GPT-2: A Deep Dive into Language Modeling

Understanding Transformers and GPT-2 Architecture, Training, and Applications

Presented to: Prof.Alberto Castellini
Presenter: Abel Abebe Bzuayene
ID: VR510555

June 21, 2025

Outline

Language Models: Overview

History

- 1990s: N-grams, statistical predictions. [3]
- 2010s: RNNs, LSTMs for sequences. [8]
- 2017: Transformers, attention-based. [1]

Applications

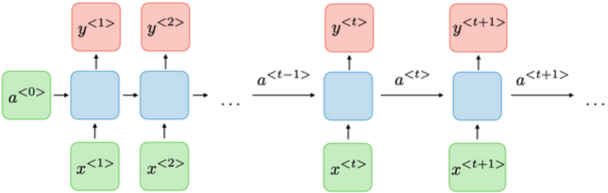
- Chatbots (e.g., customer service).
- Translation for multilingual tasks.
- Summarization for news, research.

N-gram Models

- **History**: 1990s, used statistical probabilities for text prediction. [3]
- **How It Works**: Predicts $P(w_t|w_{t-n+1}, \ldots, w_{t-1})$ based on $n-1$ prior words.
- **Example**: Trigram ($n = 3$) predicts "is" given "The sky" with $P(\text{is}|\text{The sky})$.
- **Drawbacks**: Limited to short context (n); sparse data for large n.

Recurrent Neural Networks (RNNs)

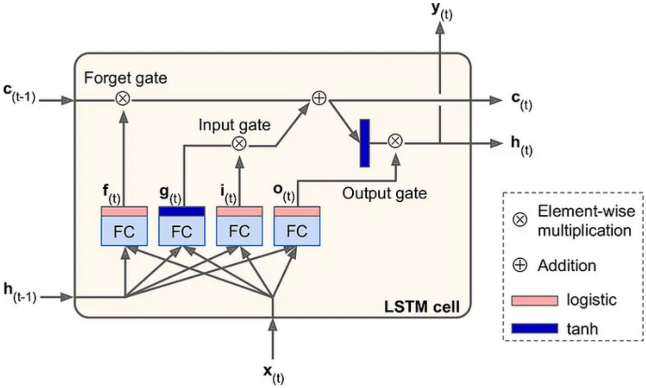
- **History**: 2010s, neural models for sequential data. [8]
- **How It Works**: Hidden state $h_t = f(h_{t-1}, x_t)$ processes tokens sequentially.
- **Example**: Predicts "blue" for "The sky is" using prior hidden states.
- **Drawbacks**: Vanishing gradients limit long dependencies; slow sequential training.



Source: stanford.edu/ shervine - CS230 RNN Cheat Sheet

Long Short-Term Memory (LSTMs)

- **History**: RNNs with memory cells enhanced in the mid-2010s. [2]
- **How It Works**: Gates (Forget, Input, Output) manage long-term dependencies and update the cell state.
- **Example**: Predicts 'meows' for 'The cat sat and'
- **Drawbacks**: Sequential processing; high computational cost.



Source: *Medium - Anish Nama*

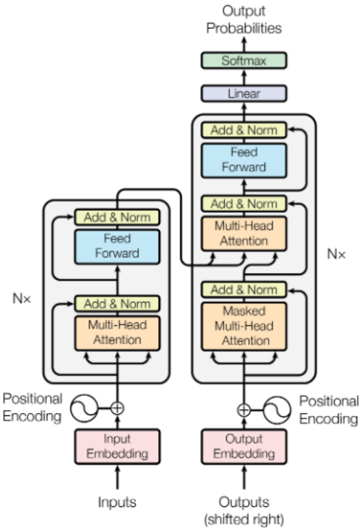
Transformers

- **History**: 2017, introduced by Vaswani et al. [1]
- **How It Works**: Self-attention $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$ processes all tokens in parallel.
- **Example**: Predicts "sat" for "The cat", attending to all prior tokens.
- **Drawbacks**: High memory usage; complex training.

Transformer Overview

- Introduced in *"Attention is All You Need"* (Vaswani et al., 2017), revolutionizing sequence modeling with self-attention.
- Processes entire sequences *in parallel* using self-attention, unlike RNNs that process sequentially.
- **Architecture:** Consists of an embedding layer, followed by encoder and decoder stacks for full sequence modeling.
- **Token Embeddings:** Input words are converted into fixed-size continuous vectors that capture semantic meaning.
- **Positional Encodings:** Inject position information into token embeddings to help the model learn order, as transformers have no recurrence.
- **Encoder:** Contains multi-head self-attention layers followed by position-wise feed-forward networks. Layer normalization and residual connections are used for training stability.
- **Decoder:** Adds masked self-attention to prevent future information leakage (causality), plus encoder-decoder attention to incorporate encoded input.

Transformer architecture



Source: Vaswani et al., Attention is All You Need (2017)

Token and Positional Embeddings

Token Embedding

- Converts input tokens (words or subwords) into continuous vector representations.
- Essential for neural networks to process raw text.
- Example: "cat" $\rightarrow [0.25, -0.5, 0.75, \ldots]$.

Positional Embedding

- Adds sequence order information to tokens.
- Uses fixed sinusoidal functions or learned embeddings.
- Example: Positional vector added to token embedding to account for position in sequence.

Self-Attention Mechanism

- Self-attention allows a model to weigh the importance of each token in a sequence relative to others, facilitating contextual understanding.
- Computes attention scores for each token pair.
- Formula:

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

- Q, K, V are derived from input embeddings.

Given an input sequence of tokens, each token is mapped to a vector of dimension d_{model}. Self-attention computes:

- **Queries** (Q): Vectors representing token questions, derived via $Q = XW^Q$, where X is the input matrix and $W^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$.
- **Keys** (K): Vectors for matching queries, computed as $K = XW^K$, with $W^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$.
- **Values** (V): Vectors containing token information, given by $V = XW^V$, where $W^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$.

These matrices enable the model to compare tokens and extract relevant features.
The self-attention output is computed as:

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

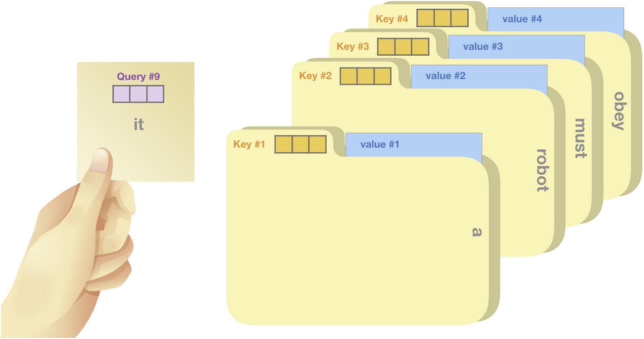
Here, QK^T produces a score matrix indicating token relationships, scaled by $\sqrt{d_k}$ to stabilize gradients. The softmax normalizes scores, and the result weights V to produce context-aware outputs. This mechanism allows the model to focus on relevant tokens.

Understanding Query, Key, and Value in Attention

Analogy: Think of a library search.

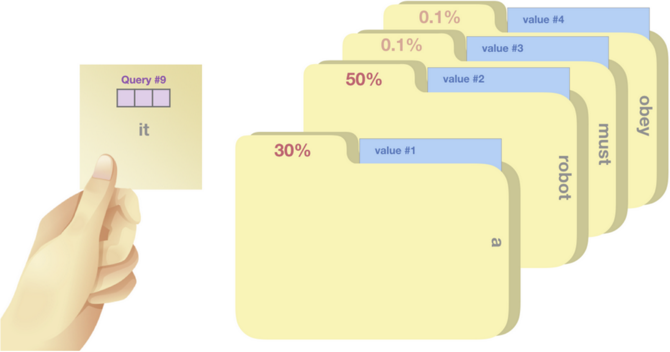
- **Query:** Like a search term you type into a catalog — it represents what you're looking for. In transformers, the query is the current token's representation used to find relevant information.
- **Key:** Like the tags or titles of all the books in the library. Each token has a key that helps determine how relevant it is to a given query.
- **Value:** Like the actual content of the books. Once the relevance (match between query and key) is calculated, we use the value to construct the final representation of the current word.

Query, Key, and Value Diagram



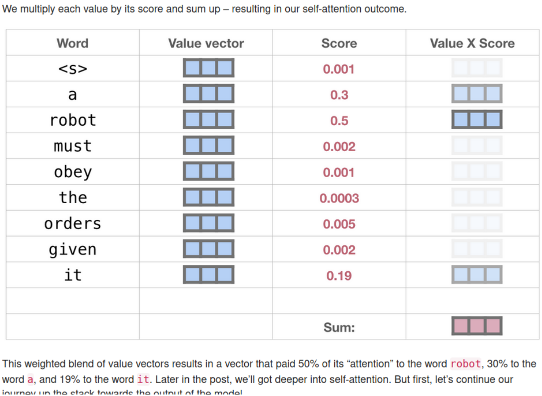
Visual representation of the attention mechanism using Query, Key, and Value.
Source: Jay Alammar, Illustrated GPT-2 (2019)

Score after applying softmax



Source: Jay Alammar, Illustrated GPT-2 (2019)

Attention Output Interpretation


We multiply each value by its score and sum up – resulting in our self-attention outcome.

Word	Value vector	Score	Value X Score
<s>		0.001	
a		0.3	
robot		0.5	
must		0.002	
obey		0.001	
the		0.0003	
orders		0.005	
given		0.002	
it		0.19	
		Sum:	

This weighted blend of value vectors results in a vector that paid 50% of its “attention” to the word `robot`, 30% to the word `a`, and 19% to the word `it`. Later in the post, we'll got deeper into self-attention. But first, let's continue our journey up the stack towards the output of the model

This weighted blend of value vectors results in a vector that paid **50% of its “attention” to the word** `robot`, **30% to the word** `a`, and **19% to the word** `it`.

Source: Jay Alammar, Illustrated GPT-2 (2019)

Self-Attention: Step-by-Step

Input: `["I", "am", "GPT"]` (using 2D vectors)

Vectors:

- `"I"`: Q = [1, 0], K = [1, 0], V = [1, 2]
- `"am"`: Q = [0, 1], K = [0, 1], V = [2, 1]
- `"GPT"`: Q = [1, 1], K = [1, 1], V = [0, 1]

Step 1: Dot Product Scores (for "I")

$$\begin{aligned} \text{score(I, I)} &= [1,0] \cdot [1,0] = 1 \\ \text{score(I, am)} &= [1,0] \cdot [0,1] = 0 \\ \text{score(I, GPT)} &= [1,0] \cdot [1,1] = 1 \end{aligned}$$

Step 2: Softmax

$$\text{softmax}([1,0,1]) = [0.422, 0.155, 0.422]$$

Step-by-Step (continued)

Step 3: Weighted Sum of Vectors (for "I")

$$\begin{aligned} \text{output}_I &= 0.422 \cdot [1, 2] + 0.155 \cdot [2, 1] + 0.422 \cdot [0, 1] \\ &= [0.422, 0.844] + [0.31, 0.155] + [0.0, 0.422] \\ &= [0.732, 1.421] \end{aligned}$$

Summary of Self-Attention

1. Compute Q, K, V vectors
2. Scores $= Q \cdot K^T$
3. Softmax the scores
4. Multiply each V by its score
5. Sum the weighted vectors $\rightarrow$ Output

Introduction to Multi-Head Attention

- Multi-head attention allows multiple attention mechanisms (heads) to run in parallel.
- Each head captures different parts or relationships within the input.
- Key formula:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

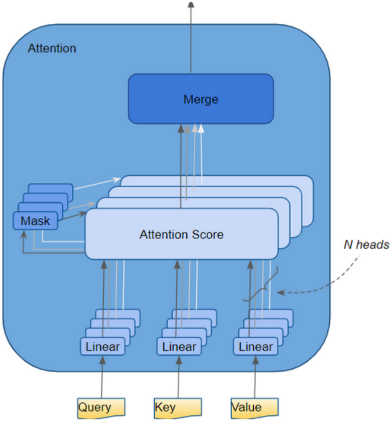
$$\text{where each head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Difference from Single-Head Attention

- In single-head attention, only one attention mechanism is used to compute the output.
- Multi-head attention allows the model to attend to different parts of the sequence simultaneously.
- **Single-head attention**: Limited by focusing on one relationship.
- **Multi-head attention**: Captures diverse relationships (long-range, local, syntax, semantics).

Simple Example with N Heads

- **Input**: A sequence of tokens {Q, K, V} (query, key, value).
- Head 1: Focuses on long-range dependencies (e.g., understanding a sentence structure).
- Head 2: Focuses on local context (e.g., understanding adjacent words).
- Each head calculates attention independently, then their outputs are concatenated and weighted by W^O.



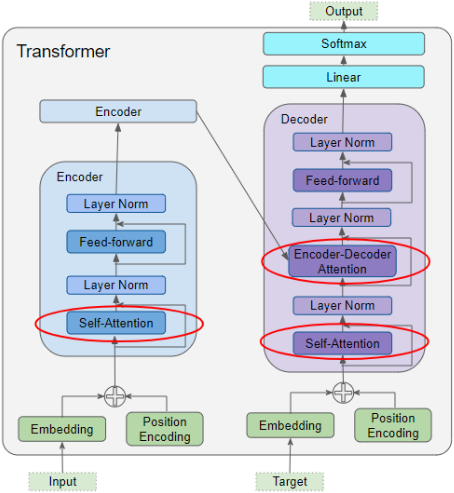
Source: Towards Data Science,

Effect of Small vs Large Number of Heads

- **Small Number of Heads**:
  - Less diversity in captured relationships.
  - Might miss important dependencies, especially long-range ones.
- **Large Number of Heads**:
  - Can capture more complex relationships, improving model performance.
  - Increases computation and memory usage.
  - Beyond a certain number of heads, returns diminish.

Attention Variants

- Self-attention: Tokens attend to each other in the same sequence.
- Cross-attention: Decoder attends to encoder outputs(key and value comes from encoder and query from decoder).
- Causal attention: Restricts attention to previous tokens.



Source: *Towards Data Science, Multi-Head Attention Deep Dive*

Why Transformers?

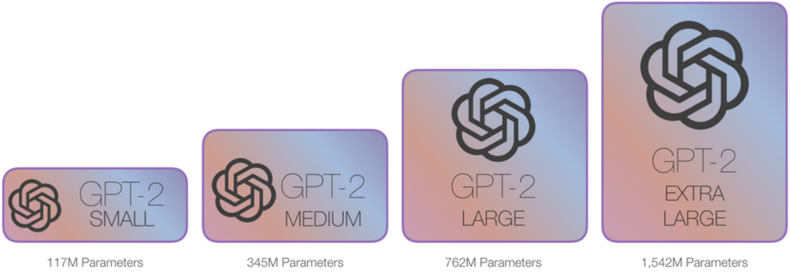
- Parallel processing speeds up training.
- Long-range dependencies handled effectively.
- Foundation for models like BERT, GPT, and T5.

Introduction to GPT-2

- Developed by OpenAI (2019), Generative Pre-trained Transformer 2 (GPT-2).
- Decoder-only model trained on $\sim$ 40GB of internet text (WebText).
- Objective: Next-token prediction with zero-shot performance on tasks like question answering and translation.
- Excels in unsupervised multitask learning.

GPT-2 Variants

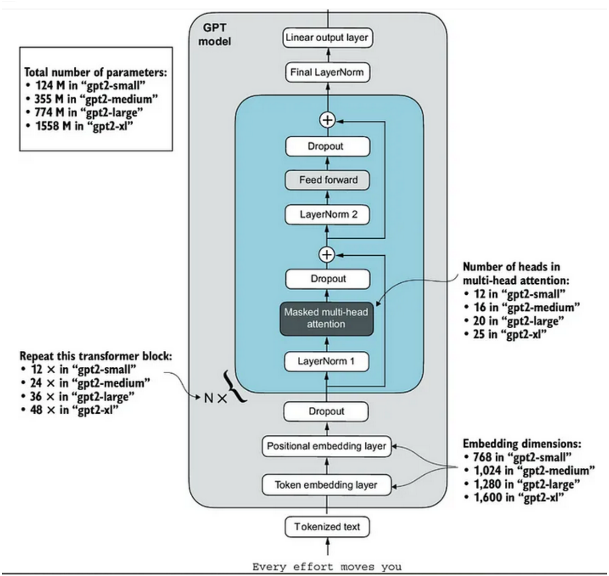
- **Small**: 124M parameters, 12 layers, 768 embedding size, 12 heads.
- **Medium**: 345M parameters, 24 layers, 1024 embedding size, 16 heads.
- **Large**: 774M parameters, 36 layers, 1280 embedding size, 20 heads.
- **XL**: 1.5B parameters, 48 layers, 1600 embedding size, 25 heads.



Source: Jay Alammar, Illustrated GPT-2 (2019)

GPT-2 Architecture

- Stacked transformer decoder layers.
- Components: Embeddings, transformer blocks(Self-attention,feed-forward network and normalization), final normalization, output head.
- Input: Tokenized text; Output: Vocabulary logits.
- Model depth is determined by the number of layers (n_{layer}).
- GPT-2 variants range from 12 to 48 layers.
- Deeper models capture more complex patterns but require more compute.
- Residual connections for stable training.
  Layer Structure:
- Two sub-layers: Attention and feed-forward.
- Layer normalization before each sub-layer.
- Residual connection: $x \leftarrow x + \text{sublayer}(x)$.



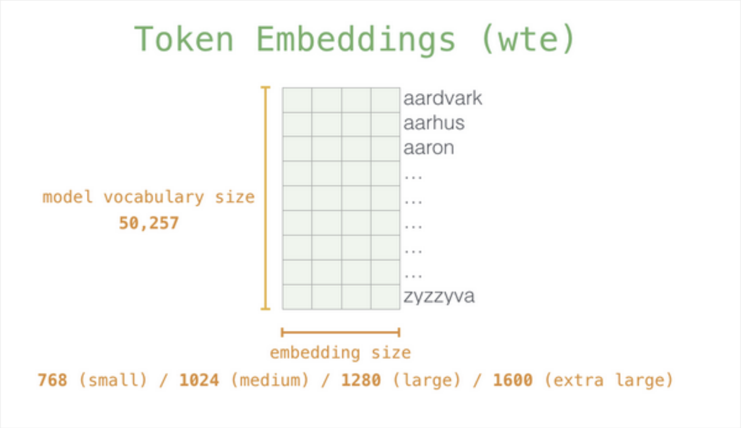
Source: Sebastian Raschka, 2024 – Building Large Language Models

Token Embeddings

- Maps tokens to dense vectors of size n_{embd}.
- Purpose: Encodes semantic meaning of tokens.
- Example: Vocabulary of 50257 tokens (GPT-2 tokenizer).
- Input Text: Transformers are powerful models for NLP tasks.
- Tokens: ['Transform', 'ers', ' are', ' powerful', ' models', ' for', ' NL', 'P', ' tasks', '.']
- Token IDs: [12110, 3938, 389, 12415, 1937, 329, 3389, 76, 4593, 13]

Token Embeddings: Role

- Converts discrete tokens into continuous representations.
- Enables the model to learn relationships between words.
- Size n_{embd} (e.g., 768 for GPT-2 Small) balances expressiveness and compute. *Source: Jay Alammar, Illustrated GPT-2 (2019)*



Each row is a word embedding: a list of numbers representing a word and capturing some of its meaning. The size of that list is different in different GPT2 model sizes. The smallest model uses an embedding size of 768 per word/token.

Positional Embeddings: Concepts and Importance

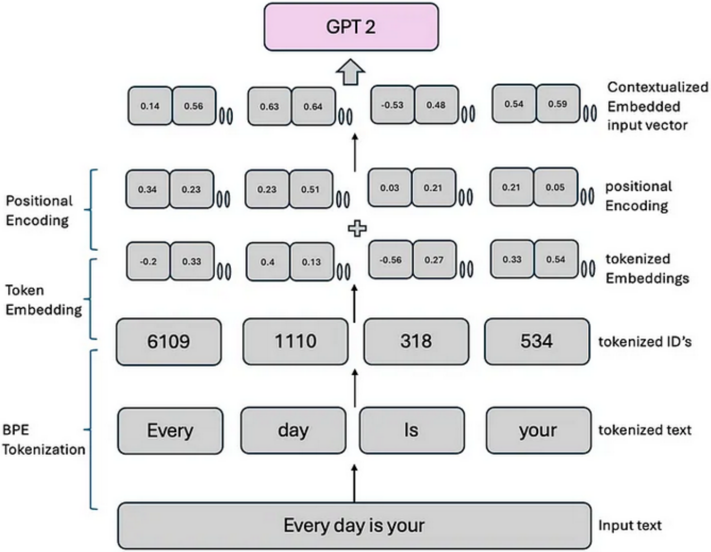
Positional Embeddings

- Encodes token positions in the sequence.
- Necessary because transformers lack inherent order.
- Learned embeddings up to *block size* (e.g., 1024).

Importance

- Ensures model distinguishes word order (e.g., “The cat” vs. “Cat the”).
- Supports fixed-length contexts up to *block size*.
- Combined with token embeddings via addition.

Input Text → BPE → TE → PE



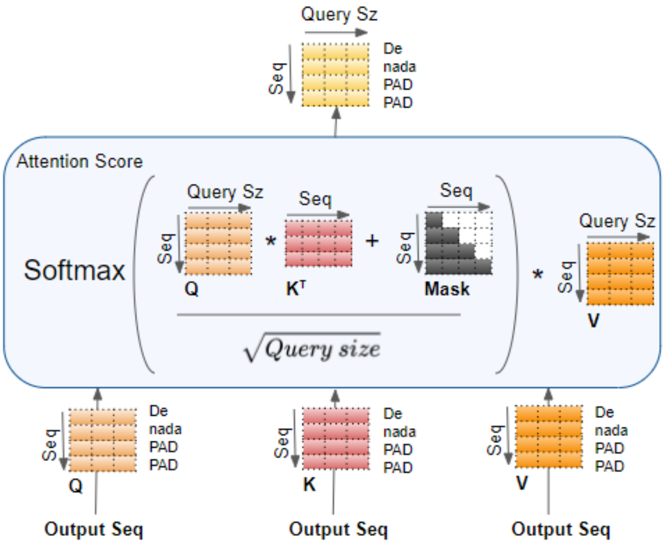
Source: Vipul Koti, Medium (2024)

Causal Self-Attention

- Attention restricted to previous tokens (autoregressive) or Computes attention scores for tokens up to position t.
- Ensures model predicts next token without seeing future ones.
- Uses masking to enforce causality.
- Scaled dot-product attention with mask:

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

- M masks future tokens with $-\infty$.

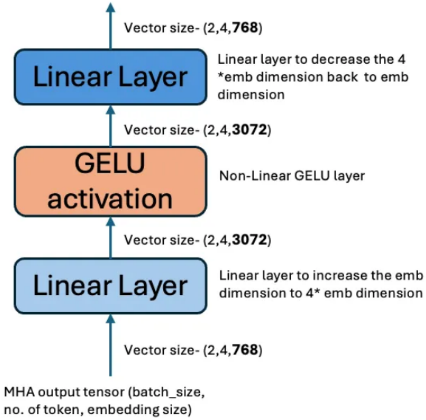


Source:

`Towards Data Science`

Feed-Forward Network

- Applies position-wise transformations to each token.
- Two linear layers with GELU activation(to introduces non-linearity.).
- Expands to $4 \times n_{\text{embd}}$ to increases expressiveness then projects back.
- Enhances model capacity for complex patterns.



Source: Vipul Koti, Medium (2024)

Layer Normalization

- Normalizes activations across embedding dimensions.
- Stabilizes training and improves gradient flow.
- Applied before attention and feed-forward layers.
- Reduces internal covariate shift.
- Formula: $\mathrm{LN}(x) = \frac{x-\mu}{\sigma} \cdot \gamma + \beta$.
- γ, β are learned parameters.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

Example: $x = [1.0, 2.0, 3.0, 4.0]$
$\mu = 2.5, \quad \sigma^2 = 1.25$

$$\hat{x} = \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}} \approx [-1.34, -0.45, 0.45, 1.34]$$

Learnable: $\gamma = [1, 1.5, 0.5, 2]$, $\beta = [0, 0.5, 1, -1]$

$$y = \gamma \cdot \hat{x} + \beta = [-1.34, -0.17, 1.22, 1.68]$$

Why LayerNorm is Essential in Transformers

Without LayerNorm:

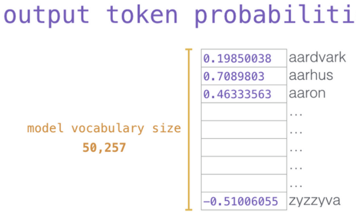
- Deep transformer stacks blow up (activations get huge or vanish).
- Training becomes unstable.
- Convergence is much slower.

With LayerNorm:

- Stable, predictable activations.
- Each token's features stay well-behaved.

Language Modeling Head

- Linear layer mapping to vocabulary size (50257).
- Shares weights with token embeddings for efficiency.
- Outputs logits for next-token prediction.



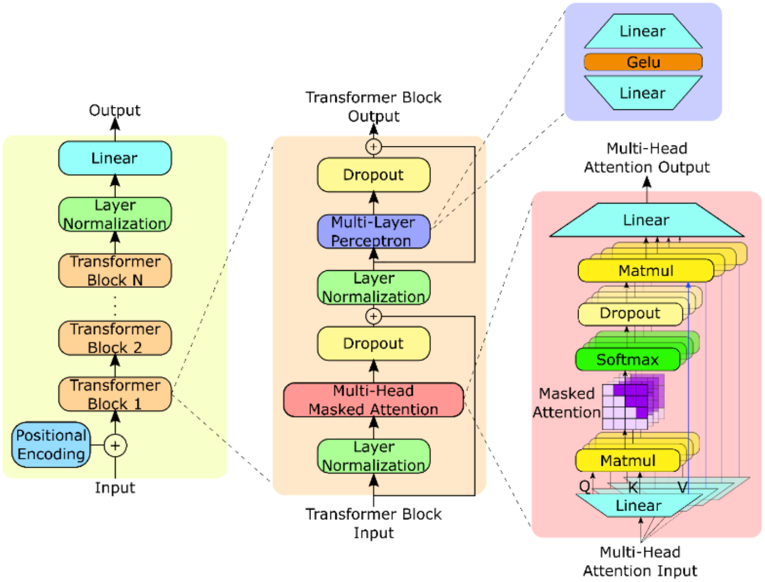
Source: Jay Alammar, Illustrated GPT-2 (2019)

- Selecting the top-1 token (highest score) is simple but often suboptimal.
- Better results come from sampling based on token probabilities, allowing more diversity(words with a higher score have a higher chance of being selected).
- A compromise is `top_k = 40`, sampling from the top 40 highest-scoring tokens.

Language Modeling Head: Weight Tying

- Reuses token embedding weights for output layer.
- Reduces parameters by $\sim n_{\text{embd}} \times 50257$.
- Maintains consistency between input and output representations.

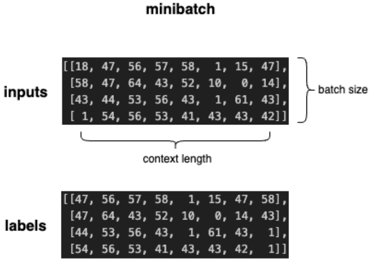
Complete GPT-2 Model Architecture



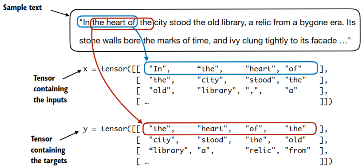
Source: FLUID-GPT

GPT-2 Training: Objective and Setup

- **Objective**: Minimize cross-entropy for next-token prediction using WebText ($\sim$40GB).
- **Hardware**: Trained on GPUs/TPUs at scale.
- **Optimizer**: AdamW (weight decay 0.1), LR $= 6 \times 10^{-4}$, cosine decay with warmup.
- **Batch Size**: $\sim$0.5M tokens (via gradient accumulation).



Token IDs



Input Text

Source: Vipul Koti, Medium (2024)

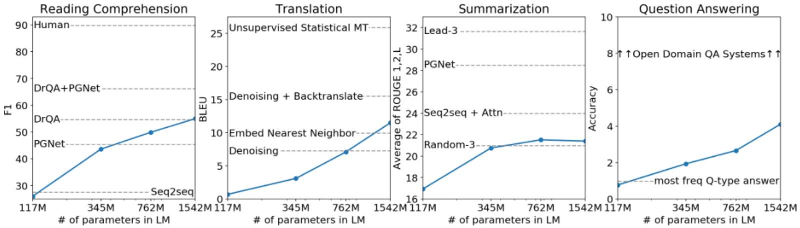
Datasets: Training Data

- **WebText**: Curated from 45M Reddit outbound links.
- **Preprocessing**: HTML cleaned using Dragnet and Newspaper extractors.
- **Wikipedia excluded** to avoid test data leakage.

Datasets: Evaluation

- **CoQA**: Conversational Question Answering.
- **LAMBADA**: Long-range dependency modeling.
- **Children's Book Test (CBT)**: Narrative comprehension.
- **Language Modeling**:
  - WikiText-2, Penn Treebank (PTB), enwik8, text8.
- **WMT-14 Fr-En**: Machine translation.
- **CNN/Daily Mail**: Text summarization.

Evaluation Metric



Source: OpenAI, Language Models are Unsupervised Multitask Learners (2019)

Accuracy, Precision & Recall

Confusion Matrix:

$$\begin{bmatrix} \text{TP} & \text{FP} \\ \text{FN} & \text{TN} \end{bmatrix}$$

TP = True Positive, FP = False Positive,
FN = False Negative, TN = True Negative

Formulas:

- **Accuracy:**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision:**

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall (Sensitivity):**

$$\text{Recall} = \frac{TP}{TP + FN}$$

Perplexity

- **Purpose**: Measures uncertainty of predicted token sequence.
- **Formula**:

$$\text{PPL} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log p(x_i)\right)$$

Example:
Sentence: "The dog chased the cat"
Assume the model predicts word probabilities:
$P(w_i \mid w_{<i}) = [0.5, 0.4, 0.2, 0.6, 0.5]$

$$\text{PPL} = e^{-\frac{1}{5}(\log 0.5+\log 0.4+\log 0.2+\log 0.6+\log 0.5)} \approx 2.42$$

Interpretation: A lower perplexity means the model is more confident in its predictions.

Key Results:

- LAMBADA: 8.6 (vs SOTA 99.8)
- WebText test set: 35.76

F1 Score (CoQA)

- **Purpose**: Evaluates token-level overlap in QA.
- **Formula**:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Example**:
  - Ground Truth: "red apple", Prediction: "apple"
  - Precision $= 1/1$, Recall $= 1/2 \Rightarrow$ F1 $= 0.67$
- **GPT-2 Performance**: F1 $= 55$ (vs human 89)

Accuracy Metrics

Children's Book Test (CBT)

- Cloze-style prediction (10-way multiple choice).
- **Example**: "She opened the __ and found a kitten." ⇒ "door"
- GPT-2: 93.3% (common nouns), 89.1% (named entities)

LAMBADA Accuracy

- Predict final word in long context.
- **Example**: "After hours of thinking, he finally said the word __" ⇒ "yes"
- Accuracy: 63.24% with stop-word filtering

Translation Metric: BLEU Score

BLEU Score

- Measures N-gram overlap between translation and reference.
- GPT-2 on WMT-14 Fr-En: 11.5 BLEU (vs 33.5 SOTA)
- **Formula (for n=2):**

$$\text{BLEU} = \sqrt{P_1 \cdot P_2}$$

- **Example:**
  - Reference: "The cat is on the mat"
  - Prediction: "The cat sat on the mat"
  - Matching unigrams: 5 / 6
  - Matching bigrams: 3 / 5
  - $$\text{BLEU} = \sqrt{\frac{5}{6} \cdot \frac{3}{5}} \approx 0.707 \Rightarrow 70.7$$

Summarization Metrics: ROUGE

ROUGE Metrics

- Measures recall-oriented overlap in summarization.
- **Types**:
  - ROUGE-1: Unigram overlap
  - ROUGE-L: Longest common subsequence
- **GPT-2 Results:** ROUGE-1 $= 29.34$, ROUGE-L $= 26.58$

Example:
Reference: "The economy is growing rapidly"
Prediction: "Economy is growing fast"
ROUGE-1 $= 3/5$ ROUGE-2 $= 2/4$ ROUGE-L $= 3/5$

GPT-2 ROUGE Scores for Summarization (TL;DR Dataset)

	R-1	R-2	R-L	R-AVG
Bottom-Up Sum	**41.22**	**18.68**	**38.34**	**32.75**
Lede-3	40.38	17.66	36.62	31.55
Seq2Seq + Attn	31.33	11.81	28.83	23.99
GPT-2 `TL;DR:`	29.34	8.27	26.58	21.40
Random-3	28.78	8.63	25.52	20.98
GPT-2 no hint	21.58	4.03	19.47	15.03

Source: OpenAI, Language Models are Unsupervised Multitask Learners (2019)

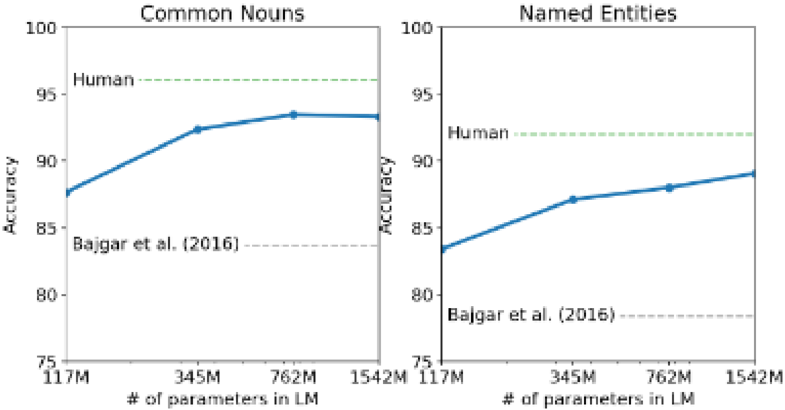
Exact Match (Natural Questions)

- **Purpose**: Strict match between predicted and true answer.
- **Formula**:

$$\text{EM} = \frac{\#\text{exact matches}}{\#\text{total questions}}$$

- **Example**:
  - Truth: "Barack Obama", Prediction: "Obama" $\Rightarrow$ No match
  - Truth: "Barack Obama", Prediction: "Barack Obama" $\Rightarrow$ Match
- **GPT-2 Results**:
  - Overall: 4.1% EM
  - Top 1% confidence: 63.1% EM $\Rightarrow$ Calibration ability

Children’s Book Test



Source: OpenAI, Language Models are Unsupervised Multitask Learners (2019)

Key Observations

- Metric performance scales log-linearly with model size
- Zero-shot transfer works best for:
  - Language modeling (7/8 SOTA)
  - Reading comprehension
- Weakest in:
  - Translation (low-resource setting) (requires explicit structure)

GPT Model Comparison

Model	Year	Parameters	Training Data
GPT-1	2018	117M	BooksCorpus
GPT-2	2019	1.5B	WebText
GPT-3	2020	175B	570GB diverse sources
GPT-3.5	2022	$\sim$175B (fine-tuned)	RLHF-aligned data
GPT-4	2023	$\sim$1T (undisclosed)	Diverse, multimodal data
GPT-4 Turbo	2023	Unknown	Optimized variant

GPT Versions: Key Improvements and Features

- **GPT-1 (2018):**
  117M parameters; trained on BooksCorpus.
  First decoder-only Transformer. Introduced language modeling for NLP.
- **GPT-2 (2019):**
  1.5B parameters; trained on WebText.
  Showed strong zero-shot ability; coherent long-text generation; OpenAI delayed release.
- **GPT-3 (2020):**
  175B parameters; trained on 570GB diverse internet text.
  Enabled few-shot and one-shot learning; capable of multi-task reasoning.
- **GPT-3.5 (2022):**
  Fine-tuned GPT-3 with RLHF.
  Improved instruction-following, factual accuracy, and conversation quality.
- **GPT-4 (2023):**
  Rumored $\sim$1T params; supports multimodal input.
  Stronger logical reasoning, visual input (images), long context windows (32K+).
- **GPT-4 Turbo (2023):**
  Optimized variant of GPT-4, Cheaper, faster, more efficient

References I

- A. Vaswani et al., *Attention is All You Need*, NeurIPS, 2017.
- S. Hochreiter and J. Schmidhuber, *Long Short-Term Memory*, Neural Computation, 1997.
- D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed., 2020.
- A. Radford et al., *Language Models are Unsupervised Multitask Learners*, OpenAI, 2019. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
- J. Alammar, *The Illustrated GPT-2*, https://jalammar.github.io/illustrated-gpt2/
- V. Koti, *From Theory to Code: GPT-2 Breakdown*, Medium, 2023. https://medium.com/@vipul.koti333/from-theory-to-code-step-by-step-implementation-and-code-bre

References II

K. Dheer, *Understanding Evolution of GPT*, LinkedIn, 2024.

A. Desai, *Text Generation Using RNN, GPT-2*, Medium, 2023.