The Anatomy of a Modern Frontier Language Model

An Expository Companion to "Attention Is All You Need," Nine Years On

Prepared for a mathematically mature reader new to machine learning

August 2026

Abstract

The Transformer of Vaswani et al. (2017) is the ancestor of every frontier large language model (LLM), yet a 2026 frontier model shares surprisingly few components with it. The encoder is gone; absolute position embeddings are gone; LayerNorm, post-norm residual placement, the ReLU feed-forward network, and the learned output softmax over a translation vocabulary have all been replaced. This paper is a self-contained mathematical exposition of the architecture that replaced them: a *pre-norm, decoder-only Transformer* with RMS normalization, rotary position embeddings, grouped-query or latent attention, gated (SwiGLU) feed-forward networks, and—at the largest scale—sparse mixture-of-experts layers, trained by maximum likelihood on next-token prediction with a handful of stabilizing auxiliary terms. We define every object from scratch, prove the small results that motivate the design choices (the relative-position property of rotary embeddings, the variance calculus behind normalization and initialization, the correctness of the online softmax underlying FlashAttention and of speculative decoding), give reference implementations in PyTorch-style Python, and close with the systems arithmetic —memory, FLOPs, and parallelism—that ultimately dictates which mathematics survives contact with hardware. Closed frontier labs do not publish their architectures; everything in the main text is grounded in the open frontier (Llama 3/4, DeepSeek-V3, Qwen 3, Mixtral, Gemma, GPT-OSS), with clearly marked asides on what is plausible but unconfirmed beyond it.

Contents

1 Preliminaries: the problem, the loss, and the notation

1.1 What a language model is

Fix a finite set $\mathcal{V}$, the *vocabulary*, with $V = |\mathcal{V}|$ elements called *tokens*. In frontier practice V is between roughly 32,000 and 260,000; tokens are byte sequences produced by a compression scheme described in §1.3, and typically correspond to words, word fragments, or individual bytes. A *language model* is a probability distribution over finite token sequences, represented through the chain rule of probability. For a sequence $x = (x_1, \ldots, x_T) \in \mathcal{V}^T$,

$$p_\theta(x_1, \ldots, x_T) \;=\; \prod_{t=1}^{T} p_\theta(x_t \mid x_1, \ldots, x_{t-1}), \tag{1}$$

where $\theta \in \mathbb{R}^P$ is the parameter vector ($P \sim 10^9$–10^{12} at the frontier). Equation (1) is an identity, not an assumption: any distribution over sequences factors this way. The modeling choice is that each conditional $p_\theta(\cdot \mid x_{<t})$ is computed by a single neural network—the same network, with the same parameters, for every position t—which maps the *prefix* $x_{<t} \coloneqq (x_1, \ldots, x_{t-1})$ to a probability vector on $\mathcal{V}$. Models of this type are called *autoregressive*, and everything a chat assistant does—answering, coding, translating—is repeated sampling from $p_\theta(\cdot \mid \text{prefix})$, appending the sample to the prefix, and iterating.

The network's raw output at position t is a vector of *logits* $z_t \in \mathbb{R}^V$, converted to probabilities by the softmax map.

Definition 1.1 (Softmax)**.** For $z \in \mathbb{R}^V$ and temperature $\tau > 0$,

$$\operatorname{softmax}(z/\tau)_i \;=\; \frac{e^{z_i/\tau}}{\sum_{j=1}^{V} e^{z_j/\tau}}, \qquad i = 1, \ldots, V.$$

Softmax is invariant under adding a constant to all coordinates ($\operatorname{softmax}(z + c\mathbf{1}) = \operatorname{softmax}(z)$), a fact used constantly in numerically stable implementations (subtract $\max_i z_i$ before exponentiating) and again in §11.2 where it makes a streaming computation of softmax possible. As $\tau \to 0^+$ the distribution concentrates on $\arg\max_i z_i$ (greedy decoding); $\tau = 1$ samples from the model's distribution as trained. The Jacobian is $\partial \operatorname{softmax}(z)_i / \partial z_j = \operatorname{softmax}(z)_i(\delta_{ij} - \operatorname{softmax}(z)_j)$, which is worth deriving once by hand because it is the innermost link of every gradient computation below.

1.2 The training objective: maximum likelihood

Let $\mathcal{D}$ be a distribution over token sequences—in practice, an enormous corpus of web text, code, books, and licensed data, on the order of 10^{13}–10^{14} tokens after filtering and deduplication. Training minimizes the negative log-likelihood, equivalently the *cross-entropy*:

$$\mathcal{L}(\theta) \;=\; -\,\mathbb{E}_{x\sim\mathcal{D}}\left[\frac{1}{T}\sum_{t=1}^{T} \log p_\theta(x_t \mid x_{<t})\right] \;=\; \mathbb{E}_{x}\left[\frac{1}{T}\sum_{t=1}^{T}\Big(\log\sum_{j\in\mathcal{V}} e^{(z_t)_j} \;-\; (z_t)_{x_t}\Big)\right]. \tag{2}$$

Two remarks connect this to things you already know.

Information-theoretic reading. If q denotes the true conditional distribution of the data, then for each prefix

$$\underbrace{-\mathbb{E}_{x_t\sim q} \log p_\theta(x_t \mid x_{<t})}_{\text{cross-entropy } H(q,p_\theta)} \;=\; \underbrace{H(q)}_{\text{entropy of the data}} \;+\; \underbrace{D_{\mathrm{KL}}(q \,\|\, p_\theta)}_{\geq 0,\; =0 \iff p_\theta = q},$$

so minimizing (2) minimizes the KL divergence to the data distribution; the irreducible floor $H(q)$ is the intrinsic entropy of language. Loss is conventionally reported in nats per token, or as *perplexity* $e^{\mathcal{L}}$, the effective branching factor.

Statistical reading. Equation (2) is the maximum likelihood estimator for the parametric family $\{p_\theta\}$. Nothing exotic: the entire "objective function" of a frontier pretraining run is MLE for a categorical GLM whose (very nonlinear) link is the network of §2–§8, optimized by stochastic first-order methods (§10.3). Auxiliary terms (load balancing, z-loss) are small additive regularizers introduced where they arise.

A crucial computational point: because of the causal structure built into the architecture (§4.2), a single forward pass on a sequence of length T produces *all T conditional distributions at once*, so (2) gives T supervised prediction problems per sequence for one network evaluation. This dense supervision is a large part of why next-token prediction is such an efficient training signal.

1.3 Tokenization in one page

Neural networks consume vectors, not text, so text must first be mapped to a token sequence. Frontier models use *byte-pair encoding* (BPE) or close variants: start with the 256 single bytes as the base vocabulary; repeatedly find the most frequent adjacent pair of tokens in a reference corpus and merge it into a new token; stop after $V - 256$ merges. Encoding applies the learned merges greedily in order. The result is a lossless, invertible compression of byte strings into sequences over $\mathcal{V}$, with common words as single tokens and rare

strings falling back to bytes. Typical English text compresses to roughly 0.7–0.8 tokens per word (≈ 4 bytes/token).

Tokenization is the one part of the pipeline that is not learned end-to-end by gradient descent, and it leaks into model behavior (arithmetic on digit groupings, character-level puzzles, multilingual efficiency). For this paper its only role is: *the model's input is a sequence of integers in* $\{1, \dots, V\}$ *and its output is a distribution over the same set.*

1.4 Notation and shape conventions

We fix notation used throughout. All vectors are *row* vectors and data matrices stack examples as rows, matching both the ML literature and the memory layout of the code: a linear map is $x \mapsto xW$ with $W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$, and maps compose left-to-right, xW_1W_2.

Symbol	Meaning	Typical frontier value
V	vocabulary size	32k–260k
T	sequence (context) length	4096 train $\to 10^5$–10^7 extended
d_{model}	residual stream width	4096–16,384
L	number of layers (blocks)	32–126
n_{h}	query heads per attention layer	32–128
n_{kv}	key/value heads ($n_{\text{kv}} \le n_{\text{h}}$)	8 typically
d_{head}	dimension per head, usually $d_{\text{model}}/n_{\text{h}}$	64–128
d_{ff}	feed-forward hidden width	$\approx \frac{8}{3} d_{\text{model}}$ (dense)
P	total parameter count	10^9–10^{12}

Activations carry shape $[B, T, d_{\text{model}}]$: batch, sequence position, feature. We suppress the batch index in mathematics (everything is applied independently per sequence) and write $X \in \mathbb{R}^{T \times d_{\text{model}}}$ for the matrix whose t-th row is the representation of position t. "Frontier" means the largest models with published architectures as of early 2026 (Llama 3/4, DeepSeek-V3/R1, Qwen 3, Gemma 3, Mixtral, GPT-OSS, Kimi K2); closed models (GPT-5-class, Claude, Gemini) are discussed only in marked asides, because their internals are not public.

We assume without further comment: linear algebra including SVD and low-rank approximation, multivariable calculus including the chain rule in Jacobian form, basic probability (expectations, variance, concentration heuristics at the level of the CLT), and elementary information theory as recalled above. Facts beyond these are stated explicitly where used.

Aside: reading (2) as a sufficient statistic for the field

Nearly every architectural choice in this paper is justified empirically by one number: the value of (2) on held-out data at matched parameter and compute budgets. The field's methodology, examined honestly, is large-scale empirical model selection over architectures, guided by scaling laws (§10.5) that make small-scale comparisons predictive of large-scale behavior. The mathematics in this paper explains *why the winning choices are sensible*—stability, invariances, computational complexity—but the arbiter was always the loss curve.

2 The decoder-only skeleton

2.1 From encoder–decoder to decoder-only

The 2017 Transformer was built for translation and had two stacks: an *encoder* that read the source sentence bidirectionally, and a *decoder* that generated the target autoregressively while attending to the encoder's output through *cross-attention*. The GPT line (2018–) observed that for pure language modeling the encoder and cross-attention are unnecessary: keep only the causal decoder stack, and condition on "input" simply by placing it earlier in the same token sequence. Every frontier LLM since is decoder-only. The word "decoder" survives only as the name of the causal masking pattern (§4.2); there is nothing left to decode *from*.

2.2 The architecture, top to bottom

Fix hyperparameters $(V, T, d_{\text{model}}, L, \dots)$ as in §1.4. The model is a map $\mathcal{V}^T \to (\Delta^{V-1})^T$ (a probability vector for every prefix) composed of three stages.

1. Embedding. A matrix $W_E \in \mathbb{R}^{V \times d_{\text{model}}}$ assigns each token $v \in \mathcal{V}$ its row $W_E[v] \in \mathbb{R}^{d_{\text{model}}}$. The input sequence $(x_1, \dots, x_T)$ becomes

$$X^{(0)} \in \mathbb{R}^{T \times d_{\text{model}}}, \qquad X_t^{(0)} = W_E[x_t].$$

In 2017 a position-dependent vector (sinusoidal "position embedding") was added here; *modern models add nothing*—position information enters inside attention via rotations (§5).

2. The residual stream and L blocks. The core of the model is a sequence of L *blocks*, each reading from and writing back to a running state $X^{(\ell)} \in \mathbb{R}^{T \times d_{\text{model}}}$ called the *residual stream*:

$$\begin{aligned} X^{(\ell+1/2)} &= X^{(\ell)} \;+\; \mathrm{Attn}^{(\ell)}\big(\mathrm{Norm}_1^{(\ell)}(X^{(\ell)})\big), \\ X^{(\ell+1)} &= X^{(\ell+1/2)} \;+\; \mathrm{MLP}^{(\ell)}\big(\mathrm{Norm}_2^{(\ell)}(X^{(\ell+1/2)})\big), \end{aligned} \qquad \ell = 0, \dots, L-1. \tag{3}$$

Here Attn is the (only) sequence-mixing operation—the sole place information moves between positions (§4); MLP acts on each position independently (§7); and Norm is RMSNorm (§3), applied to each row. In sparse models the MLP of most blocks is replaced by a mixture-of-experts layer (§8).

The arrangement in (3), with normalization *inside* the branch and the skip connection untouched, is called *pre-norm* and is universal at the frontier; the 2017 paper used *post-norm*, $X \mapsto \mathrm{Norm}(X + f(X))$, which normalizes the trunk itself. The difference matters enormously for trainability and is analyzed in §3.1.

3. Unembedding. A final normalization and a linear map to logits:

$$Z = \mathrm{Norm}_{\text{final}}\big(X^{(L)}\big)\, W_U, \qquad W_U \in \mathbb{R}^{d_{\text{model}} \times V}, \qquad p_\theta(\cdot \mid x_{\le t}) = \mathrm{softmax}(Z_t). \tag{4}$$

Row t of Z predicts token $t{+}1$. Smaller models often *tie* $W_U = W_E^\top$ to save parameters; the largest models untie them, since $2Vd_{\text{model}}$ parameters are negligible against the trunk.

2.3 The residual stream as the central object

Rewriting (3) by unrolling,

$$X^{(L)} \;=\; X^{(0)} \;+\; \sum_{\ell=0}^{L-1} \Big[\mathrm{Attn}^{(\ell)}(\cdots) + \mathrm{MLP}^{(\ell)}(\cdots)\Big], \tag{5}$$

the model is the identity map plus a sum of $2L$ learned corrections. Three consequences:

1. *Gradient flow.* Let $\mathcal{L}$ be the loss. Since $X^{(\ell+1)} = X^{(\ell)} + f_\ell(\mathrm{Norm}(X^{(\ell)}))$, the Jacobian of each block is $I + J_\ell$, so by the chain rule

$$\frac{\partial \mathcal{L}}{\partial X^{(0)}} = \frac{\partial \mathcal{L}}{\partial X^{(L)}} \prod_{\ell=L-1}^{0} (I + J_\ell) = \frac{\partial \mathcal{L}}{\partial X^{(L)}} \big(I + \text{cross terms}\big).$$

   There is an identity path from the loss to every layer's input: gradients cannot vanish by repeated multiplication of contractive Jacobians, the failure mode that made deep pre-2015 networks untrainable. (This argument is exactly the ResNet argument, and it is the reason a 100-layer stack trains at all.)

2. *A shared workspace.* Every block reads the same stream and writes into it additively. Mechanistic-interpretability work fruitfully views the stream as a communication bus of superposed features on which attention heads move information between positions and MLPs transform it in place; nothing in this paper depends on that view, but it is the right mental picture.

3. *Width is bandwidth.* d_{model} bounds how much information can flow past any layer; this is why width and depth are scaled together (§10.5).

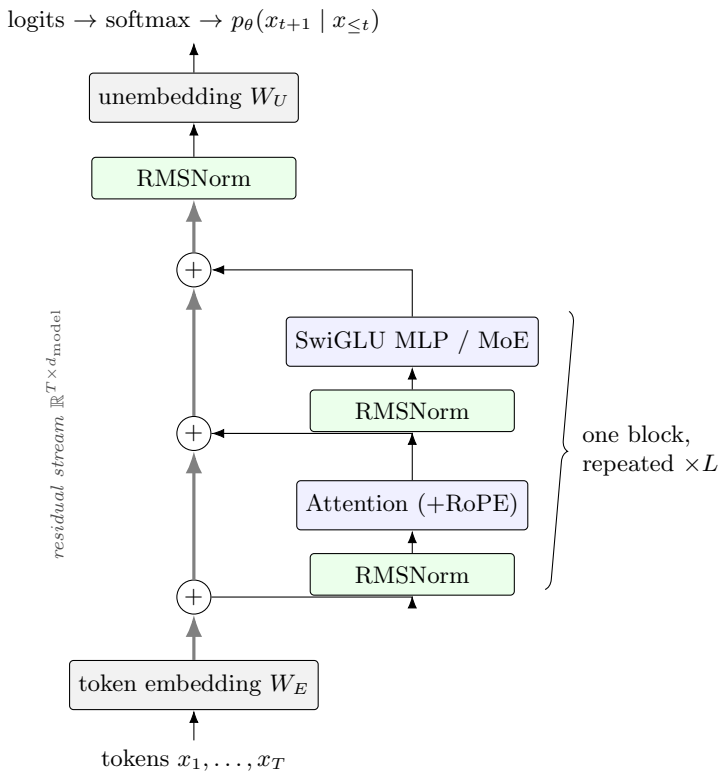

Figure 1: The decoder-only, pre-norm skeleton. The vertical spine is the residual stream; each block reads it through a normalization, computes an update in a branch, and adds the update back. Attention is the only component that mixes information across positions.

2.4 Reference implementation of the skeleton

The following PyTorch-style code is the whole model at the top level; every subsequent section fills in one class. (Real implementations differ only in fusion, precision, and caching details.)

```
import torch, torch.nn as nn

class Block(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.norm1 = RMSNorm(cfg.d_model)          # Section 3
        self.attn  = Attention(cfg)                # Sections 4-6
        self.norm2 = RMSNorm(cfg.d_model)
        self.mlp   = SwiGLU(cfg)                   # Section 7 (or MoE, Section 8)

    def forward(self, x, rope, kv_cache=None):
        x = x + self.attn(self.norm1(x), rope, kv_cache)
        x = x + self.mlp(self.norm2(x))
        return x

class Transformer(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.embed  = nn.Embedding(cfg.vocab, cfg.d_model)
        self.blocks = nn.ModuleList(Block(cfg) for _ in range(cfg.n_layers))
        self.norm_f = RMSNorm(cfg.d_model)
        self.unembed = nn.Linear(cfg.d_model, cfg.vocab, bias=False)

    def forward(self, tokens):                     # tokens: [B, T] ints
        x = self.embed(tokens)                     # [B, T, d_model]
```

```
        rope = precompute_rope(x.shape[1])          # Section 5
        for blk in self.blocks:
            x = blk(x, rope)
        return self.unembed(self.norm_f(x))         # [B, T, V] logits
```

Note what is absent relative to 2017: no position embedding added at the input, no encoder, no cross-attention, no dropout (frontier pretraining is single-epoch on effectively unbounded data, so overfitting in the classical sense is not the binding constraint), and no bias terms anywhere (§10.4).

3 Normalization: from LayerNorm to RMSNorm, from post- to pre-norm

Deep networks are compositions of hundreds of nonlinear maps; without control, the scale of activations drifts exponentially in depth, and with it the scale of gradients. Normalization layers are the control mechanism: cheap, differentiable maps that project each position's feature vector onto a set of fixed scale before it enters a block.

Definition 3.1 (LayerNorm, 2016; used in the 2017 Transformer)**.** For $x \in \mathbb{R}^d$, with learned $\gamma, \beta \in \mathbb{R}^d$ and small $\varepsilon > 0$ (e.g. 10^{-5}),

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i, \qquad \sigma^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2, \qquad \mathrm{LN}(x)_i = \gamma_i \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \beta_i.$$

Definition 3.2 (RMSNorm, 2019; universal at the frontier)**.** For $x \in \mathbb{R}^d$ with learned gain $\gamma \in \mathbb{R}^d$,

$$\mathrm{rms}(x) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \varepsilon}, \qquad \mathrm{RMS}(x)_i = \gamma_i \frac{x_i}{\mathrm{rms}(x)}.$$

Applied to a matrix $X \in \mathbb{R}^{T\times d}$, it acts row-wise (per position); no statistics are shared across positions or across the batch.

Geometrically, $x \mapsto x/\mathrm{rms}(x)$ is (up to ε and the factor $\sqrt{d}$) radial projection onto the sphere of radius $\sqrt{d}$, followed by a learned diagonal scaling. It is scale-invariant, $\mathrm{RMS}(\lambda x) = \mathrm{RMS}(x)$ for $\lambda > 0$, and differentiable away from 0 with Jacobian

$$\frac{\partial}{\partial x}\left(\frac{x}{\mathrm{rms}(x)}\right) = \frac{1}{\mathrm{rms}(x)}\left(I - \frac{x^\top x}{d\,\mathrm{rms}(x)^2}\right),$$

i.e. (to leading order) the orthogonal projector onto the tangent space of the sphere at $x/\mathrm{rms}(x)$, scaled by $1/\mathrm{rms}(x)$: gradients tangent to the sphere pass through; the radial component is killed. This is precisely the behavior wanted from a scale controller.

Why did the mean subtraction and bias β of LayerNorm disappear? Ablation showed the centering contributes essentially nothing to loss at scale, while costing an extra pass over the vector and (more importantly) an extra reduction in low-precision arithmetic; RMSNorm matches quality at lower cost. This is a recurring pattern in the 2017→2026 evolution: components survive only if they pay for themselves at matched compute.

```
class RMSNorm(nn.Module):
    def __init__(self, d, eps=1e-5):
        super().__init__()
        self.g, self.eps = nn.Parameter(torch.ones(d)), eps
    def forward(self, x):                       # x: [..., d]
        rms = torch.rsqrt(x.pow(2).mean(-1, keepdim=True) + self.eps)
        return self.g * x * rms
```

3.1 Pre-norm versus post-norm

The 2017 placement was $X \mapsto \mathrm{Norm}(X + f(X))$ (*post-norm*); the modern placement is $X \mapsto X + f(\mathrm{Norm}(X))$ (*pre-norm*), as in (3). The distinction looks cosmetic and is not.

In post-norm, the normalization sits *on the trunk*: the identity path of §2.3 is re-normalized after every block, so the end-to-end Jacobian is a product of L non-identity factors, and the clean "I + corrections" structure of (5) is destroyed. Empirically, post-norm Transformers beyond a few dozen layers diverge unless trained with a carefully tuned learning-rate *warmup*; the original Transformer's warmup schedule existed to work around exactly this. In pre-norm, the trunk is untouched; each block's contribution enters through a normalized gate, activation scale on the trunk grows only additively (empirically like $O(\sqrt{\ell})$ if block outputs were independent, since variances of sums of uncorrelated terms add), and stacks of 100+ layers train from standard initializations. The known cost of pre-norm—later blocks seeing an ever-larger trunk, so their *relative* contribution shrinks—is mild and partially addressed by the initialization scalings of §10.2.

Frontier models are pre-norm, with two refinements seen in open models and plausibly beyond:

- *Double norm / sandwich norm* (Gemma): normalize both the input *and the output* of each branch, $X + \mathrm{Norm}_{\mathrm{out}}(f(\mathrm{Norm}_{\mathrm{in}}(X)))$, bounding what a branch can write into the stream.
- *QK-norm* (§10.4): additional RMSNorms on the query and key vectors inside attention, controlling logit magnitudes at the one place in the network where inner products of learned vectors grow unboundedly.

Aside: why not BatchNorm?

Readers who have met normalization via convolutional networks may ask about BatchNorm, which normalizes each feature across the *batch*. It is unusable here: it couples examples within a batch (breaking the independence of sequences, complicating autoregressive inference where the "batch" is one token), and its train/eval statistics mismatch interacts badly with very long sequences. Per-position normalization (LayerNorm/RMSNorm) has no batch coupling and behaves identically at training and inference.

4 Attention, from scratch

Attention is the only operation in the architecture through which position t can depend on positions $s \neq t$. Everything else—normalization, MLPs, embeddings—acts on each row of X independently.

4.1 Scaled dot-product attention

The primitive is a differentiable, content-based lookup. Each position emits a *query* vector describing what it is looking for; each position offers a *key* (an address) and a *value* (a payload); a position retrieves a convex combination of values, weighted by how well its query matches each key.

Definition 4.1 (Single-head causal attention)**.** Let $Y \in \mathbb{R}^{T \times d_{\mathrm{model}}}$ be the (normalized) block input. With learned projections $W_Q, W_K \in \mathbb{R}^{d_{\mathrm{model}} \times d_{\mathrm{head}}}$, $W_V \in \mathbb{R}^{d_{\mathrm{model}} \times d_{\mathrm{head}}}$, form

$$Q = YW_Q, \qquad K = YW_K, \qquad V = YW_V \qquad (\in \mathbb{R}^{T \times d_{\mathrm{head}}}).$$

Define the *pre-softmax logits* (attention scores) $S = QK^\top / \sqrt{d_{\mathrm{head}}} \in \mathbb{R}^{T \times T}$, so $S_{ts} = \langle Q_t, K_s \rangle / \sqrt{d_{\mathrm{head}}}$. The output is

$$\mathrm{Attn}(Y) \;=\; AV, \qquad A \;=\; \mathrm{softmax}_{\mathrm{row}}\big(S + M\big) \in \mathbb{R}^{T \times T}, \tag{6}$$

where the softmax of Definition 1.1 is applied to each row, and M is the *causal mask*

$$M_{ts} = \begin{cases} 0 & s \leq t \\ -\infty & s > t. \end{cases}$$

Row t of A is a probability distribution over positions $1, \ldots, t$; the output at t is the corresponding average $\sum_{s \leq t} A_{ts} V_s$ of value vectors. Three structural points:

1. *Permutation equivariance.* Without the mask (and without RoPE, next section), (6) commutes with any permutation of the rows of Y: attention is a set operation, blind to order. Order must be injected deliberately; how to do so is the subject of §5.
2. *The $\sqrt{d_{\text{head}}}$.* If Q_t, K_s have i.i.d. zero-mean, unit-variance coordinates, then $\langle Q_t, K_s \rangle$ has mean 0 and variance d_{head} (a sum of d_{head} uncorrelated products). Dividing by $\sqrt{d_{\text{head}}}$ keeps logits $O(1)$ at initialization; without it, softmax saturates (rows collapse to near one-hot), its Jacobian $\operatorname{diag}(a) - a^\top a$ becomes tiny, and learning stalls. This is a one-line variance calculation with outsized practical importance—and its failure *later in training*, when Q, K are no longer near initialization, motivates QK-norm (§10.4).
3. *Softmax as smooth arg-max.* As logits sharpen, row t of A concentrates on $\arg\max_{s \le t} \langle Q_t, K_s \rangle$: attention interpolates between averaging and exact retrieval, differentiably.

4.2 The causal mask and why training is parallel

The mask enforces that the distribution computed at position t depends only on $x_{\le t}$, which is exactly what makes the factorization (1) valid and lets one forward pass score all T next-token predictions simultaneously (§1.2): compute (6) once as a dense $T \times T$ operation, and rows $1, \dots, T$ are, in effect, T models run on all T prefixes at once. This *teacher-forced parallelism* is the reason Transformers displaced recurrent networks: an RNN must process tokens serially even during training, while here training-time computation over the sequence is one batched matrix product—the shape hardware likes (§11).

At inference the situation inverts: generation is inherently serial (token $t{+}1$ requires sampling token t first), and efficiency instead comes from caching K, V across steps (§6).

4.3 Multi-head attention

One attention pattern per layer is too coarse: a position may simultaneously need to look at the previous token, a matching bracket 500 tokens back, and the subject of the current clause. *Multi-head attention* runs n_{h} independent copies of Definition 4.1 in parallel on projections into subspaces of dimension $d_{\text{head}} = d_{\text{model}}/n_{\text{h}}$, then concatenates and mixes:

$$\text{MHA}(Y) = \left[\, H^{(1)} \,\|\, H^{(2)} \,\|\, \cdots \,\|\, H^{(n_{\text{h}})} \,\right] W_O, \qquad H^{(i)} = \text{softmax}\!\left(\frac{(YW_Q^{(i)})(YW_K^{(i)})^\top}{\sqrt{d_{\text{head}}}} + M \right) YW_V^{(i)}, \tag{7}$$

with $W_O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$. The total parameter count, $4d_{\text{model}}^2$ per layer, is independent of n_{h}; the head count trades many low-rank attention patterns against few high-rank ones. Note each head's contribution to the output is $A^{(i)}\, Y\, W_V^{(i)} W_O^{(i)}$ where $W_O^{(i)}$ is the corresponding block of rows of W_O: the value–output pair acts as one rank-d_{head} map $W_V^{(i)} W_O^{(i)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ through which the head writes into the residual stream.

```python
import torch.nn.functional as F

def causal_attention(Y, W_q, W_k, W_v, W_o, n_heads):
    B, T, d = Y.shape
    d_h = d // n_heads
    # project, then split the feature axis into heads: [B, n_heads, T, d_h]
    Q = (Y @ W_q).view(B, T, n_heads, d_h).transpose(1, 2)
    K = (Y @ W_k).view(B, T, n_heads, d_h).transpose(1, 2)
    Vv = (Y @ W_v).view(B, T, n_heads, d_h).transpose(1, 2)

    S = Q @ K.transpose(-2, -1) / d_h**0.5          # [B, n_heads, T, T]
    mask = torch.triu(torch.ones(T, T, dtype=torch.bool), diagonal=1)
    S = S.masked_fill(mask, float('-inf'))           # causal: no future
    A = F.softmax(S, dim=-1)
    out = A @ Vv                                     # [B, n_heads, T, d_h]
    out = out.transpose(1, 2).reshape(B, T, d)       # concat heads
    return out @ W_o
```

4.4 Cost

Materializing S costs $O(T^2 d_{\text{head}})$ time and $O(T^2)$ memory per head: attention is *quadratic in sequence length*. For $T = 4096$ this is benign; for $T = 10^6$ it is not, and §9 and §11.2 describe the two responses: change the attention pattern (sliding windows, hybrids) and change the computation order (FlashAttention, which removes the $O(T^2)$ *memory* without approximation). By contrast the projections cost $O(T d_{\text{model}}^2)$; for typical training lengths the projections, not the T^2 term, dominate FLOPs. The quadratic term is a long-context and inference-memory problem more than a training-FLOPs problem.

5 Position: rotary embeddings (RoPE)

5.1 The design problem

Attention is permutation-equivariant (§4); token order must be injected. The 2017 solution added a position-dependent vector to the token embedding: sinusoidal features $\big(\sin(t/10000^{2i/d}), \cos(t/10000^{2i/d})\big)_i$, or in GPT-2/3 a learned vector per absolute position. Both encode *absolute* position and both are gone from frontier models, for two reasons. First, language statistics are largely translation-invariant: what matters to "which token should I attend to" is overwhelmingly the *relative offset* $t - s$ (and content), not the absolute indices. An architecture whose position-dependence is a function of $t-s$ builds this invariance in rather than asking the model to learn it. Second, learned absolute embeddings are undefined beyond the trained context length, and additive schemes entangle position with content in the residual stream for all downstream layers.

The modern solution, *rotary position embedding* (RoPE, Su et al. 2021), encodes position *multiplicatively, inside every attention layer, by rotating queries and keys*—the residual stream itself carries no position information.

5.2 Construction

Assume d_{head} even and write a query/key vector $q \in \mathbb{R}^{d_{\text{head}}}$ as $d_{\text{head}}/2$ coordinate pairs $(q^{(1)}, \dots, q^{(d_{\text{head}}/2)})$, $q^{(i)} \in \mathbb{R}^2$. Fix a base θ_{base} (2017-era 10^4; modern long-context models use 5×10^5 or larger, see §9) and per-pair frequencies

$$\omega_i \;=\; \theta_{\text{base}}^{-2(i-1)/d_{\text{head}}}, \qquad i = 1, \dots, d_{\text{head}}/2, \tag{8}$$

a geometric ladder from $\omega_1 = 1$ (fastest) down to $\omega_{d_{\text{head}}/2} \approx \theta_{\text{base}}^{-1}$ (slowest).

Definition 5.1 (RoPE)**.** For position $t \in \{1, 2, \dots\}$ define the block-diagonal rotation

$$R_t \;=\; \bigoplus_{i=1}^{d_{\text{head}}/2} \begin{pmatrix} \cos(\omega_i t) & -\sin(\omega_i t) \\ \sin(\omega_i t) & \cos(\omega_i t) \end{pmatrix} \;\in\; SO(d_{\text{head}}).$$

RoPE-attention replaces $Q_t \mapsto Q_t R_t^\top$ and $K_s \mapsto K_s R_s^\top$ in Definition 4.1 (rows times rotation), leaving V untouched, so the score becomes $S_{ts} = \langle Q_t R_t^\top, K_s R_s^\top \rangle / \sqrt{d_{\text{head}}}$.

Identifying each pair plane with $\mathbb{C}$, the map is simply $q^{(i)} \mapsto e^{\mathrm{i}\omega_i t} q^{(i)}$: multiply the i-th complex coordinate by a position-dependent phase. This is how it is implemented.

Theorem 5.2 (Relative-position property)**.** *For all $q, k \in \mathbb{R}^{d_{\text{head}}}$ and positions t, s,*

$$\langle q R_t^\top,\, k R_s^\top \rangle \;=\; q\, R_t^\top R_s\, k^\top \;=\; q\, R_{s-t}\, k^\top.$$

In particular the attention score between positions t and s depends on position only through the offset $s - t$.

Proof. R_t is orthogonal and the family $\{R_t\}$ is the image of the one-parameter group $t \mapsto (e^{\mathrm{i}\omega_1 t}, \dots, e^{\mathrm{i}\omega_{d/2} t})$ inside the torus of block rotations, hence a commuting family with $R_t^\top R_s = R_{-t} R_s = R_{s-t}$; substitute. □

In complex notation the score is $\sum_i \mathrm{Re}\big(q^{(i)} \overline{k^{(i)}}\, e^{\mathrm{i}\omega_i (t-s)}\big)$: each coordinate pair contributes its content correlation modulated by a sinusoid in the offset, at one frequency per pair. Fast pairs resolve fine-grained

order (adjacent tokens); slow pairs vary negligibly across thousands of tokens and act as content channels with mild long-range position sensitivity. A head can learn, e.g., "attend to offset -1" by aligning its query/key content with fast-frequency phases, or position-independent retrieval by concentrating its content in slow pairs.

Two further properties worth noting. *Norm preservation:* R_t is orthogonal, so RoPE changes no vector norms and interacts trivially with the variance reasoning of §4 and with QK-norm. *Locality of implementation:* position enters only inside each attention call; the residual stream, MLPs, and V/output paths are position-free, and no per-position parameters exist anywhere—any position t is defined, in principle, for unbounded t (what happens *statistically* for t beyond the training length is the extrapolation problem of §9).

```
def precompute_rope(T, d_h, base=500_000.0):
    freqs = base ** (-torch.arange(0, d_h, 2).float() / d_h)   # [d_h/2]
    angles = torch.outer(torch.arange(T).float(), freqs)       # [T, d_h/2]
    return torch.polar(torch.ones_like(angles), angles)        # e^{i w t}

def apply_rope(x, rope):            # x: [B, n_heads, T, d_h]
    xc = torch.view_as_complex(x.float().reshape(*x.shape[:-1], -1, 2))
    out = torch.view_as_real(xc * rope)          # phase multiply per pair
    return out.reshape_as(x).type_as(x)
# in attention:  Q = apply_rope(Q, rope);  K = apply_rope(K, rope)
```

Aside: relatives and descendants

RoPE won, but it is one member of a family of relative schemes. ALiBi (2022) adds a linear penalty $-m_h|t-s|$ to the scores—crude but extrapolation-friendly; T5 used learned bucketed relative biases. Some recent open models remove explicit position information entirely from some or all layers ("NoPE"), relying on the causal mask itself, which breaks permutation symmetry and provably allows position recovery; hybrid designs use RoPE on local-attention layers and NoPE on global ones (e.g. Llama 4). The common thread: absolute additive position embeddings are extinct at the frontier.

6 Inference economics: the KV cache, GQA, and latent attention

The most consequential attention redesigns of 2019–2025 were driven not by modeling considerations but by a single object that exists only at inference time: the key–value cache. This section derives it and follows the money.

6.1 The KV cache

Generation is serial: having produced $x_1, \ldots, x_t$, sample $x_{t+1} \sim \mathrm{softmax}(Z_t)$, append, repeat. Recomputing the full forward pass on the length-$(t{+}1)$ prefix at each step would cost $O(t)$ per token, $O(T^2)$ per sequence, in *trunk* FLOPs alone. The fix follows from causality: in every layer, the keys and values at positions $\le t$ depend only on $x_{\le t}$, which does not change as generation proceeds. So cache them. At step $t{+}1$, each layer computes Q, K, V *for the single new position*, appends (K_{t+1}, V_{t+1}) to the cache, and attends the one new query against all cached keys:

```
@torch.no_grad()
def generate(model, prompt_tokens, n_new, temp=1.0):
    cache = [LayerKVCache() for _ in model.blocks]   # per-layer K, V tensors
    x = prompt_tokens                                 # [1, T0]
    logits = model(x, kv_cache=cache)[:, -1]          # prefill: full prompt pass
    out = []
    for _ in range(n_new):
        probs = F.softmax(logits / temp, dim=-1)
        nxt = torch.multinomial(probs, 1)             # sample one token
        out.append(nxt)
        logits = model(nxt, kv_cache=cache)[:, -1]    # decode: ONE position;
    return torch.cat(out, dim=1)                      # K,V appended per layer
```

This splits inference into two regimes with different bottlenecks (§11.4): *prefill* (the prompt pass; parallel over positions, compute-bound) and *decode* (one token at a time; memory-bandwidth-bound, because each step must *read* the entire cache and all weights to produce one token).

The cache size is the problem. With standard multi-head attention (§4.3), per token per layer one stores $2n_{\text{h}}d_{\text{head}}$ numbers (K and V, all heads):

$$\text{cache bytes} \;=\; 2 \cdot L \cdot n_{\text{kv}} \cdot d_{\text{head}} \cdot T \cdot (\text{bytes/number}), \qquad n_{\text{kv}} = n_{\text{h}} \text{ for MHA.} \tag{9}$$

Concretely, a Llama-2-70B-shaped model with full MHA (L=80, n_{h}=64, d_{head}=128) at 16-bit precision stores $2 \cdot 80 \cdot 64 \cdot 128 \cdot 2 \approx 2.6\,\text{MB}$ *per token*: a 32k-token context costs ∼86 GB—more than an H100's entire memory—for *one* sequence, before weights. Since serving economics depend on batching many sequences per GPU (§11.4), shrinking (9) directly multiplies throughput. Hence:

6.2 MQA and GQA: share the keys and values

Multi-query attention (MQA, Shazeer 2019) keeps n_{h} distinct query projections but a *single* shared K and V projection: $n_{\text{kv}} = 1$, shrinking (9) by $n_{\text{h}}\times$ at a measurable quality cost. *Grouped-query attention* (GQA, 2023) interpolates: the n_{h} query heads are partitioned into n_{kv} groups ($1 < n_{\text{kv}} < n_{\text{h}}$, typically $n_{\text{kv}} = 8$), each group sharing one K/V head:

$$H^{(i)} = \text{softmax}\Big(\tfrac{Q^{(i)}(K^{(g(i))})^\top}{\sqrt{d_{\text{head}}}} + M\Big)V^{(g(i))}, \qquad g(i) = \lceil i n_{\text{kv}}/n_{\text{h}} \rceil.$$

With $n_{\text{h}} = 64, n_{\text{kv}} = 8$ the cache shrinks 8× at negligible quality loss; GQA-8 is as close to a universal frontier default as anything in this paper (Llama 3, Qwen 3, Mistral, GPT-OSS, ...). In code, the only change to the listing of §4.3 is projecting K, V to $n_{\text{kv}}d_{\text{head}}$ columns and broadcasting each K/V head across its group (`repeat_interleave` on the head axis).

6.3 Multi-head latent attention (MLA): compress, don't share

DeepSeek-V2/V3 (2024) pushed further with a linear-algebraic idea: rather than reducing the *number* of K/V heads, store a learned *low-rank compression* of all of them. Recall from §4.3 that what attention needs at position s is $K_s = Y_s W_K$ and $V_s = Y_s W_V$—linear images of the same d_{model}-vector. MLA inserts a bottleneck: with rank $r \ll n_{\text{h}}d_{\text{head}}$,

$$c_s \;=\; Y_s\, W_{DKV} \in \mathbb{R}^r \qquad \text{(cached)}, \qquad K_s^{(i)} = c_s\, W_{UK}^{(i)}, \quad V_s^{(i)} = c_s\, W_{UV}^{(i)}, \tag{10}$$

i.e. all heads' keys and values are decompressed on the fly from one shared latent c_s; only c_s (plus a small RoPE channel, below) is cached. DeepSeek-V3 uses $r = 512$ against $n_{\text{h}}d_{\text{head}} = 128 \cdot 128 = 16{,}384$: a ∼30× smaller cache than MHA, *while reporting quality above GQA*—plausibly because 128 full-resolution heads are retained, merely factored through a shared low-rank code, whereas GQA genuinely deletes distinct K/V subspaces. (Queries are also compressed through a separate rank-r_q bottleneck to save activation memory during training; that part does not affect the cache.)

Two refinements make (10) work in practice:

The absorption trick. Naively, decode-time attention must materialize $K^{(i)} = cW_{UK}^{(i)}$ for the whole cache each step. But scores are $Q^{(i)}(K^{(i)})^\top = (Y_t W_Q^{(i)} W_{UK}^{(i)\top})\, c^\top$: associativity lets W_{UK} be *absorbed* into the query projection (and W_{UV} into the output projection), so attention runs directly against the latent codes c as if they were r-dimensional keys and values. No decompression ever happens at decode time.

Decoupled RoPE. The absorption trick fails for rotary embeddings: RoPE inserts a position-dependent R_t *between* the matrices being absorbed ($W_Q R_t$ and $R_s^\top W_{UK}^\top$ cannot be merged into one static matrix since $R_t^\top R_s$ depends on both positions). MLA therefore splits each head's key into a large *content* part computed as in (10) with no RoPE, plus a small *positional* part (e.g. 64 dims/head, shared across heads) that carries RoPE and is cached separately; scores are the sum of the two inner products. This is a nice example of an architectural choice forced by the *commutative algebra of the inference optimization*, not by modeling.

Scheme	KV numbers/token/layer	Relative size	Used by
MHA (2017)	$2\,n_{\mathrm{h}} d_{\mathrm{head}}$	$1\times$	original Transformer, GPT-3
GQA-8	$2 \cdot 8 \cdot d_{\mathrm{head}}$	$n_{\mathrm{h}}/8\times$ smaller	Llama 3/4, Qwen 3, GPT-OSS
MQA	$2d_{\mathrm{head}}$	$n_{\mathrm{h}}\times$ smaller	PaLM, Falcon
MLA	$r + r_{\mathrm{rope}}$	$\sim$30$\times$ smaller	DeepSeek-V2/V3/R1, Kimi K2

Unconfirmed at the frontier: closed-lab attention

Closed frontier labs have not disclosed their attention variants. Given the serving economics above, it would be surprising if any closed frontier model still used full MHA; GQA-like sharing, MLA-like compression, and hybrid sparse/linear-attention layers (§9) are all plausible. Treat any specific claim about GPT-5-class or Claude internals as unverifiable.

7 The MLP block: from ReLU to SwiGLU

7.1 The classical multi-layer perceptron

A *multi-layer perceptron* (MLP) is an alternating composition of affine maps and a fixed coordinatewise nonlinearity $\phi\colon \mathbb{R} \to \mathbb{R}$:

$$\mathrm{MLP}(x) \;=\; \phi(xW_1 + b_1)\,W_2 + b_2, \qquad W_1 \in \mathbb{R}^{d_{\mathrm{model}} \times d_{\mathrm{ff}}},\; W_2 \in \mathbb{R}^{d_{\mathrm{ff}} \times d_{\mathrm{model}}},$$

applied to each position's vector independently (ϕ acts entrywise). The classical universal approximation theorem (Cybenko/Hornik, stated here without proof) says such two-layer maps are dense in $C(K)$ for compact $K \subset \mathbb{R}^d$ under mild conditions on ϕ; the practical content for us is simply that the MLP is the block where arbitrary per-position nonlinear computation lives, complementing attention, which is linear in V once the attention pattern is fixed. In the 2017 Transformer, $\phi = \mathrm{ReLU}(u) = \max(u, 0)$ and $d_{\mathrm{ff}} = 4d_{\mathrm{model}}$; the MLP accounts for roughly two-thirds of a dense model's parameters (§12), so its design matters.

Two smooth relatives of ReLU appear below:

$$\mathrm{SiLU}(u) = u\,\sigma(u) = \frac{u}{1 + e^{-u}}, \qquad \mathrm{GELU}(u) = u\,\Phi(u),$$

with σ the logistic function and Φ the standard normal CDF. Both are smooth, non-monotone near the origin, and asymptotic to ReLU; smoothness helps optimization at essentially no cost.

7.2 Gated linear units and SwiGLU

The frontier MLP (Shazeer 2020; adopted by PaLM, Llama, Mistral, DeepSeek, Qwen, ...) replaces the single hidden pre-activation with an *elementwise product of two projections*, one nonlinear ("gate") and one linear ("value"):

Definition 7.1 (SwiGLU block)**.** With $W_g, W_u \in \mathbb{R}^{d_{\mathrm{model}} \times d_{\mathrm{ff}}}$, $W_d \in \mathbb{R}^{d_{\mathrm{ff}} \times d_{\mathrm{model}}}$, no biases,

$$\mathrm{SwiGLU}(x) \;=\; \big(\mathrm{SiLU}(xW_g) \odot (xW_u)\big)\,W_d,$$

$\odot$ the Hadamard (entrywise) product.

Interpretation: coordinate j of the hidden layer is $\mathrm{SiLU}(\langle x, w_g^{(j)} \rangle) \cdot \langle x, w_u^{(j)} \rangle$—a learned scalar *gate* deciding how much of a learned linear *feature* passes. The block is no longer of the classical ϕ(affine) form: it is a bilinear-with-gating map, and the product introduces multiplicative interactions between projections of x that a single-activation MLP can only imitate with more width. Empirically SwiGLU gives a small but robust loss improvement at matched parameter count—one of several "no single big win, many compounding percents" changes separating 2017 from 2026.

The $\frac{2}{3}$ convention. SwiGLU has three matrices where the classical MLP has two. To compare at equal parameters, set $3d_{\mathrm{model}}d_{\mathrm{ff}}^{\mathrm{new}} = 2d_{\mathrm{model}}d_{\mathrm{ff}}^{\mathrm{old}}$, i.e. $d_{\mathrm{ff}}^{\mathrm{new}} = \frac{2}{3} \cdot 4d_{\mathrm{model}} = \frac{8}{3}d_{\mathrm{model}}$; hence the otherwise mysterious hidden sizes in published configs (e.g. Llama-3-70B: $d_{\mathrm{model}} = 8192$, $d_{\mathrm{ff}} = 28{,}672 = 3.5d_{\mathrm{model}}$, a rounded-up variant chosen for hardware-friendly divisibility).

```python
class SwiGLU(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.gate = nn.Linear(cfg.d_model, cfg.d_ff, bias=False)
        self.up   = nn.Linear(cfg.d_model, cfg.d_ff, bias=False)
        self.down = nn.Linear(cfg.d_ff, cfg.d_model, bias=False)
    def forward(self, x):
        return self.down(F.silu(self.gate(x)) * self.up(x))
```

Aside: what MLPs do, mechanistically

Interpretability work supports viewing the MLP as a key–value memory: rows of the input projections act as detectors of features present in the residual stream; the corresponding rows of W_d are what gets written back when a detector fires. Facts of the form "Dublin → Ireland" localize substantially in mid-layer MLPs. Again, none of the mathematics above depends on this reading, but it explains why the parameter-heavy MLP blocks are where "knowledge capacity" is usually said to live—and why they are the natural target for the sparsification of the next section.

8 Mixture-of-experts: decoupling parameters from FLOPs

8.1 The idea

In a dense model every parameter participates in every token's forward pass, so parameter count and per-token compute are locked together. Scaling laws (§10.5) say more parameters help; the electricity bill says FLOPs hurt. *Sparse mixture-of-experts* (MoE) breaks the lock: replace the single MLP of a block with E parallel MLPs (*experts*) plus a learned *router* that sends each token to only $k \ll E$ of them. The model then has the *capacity* of E experts but the per-token *cost* of k. One speaks of *total* versus *active* parameters: DeepSeek-V3 has 671B total, 37B active ($E = 256$ routed experts per MoE layer, $k = 8$, plus one always-on shared expert); Mixtral 8×7B popularized the recipe openly ($E = 8$, $k = 2$); GPT-OSS-120B has 117B total, 5.1B active. Nearly every open frontier-scale model since 2024 is MoE, and it is the one architectural fact about several closed frontier models that has been widely (though unofficially) reported.

8.2 The routed layer, precisely

Definition 8.1 (MoE layer with top-k routing)**.** Let $\{f_e\}_{e=1}^{E}$ be SwiGLU blocks (Definition 7.1, each with its own parameters and expert width d_{exp}, typically much smaller than a dense d_{ff}), and let $W_r \in \mathbb{R}^{d_{\text{model}} \times E}$ be the router. For a token representation $x \in \mathbb{R}^{d_{\text{model}}}$:

$$
\begin{aligned}
s &= \text{scores}(xW_r) \in \mathbb{R}^E && \text{(softmax or elementwise sigmoid over experts)} \\
\mathcal{T} &= \text{top-}k(s) \subset \{1, \dots, E\}, \quad |\mathcal{T}| = k && \text{(indices of the } k \text{ largest scores)} \\
g_e &= \frac{s_e}{\sum_{e' \in \mathcal{T}} s_{e'}} \quad \text{for } e \in \mathcal{T}, \quad 0 \text{ otherwise} && \text{(renormalized gates)} \\
\text{MoE}(x) &= \sum_{e \in \mathcal{T}} g_e \, f_e(x) \; + \; f_{\text{shared}}(x).
\end{aligned}
$$

The map $x \mapsto \mathcal{T}(x)$ is piecewise constant, hence the layer is only piecewise smooth in x; but for fixed $\mathcal{T}$ it is smooth in all *parameters*, and gradients flow to the selected experts and, through the gates g_e, to the router. The non-selected experts receive no gradient from this token—selection itself is not differentiated through, a discrete-continuous mismatch the field tolerates because it works. Routing is per *token* (each position independently), not per sequence; a single sentence will scatter its tokens across many experts.

Modern refinements, all present in DeepSeek-V3 and widely copied: *fine-grained experts* (many small experts, $E \sim$ 128–512 with $k \sim 8$, rather than 8 big ones—finer partition of the score simplex, more combinations, $\binom{256}{8} \approx 10^{14}$ selections); a *shared expert* that every token uses, absorbing common computation so routed experts can specialize; and sigmoid rather than softmax scoring, which decouples experts' scores from one another.

8.3 Load balancing

Left alone, top-k routing collapses: an expert that is slightly better early receives more gradient, improves, and attracts more traffic—a rich-get-richer dynamic whose fixed point is a few overloaded experts and many dead ones. This is fatal for a second reason that is systems, not statistics: experts are *sharded across devices* (§11.3), and the slowest (most-loaded) device gates the step time, so balanced expert load is literally a throughput constraint.

The classical fix is an auxiliary loss. For a batch of N tokens let

$$f_e = \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}\{e \in \mathcal{T}(x_n)\} \cdot \frac{E}{k} \quad \text{(normalized fraction routed to } e\text{)}, \qquad \bar{s}_e = \frac{1}{N}\sum_{n=1}^{N} \mathrm{softmax}(x_n W_r)_e,$$

and add $\mathcal{L}_{\text{bal}} = \alpha \sum_{e=1}^{E} f_e\, \bar{s}_e$ to (2) with small α. Since $\sum_e f_e = E/k \cdot k/E \cdot E$-normalized and $\sum_e \bar{s}_e = 1$ are fixed, the product sum is minimized when load is uniform (f_e constant); f_e is not differentiable but $\bar{s}_e$ is, so the router feels a push away from crowded experts. The cost is a tension: α too large distorts routing away from the loss-minimizing assignment.

DeepSeek-V3 introduced an *auxiliary-loss-free* alternative that has spread quickly: add a per-expert bias b_e *to the scores used for top-k selection only* (not to the gates g_e), and update it by a simple controller outside gradient descent—after each batch, decrement b_e for overloaded experts and increment for underloaded ones by a fixed step γ. Selection pressure equalizes load; gate values, and hence the model's output, are never directly distorted by a balancing term. (A tiny sequence-level auxiliary loss is retained as a safety net.)

```python
class MoE(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.router  = nn.Linear(cfg.d_model, cfg.n_experts, bias=False)
        self.bias    = torch.zeros(cfg.n_experts)      # balance controller, not a Parameter
        self.experts = nn.ModuleList(SwiGLU(cfg.expert_cfg) for _ in range(cfg.n_experts))
        self.shared  = SwiGLU(cfg.expert_cfg)
        self.k = cfg.top_k

    def forward(self, x):                              # x: [N_tokens, d_model]
        s = torch.sigmoid(self.router(x))              # [N, E] affinity scores
        top = torch.topk(s + self.bias, self.k, dim=-1).indices   # biased SELECTION
        gates = torch.gather(s, -1, top)               # UNbiased gate values
        gates = gates / gates.sum(-1, keepdim=True)
        out = self.shared(x)
        for j in range(self.k):                        # (real impls: grouped batched matmuls)
            e = top[:, j]
            for eid in e.unique():
                m = (e == eid)
                out[m] += gates[m, j:j+1] * self.experts[eid](x[m])
        return out
```

8.4 Why it works, and what it costs

The bet underlying MoE is that the function being learned is *conditionally sparse*: the computation useful for a Python token, a Sanskrit token, and an arithmetic token overlap only partially, so forcing all of it through one dense MLP wastes FLOPs. Empirically the bet pays: at matched *training compute*, MoE models achieve materially lower loss than dense ones, effectively shifting the compute-performance frontier. The costs are real but operational: total parameters must still live in memory (hence expert parallelism, §11.3); all-to-all communication is added to every MoE layer (tokens physically travel to their experts' devices and back); training has extra failure modes (routing collapse, dead experts); and fine-tuning sparse models is more delicate. Frontier labs have decided this trade is worth it at the top end, while dense models remain common at small and mid scale where simplicity wins.

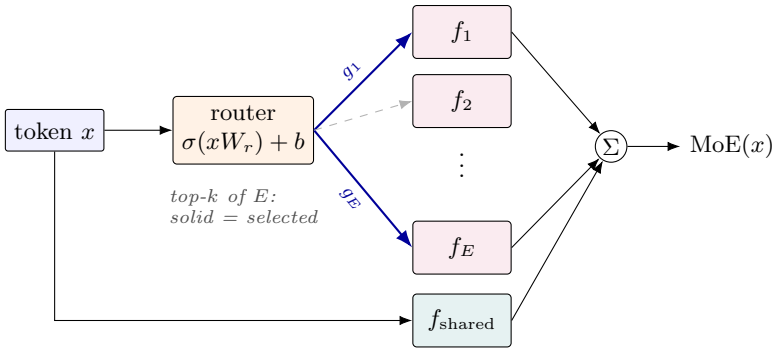

Figure 2: A mixture-of-experts layer: the router selects k of E experts per token (solid arrows) and mixes their outputs with gates g_e; a shared expert processes every token. Unselected experts (dashed) cost nothing for this token.

Aside: experts are not specialists you can name

Despite the name, experts do not cleanly correspond to human-legible domains; observed specialization is mostly at the level of token patterns and shallow syntax, with the shared expert soaking up general computation. The architecture is best thought of not as a committee of domain experts but as a learned block-sparse factorization of one very wide MLP.

9 Long context: attention patterns and position extension

Frontier models advertise contexts of 10^5–10^7 tokens, yet are pretrained mostly at 4k–8k. Bridging that gap requires attacking both faces of the quadratic problem of §4.4—the cost of the attention pattern and the statistics of position extrapolation—plus the exact-computation techniques of §11.2.

9.1 Sliding-window and hybrid attention patterns

Sliding-window attention (SWA) restricts position t to attend to $s \in [t - w + 1, t]$ for a window w (e.g. $w = 4096$ in Mistral, 1024 in Gemma 3, 128 in GPT-OSS's local layers): replace the causal mask by

$$M_{ts} = 0 \text{ if } t - w < s \le t, \qquad -\infty \text{ otherwise,}$$

making cost and cache $O(Tw)$ instead of $O(T^2)$. Information can still propagate beyond w *across layers*: position t at layer ℓ sees $s > t - w$ at layer $\ell - 1$, which saw $s' > s - w$, so the receptive field grows to ℓw after ℓ layers—the same telescoping argument as for receptive fields of stacked convolutions.

Pure SWA degrades exact long-range retrieval, so frontier models *interleave*: most layers local, a minority global. Gemma 3 uses a 5:1 local:global ratio; GPT-OSS alternates 1:1 with $w = 128$; Llama 4 interleaves RoPE-local and NoPE-global layers. The KV cache then has two regimes: local layers cache only w tokens (a sliding buffer), global layers cache everything—so the blended cache cost is dominated by the few global layers, and the local:global ratio is chosen jointly with n_{kv} against the memory budget of (9).

Aside: attention sinks

Softmax rows must sum to 1: a head has no way to say "attend to nothing." Trained models resolve this by parking excess attention mass on the first token(s), which act as a no-op sink; evicting those tokens from a sliding cache catastrophically degrades generation. Fixes: always retain the first few positions (StreamingLLM), or—as in GPT-OSS—add a learned per-head scalar to the softmax denominator, an explicit "null" option: $A_{ts} = e^{S_{ts}} / (e^{\beta_h} + \sum_{s'} e^{S_{ts'}})$.

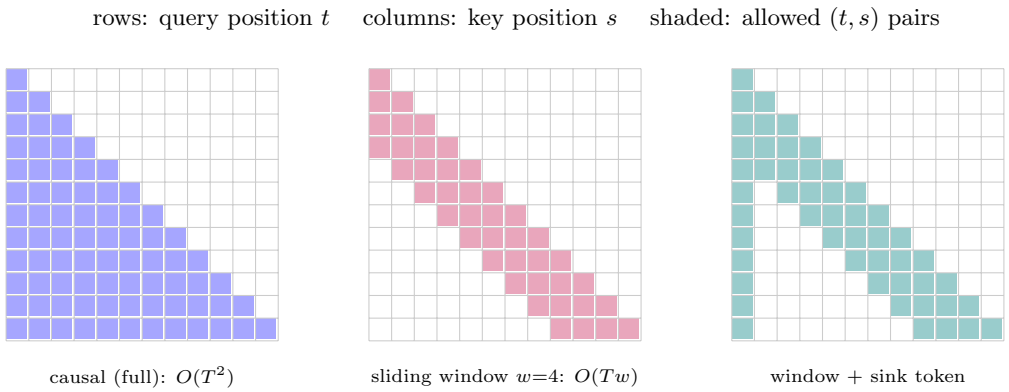

Figure 3: Attention masks. Modern models interleave full-causal layers (left) with sliding-window layers (center); some add always-visible "sink" positions (right).

9.2 Extending RoPE beyond the training length

Even with affordable attention, positions t beyond the pretraining length T_{train} present a *statistical* problem: the score contribution of pair i at offset Δ is a sinusoid in $\omega_i\Delta$, and offsets larger than any seen in training put the low-frequency pairs into phase regions the model never observed—empirically, perplexity explodes. Remedies operate on the frequency ladder (8):

- *Larger base.* Raising θ_{base} ($10^4 \to 5 \cdot 10^5$ in Llama 3, 10^6 in Gemma's global layers) slows all frequencies, so a given Δ-range covers less phase; the standard modern default, set before pretraining.
- *Position interpolation (PI).* To reach length λT_{train}, use positions t/λ: never extrapolate, only interpolate phases within the seen range. Costs resolution at short range (fast pairs are compressed too, blurring adjacent-token distinctions).
- *NTK-aware scaling / YaRN.* Interpolate *non-uniformly*: leave fast pairs (many periods fit inside T_{train}; extrapolation is genuinely periodic and safe) untouched, compress slow pairs (which would otherwise extrapolate out of seen phase), with a smooth ramp between, plus a mild rescaling of attention logits. YaRN with a short long-context fine-tune is the published recipe behind most "128k" models, including GPT-OSS.

The general lesson: the geometric frequency ladder makes "position" a multiscale quantity, and context extension is a per-scale decision about interpolation versus extrapolation.

Unconfirmed at the frontier: million-token contexts

How closed labs reach 10^6–10^7 tokens is not published. The public toolkit—hybrid local/global layers, RoPE rescaling, long-document continued pretraining, context parallelism (§11.3)—is believed to be most of the story; whether frontier closed models additionally use retrieval-like cache compression, chunked/linear attention variants (below), or memory layers is unconfirmed.

Aside: the linear-attention / state-space wing

A research line (linear attention, Mamba-style state-space models, gated linear RNNs) replaces softmax attention with recurrences whose state is *constant-size* in T: schematically, maintain $S_t = \gamma_t S_{t-1} + k_t^\top v_t \in \mathbb{R}^{d_{\text{head}} \times d_{\text{head}}}$ and read out $o_t = q_t S_t$—an outer-product accumulator playing the role of the KV cache, with $O(T)$ total cost. Pure versions lose exact retrieval (the state is a lossy compression), but *hybrids*—a few softmax layers interleaved with many linear ones—are competitive and have shipped in serious open models (Jamba, MiniMax-Text-01, Qwen3-Next, Kimi Linear; Google's published work on Griffin/RecurrentGemma is adjacent). Whether any closed frontier flagship is a hybrid is, as of early 2026, unconfirmed but widely suspected; economically, decode cost is where the pressure points.

10 Training: the full objective, initialization, optimization, and stability

10.1 The complete pretraining loss

Collecting the pieces, a modern MoE frontier model minimizes

$$\mathcal{L}(\theta) \;=\; \underbrace{\mathbb{E}\Big[\tfrac{1}{T}\textstyle\sum_t -\log p_\theta(x_t \mid x_{<t})\Big]}_{\text{cross-entropy (2)}} \;+\; \underbrace{\alpha\,\mathcal{L}_{\text{bal}}}_{\S 8.3} \;+\; \underbrace{\beta\,\mathbb{E}\Big[\tfrac{1}{T}\textstyle\sum_t \big(\log \sum_j e^{(z_t)_j}\big)^2\Big]}_{z\text{-loss (below)}} \;(+\,\lambda\,\mathcal{L}_{\text{MTP}}), \tag{11}$$

with α, β small ($\beta \sim 10^{-4}$) and the optional multi-token term explained below. That is the entire objective; there is no other signal in pretraining.

z-loss. Softmax is shift-invariant, so the log-partition $Z(z_t) = \log \sum_j e^{(z_t)_j}$ is unconstrained by the cross-entropy: logits can drift to large magnitudes with no loss penalty, until bf16 arithmetic (§11.1) starts producing infinities. The z-loss (PaLM 2022) penalizes Z^2, softly pinning the free gauge direction near $Z \approx 0$. A pure numerical-stability regularizer, motivated entirely by finite-precision concerns.

Multi-token prediction (MTP). DeepSeek-V3 attaches a small auxiliary head that, at each position, predicts token $t{+}2$ as well (sequentially conditioned through a one-block module), adding a densified planning signal; the head is discarded or reused for speculative decoding (§11.5) at inference. A modest but real quality gain in published ablations; adoption beyond DeepSeek-lineage models is uneven.

10.2 Initialization and the variance calculus

The recurring analytical tool of this paper—compute the variance of a linear functional of roughly independent coordinates and demand it stay $O(1)$—also fixes initializations. For $y = xW$ with W_{ij} i.i.d. $\mathcal{N}(0, \sigma^2)$ and $\mathbb{E}\|x\|^2 = d_{\text{in}}$: $\text{Var}(y_j) = \sigma^2 d_{\text{in}}$, so $\sigma = \Theta(d_{\text{in}}^{-1/2})$ keeps activation scale depth-independent (LeCun/He/Xavier initialization, all variants of this computation). Two Transformer-specific refinements: the projections that *write into the residual stream* (W_O, W_d) are further shrunk by $\Theta(1/\sqrt{2L})$ so that the sum (5) of $2L$ block outputs itself has $O(1)$ variance at initialization; and embeddings are initialized small with the unembedding at $\Theta(d_{\text{model}}^{-1/2})$.

Aside: muP

Maximal-update parametrization (μP) refines this calculus into a prescription for how initialization scales *and learning rates* should vary with width so that training dynamics—not just the forward pass at $t{=}0$—are width-invariant. Its practical payoff is *hyperparameter transfer*: tune the learning rate on a small proxy model, multiply by the prescribed powers of width, and use it at target scale. Published evidence and open-model reports suggest variants of this are standard practice at large labs; exact frontier recipes are unpublished.

10.3 Optimization

Training is stochastic first-order optimization on (11) over batches of 10^6–10^7 tokens. The workhorse is *AdamW*. Let $g_t = \nabla_\theta \mathcal{L}$ on the current batch; maintain per-parameter exponential moving averages

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^{\odot 2},$$

with bias corrections $\hat{m}_t = m_t/(1-\beta_1^t)$, $\hat{v}_t = v_t/(1-\beta_2^t)$, and update

$$\theta_{t+1} = \theta_t - \eta_t \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda\, \theta_t \right),$$

all operations coordinatewise; typically $\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay $\lambda \approx 0.1$ applied *decoupled* (the "W"), i.e. not through the moment estimates. The $\hat{m}/\sqrt{\hat{v}}$ ratio makes the step size per coordinate approximately invariant to the gradient's scale—a crude diagonal preconditioner—which matters because gradient

magnitudes vary across a Transformer's parameter groups by orders of magnitude. Gradients are globally *clipped* ($g \mapsto g \cdot \min(1, c/\|g\|_2)$, $c = 1$) to bound the damage of rare loss spikes. The learning rate η_t follows warmup (linear, 10^2–10^4 steps) then decay (cosine to ∼10% of peak, or trapezoidal warmup–stable–decay schedules that allow flexible run extension).

Aside: beyond Adam: Muon

A 2024–25 development worth flagging: *Muon* treats each weight *matrix* (not each scalar) as the optimization unit, orthogonalizing the momentum matrix via Newton–Schulz iteration—approximately replacing the gradient by the nearest (in spectral geometry) semi-orthogonal matrix $UV^\top$ from its SVD $U\Sigma V^\top$—before stepping. Kimi K2 (1T-parameter MoE, open) trained with a Muon variant at frontier scale, reporting better loss per FLOP than AdamW; adoption is spreading in open models. Whether closed frontier labs have moved beyond AdamW is unpublished.

10.4 Stability at scale

At 10^{25}–10^{26} FLOPs, rare instabilities dominate engineering attention: loss spikes, logit blowups, precision overflow. The architectural countermeasures, all now common:

- *QK-norm.* Apply RMSNorm to Q and K per head before RoPE and the dot product. The $1/\sqrt{d_{\text{head}}}$ scaling of §4 controls logits only near initialization; late in training, aligned Q, K directions produce logits in the hundreds ("attention entropy collapse", saturated one-hot rows, spikes). QK-norm bounds $|\langle Q_t, K_s\rangle| \le \|\gamma_Q\|\|\gamma_K\|$-type quantities by construction. Used by Gemma 3, Qwen 3, OLMo 2, ViT-22B lineage.
- *No bias terms.* All linear layers bias-free: quality-neutral at scale, removes parameters/kernels, and empirically improves stability.
- *Logit soft-capping* (Gemma 2): $z \mapsto c\tanh(z/c)$ on attention or final logits, a smooth clamp; partially displaced by QK-norm.
- *z-loss*, above; *bf16 with fp32 accumulations* (§11.1); *spike recovery* by checkpoint rewind and data skip.

10.5 Scaling laws: why anyone believes small-scale experiments

The empirical backbone of all frontier decision-making: across many orders of magnitude, held-out loss follows smooth power laws in model size N and data D,

$$\mathcal{L}(N, D) \;\approx\; \mathcal{L}_\infty + \frac{A}{N^\alpha} + \frac{B}{D^\beta}, \qquad \alpha, \beta \approx 0.3\text{–}0.5,$$

(Kaplan 2020; Hoffmann "Chinchilla" 2022). Given a compute budget $C \approx 6ND$ FLOPs (derived in §12), minimizing $\mathcal{L}$ subject to $6ND = C$ is a one-line Lagrange computation whose Chinchilla-fitted solution is $D^* \approx 20N^*$ tokens—the "compute-optimal" ratio. Frontier practice deliberately *overtrains* relative to this (D/N of 10^2–10^3; Llama-3-8B saw ∼15T tokens, $D/N \approx 1900$), because (2) is minimized per unit of *training* compute at Chinchilla, but products amortize *inference* compute over billions of queries, and a smaller overtrained model is cheaper to serve at equal quality. Architecture choices (SwiGLU vs ReLU, MoE vs dense, GQA vs MHA) are made by fitting these laws at small scale and trusting the extrapolation—the methodological answer to "how can anyone iterate on designs that cost $\$10^8$ to evaluate."

11 Systems: the hardware-shaped mathematics

Frontier architecture cannot be understood as pure function approximation; several components of the preceding sections (GQA/MLA, MoE balance, sliding windows, no-bias, RMSNorm-over-LayerNorm) exist because of hardware constraints. This section makes those constraints quantitative.

The controlling concept is *arithmetic intensity*: FLOPs performed per byte moved between GPU main memory (HBM) and compute units. An H100-class accelerator sustains $\sim 10^{15}$ bf16 FLOP/s but only

$\sim 3.4 \times 10^{12}$ bytes/s of HBM bandwidth: an operation must perform ∼300 FLOPs per byte touched to be compute-bound. Big matrix multiplications qualify; almost nothing else does. Reading the roofline: the large-matmul-heavy design of the Transformer is not incidental—architectures survive partly by being *expressible as large matmuls*.

11.1 Numerical precision

Training runs in mixed precision: parameters and gradients in `bfloat16` (8-bit exponent as in fp32, 7-bit mantissa—fp32's range with ∼3 decimal digits), with fp32 *master weights* in the optimizer and fp32 accumulation inside matmuls and reductions (summing 10^4 bf16 products in bf16 loses digits; accumulate wide, store narrow). The frontier edge is `FP8` ($E4M3$: 4 exponent, 3 mantissa bits) for matmul operands: DeepSeek-V3 demonstrated full pretraining with FP8 GEMMs using fine-grained (per-tile) scaling factors and higher-precision accumulation, roughly doubling matmul throughput on FP8-capable hardware. Precision is also why the stability devices of §10.4 exist: bf16's coarse mantissa is unforgiving of drifting logit scales. Inference weights are further *quantized* (8- or 4-bit integers/floats per channel-scaled block), trading a controlled accuracy loss for memory and bandwidth; GPT-OSS ships MoE weights in a 4-bit format natively.

11.2 FlashAttention: exact attention without the $T \times T$ matrix

Naive attention (§4) writes $S, A \in \mathbb{R}^{T\times T}$ to HBM: $O(T^2)$ memory traffic for $O(T^2 d_{\text{head}})$ FLOPs—low arithmetic intensity, memory-bound, and memory-infeasible for large T. FlashAttention (Dao 2022) computes the *same function exactly* without ever materializing S or A, by tiling Q, K, V into blocks that fit in on-chip SRAM and exploiting an algebraic fact: softmax-weighted sums can be computed in a streaming fashion.

Proposition 11.1 (Online softmax)**.** *Fix a query row q and process key/value blocks sequentially. Maintain (m, Z, o): running max of logits, running normalizer, running* unnormalized *output. On a new block with logits $s^{(b)}$ and values $V^{(b)}$, update with $m' = \max(m, \max_j s_j^{(b)})$:*

$$Z \leftarrow Z e^{m-m'} + \sum\nolimits_j e^{s_j^{(b)}-m'}, \qquad o \leftarrow o\, e^{m-m'} + \sum\nolimits_j e^{s_j^{(b)}-m'} V_j^{(b)}, \qquad m \leftarrow m'.$$

After all blocks, $o/Z = \sum_j \mathrm{softmax}(s)_j V_j$ exactly.

Proof. Invariant: after each block, $o = \sum_{j\le\text{seen}} e^{s_j-m} V_j$ and $Z = \sum_{j\le\text{seen}} e^{s_j-m}$ with m the max over seen logits. The rescaling by $e^{m-m'}$ re-references both sums to the new max (softmax's shift invariance, Definition 1.1, is what makes the running re-referencing harmless); the ratio o/Z telescopes to the full softmax average. □

Each K/V block is thus read once, combined against a resident Q tile, and discarded; HBM traffic drops from $O(T^2)$ to $O(T^2 d_{\text{head}}/\mathcal{M})$ for SRAM size $\mathcal{M}$ (and $O(T)$ extra memory), turning attention compute-bound in practice with 2–4× wall-clock gains and enabling the long-context training of §9. The backward pass recomputes S tile-wise from stored (Q, K, V, m, Z) rather than storing A—an instance of the general *activation checkpointing* trade (recompute-instead-of-store) used throughout training. The lesson generalizes and partly explains the field's kernel-centricity: the function is fixed; reorganizing the *order of summation* around the memory hierarchy is where large constant factors live.

11.3 Parallelism: fitting 10^{12} parameters on 10^4 chips

A 500B-parameter model in bf16 with Adam state needs ∼ 16 bytes/parameter = 8 TB of training state against ∼ 80 GB per GPU: the model *must* be sharded, along several composable axes.

- *Data parallelism (DP).* Replicate the model, split the batch, all-reduce gradients each step. ZeRO/FSDP refines this by sharding parameters, gradients, and optimizer state across the DP group, gathering shards just-in-time—removing DP's memory redundancy at the price of more communication.

- *Tensor parallelism (TP).* Split individual matrices across k devices. For the MLP: shard W_1 by columns and W_2 by rows, so $xW_1 = [xW_1^{(1)} \,\|\, \cdots \,\|\, xW_1^{(k)}]$ needs no communication, each device applies the nonlinearity to its slice, and $\phi(\cdot)W_2 = \sum_i \phi(\cdot)^{(i)} W_2^{(i)}$ requires one all-reduce. (Note the pairing: column-split feeding row-split costs one collective per MLP, not two—the placement is chosen from the block structure of the matrix product.) Attention shards naturally by heads. TP is communication-hungry (activations every layer) and is confined to fast-interconnect domains ($k \leq 8$, one node).
- *Pipeline parallelism (PP).* Assign contiguous layer ranges to device groups; stream micro-batches through the pipeline to keep stages busy. The idle "bubble" fraction scales like (stages $-$ 1)/micro-batches; interleaved and zero-bubble schedules shrink it.
- *Expert parallelism (EP).* MoE experts sharded across devices; each MoE layer performs an all-to-all sending tokens to their experts and back. This is why load balance (§8.3) is a systems constraint, and why DeepSeek-V3 additionally restricted each token's experts to at most 4 nodes (routing locality as an architectural term).
- *Context/sequence parallelism (CP).* Shard the *sequence* axis for very long contexts; ring-attention arrangements pass K/V blocks around devices, composing with the online softmax of Proposition 11.1, whose streaming structure is exactly what makes K/V-block passing correct.

A frontier run composes DP $\times$ TP $\times$ PP $\times$ EP ($\times$ CP) over 10^4–10^5 accelerators; the composition (which axis gets which network tier) is itself a small optimization problem solved per cluster. Determinism, checkpointing cadence, straggler and failure recovery (a 10^4-GPU run sees hardware failures daily) are the operational reality surrounding (11).

11.4 Inference serving

Decode-phase arithmetic intensity is brutal: generating one token for one sequence reads every active parameter and the sequence's whole KV cache to perform one small matvec per matrix—a few FLOPs per byte, two orders below the compute-bound threshold. Every serving technique is an attack on this:

- *Batching.* Weights read once serve B sequences' matvecs (a matvec batched over B is a matmul): intensity scales with B until KV cache exhausts memory—which is why (9), hence GQA/MLA/SWA, directly sets throughput. *Continuous batching* admits/retires sequences per token rather than per batch.
- *Paged KV cache* (vLLM). Allocate cache in fixed-size blocks with an indirection table—virtual memory for KV—eliminating fragmentation from variable-length sequences and enabling prefix sharing across requests (system prompts cached once).
- *Prefill/decode disaggregation.* Prefill is compute-bound, decode bandwidth-bound; large deployments run them on separate device pools and ship the KV cache between, so each pool runs at its own optimal batch shape.
- *Speculative decoding* (§11.5): spend cheap FLOPs to turn serial decode steps into parallel verification.

11.5 Speculative decoding, with its correctness proof

A small *draft* model q (or an MTP head, §10) proposes γ tokens $\hat{x}_{1:\gamma}$ autoregressively; the large model p scores all γ positions *in one parallel forward pass* (teacher-forced parallelism, §4.2, now exploited at inference). Proposals are accepted left to right:

Proposition 11.2 (Modified rejection sampling; Leviathan et al. 2023)**.** *Accept $\hat{x}_i$ with probability $\min\big(1,\, p(\hat{x}_i)/q(\hat{x}_i)\big)$ (conditioning on the accepted prefix suppressed). On the first rejection, sample the replacement token from the residual distribution $r(x) \propto \max\big(0,\, p(x) - q(x)\big)$ and stop. Then every emitted token is distributed exactly according to p.*

Proof. For a single step, the probability of emitting x is $q(x)\min(1, p(x)/q(x)) + \big(1 - \sum_y q(y)\min(1, \frac{p(y)}{q(y)})\big) r(x)$. The first term is $\min(q(x), p(x))$; the rejection mass is $1 - \sum_y \min(q(y), p(y)) = \sum_y (p(y) - q(y))^+$, which

is exactly the normalizer of r; so the total is $\min(q(x), p(x)) + (p(x) - q(x))^+ = p(x)$. Induct along the sequence. □

Expected accepted length grows with the overlap $\sum_x \min(p, q)$; in practice 2–4× decode speedup with *zero* change to the output distribution—a rare free lunch, purchased with idle compute capacity in the bandwidth-bound decode regime. Modern variants (EAGLE, Medusa, MTP-based self-speculation) draft with heads attached to the large model itself.

12 A worked example: auditing a frontier config

We close the technical development by counting everything for one real dense model and one real sparse model, so the reader can verify that the published numbers are forced by the architecture. All counts ignore normalization gains ($O(Ld_{\text{model}})$, negligible).

12.1 Parameter counting

Per pre-norm block (Sections 3–7), with GQA:

$$\underbrace{d_{\text{model}} d_{\text{head}} n_{\text{h}}}_{W_Q} + \underbrace{2\, d_{\text{model}} d_{\text{head}} n_{\text{kv}}}_{W_K, W_V} + \underbrace{d_{\text{head}} n_{\text{h}} d_{\text{model}}}_{W_O} + \underbrace{3\, d_{\text{model}} d_{\text{ff}}}_{W_g, W_u, W_d}.$$

Plus $2V d_{\text{model}}$ for untied embedding/unembedding.

Llama-3-70B ($d_{\text{model}} = 8192$, $L = 80$, $n_{\text{h}} = 64$, $n_{\text{kv}} = 8$, $d_{\text{head}} = 128$, $d_{\text{ff}} = 28{,}672$, $V = 128{,}256$): attention $= 8192 \cdot 128 \cdot (64 + 16 + 64) = 1.51 \times 10^8$; MLP $= 3 \cdot 8192 \cdot 28{,}672 = 7.05 \times 10^8$; per block 8.56×10^8; times 80 blocks $= 68.5$B; plus embeddings $2 \cdot 128{,}256 \cdot 8192 = 2.1$B. Total ≈ 70.6B. The name checks out, and note the split: *82% of trunk parameters are MLP*—the empirical basis for the claims of §7 and the target of §8.

DeepSeek-V3 (sparse): each MoE layer has 256 routed experts of width $d_{\text{exp}} = 2048$ at $d_{\text{model}} = 7168$ ($3 \cdot 7168 \cdot 2048 \approx 44$M each $\Rightarrow$ 11.3B per layer) — but a token touches 8 routed + 1 shared ≈ 0.40B of them. Summed over 58 MoE layers plus MLA attention, dense lead-in layers, and embeddings, this yields the published totals: 671B total, 37B active. The active fraction $\sim 5.5\%$ is the whole point of §8.

12.2 FLOPs: the $6ND$ rule

For a dense matrix $W \in \mathbb{R}^{a \times b}$, the forward product xW costs $2ab$ FLOPs per token (multiply–add pairs); the backward pass costs twice that ($2ab$ for the input gradient $\partial x = \partial y\, W^\top$, $2ab$ for the weight gradient $x^\top \partial y$). Summing over all matrices, whose entries total N (with attention-map FLOPs subleading for $T \lesssim d_{\text{model}}$, cf. §4.4):

$$C \approx \underbrace{2ND}_{\text{forward}} + \underbrace{4ND}_{\text{backward}} = 6ND \quad \text{FLOPs for } D \text{ training tokens},$$

with N the *active* parameters for MoE. Sanity check against a public run: Llama-3-70B on 15.6T tokens gives $6 \cdot 70 \times 10^9 \cdot 15.6 \times 10^{12} \approx 6.5 \times 10^{24}$ FLOPs; at 40% utilization of 10^{15} FLOP/s per GPU this is $\approx 1.6 \times 10^{10}$ GPU-seconds $\approx$ 16k GPUs for two weeks per epoch-equivalent—the right order of magnitude for the disclosed training scale, and a useful reflex: every frontier claim about model size, data, and cluster size is one multiplication away from an audit.

12.3 KV cache, revisited with real numbers

Per token, at bf16: Llama-3-70B (GQA-8): $2 \cdot 80 \cdot 8 \cdot 128 \cdot 2 = 327{,}680$ bytes ≈ 0.33 MB/token — an 8× saving over MHA's 2.6 MB (§6); a 128k context costs 43 GB/sequence. DeepSeek-V3 (MLA): $(512 + 64) \cdot 61 \cdot 2 \approx 0.07$ MB/token, under 9 GB at 128k—the difference between serving a handful of long-context users per node and serving dozens. This closes the loop opened in §6: the attention variants of 2023–25 are legible only through this arithmetic.

13 Where to go next

A short, opinionated path through the primary sources, in reading order:

1. *Vaswani et al., "Attention Is All You Need" (2017)* — reread after this paper; notice how much is scaffolding for translation.
2. *Zhang & Sennrich, RMSNorm (2019)*; *Xiong et al., "On Layer Normalization in the Transformer Architecture" (2020)* — the pre-norm analysis of §3.1 made rigorous.
3. *Su et al., RoFormer (2021)* for RoPE; *Peng et al., YaRN (2023)* for §9.2.
4. *Shazeer (2019)* for MQA; *Ainslie et al. (2023)* for GQA; *DeepSeek-V2 (2024)* for MLA, including the absorption and decoupled-RoPE details of §6.3.
5. *Shazeer, "GLU Variants Improve Transformer" (2020)* — two pages, including the famously honest attribution of success to "divine benevolence."
6. *Fedus et al., Switch Transformer (2021)*; *Mixtral (2023)*; *DeepSeek-V3 Technical Report (2024)* — the single most informative public document about frontier-scale training, covering MoE, MLA, FP8, MTP, and the cluster engineering of §11.
7. *Dao et al., FlashAttention (2022)* — read alongside Proposition 11.1.
8. *Kaplan et al. (2020)* and *Hoffmann et al., Chinchilla (2022)* for §10.5; *Yang et al., Tensor Programs V (2022)* for μP.
9. *Leviathan et al. (2023)* for speculative decoding; *Kwon et al., vLLM/PagedAttention (2023)* for §11.4.
10. The *Llama 3 Herd of Models (2024)*, *Gemma 3 (2025)*, and *GPT-OSS model card (2025)* papers as complete worked examples of everything above in one place.

What this paper deliberately excluded: everything after pretraining (supervised fine-tuning, RLHF/RLAIF, reasoning-trained models), multimodality (vision encoders and their fusion into the token stream), and tool use. The base object those systems are built on is the one described here: a pre-norm, decoder-only, RoPE-rotated, RMS-normalized, SwiGLU-gated, expert-sparse, KV-cached stack of matrix multiplications, trained by maximum likelihood to predict the next token.