Lecture 11

Demystifying Attention: A Unified Framework

Feedforward

Tokens: $\mathbf{x}_1, \ldots, \mathbf{x}_n$ with $\mathbf{x}_i \in \mathbb{R}^d$. Stack as

$$X = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d \times n}.$$

Linear map (fixed weights):

$$Y = WX, \quad W \in \mathbb{R}^{p \times d}, \quad \mathbf{y}_i = W\mathbf{x}_i.$$

Key observation: W is shared across all tokens.

Motivation: Why Rethink Feedforward?

Fixed weights ignore token–token relations and context.

$$Y = WX, \quad \mathbf{y}_i = W\mathbf{x}_i, \quad W \text{ fixed for all } i.$$

Goal: make the weights depend on the input sequence.

From Fixed Weights to Input-Dependent Weights

Replace constant W with input-conditioned $W(X)$ so that

$$Y = W(X)X, \quad \mathbf{y}_i = W(X)\mathbf{x}_i.$$

We want weights that vary with the output index i and depend on all tokens in X.

Why a Tensor is Needed

Each output token $\mathbf{y}_i$ needs its own weight matrix depending on all inputs:

$$\mathcal{W}(X) \in \mathbb{R}^{n \times n \times p \times d}.$$

Indices:

- i : output token index ($1..n$)
- j : input token index ($1..n$)
- last two axes: (p, d) = output/input feature dims

Slice notation: $\mathcal{W}_{ij} := \mathcal{W}[i, j, :, :] \in \mathbb{R}^{p \times d}$.

Weight Tensor: Definition

$$\mathcal{W}(X) \in \mathbb{R}^{n \times n \times p \times d},$$

$$\mathcal{W}_{ij} = \mathcal{W}[i, j, :, :] \in \mathbb{R}^{p \times d}.$$

$\mathcal{W}_{ij}$ maps features of input token $\mathbf{x}_j$ to features of output token $\mathbf{y}_i$. The full computation is

$$\mathbf{y}_i = \sum_{j=1}^{n} \mathcal{W}_{ij} \mathbf{x}_j.$$

The Unified Framework (Most General)

Using the 4D tensor $\mathcal{W}(X)$, we can express *all* sequence models as:

$$\mathbf{y}_i = \sum_{j=1}^{n} \mathcal{W}_{ij} \mathbf{x}_j$$

where $\mathcal{W}_{ij} \in \mathbb{R}^{p \times d}$ is a matrix mapping features of $\mathbf{x}_j$ to features of $\mathbf{y}_i$.

Challenge: Directly forming the full 4D tensor is infeasible ($\mathcal{O}(n^2pd)$ parameters).

Solution: Different models use different strategies for tractability.

The Unified Factorized Framework

Strategy 1 (Scalar Factorization): Factorize each matrix $\mathcal{W}_{ij}$ as:

$$\mathcal{W}_{ij} = w_{ij} V, \quad w_{ij} \in \mathbb{R}, \; V \in \mathbb{R}^{p \times d}.$$

This gives the **Unified Factorized Framework:**

$$\mathbf{y}_i = \sum_{j=1}^{n} w_{ij} V \mathbf{x}_j = \sum_{j=1}^{n} f_\theta(\mathbf{x}_i, \mathbf{x}_j) \, (V \mathbf{x}_j)$$

Matrix form:

$$Y = V X \, W(X)^\top$$

where $W(X) = [w_{ij}] \in \mathbb{R}^{n \times n}$ and $w_{ij} = f_\theta(\mathbf{x}_i, \mathbf{x}_j)$.

Covers: MLP, CNN, Attention, Linear Attention, KAN

Key constraint: Rank-1 factorization of each $\mathcal{W}_{ij}$ (scalar $\times$ shared matrix).

Token-Pair Weight Functions

In the **Unified Factorized Framework**, we need a function assigning a scalar to each pair:

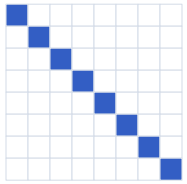
$$w_{ij} = f_\theta(\mathbf{x}_i, \mathbf{x}_j), \qquad f_\theta : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}.$$

Different f_θ define different models:

- MLP: δ_{ij} (no interaction)
- CNN: c_{i-j} (local, shift-invariant)
- Attention: $\mathrm{softmax}_j(\mathbf{q}_i^\top \mathbf{k}_j / \sqrt{d_k})$
- Linear Attn: $\boldsymbol{\phi}(\mathbf{x}_i)^\top \boldsymbol{\psi}(\mathbf{x}_j)$
- KAN: $\sum_m g_m(\mathbf{x}_i) h_m(\mathbf{x}_j)$ (separable basis)

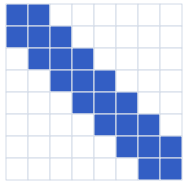
Note: RNN and SSM do *not* use scalar factorization. They use the full **Unified Framework** with matrices $\mathcal{W}_{ij} \in \mathbb{R}^{p \times d}$.

Visual: Weight Patterns in Factorized Framework



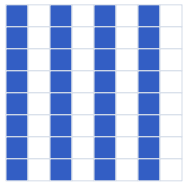
MLP $w_{ij} = \delta_{ij}$

No interaction between tokens.



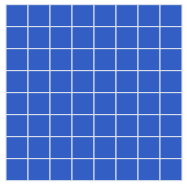
CNN $w_{ij} = c_{i-j}$

Local, shift-invariant connections.



KAN $w_{ij} = \varphi_{ij}(\mathbf{x}_j)$

Input-dependent weight functions.



Attention $w_{ij} = g(\mathbf{x}_i, \mathbf{x}_j)$

Every token can interact with every other token.

Each pattern shows the $n \times n$ matrix $W(X)$: output i vertically, input j horizontally. Dark cells indicate nonzero weights.

Two Frameworks: A Comparison

Unified Framework

Most general form: a full matrix for each token pair.

$$\mathbf{y}_i = \sum_j \mathcal{W}_{ij}\mathbf{x}_j, \qquad \mathcal{W}_{ij} \in \mathbb{R}^{p\times d}.$$

Models: MLP, CNN, Attention, Linear Attention, KAN, RNN, SSM.

Unified Factorized Framework

A constrained form with scalar weights and a shared projection.

$$\mathbf{y}_i = \sum_j w_{ij}(V\mathbf{x}_j), \qquad \mathcal{W}_{ij} = w_{ij}V, \quad w_{ij} \in \mathbb{R}.$$

Models: MLP, CNN, Attention, Linear Attention, KAN.

Key insight: Scalar factorization trades expressiveness for efficiency.

Strategy 2: Structured Dynamics (RNN/SSM)

Another strategy: use the full **Unified Framework** with structured weight matrices:

$$\mathbf{y}_i = \sum_{j=1}^{n} \mathcal{W}_{ij} \mathbf{x}_j, \quad \mathcal{W}_{ij} \in \mathbb{R}^{p \times d}$$

Instead of explicitly computing $\mathcal{W}_{ij}$, define them through **recurrent dynamics**:

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t), \quad \mathbf{y}_t = g(\mathbf{h}_t)$$

The weight matrices $\mathcal{W}_{ij}$ emerge implicitly from the dynamics:

- RNN: $\mathcal{W}_{ij} = \partial \mathbf{y}_i / \partial \mathbf{x}_j$ (depends on input sequence)
- SSM (linear): $\mathcal{W}_{ij} = C \bar{A}^{i-j} \bar{B}$ (can be precomputed!)

Key difference: Full matrices, not scalar factorization $\Rightarrow$ more expressive.

Two Types of Weight Computation

Explicit Weights: Weights are directly computed, stored in memory, then applied.

- Use **Unified Factorized Framework**: $Y = VX\,W(X)^\top$
- Computation: compute $W(X)$, then apply via matrix multiplication
- Examples: MLP, CNN, Attention, Linear Attention, KAN
- The framework describes the **actual algorithm**

Implicit Weights: Effective influence exists through sequential dynamics, weights never materialized.

- Use **Unified Framework**: $\mathbf{y}_i = \sum_j \mathcal{W}_{ij}\mathbf{x}_j$ with full matrices
- Forward pass uses recurrence: $\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$, $\mathbf{y}_t = g(\mathbf{h}_t)$
- Weights characterize information flow: $\mathcal{W}_{ij} = \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_j}$
- Examples: RNN, SSM (sequential mode)
- The framework describes **information flow pattern**, not actual computation

Feedforward (MLP)

Definition: $f_\theta(\mathbf{x}_i, \mathbf{x}_j) = \delta_{ij}$. Then

$$\mathbf{y}_i = \sum_j \delta_{ij} V \mathbf{x}_j = V \mathbf{x}_i.$$

No token interaction; W fixed.

CNN

Local, shift-invariant weights: $f_\theta(\mathbf{x}_i, \mathbf{x}_j) = c_{i-j}$ for $|i-j| \le r$.

$$\mathbf{y}_i = \sum_{|i-j| \le r} c_{i-j} V \mathbf{x}_j.$$

Change variable $k = i - j$, so $j = i - k$ and $|k| \le r$:

$$\mathbf{y}_i = \sum_{k=-r}^{r} c_k (V \mathbf{x}_{i-k}).$$

This matches the definition of a 1D discrete convolution of the sequence (VX) with kernel $[c_{-r}, \ldots, c_0, \ldots, c_r]$.

Attention: Standard Definition

Standard scaled dot–product attention:

$$Q = W_Q X, \quad K = W_K X, \quad V_{\text{val}} = W_V X.$$

The attention matrix is computed as

$$A = \text{SoftMax}\left(\frac{Q^\top K}{\sqrt{d_k}}\right) \in \mathbb{R}^{n \times n},$$

where SoftMax is applied row-wise. The output is

$$Y = V_{\text{val}} A^\top.$$

Attention: Equivalence to Unified Factorized Framework

Define queries and keys: $\mathbf{q}_i = W_Q \mathbf{x}_i,\ \mathbf{k}_j = W_K \mathbf{x}_j$.

Set the token-pair weight function

$$f_\theta(\mathbf{x}_i, \mathbf{x}_j) = \mathrm{softmax}_j\left(\frac{\mathbf{q}_i^\top \mathbf{k}_j}{\sqrt{d_k}}\right).$$

With $A = [a_{ij}], \quad a_{ij} = f_\theta(\mathbf{x}_i, \mathbf{x}_j)$, the Unified Factorized Framework gives

$$Y = VXA^\top.$$

Since $V = W_V$ and $V_{\mathrm{val}} = W_V X = VX$, this is

$$Y = V_{\mathrm{val}} A^\top,$$

which is exactly the standard scaled dot-product attention formula.

Linear Attention: Standard Definition

Replace the softmax with kernel approximation using feature maps $\phi, \psi : \mathbb{R}^d \to \mathbb{R}^r$ where $r \ll n$:

$$Q = W_Q X, \quad K = W_K X, \quad V_{\text{val}} = W_V X.$$

Define feature-mapped queries and keys:

$$\Phi = [\phi(\mathbf{q}_1), \ldots, \phi(\mathbf{q}_n)] \in \mathbb{R}^{r \times n},$$

$$\Psi = [\psi(\mathbf{k}_1), \ldots, \psi(\mathbf{k}_n)] \in \mathbb{R}^{r \times n}.$$

The output is

$$Y = V_{\text{val}}(\Phi^\top \Psi)^\top = V_{\text{val}} \Psi^\top \Phi.$$

Complexity: $\mathcal{O}(nr)$ instead of $\mathcal{O}(n^2)$.

Linear Attention in the Unified Factorized Framework

For each token pair, define the weight function as the inner product of feature maps:

$$f_\theta(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\phi}(\mathbf{q}_i)^\top \boldsymbol{\psi}(\mathbf{k}_j) = \boldsymbol{\phi}(W_Q \mathbf{x}_i)^\top \boldsymbol{\psi}(W_K \mathbf{x}_j).$$

In the Unified Factorized Framework this gives

$$\mathbf{y}_i = \sum_{j=1}^{n} f_\theta(\mathbf{x}_i, \mathbf{x}_j)\, V \mathbf{x}_j = \sum_{j=1}^{n} \boldsymbol{\phi}(\mathbf{q}_i)^\top \boldsymbol{\psi}(\mathbf{k}_j)\, V \mathbf{x}_j.$$

Matrix form: with $\Phi = [\boldsymbol{\phi}(\mathbf{q}_1), \ldots, \boldsymbol{\phi}(\mathbf{q}_n)]$ and $\Psi = [\boldsymbol{\psi}(\mathbf{k}_1), \ldots, \boldsymbol{\psi}(\mathbf{k}_n)]$, the weight matrix is $W(X) = \Phi^\top \Psi$. From the Unified Factorized Framework:

$$Y = V X\, W(X)^\top = V X\, (\Phi^\top \Psi)^\top = V X\, \Psi^\top \Phi = V_{\text{val}}\, \Psi^\top \Phi.$$

This matches exactly the standard linear attention formula.

Linear Attention: Efficient Computation

The key insight is to reorder the summation. Starting from

$$\mathbf{y}_i = \sum_{j=1}^{n} \boldsymbol{\phi}(\mathbf{q}_i)^\top \boldsymbol{\psi}(\mathbf{k}_j)\, V\mathbf{x}_j,$$

factor out $\boldsymbol{\phi}(\mathbf{q}_i)$:

$$\mathbf{y}_i = \boldsymbol{\phi}(\mathbf{q}_i)^\top \left(\sum_{j=1}^{n} \boldsymbol{\psi}(\mathbf{k}_j)\, (V\mathbf{x}_j)^\top \right).$$

Linear Attention: Summary Matrix

Precompute the summary matrix

$$S = \sum_{j=1}^{n} \boldsymbol{\psi}(\mathbf{k}_j)\,(V\mathbf{x}_j)^\top \in \mathbb{R}^{r\times p}.$$

Then $\mathbf{y}_i = \boldsymbol{\phi}(\mathbf{q}_i)^\top S \in \mathbb{R}^p$ with cost $\mathcal{O}(nr + rp)$ instead of $\mathcal{O}(n^2 p)$.

Kolmogorov–Arnold Networks (KAN): Standard Definition

Based on the Kolmogorov–Arnold representation theorem, approximate any multivariate function using a sum of univariate functions.

For token interactions, use separable nonlinear basis expansion with $m_{\max}$ basis functions:

$$g_m : \mathbb{R}^d \to \mathbb{R}, \quad h_m : \mathbb{R}^d \to \mathbb{R}, \quad m = 1, \ldots, m_{\max}.$$

Each basis pair (g_m, h_m) captures a different mode of interaction between tokens.

KAN in the Unified Factorized Framework

Define the token-pair weight as a sum of separable basis functions:

$$f_\theta(\mathbf{x}_i, \mathbf{x}_j) = \sum_{m=1}^{m_{\max}} g_m(\mathbf{x}_i)\, h_m(\mathbf{x}_j).$$

Substituting into the unified framework:

$$\mathbf{y}_i = \sum_{j=1}^{n} f_\theta(\mathbf{x}_i, \mathbf{x}_j)\, V\mathbf{x}_j = \sum_{j=1}^{n} \sum_{m=1}^{m_{\max}} g_m(\mathbf{x}_i)\, h_m(\mathbf{x}_j)\, V\mathbf{x}_j.$$

KAN: Efficient Computation

Reorder the summation to separate token-dependent and sequence-dependent terms:

$$\mathbf{y}_i = \sum_{j=1}^{n} \sum_{m=1}^{m_{\max}} g_m(\mathbf{x}_i)\, h_m(\mathbf{x}_j)\, V\mathbf{x}_j = \sum_{m=1}^{m_{\max}} g_m(\mathbf{x}_i) \left(\sum_{j=1}^{n} h_m(\mathbf{x}_j)\, V\mathbf{x}_j \right).$$

KAN: Content Channels

Precompute the $m_{\max}$ content channels:

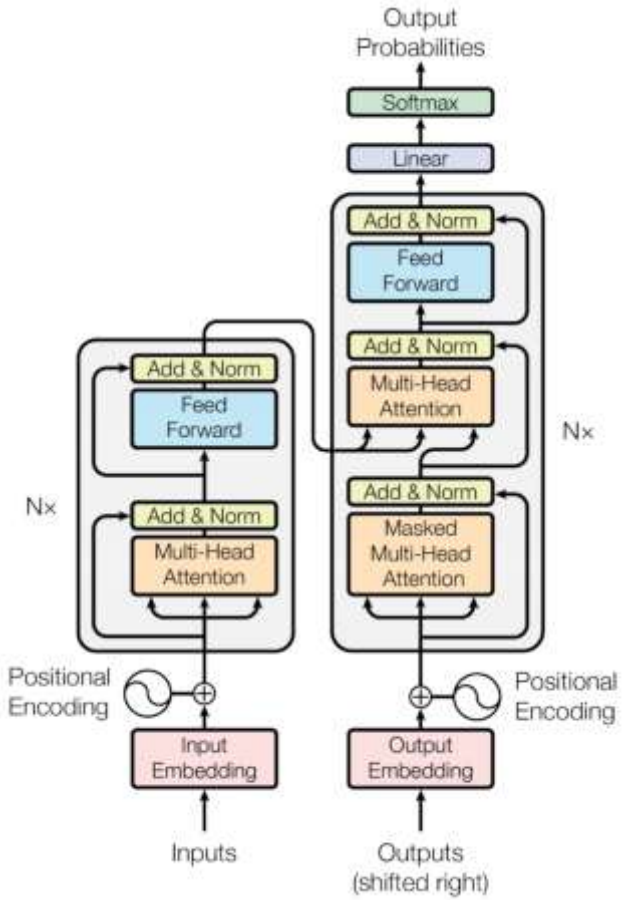
$$\mathbf{c}_m = \sum_{j=1}^{n} h_m(\mathbf{x}_j)\, V\mathbf{x}_j \in \mathbb{R}^p, \quad m = 1, \ldots, m_{\max}.$$

Then

$$\mathbf{y}_i = \sum_{m=1}^{m_{\max}} g_m(\mathbf{x}_i)\, \mathbf{c}_m.$$

Each output token is a weighted combination of $m_{\max}$ content channels, with weights $g_m(\mathbf{x}_i)$ depending on the query token.

BERT, GPT


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Generative Pre-trained Transformer (GPT)

- Improving Language Understanding by Generative Pre-Training (2018)

Alec Radford

Karthik Narasimhan

Tim Salimans

Ilya Sutskever

Bidirectional Encoder Representations from Transformers (BERT)

- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2018)

Jacob Devlin

Ming-Wei Chang

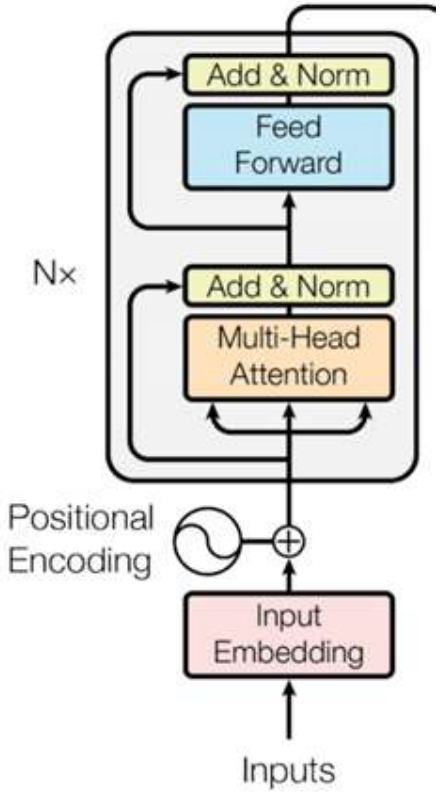
Kenton Lee

Kristina Toutanova

BERT and GPT

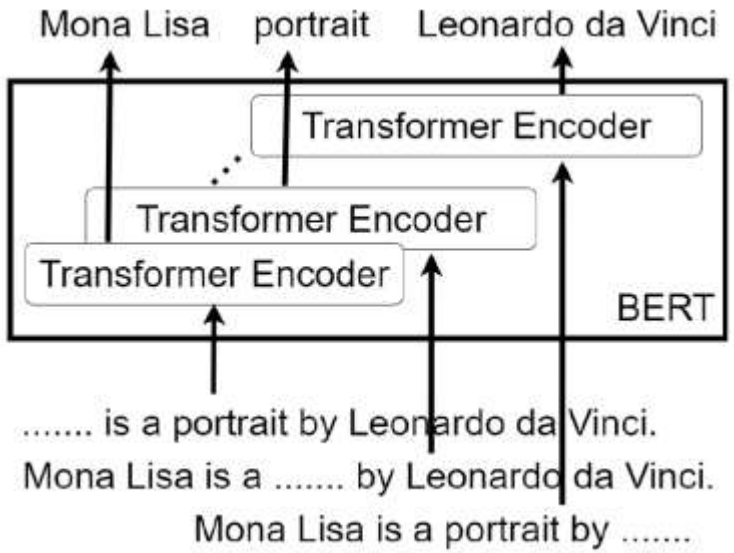
- The GPT is built using transformer decoder blocks.
- BERT is built using transformer encoder blocks.

BERT


Add & Norm
Feed Forward
Nx
Add & Norm
Multi-Head Attention
Positional Encoding
Input Embedding
Inputs

Masked language model

- Masks words in the input and asks the model to predict the missing word.



BERT: Bidirectional language model

- Masks words in the input and asks the model to predict the missing word.

- Additional task: Given two sentences (A and B), is B likely to be the sentence that follows A, or not?

BERT

- BERT is designed to pretrain bidirectional representations from unlabeled text.

BERT

- BERT is designed to pretrain bidirectional representations from unlabeled text.

- Jointly conditioning on both left and right context.

BERT

- BERT is designed to pretrain bidirectional representations from unlabeled text.
- Jointly conditioning on both left and right context.
- The pre-trained BERT model can be finetuned with just one additional output layer.

BERT

- BERT is designed to pretrain bidirectional representations from unlabeled text.
- Jointly conditioning on both left and right context.
- The pre-trained BERT model can be finetuned with just one additional output layer.
- It creates state-of-the-art models for a wide range of tasks, such as question answering and language inference.

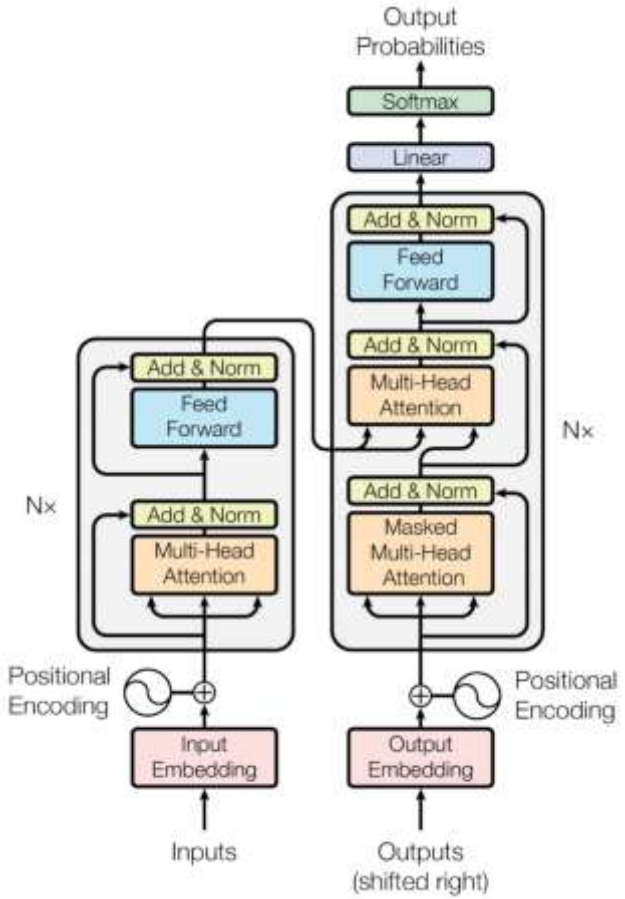
[CLS] Token in BERT

- The [CLS] token is prepended to the input text and travels through the Transformer layers alongside other tokens.
- All tokens, including [CLS], gather contextual information from the entire sequence due to the self-attention mechanism.
- For sentence-level tasks, the final hidden state of the [CLS] token is used as the sentence representation.
- During fine-tuning on a specific task, the model learns to imbue the [CLS] token with a meaningful representation of the entire sentence, optimized for that task.
- Example Usage: In classification tasks, the [CLS] token representation is fed into a classifier to determine the sentence's class.

BERT

BERT is basically a trained Transformer Encoder stack

1. **Transformer**:
    1. Encoder Layers: 6
    2. FFNN Hidden Layer Units: 512
    3. Attention Heads: 8
2. **BERT Base**:
    1. Encoder Layers: 12
    2. FFNN Hidden Layer Units: 768
    3. Attention Heads: 12
    4. Total Parameters: 110 million
3. **BERT Large**:
    1. Encoder Layers: 24
    2. FFNN Hidden Layer Units: 1024
    3. Attention Heads: 16
    4. Total Parameters: 340 million


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

BERT

- **RoBERTa**:
  - Optimizes BERT's training process by using more data, larger batch sizes, and longer training times, resulting in improved performance on NLP tasks.
- **TinyBERT**:
  - A smaller and faster version of BERT designed for resource-constrained environments, retaining competitive performance with significantly fewer parameters.

BERT

1. **Multilingual BERT**:
    1. Trained on 104 different languages, capable of "zero-shot" adaptation to a new language domain.
2. **Domain Specific BERT Variants**:
    1. BioBERT: Retrained on a biomedical corpus.
    2. SciBERT: Trained on over one million published articles.
    3. BERTweet: A RoBERTa model trained on 850 million tweets.
    4. FinBERT: Adapted to the financial domain.

Modern Encoders and Sentence Embeddings

ModernBERT (2024): 149-million and 395-million-parameter encoders; context length up to 8,192 tokens.

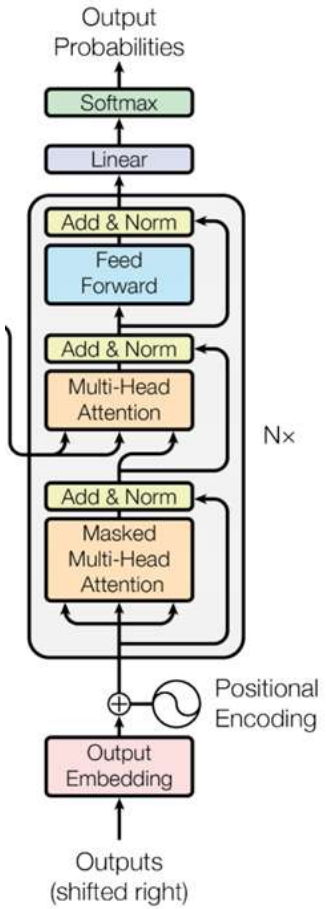
Updates include rotary position embeddings, alternating local/global attention, and efficient training.

Sentence-BERT trains sentence representations for semantic similarity and retrieval.

A raw BERT [CLS] representation is not automatically a good sentence embedding; training and pooling matter.

Sources: Warner et al., ModernBERT, arXiv:2412.13663; Reimers & Gurevych, Sentence-BERT (2019).

GPT


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Positional Encoding
Output Embedding
Outputs (shifted right)

GPT

Predict the next word, given all of the previous words

- **Architecture**:
  - Stack of Transformer decoder blocks.
  - No encoder, hence no cross-attention module.
  - Components: Positional encoding, masked multihead self-attention, and feedforward network.
- **Directionality**:
  - Only considers previous (left) words in attention, not bidirectional like BERT.
  - Utilizes masked multihead self-attention for this purpose.

GPT 1

- Released: 2018
- Parameters: 117 Million
- Layers: 12
- Training Data: Books1 Corpus (7,000 books)
- Focus: Unsupervised pre-training, Transformer architecture, large-scale language modeling

GPT 2

- Released: 2019
- Parameters: ~1.5 Billion
- Layers: 48
- Training Data: 40GB (English)
- Focus: Transformer architecture, self-attention mechanism

GPT 3

- Released: 2020
- Parameters: 175 Billion
- Layers: 175
- Training Data: 570GB (Multilingual)
- Focus: Few-shot learning, prompt engineering, Python support

GPT 4

- **Release**: March 2023 (ChatGPT Plus, Microsoft Copilot)
- **Parameters**: ~1.76 Trillion (rumored, Mixture of Experts)
- **Layers**: Unknown
- **Training Data**: More diverse, larger scale
- **Focus**: Multimodal (text and image inputs), improved few-shot learning, reasoning, and NLU/NLG

GPT-4o and o1

GPT-4o (May 2024): “o” means omni; introduced an end-to-end multimodal model spanning text, vision, and audio.

OpenAI o1-preview (September 2024): a separate reasoning model family—not “GPT-4o1.”

o1 uses reinforcement learning and additional inference-time computation for difficult tasks.

Distinguish model capabilities from the modalities and features exposed by a particular product or API.

Sources: OpenAI, Hello GPT-4o (May 13, 2024); Introducing o1-preview (September 12, 2024).

The GPT Family After GPT-4

GPT-4.1 (April 2025): coding, instruction following, and long-context processing.

GPT-5 (August 2025): a routed system combining fast responses with deeper reasoning.

Sources: OpenAI GPT-4.1 and GPT-5 announcements; OpenAI API changelog, October 26, 2026.

GPT-6 Family

GPT-6.1 Terra (October 26, 2026): emphasizes visual reasoning, computer use, and writing.

GPT-6 Astra, GPT-6 Luna, GPT-6.1 Sol, and GPT-6.1 Terra.

Other Major Model Families

Claude: Opus 5.5 (September 22), Sonnet 5.5 (September 28), and Haiku 5.5 (October 7), 2026.

Gemini 4 Argon became generally available on October 13, 2026; Gemini 3.8 Flash offers another capability–speed trade-off.

Sources: Anthropic model announcements; Google Gemini API release notes. Checked October 26, 2026.

Open-Weight Models: Published Examples

Llama 3.1 (2024): dense models with 8, 70, and 405 billion parameters.

DeepSeek-V3 (2024): mixture of experts; 671 billion total and 37 billion active parameters per token.

Qwen3-235B-A22B (2025): 235 billion total and 22 billion active parameters per token.

Open weights enable local use and adaptation; they do not automatically imply open training data or unrestricted licensing.

Sources: Meta, The Llama 3 Herd of Models; DeepSeek-V3 Technical Report; Qwen3 Technical Report.

Introduction to T5

- Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer by Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J. Liu.

- T5 stands for Text-to-Text Transfer Transformer.
- Introduced as a unified framework for addressing NLP tasks by converting them into a text-to-text format.
- Key Focus: Transfer learning - Pre-training on data-rich tasks followed by fine-tuning on downstream tasks.

T5 Architecture

- Operates on an encoder-decoder model.

- Pretrained on a multi-task mixture of both unsupervised and supervised tasks, each converted into a text-to-text format.

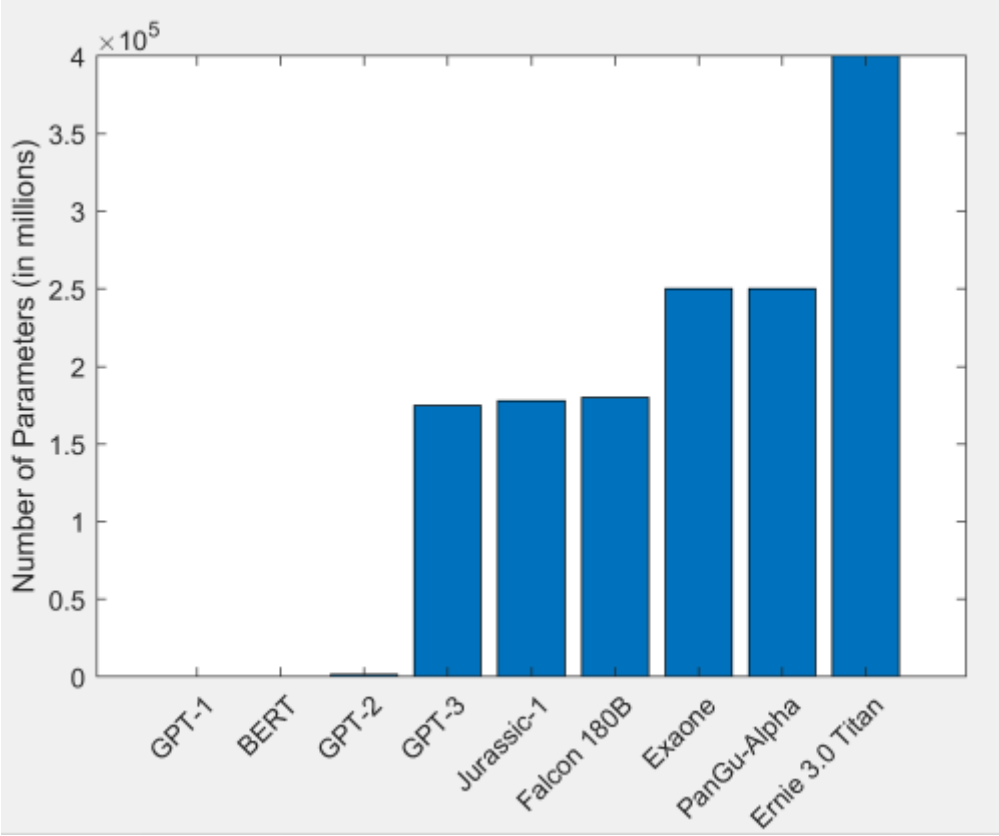
Pretraining Objectives

- Supervised training conducted on downstream tasks provided by GLUE and SuperGLUE benchmarks, reformulated into text-to-text tasks.
- Self-supervised training employs corrupted tokens: 15% of tokens are randomly removed and replaced with sentinel tokens.

Task Adaptation

- T5 adapts to various tasks by prepending task-specific prefixes to the input, e.g., for translation: "translate English to German: …", for summarization: "summarize: …".
- Achieves very good results on many benchmarks including summarization, question answering, and text classification.

Number of Parameters in Various Language Models



Number of Parameters in Various Language Models

