Yale University

Language Models and Transformers

S&DS 265 · Introductory Machine Learning · Lecture 22

Zhuoran Yang

Fall 2026

AI usage: figures were generated, and these slides polished, with the help of AI tools.

Outline

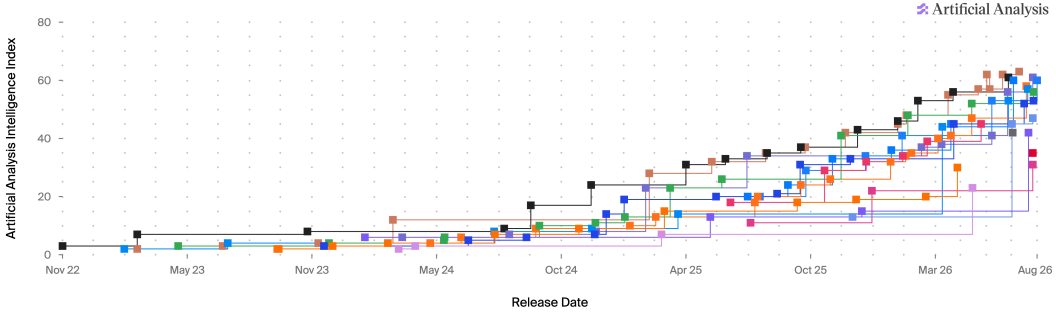
Learning Objectives

In this lecture you will learn:

1. How encoder-only, encoder–decoder, and decoder-only transformers differ;
2. How raw text is tokenized and packed into sequences that supply their own targets;
3. What the model is as a black box: one conditional distribution and one loss;
4. How that box is built from an embedding, a stack of blocks, and an unembedding;
5. How the trained model is sampled, and what temperature, top-k, and top-p change.

Outline

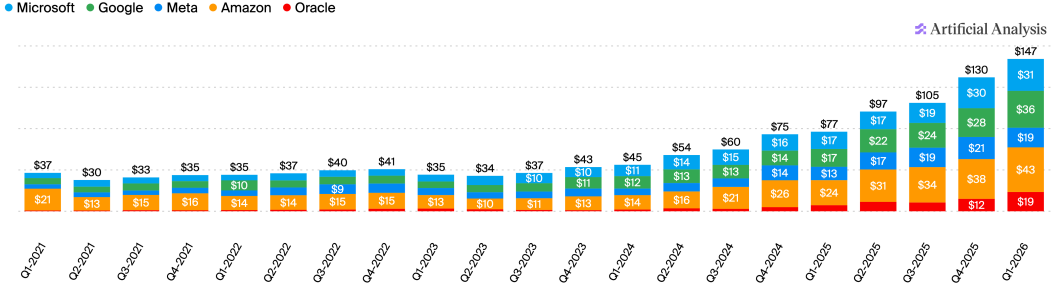
AI Boom



Each marker is a released model, placed at its release date and scored on one index of nine benchmarks. The frontier went from 3 to above 60 in under four years, and the field went from one lab to seventeen.

Artificial Analysis Intelligence Index v4.1.1, nine evaluations; artificialanalysis.ai, 21 August 2026. The index is one firm's choice of benchmarks.

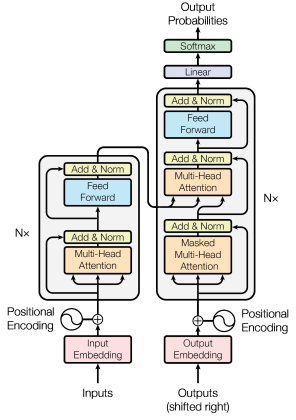
Capital Expenditure by Major Technology Companies



Each bar is one quarter's capital spending, stacked by company. Flat near $35 B through 2023, then $147 B in a single quarter — more than three times the level two years earlier.

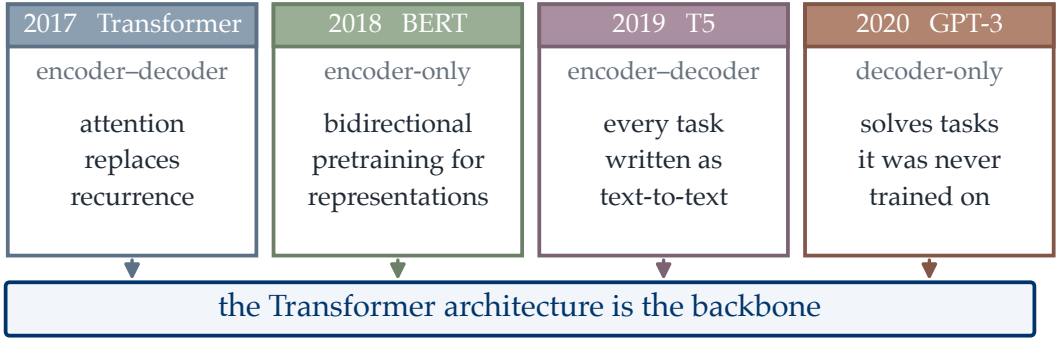
Artificial Analysis, compiled from company filings; artificialanalysis.ai, 21 August 2026. Capital expenditure covers data centres built for more than language models.

Everything Starts Here — Transformer Models



Source: Vaswani et al. (2017), "Attention Is All You Need," arXiv:1706.03762, Figure 1.

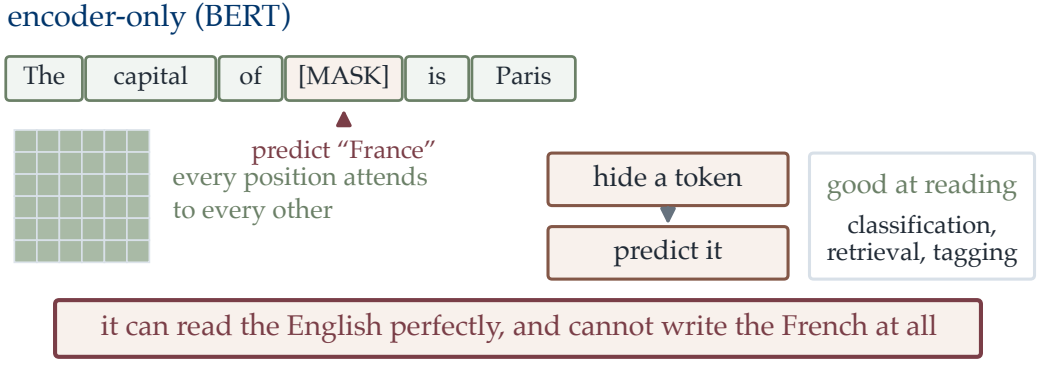
Four Papers That Set the Shape



Vaswani et al. (2017); Devlin et al. (2018); Raffel et al. (2019); Brown et al. (2020).

Outline

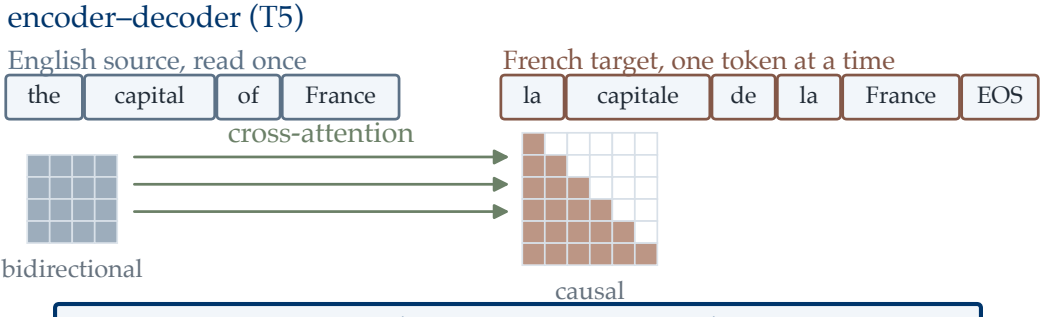
Encoder-Only Models



BERT hides a token and asks the model to recover it, so every position may look both ways. Excellent for reading; unable to write left to right.

Devlin et al. (2018), “BERT: Pre-training of Deep Bidirectional Transformers,” arXiv:1810.04805.

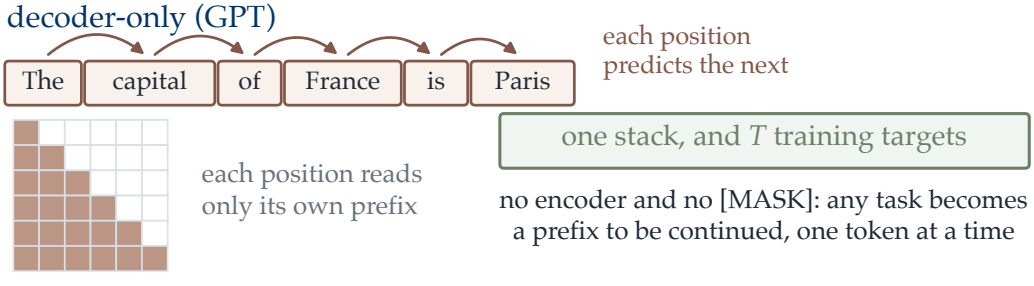
Encoder–Decoder Models



T5 keeps both halves of the 2017 diagram: a source read bidirectionally, and a target written causally that attends back into the encoder.

Raffel et al. (2019), "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," arXiv:1910.10683.

Decoder-Only Models



One causal stack, no encoder and no mask token. Every position is a training target, so a sequence of length T supplies T of them.

Brown et al. (2020), "Language Models are Few-Shot Learners," arXiv:2005.14165.

Why Decoder-Only Won for Generation

- **Training and generation are the same task**. There is no gap between the objective and how the model is used.
- **Every position supplies a target**, so one pass over a sequence yields T gradient signals rather than one.
- **Any task is a continuation**. A question, an instruction, a document to summarize — all become a prefix.

Why Decoder-Only Won for Generation

- **Training and generation are the same task**. There is no gap between the objective and how the model is used.
- **Every position supplies a target**, so one pass over a sequence yields T gradient signals rather than one.
- **Any task is a continuation**. A question, an instruction, a document to summarize — all become a prefix.

The rest of the lecture is about decoder-only models only. Encoders and encoder–decoders remain in use where the task is to read rather than to write.

Outline

One Task, Written Down

Before any architecture: a decoder-only model does exactly one thing — given the text so far, put a probability distribution on every possible next token.

Where does the training data for that come from, and what is the loss?

What a Pretraining Corpus Is

Not one source. A **deliberate mixture**, on the order of a few terabytes of text.

Reference	Books and web	Code
encyclopedia articles, textbooks, documentation	fiction, news, forums, transcripts	public repositories, in many languages

What a Pretraining Corpus Is

Not one source. A **deliberate mixture**, on the order of a few terabytes of text.

Reference	Books and web	Code
encyclopedia articles, textbooks, documentation	fiction, news, forums, transcripts	public repositories, in many languages

What goes into the mixture is a design decision.

Wikipedia Text as Training Data

One sentence out of those terabytes:

> "Machine learning algorithms can learn from data and generalise to unseen data."

Wikipedia Text as Training Data

One sentence out of those terabytes:

> "Machine learning algorithms can learn from data and generalise to unseen data."

No annotation was applied. The sentence already records, at every position, which word came next — and that is the only supervision the model will ever get.

Wikipedia Text as Training Data

One sentence out of those terabytes:

> "Machine learning algorithms can learn from data and generalise to unseen data."

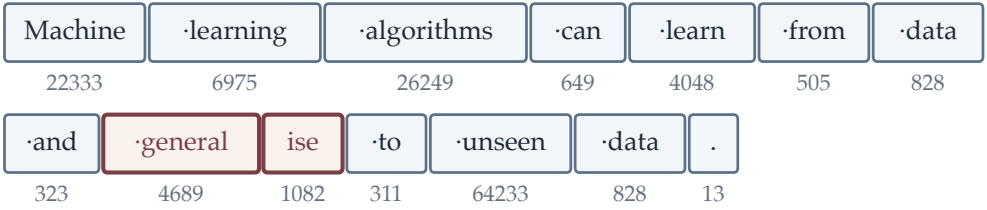
No annotation was applied. The sentence already records, at every position, which word came next — and that is the only supervision the model will ever get. This method is called self-supervised learning: part of the unlabelled data itself serves as the label.

Source: Abridged from the opening of the English Wikipedia article "Machine learning"; text is CC BY-SA 4.0.

A Real Tokenization

Machine learning algorithms can learn from data and generalise to unseen data.

14 tokens, with vocabulary rows below



· is a leading space

The model never sees characters or words. It sees integers: 12 words become 14 tokens, and the one unusual spelling splits into pieces the vocabulary already holds.

Computed with `tiktoken`'s `cl100k_base` encoder; the Llama 3 tokenizer can be tried at `lunary.ai/llama3-tokenizer`.

What a Token Is

A **tokenizer** is a fixed, learned dictionary of V byte strings. Text is greedily rewritten as a sequence of their indices,

$$x_{1:T} \in \{1, \dots, V\}^T, \qquad V \approx 10^5.$$

What a Token Is

A **tokenizer** is a fixed, learned dictionary of V byte strings. Text is greedily rewritten as a sequence of their indices,

$$x_{1:T} \in \{1, \dots, V\}^T, \qquad V \approx 10^5.$$

The dictionary is fitted to a corpus before training begins, so that frequent strings get one token each and rare ones get several.

What a Token Is

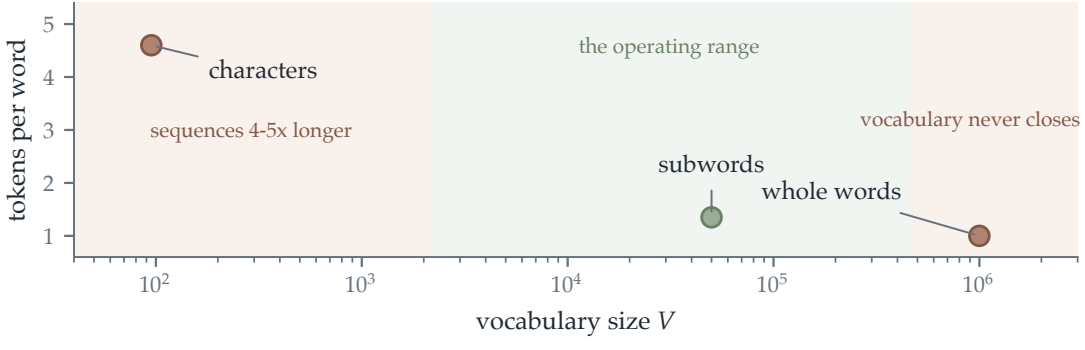
A **tokenizer** is a fixed, learned dictionary of V byte strings. Text is greedily rewritten as a sequence of their indices,

$$x_{1:T} \in \{1, \dots, V\}^T, \qquad V \approx 10^5.$$

The dictionary is fitted to a corpus before training begins, so that frequent strings get one token each and rare ones get several.

The vocabulary is chosen before training and never changes. Everything downstream — V, sequence length, cost — is measured in these units.

Why Subword Units Won



Characters stretch sequences fourfold and attention costs T^2; whole words never close the vocabulary. Subwords sit between, and V stays near 10^5.

Tokens-per-word values are typical for English text.

Reading One Sequence

Take the token sequence and cut it after position three:

$$\underbrace{\texttt{(Machine, learning, algorithms)}}_{\text{input}} \longrightarrow \underbrace{\texttt{can}}_{\text{target}}\texttt{.}$$

Reading One Sequence

Take the token sequence and cut it after position three:

$$\underbrace{\texttt{(Machine, learning, algorithms)}}_{\text{input}} \longrightarrow \underbrace{\texttt{can}}_{\text{target}}\texttt{.}$$

Cutting after every other position gives a data point: a sequence of length T supplies $T - 1$ input–target pairs.

Predict each token x_t given prefix $x_{<t} = \{x_1, \ldots, x_{t-1}\}$.

Reading One Sequence

Take the token sequence and cut it after position three:

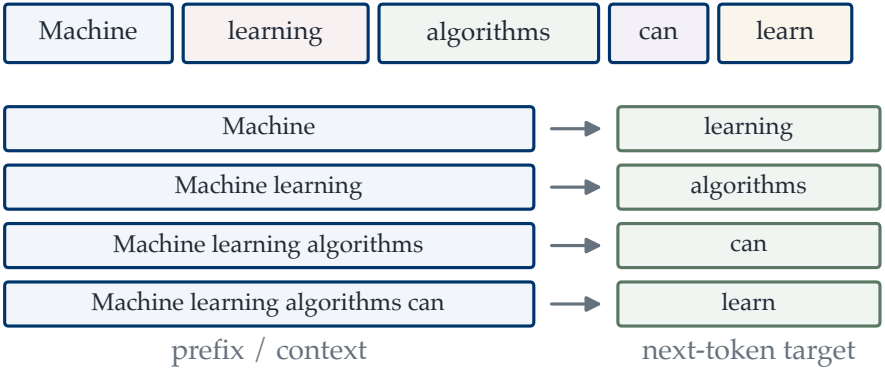
$$\underbrace{\texttt{(Machine, learning, algorithms)}}_{\text{input}} \longrightarrow \underbrace{\texttt{can}}_{\text{target}}\texttt{.}$$

Cutting after every other position gives a data point: a sequence of length T supplies $T - 1$ input–target pairs.

Predict each token x_t given prefix $x_{<t} = \{x_1, \ldots, x_{t-1}\}$.

This is ordinary supervised learning.

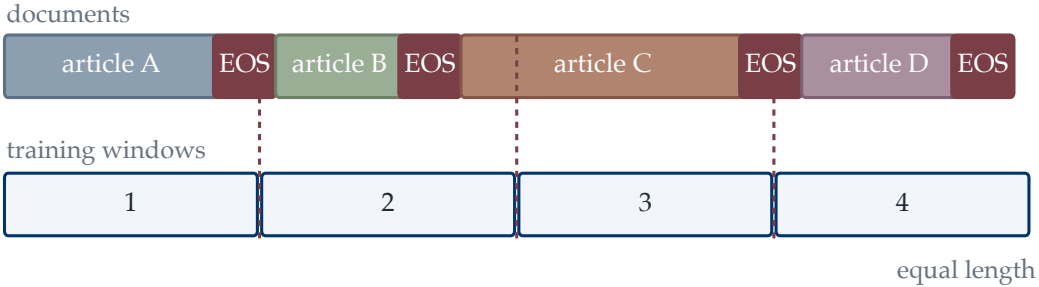
One Sequence Produces Several Training Pairs



Every prefix is an input and the token after it is the target. Nothing was annotated: the right column is the left column, shifted by one.

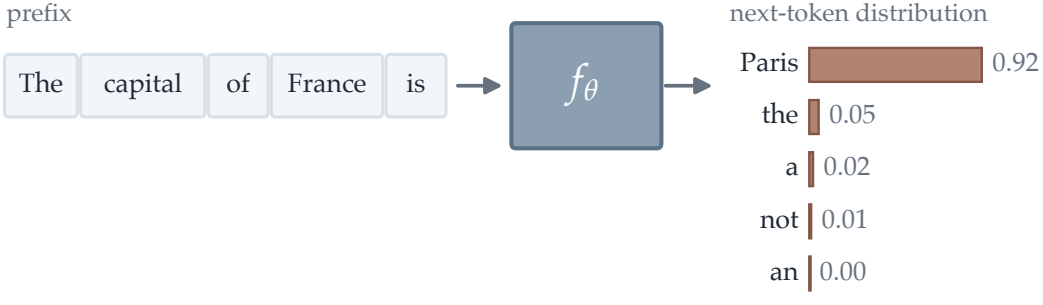
Each target is copied from the next observed token.

Packing a Corpus into Fixed Windows



Concatenate, then cut into equal windows — 4,096 or 8,192 tokens is typical — so a batch is a rectangle. A document can straddle a cut, and a window can hold several. The stop token is a token like any other, and the model must learn to emit it.

The Model, Before We Open It



A network f_θ turns the prefix into V real numbers, and a softmax turns those into a distribution over the whole vocabulary. Everything before the architecture section depends only on that one line.

Problem Setup

The network returns one real number per vocabulary entry,

$$z = f_\theta(x_{<t}) \in \mathbb{R}^V,$$

which are called **logits**. A softmax turns them into a distribution,

$$\text{softmax}(z)_v = \frac{e^{z_v}}{\sum_{u=1}^{V} e^{z_u}}, \qquad p_\theta(x_t \mid x_{<t}) = \text{softmax}(z)_{x_t},$$

and a whole document is the product $p_\theta(x_{1:T}) = \prod_{t=1}^{T} p_\theta(x_t \mid x_{<t})$.

Problem Setup

The network returns one real number per vocabulary entry,

$$z = f_\theta(x_{<t}) \in \mathbb{R}^V,$$

which are called **logits**. A softmax turns them into a distribution,

$$\text{softmax}(z)_v = \frac{e^{z_v}}{\sum_{u=1}^{V} e^{z_u}}, \qquad p_\theta(x_t \mid x_{<t}) = \text{softmax}(z)_{x_t},$$

and a whole document is the product $p_\theta(x_{1:T}) = \prod_{t=1}^{T} p_\theta(x_t \mid x_{<t})$.

Adding the same constant to every logit leaves the softmax unchanged: only differences between logits carry information. And θ is shared across every position t.

Training Loss Is Cross-Entropy

Take logs and average over the non-padding positions of the corpus:

$$\mathcal{L}(\theta) = -\frac{1}{N_{\text{tok}}} \sum_{t} \log p_\theta(x_t \mid x_{<t})$$

Training Loss Is Cross-Entropy

Take logs and average over the non-padding positions of the corpus:

$$\mathcal{L}(\theta) = -\frac{1}{N_{\text{tok}}} \sum_{t} \log p_\theta(x_t \mid x_{<t})$$

With the one-hot target $y_{tv} = \mathbf{1}\{x_t = v\}$ this is exactly $-\sum_v y_{tv} \log p_\theta(v \mid x_{<t})$: the multiclass cross-entropy of Lecture 8, with $V \approx 10^5$ classes, evaluated once per token.

Training Loss Is Cross-Entropy

Take logs and average over the non-padding positions of the corpus:

$$\mathcal{L}(\theta) = -\frac{1}{N_{\text{tok}}} \sum_{t} \log p_\theta(x_t \mid x_{<t})$$

With the one-hot target $y_{tv} = \mathbf{1}\{x_t = v\}$ this is exactly $-\sum_v y_{tv} \log p_\theta(v \mid x_{<t})$: the multiclass cross-entropy of Lecture 8, with $V \approx 10^5$ classes, evaluated once per token.

Maximum likelihood and cross-entropy are one objective written two ways.

Computing $\log p_\theta$ in Python

```
logits  = model(x[:, :-1])      # (B, T-1, V)  one score per vocab entry
targets = x[:, 1:]              # (B, T-1)     the next token at each t

loss = F.cross_entropy(logits.flatten(0, 1),   # (B*(T-1), V)
                       targets.flatten())      # (B*(T-1),)

# the same thing, written out:
logp   = F.log_softmax(logits, dim=-1)                # (B, T-1, V)
chosen = logp.gather(-1, targets[..., None])[..., 0]   # (B, T-1)
loss   = -chosen.mean()
```

Computing $\log p_\theta$ in Python

```
logits  = model(x[:, :-1])      # (B, T-1, V)  one score per vocab entry
targets = x[:, 1:]              # (B, T-1)     the next token at each t

loss = F.cross_entropy(logits.flatten(0, 1),   # (B*(T-1), V)
                       targets.flatten())      # (B*(T-1),)

# the same thing, written out:
logp   = F.log_softmax(logits, dim=-1)                # (B, T-1, V)
chosen = logp.gather(-1, targets[..., None])[..., 0]   # (B, T-1)
loss   = -chosen.mean()
```

`log_softmax` is a single PyTorch function. It subtracts the row maximum before exponentiating; without that, e^{z_v} overflows once a logit passes 88 in `float32`. Appendix A.4.

Perplexity

The average token loss and its exponential:

$$\bar{L} = -\frac{1}{T}\sum_{t=1}^{T} \log p_\theta(x_t \mid x_{<t}), \qquad \text{PPL} = e^{\bar{L}}.$$

Perplexity

The average token loss and its exponential:

$$\bar{L} = -\frac{1}{T}\sum_{t=1}^{T} \log p_\theta(x_t \mid x_{<t}), \qquad \text{PPL} = e^{\bar{L}}.$$

For correct-token probabilities .5, .25, .5 we get $\bar{L} \approx .924$ and PPL ≈ 2.52: as uncertain as choosing uniformly among 2.5 tokens at each step.

Perplexity

The average token loss and its exponential:

$$\bar{L} = -\frac{1}{T}\sum_{t=1}^{T} \log p_\theta(x_t \mid x_{<t}), \qquad \text{PPL} = e^{\bar{L}}.$$

For correct-token probabilities .5, .25, .5 we get $\bar{L} \approx .924$ and PPL ≈ 2.52: as uncertain as choosing uniformly among 2.5 tokens at each step.

Perplexity compares models only under one tokenizer and one corpus. Across tokenizers it is not comparable.

How It Is Actually Optimized

Minibatch stochastic gradient descent, exactly as in Lecture 5, with three things worth naming.

- **AdamW** is the default; **Muon**, which orthogonalizes the update for matrix-shaped parameters, is the current challenger.
- **One to two epochs.** Data is not reused much; it is cheaper to see new tokens.
- **Trillions of tokens** per run, in batches of millions.

How It Is Actually Optimized

Minibatch stochastic gradient descent, exactly as in Lecture 5, with three things worth naming.

- **AdamW** is the default; **Muon**, which orthogonalizes the update for matrix-shaped parameters, is the current challenger.
- **One to two epochs.** Data is not reused much; it is cheaper to see new tokens.
- **Trillions of tokens** per run, in batches of millions.

The regime is the opposite of Lecture 4: with so few passes, underfitting, not overfitting, is the binding constraint.

Source: Loshchilov and Hutter (2019) for AdamW; Jordan et al. (2024) for Muon.

Outline

How Big, and On How Much Data?

Pretraining a large model can take months for a single run. Companies cannot afford repeated runs on their flagship models.

Two numbers are free — the parameter count N and the training tokens D. Can the outcome of a run that has never been performed be predicted from smaller runs that are affordable?

What the Scaling Laws Are For

- **Extrapolate**: fit the curve on small runs costing 10^{18} floating-point operations (FLOPs), and read off what a large run with 10^{24} FLOPs will give.
- **Budget**: answer "what loss will C buy" before spending C.
- **Allocate**: with a fixed C, find a proper configuration (N, D).

Compute Is Roughly $6 \cdot N \cdot D$

Each parameter contributes one multiply–add per token in the forward pass, and the backward pass costs about twice the forward pass. Hence

$$C \approx 6 \cdot N \cdot D \quad \text{floating-point operations.}$$

Compute Is Roughly $6 \cdot N \cdot D$

Each parameter contributes one multiply–add per token in the forward pass, and the backward pass costs about twice the forward pass. Hence

$$C \approx 6 \cdot N \cdot D \quad \text{floating-point operations.}$$

For GPT-3, $N = 1.75 \times 10^{11}$ and $D = 3 \times 10^{11}$ give $C \approx 3.1 \times 10^{23}$, which matches the figure reported for that training run.

Compute Is Roughly $6 \cdot N \cdot D$

Each parameter contributes one multiply–add per token in the forward pass, and the backward pass costs about twice the forward pass. Hence

$$C \approx 6 \cdot N \cdot D \quad \text{floating-point operations.}$$

For GPT-3, $N = 1.75 \times 10^{11}$ and $D = 3 \times 10^{11}$ give $C \approx 3.1 \times 10^{23}$, which matches the figure reported for that training run.

Compute-optimal choice: for a fixed budget C, how should N and D be chosen?

Source: Counting argument in Appendix A.2.

The Question Is Asked at Fixed Compute

Chinchilla poses a single question: among all pairs (N, D) that consume the same budget, which attains the lowest loss?

$$\min_{N,D} L(N, D) \qquad \text{subject to} \qquad 6ND = C.$$

The Question Is Asked at Fixed Compute

Chinchilla poses a single question: among all pairs (N, D) that consume the same budget, which attains the lowest loss?

$$\min_{N,D} L(N, D) \qquad \text{subject to} \qquad 6ND = C.$$

The constraint ties the two together: at fixed C, a larger model must be trained on fewer tokens, and a smaller model can afford more. Chinchilla asks where along that trade-off the loss is lowest.

The Question Is Asked at Fixed Compute

Chinchilla poses a single question: among all pairs (N, D) that consume the same budget, which attains the lowest loss?

$$\min_{N,D} L(N, D) \qquad \text{subject to} \qquad 6ND = C.$$

The constraint ties the two together: at fixed C, a larger model must be trained on fewer tokens, and a smaller model can afford more. Chinchilla asks where along that trade-off the loss is lowest.

Every claim about scaling here is an iso-compute statement.

Source: Hoffmann et al. (2022), "Training Compute-Optimal Large Language Models," arXiv:2203.15556.

The Chinchilla Law

Fit a parametric loss surface to many runs,

$$L(N, D) = E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}},$$

and minimize it under $6ND = C$. The optimum is a pair of power laws,

$$N^{\star} \propto C^{\beta/(\alpha+\beta)}, \qquad D^{\star} \propto C^{\alpha/(\alpha+\beta)}.$$

The Chinchilla Law

Fit a parametric loss surface to many runs,

$$L(N, D) = E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}},$$

and minimize it under $6ND = C$. The optimum is a pair of power laws,

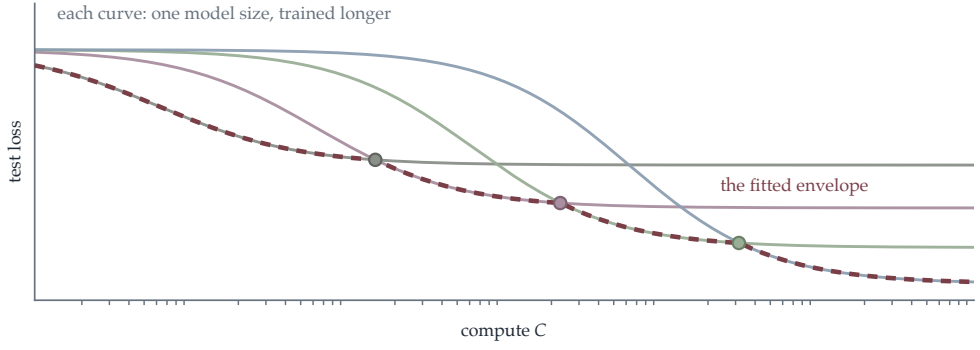
$$N^{\star} \propto C^{\beta/(\alpha+\beta)}, \qquad D^{\star} \propto C^{\alpha/(\alpha+\beta)}.$$

The fitted $\alpha \approx .34$ and $\beta \approx .28$ give both exponents near $\frac{1}{2}$, so N and D should grow together:

$$\boxed{D^{\star} \approx 20\, N^{\star}}$$

Source: Hoffmann et al. (2022), arXiv:2203.15556; the minimization is Appendix A.5.

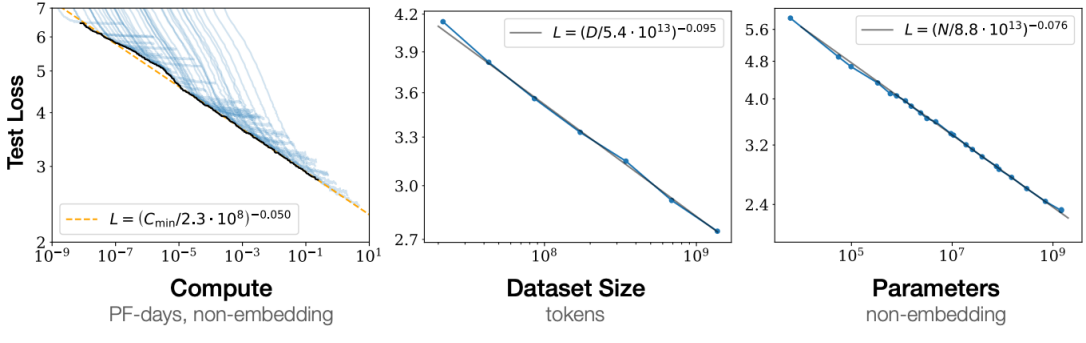
How a Scaling Plot Is Built



Several model sizes are each trained to convergence. A small model reaches its loss floor early and cannot improve beyond it; a larger one begins worse and ends lower. Each dot marks where one size stops being the best choice and hands over to the next. Joining the dots gives the scaling law — dotted, because it is fitted rather than measured.

Original course schematic; the curves are illustrative, not measured runs.

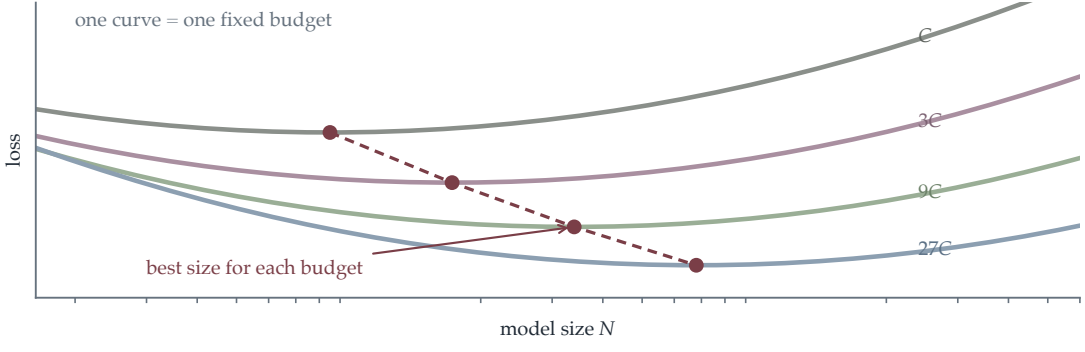
Loss Falls as a Power Law



The same envelope, taken over many runs rather than four, and then repeated against dataset size and against parameter count. Linearity on log–log axes means that each multiplication of a resource yields a fixed subtraction from the loss.

Kaplan et al. (2020), "Scaling Laws for Neural Language Models," arXiv:2001.08361, Figure 1.

One Budget, One Best Size



The same measurements, sectioned the other way: the budget is held fixed and model size is varied. Each curve has an interior minimum, and those minima shift rightward as the budget increases.

Original course schematic of the IsoFLOP analysis of Hoffmann et al. (2022); shapes are illustrative, not digitized.

Deployment Changes the Objective

A model is trained once and then run billions of times. Inference cost scales with N and not with D, so the quantity to minimize is

$$\underbrace{6ND}_{\text{training}} + \underbrace{2N \cdot (\text{tokens served})}_{\text{inference}}.$$

Deployment Changes the Objective

A model is trained once and then run billions of times. Inference cost scales with N and not with D, so the quantity to minimize is

$$\underbrace{6ND}_{\text{training}} + \underbrace{2N \cdot (\text{tokens served})}_{\text{inference}}.$$

Adding the second term pushes the optimum to models that are smaller than Chinchilla and trained longer.

Deployment Changes the Objective

A model is trained once and then run billions of times. Inference cost scales with N and not with D, so the quantity to minimize is

$$\underbrace{6ND}_{\text{training}} + \underbrace{2N \cdot (\text{tokens served})}_{\text{inference}}.$$

Adding the second term pushes the optimum to models that are smaller than Chinchilla and trained longer.

No single serving-optimal constant replaces the 20. The optimum depends on the inference volume a deployment anticipates.

Source: Sardana, Portes, Doubov and Frankle (2024), "Beyond Chinchilla-Optimal," arXiv:2401.00448.

Training Budgets in Practice

Llama 3	parameters	training tokens	tokens per parameter
8B	8×10^9	$\approx 15\,\mathrm{T}$	$\approx 1{,}900$
70B	70×10^9	$\approx 15\,\mathrm{T}$	≈ 210
405B	405×10^9	$\approx 15\,\mathrm{T}$	≈ 39
Chinchilla	any	$20N$	20

Training Budgets in Practice

Llama 3	parameters	training tokens	tokens per parameter
8B	8×10^9	$\approx 15\,\mathrm{T}$	$\approx 1{,}900$
70B	70×10^9	$\approx 15\,\mathrm{T}$	≈ 210
405B	405×10^9	$\approx 15\,\mathrm{T}$	≈ 39
Chinchilla	any	$20N$	20

All three sizes were trained on the same corpus, so the smaller the model, the further beyond the Chinchilla ratio it is trained: it is the size whose inference cost is largest relative to the cost of training it.

Source: Token budgets from the Llama 3 model report; ratios computed from those figures.

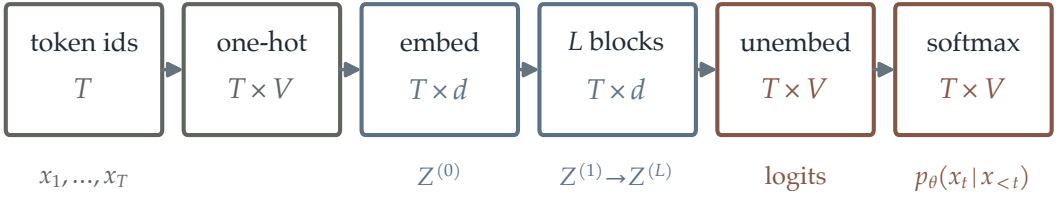
Outline

Now Open the Box

Everything so far — the data, the loss, the optimizer, the scaling laws — uses a black box f_θ that maps a prefix to V numbers.

What is the NN architecture of f_θ?

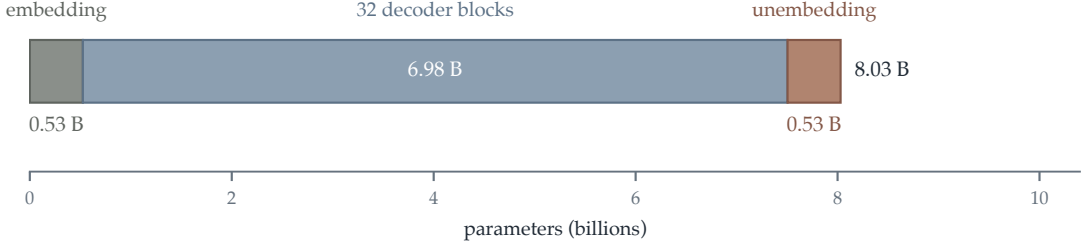
What a Transformer Maps



$T = 4{,}096$ $V = 128{,}256$ $d = 4{,}096$ $L = 32$

A transformer is one function on whole sequences: T token indices in, a $T \times V$ matrix of logits out, with row t scoring every possible continuation of the first t tokens.

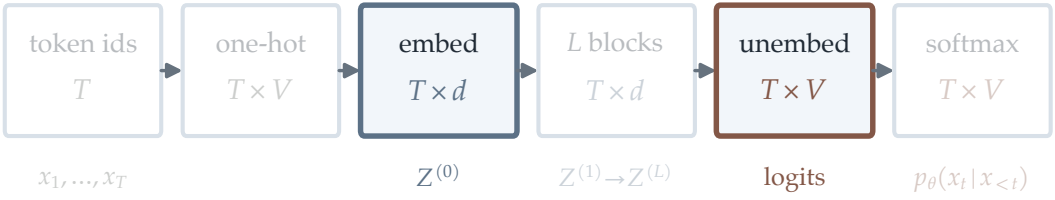
Embedding, Blocks, Unembedding



Three stages, and the arithmetic closes: $0.53 + 6.98 + 0.53 = 8.03$ billion. The two vocabulary matrices are $V \times d$ each, which is why a large vocabulary is not free.

Original arithmetic for the published Llama-3-8B configuration: $d = 4096$, $L = 32$, $V = 128{,}256$, MLP hidden 14,336, 8 key/value heads.

Where We Are: the Two Lookups



$T = 4{,}096 \quad V = 128{,}256 \quad d = 4{,}096 \quad L = 32$

The two ends of the pipeline are tables: one turns a token index into a vector, the other turns the final vector back into a score per token. Everything between them is the stack of blocks.

Tokens Must Become Vectors

Indices carry no semantic meaning: token 4,102 is not "near" token 4,103.

The **embedding table** $E \in \mathbb{R}^{V \times d}$ assigns each token index a vector,

$$x_t \longmapsto z_t^{(0)} = E[x_t] \in \mathbb{R}^d.$$

The **unembedding** $U \in \mathbb{R}^{d \times V}$ turns the final state back into logits,

$$f_\theta(x_{<t}) = z_t^{(L)} U \in \mathbb{R}^V.$$

Tokens Must Become Vectors

Indices carry no semantic meaning: token 4,102 is not "near" token 4,103.

The **embedding table** $E \in \mathbb{R}^{V \times d}$ assigns each token index a vector,

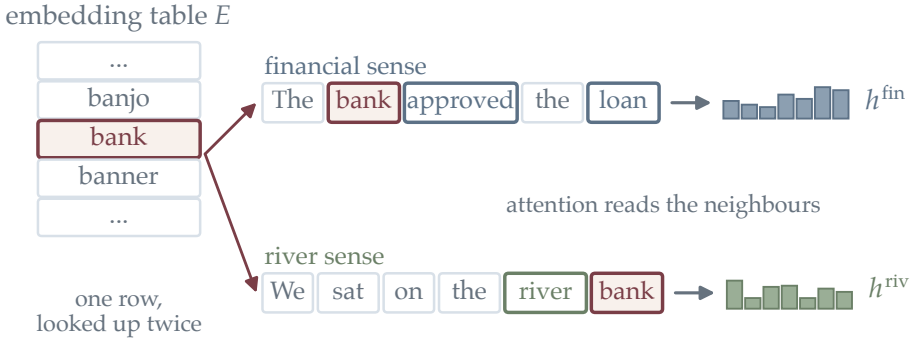
$$x_t \longmapsto z_t^{(0)} = E[x_t] \in \mathbb{R}^d.$$

The **unembedding** $U \in \mathbb{R}^{d \times V}$ turns the final state back into logits,

$$f_\theta(x_{<t}) = z_t^{(L)} U \in \mathbb{R}^V.$$

Both are ordinary parameters, learned with everything else. Some models tie $U = E^\top$ and save a further $V \cdot d$.

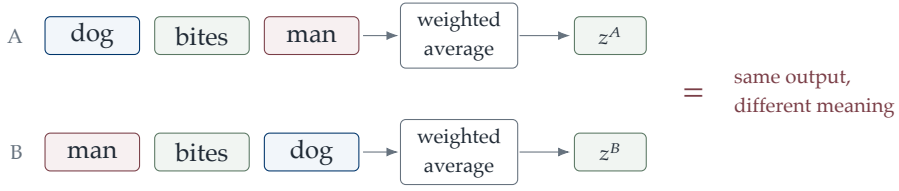
A Word's Meaning Depends on Its Context



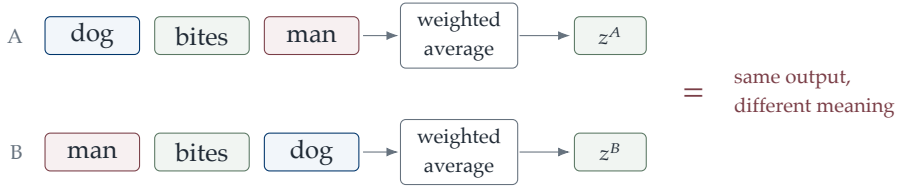
Bank is one row of E, so both sentences start from the same vector. Only the neighbours distinguish a riverbank from a lender, so a position must be able to read the other positions.

The output vectors are schematic, not measured activations.

The Order of the Words Also Matters



The Order of the Words Also Matters



Direct averaging token features is blind to order. Positional information should enter the internal features of the transformer.

Two Things the Architecture Must Supply

1. Position. The vectors must know *where* they sit, or order is invisible.
2. Mixing. A position must be able to read the others, or context is invisible.

Two Things the Architecture Must Supply

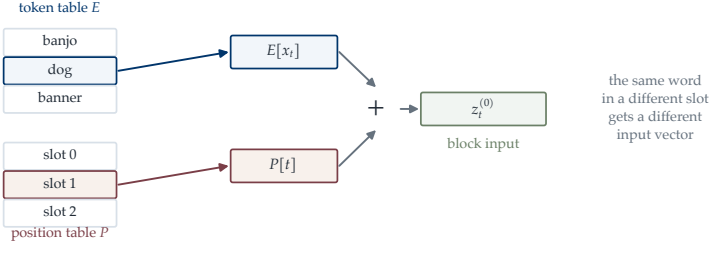
1. Position. The vectors must know *where* they sit, or order is invisible.
2. Mixing. A position must be able to read the others, or context is invisible.

Position is added to the embedding. Mixing is **attention** — the only part of the block that moves information between positions.

Everything else in a transformer block — the MLP, the normalization — acts on each position separately. Attention is the only thing that mixes token information.

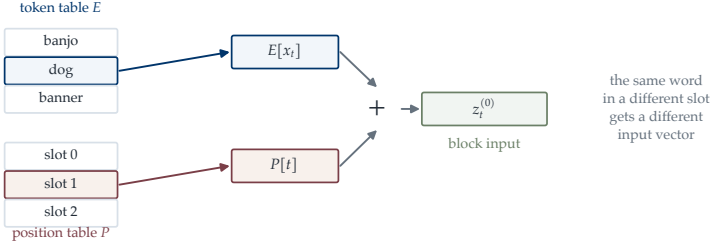
Position Enters the Embedding

The simplest form is a second table. A **positional embedding** P holds one row per *slot*, added to the token vector: $z_t^{(0)} = E[x_t] + P[t]$.



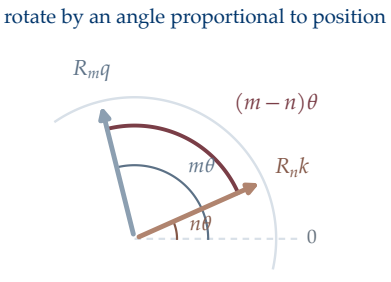
Position Enters the Embedding

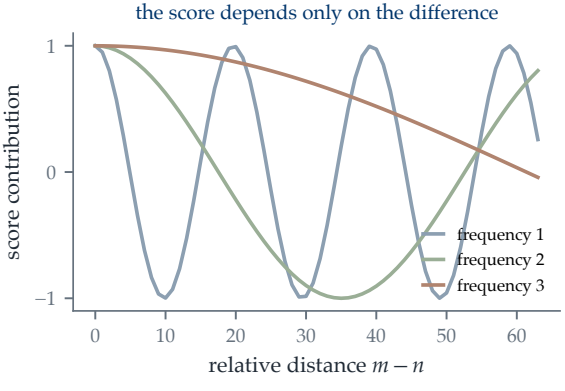
The simplest form is a second table. A **positional embedding** P holds one row per *slot*, added to the token vector: $z_t^{(0)} = E[x_t] + P[t]$.



Fixed sinusoids, a learned table, or a rotation applied later (appendix C). **A learned table runs out of rows past its training length.**

Rotary Position Embedding

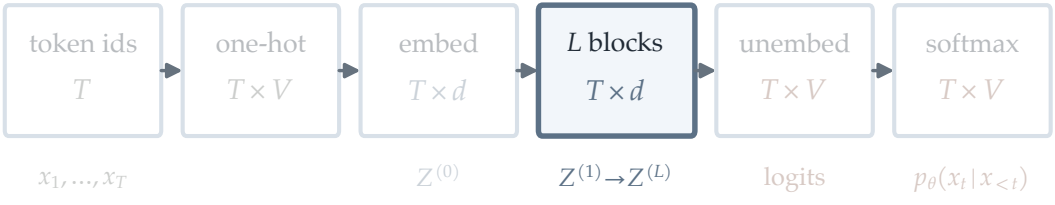




Rotate each position's vector by an angle proportional to its index. The angle between two positions then depends only on the distance between them, so the scheme behaves the same at position 50 and at 50,000, with no table to run out of.

Su et al. (2021), "RoFormer: Enhanced Transformer with Rotary Position Embedding," arXiv:2104.09864.

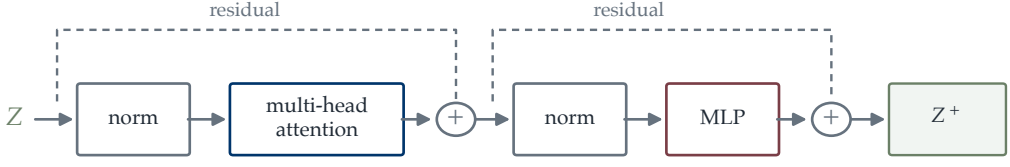
Where We Are: Inside a Block



$T = 4{,}096 \qquad V = 128{,}256 \qquad d = 4{,}096 \qquad L = 32$

The L blocks are the only part left. Every one has the same four pieces, on two residual paths — and every one maps $T \times d$ to $T \times d$.

Four Pieces of a Transformer Block



one block: $T \times d$ in, $T \times d$ out

Normalization and the MLP act on each position separately; attention is the only piece that moves information between positions. We take them in that order.

Every Shape in One Place

symbol	what it is	GPT-3
V	vocabulary size	50,257
T	context length, in tokens	2,048
d	model dimension: the same at every layer	12,288
H	attention heads per block	96
d_k	width *inside* one head, d/H	128
$4d$	hidden width of the MLP	49,152

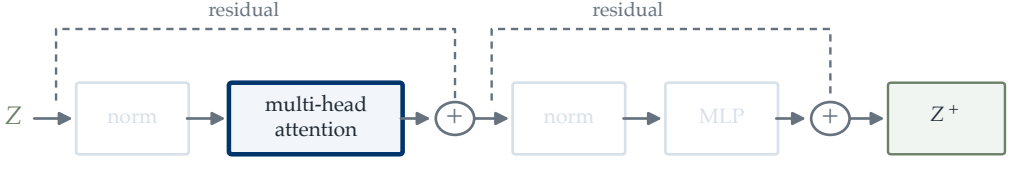
Every Shape in One Place

symbol	what it is	GPT-3
V	vocabulary size	50,257
T	context length, in tokens	2,048
d	model dimension: the same at every layer	12,288
H	attention heads per block	96
d_k	width *inside* one head, d/H	128
$4d$	hidden width of the MLP	49,152

Only d is fixed by the architecture. A block maps a sequence in $\mathbb{R}^d$ to a sequence in $\mathbb{R}^d$: the width changes *inside* the block, never across it.

Source: Numbers from Brown et al. (2020); laid out dimension by dimension in Dugas, *The GPT-3 Architecture, on a Napkin*.

Now: Attention



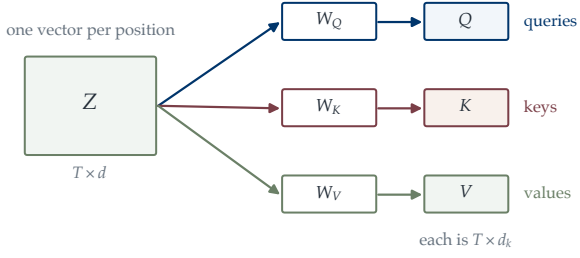
one block: $T \times d$ in, $T \times d$ out

The one piece that lets positions see each other.

Attention: Queries, Keys, and Values

Three learned projections take $Z \in \mathbb{R}^{T \times d}$ to the **attention width** d_k:

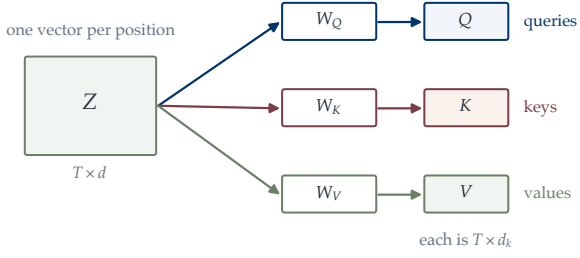
$$Q = ZW^Q, \quad K = ZW^K, \quad V = ZW^V, \qquad W^Q, W^K, W^V \in \mathbb{R}^{d \times d_k}.$$



Attention: Queries, Keys, and Values

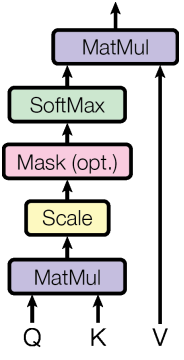
Three learned projections take $Z \in \mathbb{R}^{T \times d}$ to the **attention width** d_k:

$$Q = ZW^Q, \quad K = ZW^K, \quad V = ZW^V, \qquad W^Q, W^K, W^V \in \mathbb{R}^{d \times d_k}.$$



Every vector in Z becomes three vectors.

Scaled Dot-Product Attention



1. Compare queries with keys;
2. Normalize the scores;
3. Apply causal masking;
4. Feed scores to softmax function;
5. Average the values.

Source: Vaswani et al. (2017), Figure 2 left, credited educational excerpt.

Steps 1 and 2: Scores

Position t compares its query against the key of every position j, and divides by the width the vectors were summed over:

$$\text{score}(t \to j) = \frac{\langle q_t, k_j \rangle}{\sqrt{d_k}}.$$

Steps 1 and 2: Scores

Position t compares its query against the key of every position j, and divides by the width the vectors were summed over:

$$\text{score}(t \to j) = \frac{\langle q_t, k_j \rangle}{\sqrt{d_k}}.$$

Here d_k is the length of q_t and k_j — the **key dimension**.

A dot product of d_k terms grows like $\sqrt{d_k}$, so dividing keeps the scores at a fixed scale whatever the width.

Source: Notation follows Vaswani et al. (2017): "queries and keys of dimension d_k, and values of dimension d_v".

Steps 3 and 4: Mask, then Softmax

Position t is trained to predict what comes *next*, so it may not look ahead: keep the scores with $j \le t$ and discard the rest (causal masking). Softmax over the survivors:

$$p_t(j) = \frac{\exp(\text{score}(t \to j))}{\sum_{j' \le t} \exp(\text{score}(t \to j'))} \qquad j \le t.$$

Steps 3 and 4: Mask, then Softmax

Position t is trained to predict what comes *next*, so it may not look ahead: keep the scores with $j \le t$ and discard the rest (causal masking). Softmax over the survivors:

$$p_t(j) = \frac{\exp(\text{score}(t \to j))}{\sum_{j' \le t} \exp(\text{score}(t \to j'))} \qquad j \le t.$$

Every term is positive and the denominator is their sum, so

$$p_t(j) \ge 0, \qquad \sum_{j \le t} p_t(j) = 1.$$

$p_t(\cdot)$ is a probability distribution over the positions $1, \dots, t$: the chance that position t attends to each one.

Step 5: Average the Values

The output at position t is those probabilities applied to the values:

$$o_t = \sum_{j \le t} p_t(j)\, v_j.$$

weights sum to one, so the output stays on the same scale as the values

a bigger weight means that position contributes more

Step 5: Average the Values

The output at position t is those probabilities applied to the values:

$$o_t = \sum_{j \le t} p_t(j)\, v_j.$$

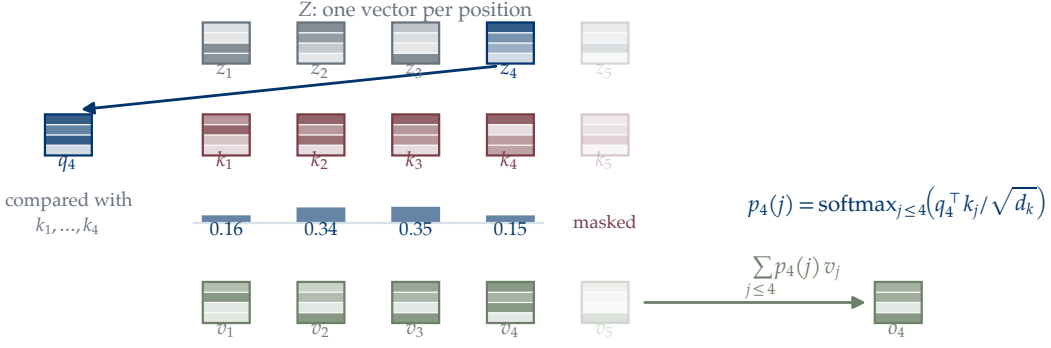
weights sum to one, so the output stays on the same scale as the values

a bigger weight means that position contributes more

A **convex combination**: o_t lies in the convex hull of the values it averaged.

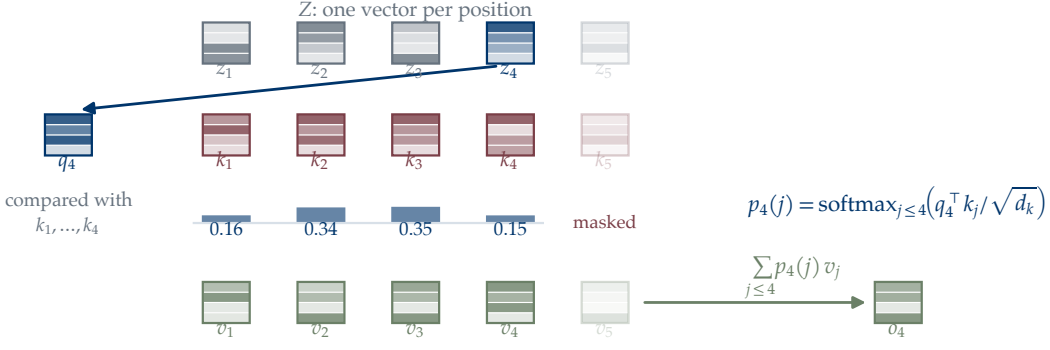
One Position at a Time

Position 4 looks at positions 1 to 4.



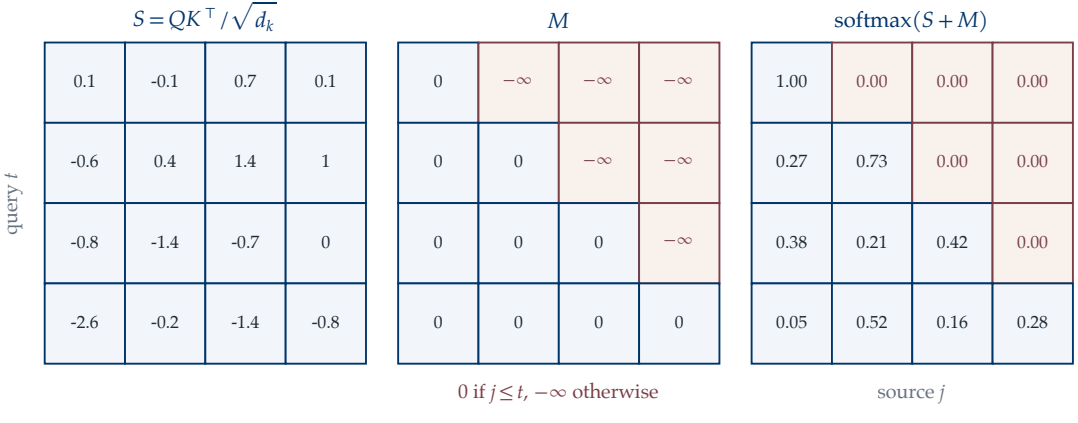
One Position at a Time

Position 4 looks at positions 1 to 4.



Position 5 contributes nothing — it has not happened yet.

The Causal Mask

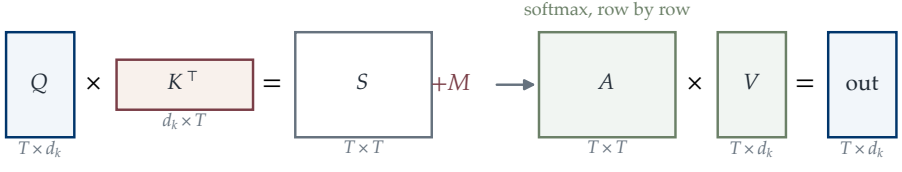


Discarding is done by adding $-\infty$ *before* the softmax, not by zeroing afterwards: $e^{-\infty} = 0$, so blocked weights vanish and each surviving row still sums to one.

Matrix Form

All T positions at once, with the mask inside the softmax:

$$S = \frac{QK^\top}{\sqrt{d_k}}, \qquad A = \mathrm{softmax}(S + M), \qquad \mathrm{Attn} = A\,V.$$

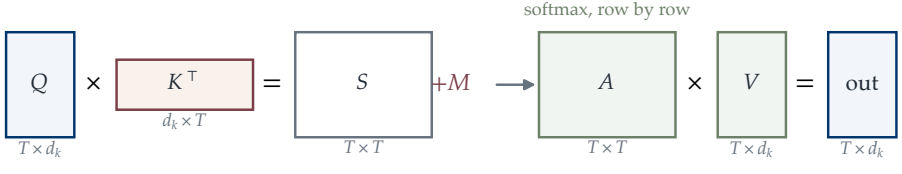


$QK^\top$ is the only step where different positions meet

Matrix Form

All T positions at once, with the mask inside the softmax:

$$S = \frac{QK^\top}{\sqrt{d_k}}, \qquad A = \mathrm{softmax}(S + M), \qquad \mathrm{Attn} = A\,V.$$



$QK^\top$ is the only step where different positions meet

Every position is computed in one pass — what makes training parallel.

Multi-Head Attention

One weighted average per position is one relationship. A pronoun may need its antecedent, the verb it attaches to, *and* the topic — so run H of them at once. Each **head** does the whole calculation on its own projections,

$$A_r = \mathrm{Attn}(ZW_r^Q,\, ZW_r^K,\, ZW_r^V) \in \mathbb{R}^{T\times d_k}, \qquad \boxed{d_k = d/H}$$

and the results are concatenated and mixed:

$$\mathrm{MHA}(Z) = \mathrm{Concat}(A_1, \dots, A_H)\, W^O, \qquad W^O \in \mathbb{R}^{d\times d}.$$

Multi-Head Attention

One weighted average per position is one relationship. A pronoun may need its antecedent, the verb it attaches to, *and* the topic — so run H of them at once. Each **head** does the whole calculation on its own projections,

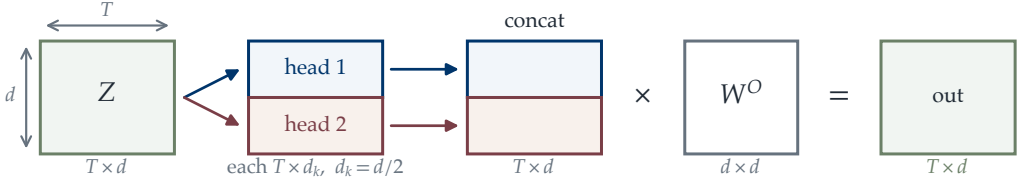
$$A_r = \mathrm{Attn}(ZW_r^Q,\, ZW_r^K,\, ZW_r^V) \in \mathbb{R}^{T\times d_k}, \qquad \boxed{d_k = d/H}$$

and the results are concatenated and mixed:

$$\mathrm{MHA}(Z) = \mathrm{Concat}(A_1, \dots, A_H)\, W^O, \qquad W^O \in \mathbb{R}^{d\times d}.$$

H slices of width d/H concatenate back to width d, and W^O is square — so MHA maps $T \times d$ to $T \times d$.

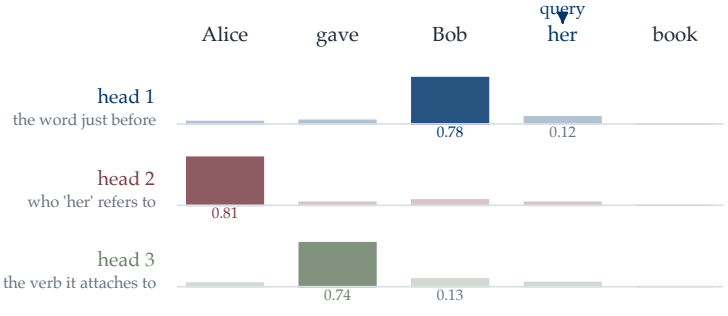
Splitting the Width Across Heads



two half-height slices stack back to full height, and W^O is square: same shape in, same shape out

Drawn to scale: width is T, height is d. Each head output is half as tall because $d_k = d/2$; the two stack back to full height, and W^O is square. So H different averages are taken in the time of one, and the shape survives.

What Different Heads Pick Up

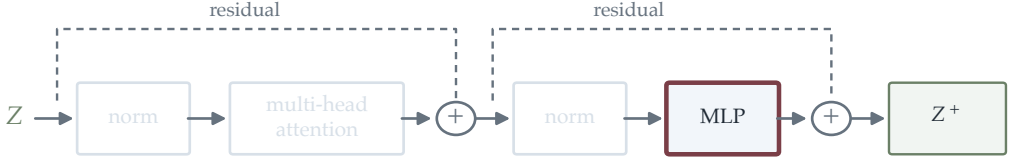


one query, three heads, three different answers; all three reach the output at 'her'

The same query word, three heads. One takes the word just before, one finds what the pronoun refers to, one finds the verb — and all three feed the output at that position. Nothing assigns these roles; the loss finds them useful.

Weights are illustrative, not measurements from a trained model.

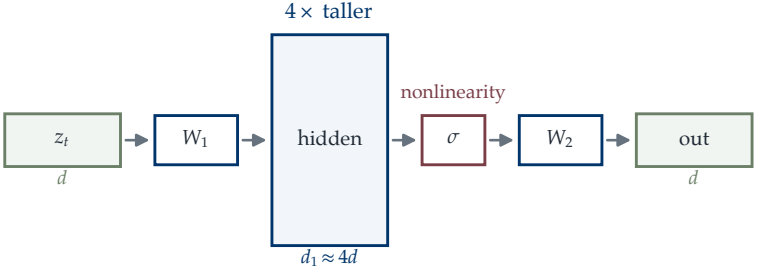
Attention Done. Now: the MLP



one block: $T \times d$ in, $T \times d$ out

Attention is finished: positions have been mixed. What remains acts on each position on its own.

The Position-Wise MLP

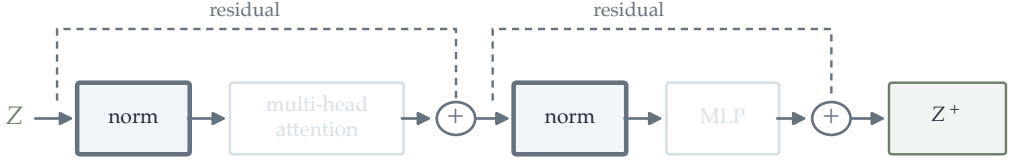


the same two matrices at every position; positions are never mixed

Two matrices with a nonlinearity between them, applied to each position separately and with the same weights everywhere. The hidden layer is four times as wide: $d \to d_1 \approx 4d$ and back, which is 49,152 for GPT-3. Most of a model's parameters live here.

Current models use a gated **SwiGLU** in place of the plain ReLU layer; see appendix D.1.

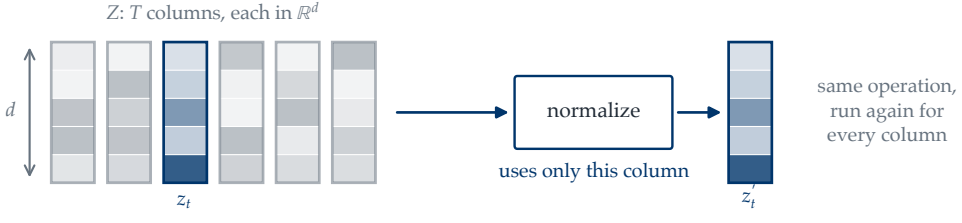
Normalization Layer



one block: $T \times d$ in, $T \times d$ out

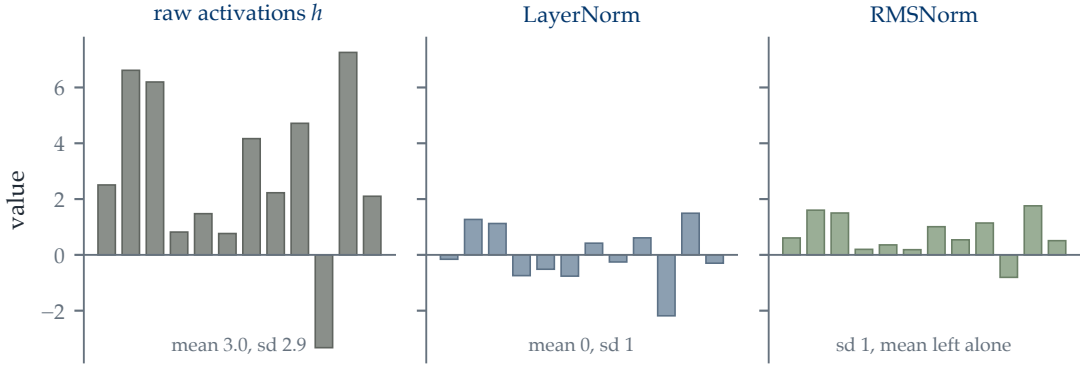
The last piece, and the one that makes deep stacks trainable.

What Normalization Acts On



A column of Z is one token's vector in $\mathbb{R}^d$. Normalization rescales that column on its own, using only its own entries, and is then run again for every position. It never mixes positions.

Normalization Keeps the Scale Fixed



$\mathrm{LN}(x) = g \odot \frac{x-\mu}{\sigma} + b$ versus $\mathrm{RMSNorm}(x) = g \odot \frac{x}{\mathrm{RMS}(x)}$, with $\mathrm{RMS}(x) = \sqrt{\frac{1}{d}\sum_i x_i^2}$. Only g and b are learnable parameters; μ, σ are computed from x itself. RMSNorm drops the mean subtraction, and is what current models use.

Ba et al. (2016) for LayerNorm; Zhang and Sennrich (2019) for RMSNorm.

The Transformer Block

A block alternates the two operations, each on a residual path:

$$\widetilde{Z} = Z + \mathrm{MHA}(\mathrm{LN}(Z)), \qquad Z^{+} = \widetilde{Z} + \mathrm{MLP}(\mathrm{LN}(\widetilde{Z})).$$

The Transformer Block

A block alternates the two operations, each on a residual path:

$$\widetilde{Z} = Z + \mathrm{MHA}(\mathrm{LN}(Z)), \qquad Z^{+} = \widetilde{Z} + \mathrm{MLP}(\mathrm{LN}(\widetilde{Z})).$$

- Attention moves information across token positions;
- The same MLP transforms each position separately with shared weights;
- Tensor shape is always $T \times d$ — preserved by normalization, attention, and the MLP;
- Residual paths let the stack refine rather than replace.

The Transformer Block

A block alternates the two operations, each on a residual path:

$$\widetilde{Z} = Z + \mathrm{MHA}(\mathrm{LN}(Z)), \qquad Z^{+} = \widetilde{Z} + \mathrm{MLP}(\mathrm{LN}(\widetilde{Z})).$$

- Attention moves information across token positions;
- The same MLP transforms each position separately with shared weights;
- Tensor shape is always $T \times d$ — preserved by normalization, attention, and the MLP;
- Residual paths let the stack refine rather than replace.

The norm sits *inside* the residual branch — see appendix D.2.

Counting the Parameters of a Block

With width d and the usual hidden size $4d$ in the MLP,

$$\underbrace{4d^2}_{W^Q, W^K, W^V, W^O} + \underbrace{8d^2}_{\text{two MLP matrices}} = 12d^2 \text{ per block.}$$

Counting the Parameters of a Block

With width d and the usual hidden size $4d$ in the MLP,

$$\underbrace{4d^2}_{W^Q, W^K, W^V, W^O} + \underbrace{8d^2}_{\text{two MLP matrices}} = 12d^2 \text{ per block.}$$

For GPT-3, $d = 12{,}288$ and $L = 96$, so $12Ld^2 \approx 1.74 \times 10^{11}$ — the headline 175 billion parameters, up to embeddings.

Counting the Parameters of a Block

With width d and the usual hidden size $4d$ in the MLP,

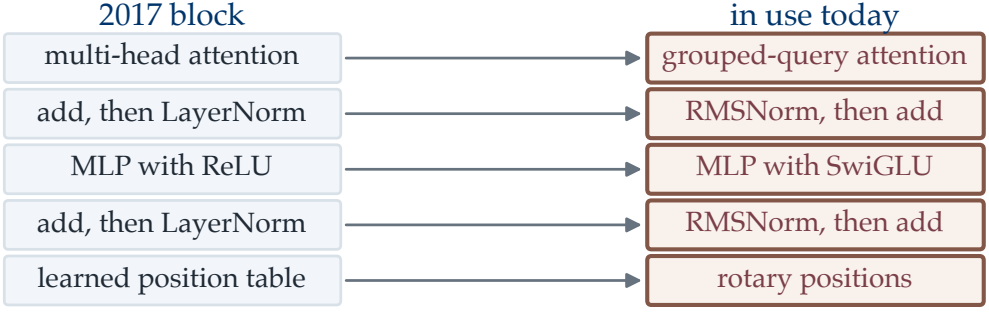
$$\underbrace{4d^2}_{W^Q, W^K, W^V, W^O} + \underbrace{8d^2}_{\text{two MLP matrices}} = 12d^2 \text{ per block.}$$

For GPT-3, $d = 12{,}288$ and $L = 96$, so $12Ld^2 \approx 1.74 \times 10^{11}$ — the headline 175 billion parameters, up to embeddings.

Depth and width are the only two knobs in that formula, which is why model families are described by a handful of integers.

Source: Configuration from Brown et al. (2020), Table 2.1.

What Changed Since 2017

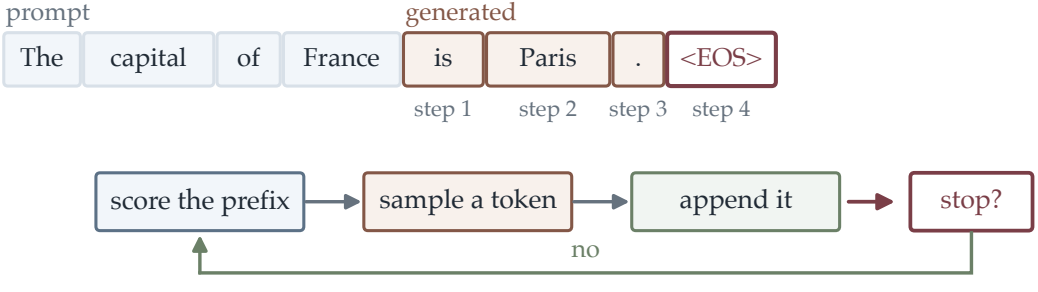


Pre-norm and RMSNorm for stability at depth, rotary for relative distance, gating for capacity, and grouped-query attention to shrink the cache.

Components as described in the Llama and PaLM model reports.

Outline

Generating Until the Stop Token



Score the prefix, choose one token, append it, repeat. The loop ends when the model emits the stop token it was trained to emit — the same token the corpus put between documents.

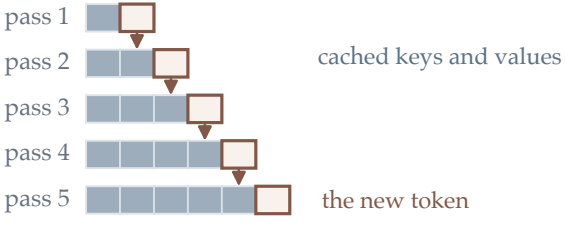
Parallel Training, Sequential Generation

training: teacher-forced

one pass, all T positions

generation: autoregressive

T forward passes, one token each



The next input does not exist until a token has been chosen. Caching the prefix helps; nothing removes the one-token-at-a-time dependence.

Choosing the Next Token: Temperature

The loop needs one token per step. The network returns **logits** $f_\theta(x_{<t}) \in \mathbb{R}^V$, and a **temperature** $\tau > 0$ rescales them before the softmax:

$$p_\tau(v) = \frac{\exp(f_\theta(x_{<t})_v / \tau)}{\sum_{v'} \exp(f_\theta(x_{<t})_{v'} / \tau)}$$

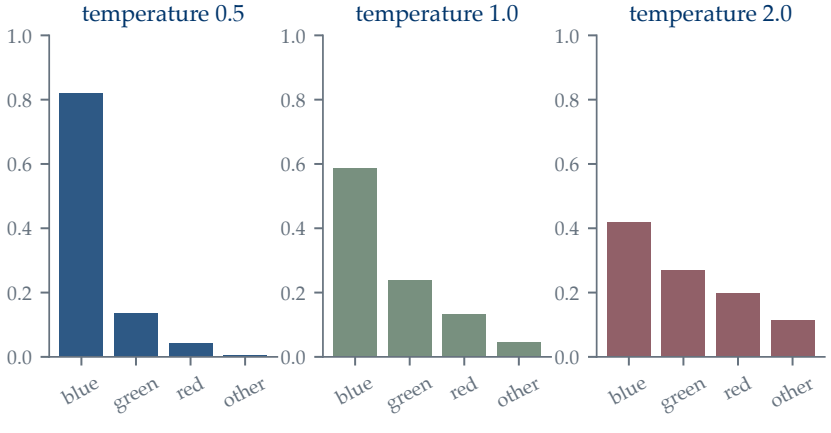
Choosing the Next Token: Temperature

The loop needs one token per step. The network returns **logits** $f_\theta(x_{<t}) \in \mathbb{R}^V$, and a **temperature** $\tau > 0$ rescales them before the softmax:

$$p_\tau(v) = \frac{\exp(f_\theta(x_{<t})_v / \tau)}{\sum_{v'} \exp(f_\theta(x_{<t})_{v'} / \tau)}$$

- $\tau \to 0$: all the mass moves to the top logit, so sampling becomes $\arg\max_v f_\theta(x_{<t})_v$ — **greedy decoding** (fully deterministic).
- $\tau = 1$: the distribution the model was trained to produce.
- τ large: the gaps shrink and the distribution flattens towards uniform.

Temperature Changes Concentration



The logits are fixed; only τ moves. No parameter changes — and the assistant goes from repetitive to erratic.

Probabilities proportional to $\exp(z_i/\tau)$ on one fixed logit vector.

What Models Actually Ship With

	temperature	also
OpenAI API default	1.0	top-p 1.0
Qwen, non-thinking	0.7	top-p 0.95, top-k 20
Qwen, thinking	0.6	top-p 0.95, top-k 20

What Models Actually Ship With

	temperature	also
OpenAI API default	1.0	top-p 1.0
Qwen, non-thinking	0.7	top-p 0.95, top-k 20
Qwen, thinking	0.6	top-p 0.95, top-k 20

Served models sample rather than take the argmax, at a temperature near one: diversity is wanted, and greedy decoding produces repetitive text.

Lecture 24 takes up the truncation rules — top-p, top-k — that sit alongside temperature.

Source: OpenAI API reference; Qwen deployment documentation.

Summary: LLM Pretraining

$$p_\theta(x_t \mid x_{<t}) = \mathrm{softmax}(f_\theta(x_{<t}))_{x_t}, \qquad \mathcal{L}(\theta) = -\frac{1}{N_{\text{tok}}} \sum_t \log p_\theta(x_t \mid x_{<t}).$$

1. Text supplies its own targets, so a corpus of n tokens is n labelled examples and one datapoint is one sequence.
2. The loss is the multiclass cross-entropy of Lecture 8 with $V \approx 10^5$ classes, optimized by AdamW over one or two epochs.
3. Held-out loss follows power laws in size and data; a training budget picks Chinchilla, and a serving budget picks something smaller.

Summary: Transformer Architecture

$$\widetilde{Z} = Z + \mathrm{MHA}(\mathrm{LN}(Z)), \qquad Z^{+} = \widetilde{Z} + \mathrm{MLP}(\mathrm{LN}(\widetilde{Z})).$$

1. f_θ is embedding, then L identical blocks, then unembedding: a map from T indices to a $T \times V$ matrix of logits.
2. Each block alternates communication across positions with computation at each position, both on residual paths.
3. Scaling by $\sqrt{d_k}$ keeps softmax gradients alive, the causal mask keeps every conditional honest, and rotary positions make attention read relative distance.

Next Lecture

Pretraining imitates recorded text. An assistant has to be useful, and no corpus contains a label for that.

Lecture 23 — Language-Model Posttraining asks where a reward for a whole response comes from, and how the policy gradients of Lectures 20–21 apply once the response is the action.

Reading for this lecture: D2L Chapter 11, Attention Mechanisms and Transformers; Vaswani et al. (2017).

Appendix

Five groups of notes skipped in the lecture hour.

A. **Scaling and cost.** Where $C \approx 6ND$ comes from, the Chinchilla minimization, and what a served model spends memory on.

B. **Scores and softmax.** Why the scores are divided by $\sqrt{d_k}$, and how a softmax is computed without overflowing.

C. **Position.** The three places positional information can enter, and how RoPE puts it in a rotation.

D. **Inside the block.** The gated activation in the MLP, and where the normalization sits.

E. **Further reading.**

A.1 Where $C \approx 6 \cdot N \cdot D$ Comes From

derive

Count multiply–adds, two floating-point operations each.

- **Forward**: every weight is used once per token, so $2 \cdot N$ operations per token.
- **Backward**: gradients with respect to activations and with respect to weights each cost about as much as the forward pass, so $4 \cdot N$.

Summing over D tokens gives $C \approx (2 + 4) \cdot N \cdot D = 6 \cdot N \cdot D$.

The count ignores attention score matrices, whose cost is $O(T)$ per token and therefore negligible while $T \ll N/d$; it also ignores the embedding and output projections. Reported training budgets nevertheless match it within a few percent.

A.2 Minimizing the Chinchilla Loss

derive

Write the budget as $ND = K$ with $K = C/6$, and substitute $D = K/N$ into $L = E + AN^{-\alpha} + BD^{-\beta}$:

$$f(N) = A\,N^{-\alpha} + B\,K^{-\beta}N^{\beta}.$$

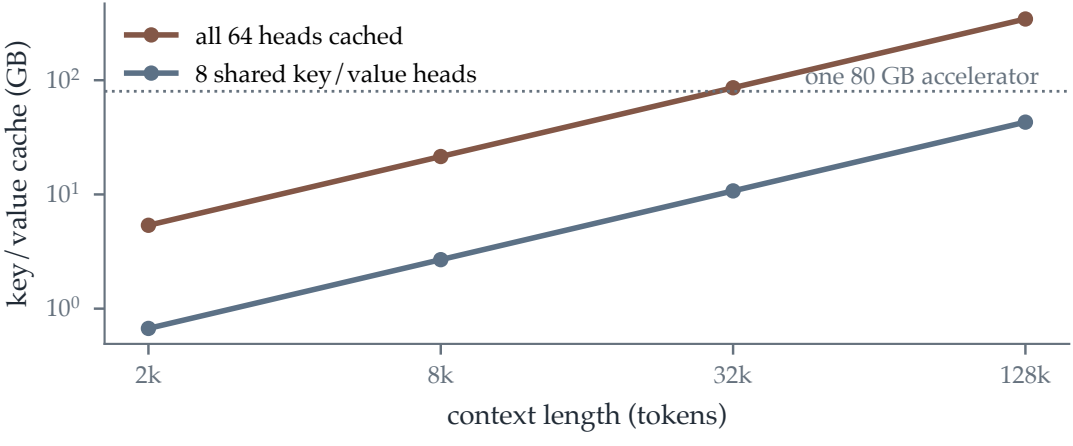
Setting $f'(N) = -\alpha A N^{-\alpha-1} + \beta B K^{-\beta} N^{\beta-1} = 0$ and collecting powers of N,

$$N^{\alpha+\beta} = \frac{\alpha A}{\beta B}\,K^{\beta} \qquad \Longrightarrow \qquad N^{\star} \propto K^{\beta/(\alpha+\beta)} \propto C^{\beta/(\alpha+\beta)},$$

and $D^{\star} = K/N^{\star} \propto C^{\alpha/(\alpha+\beta)}$.

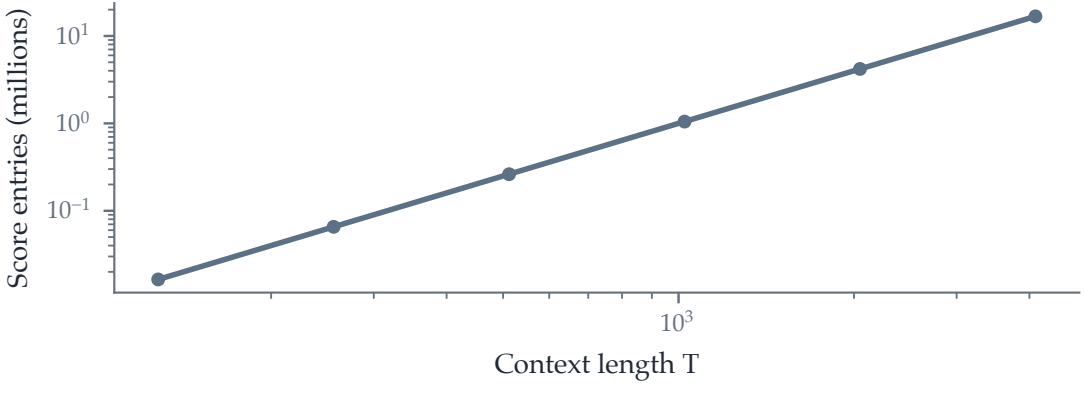
With $\alpha = .34$ and $\beta = .28$ the exponents are .45 and .55: both close enough to $\frac{1}{2}$ that the ratio $D^{\star}/N^{\star}$ is nearly constant over the fitted range, which is what makes a single number like 20 usable at all.

A.3 What the Cache Costs



The cache holds one key and one value per position, per layer, per head. It grows linearly with the conversation and is re-read at every step, so a served model spends most of its memory bandwidth here.

A.4 Why Long Context Is Expensive



The score matrix is $T \times T$, so attention cost grows quadratically in the context length while the rest of the block grows linearly.

B.1 Why the Scores Grow Like $\sqrt{d_k}$

derive

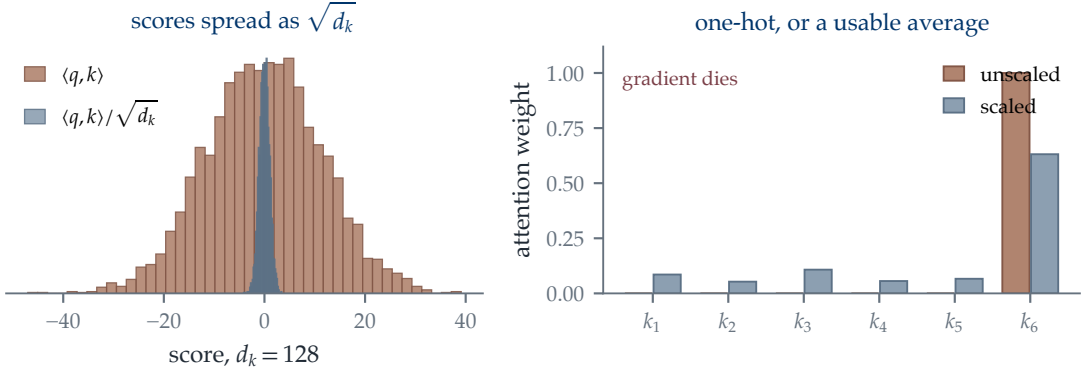
Suppose the coordinates q_j, k_j are independent, mean zero, variance one. The score is a sum of d_k independent terms,

$$\langle q, k\rangle = \sum_{j=1}^{d_k} q_j k_j, \qquad \mathrm{Var}(q_j k_j) = \mathbb{E}[q_j^2]\mathbb{E}[k_j^2] = 1,$$

so independence gives $\mathrm{Var}(\langle q, k\rangle) = d_k$ and a typical magnitude of $\sqrt{d_k}$. Dividing by $\sqrt{d_k}$ returns unit variance at every width.

At $d_k = 128$ — GPT-3's head width — the unscaled scores sit near ± 11; softmax of scores that far apart puts essentially all mass on one entry, and its Jacobian $\mathrm{diag}(a) - aa^\top$ is then numerically zero.

B.2 The Scaling, Measured



Unscaled scores spread out as $\sqrt{d_k}$, and softmax of scores that far apart puts nearly all mass on one entry — where its gradient is numerically zero. Dividing restores unit spread at every width.

B.3 The Stable Softmax

derive

Let $m = \max_u z_u$. Multiplying numerator and denominator by e^{-m} leaves the value untouched,

$$\operatorname{softmax}(z)_v = \frac{e^{z_v}}{\sum_u e^{z_u}} = \frac{e^{z_v - m}}{\sum_u e^{z_u - m}},$$

but every exponent now satisfies $z_v - m \le 0$: each term lies in $(0, 1]$, the largest is exactly 1, and the denominator is at least 1.

Taking the log of that expression, without ever forming the probability,

$$\log \operatorname{softmax}(z)_v = z_v - m - \log \sum_u e^{z_u - m}.$$

A probability of 10^{-45} underflows to 0 in `float32`, and $\log 0 = -\infty$. The right-hand side never computes one.

C.1 Three Places to Put Position

The additive table of the main line is the simplest of three families.

at the input $h_t^{(0)} = E[x_t] + P[t]$, with P fixed sinusoids or a learned table. Absolute: the vector says which slot, not which gap.

inside the softmax Leave the input alone and add a term to the logit, $\langle q_i, k_j \rangle + b_{i-j}$. Relative: only the gap enters.

on q and k Rotate the two vectors by an angle set by their positions, so the gap falls out of the inner product. Relative, and this is **RoPE**.

C.1 Three Places to Put Position

The additive table of the main line is the simplest of three families.

at the input $h_t^{(0)} = E[x_t] + P[t]$, with P fixed sinusoids or a learned table. Absolute: the vector says which slot, not which gap.

inside the softmax Leave the input alone and add a term to the logit, $\langle q_i, k_j \rangle + b_{i-j}$. Relative: only the gap enters.

on q and k Rotate the two vectors by an angle set by their positions, so the gap falls out of the inner product. Relative, and this is **RoPE**.

Only the first needs a row per position, and only the first runs out past the trained length.

C.2 Adding a Bias Inside the Softmax

Write the score with an extra term that depends only on the gap:

$$e_{ij} = \frac{\langle q_i, k_j \rangle}{\sqrt{d_k}} + b_{i-j}.$$

C.2 Adding a Bias Inside the Softmax

Write the score with an extra term that depends only on the gap:

$$e_{ij} = \frac{\langle q_i, k_j \rangle}{\sqrt{d_k}} + b_{i-j}.$$

- **T5** learns b as a scalar per head, on bucketed distances, so far-apart pairs share a parameter.
- **ALiBi** does not learn it at all: $b_{i-j} = -m \cdot (i - j)$ with m a fixed head-specific slope, a linear penalty for distance.
- **Shaw et al.** instead add a learned vector to the *key* before the product.

Source: Shaw et al. (2018), arXiv:1803.02155; T5, Raffel et al. (2020); ALiBi, Press et al. (2022).

C.3 RoPE: Rotating q and k

derive

Rotate each vector by an angle proportional to its position, $q_m \mapsto R_m q_m$ and $k_n \mapsto R_n k_n$. Because R is a rotation, $R_m^\top R_n = R_{n-m}$, so

$$\boxed{\langle R_m q_m,\, R_n k_n \rangle = q_m^\top R_{n-m}\, k_n}$$

C.3 RoPE: Rotating q and k

derive

Rotate each vector by an angle proportional to its position, $q_m \mapsto R_m q_m$ and $k_n \mapsto R_n k_n$. Because R is a rotation, $R_m^\top R_n = R_{n-m}$, so

$$\boxed{\langle R_m q_m,\, R_n k_n \rangle = q_m^\top R_{n-m}\, k_n}$$

The score depends on the positions only through $n - m$. Each pair of coordinates is rotated at its own rate, $\theta_i = 10000^{-2(i-1)/d}$, so different pairs resolve different scales of distance.

C.3 RoPE: Rotating q and k

derive

Rotate each vector by an angle proportional to its position, $q_m \mapsto R_m q_m$ and $k_n \mapsto R_n k_n$. Because R is a rotation, $R_m^\top R_n = R_{n-m}$, so

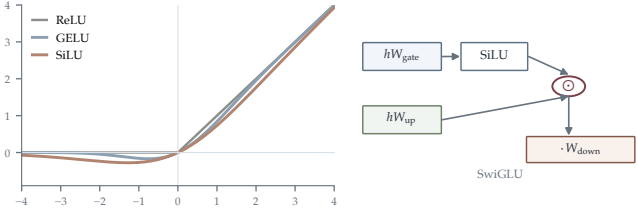
$$\boxed{\langle R_m q_m,\, R_n k_n \rangle = q_m^\top R_{n-m}\, k_n}$$

The score depends on the positions only through $n - m$. Each pair of coordinates is rotated at its own rate, $\theta_i = 10000^{-2(i-1)/d}$, so different pairs resolve different scales of distance.

Absolute rotation applied to each vector; relative distance falls out of the inner product. Nothing is stored per position, so there is no table to exhaust.

Source: Su et al. (2021), *RoFormer*, arXiv:2104.09864, equation (16).

D.1 The Activation Inside the MLP



SiLU $\text{SiLU}(x) = x \cdot \sigma(x)$, a smooth ReLU keeping a small gradient for negative inputs.

SwiGLU Two projections multiplied elementwise, $\text{SwiGLU}(z) = (\text{SiLU}(zW_{\text{gate}})) \odot (zW_{\text{up}})$, then $\cdot W_{\text{down}}$: one branch gates the other.

SwiGLU needs three matrices, so d_1 is cut to about $\frac{2}{3} \cdot 4d$ to keep the parameter count unchanged.

Source: Shazeer (2020), "GLU Variants Improve Transformer," arXiv:2002.05202.

D.2 Pre-Norm and Post-Norm

The two places to put the normalization:

$$\underbrace{Z^+ = Z + \mathrm{MLP}(\mathrm{LN}(Z))}_{\textbf{pre-norm}}, \qquad \underbrace{Z^+ = \mathrm{LN}(Z + \mathrm{MLP}(Z))}_{\textbf{post-norm}}.$$

D.2 Pre-Norm and Post-Norm

The two places to put the normalization:

$$\underbrace{Z^+ = Z + \mathrm{MLP}(\mathrm{LN}(Z))}_{\textbf{pre-norm}}, \qquad \underbrace{Z^+ = \mathrm{LN}(Z + \mathrm{MLP}(Z))}_{\textbf{post-norm}}.$$

- Pre-norm leaves the residual path untouched, so Z reaches the output unmodified and gradients reach early blocks directly. Deep stacks train without a warmup schedule.
- Post-norm normalizes the sum, putting a normalization on the identity path. This is the 2017 arrangement, and it needs careful warmup past a few dozen layers.

GPT-2 onward, and every current open model, use pre-norm.

Source: Xiong et al. (2020), "On Layer Normalization in the Transformer Architecture," arXiv:2002.04745.

E.1 Further Reading: Visual Guides

- Alammar, *The Illustrated Transformer* — https://jalammar.github.io/illustrated-transformer/
- Dugas, *The GPT-3 Architecture, on a Napkin* — https://dugas.ch/artificial_curiosity/GPT_architecture.html
- Georgia Tech, *Transformer Explainer* — https://poloclub.github.io/transformer-explainer/
- Raschka, *RoPE versus Absolute Positional Embeddings* — https://sebastianraschka.com/faq/docs/rope-vs-absolute-positional-embeddings.html
- Photon Lines, *A Visual Guide to Transformers* — https://photonlines.substack.com