TREATISE

Irreducible

The mathematics of the autoregressive Transformer

A complete, notation-stable treatise on building, training, and running an LLM: tokens, residual stream, attention, SwiGLU, cross-entropy, Jacobians, AdamW, and inference. Every symbol defined where it is used. Loss is driven toward zero. It never arrives.

CONTENTS

00 The object of study
01 Tokenization and targets
02 Embeddings and position
03 The pre-norm Transformer block
04 Causal attention
05 MLP and SwiGLU
06 Unembedding to probabilities
07 Loss, entropy, and scaling
08 Why loss is not zero
09 Backpropagation and Jacobians
10 AdamW, clip, and the schedule
11 The training loop
12 Inference and the KV cache
13 End-to-end map and catalogue

The object of study

An LLM is a map from token IDs to logits. Training only ever sees next-token conditionals.

What is being built

An LLM, in the sense used throughout this treatise, is a single differentiable function from a sequence of discrete token IDs to a matrix of logits — one row per position, one column per vocabulary item.

$$f_\theta : \{1, \dots, V\}^T \to \mathbb{R}^{T \times V} \tag{0.1}$$

V	vocabulary size (number of distinct tokens)	T	sequence length of this forward pass
d	model / residual-stream width	n	number of Transformer layers
θ	every learnable number in the network	$x_{1:T}$	token IDs, each x_t \in \{1,\dots,V\}

WHY IT MATTERS Training never “understands a paragraph as a whole” as a primitive. It only learns next-token conditionals. Every later capability is a side effect of making those conditionals accurate.

The probabilistic model is autoregressive. The joint over a sequence factorizes into a product of next-token conditionals. There is no separate “sentence model.”

$$p_\theta(x_{1:T}) = \prod_{t=1}^{T} p_\theta(x_t \mid x_{<t}) \tag{0.2}$$

$x_{<t}$	(x_1,\dots,x_{t-1}); empty when t = 1	$p_\theta(x_t \mid x_{<t})$	the model’s categorical over V tokens at step t

Two modes of the same network

The same function f_θ is used in two incompatible ways. Training has a correct next token sitting in a dataset, so a loss exists, so a gradient exists, so θ moves. Inference has no such token. The weights freeze. Only the forward map runs.

THE WHOLE MACHINE, IN ONE SENTENCE

Training differentiates the map from token IDs to next-token distributions and walks θ downhill on cross-entropy. Use-time throws the derivative away and only evaluates the map.

Forward map. Training attaches CE → Jacobians → AdamW after p. Inference samples from p.

Notation that does not change

Every later chapter uses the same letters. Residuals are $H^{(\ell)}$ with rows $h_t^{(\ell)} \in \mathbb{R}^d$. Attention is the only mixer. Everything else is applied positionwise. The unembedding produces logits $Z \in \mathbb{R}^{T \times V}$, and softmax turns a row into p_t.

UNITS

Cross-entropy in this treatise is in nats (natural log), which is what `torch.nn.CrossEntropyLoss` reports. Bits are nats divided by $\ln 2$. Perplexity is $e^{\mathcal{L}}$. Bits-per-byte is the tokenizer-fair comparison.

Tokenization and targets

Text becomes integers. Loss, perplexity, and “zero” are defined on this vocabulary, not on English words.

Raw text becomes integers

A tokenizer — almost always BPE, SentencePiece, or Unigram — cuts raw bytes into a finite vocabulary. The model never sees characters as characters. From this point on, every quantity that can go to zero is defined on those integers.

A training example is one sequence $x_{1:T}$. The supervision target at position t is the next token x_{t+1}. Position T either has no target, or sequences are packed so every position has a successor.

$$y_t \in \{0,1\}^V, \qquad (y_t)_{x_{t+1}} = 1 \tag{1.1}$$

y_t	one-hot target at position t	x_{t+1}	the true next token in the document
V	vocabulary size, typically 32k–256k		

WHY IT MATTERS Loss, perplexity, and “what zero loss would mean” live on this discrete vocabulary, not on English words. A larger vocabulary usually raises per-token loss even if the model is a better compressor.

PACKING

Documents are concatenated and split into length-T windows, with a document-boundary mask so attention does not leak across unrelated texts. Padding tokens are excluded from the loss average by a mask $\mathcal{M}$.

The causal shift

In code this is a one-position shift: logits at index t predict token $t+1$. The first token has no predecessor; the last logit is either unused or predicts the first token of the next

packed span.

$$\text{input } x_{1:T} \quad \longrightarrow \quad \text{target } x_{2:T+1} \tag{1.2}$$

Embeddings and position

Tokens enter a vector space. RoPE injects order by rotating query/key pairs.

Token embedding

A learned matrix $W_E \in \mathbb{R}^{V \times d}$ stores one row per token. Looking up token x_t is a gather; equivalently, a one-hot times W_E.

$$e_t^{\text{tok}} = (W_E)_{x_t,:} \in \mathbb{R}^d \tag{2.1}$$

W_E	token embedding matrix, V×d
e_t^{tok}	the d-vector for token x_t

$$E^{\text{tok}} = X_{\text{oh}} W_E \in \mathbb{R}^{T \times d} \tag{2.2}$$

X_{oh}	one-hot sequence, T×V

Absolute position (GPT-2 style)

Without a position signal the Transformer is a bag of tokens. Absolute embeddings add a learned vector per index.

$$h_t^{(0)} = e_t^{\text{tok}} + (W_P)_{t,:} \tag{2.3}$$

W_P	learned positions, T_max × d
$h_t^{(0)}$	residual-stream vector at layer 0

RoPE (Llama-style default)

Rotary position embeddings add no extra vector to the residual stream. After Q and K are formed, each even/odd pair of coordinates is rotated by an angle that depends on position. Inner products then depend on the difference $t - s$, not on absolute t and s separately.

$$\theta_i = \omega^{-2i/d_h} \tag{2.4}$$

d_h head dimension (even)

i pair index 0, 1, …, d_h/2 − 1

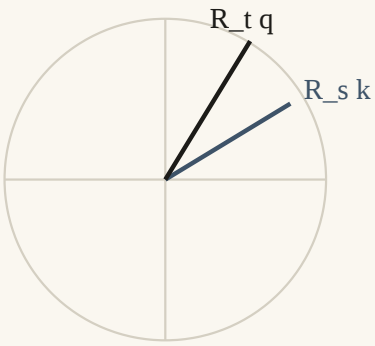
ω base frequency, usually 10,000 (or a long-context variant)

$$R_{t,i} = \begin{pmatrix} \cos(t\theta_i) & -\sin(t\theta_i) \\ \sin(t\theta_i) & \cos(t\theta_i) \end{pmatrix} \tag{2.5}$$

$$\tilde{q}_{t,(2i:2i+1)} = R_{t,i}\, q_{t,(2i:2i+1)}, \qquad \tilde{k}_{s,(2i:2i+1)} = R_{s,i}\, k_{s,(2i:2i+1)} \tag{2.6}$$

WHY IT MATTERS The rotation is orthogonal, so it preserves norms. The backward pass is the inverse rotation (the transpose). Relative position is baked into the attention score, which is why RoPE extrapolates farther than absolute embeddings.

ONE RoPE PAIR



⟨R_t q, R_s k⟩ = ⟨q, R_{t-s} k⟩

Inner product depends only on t − s.

Backward pass: apply $R^{\top}$, which equals R^{-1}.

JACOBIAN OF A ROTATION

For $R \in \mathrm{SO}(2)$, $R^\top R = I$ and $R^{-1} = R^\top$. If $q = Ru$ then $\partial L/\partial u = R^\top(\partial L/\partial q)$. No extra parameters; the angles are functions of index, not of θ.

The pre-norm Transformer block

Residual highways, RMSNorm, and the two sublayers that every modern decoder stacks.

Pre-norm residual block

Modern decoder LLMs stack n identical pre-norm blocks. Each block writes a residual delta into the stream, twice: once from attention, once from the MLP.

$$Z^{(\ell)} = H^{(\ell-1)} + \mathrm{Attn}\Big(\mathrm{Norm}(H^{(\ell-1)})\Big) \qquad (3.1)$$
$$H^{(\ell)} = Z^{(\ell)} + \mathrm{MLP}\Big(\mathrm{Norm}(Z^{(\ell)})\Big)$$

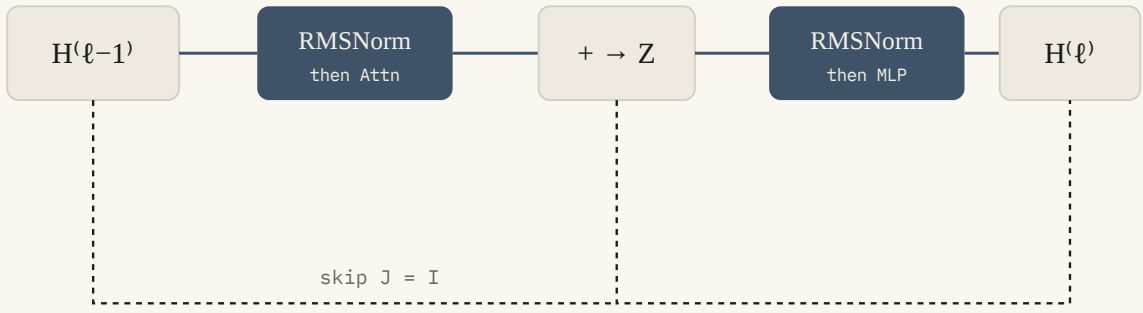
ℓ layer index, 1 … n

$H^{(\ell)}$ residual stream after layer ℓ, T×d

$Z^{(\ell)}$ stream after the attention residual

WHY IT MATTERS Residuals are the highway that lets gradients travel through dozens of layers. The skip has Jacobian I, so there is always a path that does not vanish.

PRE-NORM BLOCK



Two residual adds. Each skip contributes a Jacobian of I, which is why depth trains.

RMSNorm

RMSNorm scales a vector by its root-mean-square and a learned gain. It does not subtract the mean and has no bias. Llama-style models use it.

$$\mathrm{RMS}(u) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} u_i^2 + \varepsilon} \tag{3.2}$$

$u \in \mathbb{R}^d$ one residual-stream vector

ε small constant, e.g. 10^{-6}

$$\mathrm{RMSNorm}(u) = \gamma \odot \frac{u}{\mathrm{RMS}(u)} \tag{3.3}$$

$\gamma \in \mathbb{R}^d$ learned scale

$\odot$ elementwise multiply

WHY IT MATTERS Norms keep residual-stream scale stable so attention scores and MLP activations do not explode as depth grows.

LayerNorm

LayerNorm also subtracts the mean and usually has a bias. GPT-2 uses it. The extra projection orthogonal to $\mathbf{1}$ is the only structural difference.

$$\hat{u} = \frac{u - \mu \mathbf{1}}{\sqrt{\sigma^2 + \varepsilon}}, \qquad \mathrm{LN}(u) = \gamma \odot \hat{u} + \beta \tag{3.4}$$

μ mean of u

σ^2 variance of u

$\beta \in \mathbb{R}^d$ learned bias

WHERE THE NORM SITS

Pre-norm (norm on the block input) is the current default because it trains more stably at depth. Post-norm (norm after the residual add) was the original Transformer and is harder to optimize without careful residual scaling.

Causal attention

The only place tokens mix. Scale, mask, softmax, GQA, and the softmax Jacobian.

Queries, keys, values

From normalized input $U \in \mathbb{R}^{T\times d}$, three linear maps produce Q, K, V. They are then split into H heads of width $d_h = d/H$. If RoPE is used, Q and K are rotated now.

$$Q = UW_Q, \quad K = UW_K, \quad V = UW_V \tag{4.1}$$

W_Q, W_K, W_V	projections in R^{d × d} (or d × H d_h)	Q_h, K_h, V_h	head h, each T × d_h

Scores, scale, causal mask, softmax

$$S_h = \frac{Q_h K_h^\top}{\sqrt{d_h}} \in \mathbb{R}^{T\times T} \tag{4.2}$$

WHY IT MATTERS The √d_h scale keeps a typical dot product O(1). Without it, softmax saturates as d_h grows and gradients vanish.

Future tokens must be invisible. The causal mask writes $-\infty$ above the diagonal so those entries become zero after softmax.

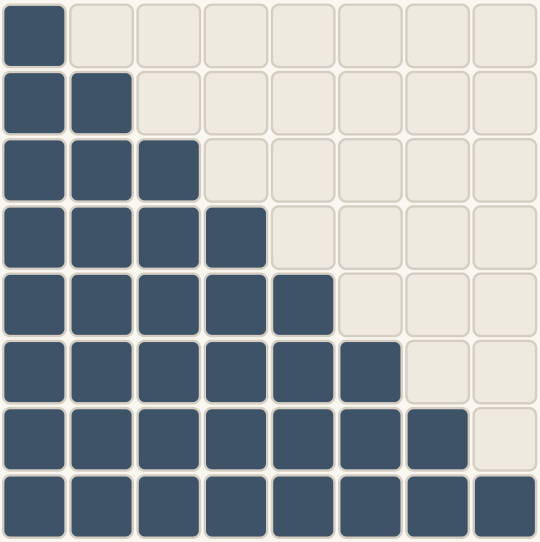
$$M_{t,s} = \begin{cases} 0 & s \le t \\ -\infty & s > t \end{cases} \tag{4.3}$$

$$A_h = \mathrm{softmax}(S_h + M), \qquad (A_h)_{t,s} = \frac{\exp((S_h + M)_{t,s})}{\sum_{s'} \exp((S_h + M)_{t,s'})} \tag{4.4}$$

$(A_h)_{t,:}$	a probability distribution over past-and-present positions

WHY IT MATTERS Attention is the only place tokens mix. Everything else is applied positionwise. The causal mask is what makes the model a next-token predictor rather than a bidirectional encoder.

CAUSAL MASK · T = 8



Steel: visible (M = 0). Paper: future (M = −∞).

$$O_h = A_h V_h, \qquad O = [O_1 \mid \cdots \mid O_H] W_O \tag{4.5}$$

W_O	output projection, d × d
O	Attn(U), written back into the residual stream

Softmax Jacobian

For one row $a = \mathrm{softmax}(z) \in \mathbb{R}^T$:

$$J_{\text{softmax}} = \frac{\partial a}{\partial z} = \operatorname{diag}(a) - aa^\top \in \mathbb{R}^{T \times T} \tag{4.6}$$

$$\frac{\partial a_i}{\partial z_j} = a_i(\delta_{ij} - a_j) \tag{4.7}$$

WHY IT MATTERS This Jacobian is symmetric, has null space along 1 (softmax is shift-invariant), and is positive semidefinite on the simplex tangent space. You will meet it again in the p − y collapse.

Grouped-query and multi-query attention

Llama-3-class models use fewer key/value heads than query heads. The math is the same; (K, V) are shared across groups. This shrinks the KV cache at inference, which is the memory that actually bounds generation length.

COMPLEXITY AND FLASH ATTENTION

Naive attention is $O(T^2 d_h)$ time and $O(T^2)$ memory for the score matrix. Flash Attention computes the same mathematical softmax attention in tiles, keeping the $T \times T$ matrix in SRAM rather than materializing it in HBM. The Jacobian is unchanged; the IO is not.

MLP and SwiGLU

Per-token computation and the wide matrices that store features.

Two-layer GELU FFN (GPT-2)

Attention moves information between positions. The MLP is the per-token computation: feature mixing, and a large fraction of stored knowledge in the wide matrices.

$$\mathrm{MLP}(u) = W_{\downarrow}\,\phi(W_{\uparrow}u + b_{\uparrow}) + b_{\downarrow} \tag{5.1}$$

- ϕ GELU
- $W_{\uparrow}$ up-projection, typically to 4d
- $W_{\downarrow}$ down-projection, back to d

$$\mathrm{GELU}(z) \approx z\,\sigma(1.702\,z) \tag{5.2}$$

SwiGLU (Llama)

Gated linear units replace the single nonlinearity with a product of a SiLU-gated branch and a linear “up” branch. There is no bias in the usual Llama formulation.

$$\mathrm{SwiGLU}(u) = W_D^{\top}\Big(\mathrm{SiLU}(W_G^{\top}u) \odot (W_U^{\top}u)\Big) \tag{5.3}$$

- W_G, W_U gate and up, d × d_ff
- W_D down, d_ff × d
- d_{ff} inner width, often ~ 8d/3 so parameter count matches 4d GELU

$$\mathrm{SiLU}(z) = z \cdot \sigma(z), \qquad \sigma(z) = \frac{1}{1 + e^{-z}} \tag{5.4}$$

$$\mathrm{SiLU}'(z) = \sigma(z) + z\,\sigma(z)\big(1 - \sigma(z)\big) \tag{5.5}$$

WHY IT MATTERS You need this derivative in the backward pass. The extra σ(z)(1−σ(z)) term is the sigmoid Jacobian.

WHERE THE PARAMETERS LIVE

In a typical 7B-class model, the MLP matrices are the majority of θ. Attention is the mixer; the MLP is the memory.

Unembedding to probabilities

A final norm, a linear map into V dimensions, and a categorical distribution per position.

Final norm and unembedding

After n blocks we have $H^{(n)} \in \mathbb{R}^{T \times d}$. A last norm, then a linear map into the vocabulary, produces logits.

$$\tilde{H} = \mathrm{Norm}(H^{(n)}), \qquad Z = \tilde{H} W_U \in \mathbb{R}^{T \times V} \tag{6.1}$$

W_U unembedding matrix, d × V

z_t row t of Z: logits at position t

Some models tie $W_U = W_E^\top$. Tied embeddings save parameters and couple the geometry of "looking up a token" with "predicting a token."

From logits to a categorical

$$p_t = \mathrm{softmax}(z_t), \qquad (p_t)_k = \frac{e^{z_{t,k}}}{\sum_{j=1}^{V} e^{z_{t,j}}} \tag{6.2}$$

p_t a probability distribution over the vocabulary

$p_t(x_{t+1})$ probability assigned to the true next token

WHY IT MATTERS This is the model's entire prediction: one categorical distribution per position. Everything before this was geometry in R^d. This is the interface to discrete text.

NUMERICAL STABILITY

Softmax is shift-invariant: $\mathrm{softmax}(z) = \mathrm{softmax}(z - c\mathbf{1})$. Implementations subtract $\max_k z_{t,k}$ before the exponential so the largest entry is 0 and nothing overflows. The Jacobian is unchanged.

Loss, entropy, and scaling

Cross-entropy is the only pretraining signal. It decomposes into entropy plus KL.

Token-level cross-entropy

The training signal is next-token negative log-likelihood. PyTorch `CrossEntropyLoss` on logits is exactly this, in nats.

$$\mathcal{L}_t = -\log p_t(x_{t+1}) = -z_{t,x_{t+1}} + \log \sum_{j=1}^{V} e^{z_{t,j}} \tag{7.1}$$

$\mathcal{L}_t$ NLL of the true next token, in nats

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{M}|} \sum_{t \in \mathcal{M}} \mathcal{L}_t \tag{7.2}$$

$\mathcal{M}$ positions that are not padding / not masked

WHY IT MATTERS This scalar is the only training signal in pretraining. Backprop exists solely to make L smaller.

Entropy plus KL

If $P^\star$ is the true next-token distribution of the data, the expected loss splits into two terms. Training can only shrink the second.

$$\mathbb{E}[\mathcal{L}] = H(P^\star) + D_{\mathrm{KL}}(P^\star \,\|\, p_\theta) \tag{7.3}$$

$H(P^\star)$	entropy of language: irreducible uncertainty	D_{KL}	extra loss from the model being wrong

$H(P^\star) > 0$ for open language, so $\mathcal{L} \to 0$ is not achievable on a general corpus. Fitted irreducible floors on web-scale English tokenizers are often around $E \approx 1.7$–1.8 nats/token.

Perplexity and bits-per-byte

$$\mathrm{PPL} = \exp(\mathcal{L}) \tag{7.4}$$

Loss 2.0 nats/token is perplexity $e^{2.0} \approx 7.39$. Loss 1.69 is about 5.4. Read it as: “as uncertain as choosing among about 5–7 equally likely tokens,” on average.

Loss is not comparable across tokenizers. Bits-per-byte (or bits-per-character) is the fairer comparison. Classic estimates of English entropy are roughly 0.6–1.3 bits per character; frontier models are often cited around ~0.7 bits per character on diverse English — already near old Shannon bounds on many corpora.

> CONVERSION
> $\text{bits} = \text{nats} / \ln 2$. A 100k-token vocabulary has random loss $\ln 100000 \approx 11.5$ nats.

Scaling-law shape

This is a forecast of pretraining loss, not a training step. DeepMind’s Chinchilla fit:

$$L(N, D) = E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}} \tag{7.5}$$

N	parameter count
D	training tokens
E	estimated irreducible loss
$AN^{-\alpha}$	penalty for finite model size
$BD^{-\beta}$	penalty for finite data

Their original fit put $E \approx 1.69$ nats/token. A later replication put E closer to 1.82. Those numbers are estimates for a data mix and a tokenizer, not a universal constant.

STAGE	LOSS (NATS)	MEANING
Random initialization	`ln V ≈ 10.8–11.8`	Uniform over 50k–128k tokens
Early training	`~4–6`	Crude word and syntax statistics
Small modern pretrain	`~2.0–2.5`	Competent next-token model
Strong large pretrain	`~1.7–2.1`	Near fitted irreducible floors
Narrow SFT, regular data	`~0.2–0.8`	The task is less uncertain than web text
Overfit a tiny set	`→ 0`	Memorization, not general language

Why loss is not zero

Context collapses meaning. It does not collapse the next token. The floor is entropy, not tautology.

Context kills branching. It does not kill the string.

Conditional entropy falls as context grows. That is not controversial.

$$H(\text{next token} \mid \text{more context}) \leq H(\text{next token} \mid \text{less context}) \tag{8.1}$$

If the prompt is only “After tomorrow is”, the calendar is unpinned. Supply “Today is Sunday” and Tuesday is the unique calendar answer. Ambiguity was unfinished context. Semantic collapse is real.

What does not follow is: therefore the next token becomes unique. All of these can be true on Sunday:

- After tomorrow is Tuesday.
- After tomorrow’s Tuesday.
- After tomorrow it will be Tuesday.
- After tomorrow is the 16th.
- After tomorrow? Tuesday.

Same fact. Different tokens. The training objective is not “recover the unique true proposition.” It is “assign probability to the exact next token that happened to be in this document.”

Two layers, easy to mix

Semantic collapse. Given enough context, the intended meaning often becomes unique. “After tomorrow,” plus “today is Sunday,” plus “we are talking about weekdays,” leaves one date.

Realizational leftover. How that meaning gets worded, whether the speaker continues the sentence at all, what they mention next, in what register — that usually stays open.

THE CLEAN CLAIM

Given enough context, many local meanings become unique. Given enough context, the next token usually does not.

A Born-rule picture is a fair analogy for *sense*: many readings exist while the sentence is incomplete; context makes one reading definite. It is a weaker analogy for the next token, because the "measurement" here is "whatever this particular writer typed," and many typings are compatible with the same collapsed meaning.

Why the floor is not mystical

A huge fraction of remaining loss is missing context, not linguistic randomness. The data-generating process is many speakers, not one oracle with one lawful continuation. The model is not given the full world: it gets a window of text, not "it is Sunday and the speaker intends to name the weekday."

Shannon already measured this with humans. People who had seen the preceding text still could not guess the next character with certainty. Human conditional entropy of English is low, not zero. An LLM is trying to match that same process.

WHEN ZERO IS REALISTIC

If the task were "output the unique true weekday," loss could approach zero on that task. Overfit a tiny repeated dataset and training loss can approach 0 — memorization, not general language. Next-token pretraining on open text is a harder and sloppier task: imitate the exact continuation this document happened to take.

Backpropagation and Jacobians

Reverse-mode AD, the p − y collapse, RMSNorm, attention, SwiGLU, and the skip whose Jacobian is I.

Chain rule and Jacobians

Training is gradient descent on $\mathcal{L}(\theta)$. Backprop is reverse-mode automatic differentiation: one backward sweep computes $\nabla_\theta \mathcal{L}$. If $u \in \mathbb{R}^m$, $v = g(u) \in \mathbb{R}^k$, and $\mathcal{L}$ is scalar:

$$J_g = \frac{\partial v}{\partial u} \in \mathbb{R}^{k \times m}, \qquad \frac{\partial \mathcal{L}}{\partial u} = J_g^\top \frac{\partial \mathcal{L}}{\partial v} \tag{9.1}$$

J_g Jacobian of the local map

$J_g^\top v$ vector-Jacobian product (VJP) — what is actually computed

In practice nobody materializes giant Jacobians. Frameworks compute VJPs. For a matrix multiply $Y = XW$, with incoming gradient $G = \partial \mathcal{L}/\partial Y$:

$$\frac{\partial \mathcal{L}}{\partial X} = G W^\top, \qquad \frac{\partial \mathcal{L}}{\partial W} = X^\top G \tag{9.2}$$

WHY IT MATTERS Those two identities run through almost the entire Transformer. Every linear map is this pair.

Softmax + cross-entropy collapses

This is the most important gradient in the whole system.

$$\mathcal{L}_t = -\sum_k (y_t)_k \log (p_t)_k \tag{9.3}$$

Using $J_{\text{softmax}} = \operatorname{diag}(p) - pp^\top$:

$$\frac{\partial \mathcal{L}_t}{\partial z_t} = p_t - y_t \tag{9.4}$$

$$\left(\frac{\partial \mathcal{L}_t}{\partial z_t}\right)_k = (p_t)_k - \mathbf{1}[k = c], \qquad c = x_{t+1} \tag{9.5}$$

WHY IT MATTERS You do not backprop through softmax and log separately. The error signal is "probability assigned minus truth." Overconfident wrong tokens get a large push; already-correct peaked predictions get a small one.

Into the residual stream and W_U

$$\frac{\partial \mathcal{L}}{\partial \tilde{H}} = \left(\frac{\partial \mathcal{L}}{\partial Z}\right) W_U^\top, \qquad \frac{\partial \mathcal{L}}{\partial W_U} = \tilde{H}^\top \left(\frac{\partial \mathcal{L}}{\partial Z}\right) \tag{9.6}$$

Then the final Norm Jacobian maps $\partial \mathcal{L}/\partial \tilde{H}$ back to $\partial \mathcal{L}/\partial H^{(n)}$.

RMSNorm Jacobian (one vector)

Let $r = \operatorname{RMS}(u)$, $\hat{u} = u/r$, $v = \gamma \odot \hat{u}$.

$$\frac{\partial r}{\partial u} = \frac{u}{d\,r}, \qquad \frac{\partial \hat{u}}{\partial u} = \frac{1}{r}\left(I - \frac{uu^\top}{d\,r^2}\right) \tag{9.7}$$

Then $\partial \mathcal{L}/\partial u$ follows by the product rule with γ. LayerNorm is the same idea after projecting orthogonal to $\mathbf{1}$ (mean removal).

Attention backward

Fix one head. Forward: $S = QK^\top/\sqrt{d_h}$, $A = \mathrm{softmax}(S + M)$, $O = AV$. Given $G_O = \partial\mathcal{L}/\partial O$:

$$\frac{\partial \mathcal{L}}{\partial A} = G_O V^\top, \qquad \frac{\partial \mathcal{L}}{\partial V} = A^\top G_O \tag{9.8}$$

$$\frac{\partial \mathcal{L}}{\partial S_{t,:}} = J_{\mathrm{softmax}}(A_{t,:})^\top \left(\frac{\partial \mathcal{L}}{\partial A_{t,:}}\right)^\top \tag{9.9}$$

$$\frac{\partial \mathcal{L}}{\partial Q} = \frac{1}{\sqrt{d_h}}\left(\frac{\partial \mathcal{L}}{\partial S}\right) K, \qquad \frac{\partial \mathcal{L}}{\partial K} = \frac{1}{\sqrt{d_h}}\left(\frac{\partial \mathcal{L}}{\partial S}\right)^\top Q \tag{9.10}$$

RoPE’s backward pass is the inverse rotation. Then $Q = UW_Q$ etc. give

$$\frac{\partial \mathcal{L}}{\partial W_Q} = U^\top \frac{\partial \mathcal{L}}{\partial Q}, \qquad \frac{\partial \mathcal{L}}{\partial U} \mathrel{+}= \frac{\partial \mathcal{L}}{\partial Q} W_Q^\top \tag{9.11}$$

The same for K, V. Multi-head: sum the $\partial\mathcal{L}/\partial U$ contributions.

Residual Jacobian

$$Z = H + F(H) \implies \frac{\partial \mathcal{L}}{\partial H} = \frac{\partial \mathcal{L}}{\partial Z} + J_F^\top \frac{\partial \mathcal{L}}{\partial Z} \tag{9.12}$$

WHY IT MATTERS The first term is the skip connection. That is why deep Transformers train: there is always a path whose Jacobian is I.

SwiGLU backward

Let $g = W_G^\top u$, $\text{up} = W_U^\top u$, $s = \text{SiLU}(g)$, $h = s \odot \text{up}$, $\text{out} = W_D^\top h$.

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial h} &= W_D \frac{\partial \mathcal{L}}{\partial \text{out}} \\ \frac{\partial \mathcal{L}}{\partial s} &= \frac{\partial \mathcal{L}}{\partial h} \odot \text{up}, \qquad \frac{\partial \mathcal{L}}{\partial \text{up}} = \frac{\partial \mathcal{L}}{\partial h} \odot s \\ \frac{\partial \mathcal{L}}{\partial g} &= \frac{\partial \mathcal{L}}{\partial s} \odot \text{SiLU}'(g) \end{aligned} \tag{9.13}$$

Then ordinary matmul rules for W_G, W_U, W_D, u.

Embedding backward

$\partial \mathcal{L} / \partial H^{(0)}$ is scattered back into the rows of W_E (and W_P if used):

$$\frac{\partial \mathcal{L}}{\partial (W_E)_{x_t,:}} \mathrel{+}= \frac{\partial \mathcal{L}}{\partial h_t^{(0)}} \tag{9.14}$$

THAT IS THE FULL BACKWARD PASS

Repeat for every layer from n down to 1, then embeddings. The scalar $\mathcal{L}$ has been routed to every weight.

LAYERNORM JACOBIAN, ENTRYWISE

After mean removal, $\hat{u}$ lives in the subspace orthogonal to 1. If $P = I - \mathbf{1}\mathbf{1}^\top/d$ and $s = \sqrt{\sigma^2 + \varepsilon}$, then $\partial\hat{u}/\partial u = s^{-1}P - s^{-3}(Pu)(Pu)^\top/d$. The RMSNorm Jacobian above is the same formula without P.

AdamW, clip, and the schedule

Turning a gradient into a new θ without destroying a billion-parameter run.

Vanilla SGD is not what runs

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L} \tag{10.1}$$

Raw gradients in a Transformer have wild scale differences across layers. LLMs use AdamW: per-parameter adaptive steps, decoupled weight decay, a warmup-cosine schedule, and global-norm clipping.

AdamW

Let $g_t = \nabla_\theta \mathcal{L}_t$ at optimizer step t.

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^{\odot 2} \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \\ \theta_t &= \theta_{t-1} - \eta_t \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} + \lambda \theta_{t-1} \right) \end{aligned} \tag{10.2}$$

η_t scheduled learning rate	β_1 first-moment decay, typically 0.9
β_2 second-moment decay, 0.95 or 0.999	λ weight decay, applied to θ directly (the “W”)
ε 10^{-8}, avoid divide-by-zero	$g^{\odot 2}$ elementwise square

WHY DECOUPLED DECAY

Adam already rescales by $\sqrt{\hat{v}}$. L2 regularization mixed into the gradient would be rescaled too, so large-gradient coordinates would be decayed less. AdamW applies $\lambda\theta$ outside that rescaling. That is the entire difference from Adam.

Warmup then cosine decay

$$\eta_t = \begin{cases} \eta_{\max} \cdot t / t_{\text{warm}} & t \le t_{\text{warm}} \\ \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos \pi \frac{t - t_{\text{warm}}}{t_{\text{total}} - t_{\text{warm}}}\right) & t > t_{\text{warm}} \end{cases} \tag{10.3}$$

Gradient clipping

Before AdamW:

$$g \leftarrow g \cdot \min\left(1, \frac{c}{\|g\|_2}\right) \tag{10.4}$$

c clip threshold, often 1.0	$\|g\|_2$ global L2 norm of the concatenated gradient

WHY IT MATTERS Adam normalizes per-parameter. Weight decay stops ‖θ‖ drifting. Clipping stops rare batches from destroying the run. The schedule starts gentle, then anneals so the model settles.

The training loop

Batch, forward, loss, backward, accumulate, clip, step. Train loss is not validation loss.

What actually runs

Repeat millions of steps:

1. Sample a batch of sequences. Shape: $B \times T$ token IDs.
2. Forward: embeddings → n blocks → logits $Z \in \mathbb{R}^{B \times T \times V}$.
3. Compute $\mathcal{L}$ on next-token targets (ignore pad / masked positions).
4. Backward: reverse-mode AD gives $g = \nabla_\theta \mathcal{L}$.
5. Optionally accumulate g over several micro-batches to fake a larger batch.
6. Clip g, AdamW update, zero grads.
7. Periodically evaluate validation loss on held-out data (no parameter update).

$$g_{\text{accum}} = \frac{1}{K} \sum_{k=1}^{K} \nabla_\theta \mathcal{L}^{(k)} \tag{11.1}$$

K number of micro-batches

$\mathcal{L}^{(k)}$ mean token loss on micro-batch k

Train loss is not the target

Batch loss is the mean of per-token losses. Mixed precision (BF16/FP16 forward, FP32 master weights and loss) is an implementation detail; the math above is the same. The loss itself is computed in FP32 so the log-sum-exp is not wrecked by underflow.

Training loss can keep falling by memorization. Validation loss estimates generalization. The entropy floor applies to both; validation is the one that tells you the model is not just reciting.

INITIALIZATION, BRIEFLY

Residual branches are often scaled by $1/\sqrt{2n}$ (GPT-2) or trained with μP so that width can change without retuning η. Embeddings are typically small Gaussian. The skip Jacobian still starts at I; that is the property that actually lets depth work.

Inference and the KV cache

Frozen θ. Temperature, top-k, nucleus. Prefill once, then decode with cached keys and values.

Weights freeze. Only the forward map runs.

Once training is finished, θ is frozen. There is no “correct” next token sitting in a dataset, so there is nothing to compute a loss on and therefore nothing to back-propagate.
Autoregressive generation:

1. Start with prompt tokens $x_{1:t}$.
2. Forward → logits z_t.
3. Form a sampling distribution from z_t.
4. Draw x_{t+1}, append, repeat.

Decoding math

$$(p_t)_k = \frac{\exp(z_{t,k}/\tau)}{\sum_j \exp(z_{t,j}/\tau)} \tag{12.1}$$

$\tau > 0$ temperature. τ → 0 is greedy: argmax_k z_{t,k}

Top-k. Keep the k largest logits, set the rest to $-\infty$, then softmax.

Nucleus / top-p. Keep the smallest set of tokens whose cumulative probability is at least p, then renormalize.

$$\mathcal{V}_p = \arg\min_{\mathcal{V}} \left\{ |\mathcal{V}| : \sum_{k \in \mathcal{V}} p_t(k) \geq p \right\} \tag{12.2}$$

No $\mathcal{L}$. No Jacobian. No Adam.

KV cache

Attention at step t only needs q_t and all past $k_{1:t}, v_{1:t}$. Store (K, V) per layer:

$$K^{\text{cache}} \in \mathbb{R}^{t \times d_k}, \qquad V^{\text{cache}} \in \mathbb{R}^{t \times d_v} \tag{12.3}$$

New token: compute new q_t, k_t, v_t, append k_t, v_t, attend. Cost per new token is $O(t)$ attention, not $O(t^2)$ from scratch.

PREFILL

O(T²)

One forward over the whole prompt. Every position attends to the past. Cache is filled.

DECODE

O(t) / token

New q attends to cached K, V. Append one row. No Jacobian. No optimizer.

PREFILL VERSUS DECODE

Prefill runs the full prompt in one forward pass, filling the cache. Decode is one-token-at-a-time and memory-bandwidth bound. That split, not the math of softmax, is why serving looks different from training.

This is the entire user-facing model: frozen θ, forward pass, sample. Training was the process that carved θ. Inference only reads it.

End-to-end map and catalogue

The stacked forward equation, the train path, the infer path, and what each piece is for.

Stacked forward map

Forward map. Training attaches CE → Jacobians → AdamW after p. Inference samples from p.

$$
\begin{aligned}
x_{1:T} &\xrightarrow{W_E,\ \text{pos/RoPE}} H^{(0)} \\
H^{(\ell-1)} &\xrightarrow{\text{Norm, Attn, }+} Z^{(\ell)} \xrightarrow{\text{Norm, MLP, }+} H^{(\ell)} \\
H^{(n)} &\xrightarrow{\text{Norm, } W_U} Z \xrightarrow{\text{softmax}} p_{1:T}
\end{aligned}
\tag{13.1}
$$

Train path

$$
p_{1:T},\ x_{2:T+1} \xrightarrow{\text{CE}} \mathcal{L} \xrightarrow{\text{reverse-mode AD / Jacobians}} \nabla_\theta \mathcal{L} \xrightarrow{\text{AdamW}} \theta_{\text{new}}
\tag{13.2}
$$

Infer path

$$
z_t \xrightarrow{\tau,\ \text{top-}k/p} \tilde{p}_t \xrightarrow{\text{sample}} x_{t+1}
\tag{13.3}
$$

What each piece is for

The whole machine is one differentiable map from token IDs to next-token distributions. Training differentiates that map and walks θ downhill on cross-entropy. Use-time throws the derivative away and only evaluates the map.

PIECE	JOB
Tokenizer	Discrete interface to text
Embedding + position	Put tokens into a space where order exists
Residual stream	Common currency every layer reads and writes
Norm	Control scale so depth does not explode
Causal attention	Move information from past tokens to the current one
MLP / SwiGLU	Per-token computation and stored features
Unembed + softmax	Turn vectors into a distribution over V
Cross-entropy	Scalar “how wrong was the next-token guess”
Softmax–CE gradient p − y	Error signal into logits
Jacobians / backprop	Route that scalar error to every weight
Skip-connection Jacobian I	Make deep nets trainable
AdamW + schedule + clip	Stable updates at billion-parameter scale
Validation loss / entropy floor	Measure generalization; zero is not the target
Sampling + KV cache	Use the trained pθ without changing it

End matter

PIECE	JOB
Tokenizer	Discrete interface to text
Embedding + position	Put tokens into a space where order exists
Residual stream	Common currency every layer reads and writes
Norm	Control scale so depth does not explode
Causal attention	Move information from past tokens to the current one
MLP / SwiGLU	Per-token computation and stored features
Unembed + softmax	Turn vectors into a distribution over V
Cross-entropy	Scalar “how wrong was the next-token guess”
Softmax–CE gradient p − y	Error signal into logits
Jacobians / backprop	Route that scalar error to every weight
Skip-connection Jacobian I	Make deep nets trainable
AdamW + schedule + clip	Stable updates at billion-parameter scale
Validation loss / entropy floor	Measure generalization; zero is not the target
Sampling + KV cache	Use the trained p_θ without changing it