

Transformers and Attention (DRAFT)

Mark A. Austin

University of Maryland

austin@umd.edu

ENCE 688P, Spring Semester 2026

October 14, 2026

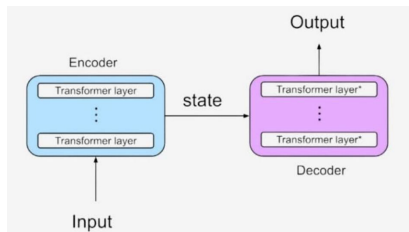
Overview

- 1 Introduction to Transformers
- 2 Computational Foundations
 - Tokenization, Embeddings, Embedding Operations
- 3 Attention
 - Query Key Value, Positional Information
 - Multi-Head Attention
- 4 Attention and Transformers
- 5 BERT Models
- 6 Applications
- 7 Appendix A: Cheatsheets

Definition of Transformers

Transformer

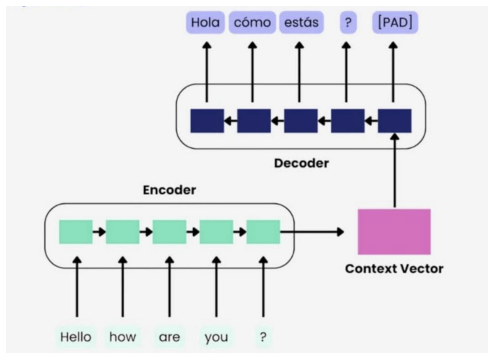
Transformer models (2017) are neural networks that use a technique called self-attention to take into account the context of elements in a sequence, not just the elements themselves.



Encoders compute meaningful embeddings of the input. Decoders then predict the next token in the sequence.

Definition of Transformers

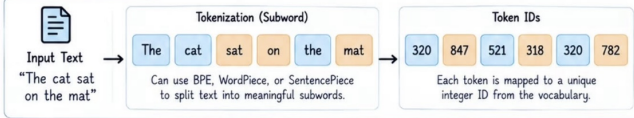
Transformer Neural Networks:



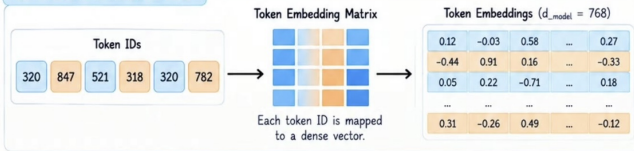
A context window (vector) is the maximum token length a model can process at once.

Transformer Overview

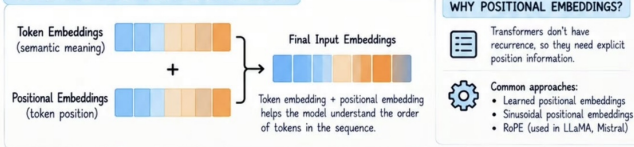
1. TOKENIZATION



2. TOKEN EMBEDDINGS

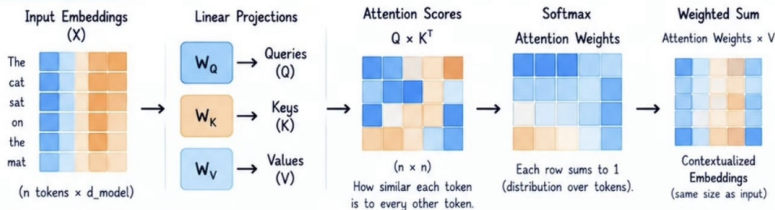


3. ADDING POSITIONAL INFORMATION



Transformer Overview

1. OVERVIEW



2. ATTENTION SCORE CALCULATION

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

Q

(Queries)

What am I
looking for?

K

(Keys)

What do I
offer?

V

(Values)

What information
do I pass?

3. EXAMPLE: ATTENTION WEIGHTS

	The	cat	sat	on	the	mat
The	0.30	0.20	0.10	0.10	0.20	0.10
cat	0.10	0.40	0.20	0.10	0.10	0.10
sat	0.05	0.25	0.40	0.15	0.10	0.05
on	0.10	0.10	0.20	0.40	0.15	0.05
the	0.25	0.15	0.10	0.10	0.30	0.10
mat	0.05	0.10	0.15	0.10	0.20	0.40

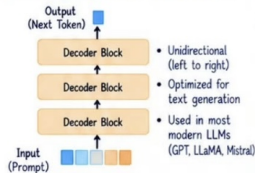
Each token
attends to all
tokens with
different weights.

Example:
"cat" gives
high attention
to "cat" and
"sat".

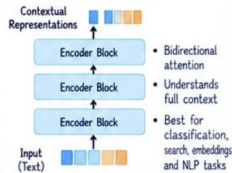
Transformer Overview

1. OVERVIEW OF TRANSFORMER VARIANTS

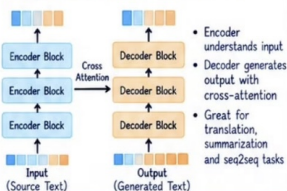
Decoder-Only (e.g., GPT, LLaMA)



Encoder-Only (e.g., BERT, RoBERTa)



Encoder-Decoder (e.g., T5, BART)



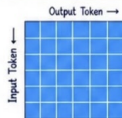
2. ATTENTION PATTERN COMPARISON

Decoder-Only (Causal Attention)



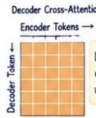
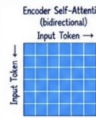
Each token can only attend to previous tokens (no future tokens).

Encoder-Only (Bidirectional Attention)



Each token can attend to all tokens in the input (both left and right).

Encoder-Decoder (Cross Attention)



Decoder attends to encoder outputs using cross-attention.

Types of Transformer Model

Encoder-Only: (e.g., BERT).

- Ideal for understanding text, classification, embeddings.

Decoder-Only: (e.g., GPT).

- Task support includes text generation, conversation, creative writing.

Encoder-Decoder: (e.g., T5).

- Focus on translation, summarization and sequence tasks.

Transformer Models

HOW IT WORKS (STEP BY STEP)



- 1 Input Embedding + Positional Encoding**
Words are converted into vectors and positional information is added to understand order.



- 2 Self-Attention**
Each word looks at every other word to understand its context and importance.



- 3 Multi-Head Attention**
Multiple attention heads learn different relationships (syntax, meaning, position).

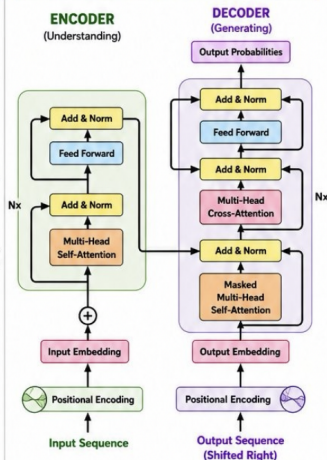


- 4 Feedforward Network**
Refines the information with fully connected layers and non-linearity.



- 5 Output**
The model produces predictions or generates new sequences.

TRANSFORMER ARCHITECTURE



KEY COMPONENTS



- Self-Attention**
Calculates how much focus each word should pay to others.



- Multi-Head Attention**
Allows the model to capture different types of relationships.



- Feedforward Network**
Adds non-linearity and enhances the representations.



- Residual Connection**
Helps training deep networks by avoiding gradient vanishing.



- Layer Normalization**
Stabilizes and speeds up the training process.



- Positional Encoding**
Adds position info since transformers process in parallel.

Transformer Models

TRANSFORMERS vs TRADITIONAL MODELS

FEATURE	RNN / LSTM	TRANSFORMER
 Processing	Sequential	Parallel
 Speed	Slow	Fast
 Long Dependencies	Weak	Strong
 Scalability	Limited	High
 Context Awareness	Limited	Global
 Training Efficiency	Low	High

TYPES OF TRANSFORMER MODELS



ENCODER-ONLY
(e.g., BERT)

Great for understanding text, classification, embeddings.



DECODER-ONLY
(e.g., GPT)

Great for text generation, conversation, creative writing.



ENCODER-DECODER
(e.g., T5)

Great for translation, summarization, and sequence tasks.

REAL-WORLD APPLICATIONS



NLP

Translation, Chatbots, Summarization, Sentiment Analysis



COMPUTER VISION

Image Classification, Object Detection, Vision Transformers



SPEECH

Speech Recognition, Transcription, Voice Assistants



RECOMMENDATION SYSTEMS

Personalized Recommendations, User Behavior Modeling



CODE & DATA

Code Generation, Bug Detection, Data Analysis

WHY TRANSFORMERS REVOLUTIONIZED AI



PARALLEL PROCESSING

Trains faster and handles more data efficiently.



BETTER CONTEXT UNDERSTANDING

Captures long-range dependencies with ease.



SCALABILITY

Performs exceptionally well with large datasets.



TRANSFER LEARNING

Pre-trained models can be fine-tuned for many tasks.



VERSATILITY

Works across text, images, audio, code and more.



Benefit of Transformers

Benefits:

This ability makes transformers ideal for:

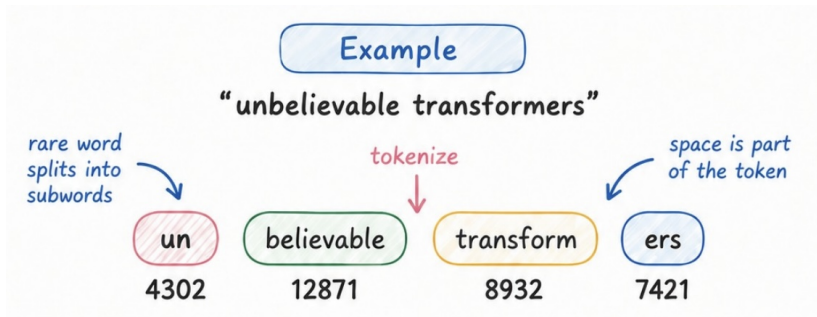
- Analyzing sentences/paragraphs of text, and
- Translating languages and understanding complex sentences.
- Computer vision (image classification, object detection, vision transformers).
- Speech recognition, transcription, voice assistants.
- Recommendation systems.

Computational Foundations

Tokenization

Tokenization

Tokenization is the process of **breaking raw text** into **smaller pieces** called **tokens**. Tokens are then mapped onto integer IDs using a fixed vocabulary so that AI can understand them.



Tokenization

Type	Pros	Cons	Illustration
Word	• Easy to interpret • Short sequence	• Large vocabulary size • Word variations not handled	<code>teddy bear</code>
Subword	• Word roots leveraged • Intuitive embeddings	• Increased sequence length • Tokenization more complex	<code>ted ##dy bear</code>
Character Byte	• No out-of-vocabulary concerns • Small vocabulary size	• Much longer sequence length • Patterns hard to interpret because too low-level	<code>t e d d y b e a r</code>

Embedding Spaces

Embedding

An **embedding** is a numerical representation of an element (e.g., token, sentence) that is characterized by vector $x \in \mathbb{R}^n$.

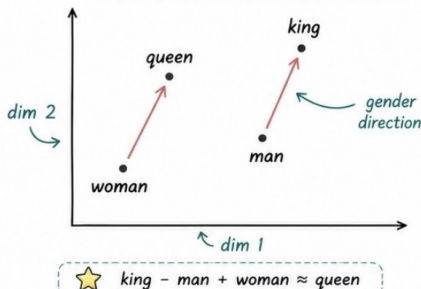
WHAT IS IT?

- ✓ Every word/token is a point (vector) in a high-dimensional space.
- ✓ Dimensions are learned by the model.
- ✓ Semantic meaning is captured by position and direction.



We can't visualize 768+ dimensions, so we project them down to 2D for intuition.

VISUALIZING IN 2D (FOR INTUITION)



Embedding Spaces

EXAMPLES OF RELATIONSHIPS

- Paris - France + Italy $\approx$ Rome
- walk + ing $\approx$ walking
- cat + plural $\approx$ cats
- teacher - man + woman $\approx$ teacher (female)

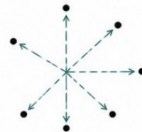
} directions
encode
meaning



These aren't perfect, but they hold surprisingly well across many word pairs.

HOW IT WORKS

- 1 Words with similar meanings are close in space.
- 2 Relationships become consistent directions.
- 3 Arithmetic on vectors captures those relationships.
- 4 This generalizes to unseen word pairs.



The model learns this space during training by predicting context.

KEY PROPERTIES



Continuous

Small changes in position mean small semantic changes.



Structured

Captures complex relationships (synonyms, analogies, roles).



Universal

(to some extent)

Patterns transfer across many languages and tasks.



High-Dimensional

Typically 512, 768, 1024, ... dims (we just can't see them!).

WHY IT MATTERS

- ✓ Enables semantic search and retrieval.
- ✓ Improves generalization and reasoning.
- ✓ Foundation for everything the model does.

Analogy Example

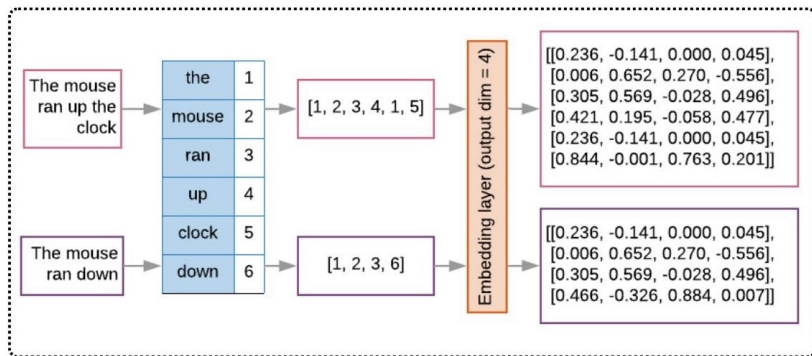
king - man + woman $\approx$ queen

If this works, the model understands the underlying concept of "gender".



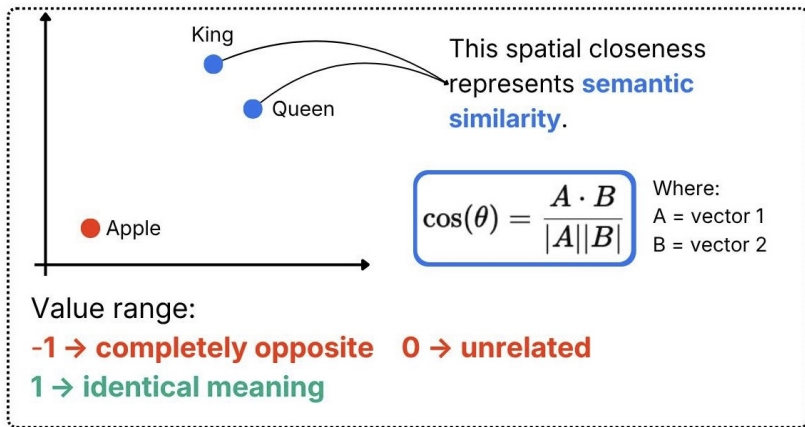
Embedding Operations

Sentence Embeddings:



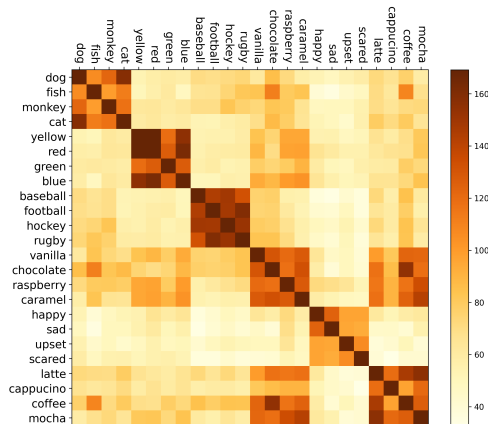
Embedding Operations

Similarity:



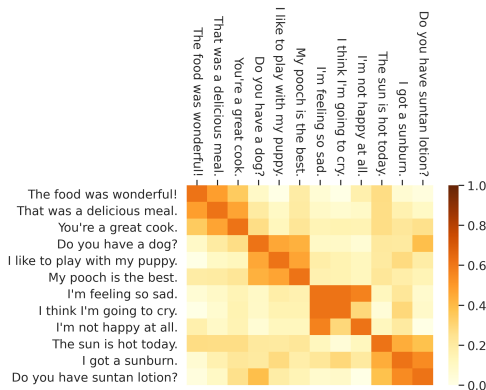
Embedding Operations

Similarity Matrix: Visual representation of similarity among pairs of words.



Embedding Operations

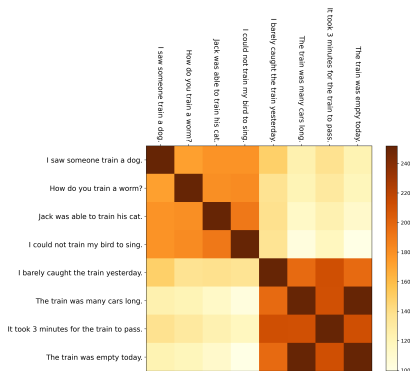
Similarity Matrix: Comparing sentence embeddings.



Source: Chapter 20, Glassner, 2021.

Embedding Operations

Similarity Matrix: Comparison of contextualized word embeddings (elmos) of “train” resulting from it use in different sentences.



Source: Chapter 20, Glassner, 2021.

Attention

Attention Mechanisms

Attention

Attention mechanisms allow models to weigh the importance of different words in a sequence.

Limitation of Past Approaches:

- Recurrent models (e.g., RNN and LSTM) use a single hidden state to store all information from the past.
- A network has to constantly decide what to remember and what to forget, often discarding information that later turns out to be important.

Central Idea of Attention:

- Do not compress everything; retrieve what matters.

Attention Mechanisms

Motivating Example:

Consider the sentence:

The kiwi slept all day because it was tired.

Figuring out that “it” relates to “kiwi” is a retrieval operation.

We implicitly search the available information and select the most relevant option.

The goal of attention is to explicitly learn the relationship among words from the data – during training attention determines what relationship reduce prediction error.

Self-Attention

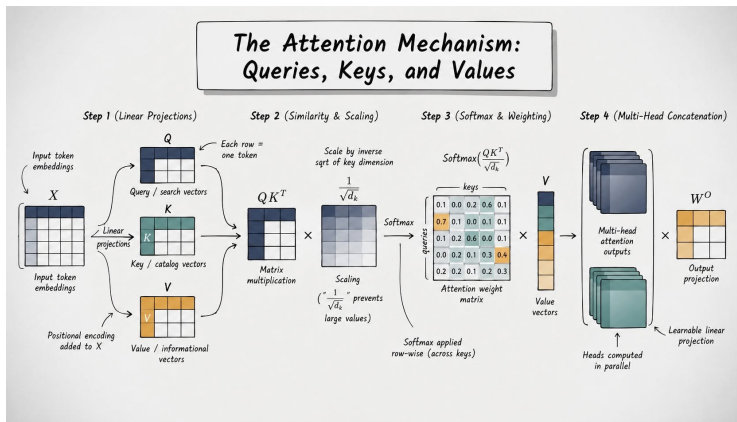
In **self-attention**, each word looks at every other word to understand its context and importance, regardless of the distance between them.

The diagram illustrates the flow of data in a Transformer-style attention mechanism. It shows a vertical stack of operations: **MatMul** (bottom), **Scale**, **Mask (opt.)**, **SoftMax**, and **MatMul** (top). Arrows indicate the flow of data: **Q** and **K** are inputs to the bottom **MatMul**; its output goes to **Scale**, then **Mask (opt.)**, then **SoftMax**. The output of **SoftMax** is multiplied (via the top **MatMul**) with the input **V** to produce the final output.

The diagram illustrates the internal structure of a Transformer Encoder block. At the bottom, three input vectors are labeled V , K , and Q . Each input vector passes through a 'Linear' layer. The outputs of these three linear layers are fed into a 'Scaled Dot-Product Attention' block. This block is part of a stack of identical blocks, with a double-headed arrow indicating the repetition. The output of the attention block is then passed through a 'Concat' block (highlighted in yellow) and finally a 'Linear' layer to produce the final output vector.

Attention Mechanisms

Computational Procedure:



Query Key Value

Language of Attention: Each token is converted into an embedding vector. For each embedding, generate three vectors:

- **Query (Q):** What am I looking for? Each word serves as a query to compute the attention scores.
- **Key (K):** What kind of information do I contain? Each word acts as a key to determine the relevance of other words.
- **Value (V):** What kind of information should be returned if I am relevant? Each word contributes as a value to the attention-weighted sum.

Query-Key-Value Interaction

Given a query q we want to know which key k the query should pay “attention” to with respect to the associated value v .

Query Key Value

Mathematical Formula: Vectors Q, K and V are derived by multiplying the input embeddings by the learned weight matrices.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V. \quad (1)$$

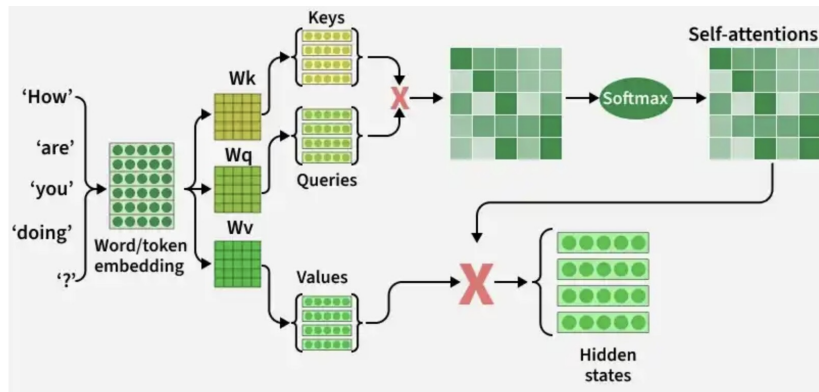
Here:

- Q: Query matrix.
- K^T : Transposed key matrix matrix.
- d_k : Dimension of the key vectors.
- V: Value matrix.

Source: [Geeksforgeeks.org](https://www.geeksforgeeks.org/)

Query Key Value

Scaled Dot-Product Attention:



Query Key Value

Scaled Dot-Product Attention:

- **Linear Transformation:** Each input word is converted into Query, Key and Value vectors using weight matrices $W(Q)$, $W(K)$ and $W(V)$.
- **Scale and Normalize:** Scores are scaled by dividing by $\sqrt{d_k}$ to prevent very large values and ensure stable training.
- **Softmax Normalization:** Scaled scores are passed through a softmax function to convert them into probabilities.
- **Final Output:** All weighted value vectors are summed to produce the final representation for each word.

Positional Information

Embedding:

Original sentence
(tokens)

YOUR

CAT

IS

A

LOVELY

CAT

Input IDs (position in
the vocabulary)

105

6587

5475

3578

65

6587

Embedding
(vector of size 512)

952.207

5450.840

1853.448

...

1.658

2671.529

171.411

3276.350

9192.819

...

3633.421

8390.473

621.659

1304.051

0.565

...

7679.805

4506.025

776.562

5567.288

58.942

...

2716.194

5119.949

6422.693

6315.080

9358.778

...

2141.081

735.147

171.411

3276.350

9192.819

...

3633.421

8390.473

We define $d_{\text{model}} = 512$, which represents the size of the embedding vector of each word

Positional Information

Position Embedding:

Original sentence

YOUR

CAT

IS

A

LOVELY

CAT

Embedding
(vector of size 512)

952.807
5450.840
1853.448
...
1.658
2671.529

171.411
3276.350
9192.819
...
3633.421
8390.473

421.659
1304.051
0.565
...
7679.805
4506.025

276.562
5567.288
58.942
...
2716.194
5119.949

6422.493
6315.880
9358.778
...
2141.081
735.147

171.411
3276.350
9192.819
...
3633.421
8390.473

+

+

+

+

+

+

Position Embedding
(vector of size 512).
Only computed once
and reused for every
sentence during
training and inference.

...
...
...
...
...
...

1664.068
8080.133
2620.399
...
9386.405
3120.159

...
...
...
...
...
...

...
...
...
...
...
...

...
...
...
...
...
...

1281.458
7992.890
912.970
3821.102
1659.217
2018.620

==

==

==

==

==

==

Encoder Input
(vector of size 512)

...
...
...
...
...

1835.479
11256.483
11813.218
...
13019.826
11510.632

...
...
...
...
...

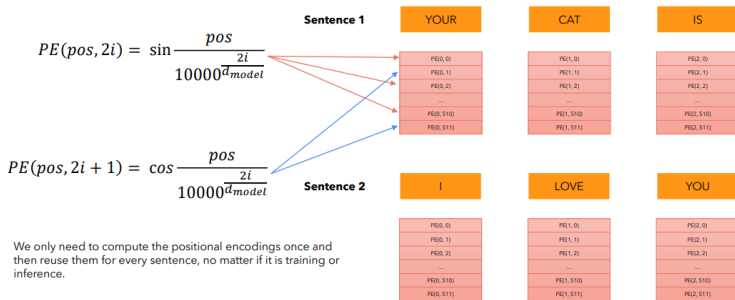
...
...
...
...
...

...
...
...
...
...

1652.869
11179.24
10105.789
...
5292.638
15409.093

Positional Information

Embedding:

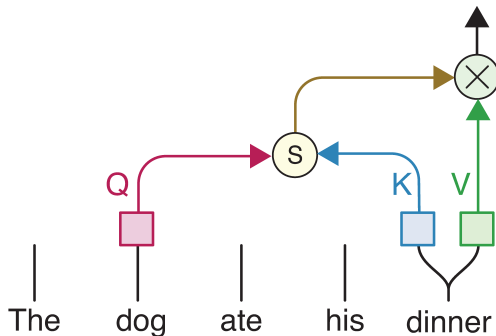


Positional Embedding $\rightarrow$ Similarity:



Attention: Simple Example

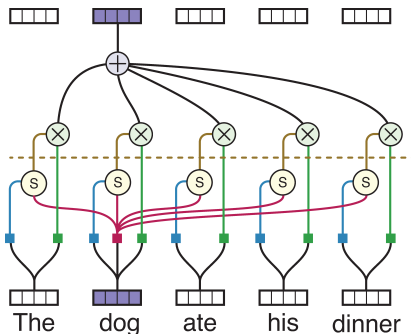
Example: The dog ate his dinner



Core step of attention: Using *dog* as the query, determine the relevance of the word *dinner*.

Attention: Simple Example

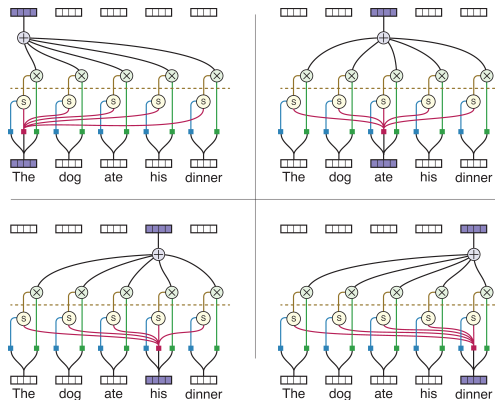
Example: The dog ate his dinner



Contribution of each word: Using attention to simultaneously determine the contribution of all five words in the sentence to the word *dog*.

Attention: Simple Example

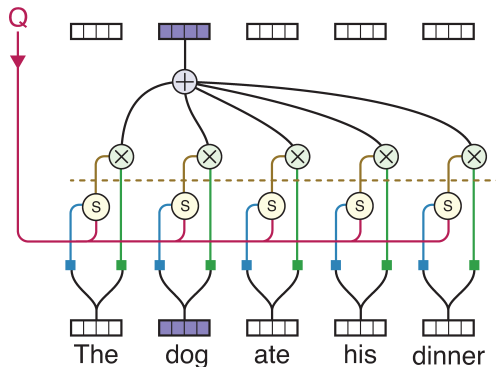
Example: The dog ate his dinner



Contribution: Applying attention to the other four words in the sentence.

Attention: Simple Example

Example: The dog ate his dinner

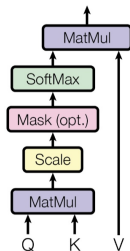


Q/KV Attention: Queries don't come from the inputs.

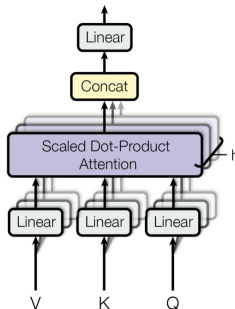
Multi-Head Attention

Multi-head attention mechanisms learn different relationships (syntax, meaning, position).

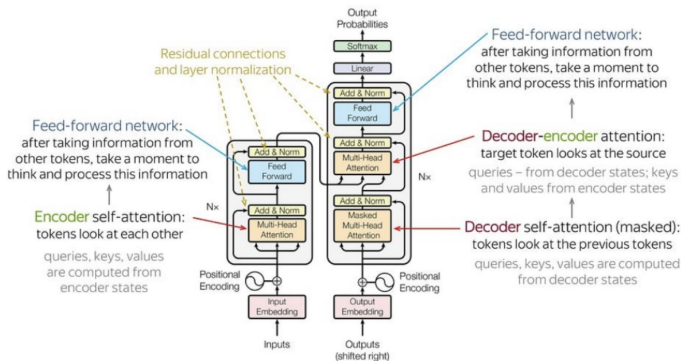
Scaled Dot-Product Attention



Multi-Head Attention



Multi-Head Attention



Input Sequence

A sequence of words or tokens that the model will process.

Multi-Head Attention

Idea: Multi-Head Attention runs multiple self-attention heads in parallel, where each head can learn to focus on different relationships in the input. The outputs from all heads are then combined to capture richer information.

Intuition

- Each attention head can focus on different relationships in the input.
- Different heads may learn to capture different patterns, such as syntax, positional relationships, or long-range dependencies.
- Multiple heads together can capture richer relationships in the context.
- The outputs are concatenated and projected back to the original dimension.

Toy Example (2 Tokens, 2 Heads)

Input Tokens: [A , B]

Let $d_{model} = 4$, $h = 2$, so $d_k = d_v = 2$

Step 1: Linear Projections (Example Values)

$$Q = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad K = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad V = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 2 & 0 & 0 & 0 \end{bmatrix}$$

Step 2: Split into 2 heads (each head has $d_k = d_v = 2$)

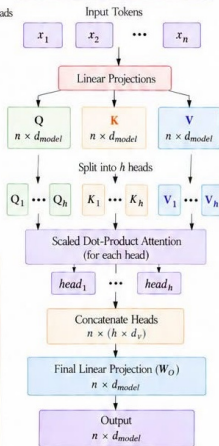
Split the columns of Q, K, V into 2 heads:

$$\begin{aligned} Q_1 &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & Q_2 &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\ K_1 &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & K_2 &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\ V_1 &= \begin{bmatrix} 1 & 0 \\ 2 & 0 \end{bmatrix} & V_2 &= \begin{bmatrix} 0 & 2 \\ 0 & 0 \end{bmatrix} \end{aligned}$$

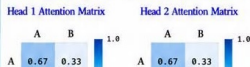
Step 3: Attention for each head (Head 1)

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

How Multi-Head Attention Works



Visualizing Multiple Attention Heads



Mathematical Formulation

Given:

$$Q = XW_Q \in \mathbb{R}^n \times d_{model} \quad (\text{Queries})$$

$$K = XW_K \in \mathbb{R}^n \times d_{model} \quad (\text{Keys})$$

$$V = XW_V \in \mathbb{R}^n \times d_{model} \quad (\text{Values})$$

For each head $i = 1, 2, \dots, h$:

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right)V_i$$

Concatenate and project:

$$\text{Multi-Head}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W_O$$

Where:

- h = number of attention heads
- d_{model} = model dimension (embedding size)
- For equal-sized heads: $d_k = d_v = d_{model} / h$
- $W_Q, W_K, W_V \in \mathbb{R}^{d_{model} \times d_{model}}$
- $W_O \in \mathbb{R}^{(h \times d_v) \times d_{model}}$

Key Points

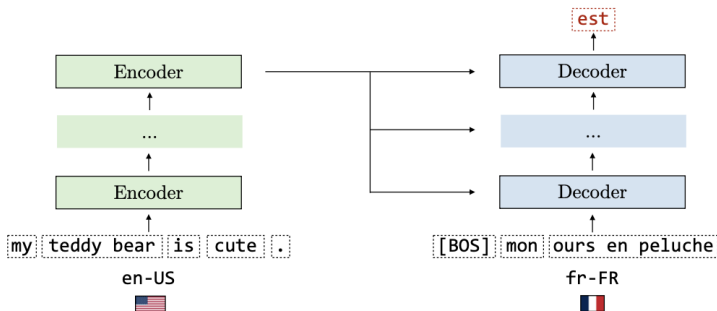
- Multiple heads run in parallel, each with its own learned projections.
- Each head can capture different types of relationships between tokens.
- Outputs from all heads are concatenated and projected back to the original dimension.
- Using multiple heads can improve the model's ability to capture complex patterns.
- Multi-head attention is a core component of Transformers and LLMs.

Attention and Transformers

Transformer Definition

Transformer

A **transformer** model uses **self-attention mechanisms** and a composition of encoders and decoders to generate token-aware, position-aware, and context-aware embeddings.

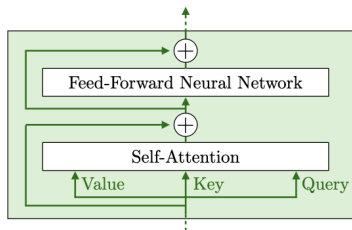


Transformer Definition

Encoders compute meaningful embeddings of the input.
Decoders then predict the next token in the sequence.

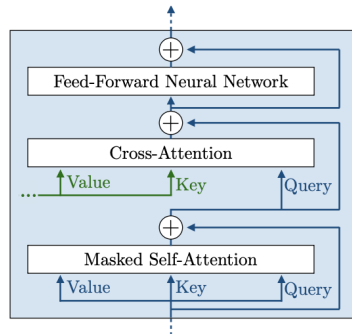
Encoder

Encoded embeddings encapsulate meaning of input



Decoder

Decoded embeddings encapsulate meaning of both input and output predicted so far

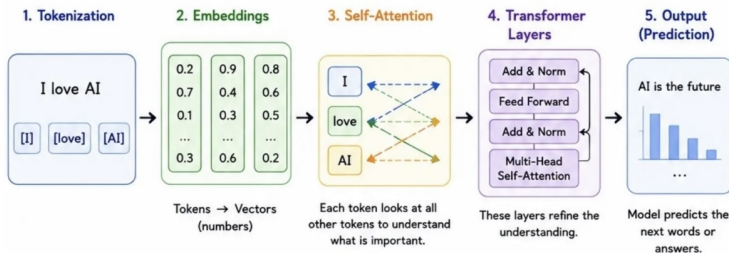


Transformer Process

Tokenization: Break text into smaller pieces called tokens.
Convert token into numbers so that AI can understand them.



Transformer: Use self-attention to understand relationships.

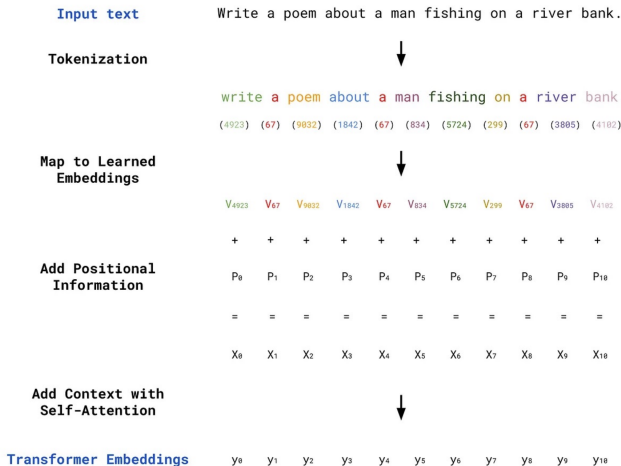


Step-by-Step Procedure

- Read input.
- Tokenizer (process input tokens in parallel – compute-bound).
- Embedding layer.
- Transformer Block (multi-head self attention; KV cache).
- Linear and softmax transformations (estimate logits and probabilities over vocabulary).
- Sampling strategy (e.g., greedy, top-K tokens, sample from cumulative probability threshold)..
- Speculative decoding (speed trick).
- Detokenizer (generate tokens one-by-one).
- Stream output (to end user).

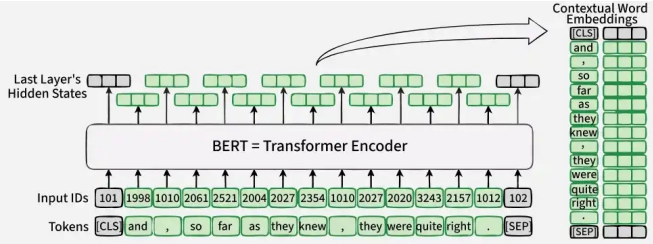
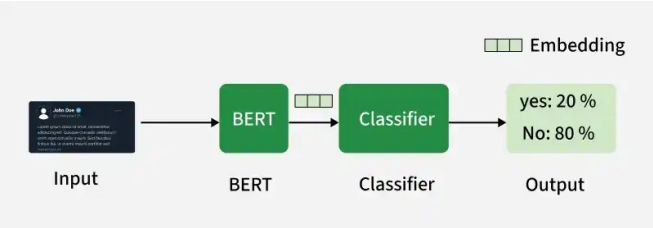
Transformer Embedding Process

Transformer Embedding Process:



BERT Models

BERT Models



BERT Models

WHAT IS BERT?

BERT is a pre-trained deep learning model designed for Natural Language Processing (NLP). It understands the full context of a word by reading text bidirectionally (both left and right).

Example:

- He deposited money in the **bank**.
 ↳ Financial Institution
- She sat near the river **bank**.
 ↳ River Side

BERT understands the context and distinguishes the correct meaning.

HOW BERT WORKS

BERT reads the entire sentence at once and understands relationships between all words.

It uses bidirectional context to capture the meaning of a word.

Example:

"I went to the bank to withdraw cash."

BERT looks at both "withdraw" and "cash" to understand that "bank" refers to a financial institution.

PRE-TRAINING TASKS

- Masked Language Modelling (MLM)**
Random words are masked and BERT predicts the missing words.

Example:

The cat sat on the [MASK].

Prediction: mat

- Next Sentence Prediction (NSP)**
BERT predicts whether the next sentence follows logically.

Example:

Sentence A: She went to the library.
Sentence B: She borrowed a book.

Prediction: IsNext (Yes)

WHY BERT IS A BREAKTHROUGH

Feature	Traditional NLP Models	BERT
Context Understanding	One-directional	Bidirectional
Training Speed	Slower	Faster with Transformers
Parallel Processing	Limited	Supported
Understanding Ambiguity	Weak	Strong
Transfer Learning	Limited	Highly Effective
Fine-Tuning	Difficult	Simple & Powerful

BERT ARCHITECTURE OVERVIEW

The architecture consists of L layers. Each layer contains a Multi-Head Attention sub-layer followed by a Feed Forward Network (FFN) sub-layer, both with residual connections and layer normalization (Add & Norm).

Input Embeddings: [CLS], [I], [went], [to], [the], [bank], [SEP]

Token Embeddings: $E_{[CLS]}$, E_I , E_{went} , E_{to} , E_{the} , E_{bank} , $E_{\text{[SEP]}}$

Segment Embeddings: E_A , E_A , E_A , E_A , E_A , E_A , E_A

Position Embeddings: 0, 1, 2, 3, 4, 5, 6

REAL-WORLD APPLICATIONS

- Search Engines**
Understands user intent and context for better search results.
- Chatbots & Virtual Assistants**
Enables natural conversations and accurate responses.
- Sentiment Analysis**
Analyzes emotions in reviews, feedback and social media.
- Question Answering**
Extracts precise answers from passages and documents.
- Named Entity Recognition**
Identifies people, organizations, locations, dates and more.
- Language Translation**
Improves multilingual understanding and translation accuracy.

INPUT REPRESENTATION

BERT uses special tokens:

Token	Purpose
[CLS]	Classification token (used for overall representation)
[SEP]	Separates sentences
[MASK]	Represents a word that is masked

Example:

[CLS] What is BERT? [SEP]

The output of [CLS] token is commonly used for classification tasks.

FINE-TUNING BERT

Steps in Fine-Tuning

- Load pre-trained BERT
- Add task-specific output layer
- Train on target dataset

BERT can be fine-tuned for many NLP tasks using smaller datasets.

BERT MODEL SIZES

Model	Layers	Hidden Units	Attention Heads	Parameters
BERT Base	12	768	12	110 Million

BERT Models

BERT vs GPT

Feature	BERT (Encoder)	GPT (Decoder)
Architecture	Encoder-only	Decoder-only
Training Direction	Bidirectional	Left-to-right
Best For	Understanding	Generation
Applications	Classification, QA, NER, Sentiment Analysis	Text Generation, Summarization, Chatbots
Context Handling	Strong Comprehension	Strong Generation

IMPACT ON SEO & DIGITAL MARKETING



SEO Shift:

- From Keyword Stuffing → To User Intent
- From Exact Match → To Semantic Relevance
- From Short Content → To Quality & Context-Rich Content

BERT helped Google understand search queries better, leading to smarter search results.

BERT VARIANTS

- 🌟 **RoBERTa** Improved training strategy, larger datasets
- 🌟 **DistilBERT** Smaller, faster and lighter version
- 🌟 **ALBERT** Parameter reduction, more efficient
- 🌟 **BioBERT** Designed for biomedical NLP tasks
- 🌟 **TinyBERT** Lightweight for mobile & deployment

BERT ADVANTAGES

- ✓ Deep context understanding
- ✓ High accuracy on NLP benchmarks
- ✓ Transfer learning makes it powerful
- ✓ Flexible fine-tuning for many tasks
- ✓ Better semantic understanding

LIMITATIONS

- ✗ Computationally expensive
- ✗ High memory usage
- ✗ Slow inference
- ✗ Limited input sequence length

THE FUTURE OF BERT & NLP



- More Efficient Models
- Multimodal Understanding
- Real-time & Context-aware AI
- Domain-Specific Adaptations
- Foundation for Advanced LLMs (GPT, TS, PaLM, Gemini, LLaMA)

PYTHON EXAMPLE (Hugging Face)

```
from transformers import BertTokenizer, BertModel
import torch

# Load tokenizer and model
tokenizer = BertTokenizer.from_pretrained(
    'bert-base-uncased')
model = BertModel.from_pretrained(
    'bert-base-uncased')

# Input text
text = "BERT is transforming NLP."
# Tokenize input
inputs = tokenizer(text, return_tensors='pt')
# Generate embeddings
outputs = model(**inputs)
print(outputs.last_hidden_state.shape)
```

This code loads a pre-trained BERT model and generates contextual embeddings for the text.

BERT revolutionized NLP by enabling machines to understand language like never before.
It remains a foundation of modern AI and continues shaping the future of intelligent applications.



Applications

Transformer Applications

References

- Amidi A., and Amidi S., Super Study Guide: Transformers and Large Language Models (First Edition), ISBN 979-8-836693-31-2, 2024.
- Auffarth B., and Kuligin L., Generative AI with LangChain (Second Edition), Packt Publishing, 2025.
- Glassner A., Deep Learning: A Visual Approach, Starch Press, San Francisco, 2021.
- Lakshmanan V., and Hapke H., Generative AI Design Patterns: Solutions to Common Challenges when building GenAI Agents and Applications, O'Reilly, 2026.
- Negro A., Futia G., Kus V., and Montagna F., Knowledge Graphs and LLMs in Action, Manning Publications, 2026.
- Raieli S., and Luculano G., Building AI Agents with LLMs, RAG and Knowledge Graphs, Packt Publishing, 2025.
- Shetty R., Modeling Intelligence: The Hidden Mathematical Structures behind Modern AI, Independently Published, 2026.

Appendix A

Cheatsheets

Appendix A. Embedding Space

LLM-Series

techwithram

Embedding Space

The high-dimensional vector space where embeddings live.
Meaningful relationships, such as gender or plural,
become consistent directional offsets that work across many word pairs.

WHAT IS IT?

- Every word/taken is a point (vector) in a high-dimensional space.
- Dimensions are learned by the model.
- Semantic meaning is captured by position and direction.

We can't visualize 768 dimensions, so we project them down to 2D for intuition.

VISUALIZING IN 2D (FOR INTUITION)

king - man + woman = queen

EXAMPLES OF RELATIONSHIPS

- Paris - France + Italy = Rome
- walk + ing = walking
- cat + plural = cats
- teacher - man + woman = teacher (female)

directions encode meaning

These aren't perfect, but they hold surprisingly well across many word pairs.

HOW IT WORKS

- Words with similar meanings are close in space.
- Relationships become consistent directions.
- Arithmetic on vectors captures those relationships.
- This generalizes to unseen word pairs.

The model learns this space during training by predicting context.

KEY PROPERTIES

- Continuous**: Small changes in position mean small semantic changes.
- Structured**: Captures complex relationships (synonyms, analogies, roles).
- Universal** (to some extent): Patterns transfer across many languages and tasks.
- High-Dimensional**: Typically 512, 768, 1024, ... dims (we just can't see them!).

WHY IT MATTERS

- Enables semantic search and retrieval.
- Improves generalization and reasoning.
- Foundation for everything the model does.

Analogy Example

king - man + woman = queen

If this works, the model understands the underlying concept of "gender".

KEY TAKEAWAY

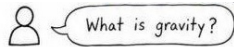
In embedding space, meanings become geometry.
and relationships become directions.
Relations live as offsets between points.

✱

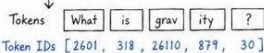


Appendix A: Transformer (Step-by-Step Procedure)

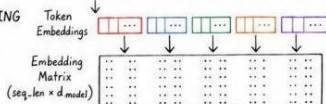
① INPUT



② TOKENIZER

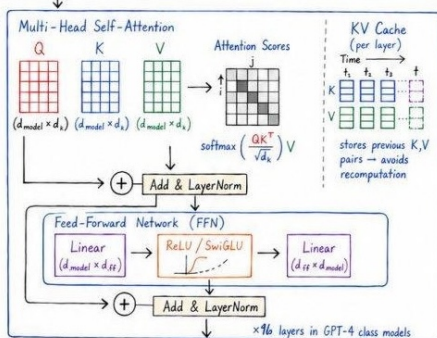


③ EMBEDDING LAYER



$d_{\text{model}} = 4096$
dimensional
vectors

④ TRANSFORMER BLOCK
($\times N$ layers)



Prefill phase:
process all input
tokens in parallel
(compute-bound)

Decode phase:
generate one
token at a time
(memory-bound)

KV Cache grows
linearly with
sequence length $\rightarrow$
main memory
bottleneck

FlashAttention:
fuses attention
ops to reduce
HBM reads

Appendix A: Transformer (Step-by-Step Procedure)

