Attention, Transformers, and Generation

ECE 57000 — October 30, 2026

David I. Inouye

Attention chooses content and allowed context

- Learned queries and keys score matches; row softmax weights the value vectors.
- A causal mask excludes future inputs before softmax; training can compute every allowed output together.
- Unmasked attention without position signals reorders its outputs when inputs are reordered.

Today: represent order explicitly, then assemble a complete transformer.

Column vectors become transposed rows in a sequence matrix

$$X = \begin{bmatrix} \boldsymbol{x}_1^\mathsf{T} \\ \vdots \\ \boldsymbol{x}_L^\mathsf{T} \end{bmatrix}, \qquad \boldsymbol{q}_\ell = W_Q \boldsymbol{x}_\ell, \qquad Q = X W_Q^\mathsf{T}$$

Likewise $K = X W_K^\mathsf{T}$ and $V = X W_V^\mathsf{T}$. **Example: four tokens, three features per vector.**

$Q \in \mathbb{R}^{4\times 3}$ — Queries

$\boldsymbol{q}_1^T = [q_{11},\ q_{12},\ q_{13}]$
$\boldsymbol{q}_2^T = [q_{21},\ q_{22},\ q_{23}]$
$\boldsymbol{q}_3^T = [q_{31},\ q_{32},\ q_{33}]$
$\boldsymbol{q}_4^T = [q_{41},\ q_{42},\ q_{43}]$

$K \in \mathbb{R}^{4\times 3}$ — Keys

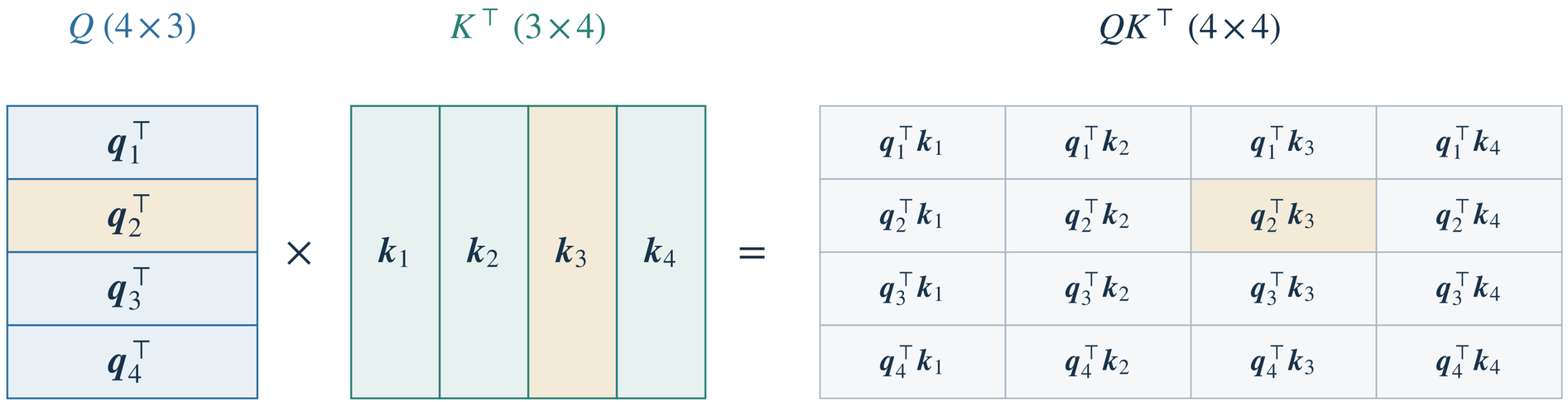
$\boldsymbol{k}_1^T = [k_{11},\ k_{12},\ k_{13}]$
$\boldsymbol{k}_2^T = [k_{21},\ k_{22},\ k_{23}]$
$\boldsymbol{k}_3^T = [k_{31},\ k_{32},\ k_{33}]$
$\boldsymbol{k}_4^T = [k_{41},\ k_{42},\ k_{43}]$

$V \in \mathbb{R}^{4\times 3}$ — Values

$\boldsymbol{v}_1^T = [v_{11},\ v_{12},\ v_{13}]$
$\boldsymbol{v}_2^T = [v_{21},\ v_{22},\ v_{23}]$
$\boldsymbol{v}_3^T = [v_{31},\ v_{32},\ v_{33}]$
$\boldsymbol{v}_4^T = [v_{41},\ v_{42},\ v_{43}]$

Four token positions; each row is the transpose of a three-feature column vector.

Each query–key inner product fills one score-matrix entry



$$(QK^\top)_{23} = \boldsymbol{q}_2^\top \boldsymbol{k}_3 = q_{21}k_{31} + q_{22}k_{32} + q_{23}k_{33}$$

Query 2 matches key 3; the resulting attention weight will multiply value 3.

$$S = \frac{QK^\top}{\sqrt{d_k}}, \qquad A = \mathrm{softmax}_{\mathrm{row}}(S), \qquad a_{\ell j} = \frac{e^{s_{\ell j}}}{\sum_{r=1}^{L} e^{s_{\ell r}}}$$

Dot products give scores; row softmax gives weights.

A sequence needs order beyond an unordered collection

Same words, different order	Requested action
Send the **report before** the slides.	Report first, slides second.
Send the **slides before** the report.	Slides first, report second.

Focus on **report**: its relationship to **before** has changed.

Without a mask or positions, self-attention gives **report the same output vector in both sentences**—just in a different row.

We want token representations to reflect order-dependent relationships.

Which change makes position explicit?

> **A causal attention layer already blocks future inputs.**
> **Which change directly supplies each token's absolute position?**
> **A. Add a position vector to its embedding**
> **B. Add another identical causal mask**
> **C. Increase the number of attention heads**

A. The mask controls access; a position vector explicitly represents location.

Causal masks restrict access; position signals describe location

A fixed causal mask **breaks arbitrary permutation equivariance**: reordering tokens changes which context they may read.

Mechanism	What it supplies
Causal mask	Which earlier/current inputs a query may access
Absolute position signal	Where a token occurs in the sequence
Relative position signal	How far apart two token positions are

The mask does not explicitly encode token indices or pairwise offsets.

Next: add position features, or rotate query/key features using their positions.

Additive positions combine token features and location features

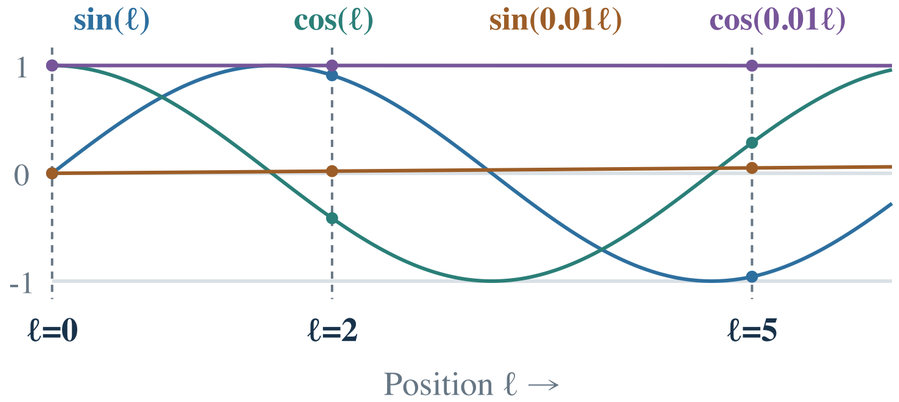
Look up the token's embedding, then **add a vector describing its position**:

$$\boldsymbol{x}_\ell = E[w_\ell, :]^\mathsf{T} + \boldsymbol{p}_\ell, \qquad \boldsymbol{x}_\ell, \boldsymbol{p}_\ell \in \mathbb{R}^d$$

Fixed sinusoids: $p_{\ell,2i} = \sin(\ell\omega_i), p_{\ell,2i+1} = \cos(\ell\omega_i), \omega_i = 10000^{-2i/d}$.

Pair index $i = 0, \ldots, d/2 - 1$; even width d.

d = 4: read four features at each slice



Position vector = values at that slice

ℓ = 0	[0.00, 1.00, 0.00, 1.00]ᵀ
ℓ = 2	[0.91, -0.42, 0.02, 1.00]ᵀ
ℓ = 5	[-0.96, 0.28, 0.05, 1.00]ᵀ

Order: sine, cosine, sine, cosine

Same token "the" at positions 0 and 2: $E[\text{the}, :]^\mathsf{T} + \boldsymbol{p}_0 \neq E[\text{the}, :]^\mathsf{T} + \boldsymbol{p}_2$.

RoPE rotates query and key pairs by their positions

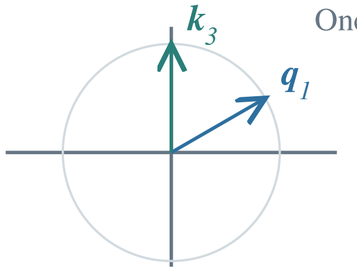
Split each query/key into **2-D feature pairs**; rotate every pair by a position-dependent angle.

$$R(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}, \qquad \tilde{\boldsymbol{q}}_\ell^{(i)} = R(\ell\theta_i)\boldsymbol{q}_\ell^{(i)}$$

$$[\underbrace{q_{\ell,0}, q_{\ell,1}}_{\text{pair 0: } \ell\theta_0}, \quad \underbrace{q_{\ell,2}, q_{\ell,3}}_{\text{pair 1: } \ell\theta_1}, \quad \ldots]^\mathsf{T}, \qquad \theta_i = 10000^{-2i/d_k}$$

Fast and slow pairs cover different distance scales. Keys use the same rotations.

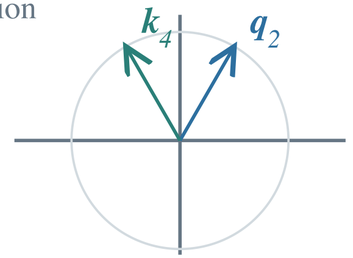
Positions 1 and 3



Angle difference 60°; dot product 0.5

One pair: same $[1, 0]^\mathsf{T}$ content; 30° per position

Positions 2 and 4



Angle difference 60°; dot product 0.5

RoPE compares positions through angle differences

Moving both tokens one step rotates both arrows by the same extra angle: **their angle difference stays the same**.

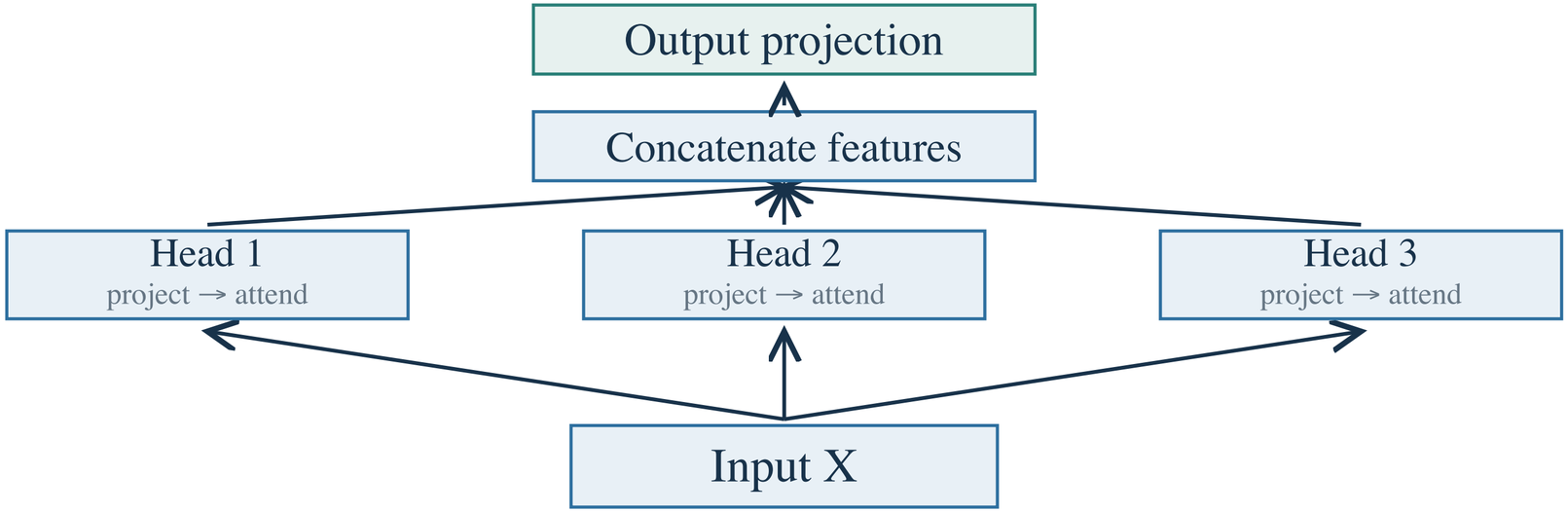
$$s\theta_i - \ell\theta_i = (s-\ell)\theta_i, \qquad (\tilde{\boldsymbol{q}}_\ell^{(i)})^\top \widetilde{\boldsymbol{k}}_s^{(i)} = (\boldsymbol{q}_\ell^{(i)})^\top R((s-\ell)\theta_i)\boldsymbol{k}_s^{(i)}$$

The full dot product **adds the matches from all feature pairs**.

Why use RoPE?	Practical consequence
Relative offset enters the query–key match directly	Same separation gives the same positional comparison, with content held fixed.
Fixed rotations, no learned position table	Few extra operations; no separate learned vector for every index.

Used in Llama 2: RoPE is itself a positional encoding, an alternative to adding absolute position vectors.

Multiple heads learn different ways to combine context



$$Y = \mathrm{Concat}(Y_1, \ldots, Y_b) W_O^\top$$

Each head has its own query/key/value projections; every head obeys the allowed context.

Next: combine attention with residual paths, normalization and a token-wise MLP.

Implementation detail: padding masks exclude absent tokens

Two messages share a batch:

	Input positions
Message A	`please` · `send` · `the` · `report`
Message B	`send` · `report` · `<PAD>` · `<PAD>`

- **Attention key mask:** padded positions cannot supply values to real queries.
- **Loss mask:** padded targets contribute no training loss.

A causal mask still applies within each real prefix.

Building a Transformer

David I. Inouye

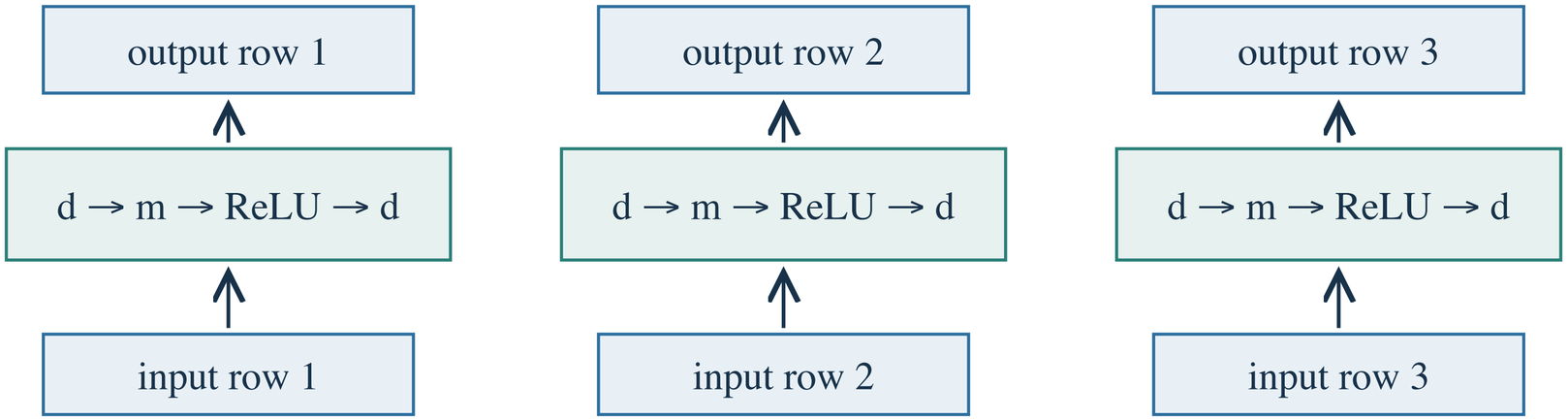
Dynamic Content · Last Updated: October 30, 2026

A transformer combines context mixing and feature transformation

Transform features → preserve a residual path → normalize → stack blocks.

Keep the same task: use an email prefix to predict its next token.

Attention mixes positions; a shared MLP transforms each row



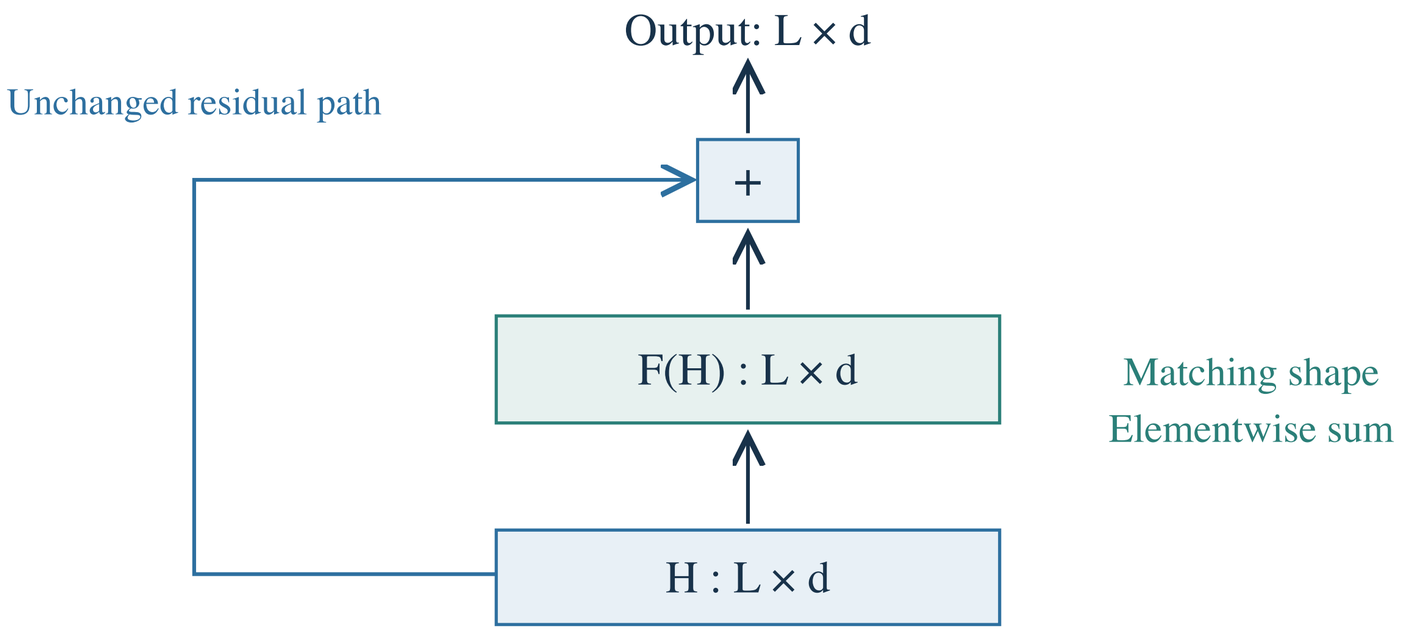
Same MLP at every position; no mixing between positions

$$\mathrm{MLP}(H) = \mathrm{ReLU}(HW_1^\top + \mathbf{1}\boldsymbol{b}_1^\top)W_2^\top + \mathbf{1}\boldsymbol{b}_2^\top$$

$$H \in \mathbb{R}^{L\times d}, \quad W_1 \in \mathbb{R}^{m\times d}, \quad W_2 \in \mathbb{R}^{d\times m}$$

Same parameters at every position; separate parameters in each block.

A residual path adds a learned update without changing the shape



$$H_{\text{out}} = H + F(H)$$

The attention output projection and the MLP's last layer both return d **features**.

LayerNorm normalizes one token's features at a time

Like BatchNorm: subtract a mean, divide by a standard deviation, then learn a scale and shift.

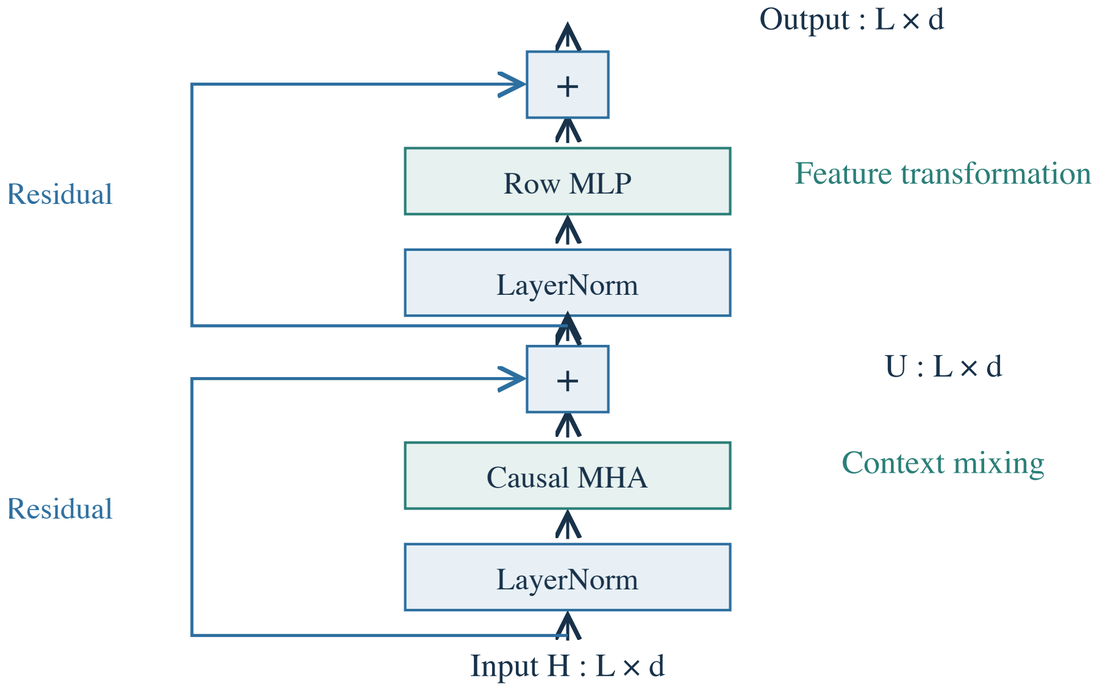
$$\mu_\ell = \frac{1}{d}\sum_{j=1}^{d} h_{\ell j}, \qquad \sigma_\ell^2 = \frac{1}{d}\sum_{j=1}^{d} (h_{\ell j} - \mu_\ell)^2$$

$$\mathrm{LN}(\boldsymbol{h}_\ell) = \boldsymbol{\gamma} \odot \frac{\boldsymbol{h}_\ell - \mu_\ell \mathbf{1}}{\sqrt{\sigma_\ell^2 + \epsilon}} + \boldsymbol{\beta}$$

- **BatchNorm (CNN):** one channel across batch examples and spatial locations.
- **LayerNorm:** one token across its d features—the last axis of `[B,L,d]`.

At evaluation, LayerNorm still uses **that token's statistics**; BatchNorm usually uses running statistics.

A pre-norm block adds two shape-preserving updates



$$U = H + \mathrm{MHA}_{\mathrm{causal}}(\mathrm{LN}_1(H))$$

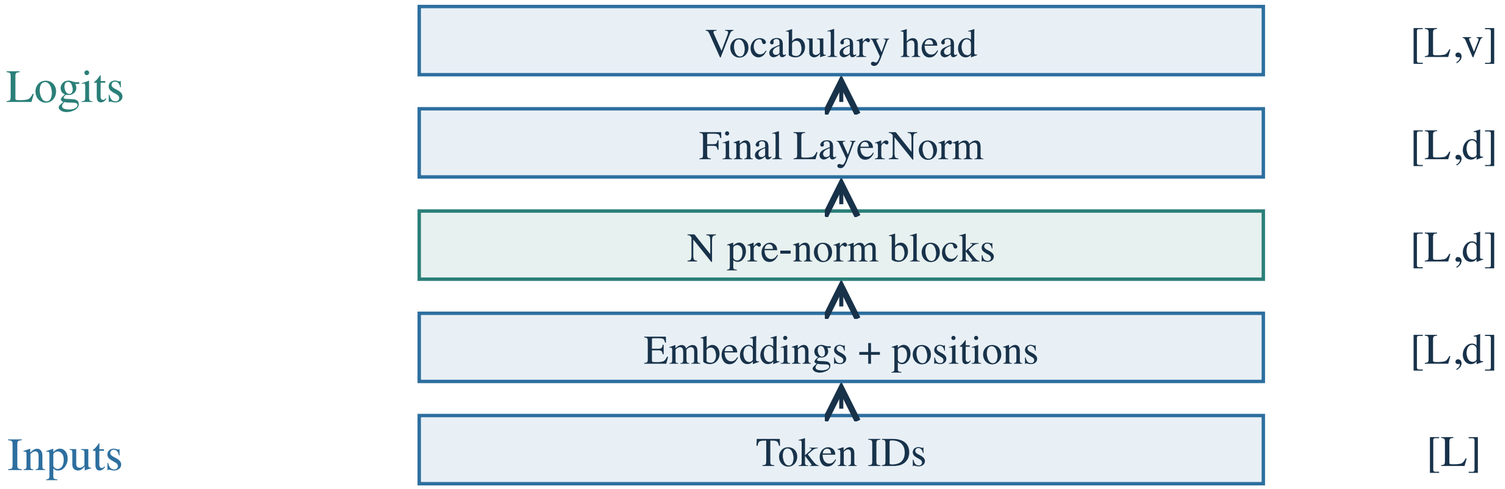
$$H_{\mathrm{out}} = U + \mathrm{MLP}(\mathrm{LN}_2(U))$$

Pre-norm is the usual modern LLM default. The original Transformer used post-norm.

Pre-norm	Post-norm
$H + F(\mathrm{LN}(H))$	$\mathrm{LN}(H + F(H))$
Normalize **before the branch**.	Normalize **after addition**.

Pre-norm leaves the bypass untouched.

Stacked blocks turn token features into next-token logits

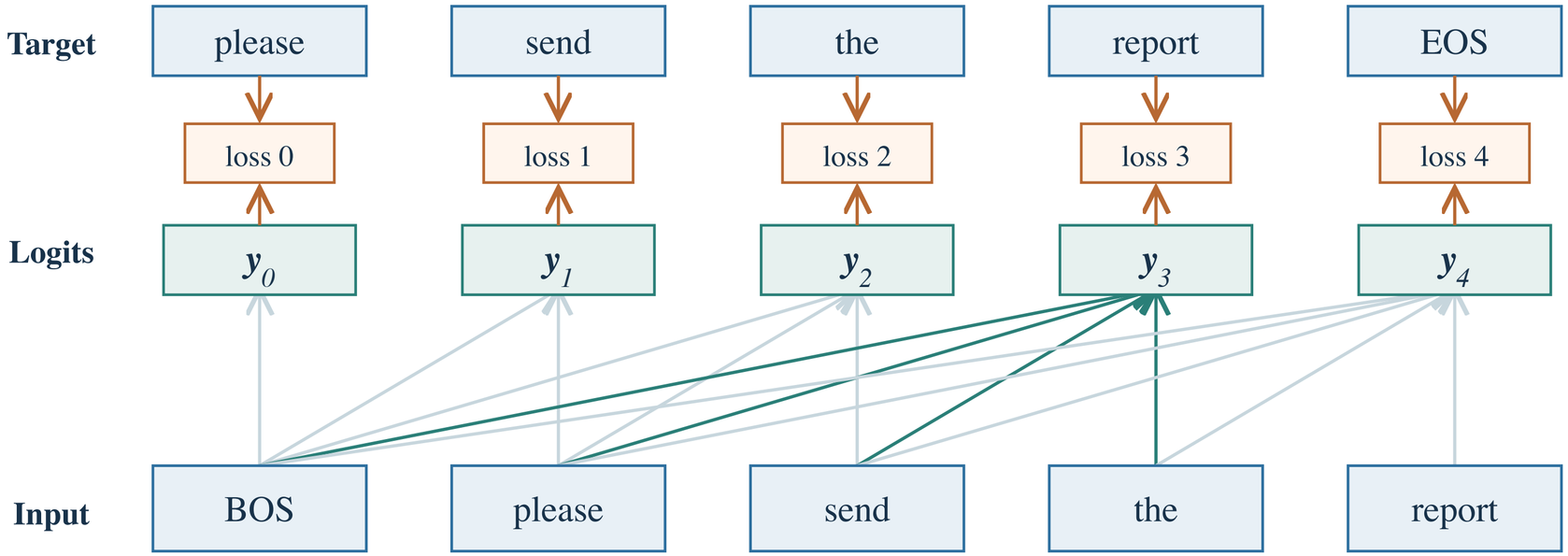


RoPE alternative: omit additive positions; rotate Q/K within each block

$$H^{(0)} = E[w_{0:L}, :] + P, \qquad Y = \mathrm{LN}_{\mathrm{final}}(H^{(N)})W_{\mathrm{vocab}}^{\mathsf{T}} + \mathbf{1}\boldsymbol{c}^{\mathsf{T}}$$

$$H^{(N)} \in \mathbb{R}^{L\times d}, \qquad W_{\mathrm{vocab}} \in \mathbb{R}^{v\times d}, \qquad Y \in \mathbb{R}^{L\times v}$$

One length-L input solves L prefix-prediction problems



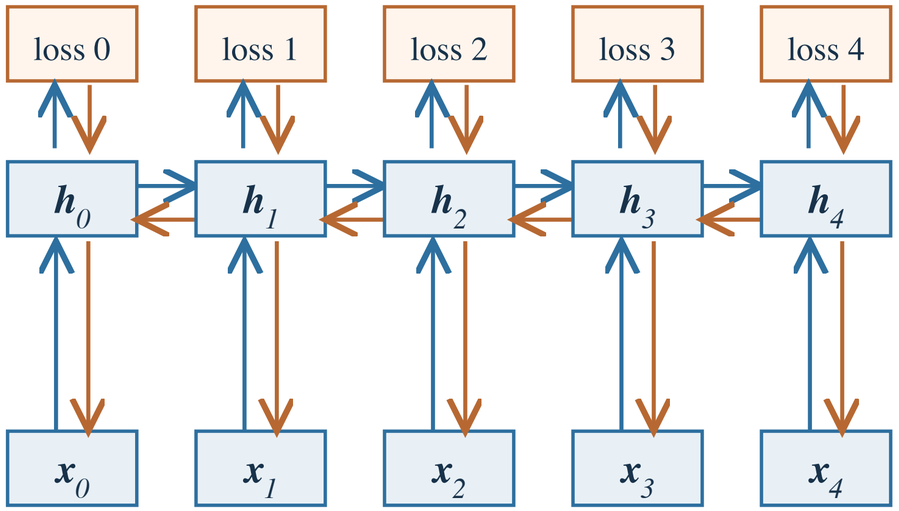
$$\mathcal{L}(\theta) = -\frac{1}{L}\sum_{\ell=0}^{L-1} \log p_\theta(w_{\ell+1} \mid w_0, \ldots, w_\ell).$$

Known inputs allow parallel work; the mask preserves each allowed prefix.

Backpropagation follows different dependency graphs

Same inputs: BOS, please, send, the, report • Same targets: please, send, the, report, EOS

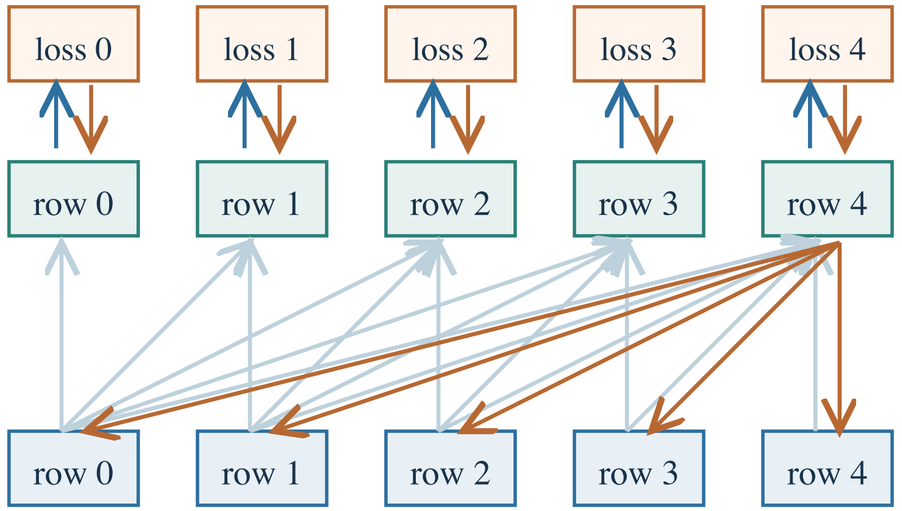
RNN: a sequential state chain



Forward: left → right; backward: right → left

Later states must wait for earlier states

Transformer: positions processed together



Shown: backward paths from the last row

Sequential across layers, not across token states

Blue: forward • Orange: backward • All losses contribute to shared parameters

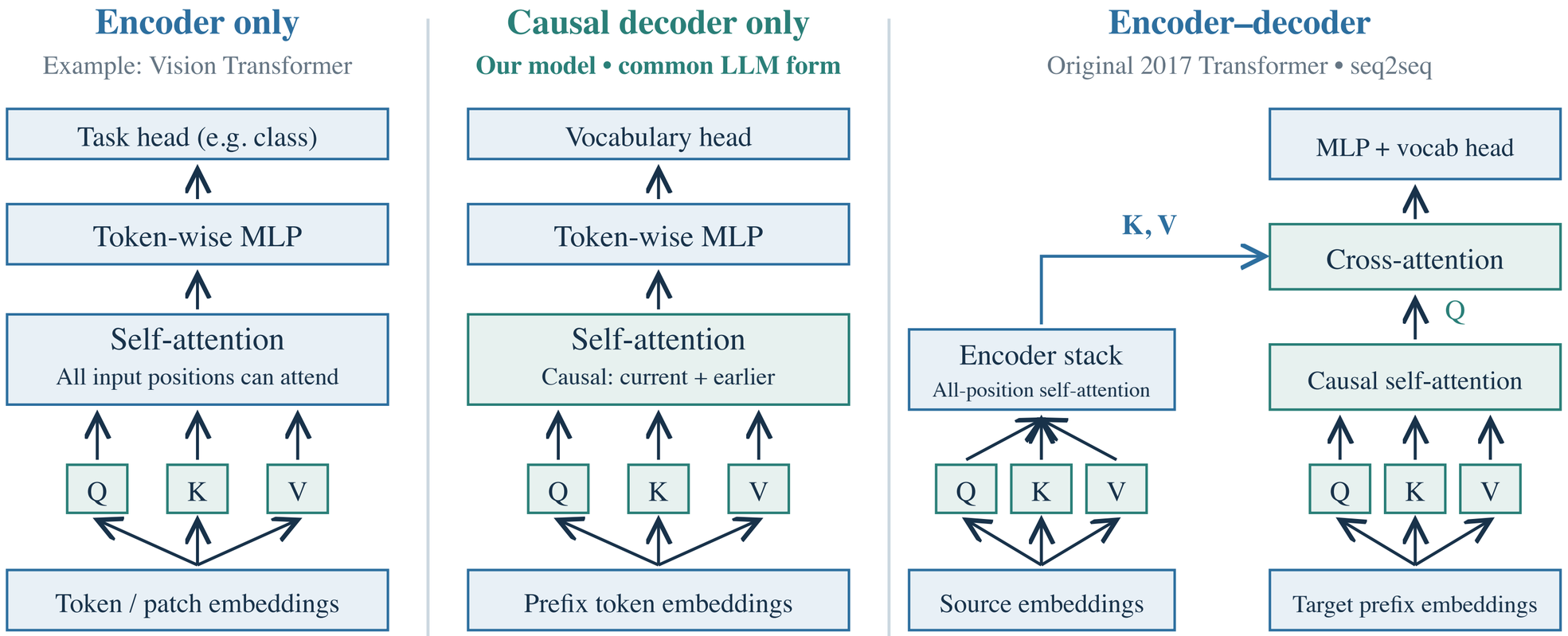
Transformer benefit: parallel work across positions in forward **and backward** computation.

Can one forward pass score all shifted targets?

> **The training batch contains each full email.**
> **Which model change would let an early prediction read its future target?**
> **A. Compute all causal output rows together**
> **B. Normalize each token over its own features**
> **C. Remove the causal attention mask**

C. Parallel computation is valid when every path respects the allowed prefix.

A causal decoder is the common decoder-only LLM architecture



Q/K/V are learned projections of the indicated stream. Layers repeat; residual paths and normalization omitted.

Our email model is a causal decoder. Other tasks can use different information paths and heads.

From Prediction to Generation

David I. Inouye

Dynamic Content · Last Updated: October 30, 2026

A next-token model becomes a generator through repeated feedback

Factorize → choose a token → append → repeat → evaluate.

We have an email-prefix model. Now we must choose a continuation and decide when to stop.

A sequence probability is a product of conditional probabilities

For a fixed prompt c and continuation tokens $w_1, \ldots, w_L$:

$$p_\theta(w_{1:L} \mid c) = \prod_{\ell=1}^{L} p_\theta(w_\ell \mid c, w_{<\ell}).$$

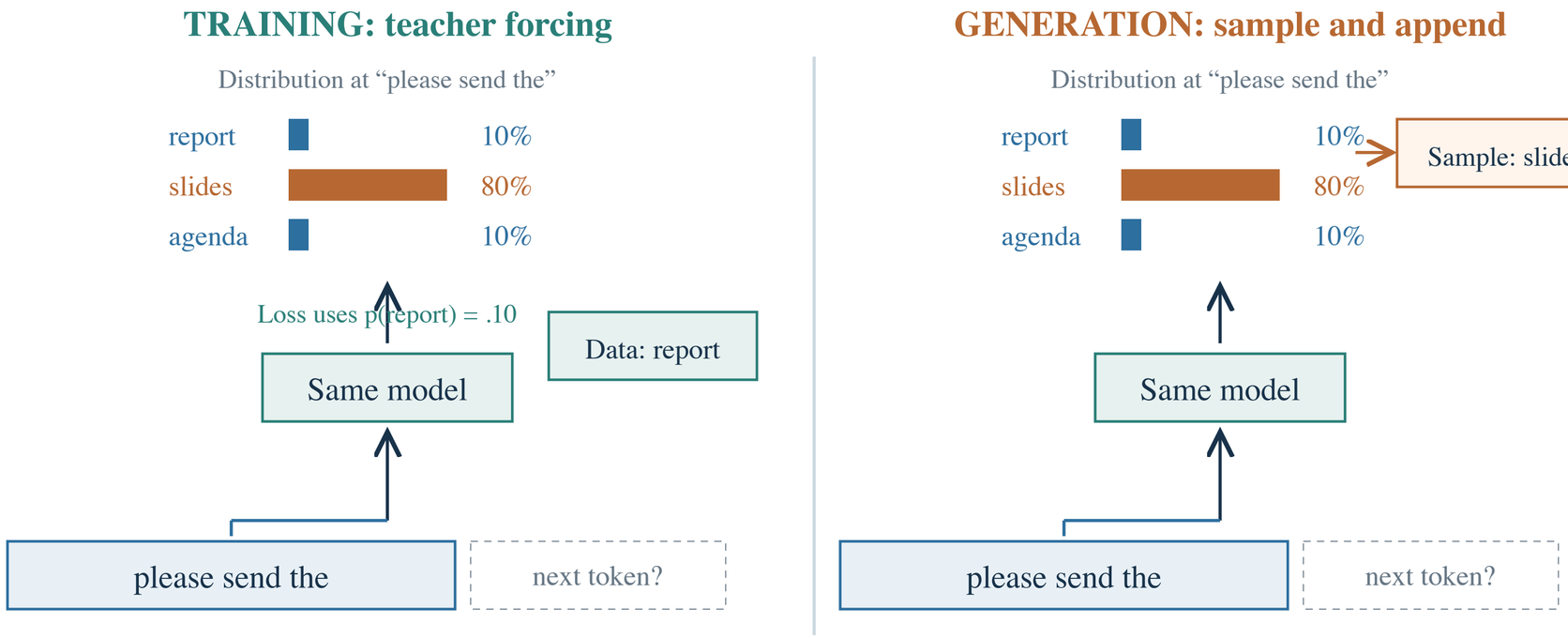
Synthetic token example:

$$p_\theta(\text{report, today, EOS} \mid \text{please send the})$$

$$= p_\theta(\text{report} \mid c)\; p_\theta(\text{today} \mid c, \text{report})\; p_\theta(\text{EOS} \mid c, \text{report,today}).$$

Taking **negative logs** turns the product into the sum used in next-token training.

Teacher forcing uses the true prefix—even when the model is wrong



Advance: append the observed token in training, or the sampled token in generation.

Training prefixes are all known in advance → a causal transformer can process them together.