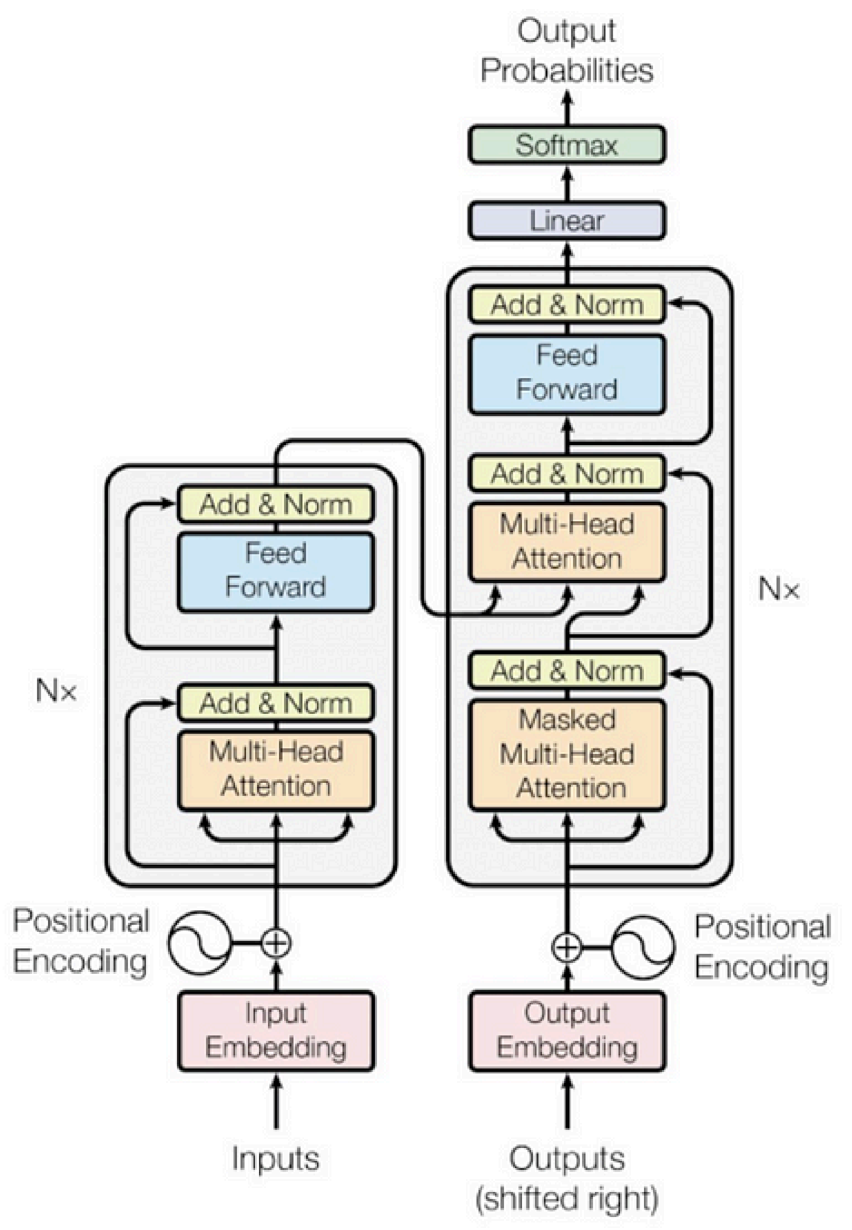
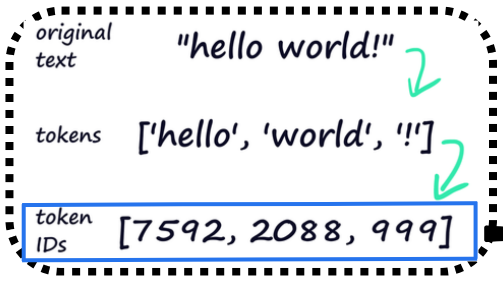
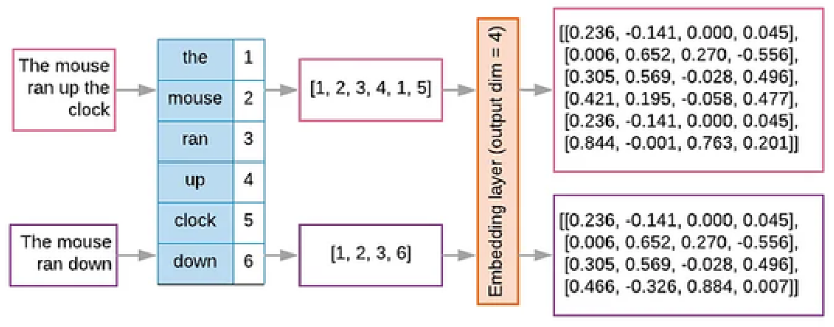
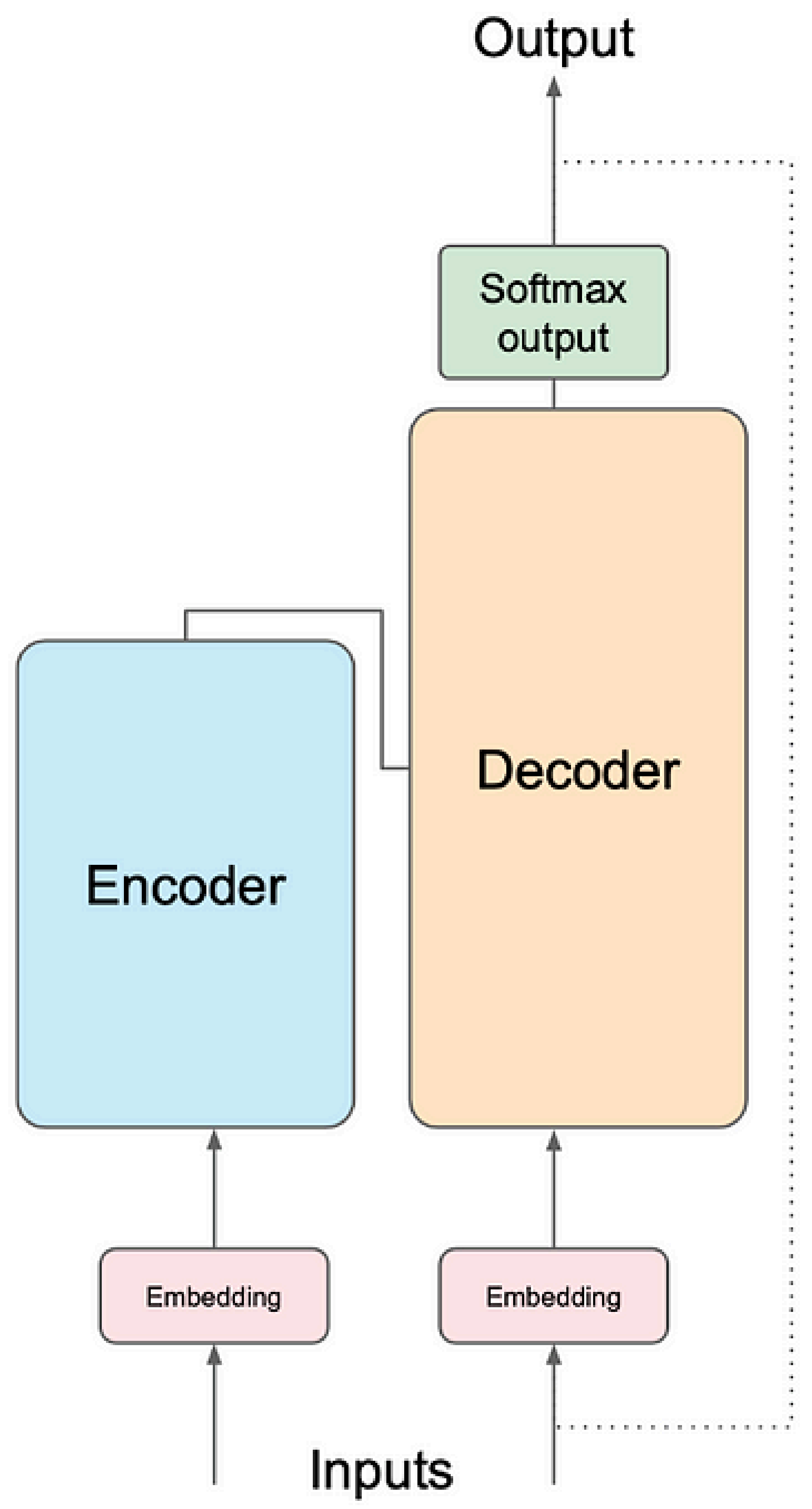
# Building Blocks of Transformers





A sequence of token IDs is created.



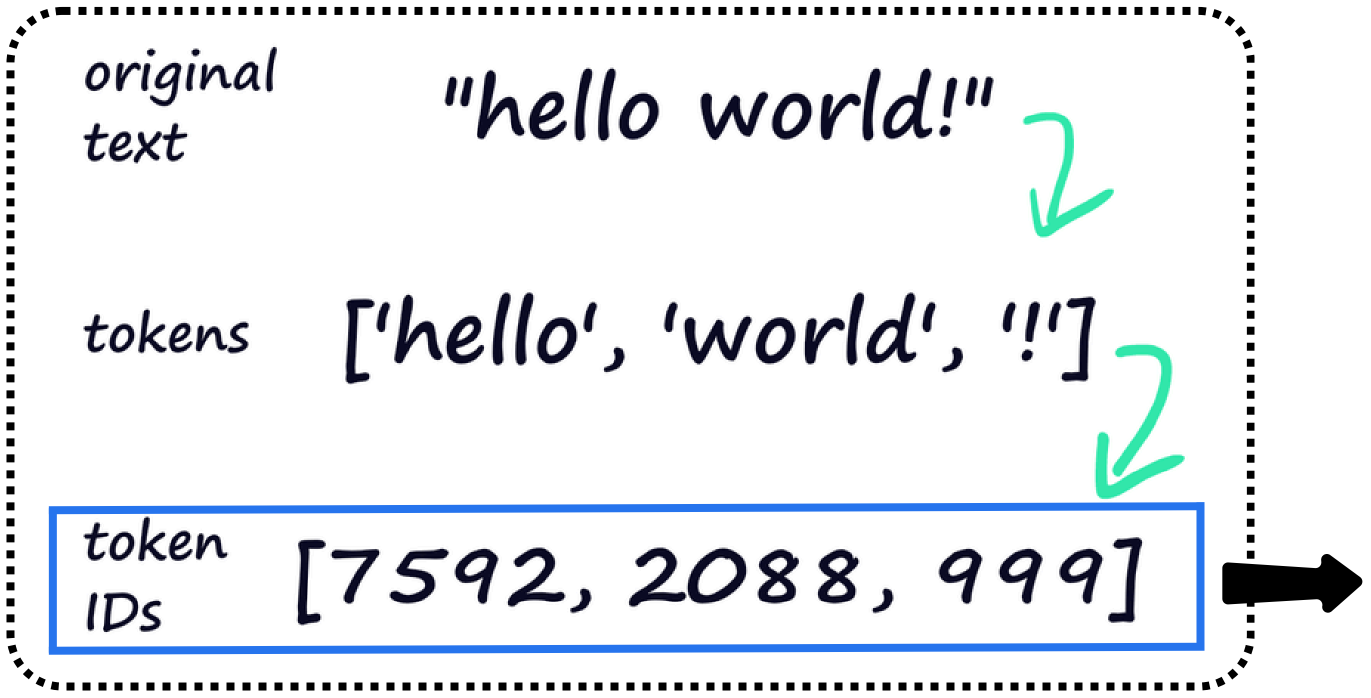


Transformers have revolutionized Natural Language Processing (NLP) by providing a flexible, efficient, and scalable architecture. Till now, we have covered the high-level understanding of transformers. But from now onwards,

let's dive deep into the fundamentals. Below, we explore how **tokenizers**, **embedding models**, **attention mechanisms**, and **pre-trained models** interconnect to form the foundation of transformers.

| | |
|---|---|
| **Tokenizer** | Converting Raw Text to Tokens |
| **Embedding Models** | Representing Tokens as Dense Vectors |
| **Attention Mechanism** | Understanding Relationships in Context |
| **Feedforward Layers** | Refining Contextual Representations |

# Tokenizers → Converting Raw Text to Tokens



A sequence of token IDs is created.

The process starts with Tokenizers, which split raw text into smaller pieces (tokens). The goal is to represent text in a structured format that a machine learning model can process.

## Why Tokenize?

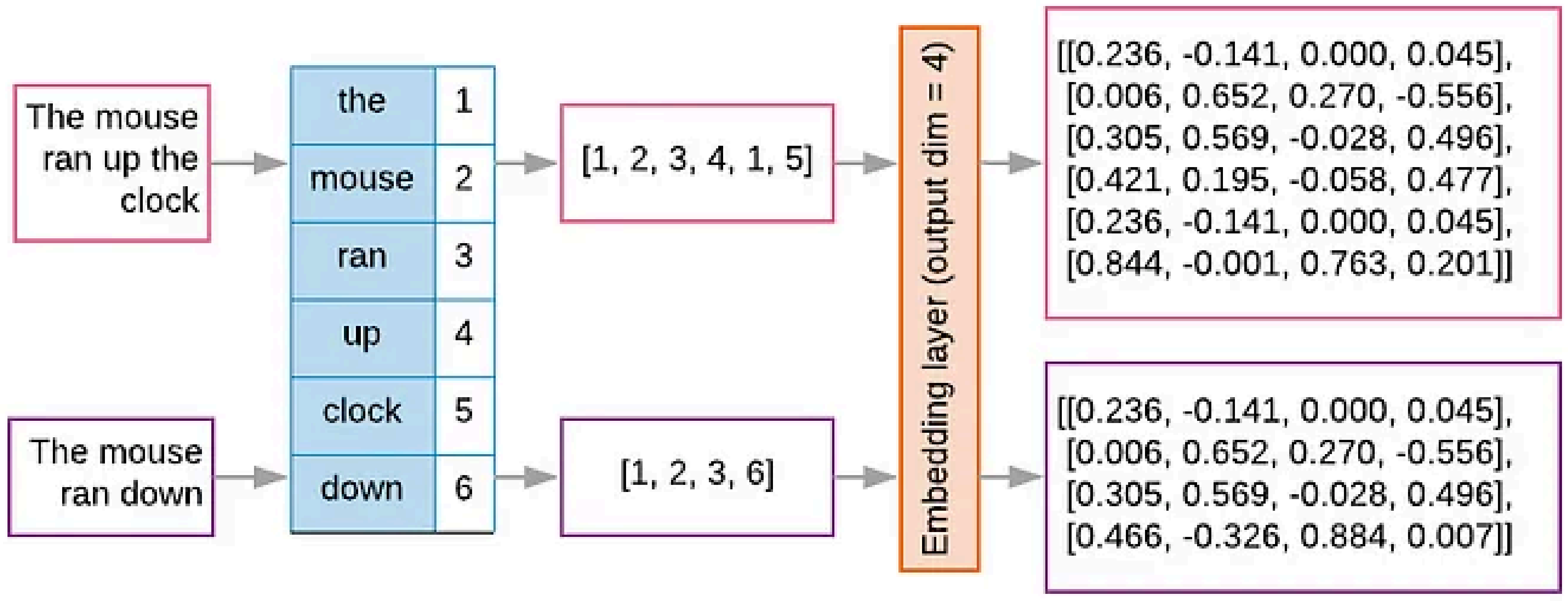
Computers process numbers, not text. Tokenizers bridge this gap by mapping words or subwords to numeric IDs.

## Types of Tokenization

Common approaches include word-based, subword-based (like Byte Pair Encoding), and character-based tokenization.

# Embedding Models

The token IDs from the tokenizer are not meaningful for machine learning models. They are passed to **embedding layers**, which convert them into **dense, continuous vector representations**.



## Why Embeddings?

Words with similar meanings (e.g., "king" and "queen") should have vectors that are close in their representation space. Embeddings capture these relationships.

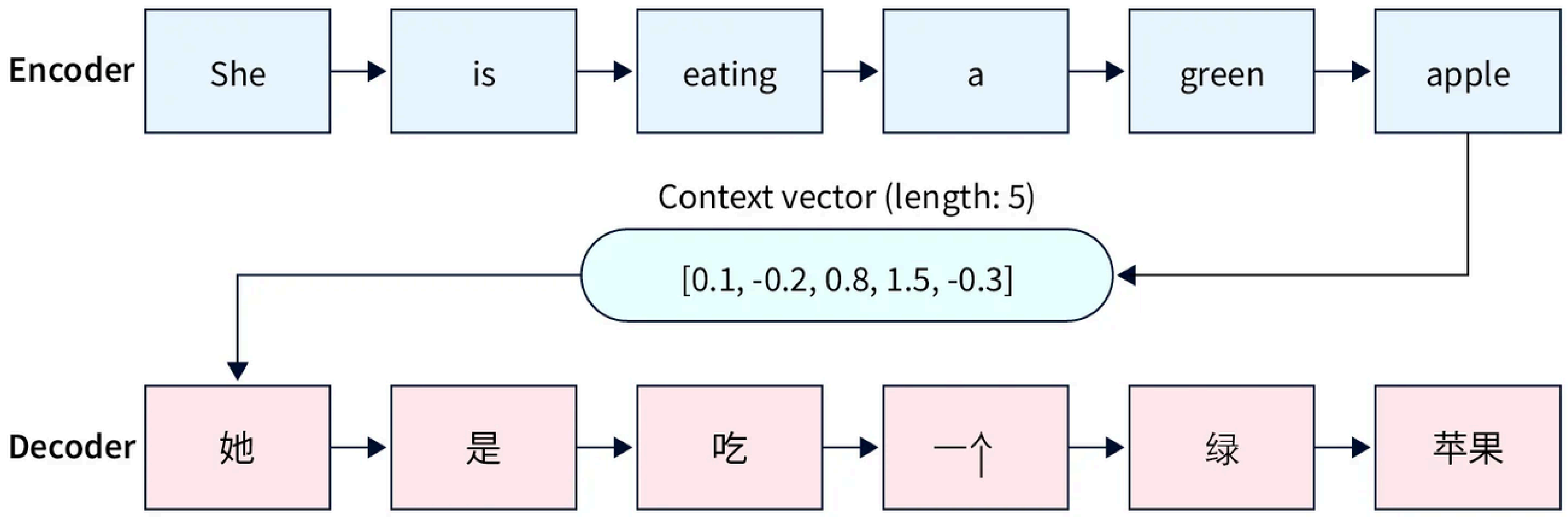
## Positional Embeddings

Transformers need a way to encode the sequence of tokens, as they lack inherent order awareness. Positional embeddings solve this by adding positional information to token embeddings.

# Attention Mechanism

*Understanding Relationships in Context*

The sequence of dense vectors from the embedding layer enters the attention mechanism, which is the heart of transformers. Attention helps the model determine how important each word is relative to others in the sequence.



## Key Processes in Attention

Self-Attention: Computes the relationships between every token in the sequence:

- Each token creates three vectors: Query (Q), Key (K), and Value (V).
- Relevance scores are computed using:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

## Multi-Head Attention

Uses multiple attention "heads" to capture different aspects of relationships.

## Example of Context Understanding

For the sentence "The cat sat on the mat," attention might focus on:

- "cat" → "sat"
- "sat" → "mat"

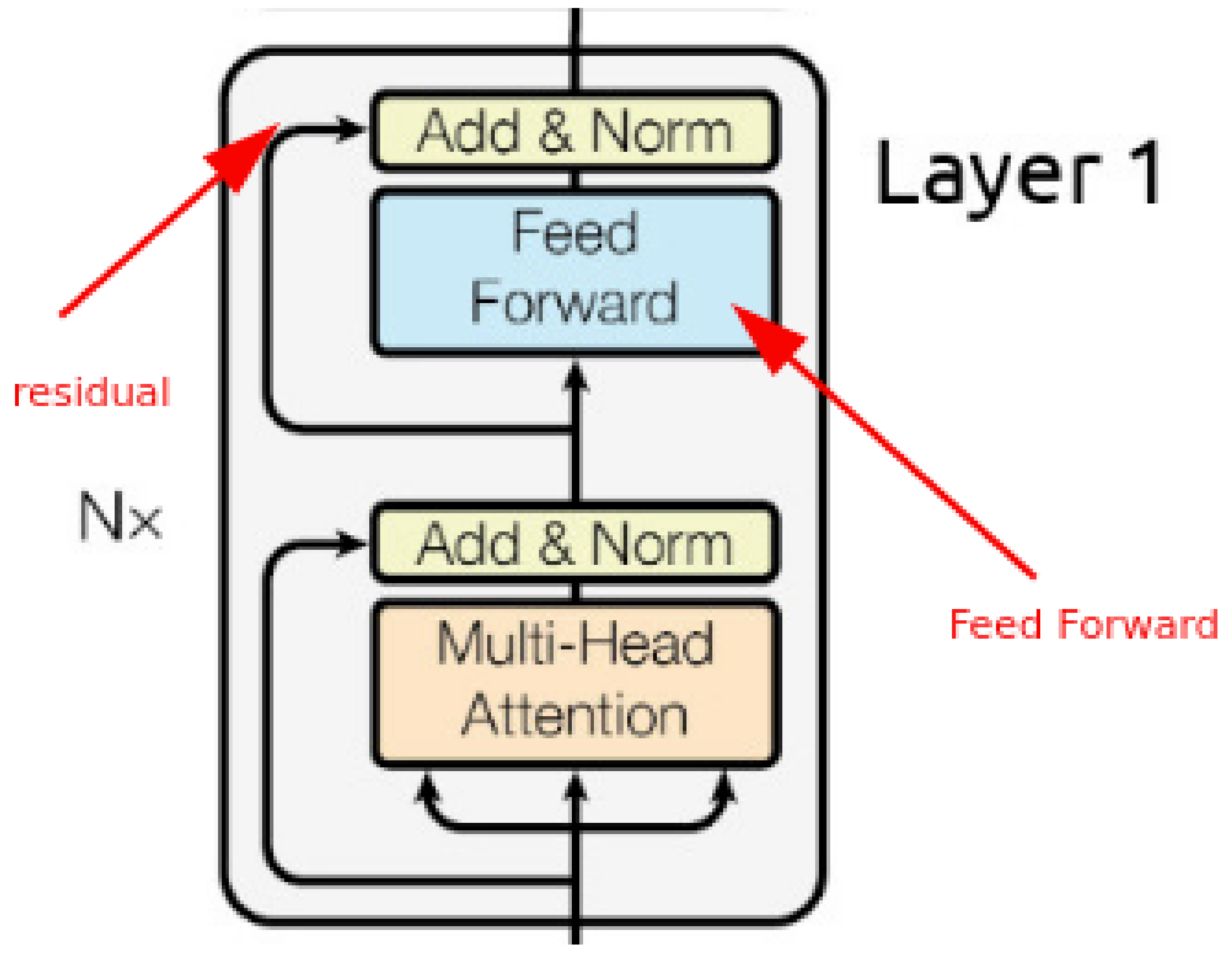
## Output of Attention Mechanism

Contextualized embeddings, where each token is enriched with information from other tokens in the sequence.

Analytics Vidhya

# Feedforward Layers

*Refining Contextual Representations*



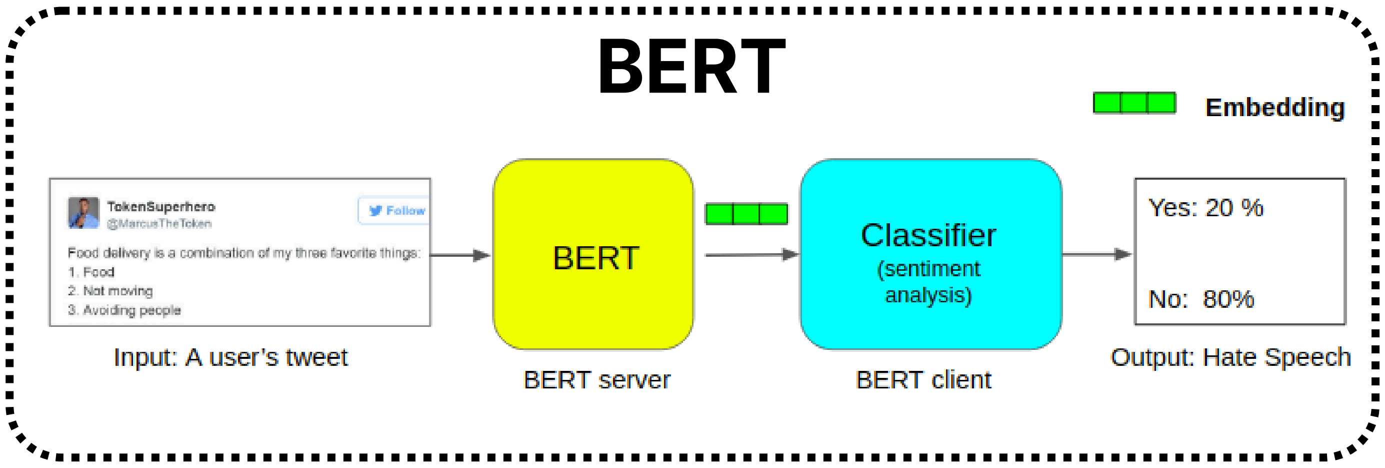
After attention, the contextualized embeddings are passed through feedforward layers for additional processing. This includes:

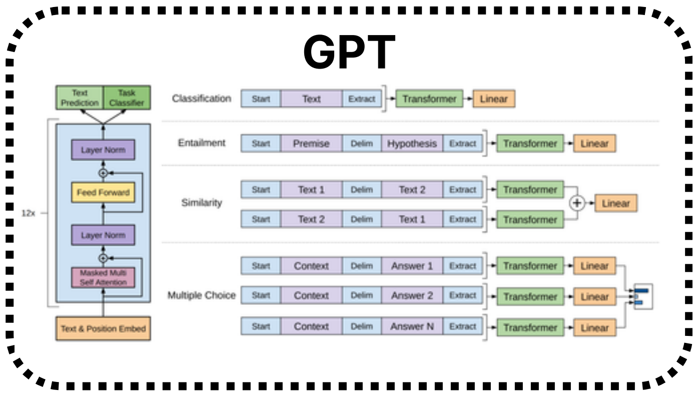
- Non-linear transformations.
- Adding the model's learned weights.

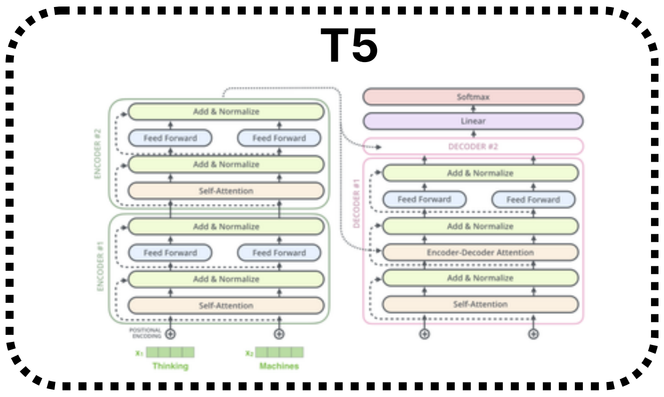
**Output of Feedforward Layers:** Refined contextual embeddings ready for downstream tasks.

# Pre-trained Models

*Leveraging Learned Representations*







Pre-trained models like BERT, GPT, and T5 use large-scale data to learn:

- Word relationships.
- Contextual understanding.
- General language representations

Instead of starting from scratch, we fine-tune these models on specific tasks (e.g., sentiment analysis, translation).

## Connection to Previous Steps

| | |
|---|---|
| **Tokenizer** | Provide input tokens to the pre-trained model. |
| **Embedding Models** | Create the initial vector representations. |
| **Attention Mechanism** | Helps the pre-trained model understand relationships within the input text. |

## Example with Pre-trained Models

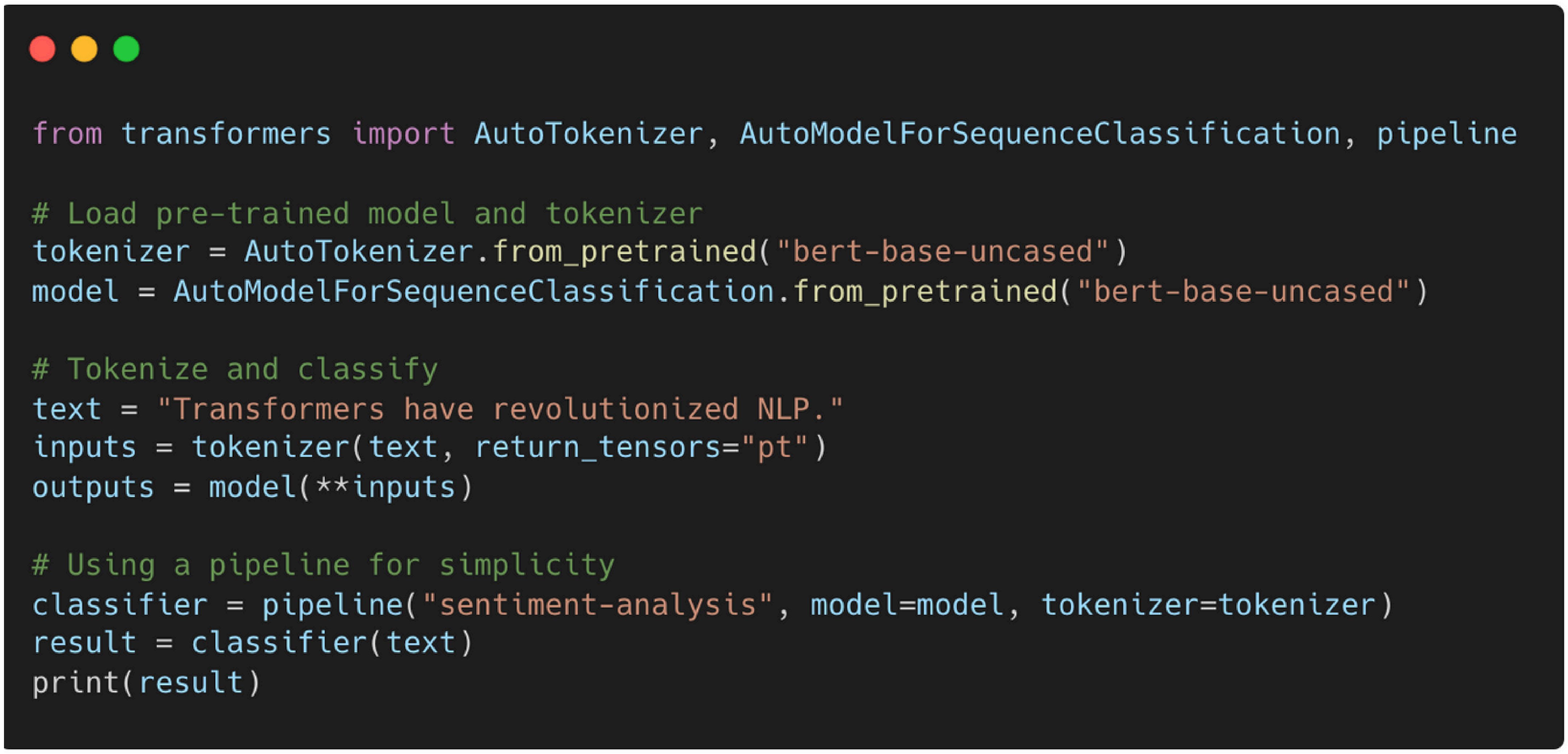
- **Input Text**: "The movie was fantastic!"
- **Pre-trained Model**: Maps tokens → embeddings → attention → predictions.
- **Output**: Sentiment: Positive.

## Fine-Tuning and Applications

Pre-trained models are generalized but can be fine-tuned for specific tasks:

- **Text Classification**: Predict a label for input text (e.g., sentiment).
- **Question Answering**: Extract an answer from a passage.
- **Conversational AI**: Generate appropriate responses.

## Unified Example



```python
from transformers import AutoTokenizer, AutoModelForSequenceClassification, pipeline

# Load pre-trained model and tokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
model = AutoModelForSequenceClassification.from_pretrained("bert-base-uncased")

# Tokenize and classify
text = "Transformers have revolutionized NLP."
inputs = tokenizer(text, return_tensors="pt")
outputs = model(**inputs)

# Using a pipeline for simplicity
classifier = pipeline("sentiment-analysis", model=model, tokenizer=tokenizer)
result = classifier(text)
print(result)
```