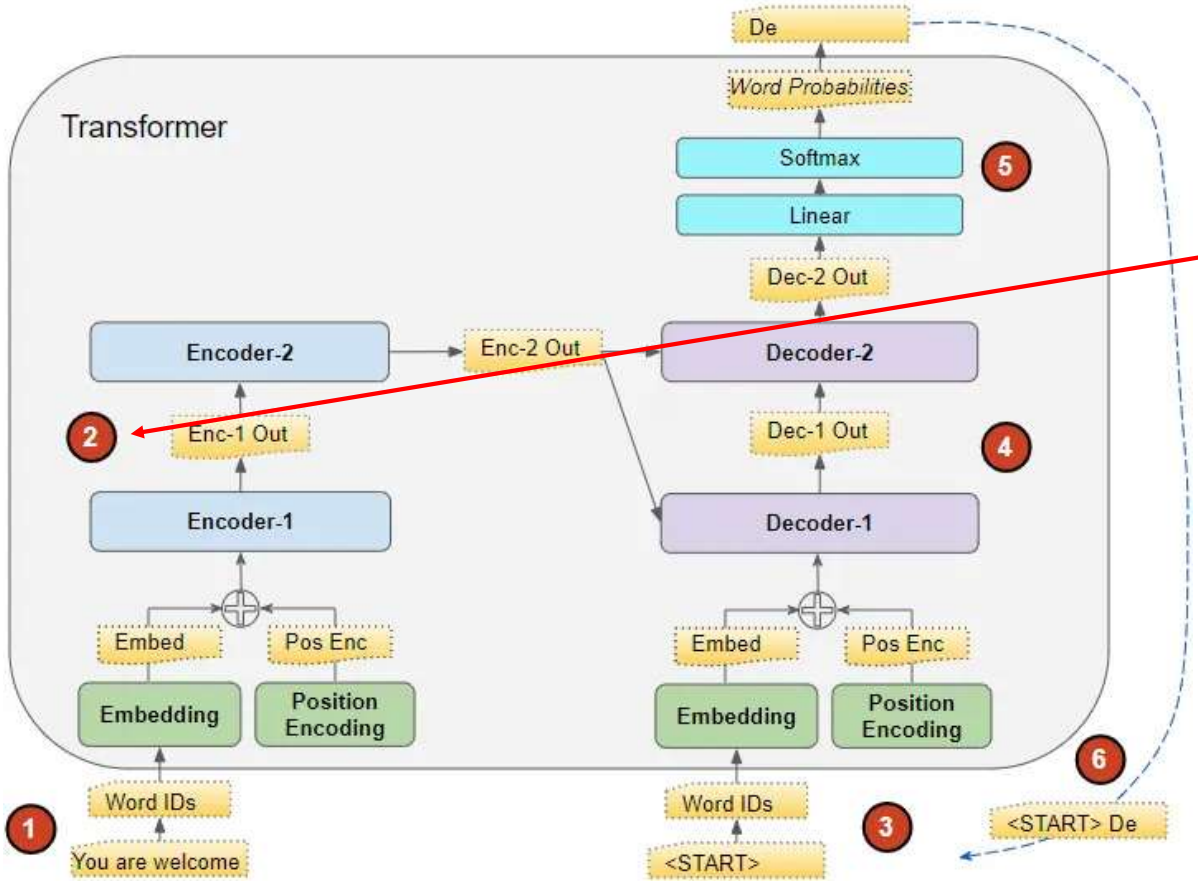
The Transformer's Loss function compares this output sequence with the target sequence from the training data. This loss is used to generate gradients to train the Transformer during back-propagation.

Working: Inference



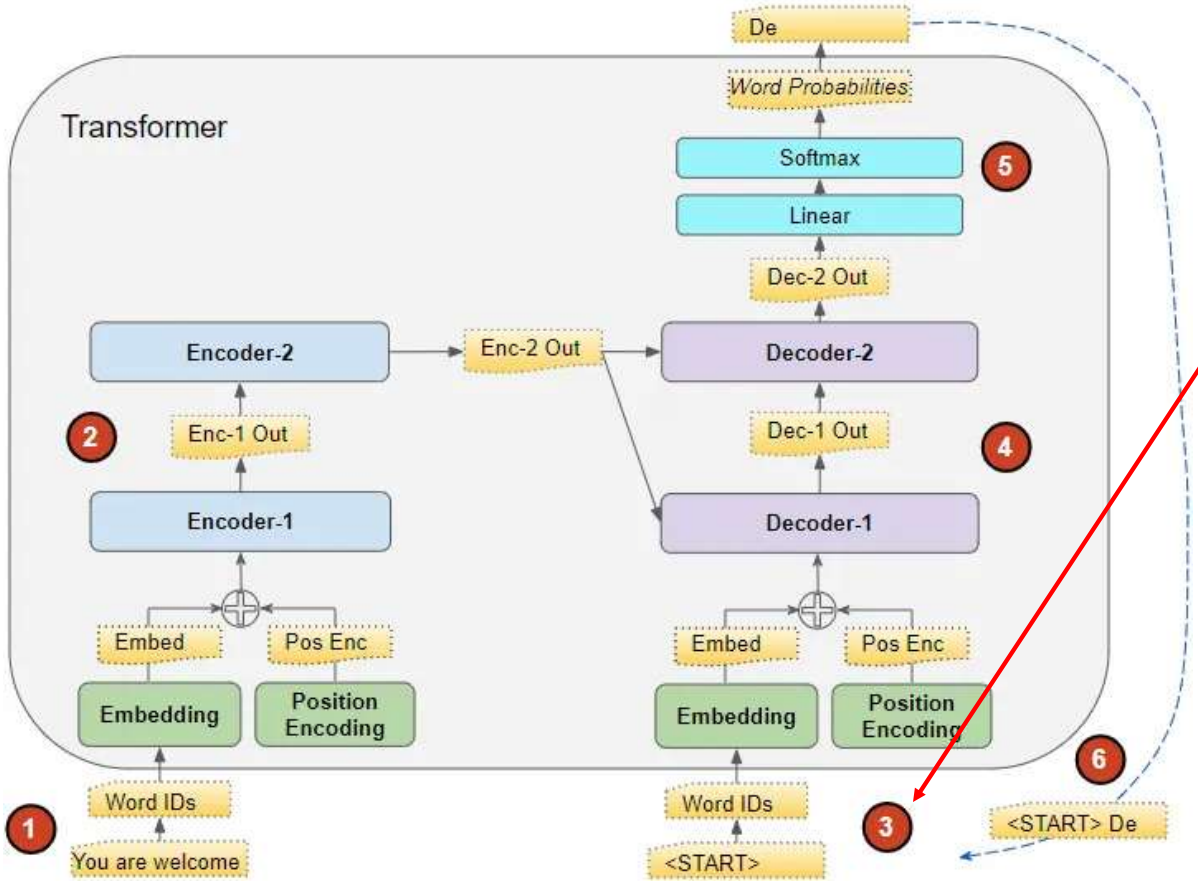
The input sequence is converted into Embeddings (with Position Encoding) and fed to the Encoder

Working: Inference



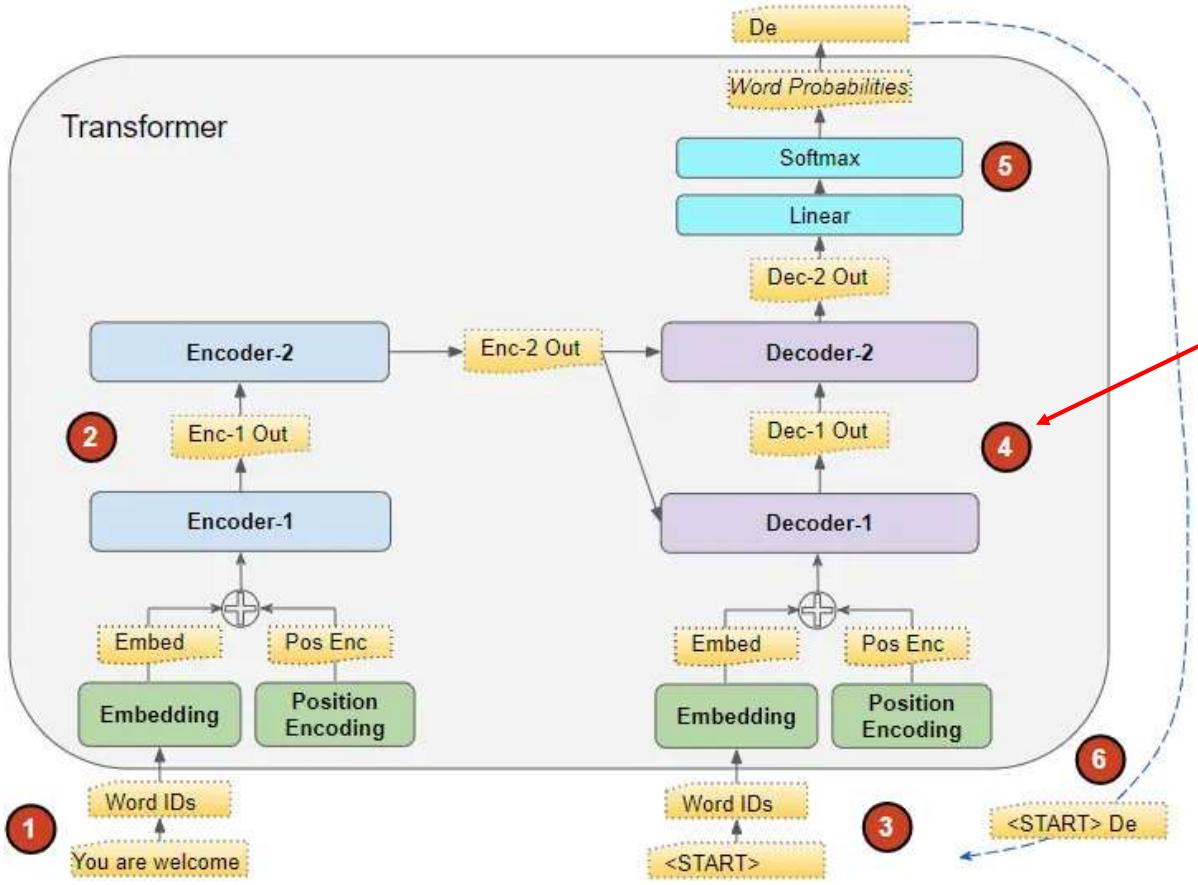
The stack of Encoders processes this and produces an encoded representation of the input sequence.

Working: Inference



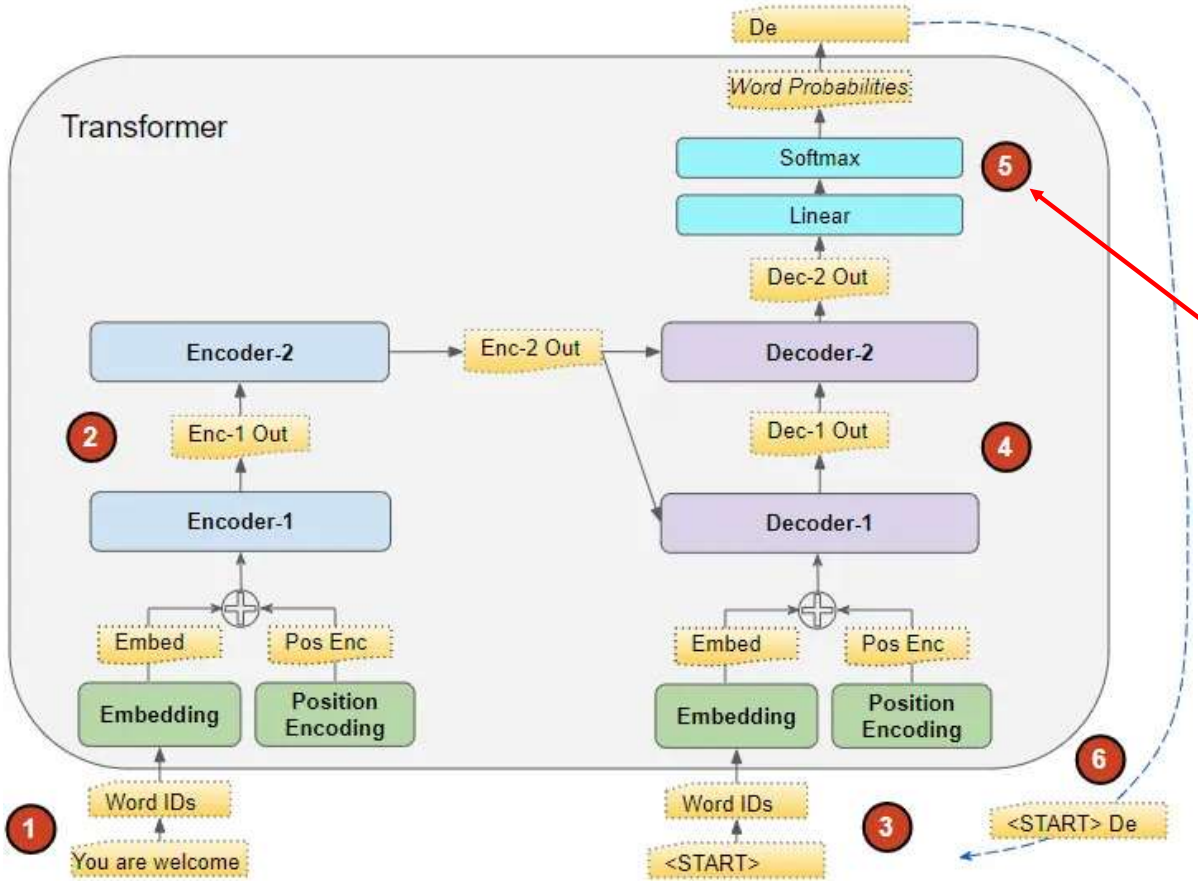
Instead of the target sequence, we use an empty sequence with only a start-of-sentence token. This is converted into Embeddings (with Position Encoding) and fed to the Decoder.

Working: Inference



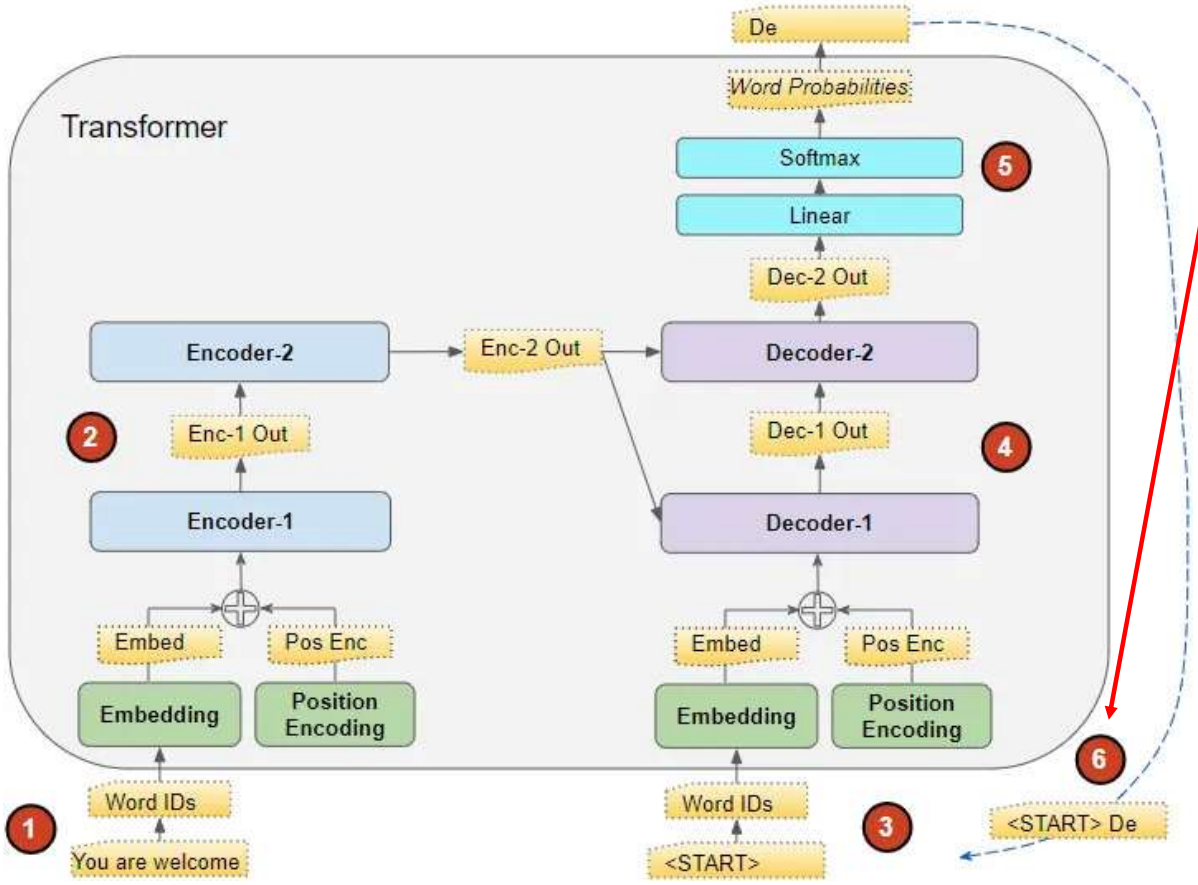
The stack of Decoders processes this along with the Encoder stack's encoded representation to produce an encoded representation of the target sequence.

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/

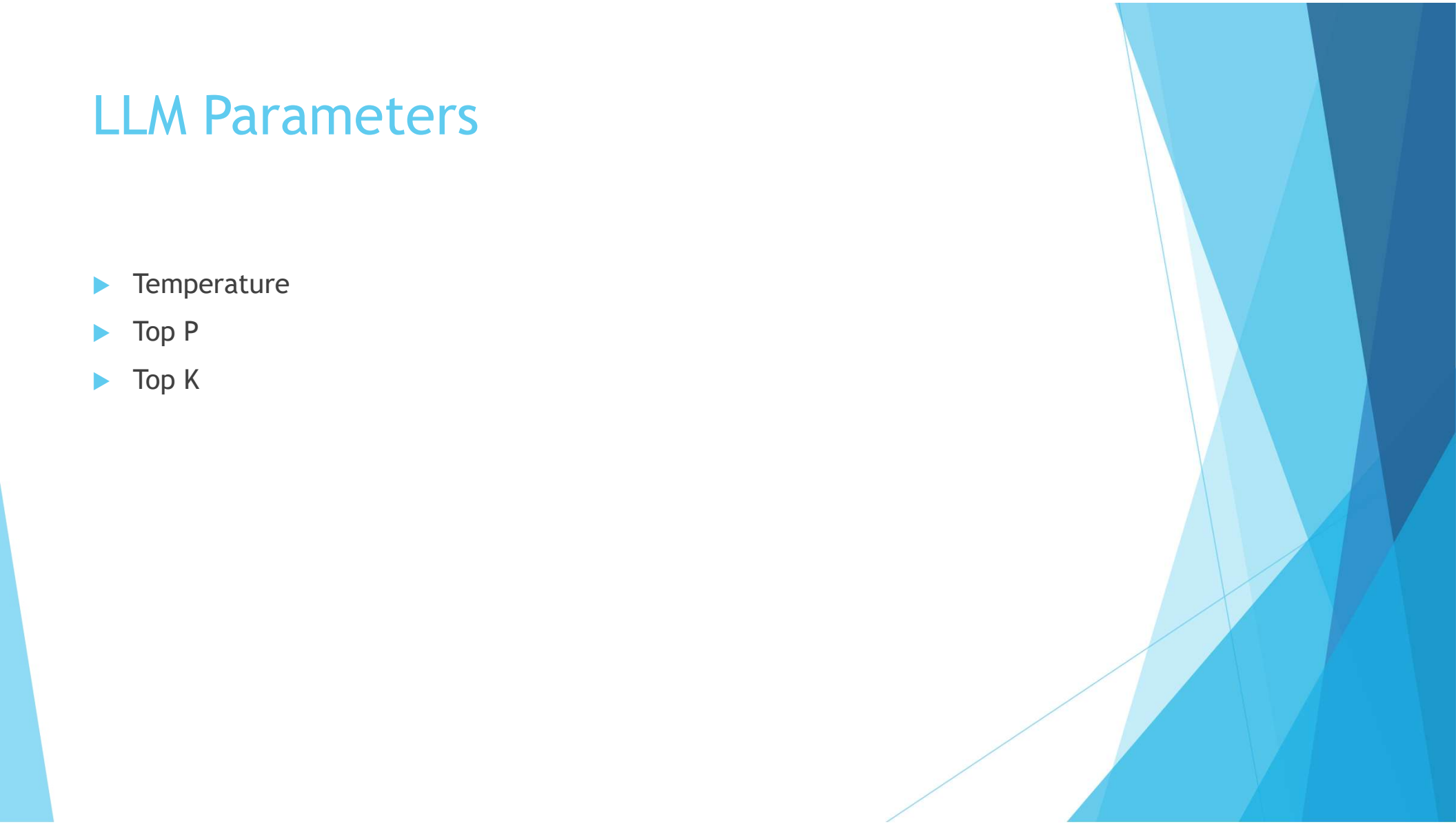
https://bbycroft.net/llm

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.

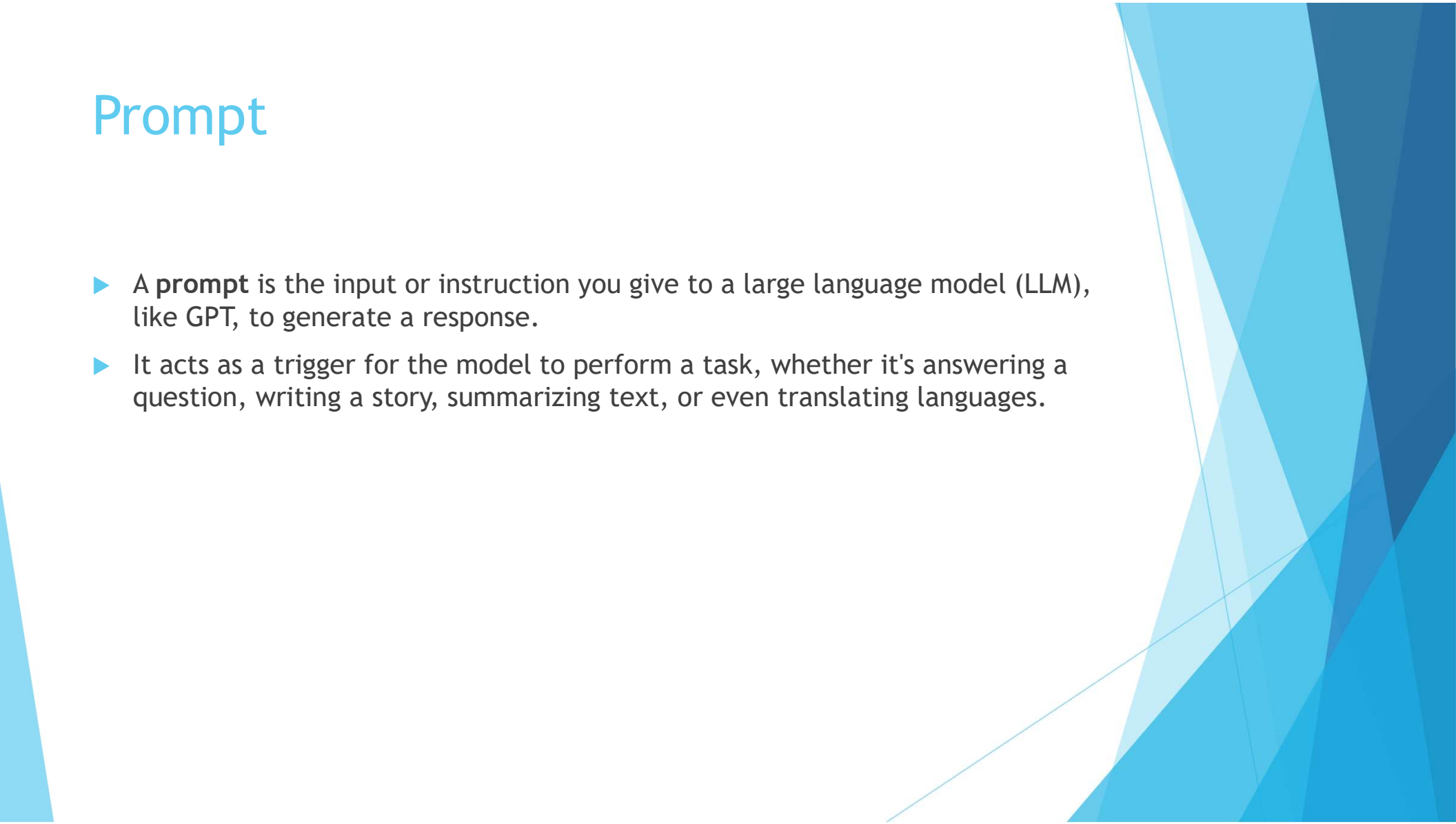
LLM Parameters

- Temperature
- Top P
- Top K



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

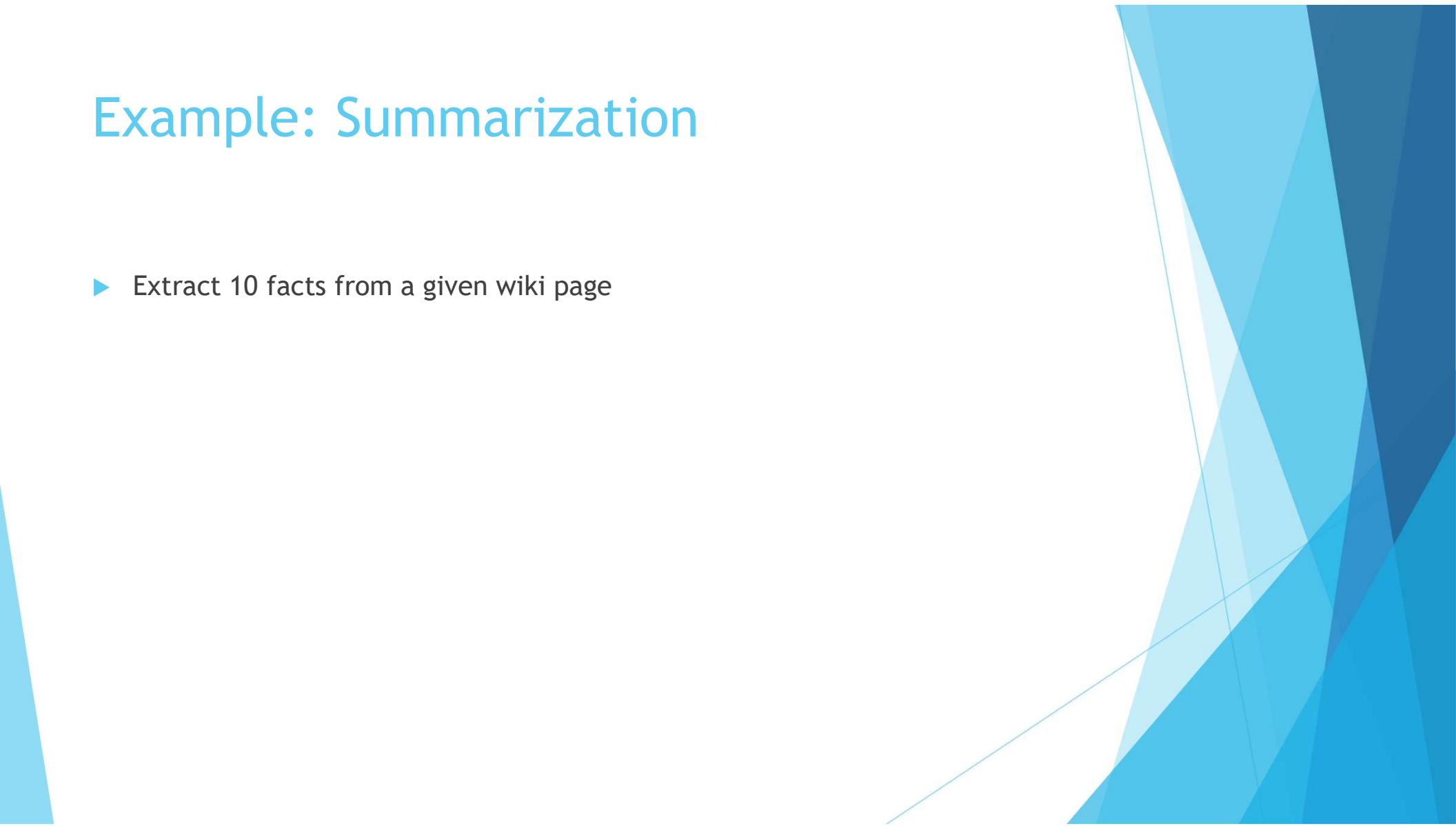
Technique	Description	Effect
Reverse Prompting	Model infers the instruction from the desired outcome.	Generates creative, indirect responses.
Contrastive Prompting	Compare two or more items and generate reasoning.	Produces balanced, reasoned evaluations.
Meta-Prompting	Model generates its own prompts 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.

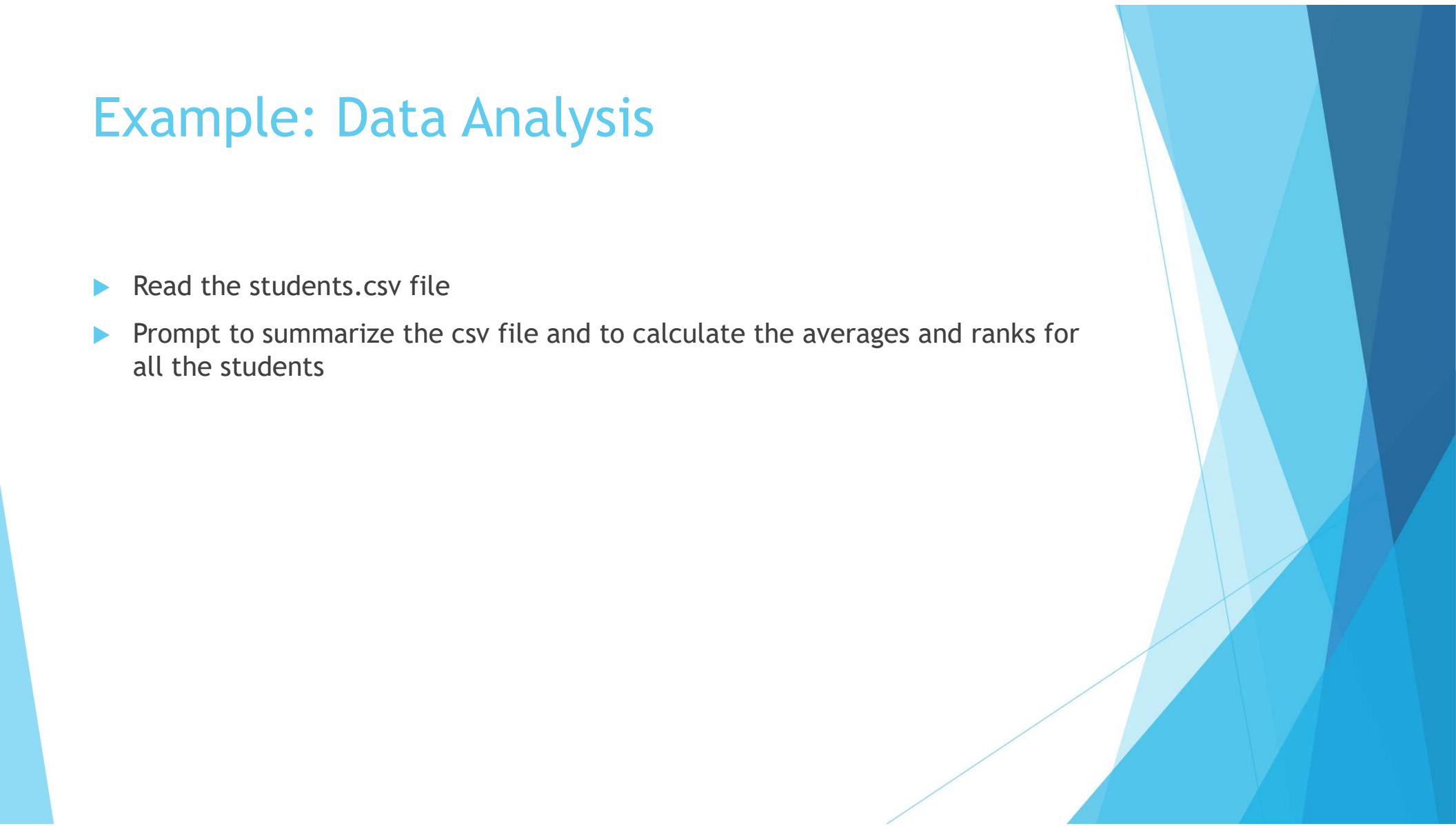
Example: Summarization

- Extract 10 facts from a given wiki page



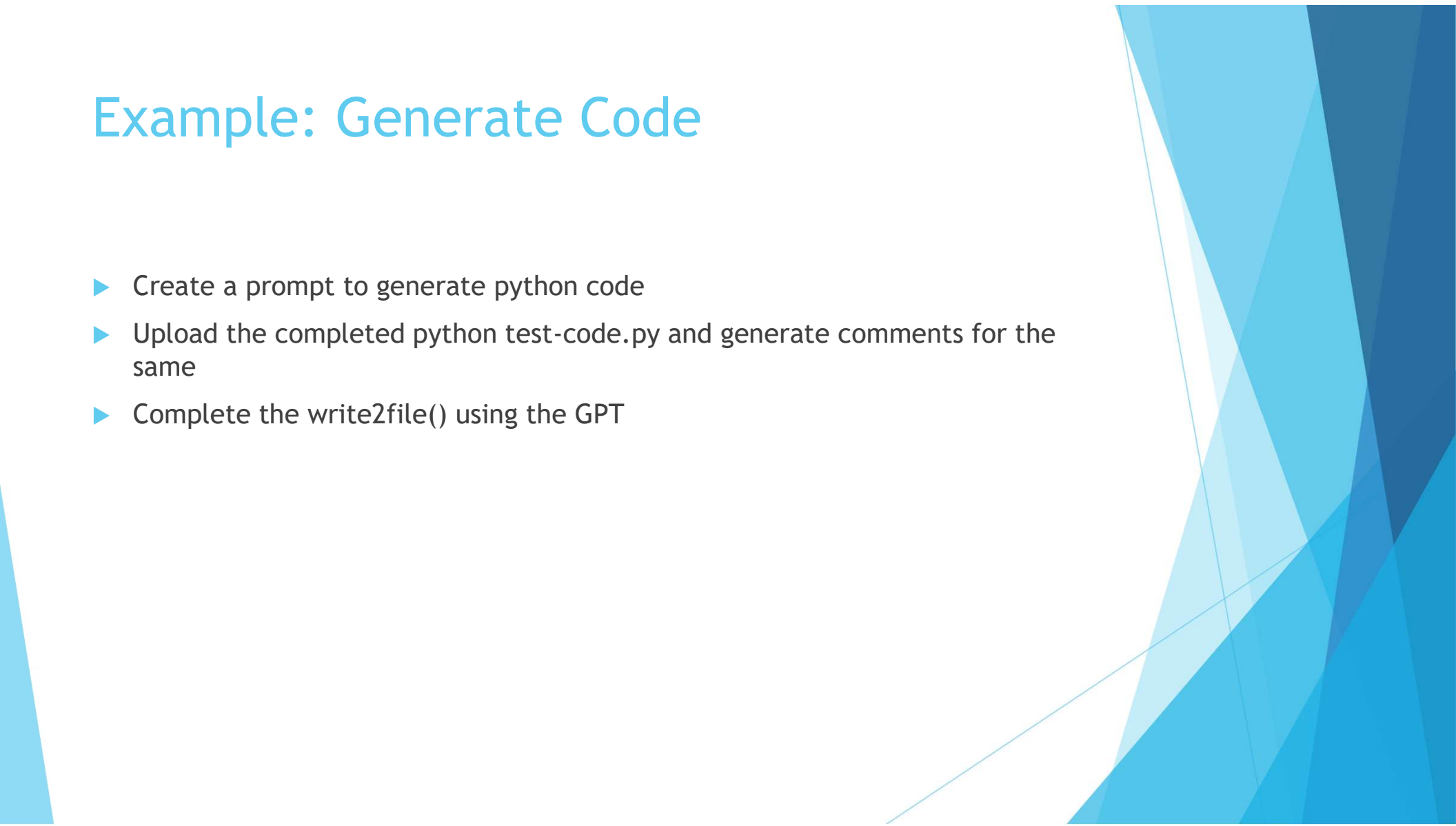
Example: Data Analysis

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



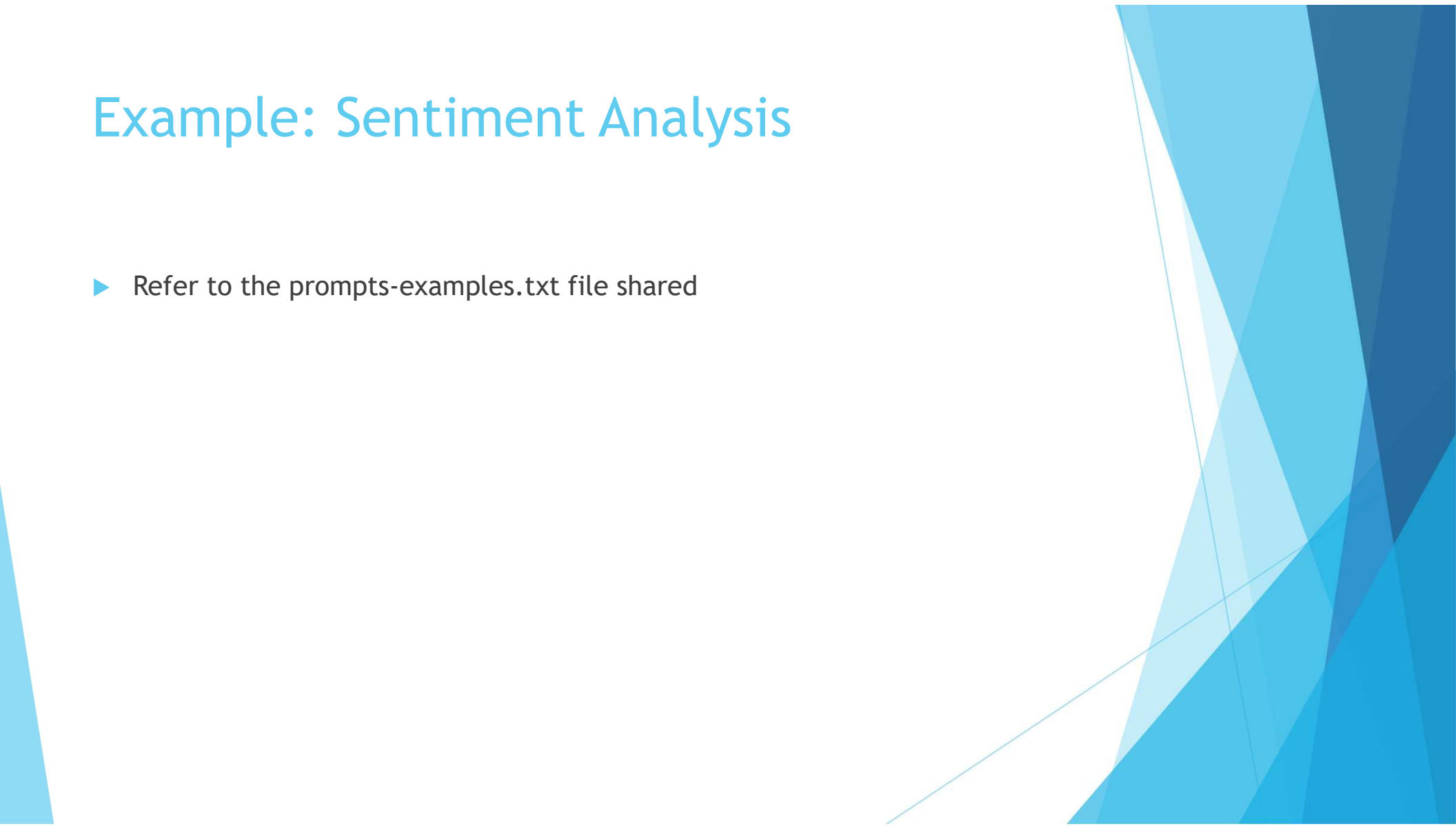
Example: Generate Code

- 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