CMPUT 654: Mathematical Foundations of Modern AI Systems Fall 2026

Lecture 1: Decoder transformers and autoregressive inference

September 1, 2026

Lecturer: Csaba Szepesvári *Note prepared by: Csaba Szepesvári*

Purpose of this note. This is an example of the expected scribe-note style: record what happened in the lecture, define the mathematical objects precisely, and distinguish the main calculation from explanatory remarks. It reconstructs and organizes the lecture's mathematical content.

Delivery note. Administrative material occupied a substantial part of the meeting. The lecture opened with the harness surrounding a language model and then zoomed in on tokenization and how a decoder transformer performs inference. Training was described verbally. The lecture also discussed the quadratic cost of attention, key–value caching, maximum context length, and limitations of attention on long contexts. No training objective, loss, or gradient equation was presented. The motivation for learning and the first learning-theoretic results were postponed to Lecture 2.

1.1 The model inside a harness

A deployed AI system is generally larger than one call to a neural network. A *harness* is the surrounding program that constructs the model's context, calls the model, interprets the result, optionally invokes tools, records the new observation, and decides whether to call the model again. It may provide instructions, conversation history, retrieved documents, external memory, or tool outputs. Schematically, one pass through the loop is

$$\boxed{\text{task state}} \xrightarrow{\text{harness}} \boxed{s \in \Sigma^*} \xrightarrow{\text{tokenizer } \tau} \boxed{x_{<t} \in [V]^*}, \tag{1.1}$$

followed by

$$\boxed{x_{<t}} \xrightarrow{\text{model}} \boxed{p_\theta(\,\cdot \mid x_{<t})} \xrightarrow{\text{decoding}} \boxed{\text{next token or action}}. \tag{1.2}$$

The *prompt* is the initial input supplied to a model call. After tokenization, the prompt tokens form the initial prefix; generated tokens then extend that prefix. Output tokens may be detokenized into text or parsed as a tool call. The resulting observation updates the task state, and the loop may begin again. A harness can use several model calls before returning an answer. The behavior of the complete system can therefore depend on the tokenizer, model parameters, decoding rule, maintained state, tools, and control loop. The rest of this note expands the tokenizer and model boxes.

1.2 Tokenization and the modeled sequence

The word *tokenization* comes from the compiler pipeline. A compiler's lexer maps a character stream to tokens such as identifiers, numerals, and operators using a language specification. An LLM tokenizer has the same broad string-to-sequence interface. Corpus data normally determines its vocabulary and segmentation rules; a programming-language specification determines a compiler's lexer.

Let Σ be a finite base alphabet and let

$$\mathcal{V} = \{v_1, \dots, v_V\} \subset \Sigma^+ \tag{1.3}$$

be a vocabulary of nonempty strings. A lossless tokenizer maps

$$\tau : \Sigma^* \longrightarrow [V]^*, \qquad \tau(s) = (x_1, \dots, x_T), \qquad s = v_{x_1} v_{x_2} \cdots v_{x_T}. \tag{1.4}$$

Detokenization concatenates the vocabulary strings. Because every vocabulary item is nonempty,

$$|s| = \sum_{t=1}^{T} |v_{x_t}| \geq T. \tag{1.5}$$

Thus tokenization, understood as segmentation over a fixed base alphabet, cannot increase sequence length and decreases it whenever at least one chosen token contains more than one base symbol. Normalization, conversion from Unicode characters to bytes, and the addition of special tokens may change the sequence before or after this segmentation; those are separate operations.

Practical tokenizers may also normalize text, first split it into coarse pieces, or begin from bytes to guarantee coverage. Some tokens coincide with useful linguistic units; corpus-based construction also produces fragments whose boundaries lack semantic significance. Appendix A gives optional background on how tokenizers may be constructed, added after the lecture.

Once the tokenizer is fixed, a stochastic language or sequence model specifies a stochastic process $(X_t)_{t \geq 1}$ whose state space is $[V]$. An autoregressive model gives a conditional distribution for the next token after every prefix. For a fixed token sequence,

$$p_\theta(x_{1:T}) = \prod_{t=1}^{T} p_\theta(x_t \mid x_{<t}). \tag{1.6}$$

A beginning-of-sequence convention initializes the process; an end-of-sequence token can represent variable-length strings.

1.3 The inference problem

Given a token prefix $x_{<t} = (x_0, \dots, x_{t-1})$, a language model returns a categorical distribution for the next token:

$$x_{<t} \longmapsto p_\theta(\cdot \mid x_{<t}). \tag{1.7}$$

An inference algorithm selects x_t from this distribution, appends it to the prefix, and repeats. Thus the model and the rule used to decode from it are distinct objects.

Orientation convention. On the board, the sequence was drawn from left to right:

$$\boxed{x_0} \quad \boxed{x_1} \quad \cdots \quad \boxed{x_{T-1}}. \tag{1.8}$$

In the matrix equations below, each position is instead represented by one *row*, and positions are stacked from top to bottom. Columns hold vocabulary coordinates, model features, head features, or source positions, depending on the matrix. Board orientation and algebraic storage therefore use different axes. We state what the rows and columns index whenever a new kind of matrix appears.

Throughout the course, every vector is a column vector. When a matrix stores one vector at each position, row i contains the transpose of that vector; thus $h_i = (H_{i,:})^\top \in \mathbb{R}^d$ for $H \in \mathbb{R}^{T \times d}$. We

write $\mathbf{1}_n \in \mathbb{R}^n$ for the all-ones column vector. Whenever a matrix calculation needs a row-shaped quantity, we write it explicitly as the transpose of a column vector.

Let the vocabulary have size V, and identify token $x_t \in [V]$ with the standard basis vector $e_{x_t} \in \{0,1\}^V$. Include a beginning-of-sequence token x_0. For T prediction positions, define $Z \in \{0,1\}^{T\times V}$ by

$$Z_{t,:} = e_{x_{t-1}}^\top, \qquad t = 1, \dots, T. \tag{1.9}$$

Rows of Z index sequence positions and columns index vocabulary items: $Z_{t,v} = 1$ precisely when $x_{t-1} = v$. With an embedding matrix $E \in \mathbb{R}^{V\times d}$ and additive absolute-position vectors $P \in \mathbb{R}^{T\times d}$, the input is

$$H^0 = ZE + P, \qquad h_t^0 = (H_{t,:}^0)^\top = E^\top e_{x_{t-1}} + P_{t,:}^\top \in \mathbb{R}^d. \tag{1.10}$$

Rows of E index vocabulary items and columns index the d model features; the embedding vector of token v is $E^\top e_v = (E_{v,:})^\top \in \mathbb{R}^d$. Rows of P and H^0 index positions, and store the transposes of position and hidden vectors. The one-hot notation makes the map explicit; an implementation performs a row lookup in E.

The transformer forward calculation can be summarized by the following block diagram:

$$\boxed{x_{<t}} \longrightarrow \boxed{H^0 = ZE + P} \longrightarrow \boxed{H^1 \longrightarrow \cdots \longrightarrow H^L} \longrightarrow \boxed{h_t^L} \longrightarrow \boxed{z_t} \longrightarrow \boxed{p_t}. \tag{1.11}$$

Here $H^0, \dots, H^L \in \mathbb{R}^{T\times d}$ are activation matrices, $h_t^L = (H_{t,:}^L)^\top \in \mathbb{R}^d$ is the final activation at position t, $z_t \in \mathbb{R}^V$ is the logit vector, and

$$p_t := p_\theta(\,\cdot \mid x_{<t}) \in \Delta_V, \qquad \Delta_V := \left\{ p \in [0,1]^V : \sum_{v=1}^V p_v = 1 \right\}, \tag{1.12}$$

is the next-token probability vector. A decoding rule selects $x_t \in [V]$ from p_t. The next sections expand this calculation.

Where is position added? In (1.10), absolute position information is added once, before the first transformer block; relative bias and RoPE instead modify attention (to be described in the next section) within each block.

1.4 One fully specified decoder block

Suppose $H^\ell \in \mathbb{R}^{T\times d}$ enters layer ℓ. The lecture described the attention calculation. This note adds the standard multiple-head structure as a correction and specifies a pre-normalized block with h heads. Usually $d_h = d/h$, so $d_h \ll d$ and the concatenated head output has width $hd_h = d$. In H^ℓ, row i is the activation at sequence position i, and column r is model-feature coordinate r:

$$H^\ell = \begin{bmatrix} (h_1^\ell)^\top \\ \vdots \\ (h_T^\ell)^\top \end{bmatrix} \in \mathbb{R}^{T\times d}, \qquad h_i^\ell = (H_{i,:}^\ell)^\top \in \mathbb{R}^d. \tag{1.13}$$

First define the normalization map. For $u \in \mathbb{R}^d$, let

$$\mu(u) = \frac{1}{d}\sum_{r=1}^d u_r, \qquad \sigma^2(u) = \frac{1}{d}\sum_{r=1}^d (u_r - \mu(u))^2, \tag{1.14}$$

and, for learned $\gamma_{\ell,m}, \beta_{\ell,m} \in \mathbb{R}^d$ and fixed $\varepsilon > 0$, define

$$\mathrm{LayerNorm}_{\ell,m}(u) = \gamma_{\ell,m} \odot \frac{u - \mu(u)\mathbf{1}_d}{\sqrt{\sigma^2(u) + \varepsilon}} + \beta_{\ell,m}, \qquad m \in \{1, 2\}. \tag{1.15}$$

On a matrix, $\mathrm{LayerNorm}_{\ell,m}$ acts on each stored position: the transpose of output row i is $\mathrm{LayerNorm}_{\ell,m}((H^\ell_{i,:})^\top)$. Thus it normalizes the d feature coordinates at each position independently. This note adds layer normalization so that the displayed block is fully specified.

Llama 3 uses *root-mean-square layer normalization* (RMSNorm). For a position activation $u \in \mathbb{R}^d$, define

$$\mathrm{RMSNorm}_{\ell,m}(u) = \gamma^{\mathrm{rms}}_{\ell,m} \odot \frac{u}{\sqrt{d^{-1}\sum_{r=1}^d u_r^2 + \varepsilon}}, \qquad \gamma^{\mathrm{rms}}_{\ell,m} \in \mathbb{R}^d. \tag{1.16}$$

RMSNorm rescales each position by the root mean square of its feature coordinates. It leaves the feature mean uncentered and, in Llama 3, uses a learned coordinatewise scale without an additive bias.

At the block-diagram level, a pre-normalized decoder layer performs two residual calculations:

$$H^\ell \longrightarrow \boxed{\text{LayerNorm} \longrightarrow \text{masked multi-head attention}} \xrightarrow{+H^\ell} G, \tag{1.17}$$

$$G \longrightarrow \boxed{\text{LayerNorm} \longrightarrow \text{MLP or MoE}} \xrightarrow{+G} H^{\ell+1}. \tag{1.18}$$

The equations below specify the calculations inside these boxes.

For head $a \in [h]$, let

$$\begin{aligned}
U &= \mathrm{LayerNorm}_{\ell,1}(H^\ell), && U \in \mathbb{R}^{T\times d},\\
Q_a &= UW^Q_{\ell,a}, && Q_a \in \mathbb{R}^{T\times d_h},\\
K_a &= UW^K_{\ell,a}, && K_a \in \mathbb{R}^{T\times d_h},\\
V_a &= UW^V_{\ell,a}, && V_a \in \mathbb{R}^{T\times d_h},\\
A_a &= \mathrm{softmax}_{\mathrm{row}}\left(\frac{Q_aK_a^\top}{\sqrt{d_h}} + M\right), && A_a \in \mathbb{R}^{T\times T},\\
O_a &= A_aV_a, && O_a \in \mathbb{R}^{T\times d_h}.
\end{aligned} \tag{1.19}$$

where

$$W^Q_{\ell,a} \in \mathbb{R}^{d\times d_h}, \qquad W^K_{\ell,a} \in \mathbb{R}^{d\times d_h}, \qquad W^V_{\ell,a} \in \mathbb{R}^{d\times d_h}, \tag{1.20}$$

and $M \in (\mathbb{R} \cup \{-\infty\})^{T\times T}$ is the causal mask

$$M_{ij} = \begin{cases} 0, & j \le i, \\ -\infty, & j > i. \end{cases} \tag{1.21}$$

The matrices U, Q_a, K_a, V_a, and O_a all have one row per sequence position. The columns of U are model features; the columns of Q_a, K_a, V_a, O_a are head features. The score and attention matrices lie in $\mathbb{R}^{T\times T}$. Their row i refers to query or output position i, and their column j refers to key/value or source position j. In particular, $(A_a)_{ij}$ is the weight with which position i uses the value at position j.

The softmax acts row by row, where a row represents one query position and its columns represent the available key/value source positions. The causal mask makes the attention weight of every future source position exactly zero.

Position mechanisms inside attention. A layer- and head-specific relative-position bias changes the attention calculation to

$$A_a = \mathrm{softmax}_{\mathrm{row}}\left(\frac{Q_a K_a^\top}{\sqrt{d_h}} + M + B_{\ell,a}\right), \qquad (B_{\ell,a})_{ij} = b_{\ell,a}\big(\rho(i-j)\big). \tag{1.22}$$

Here ρ may clip or bucket the relative displacement $i-j$, and $b_{\ell,a}$ assigns a learned scalar to each bucket. Rows index query positions and columns index key positions, so $i-j$ compares output position i with source position j. T5 uses this form with the bias parameters shared across layers and different across heads. ALiBi uses a fixed, head-dependent linear penalty such as $-m_a(i-j)$ for a causal pair $j \le i$. RoPE applies a position-dependent rotation to queries and keys in each attention layer that uses RoPE. The relative-displacement calculation for RoPE will be developed separately in the homework.

Write $O_a(U, M)$ for the output of head a produced by (1.19) from normalized input U and mask M. Define the complete multi-head self-attention map by concatenating the head outputs and applying the output projection:

$$\begin{aligned} \mathrm{SelfAttn}_M^\ell(U) &:= [O_1(U,M) \mid \cdots \mid O_h(U,M)]W_\ell^O, \qquad W_\ell^O \in \mathbb{R}^{hd_h \times d}, \\ G &= H^\ell + \mathrm{SelfAttn}_M^\ell(U). \end{aligned} \tag{1.23}$$

The block matrix $[O_1 \mid \cdots \mid O_h] \in \mathbb{R}^{T \times hd_h}$ has the head outputs $O_1, \ldots, O_h$ as its column blocks. Multiplication by W_ℓ^O maps the hd_h head features back to d model features, so $G \in \mathbb{R}^{T \times d}$ can be added to H^ℓ.

Let $\phi : \mathbb{R} \to \mathbb{R}$ be a scalar activation function. For any matrix A, we use $\phi(A)$ for its entrywise extension, $(\phi(A))_{ij} = \phi(A_{ij})$. Define the positionwise feed-forward map and its residual update by

$$\begin{aligned} R &= \mathrm{LayerNorm}_{\ell,2}(G), \\ \mathrm{FF}^\ell(R) &:= \phi\Big(RW_\ell^1 + \mathbf{1}_T (b_\ell^1)^\top\Big) W_\ell^2 + \mathbf{1}_T (b_\ell^2)^\top, \\ H^{\ell+1} &= G + \mathrm{FF}^\ell(R). \end{aligned} \tag{1.24}$$

with

$$W_\ell^1 \in \mathbb{R}^{d \times d_{\mathrm{ff}}}, \quad W_\ell^2 \in \mathbb{R}^{d_{\mathrm{ff}} \times d}, \quad b_\ell^1 \in \mathbb{R}^{d_{\mathrm{ff}}}, \quad b_\ell^2 \in \mathbb{R}^d. \tag{1.25}$$

Common choices for ϕ include $\mathrm{ReLU}(z) = \max\{0, z\}$ and $\mathrm{GELU}(z) = z\Phi(z)$, where Φ is the standard normal cumulative distribution function.

The lecture also mentioned *mixture-of-experts* layers. They replace the single dense feed-forward map by expert maps $f_1, \ldots, f_m$ and a router. For one position activation $r \in \mathbb{R}^d$, let each expert be a map $f_e : \mathbb{R}^d \to \mathbb{R}^d$. A sparse version has the form

$$f_{\mathrm{MoE}}(r) = \sum_{e \in S(r)} \rho_e(r) f_e(r), \tag{1.26}$$

where $S(r)$ is a small router-selected subset of experts and the router weights $\rho_e(r)$ are normalized over that subset. This replacement changes the positionwise sublayer; the attention calculation remains unchanged.

Transformer architectures vary in their normalization, feed-forward architecture, organization of attention heads, and position mechanism. The equations above define one concrete pre-normalized decoder block.

Figures 1 and 2 instantiate this common calculation for two dense decoder-only models. They display the complete residual paths and the placement of normalization, position information, attention, dropout, and feed-forward maps. The numerical labels are architectural dimensions, not activation shapes for one particular input sequence.

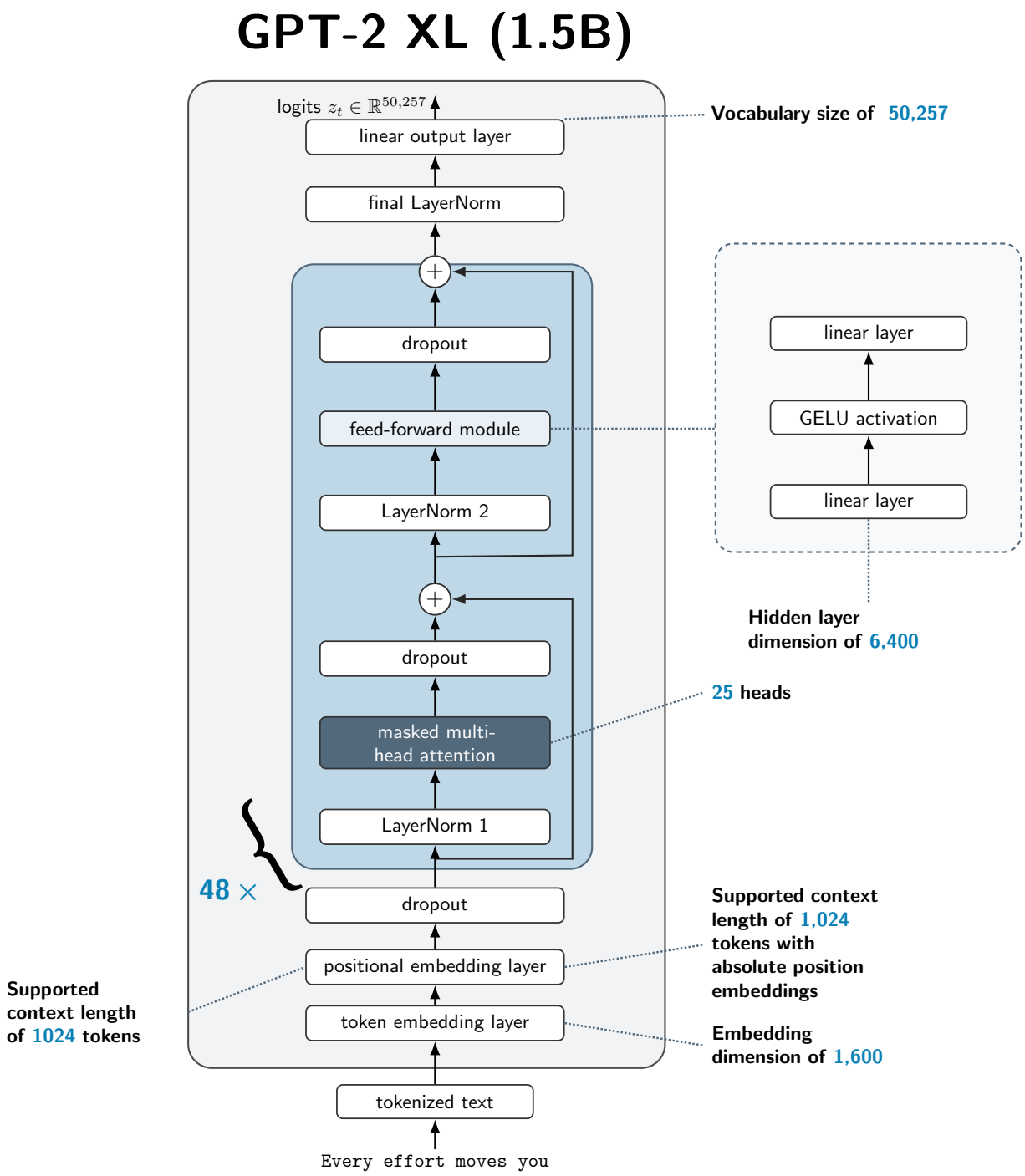

Figure 1: The GPT-2 XL decoder stack. The central blue region is one pre-normalized transformer block, repeated 48 times; the two side paths are its residual connections. The inset expands the feed-forward module into its two linear maps and GELU activation. The drawing is an independent LaTeX rendering of the architecture and dimensions catalogued by Raschka [2026].

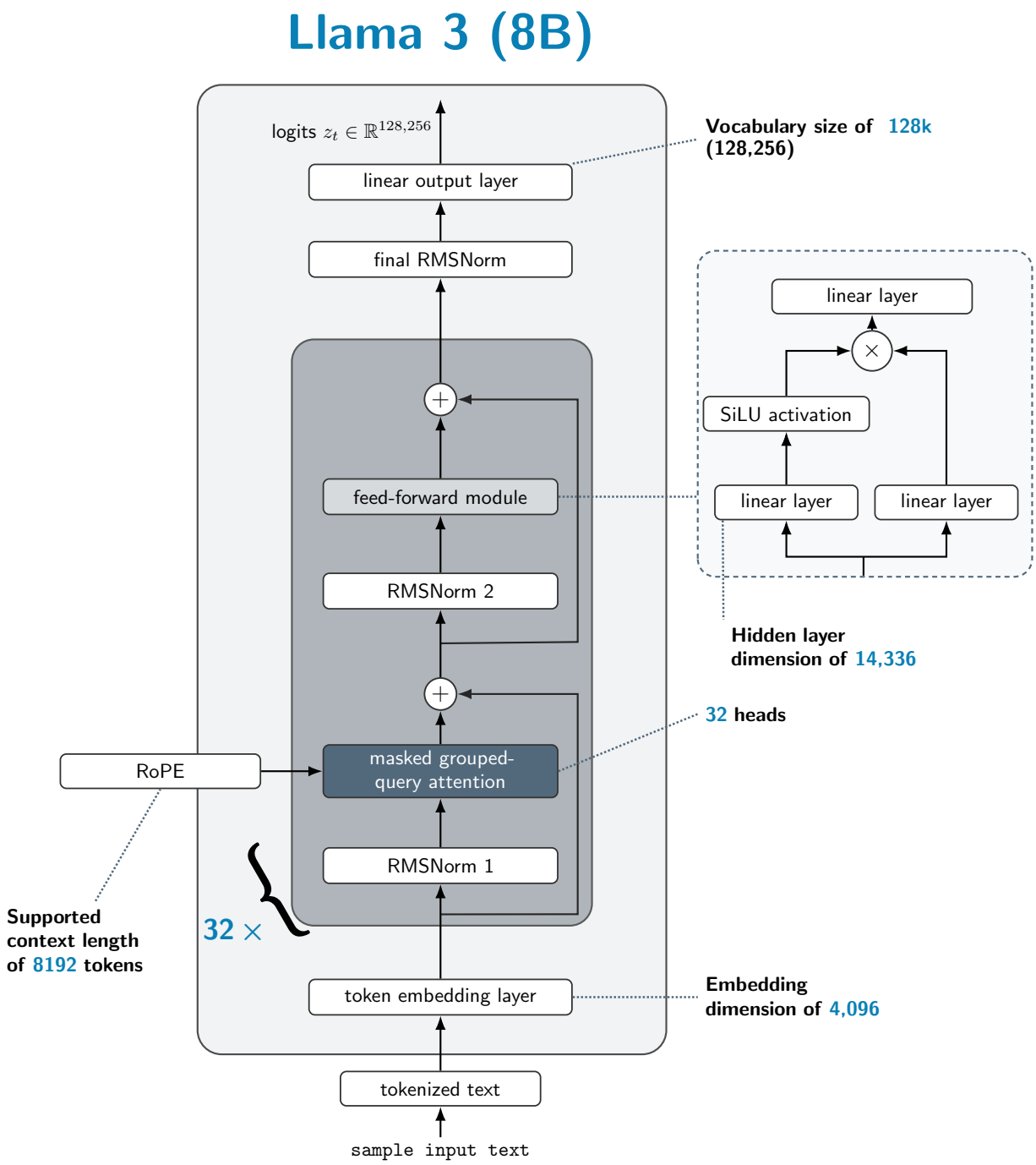

Figure 2: The Llama 3 8B decoder stack. The central gray region is one pre-normalized transformer block, repeated 32 times. RoPE modifies the query and key vectors inside grouped-query attention. The inset shows both SwiGLU branches, their coordinatewise product, and the output projection. The 32 attention heads are query heads, grouped over 8 key/value heads. The drawing is an independent LaTeX rendering of the architecture and dimensions catalogued by Raschka [2026] and checked against Grattafiori et al. [2024].

For comparisons in which these internal details would obscure the structure, write

$$\mathcal{B}_{\ell,M} : \mathbb{R}^{T\times d} \longrightarrow \mathbb{R}^{T\times d}, \qquad H^{\ell+1} = \mathcal{B}_{\ell,M}(H^{\ell}), \tag{1.27}$$

for the entire layer just specified, with attention mask M and the learned parameters of layer ℓ. Thus a stack of L blocks is the composition

$$H^{L} = \big(\mathcal{B}_{L-1,M} \circ \cdots \circ \mathcal{B}_{0,M}\big)(H^{0}). \tag{1.28}$$

The notation hides the attention, residual, normalization, and MLP or MoE calculations while preserving the input state and attention mask of each block. Appendix B uses this notation to compare three transformer architecture types: encoder-only, decoder-only, and encoder–decoder.

The key invariant is causal dependence: after L blocks, the column vector $h_t^L = (H_{t,:}^L)^\top$ depends only on $x_{<t}$.

Remark (SwiGLU). A common gated feed-forward variant is SwiGLU. For a column activation $r \in \mathbb{R}^d$ and a hidden width m, let

$$f_{\mathrm{SwiGLU}}(r) = W_o(\mathrm{swish}(W_g r) \odot (W_v r)), \qquad \mathrm{swish}(z) = \frac{z}{1+e^{-z}}, \tag{1.29}$$

where $W_g, W_v \in \mathbb{R}^{m\times d}$ and $W_o \in \mathbb{R}^{d\times m}$. The scalar function acts coordinatewise, and $\odot$ denotes coordinatewise multiplication: $(a \odot b)_i = a_i b_i$. This product creates input-dependent gating between two learned representations of r.

1.5 From a hidden vector to the next-token distribution

Let $W_U \in \mathbb{R}^{V\times d}$ and $b_U \in \mathbb{R}^V$. The final hidden vector is converted to logits and then to probabilities:

$$z_t = W_U h_t^L + b_U \in \mathbb{R}^V, \tag{1.30}$$

$$p_{t,v} := p_\theta(x_t = v \mid x_{<t}) = \frac{\exp(z_{t,v})}{\sum_{u=1}^{V} \exp(z_{t,u})}. \tag{1.31}$$

The column vector $h_t^L \in \mathbb{R}^d$ has one entry per model feature. Rows of W_U index vocabulary items and columns index model features. Consequently, $z_t \in \mathbb{R}^V$ has one logit for each vocabulary item, and the softmax turns these logits into next-token probabilities. Weight tying, the common optional choice $W_U = E$, reuses the input embedding matrix at the readout.

The single-token path is therefore

$$e_{x_{t-1}} \longrightarrow E^\top e_{x_{t-1}} \longrightarrow h_t^L \longrightarrow z_t \longrightarrow p_t. \tag{1.32}$$

The contextual vector h_t^L also depends on the earlier tokens through the masked attention blocks.

1.6 Autoregressive generation

A sampled decoder draws

$$X_t \sim p_\theta(\,\cdot \mid X_{<t}), \tag{1.33}$$

whereas greedy decoding chooses

$$X_t \in \arg\max_v p_\theta(v \mid X_{<t}). \tag{1.34}$$

In either case the selected token is appended and the procedure repeats until a stopping rule fires.

Greedy decoding is neither sampling from the model nor, in general, global optimization of the probability of the complete sequence. A locally most probable token can lead to poor later continuations.

There are narrow cases in which greedy decoding is justified. If the system must make exactly one token-valued decision and incurs zero–one loss, an argmax token is Bayes optimal. Greedy decoding is also harmless when every conditional distribution is effectively deterministic, or when a special problem has a greedy-choice property. None of these conditions holds in general for open-ended generation.

Practical note. For open-ended multi-token generation, forget greedy decoding: it is a bad idea. It throws away nearly all of the predictive distribution at every step, because a locally best token can irreversibly eliminate much better continuations.

Remark. A quantitative counterexample to repeated tokenwise argmax will be developed separately in the homework.

1.7 Attention cost and the key–value cache

Let $h_i^\ell = (H_{i,:}^\ell)^\top \in \mathbb{R}^d$ denote the *layer-position activation* at position i after layer ℓ. Computing the output for position i in one causal attention layer compares its query with i keys. Consequently, a full forward pass on a fixed length-t prefix performs

$$\sum_{i=1}^{t} i = \frac{t(t+1)}{2} \tag{1.35}$$

query–key comparisons per head. Both $QK^\top$ and the multiplication of the resulting attention matrix by V therefore require order $t^2 d_h$ arithmetic. A straightforward implementation also materializes a $t \times t$ attention matrix. The causal mask makes the upper triangle irrelevant while preserving quadratic dependence on t. An attention implementation that skips this triangle processes $t(t+1)/2$ permitted query–key pairs, compared with t^2 pairs for a bidirectional head. The saving applies whenever all sequence positions are processed together, including training, prompt prefill, and full-sequence scoring. Bidirectional attention has no masked triangle to skip. Such an implementation can therefore save nearly half of the score and attention–value arithmetic. The whole-block saving is smaller because projections, normalization, and the MLP or MoE remain. An implementation that first computes the full square matrix and masks it afterward realizes no arithmetic saving.

The first full forward pass over a supplied prompt is called *prefill*. After prefill, a *decode step* generates one new token and advances only its layer-position activations through the network. These two phases have different computational profiles.

Autoregressive decoding has additional structure. At step t and in each layer, only h_t^ℓ, the activation of the new token, must be advanced. Let $q_t, k_t, v_t \in \mathbb{R}^{d_h}$ be its query, key, and value vectors, with the layer and head indices suppressed. The attention operation for the new position is

$$\alpha_t = \operatorname{softmax}\left(\frac{K_{\le t} q_t}{\sqrt{d_h}}\right) \in \mathbb{R}^t, \qquad o_t = V_{\le t}^\top \alpha_t \in \mathbb{R}^{d_h}. \tag{1.36}$$

The matrices $K_{\le t}, V_{\le t} \in \mathbb{R}^{t \times d_h}$ store $k_1^\top, \dots, k_t^\top$ and $v_1^\top, \dots, v_t^\top$ in their rows. Thus α_t has one weight per cached source position, and o_t is the new output vector. The earlier keys and values are

therefore stored in a *KV cache*. One cached decode step uses order td_h attention arithmetic per head, and after t positions the cache stores order td_h numbers per head and layer.

Without a cache, step t recomputes the complete length-t forward pass, including all earlier layer-position activations, so the attention work over T generated positions is

$$\sum_{t=1}^{T} \Theta(t^2 d_h) = \Theta(T^3 d_h). \tag{1.37}$$

With the cache, only the new row is computed at each step, and the total is

$$\sum_{t=1}^{T} \Theta(t d_h) = \Theta(T^2 d_h). \tag{1.38}$$

Thus caching changes the asymptotic attention cost of sequential generation.

Why are queries not cached? The query q_i was used to compute the output at position i. A future position $t > i$ forms a new query q_t and compares it with the old key k_i; the resulting weight multiplies the old value v_i. Thus future computation reuses k_i and v_i. The old query q_i has completed its role, so caching it would consume memory without avoiding future computation.

1.8 Context length and a bounded-score limitation

A model call receives a *context window*: the sequence consisting of the prompt followed by any tokens generated so far. Practical systems come with a *maximum context length*, an upper bound on the number of tokens in the context window. Position handling, training range, cache memory, attention cost, and implementation choices can all constrain the maximum context length. The ability to accept T tokens and the ability to use all T tokens effectively are distinct capabilities.

Making full use of a long context window can require sharply selecting a particular position. In particular, for one query attending to T keys, write

$$s_j = \frac{q^\top k_j}{\sqrt{d_h}}, \qquad \alpha_j = \frac{e^{s_j}}{\sum_{r=1}^{T} e^{s_r}}. \tag{1.39}$$

If the scores remain in a fixed bounded range as T grows, no individual softmax weight can remain sharply concentrated. A quantitative upper bound, its proof, and the corresponding entropy consequence will be developed separately in the homework.

The conclusion is conditional on bounded scores; score gaps that grow with context length evade it.

Entropy language. For a probability vector $\alpha \in [0,1]^T$ with $\sum_{j=1}^{T} \alpha_j = 1$, its entropy is

$$\mathsf{H}(\alpha) = -\sum_{j=1}^{T} \alpha_j \log \alpha_j. \tag{1.40}$$

It is 0 for a deterministic distribution and $\log T$ for the uniform distribution. Thus small entropy describes concentrated attention and large entropy describes diffuse attention.

Related empirical phenomenon. The terms *lost in the middle* and *context rot* refer to observed failures to use long contexts uniformly or reliably, sometimes well below a model's advertised maximum length.

1.9 What was said about training

Training was described only in words. The system is shown many text prefixes together with the tokens that actually followed them. Its predictions are compared with those next tokens, and the shared weights are modified to improve the predictions across the data. This initial large-scale training stage, usually conducted on a broad corpus, is called *pretraining*. *Fine-tuning* names the operation of continuing parameter training from an existing checkpoint. *Post-training* names the stage after pretraining and may contain several fine-tuning procedures, including supervised instruction tuning and preference- or reward-based optimization. A supervised fine-tuning stage after pretraining is both fine-tuning and post-training. During *inference*, the resulting parameters are held fixed while the model calls described in this note compute predictions. The mathematics of pretraining will be developed later.

1.10 What exactly is contained in the parameter vector?

The symbol θ collects the trainable numerical parameters of the conditional model. For the particular architecture displayed in this note, and assuming that the additive position vectors are learned, one explicit accounting is

$$\theta = \Big(E, P, W_U, b_U, \{W^Q_{\ell,a}, W^K_{\ell,a}, W^V_{\ell,a}\}_{\ell,a}, \{W^O_\ell, W^1_\ell, W^2_\ell, b^1_\ell, b^2_\ell, \gamma_{\ell,1}, \beta_{\ell,1}, \gamma_{\ell,2}, \beta_{\ell,2}\}_\ell\Big). \quad (1.41)$$

With weight tying, E determines $W_U = E$. Fixed position maps and the causal mask M lie outside the learned parameters. An MoE layer replaces the dense-MLP entries above by the expert and router parameters. The token vocabulary and segmentation rules are normally constructed before neural-network training and held fixed. They determine V, the input sequences, and the shapes of E and W_U while remaining outside θ in this account.

The token prefix, hidden states, queries, keys, values, attention weights, logits, and KV cache are computed quantities outside θ. The decoding rule and the surrounding harness are also outside θ. Thus training the model means choosing the shared numerical parameters in this display; specifying the deployed AI system requires additional choices.

1.11 Take-home messages

- A deployed AI system places the language model inside a "harness" that constructs context, can maintain state and invoke tools, and may make repeated model calls.
- A lossless tokenizer segments a sequence over a base alphabet into nonempty vocabulary strings. Its token sequence cannot be longer than the base-symbol sequence and is shorter whenever a selected token contains several base symbols. Tokens may lack semantic meaning.
- A stochastic language or sequence model specifies a stochastic process whose state space is its vocabulary. An autoregressive model specifies its finite-sequence distributions through next-token conditionals.

- A decoder transformer maps a prefix to a categorical distribution by embedding tokens, applying causally masked attention and positionwise transformations, and using a softmax readout.
- Absolute position embeddings are added once at the input. RoPE and relative-position biases instead modify the attention calculation in every layer that uses them.
- The learned conditional distribution and the decoding algorithm are different objects. An argmax is optimal for a single zero–one-loss decision. Repeated tokenwise argmax generally fails to optimize a sequence-level objective and is a poor default for open-ended generation.
- A full dense-attention pass on a length-t prefix has quadratic arithmetic cost. An implementation of causal attention can skip nearly half the attention pairs used by bidirectional attention at the same length. During sequential generation, a KV cache avoids recomputing old layer-position activations and changes total attention work from cubic to quadratic in the generated length. Old queries have no future use.
- Maximum context length is an input-capacity limit. Performance on some retrieval and reasoning tasks may deteriorate before the input reaches this limit. With bounded attention scores, a single softmax head cannot remain sharply concentrated as the context grows.
- The parameter vector θ contains the learned weights of the conditional model. The current activations, KV cache, tokenizer, decoding rule, and harness lie outside θ. Lecture 2 takes up the question of why learning can produce useful values of θ.

1.12 Glossary

Harness; task state
The program surrounding the model, and the information that this program maintains between model calls.

Prompt
The initial input supplied to a model call. In the text-only setting, it is a string that tokenization converts into the initial token sequence.

Prefix
The token sequence $x_{<t}$ conditioning the prediction at position t. During generation, it consists of the tokenized prompt followed by the tokens generated so far.

Context; context window
The information made visible to one model call, and the token sequence through which that information is presented to the model.

Base alphabet; vocabulary; token
The symbols used to represent raw sequences; the finite collection of strings made available to the model; and one vocabulary item in a tokenized sequence.

Tokenizer; detokenizer
The map from a base-symbol string to vocabulary indices, and its inverse concatenation map.

Sequence orientation; matrix layout
Positions run left to right in the board diagrams. In the equations, positions are stacked as rows

and the columns index coordinates such as vocabulary items or learned features. Every vector is a column vector; a matrix row that stores a vector contains its transpose.

Stochastic process; stochastic sequence model; autoregressive model
A jointly distributed collection of random variables indexed by time; its specialization to vocabulary-valued random variables; and a specification of its finite-sequence distributions through conditional next-token distributions.

Pretraining; post-training; fine-tuning
Initial large-scale fitting on broad data; the stage of training that follows pretraining; and continued parameter training from an existing checkpoint. A post-training pipeline may contain several fine-tuning procedures.

Inference; generation
Computation with the model parameters held fixed, and repeated next-token inference and decoding that produce a continuation.

Encoder-only; decoder-only; encoder–decoder transformer
A bidirectional stack that produces contextual representations of an input; a causal stack that places prompt and continuation in one sequence; and a two-stack architecture in which a causal decoder separately reads the representation produced by a bidirectional encoder.

Self-attention; cross-attention; transformer block
Attention whose queries, keys, and values come from one sequence; attention whose decoder queries read keys and values derived from encoder output; and one repeated attention-plus-positionwise-transformation layer.

Embedding; logit; softmax readout
The vector assigned to a token, an unnormalized output score, and the map from the output scores to a categorical distribution.

Parameter; activation; layer-position activation; residual stream
A learned quantity shared across inputs; a value computed for the current input; the vector h_i^ℓ at position i after layer ℓ; and the collection of positionwise vectors carried through and updated by the transformer blocks.

Layer normalization; pre-normalized block
Normalization across the feature coordinates at one sequence position, followed by learned coordinatewise scaling and shifting; and a transformer block that applies this normalization before each learned sublayer.

ReLU; GELU; SwiGLU
Two scalar activation functions applied coordinatewise, and a gated feed-forward map formed from two learned projections and their coordinatewise product.

Attention head; query; key; value
One attention calculation; the vector specifying what an output position seeks; the vector against which it is matched at a source position; and the source vector combined into the output.

Attention score; attention weight
The scaled query–key inner product before normalization; its row-softmax coefficient after applying biases and masks.

Causal mask; prefix causality
The constraint that gives future source positions zero attention weight; and the resulting property that an output at position i cannot depend on tokens at later positions.

Multi-head attention; residual connection; MLP; mixture of experts
Parallel attention heads; an identity path around a learned transformation; the dense positionwise sublayer; and its sparsely routed expert alternative.

Absolute position embedding; relative-position bias; RoPE
Three different mechanisms for making token order available to attention.

Decoding rule; sampling; greedy decoding
The procedure that turns a next-token distribution into an output token; random generation from that distribution; and tokenwise argmax.

Prefill; decode step; KV cache
The initial full-prompt forward pass; one incremental generation step; and the stored keys and values that prevent recomputation of old activations.

Maximum context length
The token-capacity ceiling for the context window of one model call.

Attention entropy
A measure of how diffuse an attention distribution is.

Lost in the middle; context rot
Names for empirical deterioration in a trained system's use of information as its location or the total input length changes, potentially well below the maximum context length.

A Supplement: how tokenizers are constructed

This appendix adds optional background after the lecture, separate from the lecture narrative reconstructed above.

Two common corpus-based constructions illustrate the main ideas.

1. **Byte-pair encoding (BPE).** Start with base symbols, count adjacent token pairs in the training corpus, merge a most frequent pair into a new vocabulary item, and repeat until the desired vocabulary size is reached. The learned merge order determines the segmentation of new text [Sennrich et al., 2016].

2. **Unigram tokenization.** Start with a large collection of candidate substrings and probabilities $q(v)$. A segmentation is scored by

$$\prod_{t=1}^{T} q(v_{x_t}), \qquad \text{or equivalently} \qquad \sum_{t=1}^{T} \log q(v_{x_t}). \tag{1.42}$$

   Fit the probabilities to the corpus, remove candidates whose deletion hurts the likelihood least, and repeat. At encoding time one may choose a highest-scoring segmentation or sample among segmentations [Kudo, 2018; Kudo and Richardson, 2018].

B Supplement: three transformer architecture types

An encoder–decoder model is a probabilistic string-to-string map: each input string determines a distribution over output strings. A decoder-only language model is a probability distribution over strings. This supplement compares these models, together with encoder-only models, through their information flow: which tokens are available simultaneously, and which tokens must be generated from left to right?

Let $x_{1:S} = (x_1, \dots, x_S)$ denote a source sequence. In an encoder–decoder model, let $y_{1:T} = (y_1, \dots, y_T)$ denote the target sequence. Before the model generates target token y_t, it has access to the target prefix

$$y_{<t} := (y_0, y_1, \dots, y_{t-1}), \tag{1.43}$$

where y_0 is a beginning-of-target token. During teacher-forced training, the shifted inputs $y_0, \dots, y_{T-1}$ produce predictions for $y_1, \dots, y_T$ in parallel under a causal mask. The notation $\mathrm{Embed}_{\mathrm{src}}$ and $\mathrm{Embed}_{\mathrm{tgt}}$ refers to the token-and-position embedding construction in (1.10); the two subscripts indicate separately learned embedding parameters.

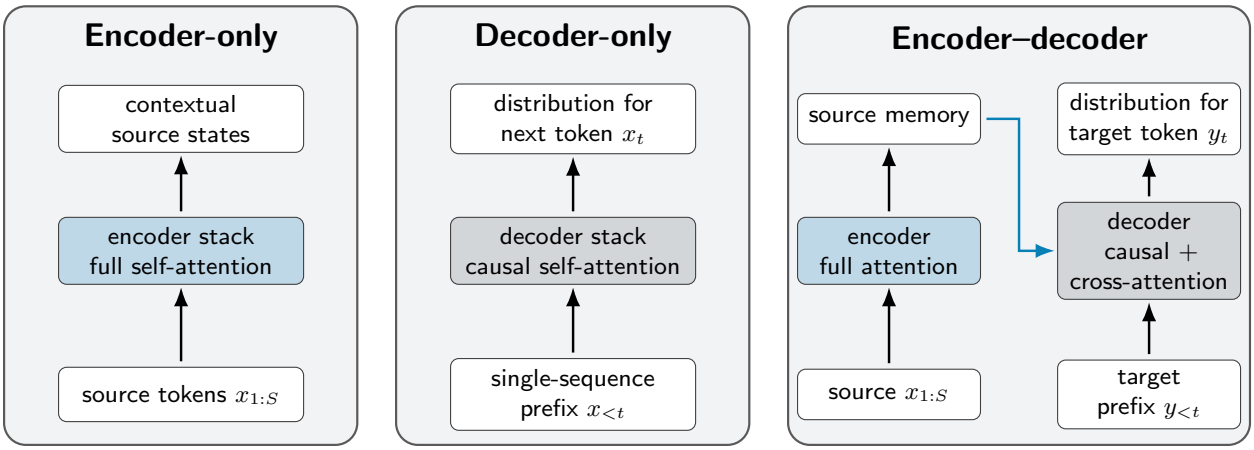

Figure 3: Information flow in three transformer architecture types. Encoder-only models contextualize a fixed input. Decoder-only models predict the continuation of one causal sequence. Encoder–decoder models first encode the complete source and then generate a target that can attend to that source memory.

Encoder-only. An encoder-only model constructs a contextual representation of a fixed input. Let $M_{\mathrm{full}} = 0$ allow every position to attend to every other position. The encoder stack is

$$E^0 = \mathrm{Embed}_{\mathrm{src}}(x_{1:S}), \qquad E^{\ell+1} = \mathcal{B}^{\mathrm{enc}}_{\ell, M_{\mathrm{full}}}(E^\ell), \quad \ell = 0, \dots, L_E - 1. \tag{1.44}$$

Thus every row of $E^{L_E} \in \mathbb{R}^{S \times d}$ can contain information from the entire source sequence. A task-specific readout converts these contextual states into predictions.

Decoder-only. A decoder-only model represents a single sequence and predicts its continuation. With the triangular mask M_{causal} from Section 1.4,

$$H^0 = \mathrm{Embed}(x_{1:T}), \qquad H^{\ell+1} = \mathcal{B}^{\mathrm{dec}}_{\ell, M_{\mathrm{causal}}}(H^\ell), \quad \ell = 0, \dots, L - 1. \tag{1.45}$$

At every depth, the state at position i depends only on tokens at positions $1, \dots, i$. A prompt and its continuation therefore occupy one causal token stream.

Encoder–decoder. An encoder–decoder model is designed for generating a target sequence conditioned on a separately supplied source sequence. The encoder first computes E^{L_E} using (1.44). This matrix is the *source memory*: it contains one contextual state for each source position and remains fixed while the target is generated. The target decoder starts from

$$D^0 = \mathrm{Embed}_{\mathrm{tgt}}(y_{<t}). \tag{1.46}$$

The compact equations below use three explicitly defined maps. The map $\mathrm{SelfAttn}^r_M(U)$ is the multi-head self-attention map defined in (1.23); it forms queries, keys, and values from the same matrix U and applies mask M. The map $\mathrm{FF}^r(U)$ is the positionwise feed-forward map in (1.24). Cross-attention uses target states for queries and source-memory states for keys and values. For target states $U \in \mathbb{R}^{m\times d}$ and source memory $E \in \mathbb{R}^{S\times d}$, one cross-attention head has

$$\begin{aligned} Q_a &= UW^{Q,\mathrm{cross}}_{r,a}, & K_a &= EW^{K,\mathrm{cross}}_{r,a}, \quad V_a = EW^{V,\mathrm{cross}}_{r,a}, \\ A_a &= \mathrm{softmax}_{\mathrm{row}}\left(\frac{Q_aK_a^\top}{\sqrt{d_h}}\right), & O_a &= A_aV_a. \end{aligned} \tag{1.47}$$

The map $\mathrm{CrossAttn}^r(U,E)$ concatenates the head outputs O_a and applies the cross-attention output projection, exactly as in (1.23). Each self-attention, cross-attention, and feed-forward sublayer has its own learned parameters. Let $\mathrm{LN}_{r,1}$, $\mathrm{LN}_{r,2}$, and $\mathrm{LN}_{r,3}$ denote their three separate rowwise normalization maps. The decoder block at depth r is then

$$\begin{aligned} G^r &= D^r + \mathrm{SelfAttn}^r_{M_{\mathrm{causal}}}\big(\mathrm{LN}_{r,1}(D^r)\big), \\ J^r &= G^r + \mathrm{CrossAttn}^r\big(\mathrm{LN}_{r,2}(G^r), E^{L_E}\big), \\ D^{r+1} &= J^r + \mathrm{FF}^r\big(\mathrm{LN}_{r,3}(J^r)\big). \end{aligned} \tag{1.48}$$

The first line mixes information among available target-prefix positions. The second line retrieves information from any source position. The third line applies the feed-forward map independently at each target position. Each line adds the sublayer output back to its input through a residual connection.

What changes computationally? For a length-n sequence, one full-attention head has n^2 permitted query–key pairs, whereas one causal head has only the lower-triangular pairs:

$$N_{\mathrm{full}}(n) = n^2, \qquad N_{\mathrm{causal}}(n) = \frac{n(n+1)}{2}. \tag{1.49}$$

A triangular attention kernel can avoid computing the masked upper triangle. These expressions count attention pairs per head; they do not include linear projections or feed-forward computation.

For source length S and target length T, L_E encoder blocks and L_D encoder–decoder blocks permit

$$N_{\text{enc-dec}} = L_ES^2 + L_D\frac{T(T+1)}{2} + L_DST. \tag{1.50}$$

The three terms count source self-attention, causal target self-attention, and target-to-source cross-attention, respectively. A fully causal decoder-only stack of L blocks on a concatenated source and target permits

$$N_{\text{dec-only}} = L\frac{(S+T)(S+T+1)}{2}. \tag{1.51}$$

Complete runtime comparisons also depend on model depths, widths, head counts, parameter counts, feed-forward layers, and the available hardware kernels.

Why are decoder-only models common for language modeling? Next-token pretraining on raw text supplies a prediction target at every position and requires no separate source–target split. Training and generation also use the same single-sequence interface. These properties simplify large-scale training and serving. Encoder–decoder models provide a different advantage for conditional generation: every target token can use a bidirectionally encoded source. In the controlled comparisons of Raffel et al. [2020], the encoder–decoder architecture was strongest at approximately matched computation in their setting.

Generation and caching. During encoder–decoder generation, the encoder computes the source memory once. Each decoder layer caches the key and value projections of that fixed memory for cross-attention. It separately grows a causal self-attention cache for the generated target. At step t, the model forms new queries for the current target position and reuses both caches.

C Supplement: codebases for experiments

This appendix lists experimental resources. Each entry supports a different kind of experiment; the list serves as a menu for selective use.

NumPy RASP-L
A small reference implementation for writing and testing causal RASP-L sequence programs: https://github.com/apple/ml-np-rasp.

RASP and Tracr
The original RASP interpreter displays selectors and intermediate sequence operations. Tracr compiles a subset of RASP into concrete transformer weights: https://github.com/tech-srl/RASP and https://github.com/google-deepmind/tracr.

nanochat
A compact end-to-end codebase containing tokenization, pretraining, fine-tuning, evaluation, and inference. It includes small CPU and Apple-Silicon configurations, although training a capable language model still requires substantial computation: https://github.com/karpathy/nanochat.

Hugging Face Transformers
A broad model-definition and checkpoint ecosystem, useful for loading models and comparing tokenizers, architectures, and generation rules: https://github.com/huggingface/transformers.

TransformerLens
An inspection library for GPT-style models, useful for recording activations and attention patterns or intervening inside a forward pass: https://github.com/TransformerLensOrg/TransformerLens.

llama.cpp
A local inference implementation that makes tokenization, quantization, decoding, and key–value-cache behavior accessible to experiments: https://github.com/ggml-org/llama.cpp.

Bibliographic remarks

The compiler analogy is literal at the level of the interface: a lexer maps a character stream into tokens. A programming-language specification determines compiler tokens; corpus frequencies help determine LLM tokens. See Aho et al. [2006]. The supplementary appendix describes the BPE construction of Sennrich et al. [2016], the unigram formulation of Kudo [2018], and the SentencePiece system of Kudo and Richardson [2018].

The transformer architecture was introduced by Vaswani et al. [2017]. Their model combined encoder and decoder stacks; the equations in this lecture are a causal decoder specialization and use a pre-normalized block.

The SwiGLU feed-forward variant was introduced by Shazeer [2020]. It uses two hidden projections and coordinatewise gating; the paper also explains how to adjust the hidden width when comparing its parameter and operation counts with an ordinary two-matrix feed-forward layer. SwiGLU is used, for example, in the Llama 3 architecture [Grattafiori et al., 2024].

Three early pretraining recipes made the architectural alternatives concrete. The original GPT work paired a causal decoder language model with task-specific supervised fine-tuning [Radford et al., 2018]. BERT instead pretrained a bidirectional encoder with a masked-token objective and then fine-tuned its representations [Devlin et al., 2019]. T5 retained the encoder–decoder architecture and expressed many tasks through one text-to-text interface [Raffel et al., 2020]. GPT-3 later demonstrated that a sufficiently large causal language model could use task descriptions and examples supplied in its context, without a gradient update at test time [Brown et al., 2020]. These are historically important successful recipes.

The modular definitions in Appendix B follow the encoder–decoder construction of Vaswani et al. [2017] and the architectural comparisons of Raffel et al. [2020]. The latter explain why parameter count and computation cannot both be matched by changing depth alone and report the strongest performance from an encoder–decoder model at approximately matched computation in their setting.

Recent publicly released model families usually preserve the causal decoder-only interface while changing the surrounding recipe and internal details. Llama 3 is a dense example [Grattafiori et al., 2024], Mistral 7B uses grouped-query and sliding-window attention [Jiang et al., 2023], and DeepSeek-V3 uses a mixture-of-experts architecture [DeepSeek-AI, 2024]. The term *open-weight* asserts release of the numerical weights; training data, code, and licensing freedoms require separate verification. For a maintained visual catalogue of model-level architecture diagrams, configuration links, and compact fact sheets, see the LLM Architecture Gallery of Raschka [2026].

The original paper used additive sinusoidal position vectors. Rotary position embeddings are due to Su et al. [2021]; their position-dependent rotations of queries and keys expose relative displacement in the attention score. The bucketed learned relative-position bias in (1.22) follows Raffel et al. [2020]. Press et al. [2022] propose ALiBi, which instead adds a fixed head-dependent linear penalty based on relative distance.

Shazeer [2019] identified KV-cache bandwidth as a decoding bottleneck and introduced multi-query attention: multiple query heads share the same key and value heads. This reduces cached key–value data and the traffic required to read it. Grouped-query attention, introduced by Ainslie et al. [2023], provides an intermediate degree of sharing.

Dao et al. [2022] introduced FlashAttention, which computes exact dense attention by tiling and an online softmax. This reduces memory traffic and avoids materializing the full attention matrix while retaining the quadratic arithmetic of exact dense attention. The course returns to this distinction in Lecture 27.

The bounded-score calculation left for homework is elementary. Chiang and Cholak [2022]

study a related length-dependent loss of confidence and show that the obstruction depends on architectural details; one of their remedies scales attention logits with $\log T$. This is useful context for the homework result.

On the empirical capabilities of trained models across context-window lengths and locations of relevant information, Liu et al. [2024] vary the position of relevant information in multi-document question answering and key–value retrieval, demonstrating the distinction between accepting a long context and using it reliably. Hong et al. [2025] use the term "context rot" for performance degradation as input length grows across a broader collection of tasks and models.

The RASP language used in Appendix C was introduced by Weiss et al. [2021] to make attention-based sequence constructions explicit. Zhou et al. [2024] introduced RASP-L for causal decoder-only transformers and proposed short-program length as an empirical predictor of algorithmic learnability and length generalization. The two formalisms force proposed computations to specify their comparisons, causal selections, aggregations, and position operations.

References

A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison–Wesley, second edition, 2006.

R. Sennrich, B. Haddow, and A. Birch. Neural Machine Translation of Rare Words with Subword Units. *ACL*, 2016. https://arxiv.org/abs/1508.07909.

T. Kudo. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. *ACL*, 2018. https://arxiv.org/abs/1804.10959.

T. Kudo and J. Richardson. SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing. *EMNLP System Demonstrations*, 2018. https://aclanthology.org/D18-2012/.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. *NeurIPS*, 2017. https://arxiv.org/abs/1706.03762.

A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving Language Understanding by Generative Pre-Training. OpenAI technical report, 2018. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *NAACL*, 2019. https://arxiv.org/abs/1810.04805.

T. B. Brown et al. Language Models are Few-Shot Learners. *NeurIPS*, 2020. https://arxiv.org/abs/2005.14165.

A. Grattafiori et al. The Llama 3 Herd of Models. 2024. https://arxiv.org/abs/2407.21783.

S. Raschka. LLM Architecture Gallery. Maintained online reference, accessed September 3, 2026. https://sebastianraschka.com/llm-architecture-gallery/.

A. Q. Jiang et al. Mistral 7B. 2023. https://arxiv.org/abs/2310.06825.

DeepSeek-AI. DeepSeek-V3 Technical Report. 2024. https://arxiv.org/abs/2412.19437.

J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding. 2021. https://arxiv.org/abs/2104.09864.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. 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. https://www.jmlr.org/papers/v21/20-074.html.

O. Press, N. A. Smith, and M. Lewis. Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation. *ICLR*, 2022. https://arxiv.org/abs/2108.12409.

N. Shazeer. Fast Transformer Decoding: One Write-Head is All You Need. 2019. https://arxiv.org/abs/1911.02150.

N. Shazeer. GLU Variants Improve Transformer. 2020. https://arxiv.org/abs/2002.05202.

J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. *EMNLP*, 2023. https://arxiv.org/abs/2305.13245.

T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *NeurIPS*, 2022. https://arxiv.org/abs/2205.14135.

D. Chiang and P. Cholak. Overcoming a Theoretical Limitation of Self-Attention. *ACL*, 2022. https://aclanthology.org/2022.acl-long.527/.

N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics*, 2024. https://arxiv.org/abs/2307.03172.

K. Hong, A. Troynikov, and J. Huber. Context Rot: How Increasing Input Tokens Impacts LLM Performance. Chroma technical report, 2025. https://www.trychroma.com/research/context-rot.

G. Weiss, Y. Goldberg, and E. Yahav. Thinking Like Transformers. *ICML*, 2021. https://proceedings.mlr.press/v139/weiss21a.html.

H. Zhou, A. Bradley, E. Littwin, N. Razin, O. Saremi, J. Susskind, S. Bengio, and P. Nakkiran. What Algorithms Can Transformers Learn? A Study in Length Generalization. *ICLR*, 2024. https://arxiv.org/abs/2310.16028.