LLM Intro Course: 2 – Architecture design & open-weight models

Recap of Attention & Transformer Block, Positional Encodings, Normalization, Activations

Simon Vary

Mathematical Institute, University of Oxford

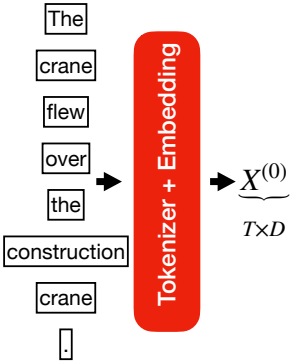
March 11, 2026

The plan for today

- **Recap:** tokenizer $\rightarrow$ embedding $\rightarrow$ Transformer block + Attention
- **Positional Encodings:** RoPE
- **Activations, Shapes, Stability**.
- **Open weight models:** examining open weight models

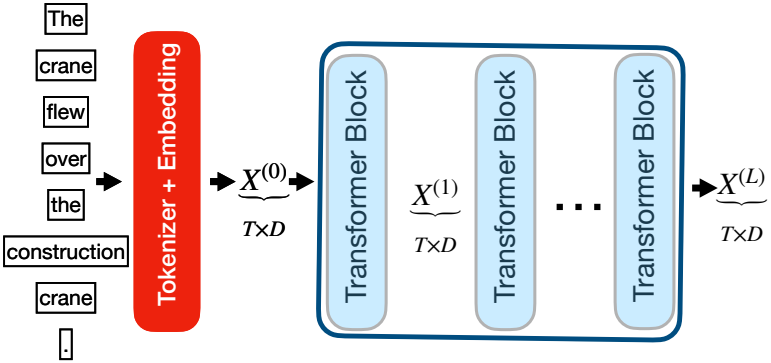
Recap

GPT2-like Decoder Model



$\text{Tokenizer} + \text{Embedding} : \Sigma^* \to \mathbb{R}^{T \times D}$ string to sequence embedding

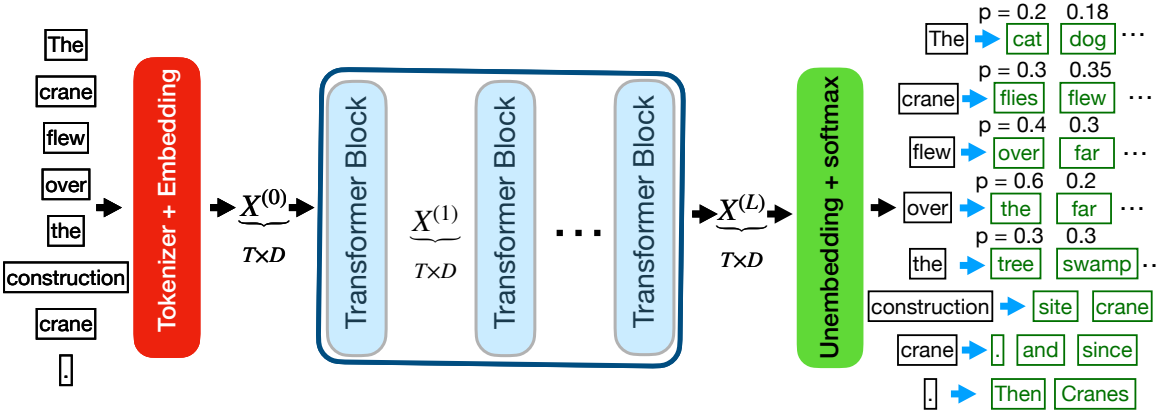
GPT2-like Decoder Model



Tokenizer + Embedding : $\Sigma^* \rightarrow \mathbb{R}^{T \times D}$ string to sequence embedding

Transformer Block : $\mathbb{R}^{T \times D} \rightarrow \mathbb{R}^{T \times D}$ sequence embeddings

GPT2-like Decoder Model

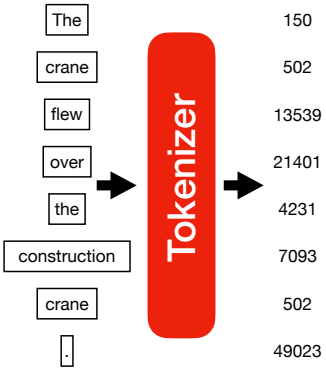


Tokenizer + Embedding : $\Sigma^* \to \mathbb{R}^{T \times D}$ string to sequence embedding

Transformer Block : $\mathbb{R}^{T \times D} \to \mathbb{R}^{T \times D}$ sequence embeddings

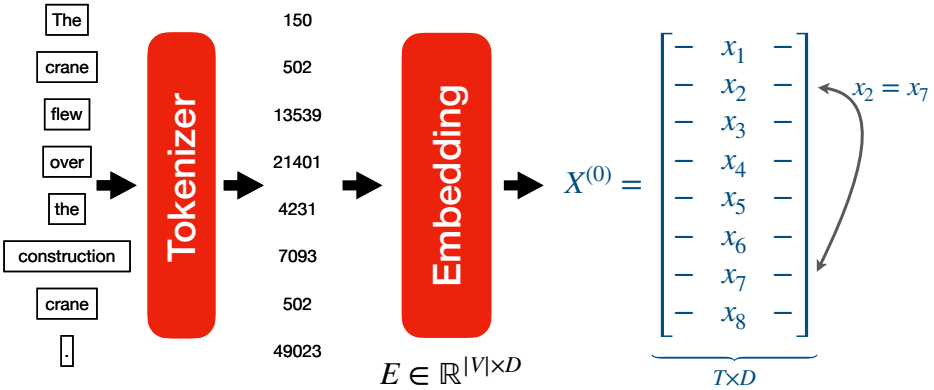
Unembedding + Softmax : $\mathbb{R}^{T \times D} \to \mathbb{R}^{T \times |V|}$ probability over token vocabulary

Part 1: Tokenizer + Embedding



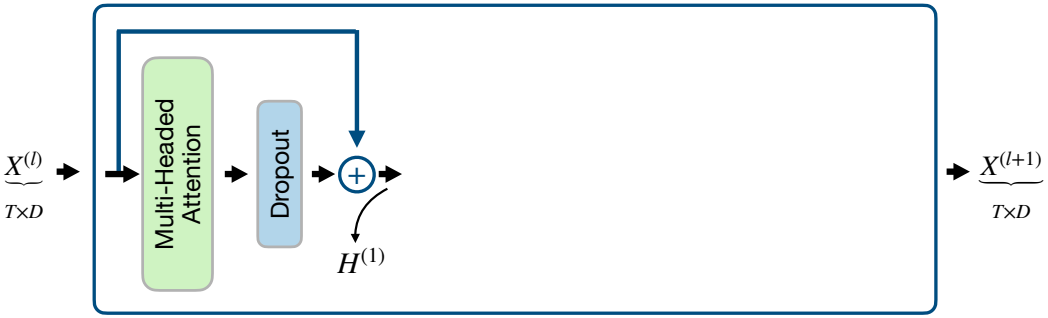
$$\text{Tokenizer} : \Sigma^* \rightarrow [1,\ldots,|V|]^T$$

Part 1: Tokenizer + Embedding



Tokenizer : $\Sigma^* \to [1,\ldots,|V|]^T$ Embedding : $[1,\ldots,|V|]^T \to \mathbb{R}^{T \times D}$

Part 2: Transformer (Decoder) Block



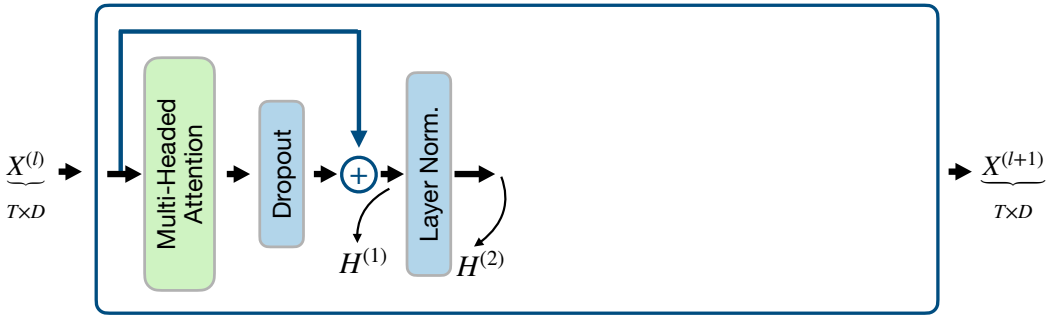
$\text{MHA} : \mathbb{R}^{T \times D} \to \mathbb{R}^{T \times D}$ Multi-Headed Attention

$\text{Dropout} : \mathbb{R}^{T \times D} \to \mathbb{R}^{T \times D}$ Set each entry to zero with prob. p

$\text{Residual connection} \oplus : \mathbb{R}^{T \times D} \to \mathbb{R}^{T \times D}$ Addition with the input

$$H^{(1)} = \text{Dropout}_p\left(\text{MHA}(X^{(l)})\right) + X^{(l)} \in \mathbb{R}^{T \times D}$$

Part 2: Transformer (Decoder) Block



$\mathrm{LN} : \mathbb{R}^{T\times D} \to \mathbb{R}^{T\times D}$

Layer Norm. forces each token embedding (not the sequence) to have zero mean and st. dev 1

$$\underbrace{H^{(2)}}_{T\times D} = \mathrm{LN}_\varepsilon\left(H^{(1)}\right) = \left(\frac{H^{(1)} - \mathbf{1}^\top \mu_{[1:T]}}{\sqrt{\varepsilon + \mathbf{1}^\top \sigma^2_{[1:T]}}}\right) \cdot \gamma + \beta$$

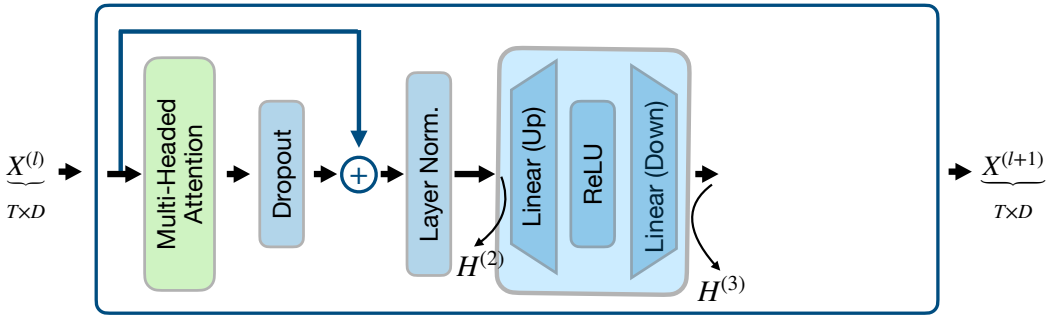
means: $\mu_{[1:T]} \in \mathbb{R}^T$

variances: $\sigma^2_{[1:T]} \in \mathbb{R}^T$

$\varepsilon = 10^{-6}$

learnable parameters: γ, β

Part 2: Transformer (Decoder) Block



$\text{FFN} : \mathbb{R}^{T \times D} \to \mathbb{R}^{T \times D}$ Feed Forward Network

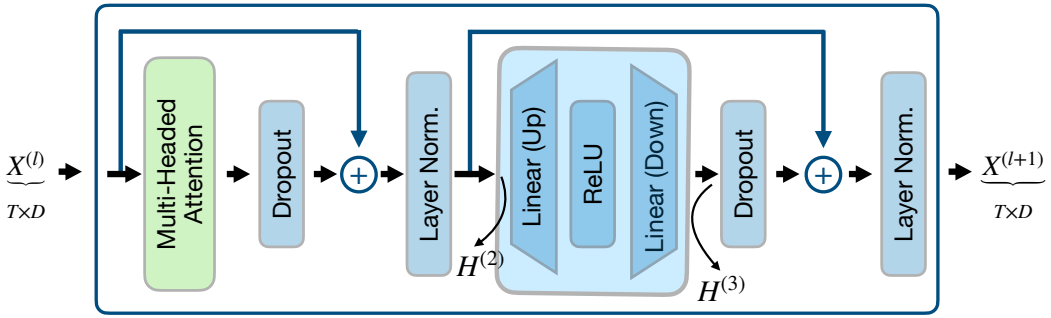
$$H^{(3)} = \text{FFN}(H^{(2)}) = \text{ReLU}(H^{(2)} W_1 + \mathbf{1} b_1^T) W_2 + \mathbf{1} b_2^T$$

$W_1 \in \mathbb{R}^{D \times 4D}$, $b_1 \in \mathbb{R}^{4D}$

$\text{ReLU}(x) = \max\{0, x\}$

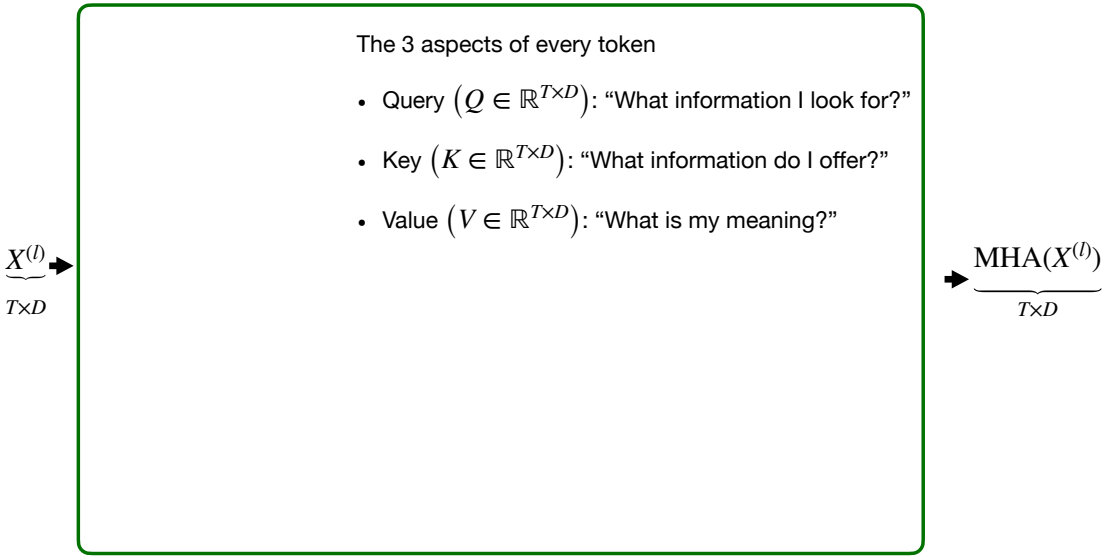
$W_2 \in \mathbb{R}^{4D \times D}$, $b_2 \in \mathbb{R}^{D}$

Part 2: Transformer (Decoder) Block



$$X^{(l+1)} = \mathrm{LN}_{\varepsilon}\left(\mathrm{Dropout}_{p}\left(H^{(3)}\right) + H^{(2)}\right) \in \mathbb{R}^{T \times D}$$

Part 2: Multi-Headed Attention (MHA) with causal mask



Part 2: Multi-Headed Attention (MHA) with causal mask

$$\underbrace{X^{(l)}}_{T\times D}$$

$$X^{(l)}W_K \to K = \underbrace{\begin{bmatrix} - & k_1 & - \\ - & k_2 & - \\ & \vdots & \\ - & k_T & - \end{bmatrix}}_{T\times D}$$

$$X^{(l)}W_Q \to Q = \underbrace{\begin{bmatrix} - & q_1 & - \\ - & q_2 & - \\ & \vdots & \\ - & q_T & - \end{bmatrix}}_{T\times D}$$

$$X^{(l)}W_V \to V = \underbrace{\begin{bmatrix} - & v_1 & - \\ - & v_2 & - \\ & \vdots & \\ - & v_T & - \end{bmatrix}}_{T\times D}$$

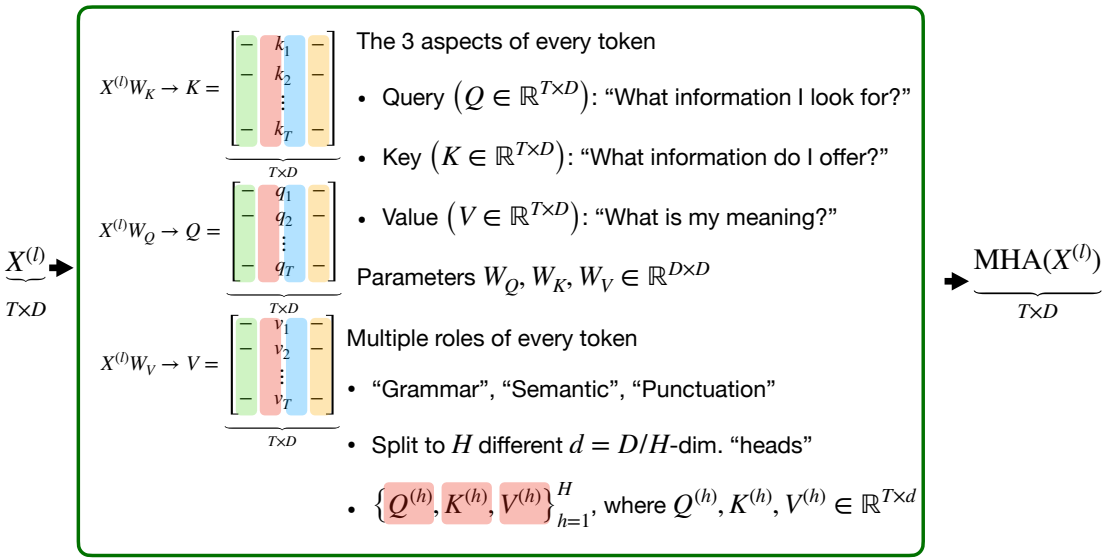
The 3 aspects of every token

- Query $\left(Q \in \mathbb{R}^{T\times D}\right)$: "What information I look for?"
- Key $\left(K \in \mathbb{R}^{T\times D}\right)$: "What information do I offer?"
- Value $\left(V \in \mathbb{R}^{T\times D}\right)$: "What is my meaning?"

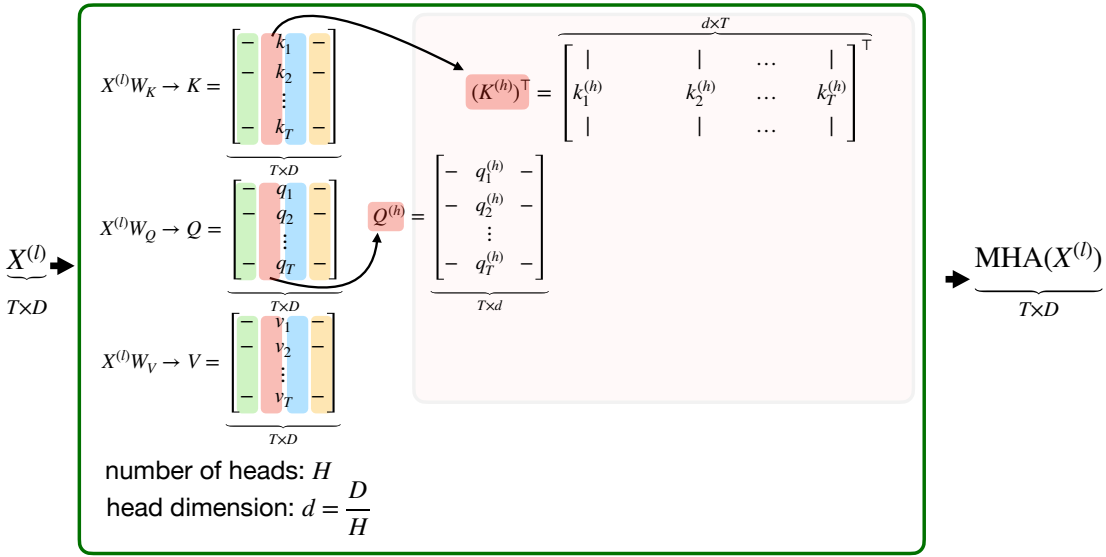
Parameters $W_Q, W_K, W_V \in \mathbb{R}^{D\times D}$

$$\underbrace{\mathrm{MHA}(X^{(l)})}_{T\times D}$$

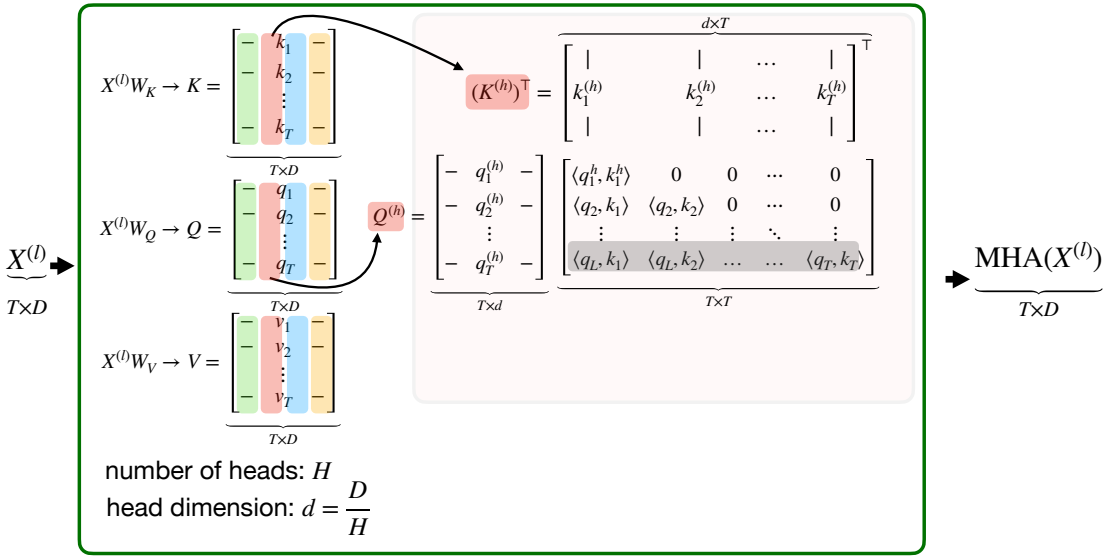
Part 2: Multi-Headed Attention (MHA) with causal mask



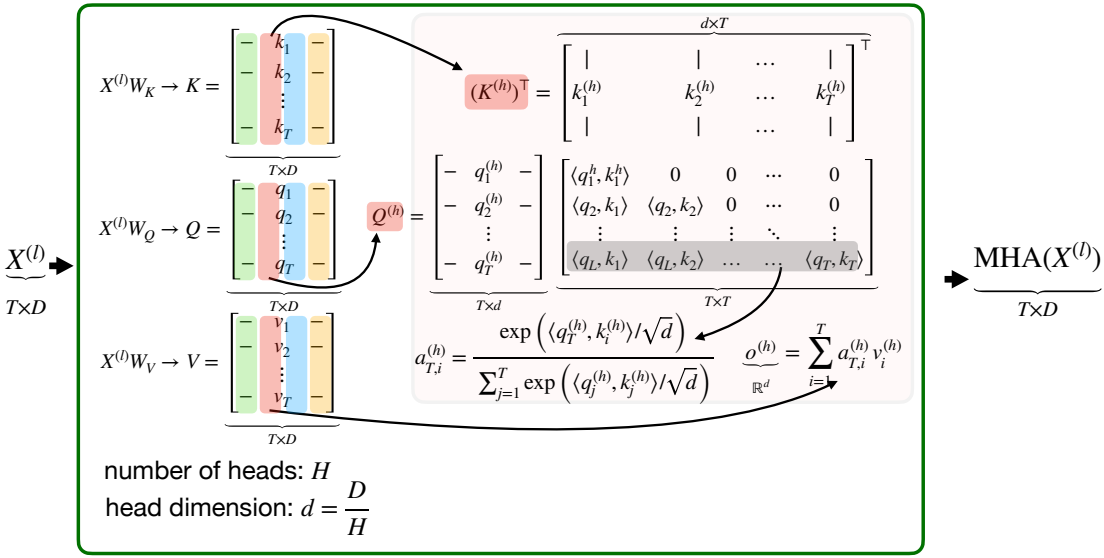
Part 2: Multi-Headed Attention (MHA) with causal mask



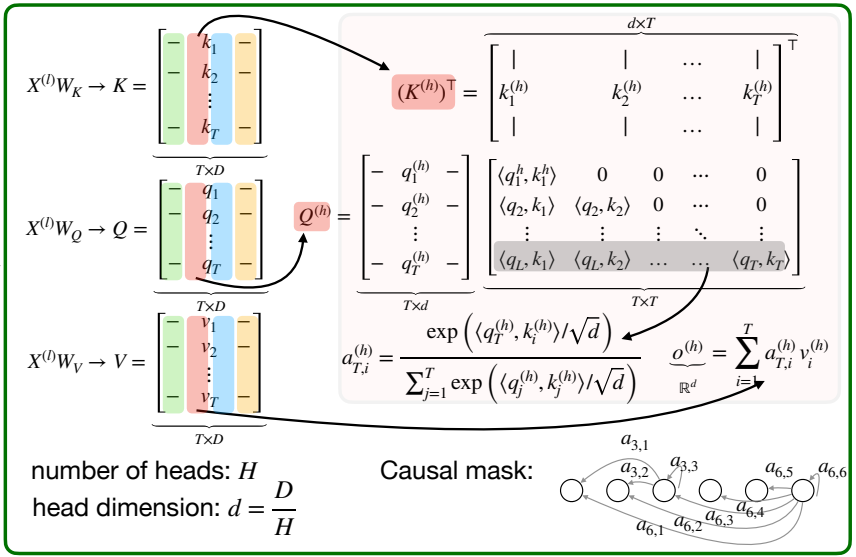
Part 2: Multi-Headed Attention (MHA) with causal mask



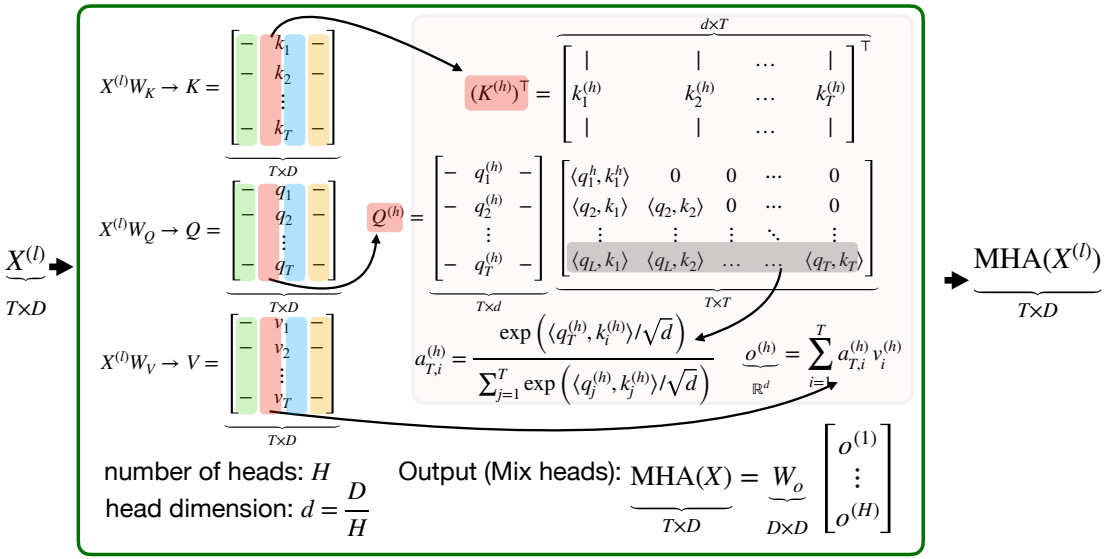
Part 2: Multi-Headed Attention (MHA) with causal mask



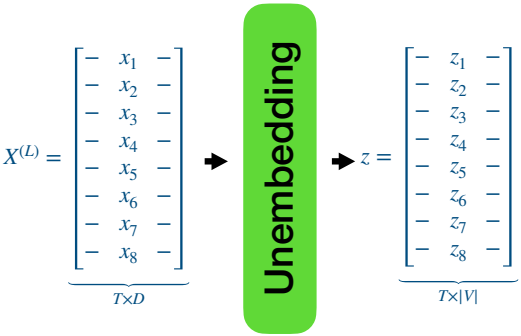
Part 2: Multi-Headed Attention (MHA) with causal mask



Part 2: Multi-Headed Attention (MHA) with causal mask

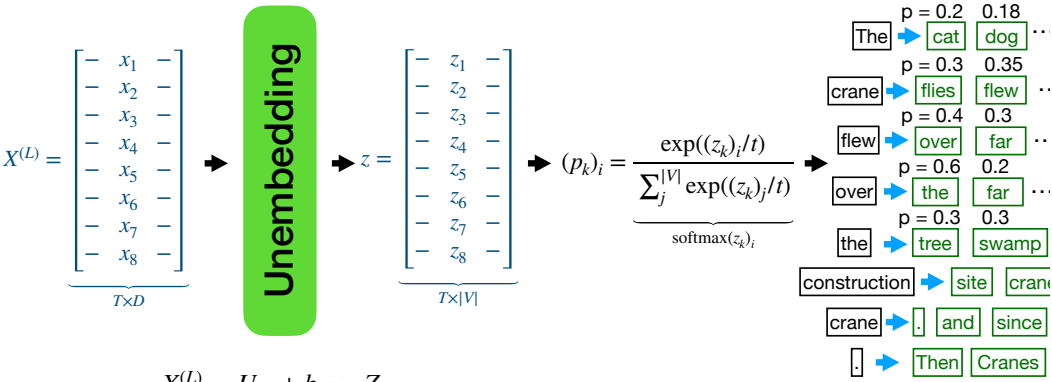


Part 3: Unembedding + softmax + training



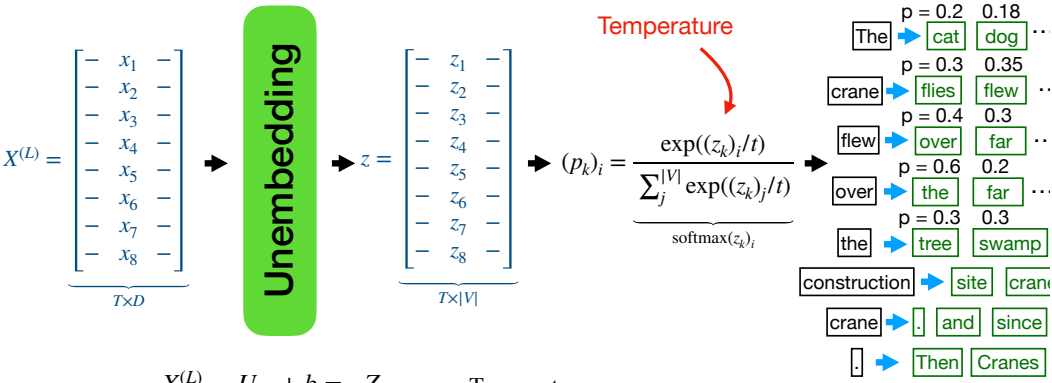
$$\underbrace{X^{(L)}}_{T \times D} \cdot \underbrace{U}_{D \times |V|} + b = \underbrace{Z}_{T \times |V|}$$

Part 3: Unembedding + softmax + training



$$\underbrace{X^{(L)}}_{T \times D} \cdot \underbrace{U}_{D \times |V|} + b = \underbrace{Z}_{T \times |V|}$$

Part 3: Unembedding + softmax + training

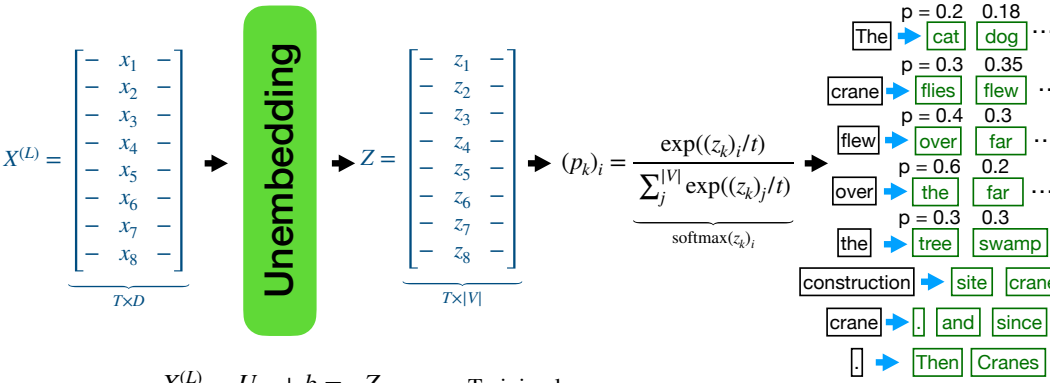


$$\underbrace{X^{(L)}}_{T\times D} \cdot \underbrace{U}_{D\times|V|} + b = \underbrace{Z}_{T\times|V|}$$

Temperature:

- t < 1: logits become larger relative to each other → sharper distribution → more deterministic.
- t > 1: logits shrink → softer distribution → more random.

Part 3: Unembedding + softmax + training



$$\underbrace{X^{(L)}}_{T\times D} \cdot \underbrace{U}_{D\times|V|} + b = \underbrace{Z}_{T\times|V|}$$

Training loss:

$$L(\theta) = -\sum_{t=1}^{7} \log p_\theta(x_{t+1} \mid x_{1:t}) = -\sum_{t=1}^{7} \log(p_k)_{x_{k+1}}$$

Architecture choices

Architecture choices

Common architecture variations

- Position Embeddings
- Activations, dimensions
- Normalization
- **Next time:** Attention variants

Architecture choices

Common architecture variations

- Position Embeddings
- Activations, dimensions
- Normalization
- **Next time:** Attention variants

Aim to understand:

- What should be the size of the hidden layers?
- How large the vocabulary should be?
- How stable is the training?

Architecture choices

Common architecture variations

- Position Embeddings
- Activations, dimensions
- Normalization
- **Next time:** Attention variants

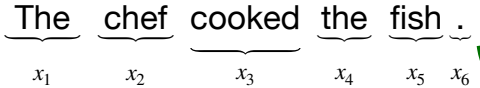
Aim to understand:

- What should be the size of the hidden layers?
- How large the vocabulary should be?
- How stable is the training?

Source: CS336: Lecture 3 on Architectures

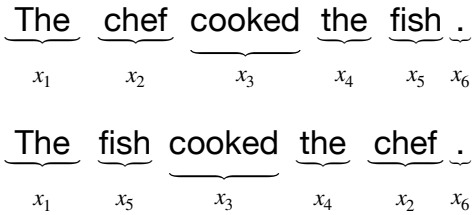
Position Embeddings

Position embedding: Absolute (learned) embedding



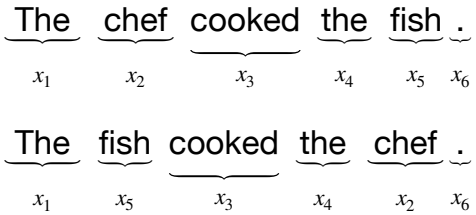
$$o_6 = \sum_{i=1}^{6} \frac{\exp\left(\langle q_6, k_i \rangle / \sqrt{d}\right)}{\sum_{j=1}^{6} \exp\left(\langle q_j, k_j \rangle / \sqrt{d}\right)} \cdot v_i$$

Position embedding: Absolute (learned) embedding



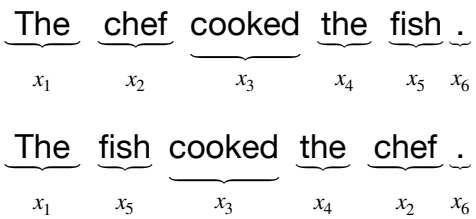
$$o_6 = \sum_{i=1}^{6} \frac{\exp\left(\langle q_6, k_i \rangle / \sqrt{d}\right)}{\sum_{j=1}^{6} \exp\left(\langle q_j, k_j \rangle / \sqrt{d}\right)} \cdot v_i$$

Position embedding: Absolute (learned) embedding

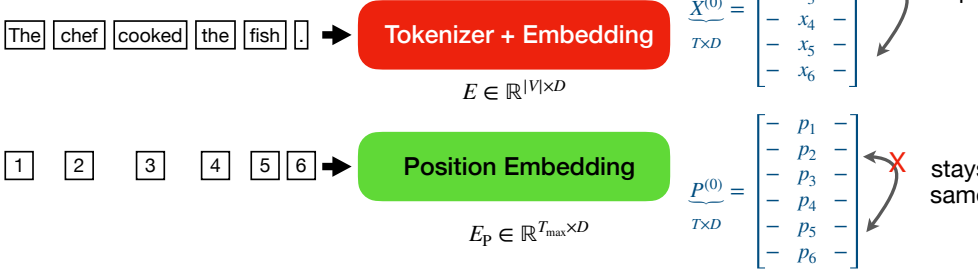


$$o_6 = \sum_{i=1}^{6} \frac{\exp\left(\langle q_6, k_i \rangle / \sqrt{d}\right)}{\sum_{j=1}^{6} \exp\left(\langle q_j, k_j \rangle / \sqrt{d}\right)} \cdot v_i$$

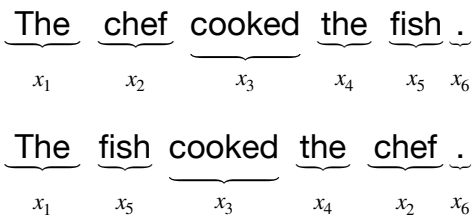
Position embedding: Absolute (learned) embedding



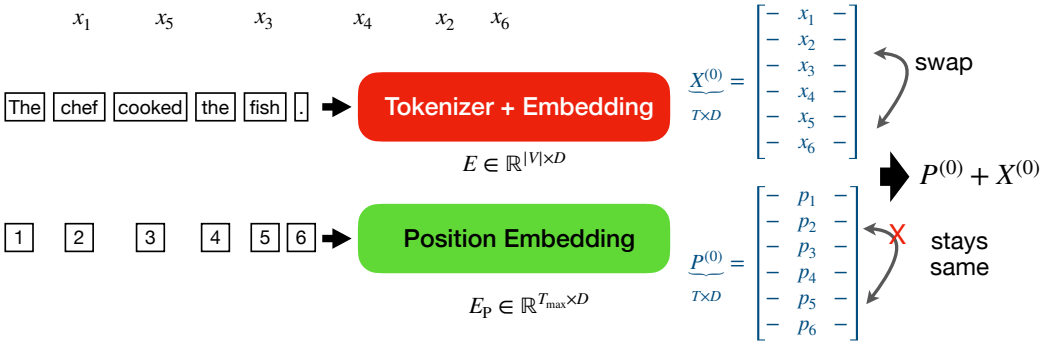
$$o_6 = \sum_{i=1}^{6} \frac{\exp\left(\langle q_6, k_i \rangle / \sqrt{d}\right)}{\sum_{j=1}^{6} \exp\left(\langle q_j, k_j \rangle / \sqrt{d}\right)} \cdot v_i$$



Position embedding: Absolute (learned) embedding



$$o_6 = \sum_{i=1}^{6} \frac{\exp\left(\langle q_6, k_i \rangle / \sqrt{d}\right)}{\sum_{j=1}^{6} \exp\left(\langle q_j, k_j \rangle / \sqrt{d}\right)} \cdot v_i$$



Variations of position embeddings

- **Absolute (learned) embeddings:**
  - Add a position vector (learned) to the embedding

Variations of position embeddings

- **Absolute (learned) embeddings:**
  - Add a position vector (learned) to the embedding
- **Sinusoidal embeddings:**
  - Add a position vector based on a frequency of position

Variations of position embeddings

- **Absolute (learned) embeddings:**
  - Add a position vector (learned) to the embedding
- **Sinusoidal embeddings:**
  - Add a position vector based on a frequency of position
- **Relative embeddings:**
  - Add a position vector *inside* of the attention computation

Variations of position embeddings

- **Absolute (learned) embeddings:**
  - Add a position vector (learned) to the embedding
- **Sinusoidal embeddings:**
  - Add a position vector based on a frequency of position
- **Relative embeddings:**
  - Add a position vector *inside* of the attention computation

Want: Inner product invariant to *absolute position* shifts

The constructed pos. embedding $f : \mathbb{R}^D \times \mathbb{N} \to \mathbb{R}^D$ has

$$\langle f(x_1, i), f(x_2, j) \rangle = g(x_1, x_2, i - j) \qquad \text{for some } g : \mathbb{R}^D \times \mathbb{R}^D \times \mathbb{N} \to \mathbb{R}.$$

- The inner product depends only on:
  - token embeddings x_1, x_2
  - relative distance $i - j$

RoPE: Rotary Position Embeddings **(Su et al., 2021)**

Idea: Rotation preserves inner products

For positions $p \in \{1, \ldots, T_{\max}\}$, if we have an orthogonal matrices $\{R_p\}_{p=1}^{T_{\max}}$, s.t.,

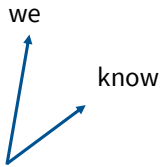
$$R_p^\top = R_{-p} \quad \text{and} \quad R_m^\top R_n = R_{n-m} \quad \Longrightarrow \quad \langle R_m x_1, R_n x_2 \rangle = \langle x_1, R_{n-m} x_2 \rangle.$$

RoPE: Rotary Position Embeddings **(Su et al., 2021)**

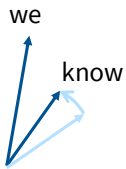
Idea: Rotation preserves inner products

For positions $p \in \{1, \ldots, T_{\max}\}$, if we have an orthogonal matrices $\{R_p\}_{p=1}^{T_{\max}}$, s.t.,

$$R_p^\top = R_{-p} \quad \text{and} \quad R_m^\top R_n = R_{n-m} \quad \Longrightarrow \quad \langle R_m x_1, R_n x_2 \rangle = \langle x_1, R_{n-m} x_2 \rangle.$$

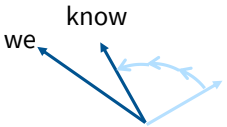


Position independent embedding



Embedding
“we know that”

Rotate we by ‘0 positions’
know by ‘1 positions’



Embedding
“of course we know”

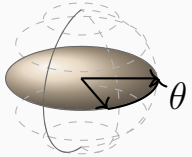
Rotate we by ‘2 positions’
Rotate know by ‘3 positions’

From CS336: Lecture 3 on Architectures

RoPE: A suitable family of rotations in $\mathbb{R}^D$

Rotation in 2D subspaces ("Givens rotation")

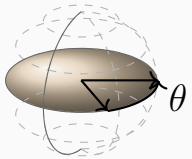
$$G(i, i+1, \theta) = \begin{pmatrix} I_{i-1} & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & I_{D-i-1} \end{pmatrix}$$



RoPE: A suitable family of rotations in $\mathbb{R}^D$

Rotation in 2D subspaces ("Givens rotation")

$$G(i, i+1, \theta) = \begin{pmatrix} I_{i-1} & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & I_{D-i-1} \end{pmatrix}$$



RoPE

Let $\theta_i = 10\,000^{-2i/D}$ (high i rotate slower)

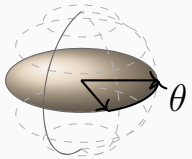
$$R_{\theta,D}(p) = \prod_{i=1}^{D/2} G(i, i+1, p\theta_i),$$

$$f(x, p) = R_{\theta,D}(p)\, x$$

RoPE: A suitable family of rotations in $\mathbb{R}^D$

Rotation in 2D subspaces ("Givens rotation")

$$G(i, i+1, \theta) = \begin{pmatrix} I_{i-1} & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & I_{D-i-1} \end{pmatrix}$$



RoPE

Let $\theta_i = 10\,000^{-2i/D}$ (high i rotate slower)

$$R_{\theta,D}(p) = \prod_{i=1}^{D/2} G(i, i+1, p\theta_i),$$

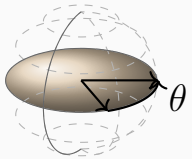
$$f(x, p) = R_{\theta,D}(p)\, x$$

$$R_{\theta,D}(p) = \begin{pmatrix} \cos(p\theta_1) & -\sin(p\theta_1) & & \\ \sin(p\theta_1) & \cos(p\theta_1) & & \\ & & \cos(p\theta_2) & -\sin(p\theta_2) \\ & & \sin(p\theta_2) & \cos(p\theta_2) \\ & & & & \ddots \end{pmatrix}$$

RoPE: A suitable family of rotations in $\mathbb{R}^D$

Rotation in 2D subspaces ("Givens rotation")

$$G(i, i+1, \theta) = \begin{pmatrix} I_{i-1} & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & I_{D-i-1} \end{pmatrix}$$



RoPE

Let $\theta_i = 10\,000^{-2i/D}$ (high i rotate slower)

$$R_{\theta,D}(p) = \prod_{i=1}^{D/2} G(i, i+1, p\theta_i),$$

$$f(x, p) = R_{\theta,D}(p)\, x$$

$$R_{\theta,D}(p) = \begin{pmatrix} \cos(p\theta_1) & -\sin(p\theta_1) & & \\ \sin(p\theta_1) & \cos(p\theta_1) & & \\ & & \cos(p\theta_2) & -\sin(p\theta_2) \\ & & \sin(p\theta_2) & \cos(p\theta_2) \\ & & & & \ddots \end{pmatrix}$$

- Can be viewed as sampled Fourier phases of a Dirac delta (point mass) at position p.

RoPE: Where it is used?

Where it is used?

For hidden states $X \in \mathbb{R}^{T \times D}$, first compute

$$Q = XW_Q, \qquad K = XW_K, \qquad V = XW_V.$$

Then, in each attention head and at each position t,

$$\widetilde{q}_t = R_{\theta,d_h}(t)\, q_t, \qquad \widetilde{k}_t = R_{\theta,d_h}(t)\, k_t, \qquad \widetilde{v}_t = v_t.$$

Effect on attention scores

$$a_{mn} = \frac{\widetilde{q}_m^\top \widetilde{k}_n}{\sqrt{d_h}} = \frac{q_m^\top R_{\theta,d_h}(n-m)\, k_n}{\sqrt{d_h}}.$$

- RoPE is applied **after the** Q, K, V **projections**, separately in **each head**.
- **High-frequency** components vary quickly with position $\Rightarrow$ capture **local** information.
- **Low-frequency** components vary slowly with position $\Rightarrow$ capture **global / long-range**.

RoPE: Frequency experiment

RoPE: Frequency experiment

Normalization / Stability

Why normalization?

- Representations can grow across many Transformer blocks.

Why normalization?

- Representations can grow across many Transformer blocks.
- Large activation scales makes optimization unstable.

Why normalization?

- Representations can grow across many Transformer blocks.
- Large activation scales makes optimization unstable.
- Normalization keeps representations on a controlled scale.

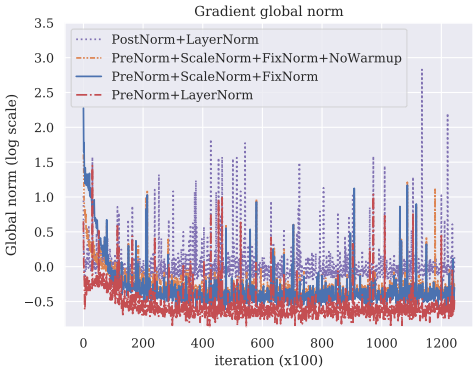
Why normalization?

- Representations can grow across many Transformer blocks.
- Large activation scales makes optimization unstable.
- Normalization keeps representations on a controlled scale.
- Improve:
  - gradient flow,
  - training stability,
  - depth scalability.

Why normalization?

- Representations can grow across many Transformer blocks.
- Large activation scales makes optimization unstable.
- Normalization keeps representations on a controlled scale.
- Improve:
  - gradient flow,
  - training stability,
  - depth scalability.

Main role in Transformers Keep each sublayer input well-conditioned before attention / FFN computations.



Pre-norm vs post-norm

Post-norm (original Transformer)

$$x_{l+1} = \mathrm{Norm}\big(x_l + F(x_l)\big)$$

- *After* the addition and MHA (or FFN).
- Harder to optimize in very deep Transformers.

Pre-norm vs post-norm

Post-norm (original Transformer)

$$x_{l+1} = \mathrm{Norm}\big(x_l + F(x_l)\big)$$

- *After* the addition and MHA (or FFN).
- Harder to optimize in very deep Transformers.

Pre-norm (modern LLMs)

$$x_{l+1} = x_l + F(\mathrm{Norm}(x_l))$$

- *Before* MHA (or FFN).
- Cleaner residual path for gradients.
- More stable for deep, large-scale training.

Pre-norm vs post-norm

Post-norm (original Transformer)

$$x_{l+1} = \mathrm{Norm}\big(x_l + F(x_l)\big)$$

- *After* the addition and MHA (or FFN).
- Harder to optimize in very deep Transformers.

Pre-norm (modern LLMs)

$$x_{l+1} = x_l + F(\mathrm{Norm}(x_l))$$

- *Before* MHA (or FFN).
- Cleaner residual path for gradients.
- More stable for deep, large-scale training.

Most modern LLMs use **pre-norm** because it trains more reliably at scale.

LayerNorm vs RMSNorm

LayerNorm

Normalizes both mean and variance across features:

$$\mathrm{LayerNorm}(x) = \gamma \odot \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \varepsilon}} + \beta$$

- centers and rescales + learned shift β,

RMSNorm

Normalizes only the root-mean-square:

$$\mathrm{RMSNorm}(x) = \gamma \odot \frac{x}{\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \varepsilon}}$$

- does *not* subtract the mean,
- simpler and cheaper, widely used in modern LLMs.

Typical examples (LayerNorm):
Original Transformer, BERT, GPT-2/3, OPT

Typical examples (RMSNorm):
T5, PaLM, LLaMA-family, Gemma

Activation functions: ReLU, GeLU, Swish, ELU, GLU, GeGLU, ReGLU, SeLU, SwiGLU, LiGLU...

Activation functions in Transformer FFNs

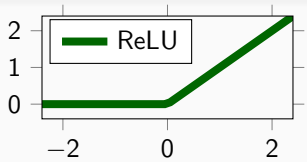
ReLU, GeLU, Swish, ELU, GLU, GeGLU, ReGLU, SeLU, SwiGLU, LiGLU...

The choice of activation in Transformer FFNs strongly affects expressivity, optimization, and performance.

Common activations: ReLU → GeLU

ReLU (Rectified Linear Unit)

$$\mathrm{FFN}(x) = \max\{0, xW_1\}W_2$$

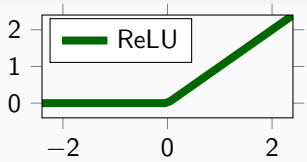


Models: Original Transformer, T5, Gopher, Chinchilla, OPT ...

Common activations: ReLU → GeLU

ReLU (Rectified Linear Unit)

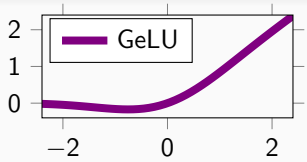
$$\mathrm{FFN}(x) = \max\{0, xW_1\}W_2$$



Models: Original Transformer, T5, Gopher, Chinchilla, OPT ...

GeLU (Gaussian Error Linear Unit)

$$\mathrm{FFN}(x) = \mathrm{GeLU}(xW_1)W_2$$
$$\mathrm{GeLU}(x) = x\Phi(x)$$
$$\Phi(x) \coloneqq \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$$

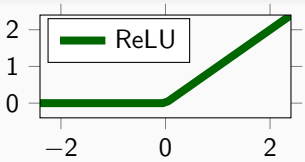


Models: GPT1/2/3, GPTJ, BERT, ...

Common activations: ReLU → GeLU

ReLU (Rectified Linear Unit)

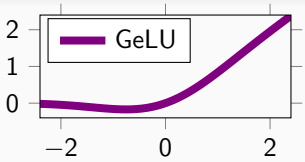
$$\mathrm{FFN}(x) = \max\{0, xW_1\}W_2$$



Models: Original Transformer, T5, Gopher, Chinchilla, OPT ...

GeLU (Gaussian Error Linear Unit)

$$\mathrm{FFN}(x) = \mathrm{GeLU}(xW_1)W_2$$
$$\mathrm{GeLU}(x) = x\Phi(x)$$
$$\Phi(x) \coloneqq \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$$



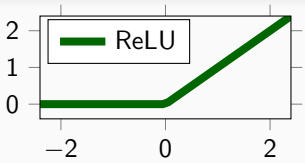
Models: GPT1/2/3, GPTJ, BERT, ...

- **Smoother non-linearity:** no non-smooth kink at 0.

Common activations: ReLU → GeLU

ReLU (Rectified Linear Unit)

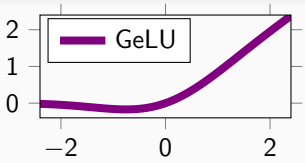
$$\mathrm{FFN}(x) = \max\{0, xW_1\}W_2$$



Models: Original Transformer, T5, Gopher, Chinchilla, OPT ...

GeLU (Gaussian Error Linear Unit)

$$\mathrm{FFN}(x) = \mathrm{GeLU}(xW_1)W_2$$
$$\mathrm{GeLU}(x) = x\Phi(x)$$
$$\Phi(x) \coloneqq \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$$



Models: GPT1/2/3, GPTJ, BERT, ...

- **Smoother non-linearity:** no non-smooth kink at 0.
- **Soft-gating:** preserves information of small negative activations (instead of setting them 0).

Gated activations (Gated Linear Units, *GLU)

Idea: Separate "gating" from the linear transform

$$\max\{0, xW_1\} \quad \rightarrow \quad \max\{0, \underbrace{xW_1}_{\text{map for gating}}\} \otimes \underbrace{(xV)}_{\text{linear map on } x}$$

$$\mathrm{FFN}_{\mathrm{ReGLU}}(x) = (\max\{0, xW_1\} \otimes (xV))\, W_2$$

Gated activations (Gated Linear Units, *GLU)

Idea: Separate "gating" from the linear transform

$$\max\{0, xW_1\} \quad \rightarrow \quad \max\{0, \underbrace{xW_1}_{\text{map for gating}}\} \otimes \underbrace{(xV)}_{\text{linear map on } x}$$

$$\mathrm{FFN}_{\mathrm{ReGLU}}(x) = (\max\{0, xW_1\} \otimes (xV))\, W_2$$

GeGLU (=GeLU + GLU)

$$\mathrm{FFN}_{\mathrm{GeGLU}}(x) = (\mathrm{GeLU}(xW_1) \otimes (xV))\, W_2$$
$$\mathrm{GeLU}(x) = x\Phi(x)$$

Models: T5-v1.1, Gemma, ...

Gated activations (Gated Linear Units, *GLU)

Idea: Separate "gating" from the linear transform

$$\max\{0, xW_1\} \quad \rightarrow \quad \max\{0, \underbrace{xW_1}_{\text{map for gating}}\} \otimes \underbrace{(xV)}_{\text{linear map on } x}$$

$$\mathrm{FFN}_{\mathrm{ReGLU}}(x) = (\max\{0, xW_1\} \otimes (xV))\, W_2$$

GeGLU (=GeLU + GLU)

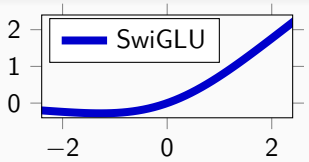
$$\mathrm{FFN}_{\mathrm{GeGLU}}(x) = (\mathrm{GeLU}(xW_1) \otimes (xV))\, W_2$$
$$\mathrm{GeLU}(x) = x\Phi(x)$$

Models: T5-v1.1, Gemma, ...

SwiGLU (Swish Gated Linear Unit)

$\mathrm{Swish}(x) = x\,\mathrm{sigmoid}(x)$, where $\mathrm{sigmoid}(x) = 1/(1+e^{-x})$

$$\mathrm{FFN}_{\mathrm{SwiGLU}}(x) = (\mathrm{Swish}(xW_1) \otimes (xV))\, W_2$$



Models: LLaMA, PaLM, Mistral, ...

Comparisons / Summary (**Shazeer 2020**)

See https://arxiv.org/abs/2002.05202

Shapes / Depth

FFN size to D ratio

Within the Transformer block:

- **Model Dimension** (D): size of the residual stream and token embeddings.
- **Feedforward Dimension** (D_{FFN}): size of the intermediate hidden layer inside the FFN.

The Consensus Rule

Expansion factor (up-projection then down-projection) that is almost universal:

$$D_{\mathrm{FFN}} = 4 \times D$$

FFN size to D ratio

Within the Transformer block:

- **Model Dimension** (D): size of the residual stream and token embeddings.
- **Feedforward Dimension** (D_{FFN}): size of the intermediate hidden layer inside the FFN.

The Consensus Rule

Expansion factor (up-projection then down-projection) that is almost universal:

$$D_{\text{FFN}} = 4 \times D$$

Exception: GLU Variants

GLU variants use an extra gating projection. To maintain the same parameter count as a standard FFN, the hidden dimension is scaled down by 2/3:

$$D_{\text{FFN}} = \frac{8}{3} \times D \approx 2.67 \times D$$

Notable Ratios: LLaMA/Qwen/DeepSeek (∼2.67), Mistral/LLaMA-2 (3.5), PaLM (4.0).

Attention dimension to head dimension ratio

Within the Multi-Head Attention mechanism, we have:

number of heads (H), **head dimension** (d), **model dimension** (D).

The Consensus Rule

The concatenated dimension of all heads typically equals the model dimension (a ratio of 1):

$$H \times d = D \implies \frac{H \times d}{D} = 1$$

Attention dimension to head dimension ratio

Within the Multi-Head Attention mechanism, we have:
number of heads (H), **head dimension** (d), **model dimension** (D).

The Consensus Rule

The concatenated dimension of all heads typically equals the model dimension (a ratio of 1):

$$H \times d = D \implies \frac{H \times d}{D} = 1$$

Exceptions

Most models (e.g., GPT-3, LLaMA-2, T5 v1.1) have 1:1 ratio

- **Notable Exceptions:**
  Some Google models deviate (e.g., T5 has a ratio of 16, LaMDA is 2, PaLM is $\sim$1.5).
- **Bhojanapalli et al. (2020):**
  argued against the 1:1 ratio due to potential "low rank bottlenecks" but empirical evidence shows this is rare.

Aspect Ratios: Depth vs. Width

How many layers of Transformer blocks (L) vs. model dimension (D)

- **Consensus ($\frac{D}{L} \sim$ 100–130):**
  - GPT-3, OPT, Mistral, and Qwen all use **128**
  - LLaMA, LLaMA-2, and Chinchilla use **102**
- **Others:** GPT-2 (33) or T5-11B (43) much deeper. BLOOM (205) is much wider.

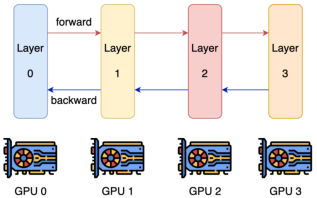
Aspect Ratios: Depth vs. Width

How many layers of Transformer blocks (L) vs. model dimension (D)

- **Consensus ($\frac{D}{L} \sim$ 100–130):**
  - GPT-3, OPT, Mistral, and Qwen all use **128**
  - LLaMA, LLaMA-2, and Chinchilla use **102**
- **Others:** GPT-2 (33) or T5-11B (43) much deeper. BLOOM (205) is much wider.

Why not infinitely deep? **(Tay et al., 2021)**

- **Latency:** Hard to parallelize ⇒ Very deep models have higher latency.
- **Width Advantage:** Naively parallelizable across devices.



Other modern architectural choices

- **Dropout often removed**
  - Modern pre-training on massive datasets
  - Overfitting is less of a concern

Other modern architectural choices

- **Dropout often removed**
  - Modern pre-training on massive datasets
  - Overfitting is less of a concern
- **Biases often removed in FFNs**
  - Use $y = Wx$ instead of $y = Wx + b$
  - Slightly fewer parameters and a small throughput gain
  - Usually little or no loss in quality

Other modern architectural choices

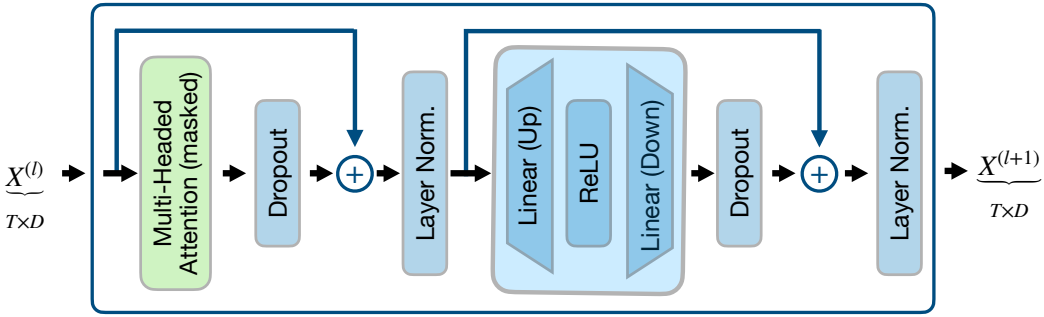
- **Dropout often removed**
  - Modern pre-training on massive datasets
  - Overfitting is less of a concern
- **Biases often removed in FFNs**
  - Use $y = Wx$ instead of $y = Wx + b$
  - Slightly fewer parameters and a small throughput gain
  - Usually little or no loss in quality
- **Weight decay is still used**
  - Even if dropout disappears, regularization does not
  - Weight decay helps control weight growth and stabilize training

Ablations **(Kaplan et al., 2020)**

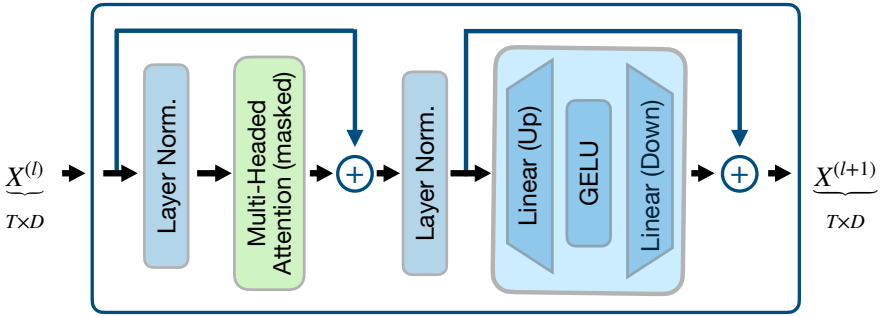
See https://arxiv.org/abs/2001.08361

Parallel stream (Optional)

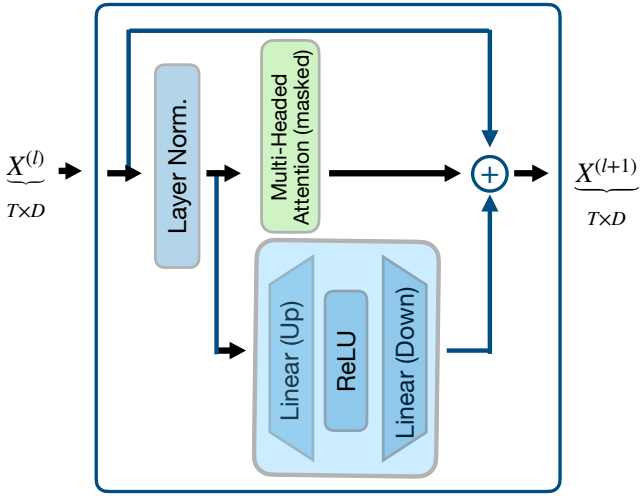
Parallel the residual stream



Parallel the residual stream



Parallel the residual stream



Takeaways

Takeaways: Modern Transformer Decoder Block

- **Normalization placement:** Post-Norm $\Rightarrow$ **Pre-Norm**
  - Normalizes inputs *before* sublayers $\Rightarrow$ better training stability for deep networks.

Takeaways: Modern Transformer Decoder Block

- **Normalization placement:** Post-Norm $\Rightarrow$ **Pre-Norm**
  - Normalizes inputs *before* sublayers $\Rightarrow$ better training stability for deep networks.
- **Normalization type:** Layer Norm $\Rightarrow$ **RMSNorm**
  - Removes the mean-centering $\Rightarrow$ computationally cheaper while maintaining model quality.

Takeaways: Modern Transformer Decoder Block

- **Normalization placement:** Post-Norm $\Rightarrow$ **Pre-Norm**
  - Normalizes inputs *before* sublayers $\Rightarrow$ better training stability for deep networks.
- **Normalization type:** Layer Norm $\Rightarrow$ **RMSNorm**
  - Removes the mean-centering $\Rightarrow$ computationally cheaper while maintaining model quality.
- **Activations in Feed-Forward (FFN):** ReLU $\Rightarrow$ **SwiGLU**
  - Replaces 2-layer MLP with a 3-layer *gated* linear unit $+ \mathrm{Swish}(x)$.

Takeaways: Modern Transformer Decoder Block

- **Normalization placement:** Post-Norm $\Rightarrow$ **Pre-Norm**
  - Normalizes inputs *before* sublayers $\Rightarrow$ better training stability for deep networks.
- **Normalization type:** Layer Norm $\Rightarrow$ **RMSNorm**
  - Removes the mean-centering $\Rightarrow$ computationally cheaper while maintaining model quality.
- **Activations in Feed-Forward (FFN):** ReLU $\Rightarrow$ **SwiGLU**
  - Replaces 2-layer MLP with a 3-layer *gated* linear unit $+ \mathrm{Swish}(x)$.
- **Parameters:** Biases $\Rightarrow$ **Remove Biases**
  - Bias terms ($+b$) are dropped to save memory and increase throughput.

Takeaways: Modern Transformer Decoder Block

- **Normalization placement:** Post-Norm $\Rightarrow$ **Pre-Norm**
  - Normalizes inputs *before* sublayers $\Rightarrow$ better training stability for deep networks.
- **Normalization type:** Layer Norm $\Rightarrow$ **RMSNorm**
  - Removes the mean-centering $\Rightarrow$ computationally cheaper while maintaining model quality.
- **Activations in Feed-Forward (FFN):** ReLU $\Rightarrow$ **SwiGLU**
  - Replaces 2-layer MLP with a 3-layer *gated* linear unit $+ \mathrm{Swish}(x)$.
- **Parameters:** Biases $\Rightarrow$ **Remove Biases**
  - Bias terms ($+b$) are dropped to save memory and increase throughput.
- **Positional Encoding:** Absolute (Input) $\Rightarrow$ **RoPE** in every head of MHA
  - Injected directly *inside* MHA to model relative distances $\Rightarrow$ improving long-context modelling.