The Evolution of Large Language Model Architectures: From the Transformer to Hybrid Models (2017–2026)

Finn Reimann
Independent Researcher
mail@finnrmn.com

16. September 2026

Abstract

The Transformer of 2017 still sits at the core of every current large language model, yet almost everything around it has been rebuilt, some parts repeatedly. This survey traces the architectural development of text-only LLMs from 2017 to early 2026 as four epochs, each driven by a concrete bottleneck. In the exploration epoch (2018–2019), encoder-, decoder-, and encoder-decoder designs competed, and decoder-only prevailed for structural reasons that paid off only later. In the scaling epoch (2020–2022), capability was bought with parameters and data, while the first efficiency ideas failed for lack of a binding problem. In the efficiency epoch (2023–2024), inference cost became that problem: the field converged on the modern decoder recipe, attacked the KV cache factor by factor (GQA, MLA, sliding windows, paging), and revived mixture-of-experts and recurrent sequence mixing; nearly every mechanism was a rediscovery of a previously shelved idea. In the convergence epoch (2025–2026), independent model families arrived at the same design principle: a small minority of full-attention layers, interleaved with linear mixers and routed experts, retained to restore the exact recall that fixed-size states surrender. In parallel, reasoning models opened a test-time-compute axis whose cost lands on the same object: the memory a model carries per generated token. I synthesize four long-run development lines, compare the architecture families qualitatively, and state the open problems, including the 2026 dissent over the attention operator itself. Every core concept is explained once and in full, so the survey serves simultaneously as an introduction for students and a reference for researchers.

Keywords large language models · Transformer · architecture survey · key-value cache · mixture-of-experts · state-space models · linear attention · hybrid architectures · test-time compute · scaling laws

Preprint, version 1.0 (September 2026), not peer-reviewed. Available at https://finnrmn.com/papers/llm-architectures/. DOI: 10.5281/zenodo.22785090.

1 Introduction

1.1 Motivation and research questions

In June 2017, a machine-translation paper proposed doing away with recurrence and convolutions in favor of an attention-only architecture Vaswani et al. (2017). Nine years later, its descendants write software, pass professional examinations, and reason through problems token by token, and the architecture at their core is still the one that paper drew. Yet everything "around" that core has been rebuilt: how models are trained, how large they are, which parameters a token touches, how attention is paid for, and above all, how much memory a model must carry for every token it generates.

A newcomer who opens the literature today faces this history in shuffled form: thousands of papers, dozens of named mechanisms (e.g., GQA, MLA, MoE, SSM, NSA, ...), and survey treatments that either explain everything at once or one axis at a time. What is hard to find is the "story" behind the evolution, the problem each wave of architectures was solving, why particular answers won while technically impressive alternatives vanished, and why ideas pronounced dead in 2020 run the frontier in 2026. This survey tells that story, and commits to discipline while telling it: every architectural wave is presented as a response to a concrete, nameable bottleneck, and every "winner" comes with an honest account of what it cost.

Three research questions organize the paper:

RQ1 (descriptive): Which architectural approaches and optimizations were developed between 2017 and 2026, and how do they build on one another?

RQ2 (analytical): Which bottlenecks drove each wave of development (e.g., trainability, capability, inference cost, context length, reasoning), and with which trade-offs were they resolved?

RQ3 (synthetic): Which long-run development lines emerge, where, and on what, has the field converged as of 2026?

Figure 1 maps the terrain these questions cover: the principal papers of this survey's corpus on one timeline from 2017 to 2026, grouped by research line and connected by descent.

1.2 Scope

The subject is the **architecture of text-only large language models**: the sequence-mixing operator and its inference state, parameter activation (dens vs. routed), the components of the block, and the optimizations that act on them (cache compression, attention sparsification, hybrid layer allocation). Scaling and alignment appear to the extent that they explain architectural development: scaling laws reshaped what models were built, and RLHF (Reinforcement Learning from Human Feedback) turned them into the deployed systems whose serving cost then drove the architecture. Outside the scope entirely: other modalities and their fusion architectures, training infrastructure and parallelism, prompting techniques and applications, and benchmark league tables.

Note on the scope: Since GPT-4 (March 2023), no closed frontier lab has disclosed a model architecture. This servery therefore treats closed models as milestones in the narrative while conducting technical analysis only where primary documentation exists. Additional note on evidence: mechanisms and number in this survey are draw from the primary papers, and quantitative comparisons across models are deliberately avoided: the published figures of this literature are self-reported under training data, budgets, and evaluation protocols, so cross-model tables in this survey compare designs qualitatively rather than scores numerically.

1.3 Related surveys

Encyclopedic surveys, most prominently the continuously updated Zhao et al. (2026) and its successors, cover the entire LLM lifecycle: data, training, adaptation, evaluation, applications. Architecture is one chapter among many, treated as an inventory. This paper is the complement, narrow in subject and deep in development: it explains why the inventory looks the way it does.

Single-axis surveys treat one architectural dimension exhaustively: efficient attention (Tay et al. (2022)) foundational taxonomy, which predates Mamba and the hybrid wave; Sun et al. (2025) recent "Speed Always Wins", the strongest current catalogue of efficient architectures including hybrids and diffusion LLMs), mixture-of-experts (MoE) Cai et al. (2025), state-space models (Qu et al. (2026)), small models Sakib et al. (2025), and reasoning Zhang et al. (2026). These are reference works for their axes; none narrates how the axes interact over time, and the central developments of 2023-2026 (GQA → MLA, the recurrence renaissance, hybrid convergence, reasoning's cost landing on the cache) are precisely interactions between axes.

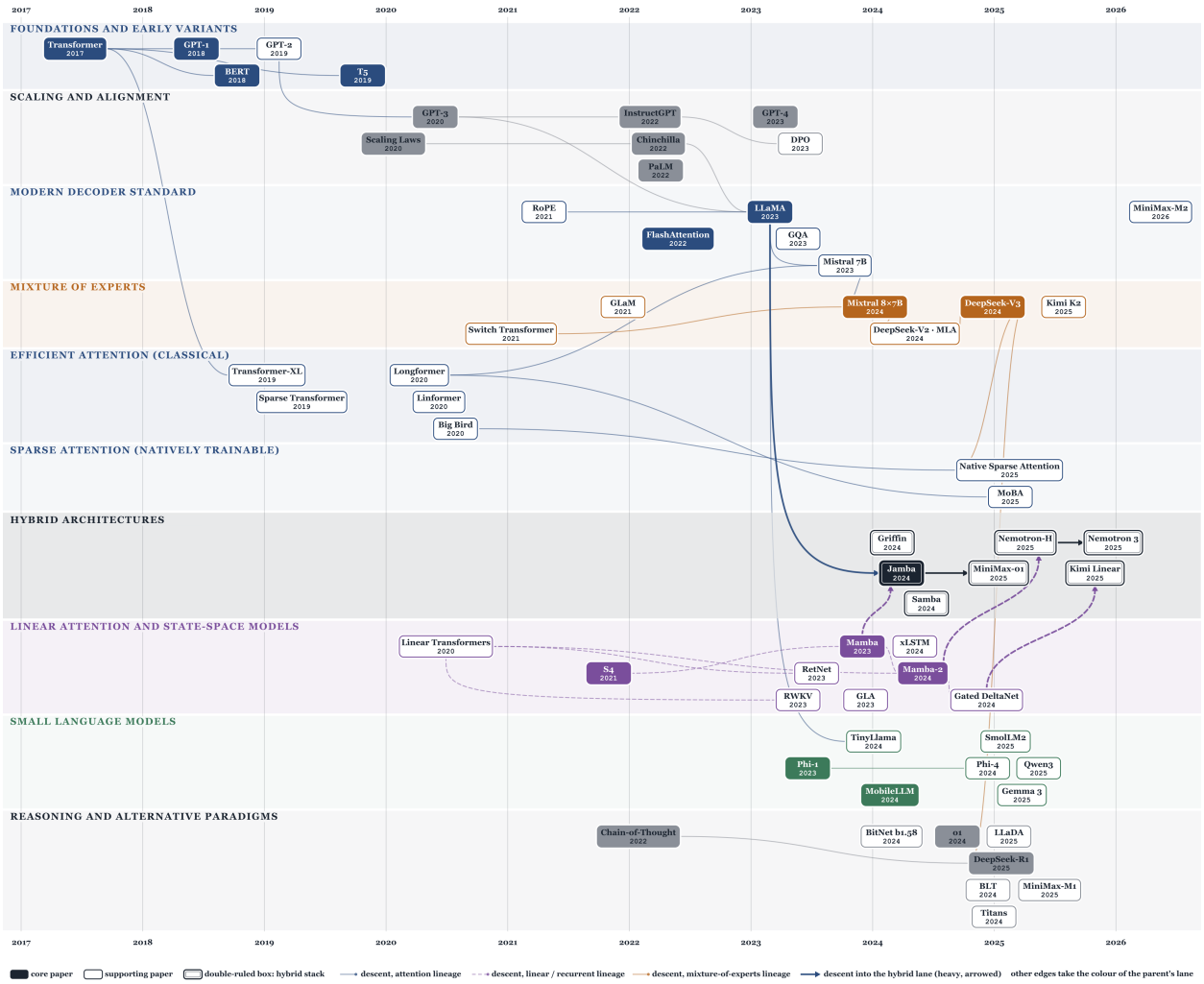

Figure 1: Genealogy of LLM architectures, 2017–2026, on a single time axis. Ten research lanes run top to bottom (foundations, scaling and alignment, the modern decoder standard, mixture-of-experts, classical efficient attention, natively trainable sparse attention, hybrid architectures, linear attention and state-space models, small models, reasoning and alternative paradigms), and time runs left to right at the arXiv v1 date (release date for GPT-1, GPT-2, and o1). Filled chips are core papers of this survey's corpus, outlined chips supporting ones, and double-ruled chips hybrid stacks; edges denote conceptual descent in the color of the parent's lane, and the heavy arrows enter the hybrid lane from above (attention lineage) and from below (recurrent lineage). Read across the 2023 boundary, the timeline shows three things at once. Lanes that dominate 2023–2026 are seeded in 2019–2021 and then lie fallow (Sparse Transformer, Switch Transformer, Linear Transformers, S4): the seed→harvest pattern of Section 7.1. After 2023 the modern-decoder recipe consolidates, MoE returns to production (Mixtral → DeepSeek-V3 → Kimi K2), and the linear/recurrent line matures (Mamba → Mamba-2 → Gated DeltaNet); the two lineages meet in the hybrid lane, from Jamba and Griffin/Samba to MiniMax-01, Nemotron-H, Nemotron 3, and Kimi Linear. MiniMax-M2 (top right, 2026) marks the return to full attention, the dissent of Section 6.7, visible on the timeline itself. A chip's row within a lane only prevents overlap and carries no meaning.

Historically framed treatments of the Transformer lineage exist but stop before the epoch that motivates this paper (the 2025–2026 convergence of independent families on hybrid designs, trainable sparsity, and the test-time-compute axis) or shift to the systems level.

Against all three groups, this survey's contribution is the combination of: (a) a **bottleneck-driven narrative**, with architectures explained as answers to binding constraints, including why losers lost and why failed ideas returned; (b) **coverage through early 2026**, including the hybrid convergence, its parameter-level dissent, and the interaction of reasoning with inference state; and (c) a **didactic architecture** designed to serve as a first text as well as a reference, described next.

1.4 How to read this survey

The paper is organized as **four epochs**, each defined by its binding bottleneck, each told in the same shape: *Setting* (what constrained the filed), *Developments* (the new approaches, from their primary sources), *Retrospect* (what won, what lost, why, and at what trade-off).

- **Chapter 2 (Foundations)**: the 2017 Transformer, explained in detail. Its KV-Cache (2.5) is the single most load-bearing section of this paper.
- **Chapter 3 (Epoch I, Exploration, 2018-19)**: encoder/decoder/both; why decoder-only won.
- **Chapter 4 (Epoch II, Scaling, 2020-22)**: GPT-3 scaling laws, RLHF, and the first premature efficiency wave.
- **Chapter 5 (Epoch III, Efficiency and the open wave, 2023-24)**: the modern recipe, the attacks on the cache, the MoE and recurrence renaissance, first hybrid, and small models.
- **Chapter 6 (Epoch IV, Convergence and new axes, 2025-26)**: hybrid convergence and its dissent, trainable sparsity, text-time compute, the trillion-scale open frontier.
- **Chapter 7-8**: synthesis (four development lines, comparison, open problems) and conclusion.

A Note on how to read: every core concept is explained once, in full, at the point where it enters the mainline; everything later is a delta against that explanation. A reader with Transformer background can skim Chapter 2 and rely on the back-references; a reader without it should read Chapter 2 carefully, after which every later mechanism in the paper is at most one step from something already explained. Mathematical notation is kept minimal and uniform (see 2).

2 Foundations: The Transformer

Every architecture in this survey is a descendant of one model: the Transformer, introduced by Vaswani et al. (2017) in 2017 for machine translation. This chapter explains that model in detail, because the rest of the survey is written as a series of deltas against it. Every reader should understand the core mechanisms, in particular the KV cache (sec. 2.5), as this is crucial for understanding the improvements and optimizations that emerged in the years that followed.

The chapter works form the outside in, and its level rises as it goes. Section 2.1 shows the whole model, deliberately at a level a newcomer to the field can follow. In the following sections elements and formulas (e.g., attention, block, embedding, kv-cache, quadratic problem) are discussed in greater depth and detail, offering added value even for readers who are already familiar with the topic.

Table 1 collects the notation used throughout the survey. Symbols that occur in a single section only are defined where they appear.

Table 1: Notation used throughout the survey. The values these quantities take in the original base model are listed in table 3.

Symbol	Definition
n	Length of the input sequence, measured in tokens.
d	Model dimension: the number of components of the vector that represents one token.
L	Number of blocks stacked to form the model. The original paper denotes the same quantity by N.
h	Number of attention heads computed in parallel within one attention sublayer.
d_h	Dimension of a single attention head, equal to the model dimension divided by the number of heads.
h_{kv}	Number of key/value heads. Equal to h in the original design; smaller in several later variants.
d_{ff}	Inner dimension of the position-wise feed-forward network.
$\lvert V \rvert$	Size of the vocabulary: the number of distinct tokens the model can read and produce.

2.1 Architecture overview

What it was built for. The Transformer was originally built as a translation model: it reads a sentence in one language and writes it in another. In 2017 the standard design for this task was a pair of recurrent networks, an *encoder* that read the source into a sequence of vectors and a *decoder* that produced the translation one word at a time, joined by an *attention* mechanism that let the decoder look back at the source while writing (Sutskever et al., 2014; Bahdanau et al., 2015). Tokens flow through a recurrent network in strict order, so training cannot be parallelized, and information from distant tokens has to survive a long chain of intermediate steps. The Transformer keeps the encoder-decoder architecture, the attention, but removes the recurrence: every position is processed in parallel, and the distance between any two tokens is one attention step. Vaswani et al. (2017) argue the case on three criteria: (1) computation per layer, (2) the number of operations that must run in sequence, and (3) the path length between distant positions. The practical payoff was speed: the model reached a new state of the art translation quality after as little as twelve hours of training on eight GPUs (Vaswani et al., 2017).

What it is made of. Underneath, the Transformer is an ordinary neural network (NN). Every token is represented by a vector of d numbers, so a sentence of n tokens is an $n \times d$ matrix. Each layer transforms that matrix by multiplying it with learned weight matrices and applying a simple nonlinear function. Furthermore layers are stacked like hidden layers in an NN, so the output of one is the input of the next. The model follows the basic principle of machine learning (ML): nothing is programmed by hand: the weights start random and are adjusted by gradient descent to reduce a loss; here the cross-entropy loss between the model's predicted distribution over the next token and the token that actually follows. What distinguishes the Transformer from other networks is not this recipe but the particular layer types it stacks and the way it arranges them. One structural fact organizes the rest of this chapter: inside the stack, every layer maps an $n \times d$ matrix to an $n \times d$ matrix. Only the two ends change shape, the embedding at the bottom, which turns token identities into vectors, and the output head at the top, which turns vectors back into probabilities over the vocabulary.

Reading the figure. Figure 2 shows the model architecture from Vaswani et al. (2017). It is read from the bottom up: the Left stack is the encoder, which receives the source sentence; the right stack is the decoder, which produces the translation. Each stack is $L = 6$ identical blocks, and the whole architecture is assembled from five kinds of building blocks:

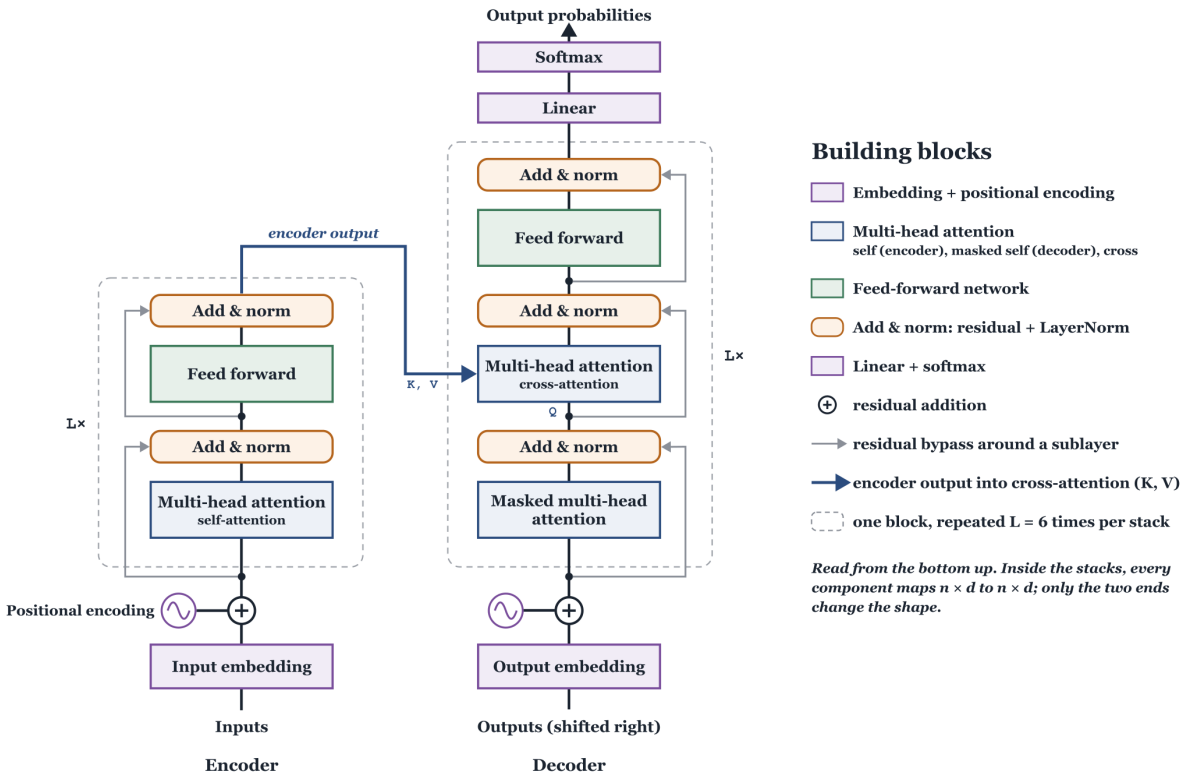

Figure 2: The Transformer as encoder-decoder, with the building blocks color-coded. Left: the encoder reads the source sentence. Right: the decoder writes the target, attending to its own earlier tokens (masked), and through cross-attention, to the encoder's output. (Based on Vaswani et al., 2017, Figure 1)

- **Token embedding and positional encoding** (sec. 2.4), which turn input tokens into vectors and adds unique positional information.
- **Multi-head attention** (sec. 2.2), the core of the architecture that enables the model to learn information and dependencies between tokens in a sequence.
- **The feed-forward network** (sec. 2.3), processes each token's representation independently through non-linear projections to store factual knowledge and learn richer, higher-level feature abstractions.
- **Add & norm** (sec. 2.3), the residual connection and layer normalization wrapped around every sublayer. Enabling information to pass throw the network without changes, to reach deeper layers.
- **Linear and softmax** (sec. 2.4), which turn the output vector of the decoder into a probability distribution to select the next, following generated token.

The two stacks read differently. The encoder sees the whole source at once: every position may attend to every other, so its output vector already reflects the following tokens. The decoder writes the target left to right, and each of its position may attend only to earlier target positions, but also through cross-attention, to the entire encoder output.

How it is trained. The model is trained by *teacher forcing*: the decoder receives the correct translation, shifted one position to the right and prefixed with a start symbol ($< s >$), and learns to predict at every position the token that comes next (Table 2). Because all positions are computed in parallel, one pass over the sentence yields twelve predictions at once, and the loss is the cross-entropy summed over them. The shift is what prevents cheating: the input at position t is token $t-1$, so the prediction for token t never sees token t itself. The causal mask of section 2.3 closes the remaining loophole, attention to tokens further to the right.

Table 2: Decoder teacher forcing example: "<s> The animal didn't cross the street because it was too tired". Shown at word level for readability; the actual tokens are subwords (see Section 2.4).

Position	1	2	3	…	10	11	12
Decoder input	<s>	The	animal	…	was	too	tired
Target	The	animal	didn't	…	too	tired	</s>

Reference configuration. Table 3 lists the base configuration of the original paper. Its number are the survey's reference point: the parameter arithmetic of section 2.3 and the cache arithmetic of section 2.6 both start from them.

Table 3: The base configuration of the Transformer from Vaswani et al. (2017, Table 3).

Symbol	Meaning	Base model
$L(\text{former}N)$	number of transformer blocks	6
d	embedding dimension	512
h	number of attention heads	8
d_h	dimension per head	64
d_{ff}	feed-forward dimension	2048
$\lvert V\rvert$	vocabulary size (byte-pairs)	$\approx$ 37,000
$\boldsymbol{\Sigma}$	total parameter count	65 M

2.2 Scaled dot-product and multi-head attention

A language model has to solve a routing problem: the meaning of a token depends on other tokens, and which ones matter changes from sentence to sentence. In *"The animal didn't cross the street because **it** was too tired"*, resolving *it* requires looking at *animal*, but swapping *tired* for *wide* and the relevant token becomes *street*. Recurrent networks, the dominant architecture before 2017, handled this by passing a summary vector form token to token: a game of telephone in which distant information had to survive many intermediate steps. Self-attention removes the telephone line: every token looks at every other token directly, in a single step, and learns where to look.

The mechanism is a soft lookup. From its current representation, every position derives three vectors through learned linear projections: a **query** ("what am I looking for?"), a **key** ("what do I contain?"), and a **value** ("what do I contribute if selected?"). Positions i scores its query against every key by a dot product, converts the scores into weights with a softmax, and returns the weighted average of the values. Stacking the queries, keys and values of all n positions into matrices Q, K, V, the whole operation is two matrix products:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_h}}\right)V \tag{1}$$

The division by $\sqrt{d_h}$ is a numerical safeguard: for high-dimensional vectors the dot products grow large (their variance scales with d_h), which pushes the softmax into saturated regime with vanishing gradients. Rescaling with the vector dimension of the attention-heads restores unit variance Vaswani et al. (2017).

Figure 3 shows what the weights look like on the example, for one head, for one head, with queries as rows and keys as columns. In panel (a) the row of *it* places most of its weight on the key *animal*; in panel (b), with *tired* swapped for *wide*, the same row moves its weight to *street*. Ever row sums to one. Panel (c) shows the casual mask explained in section 2.3.

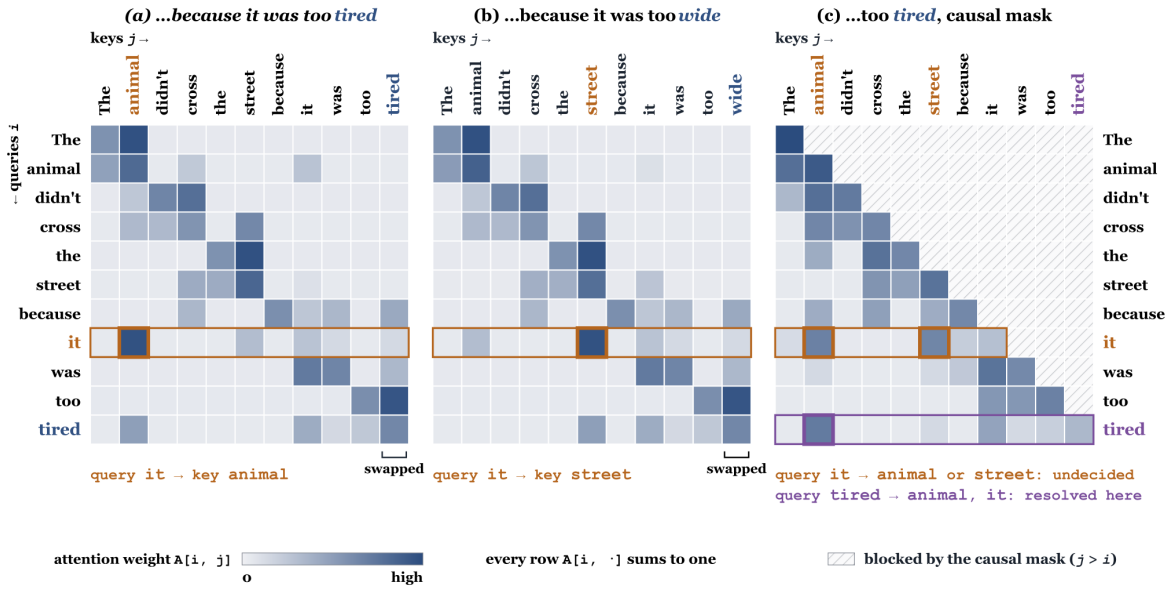

Figure 3: The attention weights of one head as an $n \times n$ matrix, queries as rows and keys as columns. Left: in *"... because it was too tired"* the row of the query *it* places most of its weight on the key *animal*. Right: with *tired* swapped for *wide*, the same row shifts its weight to *street*. Every row sums to one; the weights are illustrative, not measured.

A single attention operation can only express one pattern of relevance per position, because the softmax averages everything it selects. The Transformer therefore runs h **heads** in parallel ($h = 8$ in the original base model), each with its own projections at reduced dimensions $h_h = \frac{d}{h}$, and concatenates and re-projects their outputs:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\, W^O, \qquad \text{head}_i = \text{Attention}(QW_i^Q,\, KW_i^K,\, VW_i^V) \quad (2)$$

Because each head works at $\frac{1}{h}$ of the full dimension, the total cost is close to one full-dimension head, but the heads can specialize: one may track syntactic dependencies while another tracks coreference Vaswani et al. (2017). Two structural facts, easy to overlook here, drive much of the later story. First, the attention weight matrix has $n \times n$ entries, since every query meets every key, causing the quadratic cost analyzed in section 2.6. Second, in this original design every head carries its own keys and values, so the number of key/value heads $h_k v$ equals h. Later work (sec. 5.3) breaks exactly this identity, which is why h_{kv} is carried as a separate symbol in this work from the start.

Figure 2 contains this attention mechanism three times. In the encoder's self-attention, queries, keys, and values all come from the same source sequence and every position sees every other. In the decoder's self-attention they come from the target sequence, and a mask restricts each position to its left, to ensure that the next token to be predicted by the model is not already provided as the solution. In cross-attention the queries come from the decoder and the keys and values from the encoder's output.

2.3 The Transformer block

Attention is embedded in a repeating unit, the **Transformer block**, which is stacked L times from each stack of the model. Every sublayer in the block maps an $n \times d$ matrix to an $n \times d$, and every sublayer is wrapped in the same pattern, $\text{LayerNorm}(x + \text{Sublayer}(x))$, a residual connection followed by a normalization, drawn as *Add & Norm* in Figure 2. An encoder block has two sublayers, self-attention and a feed-forward network; the original decoder block hast three, with cross-attention inserted between them.

Masked multi-head attention. The decoder's self-attention carries a mask. Before the softmax, every score between a query at position i and key at position $j > i$ is set to $-\infty$, so that the corresponding weight becomes exactly zero after

the softmax; the surviving weight form a lower triangle. The purpose is consistency between training and inference. During training (Table 2) all positions are predicted in parallel from the shifted target, without the mask position t could simply read its answer from position $t+1$. At inference the future does not exist yet, so a model that had learned to rely on it would fail. Panel (c) of Figure 3 shows what the mask does to the example. An encoder has no mask and such constraint, which is why panels (a) and (b) are legitimate encoder patterns and not decoder ones. Section 3 returns to this difference as the divide between BERT and GPT.

Feed-forward network. The second sublayer is a position-wise feed-forward network (FFN): two linear layers with a ReLU activation in between, $\mathrm{FFN}(x) = \max(0,\, xW_1 + b_1)W_2 + b_2$, applied to every position independently and identically, with an inner dimension $d_{\text{ff}} = 2048$ against $d = 512$ in the base model (Vaswani et al., 2017). No information crosses positions here. If attention is where a token gathers what it needs from the rest of the sentence, the FFN is where it works on what it gathered. Interpretability work later found that these layers behave like key-value memories in which patterns and facts are stored (Geva et al., 2021): the FFN is where the model keeps most of what it knows, while attention decides where information flows. If we take a closer look at the parameter count, this interpretation seems plausible. Per block, four attention projections hold $4d^2$ parameters and the two FFN matrices $2 \cdot d \cdot d_{\text{ff}} = 8d^2$; at $d = 512$ that is about 4.2 million against 16.8 million, so roughly three quarters of the parameters in a dense block sit in the FFN. Section 5 shows how mixture-of-experts exploits precisely this imbalance, replicating the FFN many times while activating only a fraction of it per token inference.

Residual connections. Each sublayer adds its output to its input rather than replacing it. The idea came from computer vision, where stacking more layers had made networks harder to train, not better, until He et al. (2016) let every layer learn only a correction to its input. Two things follow. Gradients have a direct path from the loss to every layer, so early layers keep learning even in deep stack. And a single d-dimensional vector per position, the *residual stream*, flows through the entire depth of the model, with every sublayer writing incremental updates to it. Interpretability work later showed that this stream behaves like a prediction being progressively refined layer by layer (Geva et al., 2021).

Layer normalization. For each position, layer normalization centers the input vector $x \in \mathbb{R}^d$ by subtracting its mean μ and scales it to unit variance using the standard deviation σ across its d feature dimensions. Subsequently, it applies a learned gain parameter γ and bias β (Ba et al., 2016):

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i \qquad \sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2 + \epsilon} \qquad \mathrm{LayerNorm}(x) = \gamma \odot \frac{x-\mu}{\sigma} + \beta \tag{3}$$

Without it, the magnitudes in the residual stream drift as depth grows and training becomes unstable. The position of the normalization in the network was original after each residual sum ("post-norm"), which interrupts the stream once per sublayer; in later variants its placed before the sublayer ("pre-norm") leaving the stream unbroken and trains more stably at depth (Xiong et al., 2020), and it became one of the changes of the modern recipe (sec. 5.2). Figure 4 shows both arrangements.

Cross-attention. The decoder's third sublayer is attention with a different wiring: queries from the decoder's residual stream, keys and values from the encoder's final output. For the task originally intended translation task (text from one language to another), the decoder has access to the unmasked attention of the sequence, but in the source language, and is expected to learn the next tokens of the target language in this context. In the language-model line of this survey the sublayer disappears together with the encoder (sec. Section 3).

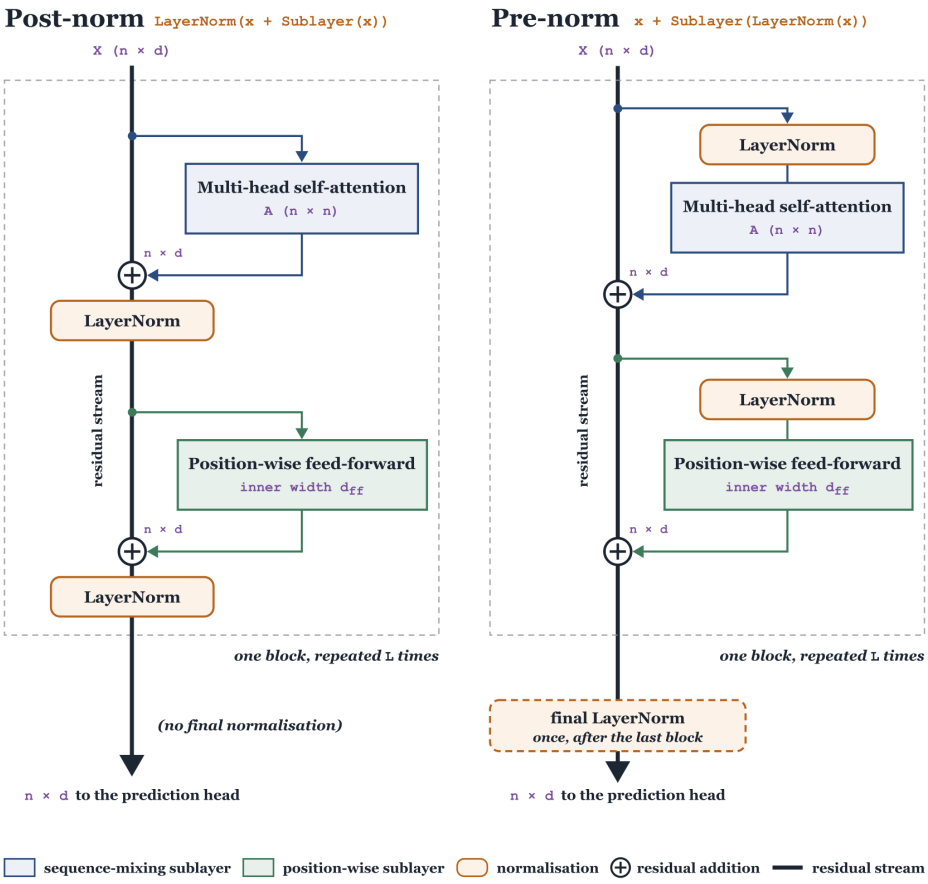

Figure 4: One Transformer block, in both normalization placements. Left: the original post-norm arrangement of 2017, in which the residual stream (heavy vertical line) is interrupted by a LayerNorm after each residual addition. Right: the pre-norm arrangement that later became standard (Section 5.2), in which normalization moves inside each branch, the stream runs unbroken, and a single final norm follows the last block.

2.4 Embeddings, positional encoding, output layer

Four components sit outside the repeating block. Two before turn text into the $n \times d$ matrix the block operate on; two after turn the final matrix back into a choice of the next token.

Tokens. The model does not operate on word or character level. A tokenizer first splits text into units from a fixed vocabulary of subwords, learned from the training corpus by byte-pair encoding: Throughout the entire training text, frequently occurring pairs of characters are combined into a single token. This process is repeated until the desired number of token pairs (the vocabulary) is reached (Sennrich et al., 2016). The result is that frequent words stay whole, rare words fall apart into pieces, and nothing is ever out of vocabulary The original English-German model uses one vocabulary shared between both languages of about 37,000 tokens. On the example *animal* and *street* are single tokens, while *didn't* and *überquerte* split into two or three (e.g., `didn` `'t` and `ü` `ber` `qu` `ert`). The tokenizer fixes $|V|$, and with it the size of the two matrices that follow.

Token embedding. An embedding is a learned table with $|V|$ rows and d columns; token number k becomes row k. This is the first change of shape in the model: a sequence of n token identities becomes the $n \times d$ matrix on which everything else operates. The table is large, at $|V| \approx 37{,}000$ and $d = 512$ it holds about 19 million parameters, nearly a third of the base model, and scale its rows by $\sqrt{d}$ before use (Vaswani et al., 2017).

Positional encoding. Self-attention is blind to order: shuffle the input tokens and the outputs shuffle correspondingly, because attention weights depend only on content. The original model injects order by adding a **positional encoding** to

the token embeddings at the beginning of the stack, a fixed vector per position built from sinusoids of geometrically spaced wavelengths:

$$PE_{(pos,\,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right), \qquad PE_{(pos,\,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right) \tag{4}$$

Each pair of dimensions oscillates at its own wavelength, from 2π to $10000 \cdot 2\pi$, so every position receives a unique signature across the d dimensions, much as the hands of a clock identify the exact time. This mechanic was chosen for two hoped-for properties: that a fixed offset corresponds to a fixed linear transformation of encoding, which should make relative positions easy to learn, and that the model might extrapolate to lengths unseen in training; a learned table of position vectors performed identically in their experiments (Vaswani et al., 2017). Two consequences matter downstream. Position enters once, additively, at the start of the stack, so it is a property of the representation rather that pf the attention operator; and the coding is absolute. Both hopes were only partially fulfilled, and position encoding remained a site of persistent renovation, with RoPE, ALiBi, and YaRN revising exactly these two properties (sec. 5.2, 5.3).

Linear and softmax. At the end of the decoder stack, a linear layer maps each position's d-dimensional vector to $|V|$ scores, the *logits*, one per vocabulary entry, and a softmax function turns them into a probability distribution over the next token. This is the second change of shape: $n \times d$ becomes $n \times |V|$. the linear layer is another $|V| \times d$ matrix, and the original model shares it with the embedding table, so the row that represents *it* on the way in also scores *it* on the way out (Vaswani et al., 2017; Press and Wolf, 2017). During training all n rows of the output are used, one prediction per position (Table 2); during inference only the last row matters.

From distribution to text. A distribution is not yet a token. Figure 5 follows one step: after . . . *because it was too*, the top-of-stack vector of the last position passes through the linear layer and the softmax, and the resulting distribution splits between *tired* and *wide*, the two continuations already met in section 2.2. Nothing in the input rules either one out, so the model assigns probability to both, with *tired* ahead. A decoding rule then commits to one token: the most probable one (greedy decoding, which takes *tired*), a random draw from the distribution (sampling, which would return *wide* roughly one time in four), or, in the original translation setting, a beam search that keeps the four most probable partial outputs in play (Vaswani et al., 2017). Whatever the rule, the chosen token is appended to the input and the model runs again. That loop is generation, and its cost is the subject of the next two sections.

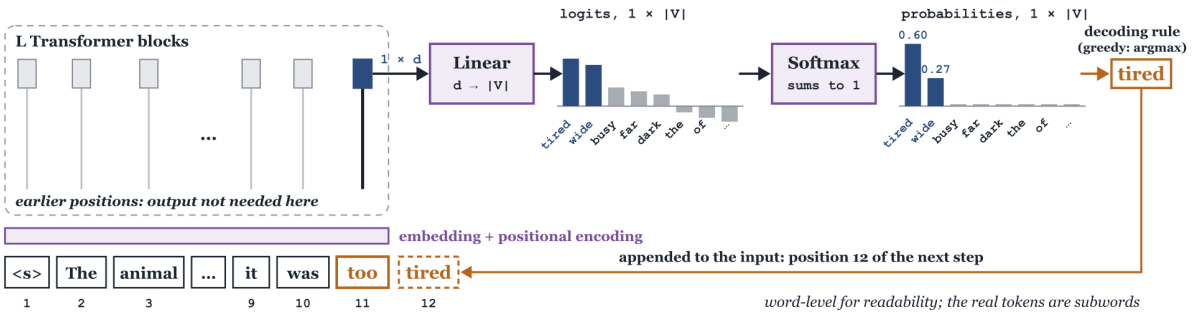

Figure 5: One inference step. The d-dimensional vector at the top of the stack for the last position (*too*) passes through the linear layer, which produces one logit per vocabulary entry, and the softmax, which turns the logits into probabilities. The distribution splits between *tired* and *wide*; a decoding rule picks *tired*, which is appended to the input as position 12 and becomes the last position of the next step. Bar heights are illustrative, and the vocabulary is truncated to eight entries.

2.5 Autoregressive inference and the KV cache

How a Transformer is trained and how it is used to generate text at inference are two different computational processes. It lies in what each step computes and in what limits that computation, and after 2022 it became the dominant force in architecture design.

The Differ. During training the entire target sequence is known in advance, so all n positions are processed in one parallel pass over the shifted input (Table 2). While during inference token generation is autoregressive: the model reads the input, predicts one token, appends it, and runs again on the extended input, so sequence of n tokens costs n forward passes. The causal mask of section 2.3 is what makes these passes consistent with the single pass of training. Because no position ever attends to its right, the representation of position j is the same whether the sequence ends at j or continues beyond it, in every layer of the stack. The model can be trained in parallel and used sequentially without changing a single weight.

One token generation. The same consistency determines how much of each pass is genuinely new. Figure 6 continues the step of the example from Figure 5: *tired* has been appended as position 12, and one head of one block now processes the twelve-token input. Consider the $n \times n$ attention matrix of that head under the causal mask, as in panel (c) of Figure 3. Row i is computed from query i and the keys of positions 1 to i; nothing tp its right enters. When the sequence grows from eleven position to twelve, rows 1 to 11 are therefore exactly what eleven earlier steps already computed; recomputing would only reproduce known numbers. Row 12 is the only new row, and needs: the query of the new position and the keys of all twelve positions. The weighted sum that follows behaves the same way: only output row 12 is needed, and it needs the values of all twelve positions. Everything else in the step is either unchanged or unused. The queries of positions 1 to 11 are unused, because a query serves only its own row.

The cache. The keys and values of positions 1 to 11 are unchanged, which makes them safe to store: each is a position-wise projection of that position's representation (Section 2.2), which the causal mask guarantees never changes. Keys and values of the past are frozen. At each step the model computes query, key, and value for the one new position only, appends the new key and value to a buffer, and lets the single new query attend over the entire buffer. That buffer is the **key-value cache (KV cache)**. In Figure 6 it holds eleven keys and eleven values (blue): the step adds k_{12} and v_{12} (orange), computes one row of attention weights and one output row, and hands that row to the output head of Figure 5. Without the cache, every step would push the entire prefix through the stack again, and the arithmetic per step would grow with the length of the prefix; now each step performs one position's worth of arithmetic and reads what earlier steps have stored. The bookkeeping repeats in every head of every block, so the cache is not one buffer but $L \cdot h_{kv}$ of them.

The cost. Caching the keys and values of earlier computation comes with a cost. The computational overhead reduces, but the memory cost increases: per token each of the L blocks stores one key and value for each of the h_{kv} heads of width d_h, so a sequence of length n tokens occupies

$$\underbrace{2}_{K \text{ and } V} \cdot n \cdot L \cdot h_{kv} \cdot d_h \tag{5}$$

elements. For the base model of Table 3 that is $2 \cdot 6 \cdot 8 \cdot 64 = 6{,}144$ entries per token, 12 KB at 16-bit precision. Every factor except n s fixed at design time, and was later (discussed in Section 5) changed by some architectural proposal: h_{kv} by grouped-query attention, the product $h_{kv}d_h$ by low-rank compression (MLA), L by hybrid stacks in which only a minority of blocks keeps a cache, and the dependence on n itself by operators whose state does not grow at all.

Memory bandwidth. Why the cache matters so much becomes clear from the two phases of generation. During prefill, the prompt is processed all at once: attention is a pair of dense matrix products over all positions in parallel, exactly the workload GPUs are built for, limited by raw compute, and computationally the same regime as training; its by-product is the initial cache. During inference, the model emits one token per step, and each step must read the entire cache, together with the model weights, from memory to perform one position's worth of arithmetic. Inference is therefore bound not by compute but by memory bandwidth; in the words of Ainslie et al. (2023), decoder inference is bottlenecked by "the memory bandwidth overhead from loading decoder weights and all attention keys and values at every decoding step".

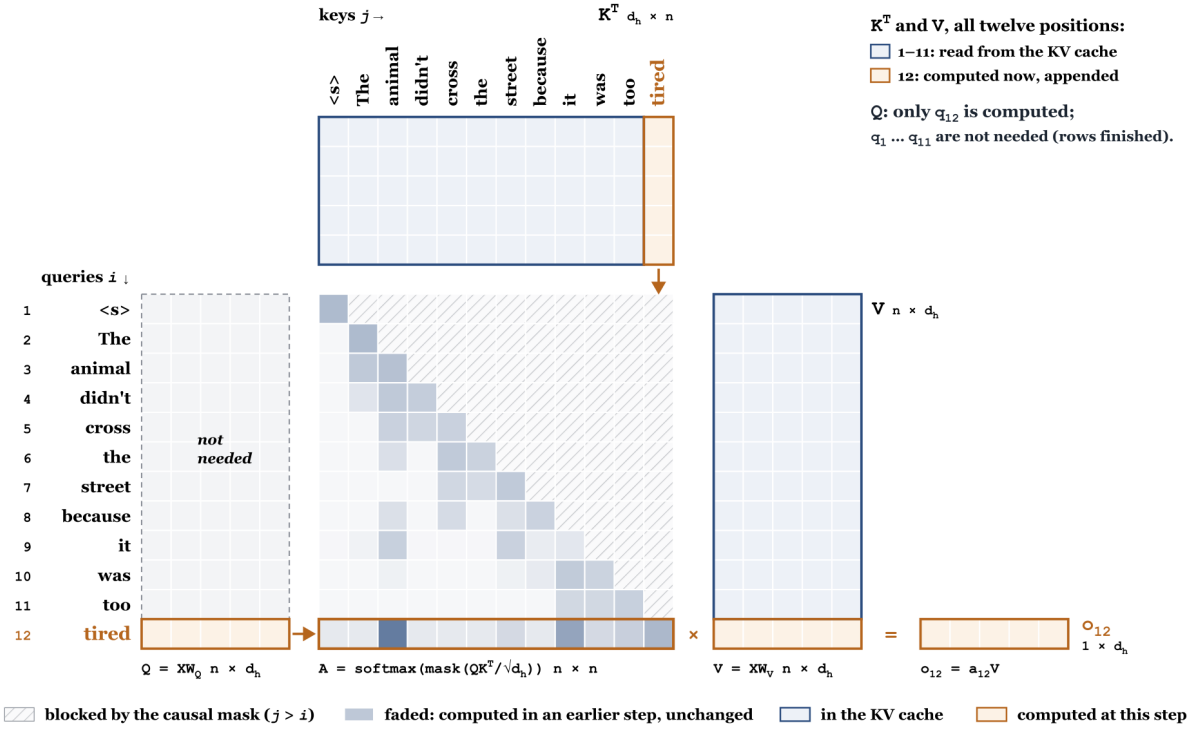

Figure 6: One decoding step with the KV cache, continuing Figure 5: *tired* appended as position 12, one head of block processes the twelve-token input, laid out as the matrix product (Q left, K^T top, mask attention matrix center, V right). Weights are illustrative, not measured.

2.6 The quadratic problem

The costs of this architecture, made explicit, define the agenda for the entire efficiency literature.

Compute. A self-attention layer costs $O(n^2\, d)$ operations, because all n queries meet all n keys. Vaswani et al. (2017) tabulate this against $O(n\, d^2)$ for a recurrent layer and note the condition under which attention wins: it is cheaper than recurrence only while $n < d$. For the sentence-length sequence of 2017 (n in the tens, $d = 512$) the condition held comfortably. For the long-context models of 2020s (n in the hundreds of thousands, d in the low ten-thousands) it fails by orders of magnitude, and the failure was anticipated: the same paper already sketches restricted self-attention over a local neighborhood as future work, the seed of the entire sparse-attention line (Vaswani et al., 2017).

Memory. The KV cache grows linearly in n, but it is re-read at every decode step, so the bandwidth cost accumulated over generating n tokens is itself quadratic. Quadratic compute is a prefill problem; linear state growth is a decode problem. Keeping the two apart is essential for reading the literature: FlashAttention (Section 5.3) is an exact-attention implementation whose gains are in memory traffic, not operation count, while linear-attention models (Section 5.5) change the asymptotics themselves.

A back-of-envelope example makes the stakes concrete. Take a typical dense 7B-parameter model of 2023, with $L = 32$ blocks, $h = 32$ heads (full multi-head attention), and $d_h = 128$: the cache costs $2 \cdot 32 \cdot 32 \cdot 128 = 262{,}144$ entries per token, i.e. half a megabyte at 16-bit precision. A single 128k-token context then occupies roughly **64 GB**, the cache alone outweighing the model's own weights (14 GB) several times over, before any batching. Numbers of this kind, not accuracy curves, explain why the years after 2022 produced GQA, MLA, sliding windows, linear attention, and ultimately the hybrid stacks of 2025–2026.

3 Epoch I: Exploration (2018–2019)

3.1 The setting

In early 2018 the Transformer was a machine-translation architecture: an encoder-decoder trained from scratch on parallel text and evaluated on translation benchmarks. Within eighteen months it became the standard architecture of natural language processing. The decisive change was not on scale but a new training paradigm, **pretraining**. Instead of training a separate model per task on scarce labeled data, a model is trained once on large amounts of unlabeled text with a self-supervised objective, predicting held-out or upcoming tokens, and is then adapted to each downstream task by fine-tuning. Pretraining thus turns raw text into a supervision signal. The open question of the epoch followed directly: *which part of the Transformer is pretrained, and with which objective?* Three answers appeared between mid 2018 and late 2019, each retraining a different subset of the original encoder-decoder: the encoder alone (BERT, Section 3.2), the decoder alone (GPT, Section 3.3), or both stacks (T5, Section 3.4). All three share the attention operator, the block, and the position scheme of Section 2. They differ in two coupled design choices: the **attention mask** (bidirectional or causal) and the **pretraining objective** (masked, next-token, or span corruption). These two choices determine whether the model can generate text, whether it can use the KV cache of Section 2.5, and what fraction of the training tokens yields a supervision signal. Section 3.6 compares the three variants on exactly these properties.

3.2 Encoder-only: BERT and masked language modeling

BERT (Devlin et al., 2019) retrains only the encoder stack. The causal mask is removed, so every position attends to every other position in both directions; the decoder, and with it the cross-attention, is dropped. The authors describe the architecture as otherwise "almost identical" to the original; the contribution is the pretraining objective. A bidirectional model can see the next token, so next-token prediction is not a valid objective for it. BERT instead uses **masked language modeling**: 15 % of the input tokens are selected, most of them are replaced by a dedicated mask token, and the model predicts the original tokens at these positions from the context on both sides. A second objective, next-sentence prediction, trains the model on the relation between two text segments.

BERT-Large (L=24, d=1024, 340M parameters) set the state of the art on eleven language-understanding benchmarks. On GLUE it scored 80.5, against 72.8 for the best decoder-only model at the time, GPT-1 (Section 3.3) (Devlin et al., 2019). For tasks that map a given text to a label or a span (classification, extraction, question answering), bidirectional context is the more suitable inductive bias, and encoder-only models remained the standard choice for these tasks for several years.

The design has two structural limitations, both consequences of the mechanics in Section 2. First, an encoder-only model cannot generate text: masked prediction defines no ordering over positions, so there is no procedure for producing tokens one at a time. Second, and decisive for deployment (Section 3.6), it cannot use the KV cache of Section 2.5. Caching requires that a position's keys and values are final once computed; in a bidirectional stack every representation depends on both sides of the sequence, so appending one token changes all previous keys and values. In addition, the objective is sparse: only the 15 % of selected positions contribute to the loss. The authors note this cost and expect that more pretraining steps may be needed for convergence (Devlin et al., 2019).

3.3 Decoder-only: GPT-1, GPT-2, and next-token prediction

GPT-1 (Radford et al., 2018), published four months before BERT, retains the complementary half: the decoder stack with its causal mask, without encoder and cross-attention. Its objective is standard language modeling, **next-token prediction**. The targets are the input sequence shifted by one position, so every token of every sequence contributes to the loss, without any additional labeling. The paper's contribution is the complete transfer recipe: generative pretraining on unlabeled text (a 12-block model trained on the BooksCorpus dataset), followed by discriminative fine-tuning with one added linear layer per task. Its ablation quantifies the effect of pretraining: without it, the average score across the evaluated benchmarks drops from 74.7 to 59.9 (Radford et al., 2018).

GPT-2 (Radford et al., 2019) keeps the architecture and scales it to 1.5B parameters over 48 blocks, with a byte-level vocabulary. It introduces one architectural change with lasting consequences: layer normalization is moved from the output to the input of each sublayer. This pre-norm placement stabilizes the training of deep stacks and became the standard (Section 5.2). The conceptual change concerns the interface. If a task can be expressed as text ("translate to French:…"), a sufficiently good language model performs it without a task-specific head. Evaluated zero-shot, without any fine-tuning, GPT-2 set the state of the art on 7 of 8 language-modeling datasets. The paper records two limitations: zero-shot performance on most downstream tasks was still far from usable, and all four model sizes underfit the training corpus (Radford et al., 2019). Both point to scale as the next variable, the subject of Section 4.

3.4 Encoder–decoder: T5 and text-to-text

T5 (Raffel et al., 2020) retains the complete original architecture: encoder and decoder, connected by **cross-attention**, the one component that the other two variants discard. Cross-attention is the attention operator of section 2.2 with queries taken from the decoder and keys and values taken from the encoder output. On this architecture, Raffel et al. impose a uniform **text-to-text** format: every task (translation, summarization, classification, and even regression) is cast as text input and text output. The pretraining objective, **span corruption**, lies between masked language modeling and next-token prediction: contiguous spans of the input are removed and replaced by numbered sentinel tokens, and the decoder generates only the missing spans, in order. The encoder thus reads bidirectionally, the decoder writes autoregressively, and the short targets keep the cost per training step low (Raffel et al., 2020).

T5 is as much a systematic study as a model. The paper reports hundreds of controlled comparisons of objectives, architectures and transfer schemes, and its largest model (11B parameters) set a new SuperGLUE state of the art of 88.9, within one point of the human baseline (Raffel et al., 2020). One comparison is central to this survey. At matched training compute, the encoder–decoder outperforms a decoder-only language model on GLUE by a wide margin (83.3 vs. 74.7). The authors note that the comparison is confounded: at equal FLOPs, the encoder–decoder has twice as many parameters, and parameter-matched variants narrow the gap considerably (Raffel et al., 2020). On this evidence they recommend the encoder–decoder form. At the end of 2019, the benchmark results thus favored bidirectional encoders and gave no indication that decoder-only models would prevail.

3.5 Seeds of later lines

Three papers from 2019 introduced mechanisms that had little immediate effect but became central in later epochs. The pattern recurs throughout this survey: efficiency techniques often precede, by several years, the bottleneck that makes them necessary.

- **Transformer-XL** (Dai et al., 2019) lets a decoder reuse the hidden states of the previous text segment instead of recomputing them. The cached states extend the effective context, and a relative position encoding makes them addressable from the current segment. Carrying a fixed-size state forward instead of attending over the full history is the principle behind the recurrent models of Section 5.5.
- **Sparse Transformer** (Child et al., 2019) factorizes attention so that each position attends to a structured subset of positions, which reduces the cost to $O(n\sqrt{n})$. GPT-3 used its alternating dense and local pattern (Brown et al., 2020), and the line of sparse attention it opened leads to Native Sparse Attention in 2025 (Section 6.3).
- **Multi-query attention** (Shazeer, 2019) shares a single key/value head across all query heads, which shrinks the KV cache of Section 2.5 by a factor of h at a small cost in quality. The paper received little attention in 2019, when decoding was not yet bound by memory bandwidth, and became the direct ancestor of grouped-query attention (Section 5.3) four years later.

The epoch also produced a negative example. XLNet (Yang et al., 2019) trains an autoregressive model over permuted factorization orders, which captures bidirectional context without a mask token, and it outperformed BERT across a broad set of benchmarks. The approach was nevertheless not adopted: the two-stream attention it requires is complex, and the simpler objectives it competed with reached comparable results at scale without additional machinery. This is the first instance of a selection criterion that recurs throughout this survey: between two approaches of similar performance, the field adopts the simpler one.

3.6 Retrospect: why decoder-only won

Table 4: The three architectural variants of 2018–2019, compared on the properties that determined their long-term adoption.

Property	Encoder-only (BERT)	Decoder-only (GPT)	Encoder-decoder (T5)
Attention	bidirectional	causal	bi (enc) + causal (dec)
Objective	masked LM	next-token prediction	span corruption
Positions supervised	∼15 %	100 %	∼15 % (spans)
Can generate?	no	yes	yes (via decoder)
KV cache (Section 2.5)	impossible	full	decoder only
Benchmark standing 2019	dominant (NLU)	behind	strongest overall
Mainline status 2026	retrieval & embeddings	**the LLM default**	translation, niche seq2seq

The benchmark results of 2019 and the development of the following years point in opposite directions: the two variants with a bidirectional encoder led on GLUE and SuperGLUE, yet the decoder-only form became the default. Three structural properties explain this outcome; none of them is a benchmark score.

Signal Density. Next-token prediction produces a loss term at every position; the masked and span objectives do so at about 15 % of positions. At fixed data and compute, the autoregressive model therefore obtains more training signal per document. The advantage grows with scale, and BERT's authors had already acknowledged its cost (Devlin et al., 2019).

The text interface. A generative model needs not task-specific heads: the task specification is part of the input text (Section 3.3), so one set of weights serves an open-ended set of tasks. The zero-shot setting of GPT-2 developed one epoch later into in-context learning (Section 4.3) and into the interface of deployed LLMs. Encoder-only model, by contrast, require fine-tuning and thus a separately adapted copy of the model per task.

Cacheability. Only a causal model can use the KV cache of Section 2.5 without restriction. A bidirectional stack cannot cache at all, because appending a token invalidates every stored representation; an encoder-decoder caches only in the decoder and additionally pays for the second stack and cross-attention. The interface economics of Section 2.5, including the prefill/decode asymmetry, therefore apply in full only to the decoder-only form.

None of these properties was the stated selection criterion at the time. The papers of this epoch compare accuracy, and on accuracy the bidirectional objectives led (Devlin et al., 2019; Raffel et al., 2020). The decisive properties became relevant only later: signal density under scaling, cacheability in deployment, and the text interface for generality. The question was settled empirically by GPT-3 rather than by a controlled comparison. The other two variants remain in use: encoder-only models underlie much of today's retrieval and embedding infrastructure, and encoder-decoders persist where input and output are different objects, as in translation. From 2020 onward, however, "language model" refers by default to a causal decoder.

4 Epoch II: Scaling (2020–2022)

4.1 The setting

At the end of Epoch I the decoder-only line trailed the bidirectional variants on the language-understanding benchmarks. At the same time, the authors of GPT-2 had found that all four of their models underfit the training corpus, so additional capacity and data still reduced the loss (Radford et al., 2019). The bottleneck of this epoch was **capability**: pretrained models still required task-specific fine-tuning, and without it their performance was low. The hypothesis of 2020–2022 was that this limitation was not architectural and that scale alone would remove it.

The epoch opened with the quantification of that hypothesis, not with a model. In January 2020, Kaplan et al. (2020) measured how the loss depends on model size, data and compute, and derived from the measurements a rule for allocating a training budget (Section 4.2). GPT-3, published in May 2020, applied the rule at the largest scale to date with the architecture held fixed, and established in-context learning as a new usage regime (Section 4.3). The models of 2021–2022 followed the same rule until Chinchilla corrected it in March 2022 (Section 4.4). PaLM, published one week later, is the largest model of the original allocation and already contained most of the next epoch's architectural model recipe (Section 4.5). The chapter then covers the alignment procedure that turned the model into an assistant (Section 4.6). Its second half follows the parallel line that matters most for the later chapters: the efficient-attention methods (Section 4.7) and the first mixture-of-experts models (Section 4.8). Both were published alongside the scaling results, and neither was adopted at the time. Section 4.9 summarizes why, and Epoch III reverses the decision.

4.2 Scaling laws: the quantitative case for scale

Kaplan et al. (2020) provided the empirical basics for the epoch. They trained decoder-only language models spanning several orders of magnitude in size and measured the test loss $\mathcal{L}$ (cross-entropy in nats per token) as a function of three resources: the number of non-embedding parameters N, the dataset size D in tokens, and the training compute C. The central result is that, as long as the other two resources are not limiting, the loss follows a **power law** in each resource:

$$\mathcal{L}(N) = \left(\frac{N_c}{N}\right)^{\alpha_N}, \qquad \mathcal{L}(D) = \left(\frac{D_c}{D}\right)^{\alpha_D}, \qquad \mathcal{L}(C_{\min}) = \left(\frac{C_c}{C_{\min}}\right)^{\alpha_C}, \tag{6}$$

with fitted exponents $\alpha_N \approx 0.076$, $\alpha_D \approx 0.095$ and $\alpha_C \approx 0.050$. N_c, D_c and C_c are fitted scale constants, and $C_{\min}$ is the smallest compute at which a given loss is reached, i.e. compute spent on the most efficient combination of model size and training steps. A power law is a straight line on log-log axes, and the exponent is its slope: each tenfold increase in N multiplies the loss by $10^{-0.076} \approx 0.84$, a reduction of about 16 %; each tenfold increase in D reduces it by about 20 %, and each tenfold increase in compute along the efficient frontier by about 11 %. The gains per decade are small, but the trends show no saturation within the measured range, which for some of them spans more than seven orders of magnitude (Kaplan et al., 2020). This regularity made scale a plannable investment: the loss of a larger model could be predicted before training it.

The single-variable laws hold only when the other resources are abundant. The interaction of model size and data is captured by a joint fit,

$$\mathcal{L}(N, D) = \left[\left(\frac{N_c}{N} \right)^{\alpha_N/\alpha_D} + \frac{D_c}{D} \right]^{\alpha_D}, \tag{7}$$

which reduces to $\mathcal{L}(N)$ for $D \to \infty$ and to $\mathcal{L}(D)$ for $N \to \infty$. Its practical content is a condition against overfitting: for the data term to stay negligible, D must grow with N, but only sublinearly, roughly as $D \propto N^{0.74}$, so a model ten times larger needs about 5.5 times more data.

Two findings from these fits shaped the practice of the following years. (1) Model shape has little effect: at fixed N, varying the depth-to-dim ratio by a factor of 40 changes the loss only slightly, which justified leaving the architecture unchanged. (2) The fits determine how a fixed compute budget should be split between model size and training steps. Since the loss at fixed N also falls as power law in the number of steps, the budget can buy either a larger model or a longer run, and the compute-efficient optimum is

$$N \propto C_{\min}^{0.73}, \qquad D \propto C_{\min}^{0.27}, \tag{8}$$

i.e. a tenfold larger budget should go into about 5.4 times more parameters but only about 1.9 times more training tokens. The compute-efficient procedure is therefore to train very large models on comparatively little data and to stop well before convergence. This rule, published four months before GPT-3, determined how the models of 2020–2022 were built: hundred-billion-parameter models trained on a few hundred billion tokens. The Chinchilla correction (Section 4.4) later revises these two exponents.

4.3 GPT-3: scale as an interface

GPT-3 (Brown et al., 2020) is the first model built on these laws. It contains no architectural novelty: a family of eight models from 125M to 175B parameters, all with the GPT-2 architecture and one modification, alternating dense and locally banded spars attention layers, the pattern of the Sparse Transformer (Section 3.5). Depth, width and number of heads were chosen for computational efficiency and load balancing across GPUs, not for modeling reasons. The training budget follows the allocation of Section 4.2: the 175B model was trained on 300B tokens with a context length of 2048 token, i.e. far from convergence, and the paper reports that the power law of Kaplan et al. (2020) continues over two further orders of magnitude of compute with only small deviations (Brown et al., 2020).

The contribution is a new usage regime. Instead of being fine-tuned, GPT-3 receives a small number of demonstrations (typically 10 to 100) in its context window and continues the pattern, without any weight update. The authors called this **in-context learning** (ICL). It turned the task-as-text formulation of GPT-2 (Section 3.3) into measurable quantity: few-shot performance rises steeply with the model size, and the gap between zero-shot and few-shot performance widens with scale, i.e. larger models extract more form the demonstrations (Brown et al., 2020). The absolute results were uneven: 86.4 % on LAMBADA in the few-shot setting, but 71.8 on SuperGLUE against 89.0 for fine-tuned systems. The relevant result is the breadth rather than any single score: one frozen model, prompted, was competitive across a wide range of tasks simultaneously. The text interface anticipated in Epoch I now existed as a deployable artifact, and both products and alignment research reoriented around it.

4.4 The Chinchilla correction

Between 2020 and 2022 the allocation of Section 4.2 was applied without change: GPT-3 (175B parameters, 300B tokens) and Gopher (280B, 330B) were trained on roughly one to two tokens per parameter (Hoffmann et al., 2022). Hoffman et al. repeated the analysis of Kaplan et al. (2020) in March 2022 with over 400 models and one methodological correction: the earlier study had used one learning-rate schedule for runs of every length, which biased the fit. They

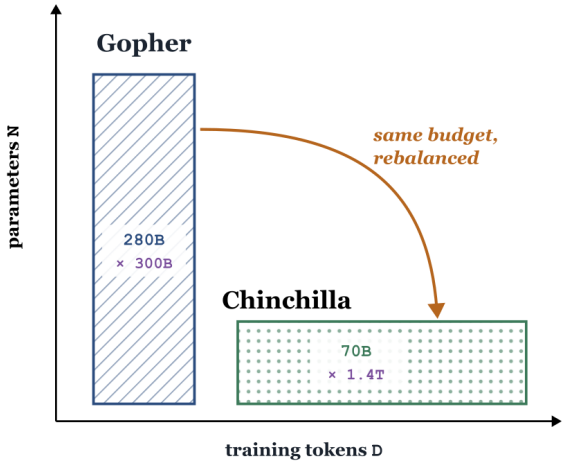

Figure 7: The Chinchilla correction: training compute is proportional to the area $N \cdot D$. At Gopher's budget, the rebalanced allocation (a quarter of the parameters on nearly five times the tokens) scores 67.6 against 60.0 on 5-shot MMLU (Hoffmann et al., 2022). The rectangles have equal area because the source states equal training FLOPs; their proportions are schematic.

arrive at a different allocation: **parameters and data should be scaled in equal proportion**, $N \propto C^{0.5}$ and $D \propto C^{0.5}$ in place of the exponents of Section 4.2, so doubling the model requires doubling the training tokens. (The frequently cited "about 20 tokens per parameter" is a reconstruction from their tables; the paper states only the proportionality.) The demonstration was Chinchilla: at the same training compute as the 280B-parameter Gopher, a 70B model trained on 1.4T tokens outperformed it on most benchmarks (67.6 % vs. 60.0 % on MMLU; Figure 7). The authors conclude that essentially every large model of the era was "significantly undertrained" (Hoffmann et al., 2022). One consequence extends beyond training economics and connects to the next epoch: the compute-optimal model is four times smaller, and a smaller model is cheaper to serve. A training-time result thus also reduced inference cost, the first instance of the trade-off between the two budgets that Epoch III makes explicit.

4.5 The peak of dense scaling: PaLM

PaLM (Chowdhery et al., 2023), published one week after Chinchilla, is the largest model trained under the allocation of Section 4.2: 540B dens parameters on 780B tokens, which the Chinchilla criterion classifies as heavily undertrained. It produced the strongest capability results of the epoch, including large gains on reasoning benchmarks when prompted to produce intermediate steps. for this survey its interest is architectural. PaLM combines SwiGLU activations (Shazeer, 2020), rotary position embeddings (RoPE) (Su et al., 2024), no bias terms and multi-query attention (MQA) (Shazeer, 2019), the cache-reducing method of Section 3.5, adopted here at flagship scale for decoding speed, with parallel block formulation that computes attention and FFN side by side for training throughput. Most components of the modern decoder recipe (Section 5.2) were therefore in use before the epoch that standardized them.

4.6 From model to assistant: RLHF

This section describes no architectural change: operator, block and cache are untouched, and only the weights change. It is included because the procedure changed what the artifact is, and because the deployment pressures of Epoch III follow from it.

A pretrained language model continues text, it does not answer requests. Ouyang et al. (2022) state the problem as a mismatch of objectives: predicting the next token of a web page is not the objective "follow the user's instructions helpfully and safely", and they call the pretraining objective *misaligned* with it. Scale does not close the gap, since "making language models bigger does not inherently make them better at following a user's intent" (Ouyang et al., 2022). GPT-3 needed demonstrations in its context window (Section 4.3), but an assistant should need only the instruction. InstructGPT, published in March 2022, closes the gap with the three-stage pipeline of Figure 8. The pipeline itself was not new: Stiennon et al. (2020) had applied the same three stages to summarization in 2020; InstructGPT applies them to the general prompt distribution of a deployed API.

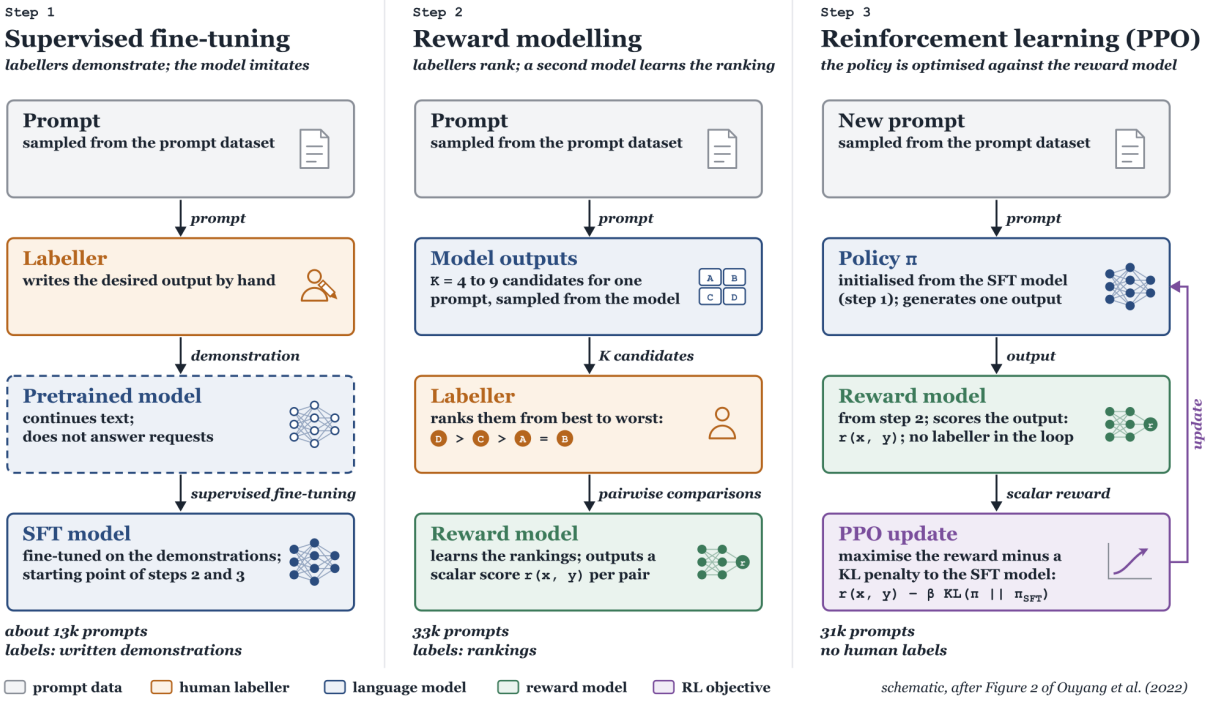

Figure 8: The three stages of InstructGPT, redrawn after Figure 2 of Ouyang et al. (2022). Stage 1: a labeler writes the desired output for a sampled prompt, and the pretrained model is fine-tuned on these demonstrations (about 13k prompts). Stage 2: for each prompt, $K = 4$ to 9 outputs of the current model are ranked by a labeler, and a reward model is trained on all pairwise comparisons to output one scalar score per prompt–output pair (33k prompts). Stage 3: the policy, initialized from the stage-1 model, generates an output for a new prompt, the reward model scores it, and PPO maximizes the score minus a KL penalty towards the stage-1 model (31k prompts, no human labels). Human labels enter only in stages 1 and 2. Schematic; the prompt counts are the training-split sizes of the source's Table 6.

Stage 1: supervised fine-tuning. The prompts came from users of an earlier version of the model in the OpenAI API playground. Their distribution is the everyday work of an assistant rather than the benchmarks of Section 3.6: 45.6 % open-ended generation, 12.4 % open question answering, 11.2 % brainstorming and 8.4 % chat, with summarization at 4.2 %. For each of about 13k prompts, one of about 40 contracted labelers writes the desired output by hand; for the prompt "List five ideas for how to regain enthusiasm for my career", the demonstration is such a list. The pretrained model is then fine-tuned on the (prompt, demonstration) pairs with the ordinary next-token objective of Section 3.3, so nothing changes except the data. The result, the **SFT model**, is already preferred by the labelers to the base model, prompted or not, and it is the starting point of both remaining stages (Ouyang et al., 2022).

Stage 2: reward modeling. The second stage replaces writing with judging. For each of 33k prompts, $K = 4$ to 9 outputs are sampled from the current model, and a labeler orders them from best to worst, ties allowed (D $>$ C $>$ A $=$ B in Figure 8). A ranking of K outputs contains $K(K-1)/2$ pairwise comparisons, 6 for $K = 4$ and 36 for $K = 9$, from a single labeling session. A separate **reward model** is trained on these pairs. It is the SFT model with its unembedding layer removed, so that it outputs one scalar $r_\theta(x, y)$ for a prompt x and an output y instead of a distribution over tokens, and its loss is a logistic loss on the score difference of each pair: the preferred output must score higher. One 6B-parameter reward model serves every policy size, including 175B, because training a 175B reward model proved unstable (Ouyang et al., 2022).

Stage 3: reinforcement learning. The policy starts as a copy of the SFT model. A prompt is sampled, the policy generates one output, the reward model scores it, and the policy is updated with proximal policy optimization (PPO) (Schulman et al., 2017). The environment is a bandit: one episode is one prompt and one response, with no further turns. The objective is

$$\text{objective}(\phi) = \mathbb{E}_{(x,y)\sim\mathcal{D}_{\pi_\phi^{\text{RL}}}}\Big[r_\theta(x,y) - \beta\log\frac{\pi_\phi^{\text{RL}}(y\mid x)}{\pi^{\text{SFT}}(y\mid x)}\Big] + \gamma\,\mathbb{E}_{x\sim\mathcal{D}_{\text{pretrain}}}\big[\log\pi_\phi^{\text{RL}}(x)\big], \tag{9}$$

where π_ϕ^{RL} is the policy, π^{SFT} the frozen stage-1 model and r_θ the reward model. The first term pulls the policy towards what the labelers preferred. The second is a per-token penalty on the divergence from the SFT model, added to "mitigate over-optimization of the reward model" (Ouyang et al., 2022): the reward model is a learned approximation that is reliable only near the outputs it was trained on, and without the penalty the policy drifts to outputs that score highly without being better. The third term mixes gradients of the pretraining objective into the update and is the answer to the alignment tax discussed below. The paper calls models with $\gamma = 0$ "PPO" and models with $\gamma > 0$ "PPO-ptx", and "InstructGPT" denotes the PPO-ptx models unless stated otherwise. This stage used 31k prompts and no human labels: humans enter the pipeline only in stages 1 and 2 (Ouyang et al., 2022).

The result changed the interpretation of parameter counts. On the authors' prompt distribution, labelers preferred the outputs of the aligned 1.3B model to those of the 175B base model, "despite having 100x fewer parameters", and at equal size the aligned model also won against the base model prompted with few-shot demonstrations, the interface of Section 4.3 (Ouyang et al., 2022). At less than 2 % of the pretraining compute (Ouyang et al., 2022; Brown et al., 2020), alignment outweighed two orders of magnitude of scale in judged usefulness. One example from the paper shows what the labelers were judging. Asked in French to write a short story about a frog that travels back in time to ancient Greece, GPT-3 175B continues with further instruction-like sentences of the form "write a story about . . . ", in French, and never writes the story; InstructGPT 175B writes it. Over 96 % of the fine-tuning data is English, so the behavior generalized beyond the distribution it was trained on (Ouyang et al., 2022).

Three qualifications apply. The results are preference rates under the authors' own labelers, about 40 contractors, mostly English-speaking, on the authors' own prompt distribution, not benchmark scores; the paper itself asks to whom the model is aligned. The procedure incurs an **alignment tax**: the PPO models regressed against GPT-3 on public NLP datasets, notably SQuAD, DROP, HellaSwag and French-to-English translation, and the pretraining term of the objective reduces these regressions without removing them. And the aligned model follows instructions even when they are harmful, which the authors name as its greatest limitation (Ouyang et al., 2022). Later work simplified the pipeline: direct preference optimization (Rafailov et al., 2023) removes the explicit reward model, and the reasoning models of Section 6.4 replace it with rule-checked rewards, but the three-stage form remains the conceptual reference. ChatGPT, released in November 2022 and trained with this procedure, closes the epoch (Section 4.9).

4.7 Early answers to the quadratic problem

While the mainline scaled, a parallel literature addressed the $O(n^2)$ cost of attention (Section 2.6). The wave ran concurrently with GPT-3: Longformer appeared in April 2020, Linformer and linear attention in June, BigBird in July and Performer in September of the same year. It is beast described as four patterns rather than as individual papers. Each answers the question of which entries of the $n \times n$ attention matrix can be left uncomputed.

- **Fixed sparsity.** Longformer (Beltagy et al., 2020) restricts attention to a sliding window a small number of task-designated global tokens ($O(n \cdot w)$). BigBird Zaheer et al. (2020) adds random connections and supplies a theoretical result: with a global token, sparse attention remains a universal approximator. In both, the pattern is a hyperparameter, fixed by the designer and identical for every input.
- **Low rank.** Linformer (Wang et al., 2020) observes that the attention matrix is approximately low-rank and projects keys and values along the sequence axis. The cost becomes linear, but the projection is tied to a fixed sequence length, and the paper does not address causal decoding.
- **Kernel approximation.** Performer (Choromanski et al., 2021) approximates the softmax kernel with random feature maps, which makes attention linear without a designed sparsity pattern.
- **Linear attention as recurrence.** Katharopoulos et al. (2020) replace the softmax with a factorizable similarity $\phi(q)^\top\phi k$ and reorder the computation: $(\phi(Q)\phi(K)^\top)V = \phi(Q)(\phi(K)^\top V)$. The left-hand side materializes the $n \times n$ matrix, the right-hand side maintains a $d_\phi \times d_h$ state of constant size that is updated once per token. A causally masked Transformer of this form is a recurrent network: it passes a summary from token to token as the recurrent models of Section 2.2 do, but with matrix-value state. The limitation is fundamental: the exact softmax corresponds to an infinite-dimensional ϕ, so every finite approximation loses information, which means it loses the ability to retrieve individual past tokens exactly.

A fifth approach came from outside the Transformer literature in October 2021. S4 (Gu et al., 2022) revives the classical **state space model**, a linear recurrence $x_t = \bar{A}x_{t-1} + \bar{B}u_t$ with an initialization designed for long-range memory, and

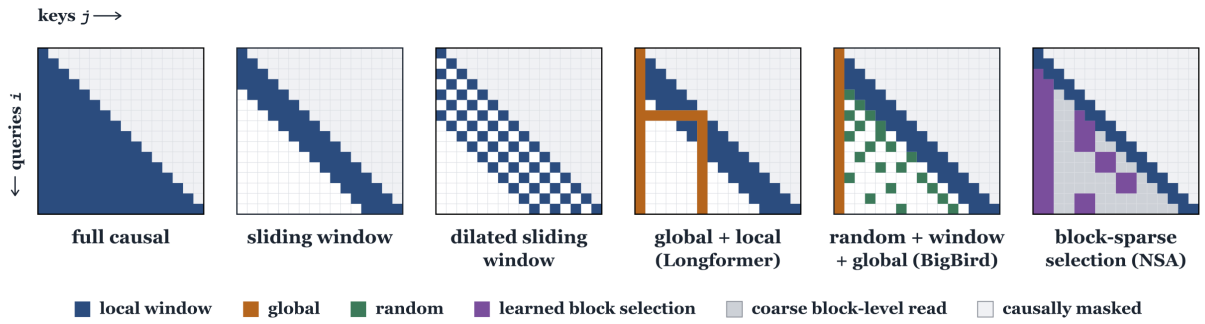

Figure 9: The attention-mask patterns of the first efficiency wave on grids of equal size: full causal attention compared with sliding-window, dilated-window, global-plus-local (Longformer), random-plus-window-plus-global (BigBird) and block-sparse structure. Each marked cell is a query–key pair that is computed. In all sparse patterns the connectivity is a hyperparameter, fixed before training and identical for every input; the trainable sparsity of Section 6.3 removes this restriction.

shows that it can be trained in parallel as convolution and run at inference as recurrence with fixed-size state. It solved long-range benchmarks on which all previous architectures had failed, including the 16k-token Path-X task, while on language modeling it still trailed Transformers by about one perplexity point by its authors' measurement. The line from S4 to Mamba (Section 5.5) is the development of this epoch with the largest consequences for later architectures.

None of these methods of Figure 9 became a default. Tay et al. (2022) list the reasons in their survey of the field: no variant had displaced standard attention, several cannot be causally asked, custom CUDA kernels made adoption costly, and "linear" methods were often slower than well-implemented quadratic attention at practical sequence lengths. The underlying reason is economic and frames the next epoch: in 2020—2022 the binding constraint was training compute at a context length of 2k tokens. Attention was a small share of that budget, and no deployment yet served conversations of millions of tokens. The first efficiency wave addressed a bottleneck that did not yet exist.

4.8 Early MoE: proven, then shelved

Mixture-of-experts (MoE) followed the same pattern of early demonstration and delayed adoption, and its three landmark papers appeared in parallel with the scaling results of Section 4.2-Section 4.4. GShard (Lepikhin et al., 2021) (June) scaled a translation model to 600B parameters by replacing the FFNs with banks of routed experts; Switch Transformer (Fedus et al., 2022) (January) simplified routing to top-1 and exceeded one trillion parameters, reporting up to 7× faster pretraining than T5-Base at the same computational budget; GLaM (Du et al., 2022) (December) trained a 1.2T-parameter decoder that activates about 8 % of its parameters per token and matched GPT-3 quality at a third of the training energy. The decoupling that MoE provides, a parameter count independent of per-token compute, was thus established. Adoption nevertheless stalled: routed training was unstable and required model-specific stabilization techniques, expert parallelism required unusual infrastructure, and the Chinchilla result of March 2022 had just shown that the cheapest improvement was more data through a dense model. MoE remained confined to a few laboratories until the inference economics of Epoch III made its trade-off attractive (Section 5.4).

4.9 Retrospect

The epoch began with a measurement and ended with a product. In January 2020, Kaplan et al. quantified the effect of scale, in November 2022, ChatGPT turned the scaled and aligned model into infrastructure. In between, scale worked: loss follows power laws regular enough to plan budgets against. A frozen decoder with demonstrations in its context replaced task-specific engineering. RLHF turned the model into an assistant. The decoder-only form, behind on the benchmarks of 2019, was now the default meaning of "language model". The properties identified in Section 3.6, supervision at every position and the text interface, were exactly those on which scale and in-context learning paid off.

The success of this epoch created the bottleneck of the next. Models were now very large and dense and, after ChatGPT, served hundreds of millions of users, with every generated token paying the full $O(n^2)$ attention cost and reading the entire KV cache of Section 2.5 from memory. The efficiency techniques that had not been adopted (sparse patterns, linear recurrences, state space models, routed experts and multi-query attention) remained available in the literature. In addition, assistants had acquired a property that received little attention at the time: prompted to reason step by step,

they generated more tokens to produce better answers (Wei et al., 2022), i.e. they spent compute at inference rather than in training. This is a third scaling axis, whose cost is the subject of Epoch IV. The change of direction was driven by economics rather than by a research result: once inference cost became the decisive metric, the methods of Section 4.7 and Section 4.8 were readopted within eighteen months. That development, and the open-weights wave that carried it, is Epoch III.

5 Epoch III: Efficiency and the Open Wave (2023–2024)

5.1 The setting

At the end of Epoch II the language model had become a product (Section 4.9). ChatGPT served hundreds of millions of users with a dense decoder, and every generated token paid the attention cost of Section 2.6 and re-read the KV cache of Section 2.5. The bottleneck of this epoch was therefore **inference cost**. In Epoch II the binding quantity had been training compute, spent once per model; from 2023 it was the cost per generated token, paid on every request. Almost every method in this chapter had been published between 2017 and 2021 and not adopted (Sections 4.7 and 4.8); it was adopted in 2023–2024, once the bottleneck it addresses had become the binding one.

Two publications in the first quarter of 2023 determined which models this survey can follow. In February, Meta released LLaMA (Touvron et al., 2023a), a family of decoder-only models from 7B to 65B parameters trained exclusively on publicly available data. The paper states its objective as the best possible performance at a given inference budget, in explicit contrast to the Chinchilla objective of Section 4.4, which "disregards the inference budget, which becomes critical when serving a language model at scale" (Touvron et al., 2023a). The models were therefore trained past the compute-optimal point, the 7B and 13B models on 1.0T tokens and the 33B and 65B models on 1.4T tokens, and the 7B model was still improving when its run ended. LLaMA-13B outperformed GPT-3 (175B) on most benchmarks, and LLaMA-65B was competitive with Chinchilla-70B and PaLM-540B (Touvron et al., 2023a). The weights were released to the research community; Llama 2 (Touvron et al., 2023b), published in July 2023, extended the release to commercial use. In March 2023, one month after LLaMA, OpenAI published the GPT-4 Technical Report, which states that "this report contains no further details about the architecture (including model size), hardware, training compute, dataset construction, training method, or similar" (OpenAI, 2023). From this point on the strongest models are undocumented, and the architecture history of this survey is, from 2023, the history of open models: the model families of Meta, Mistral and DeepSeek and the academic groups working on open weights.

The chapter is organized by component. Section 5.2 states the decoder recipe that LLaMA fixed and that every later open model inherits: rotary position embeddings, RMSNorm in pre-norm placement and a gated feed-forward network, each a substitution for one component of Section 2. Section 5.3 covers the methods that reduce the KV cache or the cost of reading it: exact attention with less memory traffic (FlashAttention), fewer key/value heads (GQA), a bounded attention window (Mistral), paged cache memory and context extension. Section 5.4 explains mixture-of-experts routing in full, since this is where it enters the mainline, and follows it from Mixtral to DeepSeek-V3, including the latent-attention cache of DeepSeek-V2. Section 5.5 does the same for fixed-state recurrence, from Mamba to the delta rule. Section 5.6 covers the first hybrids of recurrence and attention, Section 5.7 the small-model literature that applies the same economics at the other end of the scale, and Section 5.8 summarizes what the epoch established.

5.2 The modern decoder recipe

LLaMA's architecture section states its complete difference from the original Transformer as three substitutions, each taken from earlier work and none of them original to LLaMA (Touvron et al., 2023a): rotary position embeddings (Su et al., 2024) replace the additive positional encoding of Section 2.4, RMSNorm (Zhang and Sennrich, 2019) in pre-norm placement (Xiong et al., 2020) replaces the post-norm LayerNorm of Section 2.3, and SwiGLU (Shazeer, 2020) replaces the ReLU feed-forward network. PaLM had used two of them, SwiGLU and RoPE, at 540B one year earlier (Section 4.5); LLaMA is the model in which all three became the open standard. The paper defines none of them and contains no ablation of their contributions, so the evidence for each is in its source, summarized below. Table 5 lists the result against the block of Section 2.

Rotary position embedding. The original model adds a position vector to the token embedding once, at the beginning of the stack (Section 2.4), so position is a property of the representation and enters attention only indirectly. Su et al. (2024) instead apply position inside the attention operator, to the projected queries and keys of every block. The per-head dimension d_h is split into $d_h/2$ pairs of coordinates, and pair t of a query at position i is rotated by the angle $i\theta_t$, with $\theta_t = 10000^{-2(t-1)/d_h}$; the key at position j is rotated by $j\theta_t$ (Figure 10). Writing $R_{\Theta,i}$ for this block-diagonal rotation matrix, the attention score between a query q_i and a key k_j becomes

$$\langle R_{\Theta,i}\, q_i,\ R_{\Theta,j}\, k_j \rangle \ = \ q_i^{\top} R_{\Theta,\, j-i}\, k_j, \tag{10}$$

because rotations in the same plane compose by adding their angles, so the product of the two rotations depends only on the offset $j - i$: an absolute construction yields exact relative dependence. Three properties follow. Position is applied to queries and keys only, so it never enters the values or the residual stream. It is applied in every block rather than once at the input, and the geometrically spaced angles give each pair its own wavelength, the same design as the sinusoids of Section 2.4. Because the rotation is a fixed function of the position index, a pretrained model can be run at positions beyond its training length. How well that works, and how the angles can be rescaled to make it work better, is the subject of context extension in Section 5.3. ALiBi (Press et al., 2022), which removes position embeddings and adds a fixed, head-specific penalty proportional to the query-key distance to every attention score, is the main alternative at this site and was not adopted by the open mainline.

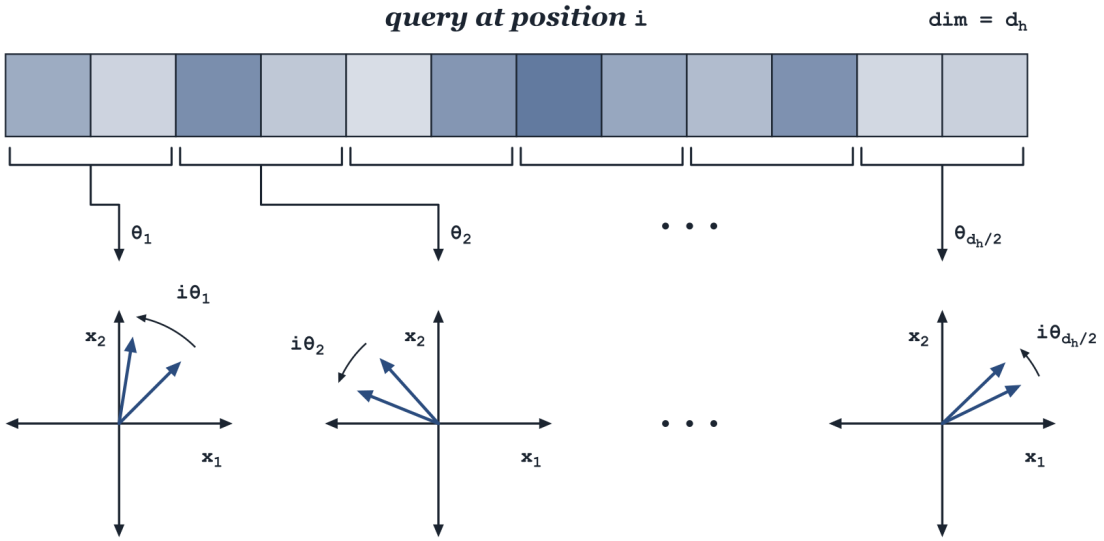

Figure 10: Rotary position embedding for one query at position i. The d_h coordinates of the projected query are read as $d_h/2$ pairs, and each pair is a point in its own plane. Pair t is rotated by the angle $i\theta_t$, with the angles θ_t falling geometrically from $\theta_1 = 1$ towards 10^{-4}; the key at position j is rotated by $j\theta_t$ in the same planes, so the score between the two depends only on the offset $j - i$. Adapted from the ICLR 2025 blog post "Positional Embeddings in Transformer Models" (Kumar et al., 2025).

Normalization: RMSNorm in pre-norm placement. Two changes at the normalization site are bundled. Layer normalization (Section 2.3) shifts the input vector $x \in \mathbb{R}^d$ by subtracting its mean μ and scales it using σ before applying the learned gain γ and bias β. Zhang and Sennrich (2019) hypothesize that the scaling invariance is the primary driver of training stability and discard the mean-centering operation as well as the bias β. Instead, Root Mean Square Normalization (RMSNorm) scales the vector solely by its root mean square $\mathrm{RMS}(x)$:

$$\mathrm{RMSNorm}(x) = \gamma \odot \frac{x}{\mathrm{RMS}(x)}, \qquad \mathrm{RMS}(x) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \epsilon} \tag{11}$$

where $\gamma \in \mathbb{R}^d$ denotes the learned gain parameter. The saving is one statistic per vector. The paper reports a training speedup of 7–9 % for a Transformer at equal translation quality (26.8 against 26.6 BLEU on newstest2014) and measures nothing at inference (Zhang and Sennrich, 2019). The second change is placement. The original block normalizes the sum of the residual stream and the sublayer output (post-norm), pre-norm normalizes the input of each sublayer instead and adds one final normalization after the last block (Figure 4). GPT-2 had adopted the placement empirically (Section 3.3); Xiong et al. (2020) supply the reason. At initialization, the gradient norm at the last block scales as $O(d\sqrt{\ln d})$ under post-norm but as $O(d\sqrt{\ln d / L})$ under pre-norm, which the authors give as the reason post-norm

Table 5: The Transformer block of 2017 (Section 2.3) and the modern decoder block, component by component. Everything not listed is unchanged.

Component	Transformer (2017)	Modern decoder (LLaMA line)	Origin of the change
Overall form	encoder–decoder	decoder-only	GPT-1, 2018 (Section 3.3)
Position	sinusoidal vector, added once at the input	RoPE: rotation of queries and keys in every block	Su et al. (2024)
Norm placement	post-norm (after the residual sum)	pre-norm (before each sublayer)	GPT-2, 2019; Xiong et al. (2020)
Normalizer	LayerNorm (mean and variance)	RMSNorm (re-scaling only)	Zhang and Sennrich (2019)
FFN	ReLU, two matrices, $d_{\text{ff}} = 4d$	SwiGLU gate, three matrices, inner width $\frac{2}{3} \cdot 4d$	Shazeer (2020)
KV heads	$h_{kv} = h$	GQA: $h_{kv} = 8 \ll h$	Shazeer (2019); Ainslie et al. (2023) (Section 5.3)
Biases	present	usually removed	PaLM, 2022 (Section 4.5)
Attention operator, residual stream, block interface	unchanged	unchanged	–

training requires a small initial learning rate and a warm-up stage that pre-norm can omit. Both changes leave the residual stream unbroken from the embedding to the output head.

Gated feed-forward network: SwiGLU. The feed-forward sublayer of Section 2.3 projects each position up to d_{ff} dimensions, applies a ReLU and projects back. Shazeer (2020) replaces the first projection and its rectifier by a gated linear unit: the input is projected twice, one projection passes through the Swish non-linearity $\text{Swish}(z) = z\,\sigma(z)$ and multiplies the other element-wise, and the product is projected back:

$$\text{FFN}_{\text{SwiGLU}}(x) = \big(\text{Swish}(xW_g) \otimes xW_1\big)\, W_2. \tag{12}$$

The sublayer now holds three matrices instead of two, and to keep the parameter count constant the inner width is reduced to two thirds of its former value, in LLaMA's notation $\frac{2}{3} \cdot 4d$ (Touvron et al., 2023a). At matched parameters and compute, the variant reaches a lower pretraining loss than the ReLU baseline (1.636 against 1.677 log-perplexity on the T5 span-corruption objective of Section 3.4). The evidence comes from one encoder–decoder setting, and the author offers no explanation for the improvement (Shazeer, 2020); the field adopted the substitution on this evidence.

The recipe held for the whole epoch. Llama 2 (July 2023) names exactly two departures from LLaMA, a trained context length doubled from 2,048 to 4,096 tokens and grouped-query attention at its two larger sizes (Section 5.3), and was trained on 2T tokens (Touvron et al., 2023b). Llama 3 (July 2024) lists SwiGLU, rotary embeddings and 8 key/value heads in one table for its 8B, 70B and 405B models, pretrained on 15.6T tokens, and describes its architecture as not deviating significantly from its predecessors (Llama Team, 2024). Mistral 7B and Mixtral (Sections 5.3 and 5.4) use the same components. None of the substitutions changes the attention operator of Section 2.2 or the structure of the block of Section 2.3: queries, keys and values are still formed by linear projections of one residual stream, compared by scaled dot products, normalized by a softmax and used to average the values, and the result is still followed by a position-wise sublayer. A reader comparing a LLaMA block with an original block finds component-for-component replacements of identical shape. The changes with structural consequences, treated in the rest of this chapter, all concern what surrounds the operator: how many keys and values are cached, how the operator is scheduled on the hardware, which positions it may see, and which of its sublayers a token actually reads.

5.3 Inference efficiency: the KV cache and memory traffic

Section 2.5 established that decoding is bound by memory bandwidth: every generated token re-reads the model weights and the entire KV cache, whose size per sequence is $2 \cdot n \cdot L \cdot h_{kv} \cdot d_h$ elements. In 2023 the trained context lengths grew from 2k to 32k tokens and beyond, serving batches grew with demand, and the cache became the binding resource, in the example of Section 2.6 several times the size of the weights. This section follows the factors of the cache expression rather than publication order, and gives the dates in the text: an exact algorithm that reduces the memory traffic of attention without changing its output (FlashAttention, May 2022), a reduction of h_{kv} (GQA, May 2023), a bound on n (sliding-window attention, October 2023), the systems layer that manages the cache (September 2023), and the extension of the context length itself.

Exact attention with less memory traffic: FlashAttention. Dao et al. (2022) diagnosed that attention on a GPU is limited not by arithmetic but by traffic between high-bandwidth memory (HBM) and the much smaller on-chip SRAM. The standard implementation writes the $n \times n$ score matrix to HBM, reads it back for the softmax, writes the probabilities and reads them again for the product with V. FlashAttention computes the same softmax attention exactly, in tiles of Q, K and V sized to fit in SRAM, and never materializes the full matrix. The softmax is computed incrementally: for each row two running statistics are carried, the maximum m and the normalizer ℓ, and the statistics of two tiles combine as

$$m = \max\left(m^{(1)}, m^{(2)}\right), \qquad \ell = e^{\,m^{(1)}-m}\,\ell^{(1)} + e^{\,m^{(2)}-m}\,\ell^{(2)}, \tag{13}$$

so the accumulated output is rescaled whenever a new tile raises the maximum, and the result equals the softmax over the full row. The backward pass stores no attention matrix either: it recomputes the tiles from Q, K, V and the two statistics. The number of HBM accesses falls from $\Theta(nd_h + n^2)$ to $\Theta(n^2 d_h^2/M)$, with M the SRAM size, and the authors prove that no exact attention algorithm does asymptotically better (Dao et al., 2022). The measurement that carries the argument is one attention layer of GPT-2 medium at sequence length 1,024 on an A100, forward and backward pass together: FlashAttention performs more floating-point operations (75.2 against 66.6 GFLOPs), moves nine times less data (4.4 against 40.3 GB) and runs in 7.3 ms against 41.7 ms (Dao et al., 2022). Two boundaries matter for the rest of the survey. FlashAttention is not an approximation and not a member of the sparse or linear family of Section 4.7; its operation count remains quadratic in n, and only the memory traffic is reduced. And it does not change the KV cache: its gains are on the compute-bound side of the prefill/decode asymmetry of Section 2.5, in training and prefill. It removed the memory ceiling on context length in training, and the long contexts that put pressure on the cache in 2023 were trained with it.

Fewer key/value heads: from MQA to GQA. The cheapest factor to reduce is h_{kv}. **Multi-query attention**(MQA) (Section 3.5) sets $h_{kv} = 1$: one key head and one value head serve all h query heads, and the cache shrinks by the factor h. PaLM used it at 540B (Section 4.5), but Ainslie et al. (2023) report in May 2023 that the endpoint costs quality, is unstable in training, and is wasteful to shard across the devices that serve a large model. **Grouped-query attention** (GQA) partitions the h query heads into h_{kv} groups and gives each group one key head and one value head; $h_{kv} = h$ is multi-head attention, $h_{kv} = 1$ is multi-query attention, and the cache is $2nLh_{kv}d_h$ for any choice in between (see Figure 12). The paper's second contribution is a conversion recipe: the key and value projections of the heads in each group are mean-pooled into one, and the converted checkpoint is trained further for 5 % of the original pretraining steps. On T5-XXL, the converted model with eight groups reaches the quality of the multi-head original (47.1 against 47.2 on the paper's average over seven tasks) at close to multi-query inference time (0.28 against 1.51 seconds per sample) (Ainslie et al., 2023). The stated reasons for adoption are cache size and serving. Llama 2 (July 2023) adopted GQA at 34B and 70B only, because at those sizes the cache limits the batch size, and chose the group count so that the key/value heads can be shared across the eight GPUs of one serving node (Touvron et al., 2023b). Mistral 7B (October 2023), Mixtral, Llama 3 (July 2024, at every size from 8B to 405B) and gpt-oss (Section 6.5) all settled on **8 key/value heads** while the number of query heads grows with the model (Jiang et al., 2023, 2024; Llama Team, 2024). GQA is the first widely adopted architectural change that no training-side argument supports: by its authors' measurements it costs a little quality, it saves nothing in training, and it was adopted for the size of the KV cache of Section 2.5 alone.

Bounding the window: Mistral 7B. Multi-query and grouped-query attention reduce the coefficient of n; sliding-window attention bounds n itself. Mistral 7B (Jiang et al., 2023), published in October 2023, restricts each position to the $W = 4096$ preceding positions, the pattern of Longformer (Section 4.7) and of the local attention that Vaswani et al. (2017) had proposed as future work (Section 2.6). Information still travels further than W, by one window per layer: after L layers a position can be influenced by tokens up to $L \cdot W$ positions back, which for Mistral's 32 layers the authors state as a theoretical span of about 131k tokens. The consequence for the cache is qualitative. Keys and values older than W are never read again, so they can be overwritten: the cache is a rolling buffer of W positions, in which position i is stored at index $i \bmod W$, and its size stops growing once $n > W$, at $2Lh_{kv}d_h \cdot W$ elements. The authors report an eightfold reduction of cache memory at a sequence length of 32k tokens without loss of quality, and Mistral 7B outperformed Llama 2 13B on every benchmark they evaluated (Jiang et al., 2023). The window is combined with GQA (32 query heads, 8 key/value heads), so the two reductions multiply. Xiao et al. (2024) had examined the complementary case two weeks earlier: a model trained with full attention at a fixed length that is given a window only at inference. Such a model fails as soon as the first tokens of the sequence leave the window, because it has learned to place a large share of its attention weight on those first tokens regardless of their content; the authors call them **attention sinks**. Keeping the keys and values of the first few tokens alongside the recent window restores stable generation, for Llama 2 and other models up to 4 million tokens, on a finite cache and without fine-tuning (Xiao et al., 2024).

The systems layer: paged cache memory. In September 2023, Kwon et al. (2023) treated the cache as a memory-management problem rather than an architectural one. Serving systems had reserved one contiguous region of memory per request, sized for the maximum length, so that a large share of the reserved cache memory was never used and could not be given to other requests. **PagedAttention** stores the cache in fixed-size pages allocated on demand, as an operating system pages virtual memory, which brings the waste to near zero and lets requests share pages for a common prefix. The serving system built on it, vLLM, raised throughput by 2–4× at equal latency over the best previous systems (Kwon et al., 2023) and underlies much of the open serving infrastructure since. The architecture is untouched; the reorganization shows what the field was engineering against in 2023.

Extending the context. The trained context length also moved: from 2,048 tokens in LLaMA to 4,096 in Llama 2 and 8,192 in Llama 3, whose supported window was then raised to 128k by continued pretraining on longer sequences, with the rotary base frequency raised from 10,000 to 500,000 (Llama Team, 2024). The cheaper route uses the structure of RoPE. Position enters only through the angles $i\theta_t$, so an existing model can be run at longer lengths by rescaling the angles until the longest wavelengths again span the window; position interpolation and YaRN (Peng et al., 2024) do this with a short fine-tuning stage instead of a retraining, and YaRN extends Llama 2 from 4k to 64k and 128k tokens by a per-dimension rescaling of the frequencies and a temperature on the attention scores. Every extended context, however obtained, is a longer n in the cache expression, and the methods of this section determine what that costs.

5.4 Mixture of experts: from Mixtral to DeepSeek-V3

The second family to be adopted was mixture of experts. Section 4.8 recorded its demonstration (GShard, Switch Transformer, GLaM) and the reasons adoption stalled. This section explains the mechanism in full, since this is where it enters the mainline, and follows its production form from Mixtral (January 2024) to DeepSeek-V3 (December 2024).

The mechanism. The feed-forward sublayer of Section 2.3 holds about two thirds of a dense block's parameters (more once GQA shrinks the key and value projections) and is applied identically to every position. An MoE layer keeps the position-wise form and replaces the single FFN by E parallel FFNs, the **experts**, together with a **router**: one trainable matrix W_r whose output, normalized by a softmax, scores every expert for the current token. Only the k highest-scoring experts are evaluated, and the layer output is their gate-weighted sum (Shazeer et al., 2017; Fedus et al., 2022):

$$y = \sum_{i=1}^{E} r(x)_i \, \mathrm{FFN}_i(x), \qquad r(x) = \mathrm{softmax}\big(\mathrm{TopK}(xW_r,\, k)\big), \tag{14}$$

where TopK keeps the k largest logits and sets the others to $-\infty$, so that their gates are exactly zero and the corresponding experts are not computed. Because the surviving gate values multiply the expert outputs, the router receives gradients through ordinary backpropagation. Shazeer et al. (2017) routed to $k = 4$ and $k = 2$ experts; Switch Transformer showed that $k = 1$ suffices and simplifies the implementation; the production models below use $k = 2$ (Mixtral) and $k = 6$ to 8 over smaller experts (DeepSeek). Figure 11 shows one such layer in place of the feed-forward sublayer of a decoder block.

The result is the decoupling that Section 4.8 stated informally, now with two quantities. The **total parameter count** P counts every weight in the checkpoint and determines the memory a deployed model occupies; the **activated parameter count** P_a counts the weights one token reads in a forward pass and determines its compute. In a dense model the two coincide. In an MoE layer P grows with E while P_a depends only on k, so a model can hold many times the parameters of a dense model at the same per-token cost. What the layer does not touch is as important: the experts are position-wise, so the attention sublayer, and with it every factor of the cache expression of Section 2.5, is unchanged. Routing over experts and the sequence-mixing operator are chosen independently, which is what allows the hybrids of Section 5.6 and Section 6 to combine them.

The standing cost is that routing does not stabilize itself. Shazeer et al. (2017) describe the failure: "the gating network tends to converge to a state where it always produces large weights for the same few experts. This imbalance is self-reinforcing, as the favored experts are trained more rapidly and thus are selected even more". Three measures became standard. An auxiliary **load-balancing loss** penalizes uneven routing; in the Switch form,

$$\mathcal{L}_{\mathrm{aux}} = \alpha \, E \sum_{i=1}^{E} f_i \, \bar{r}_i, \tag{15}$$

where f_i is the fraction of the batch's tokens routed to expert i and $\bar{r}_i$ the mean router probability it received; the product is minimized when both are uniform, and the gradient flows through $\bar{r}_i$ (Fedus et al., 2022). An **expert capacity**, a fixed number of tokens per expert and batch, keeps tensor shapes static on the accelerator; tokens beyond the capacity skip the layer and pass on through the residual connection. And the router's arithmetic is kept in full precision, because routing in bfloat16 diverges (Fedus et al., 2022). On a 256-expert model of Shazeer et al. (2017), switching the balancing terms off raises test perplexity from 35.6 to 39.8 and lets the busiest expert carry 17.8 times the mean load. MoE therefore buys compute efficiency with systems complexity, and whether the trade pays depends on whether a model is trained once or serves billions of tokens.

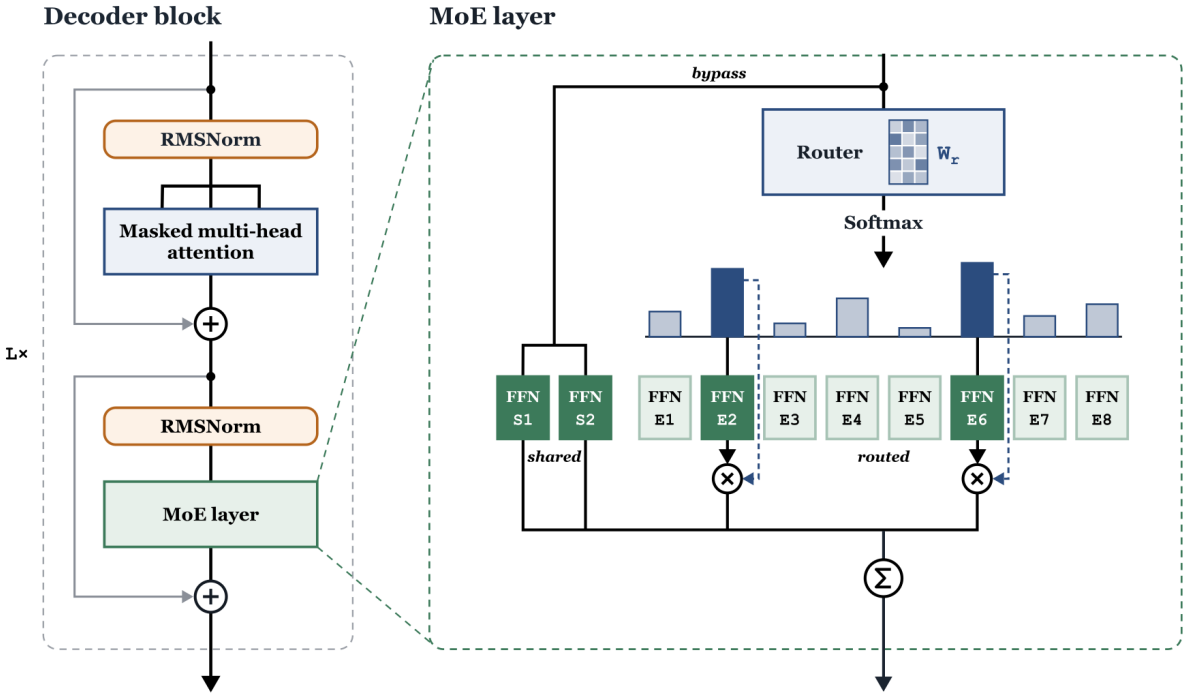

Figure 11: A mixture-of-experts layer in a pre-norm decoder block. Left: the MoE layer takes the place of the feed-forward sublayer; attention, normalization and residual stream are unchanged. Right: the router, one trainable matrix W_r followed by a softmax, scores the E routed experts (here $E = 8$) for the current token; only the k highest-scoring experts (here $k = 2$) are evaluated, and their outputs are multiplied by the gate values and summed. Shared experts (here $E_s = 2$), present in DeepSeekMoE and its successors, bypass the router and are applied to every token without a gate. All experts are held in memory and counted in the total parameter count P; the token reads only the $k + E_s$ filled ones, which is what the activated count P_a measures. Based on the illustrations in the blog post "A Visual Guide to Mixture of Experts (MoE)" (Grootendorst, 2024).

Mixtral: the open demonstration. Mixtral 8x7B (Jiang et al., 2024), released in January 2024 with open weights, is a Mistral 7B decoder in which every feed-forward sublayer is replaced by an MoE layer of 8 experts with top-2 routing. Each token has access to 47B parameters and uses 13B; the name is not a parameter count, since the attention and embedding parameters are shared by all experts and counted once. On the authors' benchmarks the model matches or outperforms Llama 2 70B, which reads more than five times the parameters per token (Jiang et al., 2024). Two findings in the report matter beyond the scores. The routing analysis finds no specialization by topic: expert choice on The Pile is aligned with syntax rather than domain, and consecutive tokens are routed to the same expert far more often than chance (27.9 % of consecutive pairs at the middle layer against a 12.5 % baseline), which the authors treat as a load-balancing concern for expert parallelism (Jiang et al., 2024). The word "expert" should therefore not be read in its everyday sense. And the report does not describe the training-side balancing measures of the founding papers at all; by 2024 they were treated as standard infrastructure rather than as a contribution.

DeepSeekMoE: fine-grained and shared experts. Three days after Mixtral, Dai et al. (2024) revised what an expert is. They name two problems of the conventional layout with 8 or 16 whole-FFN experts: each expert must hold many

unrelated kinds of knowledge, and knowledge that every token needs is learned redundantly by several experts. Two changes address them at constant parameters and compute. **Fine-grained segmentation** splits each expert into m smaller ones, cutting the inner width to $1/m$, and activates m times as many per token; the number of distinct expert combinations rises from $\binom{16}{2} = 120$ for 16 experts with top-2 routing to $\binom{64}{8} \approx 4.4 \times 10^9$ after splitting each into four (Dai et al., 2024). **Shared experts** are applied to every token and bypass the router (Figure 11), so that common knowledge is stored once; the number of routed experts activated per token is reduced by the same count. The layer becomes

$$y_t = u_t + \sum_{i=1}^{E_s} \mathrm{FFN}_i(u_t) + \sum_{i=E_s+1}^{E_s+E} g_{i,t}\,\mathrm{FFN}_i(u_t), \tag{16}$$

with u_t the attention output at position t, E_s shared experts without a gate, and gates $g_{i,t}$ that equal the router's softmax affinity for the k selected routed experts and zero otherwise. The ablation at 2B parameters isolates the second mechanism: replacing the shared expert by one more routed expert at the same compute raises the Pile loss from 1.808 to 2.414 (Dai et al., 2024). DeepSeekMoE 16B, with 2 shared and 64 routed experts of which 6 are activated, reads 2.8B of its 16.4B parameters per token and matches LLaMA2 7B at about 40 % of its compute per token (Dai et al., 2024).

DeepSeek-V2: multi-head latent attention. DeepSeek-V2 (DeepSeek-AI, 2024a), published in May 2024, scaled the layout to 236B total and 21B activated parameters (2 shared and 160 routed experts, 6 activated) and added the epoch's most consequential change to the cache. GQA reduces the number of cached heads; **multi-head latent attention (MLA)** changes the cached object. A down-projection compresses the block input at position t into one latent vector, and two up-projections reconstruct the keys and values of all heads from it:

$$c_t = W^{DKV} x_t \in \mathbb{R}^{d_c}, \qquad k_t^C = W^{UK} c_t, \qquad v_t^C = W^{UV} c_t, \qquad d_c \ll h\, d_h. \tag{17}$$

Only c_t is cached. The up-projections are fixed matrices, so at inference W^{UK} can be folded into the query projection and W^{UV} into the output projection, and the per-head keys and values are never formed: attention is computed directly against the latents. One obstacle shapes the design. RoPE (Section 5.2) multiplies each key by a rotation that depends on the key's position, and a position-dependent matrix between W^Q and W^{UK} cannot be folded away; applying RoPE to the reconstructed keys would force the keys of every earlier position to be recomputed at every decode step. MLA therefore carries position on a separate path: a small rotary key $k_t^R = \mathrm{RoPE}(W^{KR} x_t)$ of dimension d_r, shared by all heads, is cached next to the latent and concatenated to every head's content key, and the per-head queries receive a matching rotary part. The cache per token is $(d_c + d_r)\,L$ elements against $2hd_hL$ for multi-head attention (Figure 12). DeepSeek-V2 sets $d_c = 4d_h = 512$ and $d_r = d_h/2 = 64$, so that the cache equals that of GQA with 2.25 groups, while its 128 query heads keep their own content projections (DeepSeek-AI, 2024a). The controlled comparison separates MLA from GQA. In the authors' ablation on two MoE models, MLA matches or exceeds multi-head attention on their benchmarks, with one exception at the smaller scale, while retaining 14 % and 4 % of the cache; on a 7B dense model, GQA with eight groups and MQA both lose measurably to multi-head attention (41.2 and 37.9 against 45.2 on MMLU) (DeepSeek-AI, 2024a). The deployment figures, a 93.3 % smaller cache and 5.76 times the generation throughput of the dense DeepSeek 67B, bundle the architecture with FP8 weights and a cache quantized to about six bits, as the paper itself notes (DeepSeek-AI, 2024a).

DeepSeek-V3. DeepSeek-V3 (DeepSeek-AI, 2024b), published in December 2024, carried the template to 671B total and 37B activated parameters: 61 blocks, MLA with the same d_c and d_r, and MoE layers with 1 shared and 256 routed experts of which 8 are activated per token, pretrained on 14.8T tokens in 2.788 million H800 GPU-hours. Its one change to routing removes the auxiliary loss: a per-expert bias is added to the affinity scores for the top-k decision only, and is raised or lowered after every training step according to whether the expert was under- or overloaded, so that balance is maintained without a term in the loss that competes with the language-modeling objective (DeepSeek-AI, 2024b). The model became the reference design of the open frontier, which Section 6.5 follows to a trillion parameters.

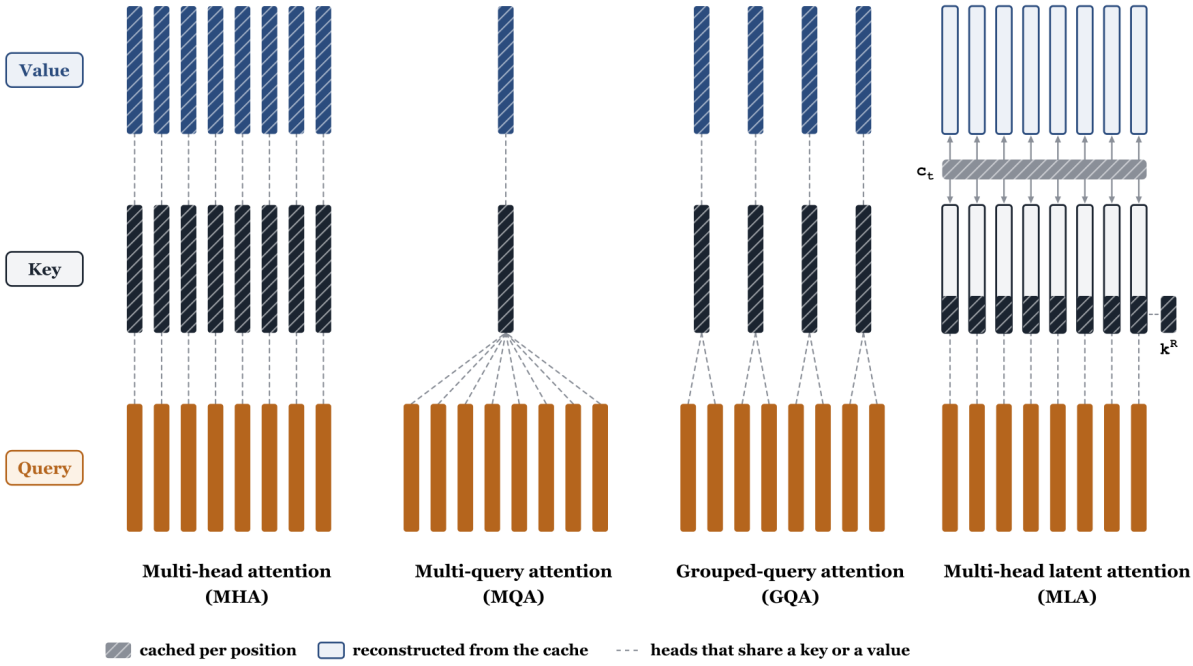

Figure 12: Query heads (orange), key heads (dark) and value heads (blue) under four attention designs, drawn for $h = 8$ query heads; dashed lines mark the heads that share a key or a value, hatched objects are what the KV cache holds per position, outlined objects are reconstructed from it at use. Multi-head attention stores a key and a value for each of the h heads ($2hd_hL$ elements for L blocks); multi-query attention shares one key/value head across all query heads ($2d_hL$); grouped-query attention stores one pair per group ($2h_{kv}d_hL$, here $h_{kv} = 4$); multi-head latent attention stores one latent vector c_t and one decoupled rotary key k^R ($(d_c + d_r)L$): the latent is up-projected into every key and value head, the rotary key is concatenated to every key head, and with the absorption described in Section 5.4 the per-head keys and values are never formed at all. Based on the illustrations in the blog post "Attention Variations—MQA vs GQA vs MHA vs MLA" (VerticalServe, 2024).

5.5 Fixed-state recurrence: from Mamba to the delta rule

The third family was the oldest. Section 4.7 left two fixed-state alternatives to attention: linear attention, whose running state $\phi(K)^\top V$ is updated by adding one outer product per token, and the state space model S4, a linear recurrence with fixed dynamics that can be trained as a convolution. Both have the same limitation: the update rule is the same for every token. Linear attention adds every token into the state with the same weight and never removes anything, and S4's transition matrices are fixed parameters, so neither can decide, on the basis of the current token, what to keep and what to forget. Full attention has no such problem because it keeps every key and value and re-reads them, which is exactly what makes it expensive. The development of 2023–2024 was to make the state update depend on the input. This section explains it as a sequence of three steps: selection (Mamba), the identification of state space models with linear attention (Mamba-2), and targeted overwriting (the delta rule). Table 6 places the resulting family against the cache-bearing designs of Sections 5.3 and 5.4.

Mamba: selective state spaces. In December 2023, Gu and Dao (2024) took the S4 recurrence as their starting point, written here with state s_i and input x_i (Section 4.7 used S4's own symbols):

$$s_i = \bar{A}_i\, s_{i-1} + \bar{B}_i\, x_i, \qquad y_i = C_i\, s_i, \qquad \bar{A}_i = \exp(\Delta_i A), \tag{18}$$

where the state has N_s dimensions per channel, A is a fixed diagonal matrix, and the input matrix B_i, the output matrix C_i and the step size Δ_i are functions of the current token x_i (linear projections, with a softplus for the step size). This input dependence is the contribution, **selectivity**. The step size acts as a gate: a large Δ_i drives $\bar{A}_i$ towards zero and resets the state to the current input, a small Δ_i keeps $\bar{A}_i$ near the identity and carries the state through unchanged; B_i

decides what of the token is written and C_i what is read out. The price is the convolutional training form of S4, which requires time-invariant dynamics. Mamba replaces it with a hardware-aware scan: the state is kept in on-chip SRAM rather than materialized in HBM, and the intermediate states are recomputed in the backward pass, the principle of FlashAttention (Section 5.3) applied to a recurrence. The ablations show what selectivity buys. On a synthetic copying task that requires content-dependent filtering, the non-selective layer scores 18.3 % and the selective one 97.0 %. On Pile perplexity at 350M parameters, enlarging the state from $N_s = 1$ to 16 changes almost nothing while the dynamics are fixed (9.88 to 9.81) and changes a great deal once they are selective (9.73 to 8.71) (Gu and Dao, 2024): state capacity is useful only if the model controls what is written into it. At 2.8B parameters, trained on 300B tokens, Mamba reaches an average zero-shot accuracy of 63.3 % against 61.7 % for Pythia-6.9B, the first attention-free model to match a Transformer of twice its size, and decodes with about five times the throughput of a Transformer of similar size, because without a KV cache it can serve much larger batches (Gu and Dao, 2024). Its inference state is a constant, N_s dimensions per channel and layer, independent of n. The authors' own limitation applies: nothing was evaluated above 2.8B parameters.

Mamba-2: state space models are linear attention. In May 2024, Dao and Gu (2024) showed that the two fixed-state families of Section 4.7 are one. Unrolling the recurrence writes the output as a matrix product $y = Mx$ whose entries are $M_{ji} = C_j^\top \bar{A}_j \cdots \bar{A}_{i+1} B_i$ for $j \geq i$. If the transition matrices are restricted to scalar multiples of the identity, $\bar{A}_i = a_i I$, each entry factors into a decay product and a dot product,

$$M_{ji} = \left(a_j \cdots a_{i+1}\right) C_j^\top B_i, \tag{19}$$

which is masked linear attention with C as the queries, B as the keys, x as the values and the cumulative decay as a data-dependent causal mask; the state dimension N_s is the feature dimension d_ϕ of Section 4.7. The recurrent form and the attention form are two ways of evaluating the same structured ("semiseparable") matrix, and the duality yields a training algorithm that computes the diagonal blocks with matrix multiplications on tensor cores and passes the state between blocks by a recurrence, 2–8× faster than Mamba's scan and faster than FlashAttention-2 beyond 2k tokens (Dao and Gu, 2024). The restriction to a scalar transition slightly reduces expressiveness, which the delta rule below partly restores. One experiment in the paper is the origin of the hybrid designs of Section 5.6 and Section 6. In a 48-layer sweep at 350M parameters, Pile perplexity is 8.60 with no attention layers, 8.26 with 6 and 8.50 with 24; at 2.7B parameters on 300B tokens, 6 attention layers among 64 reach 5.95 against 6.09 for the pure model and 6.13 for a Transformer of the same size (Dao and Gu, 2024). A small number of attention layers is not a compromise between two designs but an interior optimum, and the authors' hypothesis is that the recurrent layers act as a general sequence transformation while the attention layers act as retrieval. Section 5.6 returns to it.

The delta rule: targeted forgetting. A scalar decay forgets every stored association in the same proportion. The gated linear attention line took two further steps. GLA (December 2023) (Yang et al., 2024a) makes the decay a data-dependent gate on the matrix state of linear attention and supplies a chunkwise, hardware-efficient training algorithm for it; DeltaNet (June 2024) (Yang et al., 2024b) replaces the additive state update by the **delta rule** of classical associative memory and makes it trainable in parallel over the sequence through a compact representation of products of Householder matrices. Writing the state in the orientation of Section 4.7, $S_i \in \mathbb{R}^{d_k \times d_v}$ with $S_{i-1}^\top k_i$ the value currently stored under the key k_i, the delta rule first subtracts that value and then writes the new one. Gated DeltaNet (December 2024) (Yang et al., 2025) combines both operations:

$$S_i = \alpha_i \left(I - \beta_i \, k_i k_i^\top\right) S_{i-1} + \beta_i \, k_i v_i^\top, \tag{20}$$

where $\alpha_i \in (0, 1)$ is the decay, which clears stale context uniformly, and β_i the writing strength, which controls how far the old value under k_i is replaced by the new one. Setting $\alpha_i = 1$ recovers DeltaNet, and $\beta_i = 0$ a Mamba-2-style decay. At 1.3B parameters on 100B tokens the combination reaches a higher average zero-shot accuracy than Mamba-2, DeltaNet and the Transformer baseline (55.3 % against 54.9 %, 52.1 % and 52.3 %), and on a synthetic retrieval task at 4k tokens it scores 92.2 % against 56.2 % for decay alone and 18.6 % for the delta rule alone (Yang et al., 2025). The authors are equally clear about the cost: on a pass-key task at 8k tokens the ungated delta rule scores 98.8 % against 91.8 % for the gated one, because the gate discards information (Yang et al., 2025). The gates do not add capacity to a fixed state; they allocate it.

The family. Other lines arrived at the same design from other starting points: RWKV (May 2023) (Peng et al., 2023) from attention-free recurrent networks, with a learned per-channel decay; RetNet (July 2023) (Sun et al., 2023) from a decay-based retention operator with parallel, recurrent and chunkwise forms; Griffin (February 2024) (De et al., 2024) from a gated linear recurrence (RG-LRU), whose pure form, Hawk, outperforms Mamba on downstream tasks at equal

Table 6: The linear and recurrent family, 2020–2024. In every row the per-layer state is fixed before the sequence is seen and does not grow with n; the rows differ in how the state is written and cleared and in the form used for training.

Model	Year	State update	Forgetting	Training form
Linear attention (Katharopoulos et al., 2020)	2020	additive accumulation of $kv^\top$	none	parallel (associativity)
S4 (Gu et al., 2022)	2021	fixed linear dynamics	fixed decay, input-independent	convolution
RWKV (Peng et al., 2023)	2023	additive, per-channel decay	learned per channel, input-independent	parallel
RetNet (Sun et al., 2023)	2023	additive, exponential decay	fixed per head	parallel / recurrent / chunkwise
Mamba (Gu and Dao, 2024)	2023	input-selective dynamics (Δ_i, B_i, C_i)	learned, input-dependent	hardware-aware scan
GLA (Yang et al., 2024a)	2023	additive with gated decay	learned gate, input-dependent	chunkwise
Griffin / Hawk (De et al., 2024)	2024	gated linear recurrence (RG-LRU)	learned gate, input-dependent	parallel scan (Griffin adds local attention)
xLSTM (Beck et al., 2024)	2024	exponential gating, matrix-valued cell (mLSTM)	learned gate, input-dependent	parallel (mLSTM), recurrent (sLSTM)
Mamba-2 (Dao and Gu, 2024)	2024	scalar decay (SSD)	learned scalar, input-dependent	chunked matrix multiplication (duality)
DeltaNet (Yang et al., 2024b)	2024	delta rule (targeted overwrite)	replacement under the written key	chunkwise (Householder products)
Gated DeltaNet (Yang et al., 2025)	2024	decay and delta rule combined	both	chunkwise

size; and xLSTM (May 2024) (Beck et al., 2024) from the LSTM itself, with exponential gating and a matrix-valued cell. Table 6 lists them. The convergence is on requirements rather than on a winner: a state of fixed size, an input-dependent rule for writing to it and clearing it, and a training form that is not a serial loop. What none of them escapes is stated by the Gated DeltaNet authors themselves: a state of fixed dimension can hold only a bounded number of distinguishable key–value associations, and once a sequence contains more, retrieval degrades. Constant inference state is therefore bought with exact recall. Section 6 shows that every hybrid design of 2025 is a response to this limitation.

5.6 First hybrids

Fixed-state models have constant state and bounded recall; attention has exact recall and growing state. The composition question follows: how few attention layers does a recurrent stack need to recover exact recall? The Mamba-2 sweep of Section 5.5 gave a first answer, six among 64, and three models of 2024 gave it production form.

Griffin (February 2024) (De et al., 2024) interleaves its gated recurrence with local sliding-window attention rather than full attention, so its state is bounded by construction, and reaches the quality of Llama 2 with about six times fewer training tokens by its authors' measurement. Jamba (March 2024) (Lieber et al., 2024) is the first hybrid with full attention at production scale. Its unit is a block of eight layers in which attention and Mamba layers alternate at a ratio of 1:7, with an MoE sublayer (16 experts, top-2) in every second layer; the released model stacks four such blocks, for 52B total and 12B activated parameters, supports contexts of 256k tokens, and was designed to fit one 80 GB GPU, which it does even when processing texts of 140k tokens. The cache is paid for the attention minority only: at 256k tokens and 16-bit precision the authors tabulate 4 GB for Jamba against 32 GB for Mixtral and 128 GB for a 7B-class Llama 2 with full multi-head attention (Lieber et al., 2024). Their ablations at 1.3B and 7B parameters find virtually no difference between ratios of 1:3 and 1:7, and scaling the Mamba layers required an RMSNorm on their internal activations to remove loss spikes (Lieber et al., 2024). Samba (June 2024) (Ren et al., 2025) showed that the simplest composition suffices: Mamba layers, sliding-window attention and MLPs, at 3.8B parameters on 3.2T tokens, trained at 4k tokens, run at 256k tokens with perfect memory recall on the authors' test and at 1M tokens with still-improving perplexity, and 3.73 times the throughput of a GQA Transformer on 128k-token prompts (Ren et al., 2025). The design principle that Epoch IV adopts at scale is complete in these three models: a small minority of attention layers, retained to supply the exact recall a fixed state cannot, and everything else as cheap as possible. Which minority, and in which unit it is counted, is the disagreement Section 6.2 documents.

5.7 Small models: the other end of the scale

The inference economics that moved the frontier to MoE and hybrids also produced a literature in the opposite direction: models of one to fourteen billion parameters that run on one GPU or on a phone. It inverts the logic of Epoch II deliberately, and three strategies can be separated.

Data quality in place of scale. phi-1 (June 2023) (Gunasekar et al., 2023) trained a 1.3B model for four days on eight A100 GPUs on under 7B tokens of code selected or generated for "textbook quality" (web code filtered by a classifier, plus synthetic textbooks and exercises written by GPT-3.5) and reached 50.6 % pass@1 on HumanEval, above every model in its comparison table except GPT-4, including models trained on a hundred times the tokens (Gunasekar et al., 2023). When the training data is manufactured, contamination of the benchmark is the first hypothesis to exclude, and the paper's own test is a strong one: every training exercise similar to a HumanEval problem (by syntax-tree and embedding similarity) is removed and the model retrained from scratch. At the most aggressive threshold, which removes more than 40 % of the exercises, the score falls from 50.6 % to 45.1 %, so most of the result survives the test (Gunasekar et al., 2023). Neither the corpus nor the generation prompts were released, so the result cannot be reproduced from the paper. phi-4 (December 2024) (Abdin et al., 2024) scaled the recipe to 14B parameters and about 10T tokens, of which 40 % are synthetic, and validated on 78 contest problems published after its training cutoff. It also marks the boundary of the thesis: a synthetic-only mixture underperformed on knowledge-heavy benchmarks and hallucinated more (Abdin et al., 2024). Synthetic data substitutes for scale on reasoning-shaped tasks and not on knowledge-shaped ones.

Architecture under a memory budget. Below a billion parameters, the scaling-law finding that shape hardly matters (Section 4.2) no longer holds. MobileLLM (February 2024) (Liu et al., 2024) trained a grid of models of about 125M and 350M parameters that differ only in depth and width and found that at fixed size **deep and thin** wins: 30 blocks at width 512 beat 12 blocks at width 768 (44.8 against 43.9 on the paper's eight-task average), where earlier models of this size had used about 12 blocks. Three further choices follow from where the parameters sit. The input and output embeddings take more than 20 % of a 125M model (against 0.7 % of a 70B model), so sharing one matrix for both frees 11.8 % of the budget, which is reinvested in depth; grouped-query attention with a 4:1 head ratio serves here as a weight saving rather than a cache reduction; and two adjacent blocks can share their weights and be executed twice, so that a block loaded into the roughly 20 MB of on-chip SRAM of a phone is used twice per pass. The paper supports the argument with an on-device measurement: on an iPhone 13 the shared-weight model executes in 16.0 ms against 15.6 ms for the unshared model of the same size, while a 60-block model of the same executed depth without sharing takes 29.0 ms (Liu et al., 2024).

Overtraining on purpose. TinyLlama (January 2024) (Zhang et al., 2024) trained a 1.1B model of the Llama architecture on up to 3T tokens, and SmolLM2 (February 2025) (Ben Allal et al., 2025) a 1.7B model on about 11T tokens, both far beyond the Chinchilla allocation of Section 4.4. TinyLlama states LLaMA's inference-budget argument (Section 5.1) as its reason, and SmolLM2 describes its budget as deliberate overtraining relative to the compute-optimal allocation: training compute is spent once, inference cost is paid on every request, so at a fixed serving cost the best model is the smallest one that can be trained to the required quality, however many tokens that takes. The compute-optimal allocation was not wrong; it optimizes an objective that stopped being the objective when models became services.

5.8 Retrospect

Every mechanism in this chapter optimizes serving rather than training, and several pay a training-side price for it: GQA costs quality by its authors' measurement, MoE adds balancing losses and capacity limits, selective recurrence gives up the convolutional training form, and the small models spend far more training compute per parameter than the compute-optimal allocation prescribes. The bottleneck named in Section 5.1 explains the selection. It also explains the timing: multi-query attention (2019) was adopted in 2023, windowed attention (2019–2020) shipped in a production model in 2023, state space models (2021) reached the mainline in 2023, and mixture of experts (2017 in form, 2021 at scale) became a downloadable artifact in 2024. None of the ideas was new; the constraint that made them worth their cost was.

Two properties of the resulting design space carry into Epoch IV. The first is that the epoch established two decouplings and that they are orthogonal. Mixture of experts decouples the total parameter count from the compute per token and touches only the position-wise sublayer; fixed-state recurrence decouples the inference state from the sequence length and touches only the sequence-mixing sublayer. They modify disjoint components of the block of Section 2.3, so they compose without interference, which is why Jamba could combine them in 2024 and why every flagship design of

2025 is some combination of routed feed-forward sublayers, a cheap sequence-mixing majority and a small attention minority. The second is the cost of the second decoupling, stated in Section 5.5: a state of fixed size cannot store an unbounded number of exact associations, so constant inference state is bought with exact recall, and the attention layers retained in the hybrids of Section 5.6 are retained to buy it back. By the end of 2024 every component was mature, the recipe of Section 5.2, the reduced caches of Sections 5.3 and 5.4, routed experts and gated recurrences, and the question of how to compose them was open. Epoch IV is the field answering that question from several directions at once and, as Section 6.7 records, not agreeing on the answer.

6 Epoch IV: Convergence and New Axes (2025–2026)

6.1 The setting

By the end of 2024 every component of the later designs was available (Section 5.8): the decoder recipe of Section 5.2, routed feed-forward sublayers, a family of fixed-state sequence mixers, compressed and windowed caches, and the demonstration that a small number of attention layers restores the exact recall that a fixed state lacks (Section 5.6). Two pressures drove their combination. The first was the cost of context: windows of 100k to one million tokens moved from demonstrations into products. The second was new. From late 2024, reasoning models generated thousands of tokens per answer before giving it (Section 6.4), and every one of these tokens is a decode step that re-reads a growing KV cache (Section 2.5). Reasoning therefore multiplied the pressure on inference state rather than relieving it. This chapter covers the hybrid designs that combine the components (Section 6.2), the return of sparse attention in trainable form (Section 6.3), test-time compute as a third scaling axis (Section 6.4), the consolidation of the open frontier on one template (Section 6.5), and the paradigms that reject the premises of the mainline (Section 6.6). Section 6.7 records that the field converged on a design principle without converging on its parameter.

6.2 Hybrid convergence

Within about eighteen months, at least six independently developed model families adopted the same design: **a small minority of full-attention layers interleaved with linear or recurrent layers, usually combined with routed (MoE) feed-forward sublayers.** In order of publication:

- **MiniMax-01** (MiniMax, 2025a) (January 2025): 456B total and 45.9B activated parameters, the largest linear-attention deployment to date, with one softmax-attention block after every seven lightning-attention blocks; trained to a context of 1M tokens, with retrieval demonstrated at 4M.
- **Nemotron-H** (NVIDIA, 2025a) (April 2025): a dense hybrid without MoE, with attention in about 8 % of the layers, spread evenly (4 of 52 layers in the 8B model), and a reported long-context inference throughput of up to 3× that of Transformers of the same class.
- **Qwen3-Next** (Qwen Team, 2025) (September 2025): the first flagship open line with the pattern, built from repeating cells of three Gated DeltaNet layers and one attention layer, with an MoE sublayer after every mixer (80B total, 3B activated); the Qwen3.5 series inherits the layout.
- **Kimi Linear** (Kimi Team, 2025b) (October 2025): three Kimi Delta Attention layers per full-attention layer, where the full-attention layers use MLA (Section 5.4), so that both cache reductions of Epoch III are combined in one design (48B total, 3B activated).
- **Nemotron 3** (NVIDIA, 2025b) (December 2025): MoE-plus-Mamba stacks with "only a select few attention layers", discussed below.
- In production: IBM Granite 4.0 (October 2025) with a 9:1 ratio of Mamba-2 to attention layers (Soule and Bergmann, 2025), Falcon-H1 (May 2025) with attention and Mamba heads computed in parallel within one layer, and Tencent's 560B Hunyuan-TurboS (Falcon LLM Team, 2025; Tencent Hunyuan Team, 2025).

The shared reason. Two features are common to these systems beyond the layer ratio. First, the stated reason is the same wherever one is given, and it is the limitation recorded at the end of Epoch III (Section 5.5): a fixed state cannot perform exact retrieval. MiniMax concludes from its own experiments that pure linear attention is "unsuitable" for LLMs because it fails the recall that in-context learning requires; Kimi Linear names finite state capacity as the motivating bottleneck; and the Mamba-3 authors expect their layers to be used mainly inside hybrids for the same reason (MiniMax, 2025a; Kimi Team, 2025b; Lahoti et al., 2026). The attention minority is therefore not a compromise between two designs but a component added for recall. Second, several of these systems remove position encoding from the attention layers entirely (Jamba, Kimi Linear, Nemotron 3, Mamba-3): the recurrent layers carry position implicitly, and the attention layers operate as order-independent retrieval (Lieber et al., 2024; Kimi Team, 2025b; NVIDIA, 2025b).

Controlled evidence. The production pattern agrees with the controlled studies. In July 2025, Wang et al. (2025) trained 72 matched models across six linear-attention variants and interleaving ratios from 24:1 to 3:1. Language-modeling quality is nearly constant across the entire range, whereas recall rises with the attention share and saturates at about 3:1, where roughly a quarter of the layers provide near-Transformer recall at a several-fold smaller cache. Kimi Linear's ratio sweep shows the same flat region. Gemma 3 (Gemma Team, 2025) (March 2025) reports the analogous result on a neighboring axis, five local-attention layers per global one, with perplexity unchanged even at 7:1: a different mechanism with the same design response.

The allocation parameter. The principle converged; its parameter did not. Jamba interleaves at 1:7, MiniMax-01 at one block in eight, Nemotron-H at about 8 %, Kimi Linear at 3:1, and Wang et al. (2025) recommend 3:1 to 6:1. These figures are not in the same units: some count blocks, some sublayers, some percentages with different denominators, and Nemotron 3 states no ratio at all but publishes a per-layer placement string whose runs are not uniform. This scatter is evidence rather than noise. When independent groups agree on a principle but arrive at different values of its parameter, the constraint is not sharp: a broad interior region of the allocation space is adequate for quality, and each group settled where its deployment budget, operator choice and depth placed it. The sweeps predict exactly this: the allocation is weakly determined by quality and fixed by cost.

The cost of the minority. The retained attention layers dominate the cost they were introduced to reduce. At a context of one million tokens, MiniMax-01 reports that its softmax layers, one eighth of the stack, account for about 95 % of attention latency (MiniMax, 2025a), and Kimi Linear states the same bound analytically: the achievable speedup approaches the layer ratio and cannot exceed it. Hybrids reduce the growth of inference state; they do not remove it, and several of the savings reported in this literature are design targets rather than measurements.

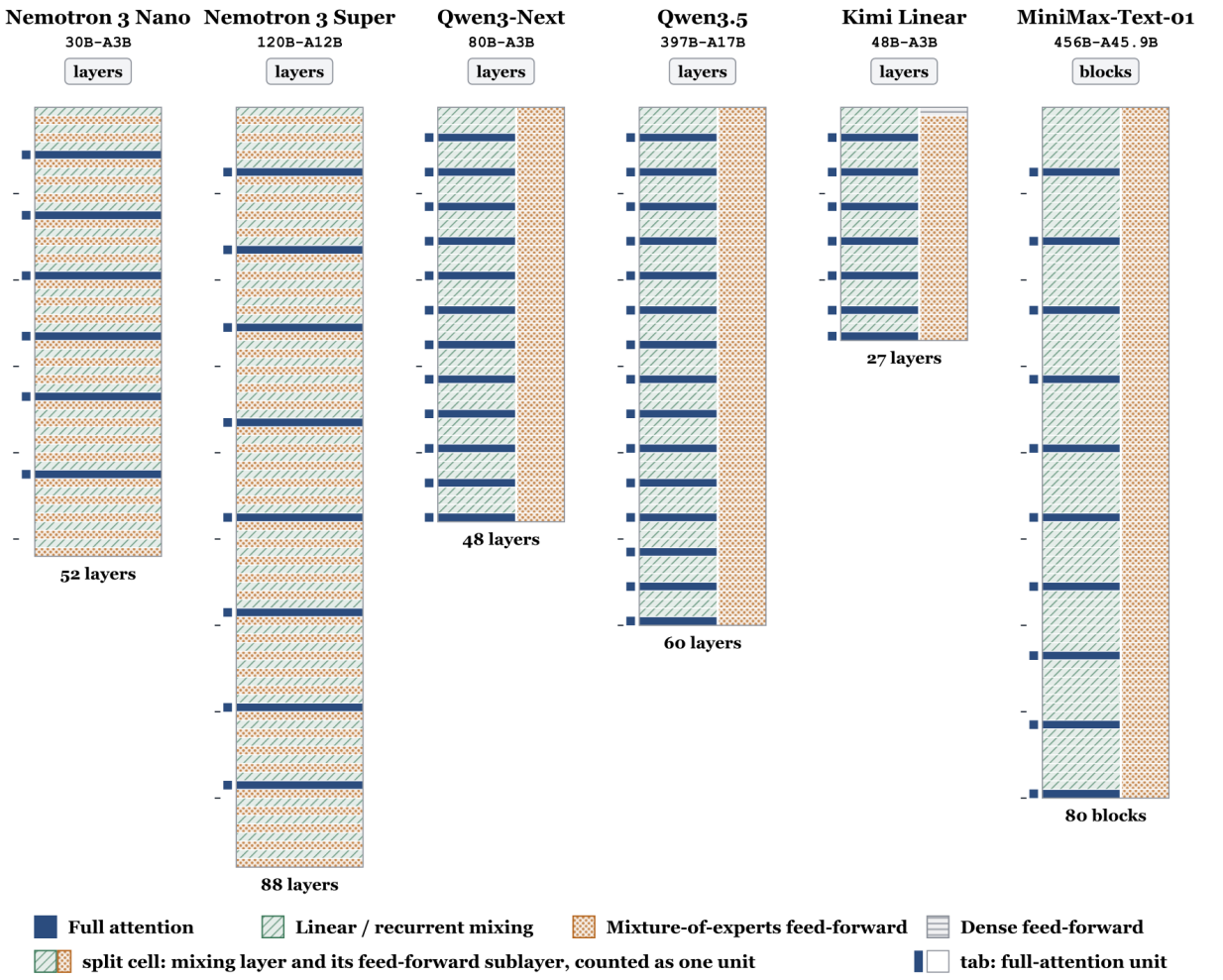

Figure 13: Layer allocation in six hybrid stacks, one column per model with layer 1 at the top, each drawn in the unit of its own source and not converted to a common unit: one cell per sublayer for Nemotron 3 (the released layout strings, character by character), one cell per mixing layer with its feed-forward sublayer for Qwen3-Next, Qwen3.5 and Kimi Linear, and one cell per block for MiniMax-Text-01. Full-attention cells, the only ones that write a KV cache, are marked against the linear or recurrent majority (Mamba-2, Gated DeltaNet, KDA, lightning attention). Kimi Linear and MiniMax-Text-01 follow their released configurations (Moonshot AI, 2025; MiniMax, 2025c): 27 layers with full attention at layers 4, 8, 12, 16, 20, 24 and 27 and a dense feed-forward in the first layer, and 80 blocks with softmax attention in every eighth block. The figure shows what a ratio cannot: where in the depth the retained attention sits, and that the spacing is not always the uniform period that a single number implies.

Table 7: The hybrid stacks. The allocation column reproduces each source's own statement and is not converted to a common unit, because the sources count different objects (blocks, sublayers, percentages with different denominators); "–" marks an allocation that no primary source states numerically.

System	Year	Params (total/act.)	Mixer majority	Attention minority (as stated)	MoE
Jamba (Lieber et al., 2024)	2024	52B / 12B	Mamba	1:7 attention:Mamba (paper's layers)	16 experts, top-2, every 2nd layer
Samba (Ren et al., 2025)	2024	3.8B dense	Mamba	interleaved sliding-window (no ratio)	–
MiniMax-01 (MiniMax, 2025a)	2025	456B / 45.9B	Lightning (linear)	1 softmax block per 7 (blocks)	32 experts, top-2
Nemotron-H (NVIDIA, 2025a)	2025	8B & 56B models, dense	Mamba-2	∼8 % of layers (sublayers; 4 of 52 at 8B)	–
Qwen3-Next (Qwen Team, 2025)	2025	80B / 3B	Gated DeltaNet	1 per 4 mixing layers (model card layout)	512 experts, 10 + 1 shared
Kimi Linear (Kimi Team, 2025b)	2025	48B / 3B	Kimi Delta Attention	3:1 KDA:**MLA** (blocks)	256 experts, 8 incl. shared
Nemotron 3 Nano (NVIDIA, 2025b)	2026	31.6B / 3.2B	Mamba-2	– (6 of 52 layers; no ratio stated)	128 experts, 6 active
Gemma 3 (neighboring axis) (Gemma Team, 2025)	2025	dense	*local* attention	5:1 local:global (layers)	–

6.3 Trainable sparsity and fusion

The sparse-attention line of 2019–2020 (Section 4.7) also returned, in a form shaped by the failure of the first wave. Native Sparse Attention (NSA) (Yuan et al., 2025) (February 2025) names two reasons for that failure: sparsity patterns imposed after pretraining diverge from the model's optimization trajectory, and token-level selection breaks the contiguous memory access that modern kernels require. NSA therefore trains the sparsity. For each query, three parallel branches attend over compressed block summaries, over the top-scoring blocks at full resolution, and over a sliding window, and their outputs are combined by learned gates:

$$o_t = \sum_{c \in \{\mathrm{cmp},\, \mathrm{slc},\, \mathrm{win}\}} g_t^c \cdot \mathrm{Attn}\big(q_t,\, \tilde{K}_t^c,\, \tilde{V}_t^c\big), \tag{21}$$

where $\tilde{K}_t^c$ and $\tilde{V}_t^c$ are the keys and values that branch c makes visible to position t and $g_t^c \in [0, 1]$ is the gate of that branch. The attention scores of the compression branch are reused as the block scores of the selection branch, so the pattern is learned end-to-end without an auxiliary objective, and the block sizes are chosen for the hardware from the start. In DeepSeek's matched-pretraining comparison the sparse model outperforms full attention on 7 of 9 general benchmarks and on the long-context suite, which the authors interpret as a regularization effect rather than only a cost reduction. MoBA (Lu et al., 2025) (February 2025) arrives at a similar design with a parameter-free block selector and notes that sliding windows and attention sinks are fixed special cases of a learned gate.

Sparse and linear attention are different purchases, and the field sorted them accordingly. Sparse attention still stores the full cache; it saves reads, not state. This is why the hybrid stacks of Section 6.2 chose linear mixers for their majority, and why Kimi Linear, after comparing the two directly, kept sparse attention out of its design (Kimi Team, 2025b). The separation may not be permanent: MiniCPM-SALA (MiniCPM Team, 2026) (2026) combines sparse and linear attention in one model, merging the two routes away from $O(n^2)$ that had developed separately since 2019.

6.4 Test-time compute: the third scaling axis

The mechanism had been documented since January 2022. Chain-of-thought prompting (Wei et al., 2022) showed that prompting a large model to produce intermediate reasoning steps raises its accuracy on multi-step problems (PaLM on GSM8K: 17.9 % to 56.9 %), with two findings that matter here. The gains appear only at scale, at roughly 100B parameters, while below about 10B the technique reduces accuracy; and the decisive ablation shows that tokens without content contribute nothing: prompting the model to emit meaningless filler of equal length leaves accuracy at the baseline. Generated tokens are the unit in which this axis is paid for, not the mechanism of the improvement.

OpenAI's o1 (OpenAI, 2024) (September 2024) turned the observation into a scaling axis: a model trained with reinforcement learning to produce long chains of thought before answering, with performance reported to improve both with more reinforcement learning and with more thinking time at inference. The architecture was not disclosed. The open replication followed within months and documented the procedure. DeepSeek-R1 (DeepSeek-AI, 2025) (January 2025) is the cleanest controlled experiment in this survey: starting from DeepSeek-V3-Base, reinforcement learning against rule-verifiable rewards alone (checked mathematical answers, compiled code; no learned reward model and no supervised reasoning data) raised AIME pass@1 from 15.6 % to 71.0 %, and the average response length grew over training without being optimized directly; the authors write that the model "naturally learns to solve reasoning tasks with more thinking time". In the final comparison, R1 scores 79.8 on AIME against 39.2 for DeepSeek-V3, two models whose architecture is identical in every respect. s1 (Muennighoff et al., 2025) (January 2025) demonstrates a minimal form of the same axis: after fine-tuning on one thousand examples, forcing the model to continue thinking changes its accuracy.

This independence from architecture is exactly why the axis belongs in an architecture survey. Reasoning tokens are spent entirely in decode, the memory-bound regime of Section 2.5. A long prompt is prefill: parallel, compute-bound and cheap per token. A long chain of thought is generated one token at a time, and each step re-reads a cache that the chain itself extends. The same sequence length is therefore more expensive when the model produces it than when the user supplies it, and reasoning models produce long sequences. The cache reductions of Epoch III (GQA, MLA, hybrids) thus gained value when models began to reason; the two axes of Epoch IV multiply rather than add. Their combination followed in June 2025: MiniMax-M1 (MiniMax, 2025b) is a reasoning model trained on the hybrid-linear MiniMax-01 base, long generation on a substrate with cheap state.

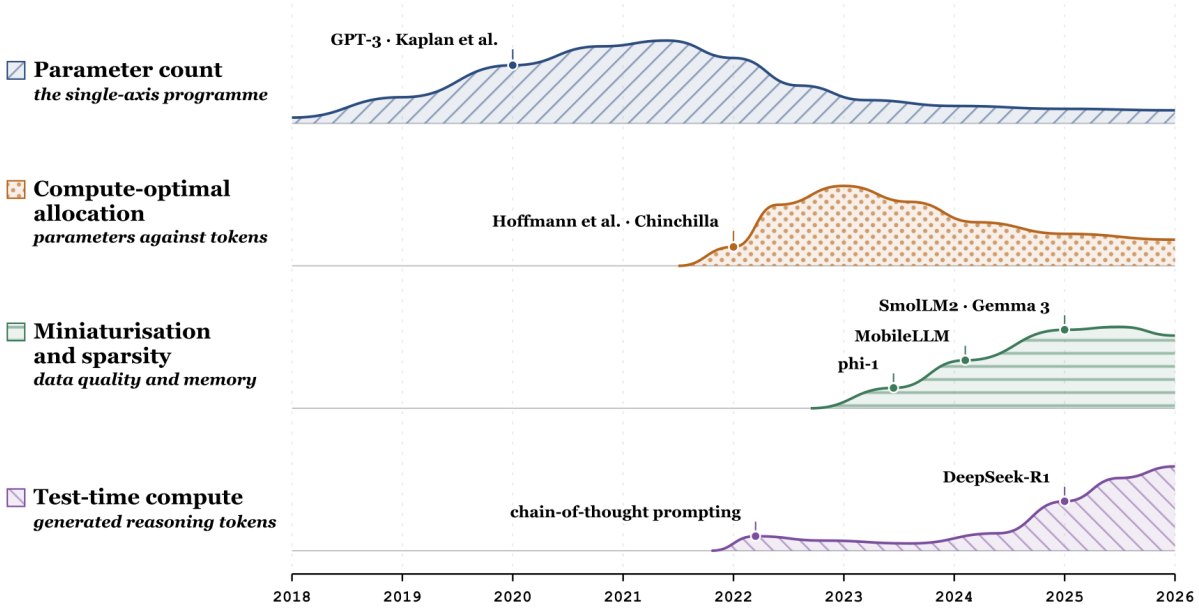

Figure 14: The scaling axis pursued by the field, by year, as this survey reads the literature. Each band is one route to capability, placed against its defining evidence: parameter count (GPT-3; Kaplan et al., 2020), the compute-optimal balance of parameters and tokens (Hoffmann et al., 2022), small models through data quality and deliberate overtraining (phi-1, MobileLLM, SmolLM2, Gemma 3), and test-time compute (chain-of-thought prompting, DeepSeek-R1). The height of a band in a given year shows how much of the field's effort that axis absorbed in the literature of that year, in the survey's own judgment: a qualitative reading, not a measured quantity, and the figure has no vertical scale. The axes are not commensurable (Section 7.1); what connects the newest to the others is that its cost falls entirely on the decode-time state of Section 2.5.

6.5 The open frontier at trillion scale

While architectures diversified below the frontier, the open frontier consolidated on one template. Kimi K2 (Kimi Team, 2025a) (July 2025) scaled the DeepSeek-V3 design (Section 5.4) to about one trillion total parameters at about 32B activated: MLA, hundreds of fine-grained experts, and a higher sparsity. GLM-4.5 (GLM-4.5 Team, 2025) (July 2025)

is a 355B MoE with a switchable reasoning mode. OpenAI's gpt-oss (OpenAI, 2025) (August 2025), the company's first open-weights architecture disclosure since GPT-2, is an MoE decoder with grouped-query attention (64 query heads, 8 key/value heads) and alternating full and windowed attention, i.e. the same recipe as the other open models. The dense frontier of Epoch II no longer exists; at the scale of 2025–2026, every documented frontier model is a sparse MoE with a compressed or partial cache.

6.6 Alternative paradigms

Alongside the mainline, several research programs reject premises that every mainline model shares. Each is included here for one reason: it removes a component that all mainline models have in common, and none has been demonstrated at mainline scale. They show that the mainline's treatment of inference state is a choice, not a necessity.

Diffusion language models remove left-to-right generation. **LLaDA** (Nie et al., 2025) (February 2025) trains a bidirectional mask predictor and generates by iteratively unmasking an entire response. A KV cache is impossible by construction, because every position's representation changes at every step, so cached state is replaced by full recomputation per step; whether this is cheaper depends on the ratio of denoising steps to generated tokens, a hyperparameter. **Mercury** (Inception Labs et al., 2025) (2025) made the speed argument commercially, with its fastest variant above 1,000 tokens per second on an H100 (1,109 for Mercury Coder Mini, 737 for the Small variant). **BitNet b1.58** (Ma et al., 2024) (February 2024) changes the precision: ternary weights $\{-1, 0, 1\}$ trained natively, with the prospect of order-of-magnitude savings in arithmetic, while the cache, the object this survey tracks, is unchanged. **Titans** (Behrouz et al., 2025) (December 2024) makes memory a quantity learned at inference: a small network whose weights are updated by gradient descent while the sequence is processed, with a surprise measure deciding what is written. **BLT** (Pagnoni et al., 2025) (December 2024) removes the tokenizer and learns dynamic byte patches whose compute adapts to the complexity of the input. Under this survey's question, what a model's inference state is, they give three mutually exclusive answers: shrink the weights around the state, abolish the state and recompute, or make the state a quantity the model learns while running. The mainline chose a fourth: keep the state and reduce its size.

6.7 Retrospect: convergence and dissent

What had converged by 2026 can be listed: decoder-only architectures everywhere; routed feed-forward sublayers at every scale that can afford the memory; the hybrid principle, a small full-attention minority retained for recall, stated in near-identical terms by independent groups; and the removal of position encodings from attention layers in several designs. The epoch also added a scaling axis that is independent of architecture and yet places its entire cost on the one object that architecture had spent years reducing.

What did not converge is at least as informative. The allocation parameter is stated in incompatible units or not at all. And the attention question itself reopened: MiniMax, the flagship of the linear-hybrid line, returned to full softmax attention in MiniMax-M2, citing quality gaps of efficient attention on multi-hop reasoning and long context together with the greater maturity of full-attention infrastructure (MiniMax, 2026), while Qwen's mainline adopted DeltaNet hybrids, DeepSeek trainable sparsity, and MiniCPM the fusion of sparse and linear attention. Four frontier groups gave four different answers to the same question at the same time. This survey reads the disagreement as underdetermination rather than confusion: at current scales and prices the design space does not single out an answer, so deployment economics rather than expressiveness selects each group's point. What is unanimous is the target. Every branch of the 2026 disagreement is a different strategy against the same quantity, the memory a model must hold and re-read per generated token. How the nine-year development arrived at this consensus, and what it leaves open, is the subject of Section 7.

7 Synthesis: Nine Years in Four Lines

The epoch chapters follow the chronology. This chapter reads the same material along four development lines that span the whole period and together answer the third research question of Section 1.1: in which direction the field has moved over nine years, as distinct from the direction any single year appeared to take.

7.1 The four lines

Line 1: the overall form was chosen once and not revisited. The choice between encoder-only, decoder-only and encoder–decoder models in 2018–2019 (Section 3) is the only point in the nine years at which the overall form of the language model was open. Every later development took place inside the causal decoder. As Section 3.6 showed, the form was not selected on the benchmark evidence of its time, which favored the bidirectional variants; its three

structural properties, supervision at every position, the text interface and cacheability, became decisive only under the scale of Epoch II and the deployment pressure of Epoch III. The methodological consequence is that the most consequential architecture comparison of the period was settled before the criteria that decided it existed.

Line 2: the core is unchanged, the periphery was rebuilt, and one quantity was reduced factor by factor. A comparison of an open decoder of 2026 with the original of 2017 shows the attention operator, the residual stream and the block interface unchanged; the durable changes are component-for-component substitutions (RoPE, RMSNorm, SwiGLU; Section 5.2) that were settled by 2023 and have not moved since. The sustained development concerns the KV cache of Section 2.5, whose size per sequence is $2 \cdot n \cdot L \cdot h_{kv} \cdot d_h$. Read in order, the architecture history after 2022 reduces this expression one factor at a time, from the cheapest intervention to the most expensive: first h_{kv} (MQA and GQA, a configuration change; Section 5.3), then the product $h_{kv}d_h$ (MLA, which changes the cached object; Section 5.4), then the operator that produces a cached object at all (linear and state-space mixing with constant state; Section 5.5), then L (hybrids, in which only a minority of layers keeps a cache; Sections 5.6 and 6.2), and finally the dependence on n itself (sliding windows and attention sinks; Section 5.3). Each site was addressed once the previous one was exhausted. None of this sequence is visible in a cost model stated in training FLOPs, which is why the efficiency wave of Epoch II, evaluated in those terms, was not adopted (Section 4.7), and why the same ideas were adopted once the binding constraint moved to decode-time memory (Section 5.8).

Two decouplings underlie this sequence, and their orthogonality structures the current design space. Mixture of experts separates the total parameter count from the compute per token and modifies only the position-wise FFN; linear and recurrent mixing separates the inference state from the sequence length and modifies only the sequence-mixing operator. Because they modify disjoint components of the block, they compose without additional machinery, which is what makes the hybrid MoE stacks of Epoch IV possible. Neither is free: the first is paid in memory footprint and routing infrastructure, the second in exact recall. The hybrid principle, a small full-attention minority retained to restore recall, is the field's current answer to the second cost.

Line 3: methods were adopted when their bottleneck arrived, not when they were published. The most frequent pattern in this survey is that nearly every mechanism of Epochs III and IV had been published years earlier and was not adopted at the time. Table 8 lists the cases.

Table 8: Publication and adoption of the main efficiency mechanisms. The dormancy is the interval between the first publication of a mechanism and its first adoption in a widely used model.

Mechanism	First published	Adopted	Dormancy
Multi-query attention	2019	GQA, 2023	4 years
Sparse attention patterns	2019–20	GPT-3 (partial); NSA, 2025	5–6 years
Sliding-window attention	Longformer, 2020	Mistral 7B, 2023	3 years
State-space recurrence	S4, 2021	Mamba, 2023; hybrids, 2025	2–4 years
Mixture of experts	2017 (form)	Mixtral, 2024	7 years
Chain of thought	2022	o1/R1, 2024–25	2 years

In every case the mechanism was evaluated against the constraint that was binding at the time and did not pay off under it; what changed was the constraint, not the mechanism. The corollary for research is that a negative result for an efficiency method is a statement about the cost structure at the time of evaluation, not about the method.

Line 4: scaling split into four axes. The single axis of Epoch II, more parameters, split into four that are not commensurable: larger models (parameter count), balanced allocation (the compute-optimal ratio of parameters to tokens), smaller models (data quality and deliberate overtraining once serving cost dominates), and longer generation (test-time compute). The newest axis connects to Line 2: a model that reasons spends its budget entirely in the decode regime, on a cache that its own chain of thought extends, so the axis that leaves the computational graph unchanged places its entire cost on the object that every architectural line has been reducing. This concentration of pressures on one quantity, more than any single design, characterizes the state of the field in 2026.

7.2 The big picture

Table 9 compares the architecture families qualitatively. Its cells are this survey's assessment, not measured values, because no two published figures in this literature were obtained under comparable conditions (Section 7.3).

Table 9: The architecture families compared qualitatively. The cells are this survey's assessment, not sourced numbers.

Family	Inference state	Exact recall	Status 2026	Representatives
Encoder-only	n/a (no generation)	–	retrieval/embeddings niche	BERT line
Encoder–decoder	cache (decoder only)	full	translation, niche seq2seq	T5 line
Dense decoder, full attention	cache, linear in n	full	quality/simplicity default	LLaMA, MiniMax-M2
+ compressed cache (GQA/MLA)	cache, reduced slope	∼full	universal at scale	Llama 3, DeepSeek-V3
+ windowed/sparse attention	bounded or selected cache	high (learned)	growing (long context)	Mistral, Gemma, NSA
Linear / SSM (pure)	constant	bounded by state	research; niche deployments	Mamba, RWKV, RetNet
Hybrid (minority attention)	cache for minority + constant	near-full	the 2025–26 convergence	Jamba, Qwen3-Next, Nemotron, Kimi Linear
MoE (orthogonal axis)	unchanged	unchanged	default at frontier	Mixtral, DeepSeek-V3, Kimi K2, gpt-oss
Alternative paradigms	none / weights / recompute	varies	demonstrations	LLaDA, BitNet, Titans

Figure 1 (Section 1) presents the same content chronologically: one lane each for the attention, recurrence, sparsity, MoE and scaling lines, with the mechanisms of Table 8 marked at their publication and at their adoption.

7.3 Open problems

The recall limit of fixed states. Whether the recall gap of linear mixers is an engineering problem or a fundamental limit is open. Theory provides one conditional lower bound: sufficiently sparse attention requires a number of layers polynomial in the sequence length to reproduce some behaviors that a single full-attention layer performs (Zaheer et al., 2020), and the delta-rule line shows that better state management narrows the gap without closing it (Section 5.5). Every hybrid embodies the assumption that the remaining gap is worth exactly a minority of attention layers; whether that minority can reach zero is not known.

Comparability. Nearly every number in this literature is self-reported under non-matched data, budgets, tokenizer and evaluation protocols, and the quantities that govern deployment are the least reported. Few model papers state their per-token inference state, although it is fully determined by the published configuration; decode latency is reported against baselines on different serving stacks; and negative results, the allocations and operators that were tried and abandoned, are almost never published. Every convergence claim, including those of this survey, is therefore a claim about what was published, not about what was learned.

The undocumented frontier. Since GPT-4 (Section 5.1), no closed frontier laboratory has published its architecture. Everything in this survey after early 2023 describes the open record. Whether the closed frontier follows it, as the conventional design of gpt-oss suggests (Section 6.5), or diverged years ago cannot be determined from public sources, and this bounds every statement of the form "the field converged".

The unsettled operator. The disagreement of 2026 (Section 6.7) is an open experiment in production: full attention, DeltaNet hybrids, trainable sparsity and sparse-linear fusion are deployed simultaneously by different frontier groups. The controlled evidence that would decide between them, ratio sweeps above one billion parameters, a factorial comparison of dense versus routed and full versus fixed-state mixing at matched budgets, and recall measured at long contexts, does not exist in public.

7.4 Outlook

This section states tendencies that follow from the documented pressures, not forecasts. The pressure that produced every line of this survey, the economics of decode, is increasing: reasoning models multiply the generated tokens, agentic workloads multiply the resident contexts, and both costs are paid in inference state. On current evidence the near term therefore brings more hybridization rather than less. The open question is how the attention minority develops: whether it shrinks towards zero through better state management, becomes cheaper per layer through MLA-style compression

or learned sparsity, or, as MiniMax-M2 shows, remains full attention while savings are sought elsewhere. Diffusion generation is the one alternative that removes the cache entirely; if serving economics ever favor recomputation over stored state, the central object of the mainline disappears.

Two regularities of the past nine years are the safest to extrapolate. In no year was the operator of 2017 displaced, and in every year the machinery around it was rebuilt. And methods in this field are not discarded; they are adopted when their constraint arrives. For evaluating the next architecture, this survey suggests the question it has applied to every design since Section 2: what does the design do to the state a model must hold per generated token? Over nine years, that question has separated the durable changes from the incidental ones.

8 Conclusion

This survey posed three questions about the architecture of large language models between 2017 and 2026 (Section 1.1). The answers, one paragraph each:

RQ1: what was developed, and how does it build on itself? One architecture, the Transformer of 2017, and nine years of variation around it. The variation concentrates on four sites: the overall form, settled in favor of the causal decoder by 2020. The peripheral block components, settled into the RoPE, RMSNorm and SwiGLU recipe by 2023 and unchanged since. The parameter-activation axis, dense until the MoE adoption of 2024 and routed at nearly every frontier scale since. And the sequence-mixing operator with its inference state, the one site of continuous and unfinished change, from full attention through compressed and windowed caches to selective recurrences and, at present, hybrid stacks that interleave the two. The building-on is literal: nearly every mechanism of Epochs III and IV revives a method published between 2017 and 2021 and adopted when its bottleneck arrived.

RQ2: which bottlenecks drove each wave, and at which trade-offs? Each epoch is the response to one binding constraint. (1) Trainability and transfer drove the exploration of 2018–2019, at the trade-off between bidirectional quality and generative generality. (2) Capability drove the scaling of 2020–2022, at the cost of training compute, corrected by the Chinchilla allocation. (3) Inference cost drove 2023–2024, whose characteristic trade is that GQA and its successors accepted lower quality and additional training-side complexity to reduce decode-time state, and that linear mixers gave up exact recall for constant state. (4) And the combined pressure of context length and machine reasoning drove 2025–2026, where the hybrid principle, a small full-attention minority that restores the recall fixed states lack, is the field's current answer. No decoupling was free, and the costs, recall gaps, routing infrastructure and latency-dominating attention minorities, are documented in the primary sources themselves.

RQ3: where has the field converged? On more than is usually noted and less than is usually claimed. Converged: the decoder-only form, the modern block recipe, routed feed-forward layers at scale, the hybrid principle together with its stated reason, recall. And, across all groups, the identification of per-token inference state as the decisive quantity. Not converged: the hybrid allocation parameter, which is stated in incompatible units or not at all, and, as of 2026, the attention operator itself, with major groups simultaneously deploying full attention, DeltaNet hybrids, trainable sparsity and sparse-linear fusion. This survey reads the disagreement as the signature of a design space that quality alone does not determine at current scales, in which deployment economics rather than expressiveness selects each group's design.

Two limits are part of the answer. (1) The account describes the documented, that is, the open record, because no closed frontier laboratory has published an architecture since early 2023. (2) Its comparisons are qualitative by necessity, because the published figures of this field are self-reported under conditions that, at most, no two papers share. Within these limits the nine-year development supports one summary. *The Transformer remained, everything around it was rebuilt, and the axis of that rebuilding, from the uncacheable encoder of 2018 to the token budgets of the reasoning models of 2026, is the memory a model must hold for each token it generates.*

Use of AI Tools

AI tools were used for the following two purposes in the preparation of this paper. (1) In the research, they assisted in finding and working through the literature. (2) In the writing, they assisted with the language: help with English, correcting spelling errors and improving sentence structure. I reviewed all AI-assisted output and take full responsibility for the content of this survey.

References

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J. Hewett, Mojan Javaheripi, Piero Kauffmann, James R. Lee, Yin Tat Lee, Yuanzhi Li, Weishung Liu, Caio C. T. Mendes, Anh Nguyen, Eric Price, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Xin Wang, Rachel Ward, Yue Wu, Dingli Yu, Cyril Zhang, and Yi Zhang. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024. URL https://arxiv.org/abs/2412.08905.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.298. URL https://aclanthology.org/2023.emnlp-main.298/.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. URL https://arxiv.org/abs/1607.06450.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015. URL https://arxiv.org/abs/1409.0473.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xLSTM: Extended long short-term memory. In *Advances in Neural Information Processing Systems*, volume 37, pages 107547–107603, 2024. doi: 10.52202/079017-3417. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/c2ce2f2701c10a2b2f2ea0bfa43cfaa3-Abstract-Conference.html.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. In *Advances in Neural Information Processing Systems*, volume 38, pages 125925–125962, 2025. doi: 10.52202/085713-3786. URL https://openreview.net/forum?id=8GjSf9Rh7Z.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020. URL https://arxiv.org/abs/2004.05150.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Agustín Piqueres Lajarín, Hynek Kydlíček, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Ben Burtenshaw, Clémentine Fourrier, Haojun Zhao, Hugo Larcher, Mathieu Morlon, Cyril Zakka, Colin Raffel, Leandro von Werra, and Thomas Wolf. SmolLM2: When smol goes big — data-centric training of a fully open small language model. In *Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=3JiCl2A14H.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. A survey on mixture of experts in large language models. *IEEE Transactions on Knowledge and Data Engineering*, 37(7):3896–3915, 2025. doi: 10.1109/TKDE.2025.3554028. URL https://doi.org/10.1109/TKDE.2025.3554028.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019. URL https://arxiv.org/abs/1904.10509.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with Performers. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Ua6zuk0WRH.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov,

Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. PaLM: Scaling language modeling with Pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023. URL https://jmlr.org/papers/v24/22-1144.html.

Damai Dai, Chengqi Deng, Chenggang Zhao, Runxin Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1280–1297. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.70. URL https://aclanthology.org/2024.acl-long.70/.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1285. URL https://aclanthology.org/P19-1285/.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 10041–10071. PMLR, 2024. URL https://proceedings.mlr.press/v235/dao24a.html.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pages 16344–16359, 2022. doi: 10.52202/068431-1189. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html.

Soham De, Samuel L. Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, Guillaume Desjardins, Arnaud Doucet, David Budden, Yee Whye Teh, Razvan Pascanu, Nando De Freitas, and Caglar Gulcehre. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024. URL https://arxiv.org/abs/2402.19427.

DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024a. URL https://arxiv.org/abs/2405.04434.

DeepSeek-AI. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024b. URL https://arxiv.org/abs/2412.19437.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. URL https://arxiv.org/abs/2501.12948.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423/.

Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V. Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. GLaM: Efficient scaling of language models with mixture-of-experts. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 5547–5569. PMLR, 2022. URL https://proceedings.mlr.press/v162/du22c.html.

Falcon LLM Team. Falcon-H1: A family of hybrid-head language models redefining efficiency and performance. *arXiv preprint arXiv:2507.22448*, 2025. URL https://arxiv.org/abs/2507.22448.

William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. URL https://jmlr.org/papers/v23/21-0998.html.

Gemma Team. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025. URL https://arxiv.org/abs/2503.19786.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.446. URL https://aclanthology.org/2021.emnlp-main.446/.

GLM-4.5 Team. GLM-4.5: Agentic, reasoning, and coding (ARC) foundation models. *arXiv preprint arXiv:2508.06471*, 2025. URL https://arxiv.org/abs/2508.06471.

Maarten Grootendorst. A visual guide to mixture of experts (MoE). Exploring Language Models (newsletter), 2024. URL https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mixture-of-experts. Accessed: 2026-09-15.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=uYLFoz1vlAC.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023. URL https://arxiv.org/abs/2306.11644.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, 2016. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 30016–30030, 2022. doi: 10.52202/068431-2176. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/c1e2faff6f588870935f114ebe04a3e5-Abstract-Conference.html.

Inception Labs, Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, Aditya Grover, and Volodymyr Kuleshov. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*, 2025. URL https://arxiv.org/abs/2506.17298.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023. URL https://arxiv.org/abs/2310.06825.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024. URL https://arxiv.org/abs/2401.04088.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. URL https://arxiv.org/abs/2001.08361.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5156–5165. PMLR, 2020. URL https://proceedings.mlr.press/v119/katharopoulos20a.html.

Kimi Team. Kimi K2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025a. URL https://arxiv.org/abs/2507.20534.

Kimi Team. Kimi Linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025b. URL https://arxiv.org/abs/2510.26692.

Abhinav Kumar, Adesh Gupta, Mansi Gupta, and Shivank Garg. Positional embeddings in transformer models: Evolution from text to vision domains. In *ICLR Blogposts*, 2025. URL https://iclr-blogposts.github.io/2025/blog/positional-embedding/.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626. Association for Computing Machinery, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Aakash Lahoti, Kevin Y. Li, Berlin Chen, Caitlin Wang, Aviv Bick, J. Zico Kolter, Tri Dao, and Albert Gu. Mamba-3: Improved sequence modeling using state space principles. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=HwCvaJOiCj.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, Omri Abend, Raz Alon, Tomer Asida, Amir Bergman, Roman Glozman, Michael Gokhman, Avshalom Manevich, Nir Ratner, Noam Rozen, Erez Schwartz, Mor Zusman, and Yoav Shoham. Jamba: A hybrid Transformer-Mamba language model. *arXiv preprint arXiv:2403.19887*, 2024. URL https://arxiv.org/abs/2403.19887.

Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, Liangzhen Lai, and Vikas Chandra. MobileLLM: Optimizing sub-billion parameter language models for on-device use cases. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 32431–32454. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24ce.html.

Llama Team. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, Yuxin Wu, Yutao Zhang, Zhilin Yang, Xinyu Zhou, Mingxing Zhang, and Jiezhong Qiu. MoBA: Mixture of block attention for long-context LLMs. In *Advances in Neural Information Processing Systems*, volume 38, pages 20267–20292, 2025. doi: 10.52202/085713-0602. URL https://openreview.net/forum?id=RlqYCpTu1P.

Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The era of 1-bit LLMs: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*, 2024. URL https://arxiv.org/abs/2402.17764.

MiniCPM Team. MiniCPM-SALA: Hybridizing sparse and linear attention for efficient long-context modeling. *arXiv preprint arXiv:2602.11761*, 2026. URL https://arxiv.org/abs/2602.11761.

MiniMax. MiniMax-01: Scaling foundation models with lightning attention. *arXiv preprint arXiv:2501.08313*, 2025a. URL https://arxiv.org/abs/2501.08313.

MiniMax. MiniMax-M1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025b. URL https://arxiv.org/abs/2506.13585.

MiniMax. MiniMax-Text-01: Model configuration file (config.json). Hugging Face model repository, 2025c. URL https://huggingface.co/MiniMaxAI/MiniMax-Text-01/blob/main/config.json. Accessed: 2026-09-15.

MiniMax. The MiniMax-M2 series: Mini activations unleashing max real-world intelligence. *arXiv preprint arXiv:2605.26494*, 2026. URL https://arxiv.org/abs/2605.26494.

Moonshot AI. Kimi-Linear-48B-A3B-Instruct: Model configuration file (config.json). Hugging Face model repository, 2025. URL https://huggingface.co/moonshotai/Kimi-Linear-48B-A3B-Instruct/blob/main/config.json. Accessed: 2026-09-15.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20275–20321. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1025. URL https://aclanthology.org/2025.emnlp-main.1025/.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In *Advances in Neural Information Processing Systems*, volume 38, pages 56354–56392, 2025. doi: 10.52202/085713-1689. URL https://openreview.net/forum?id=KnqiC0znVF.

NVIDIA. Nemotron-H: A family of accurate and efficient hybrid Mamba-Transformer models. *arXiv preprint arXiv:2504.03624*, 2025a. URL https://arxiv.org/abs/2504.03624.

NVIDIA. NVIDIA Nemotron 3: Efficient and open intelligence. *arXiv preprint arXiv:2512.20856*, 2025b. URL https://arxiv.org/abs/2512.20856.

OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. URL https://arxiv.org/abs/2303.08774.

OpenAI. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*, 2024. URL https://arxiv.org/abs/2412.16720.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025. URL https://arxiv.org/abs/2508.10925.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744, 2022. doi: 10.52202/068431-2011. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html.

Artidoro Pagnoni, Ram Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srinivasan Iyer. Byte latent transformer: Patches scale better than tokens. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9238–9258. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.453. URL https://aclanthology.org/2025.acl-long.453/.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Xingjian Du, Matteo Grella, Kranthi Kiran GV, Xuzheng He, Haowen Hou, Jiaju Lin, Przemysław Kazienko, Jan Kocoń, Jiaming Kong, Bartłomiej Koptyra, Hayden Lau, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Guangyu Song, Xiangru Tang, Bolun Wang, Johan S. Wind, Stanisław Woźniak, Ruichong Zhang, Zhenyuan Zhang, Qihang Zhao, Peng Zhou, Qinghua Zhou, Jian Zhu, and Rui-Jie Zhu. RWKV: Reinventing RNNs for the transformer era. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14048–14077. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-emnlp.936. URL https://aclanthology.org/2023.findings-emnlp.936/.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=wHBfxhZu1u.

Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163. Association for Computational Linguistics, 2017. doi: 10.18653/v1/E17-2025. URL https://aclanthology.org/E17-2025/.

Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=R8sQPpGCv0.

Haohao Qu, Liangbo Ning, Rui An, Wenqi Fan, Tyler Derr, Hui Liu, Xin Xu, and Qing Li. A survey of Mamba. *ACM Transactions on Intelligent Systems and Technology*, 17(5):1–43, 2026. doi: 10.1145/3819582. URL https://doi.org/10.1145/3819582.

Qwen Team. Qwen3-Next-80B-A3B-Instruct. Hugging Face model card, 2025. URL https://huggingface.co/Qwen/Qwen3-Next-80B-A3B-Instruct. Accessed: 2026-08-06.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018. URL https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pages 53728–53741, 2023. doi: 10.52202/075280-2338. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL https://jmlr.org/papers/v21/20-074.html.

Liliang Ren, Yang Liu, Yadong Lu, Yelong Shen, Chen Liang, and Weizhu Chen. Samba: Simple hybrid state space models for efficient unlimited context language modeling. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=bIlnpVM4bc.

Tanjil Hasan Sakib, Md. Tanzib Hosain, and Md. Kishor Morol. Small language models: Architectures, techniques, evaluation, problems and future adaptation. *arXiv preprint arXiv:2505.19529*, 2025. URL https://arxiv.org/abs/2505.19529.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725. Association for Computational Linguistics, 2016. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162/.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019. URL https://arxiv.org/abs/1911.02150.

Noam Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020. URL https://arxiv.org/abs/2002.05202.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Kate Soule and Dave Bergmann. IBM Granite 4.0: Hyper-efficient, high performance hybrid models for enterprise. IBM announcement, 2025. URL https://www.ibm.com/new/announcements/ibm-granite-4-0-hyper-efficient-high-performance-hybrid-models. Accessed: 2026-09-15.

Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. Learning to summarize from human feedback. In *Advances in Neural Information Processing Systems*, volume 33, pages 3008–3021, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/hash/1f89885d556929e98d3ef9b86448f951-Abstract.html.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. doi: 10.1016/j.neucom.2023.127063. URL https://doi.org/10.1016/j.neucom.2023.127063.

Weigao Sun, Jiaxi Hu, Yucheng Zhou, Jusen Du, Disen Lan, Kexin Wang, Tong Zhu, Xiaoye Qu, Yu Zhang, Xiaoyu Mo, Daizong Liu, Yuxuan Liang, Wenliang Chen, Guoqi Li, and Yu Cheng. Speed always wins: A survey on efficient architectures for large language models. *arXiv preprint arXiv:2508.09834*, 2025. URL https://arxiv.org/abs/2508.09834.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023. URL https://arxiv.org/abs/2307.08621.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, volume 27, 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/hash/5a18e133cbf9f257297f410bb7eca942-Abstract.html.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28, 2022. doi: 10.1145/3530811. URL https://doi.org/10.1145/3530811.

Tencent Hunyuan Team. Hunyuan-TurboS: Advancing large language models through Mamba-Transformer synergy and adaptive chain-of-thought. *arXiv preprint arXiv:2505.15431*, 2025. URL https://arxiv.org/abs/2505.15431.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and

Guillaume Lample. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a. URL https://arxiv.org/abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b. URL https://arxiv.org/abs/2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

VerticalServe. Attention variations—MQA vs GQA vs MHA vs MLA. VerticalServe Blogs (Medium), 2024. URL https://verticalserve.medium.com/group-query-attention-58283b337c65. Accessed: 2026-09-15.

Dustin Wang, Rui-Jie Zhu, Steven Abreu, Yong Shan, Taylor Kergan, Yuqi Pan, Yuhong Chou, Zheng Li, Jibin Wu, Ge Zhang, Wenhao Huang, and Jason Eshraghian. A systematic analysis of hybrid linear attention. *arXiv preprint arXiv:2507.06457*, 2025. URL https://arxiv.org/abs/2507.06457.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020. URL https://arxiv.org/abs/2006.04768.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837, 2022. doi: 10.52202/068431-1800. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10524–10533. PMLR, 2020. URL https://proceedings.mlr.press/v119/xiong20b.html.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 56501–56523. PMLR, 2024a. URL https://proceedings.mlr.press/v235/yang24ab.html.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems*, volume 37, pages 115491–115522, 2024b. doi: 10.52202/079017-3668. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/d13a3eae72366e61dfdc7eea82eeb685-Abstract-Conference.html.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=r8H7xhYPwz.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. XLNet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/hash/dc6a7e655d7e5840e66733e9ee67cc69-Abstract.html.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23078–23097. Association for

Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1126. URL https://aclanthology.org/2025.acl-long.1126/.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big Bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems*, volume 33, pages 17283–17297, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/hash/c8512d142a2d849725f31a9a7a361ab9-Abstract.html.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/hash/1e8a19426224ca89e83cef47f1e7f53b-Abstract.html.

Duzhen Zhang, Zhong-Zhi Li, Ming-Liang Zhang, Jiaxin Zhang, Zengyan Liu, Yuxuan Yao, Haotian Xu, Junhao Zheng, Xiuyi Chen, Yingying Zhang, Fei Yin, Jiahua Dong, Zhijiang Guo, Le Song, and Cheng-Lin Liu. From System 1 to System 2: A survey of reasoning large language models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 48(3):3335–3354, 2026. doi: 10.1109/TPAMI.2025.3637037. URL https://doi.org/10.1109/TPAMI.2025.3637037.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. TinyLlama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024. URL https://arxiv.org/abs/2401.02385.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Zican Dong, Yupeng Hou, Beichen Zhang, Yingqian Min, Junjie Zhang, Peiyu Liu, Xiaolei Wang, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Yiwen Hu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *Frontiers of Computer Science*, 20(12):2012627, 2026. doi: 10.1007/s11704-026-60308-3. URL https://doi.org/10.1007/s11704-026-60308-3.