DDA4220 · Fall 2026

Lecture 12

Deep Learning and Applications

Transformers and Vision Transformers

Zhen Liu

CUHK-Shenzhen

Lecture outline

01 Recap: RNNs and attention

02 Building a Transformer

03 Transformer architectures

04 Transformers for vision

* denotes optional slides

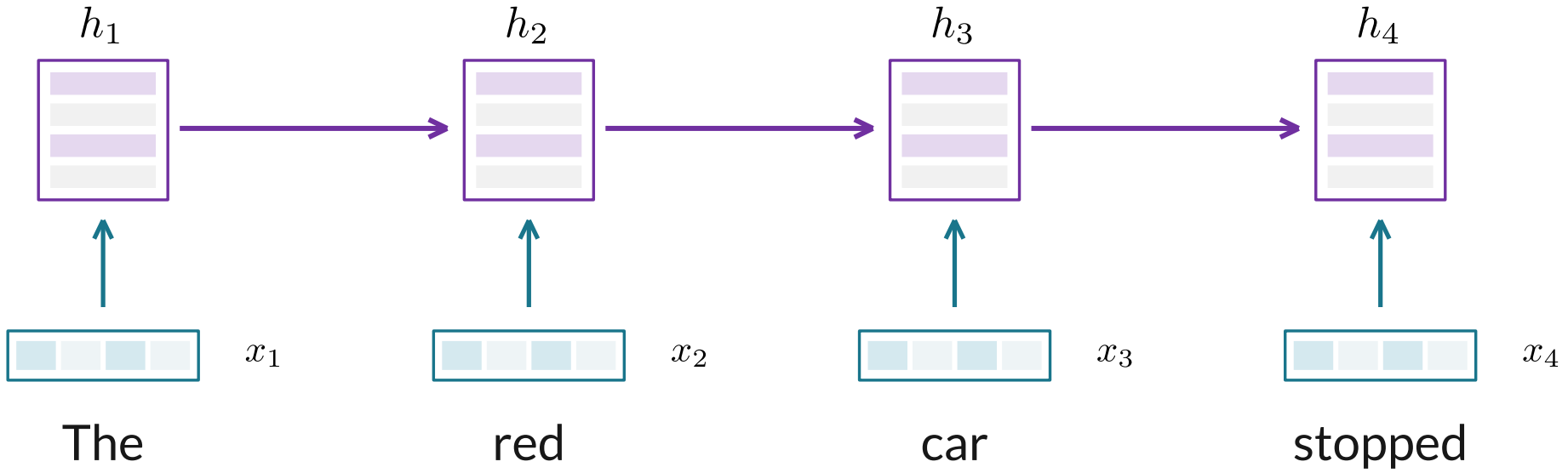
RNN inputs and outputs

Input: one feature vector per token

$$h_i = f(x_i, h_{i-1})$$

The same learned update f at every position

Outputs: hidden states (contextual features)



Each state needs the previous state before it can be computed.

Adapted from Stanford CS224n

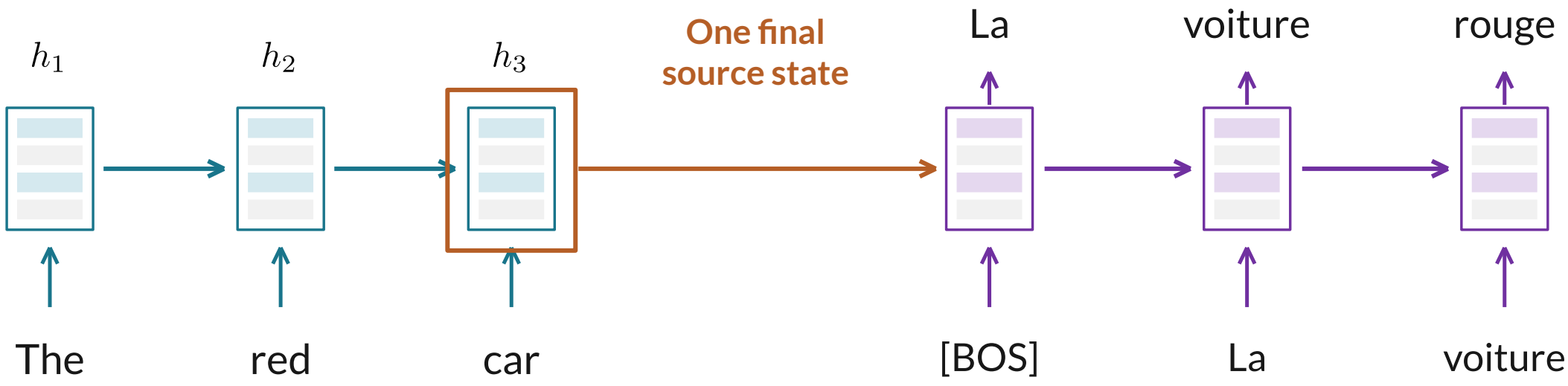
The fixed-summary bottleneck

English source

Encoder states

French target

Selected from next-token distributions



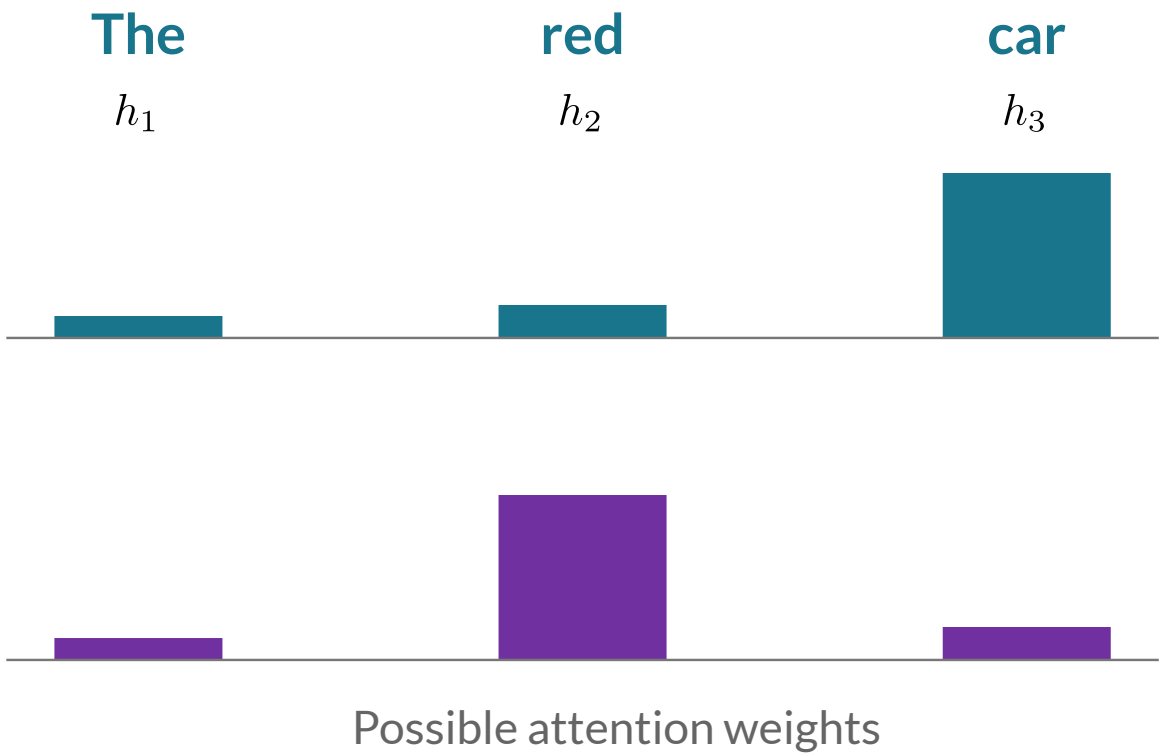
[BOS]: beginning of sequence La: the voiture: car rouge: red

More source details must fit in the same fixed-width summary.

Adapted from Stanford CS224n

Source attention at different steps

Keep all source states for “The red car”.



The decoder state changes the readout. A fixed average cannot do this.

Adapted from Stanford CS224n

Attention as a weighted readout

Compare

$$e_j = \text{score}(s, h_j)$$

A learned function matches the decoder state to each source state.

Normalize

$$\alpha_j = \frac{\exp(e_j)}{\sum_\ell \exp(e_\ell)}$$

Softmax over source positions. Weights are nonnegative and sum to 1.

Retrieve

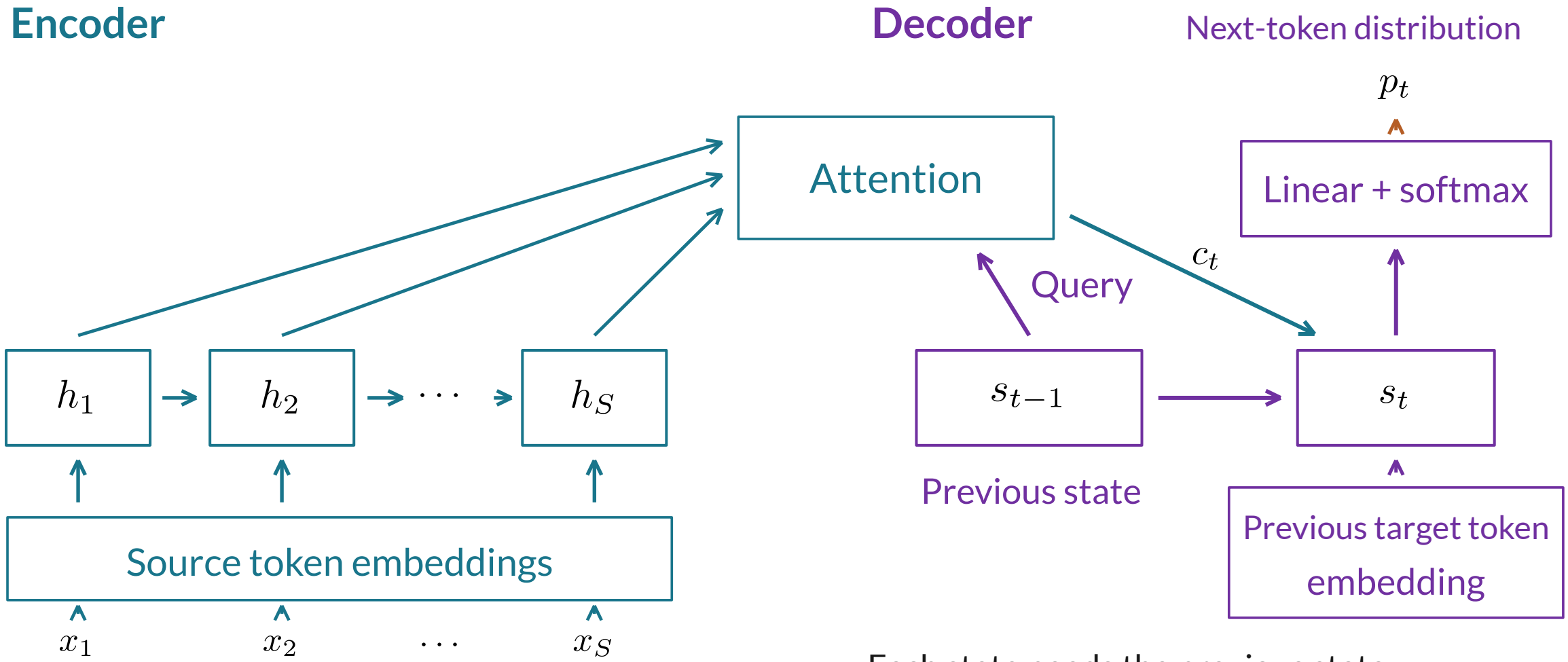
$$c = \sum_j \alpha_j h_j$$

A weighted feature vector, not a selected word.

s: decoder state h_j: source state c: retrieved context

The prediction loss trains the scoring function.

Recurrence remains



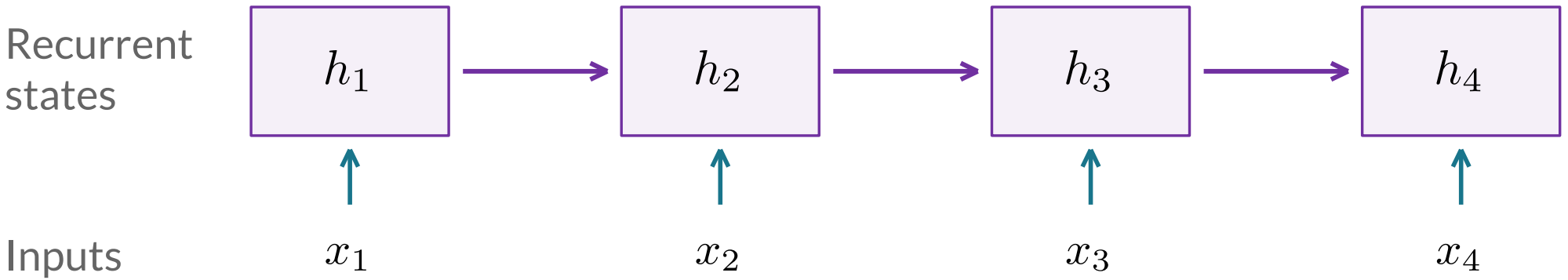
Each state needs the previous state.

Source: Adapted from CMU 11-785

Sequential training in RNNs

All input tokens are known during training.

Hidden states still have to be computed in order.



Backpropagation through time follows the chain in reverse.

Within each sequence, these dependencies limit GPU parallelism.

Adapted from Stanford CS224n

Self-attention in the encoder

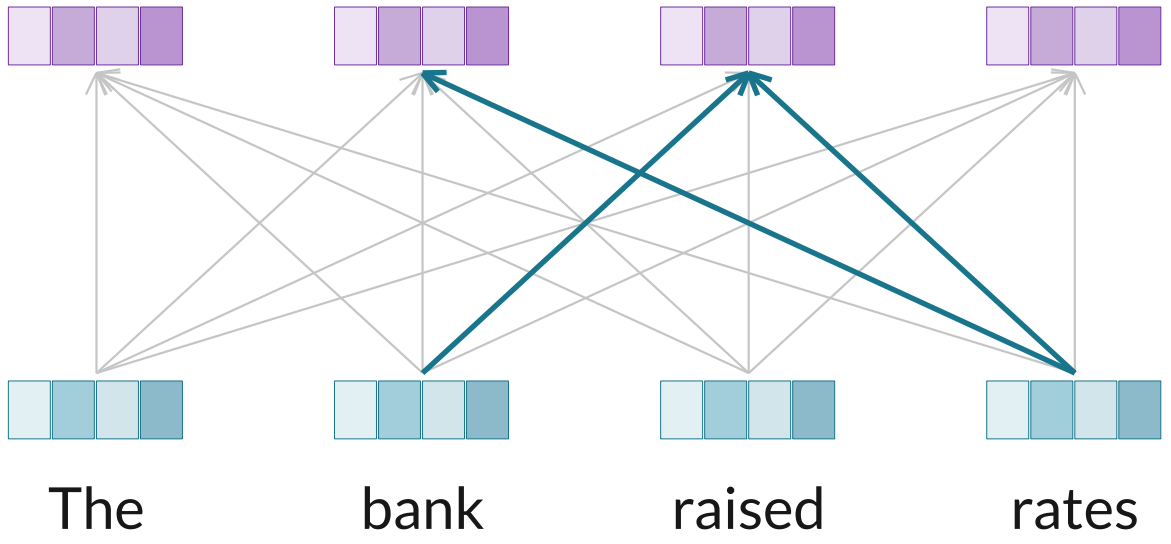
Build a contextual feature for every token in the input.

The bank raised rates.

“rates” clarifies which kind of bank.

“bank” and “rates” help explain what “raised” means.

One contextual output per token



Each output can read every input, including its own token.

Connections adapted from Stanford CS224n

Attention as retrieval

Library search: match catalog descriptions, then retrieve book contents.

Query

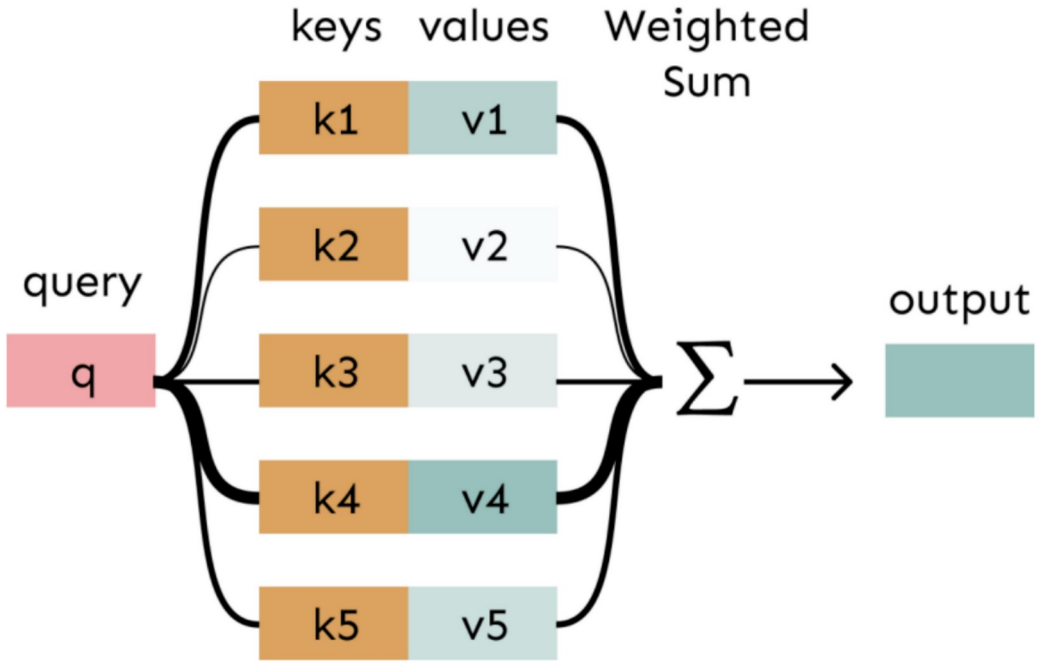
What you are looking for.

Key

Description used to match an item.

Value

Content retrieved from that item.



Attention returns a weighted mixture of value vectors.

Source: Berkeley CS288

Learning queries, keys and values

Useful context need not be the most similar token.

The **dog** ate because **it** was hungry.

“it” needs information about “dog”.

Queries and keys

Learn which tokens should match.

Values

Learn which features to pass onward.

The task loss trains all three maps through backpropagation.

Query, key and value projections

x_i: input token feature, written as a row vector.

$q_i = x_i W_Q$	**Query**	Request
$k_i = x_i W_K$	**Key**	Matching description
$v_i = x_i W_V$	**Value**	Retrieved content

The learned W matrices are shared across token positions.

Other differentiable maps, including MLPs, are possible.
Linear layers are the standard choice: simple and efficient.

Self-attention computation

For one query position i, read all permitted input positions j.

$$e_{ij} = \frac{q_i k_j^\top}{\sqrt{d_k}}$$

Compare with every key

d_k is the query/key width. Scaling keeps wider dot products from becoming too large.

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_\ell \exp(e_{i\ell})}$$

Normalize over input positions

These are attention weights, not vocabulary probabilities.

$$o_i = \sum_j \alpha_{ij} v_j$$

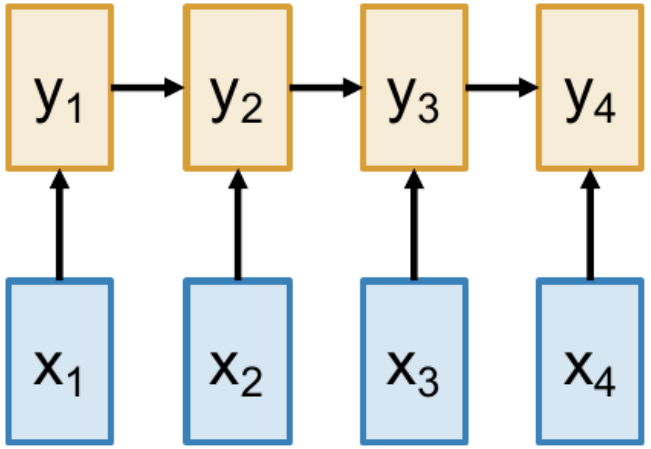
Combine values into a feature

Repeat for every query: one contextual output per input token.

Three ways to build context

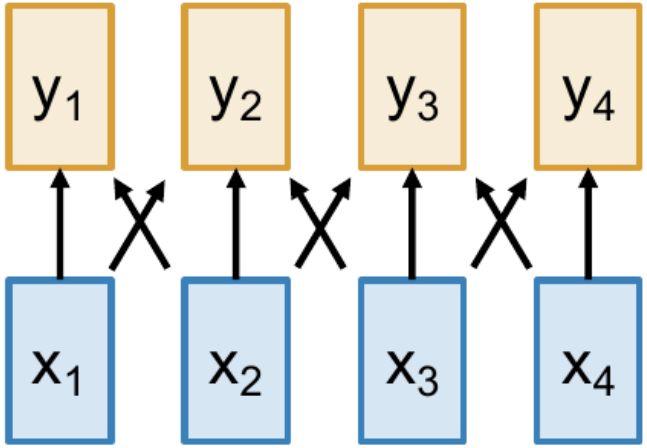
RNN

Input features below, updated features above



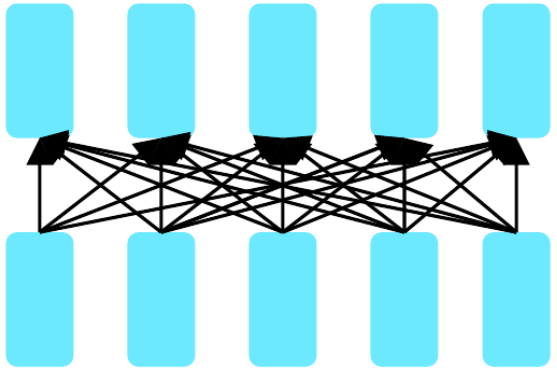
Build context step by step

Convolution



Read local neighbors

Self-attention



Read across available input

One attention layer connects distant tokens and updates all positions together.

Source: Stanford CS231n and CS224n

Lecture outline

01 Recap: RNNs and attention

02 Building a Transformer

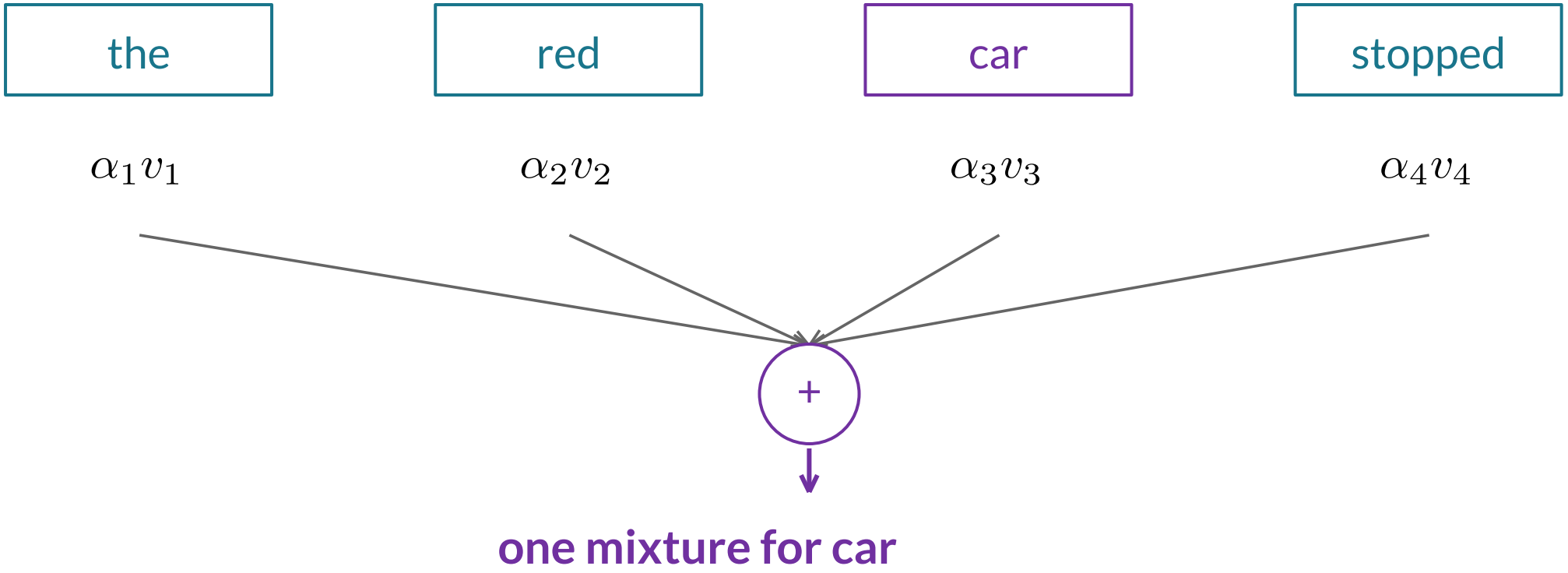
03 Transformer architectures

04 Transformers for vision

One attention mixture

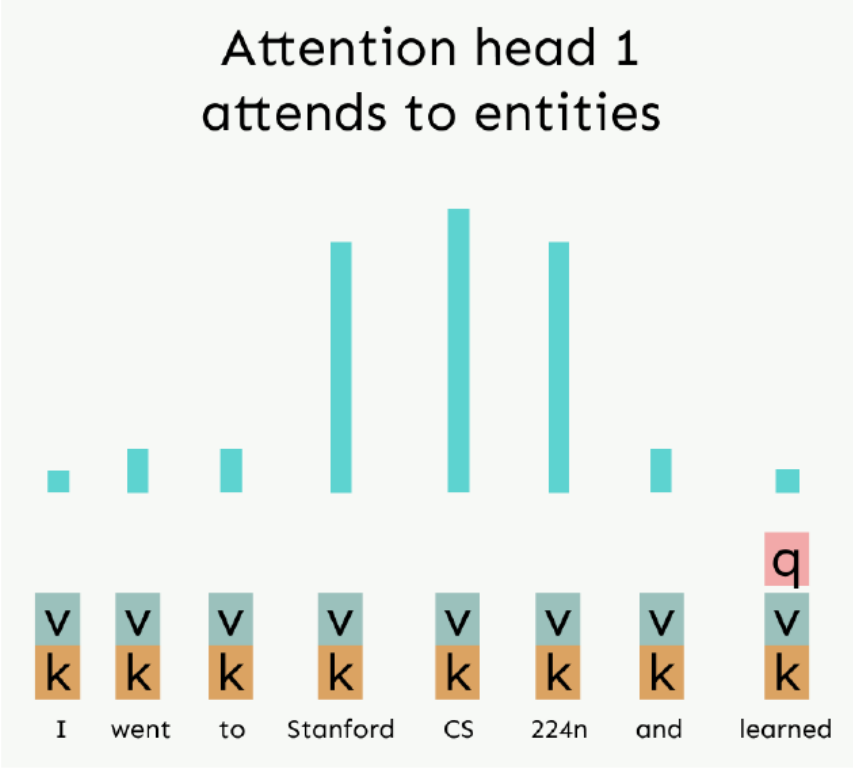
One head produces one set of attention weights for a query.

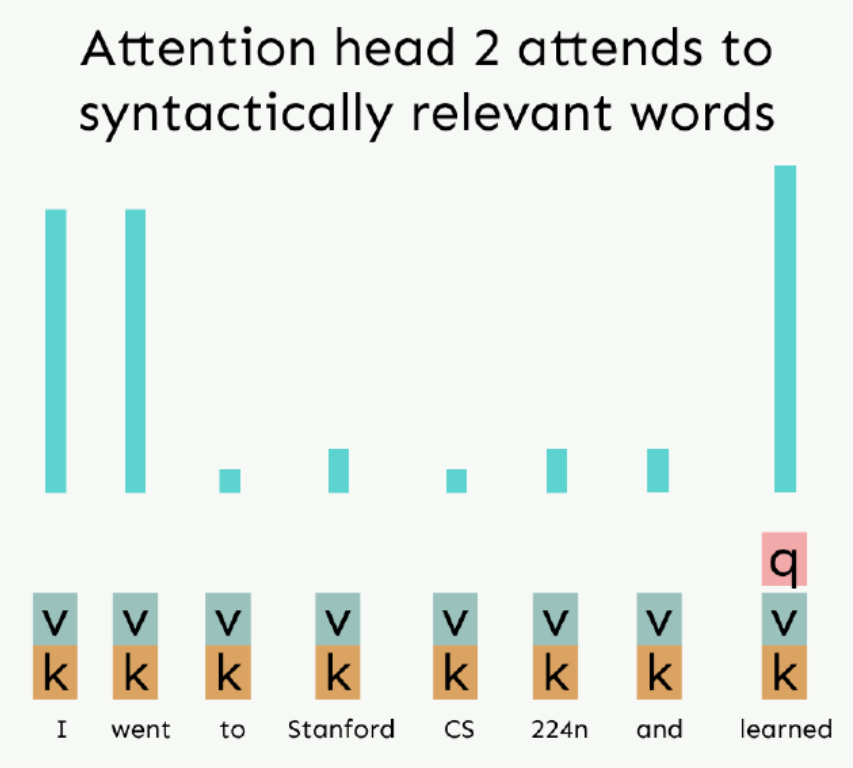
An attribute and an event may call for different mixtures.



Multiple attention heads

Illustration: heads can use different attention patterns.

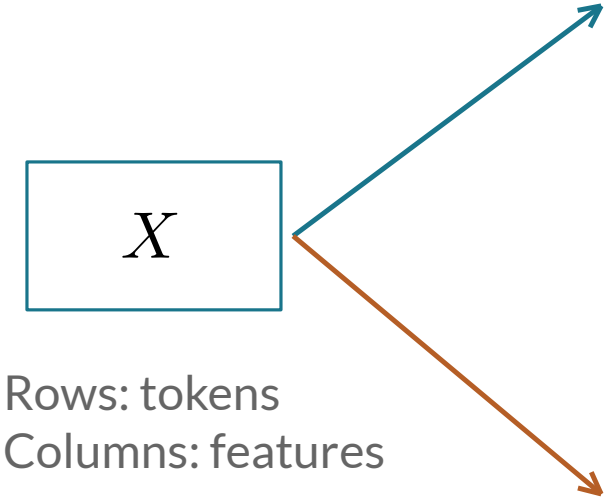




Source: Stanford CS224n

Independent projections

Each head learns its own queries, keys and values.



Rows: tokens
Columns: features

Head 1

$$Q_1 = XW_1^Q, \quad K_1 = XW_1^K, \quad V_1 = XW_1^V$$

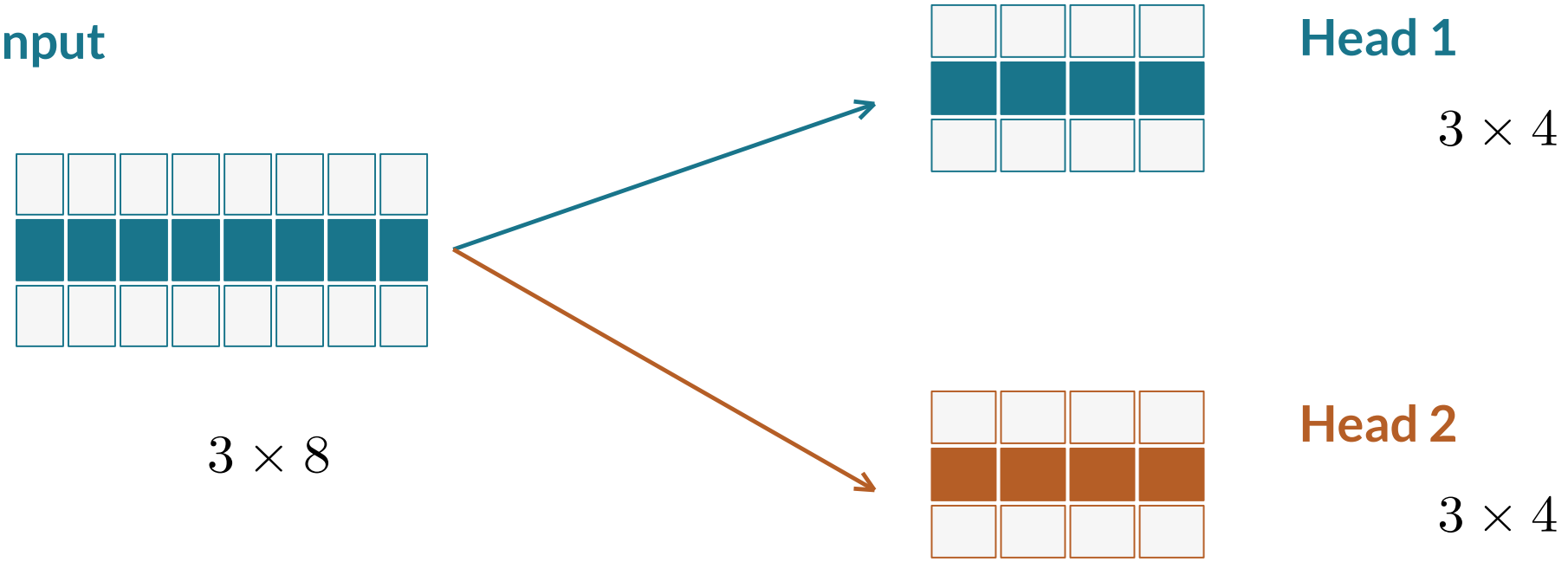
Head 2

$$Q_2 = XW_2^Q, \quad K_2 = XW_2^K, \quad V_2 = XW_2^V$$

The input is shared. The projection weights are different.

Per-head feature width

Example: three positions, eight features, two heads



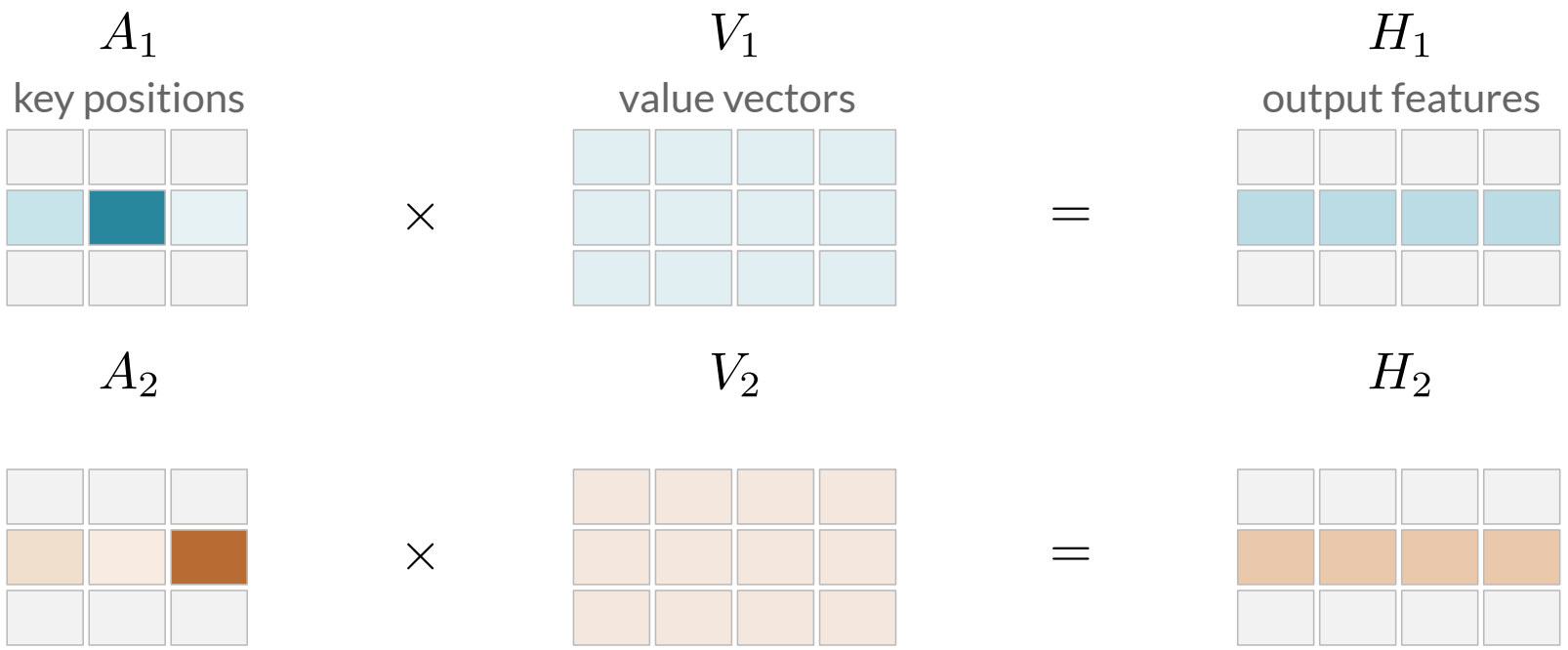
Each head projects the full input vector into four features.

Attention weights in each head

For each head $$A_h = \text{softmax}\left(\frac{Q_h K_h^\top}{\sqrt{d_k}}\right)$$

h: head index
d_k: query/key width

Softmax is applied across each query row.

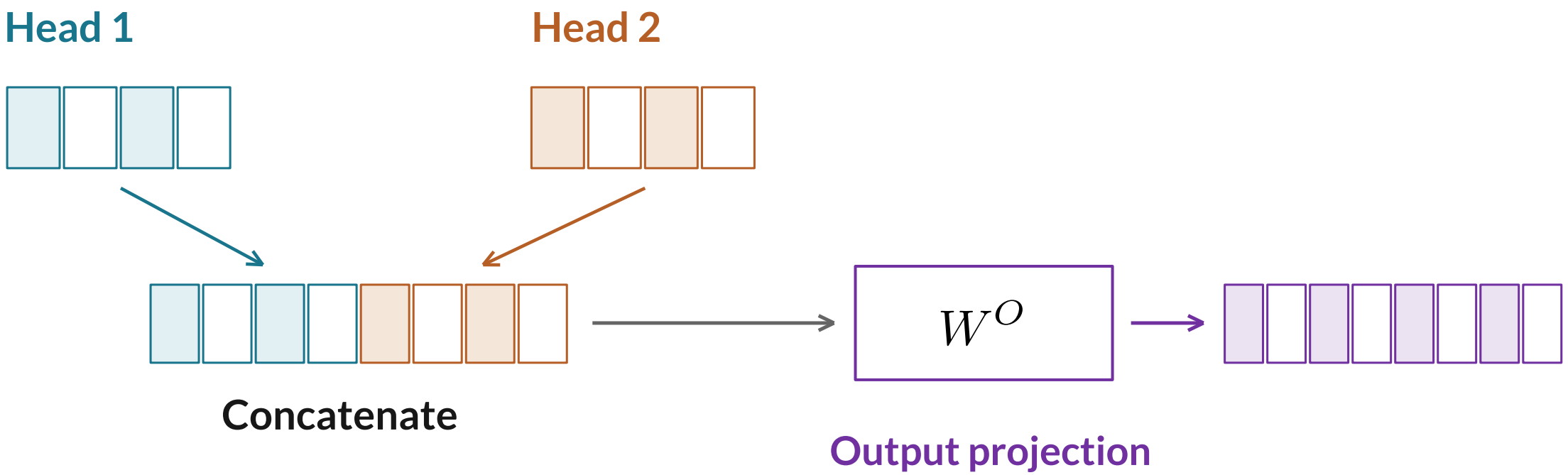


The highlighted query receives a different mixture in each head.

Adapted from Stanford CS231n

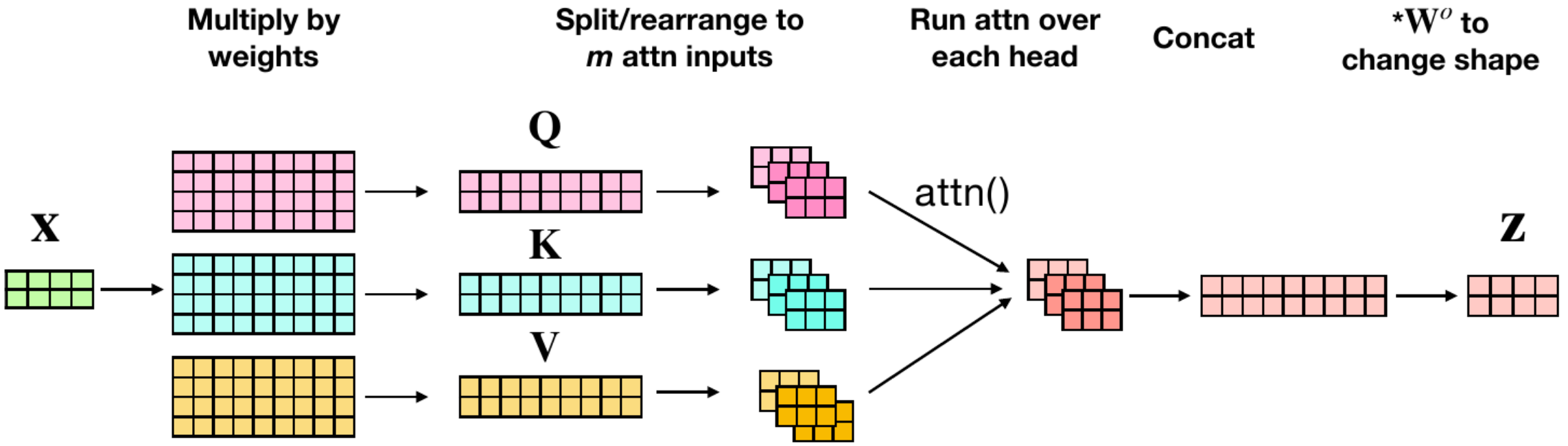
Combining the heads

For one token position



The learned projection mixes the head features.

Multihead attention



Concat joins the head outputs for each token.

$$\text{MultiHead}(X) = \text{Concat}(H_1, \ldots, H_m)W_O$$

$X,\ Z$ input / output token vectors
m number of heads

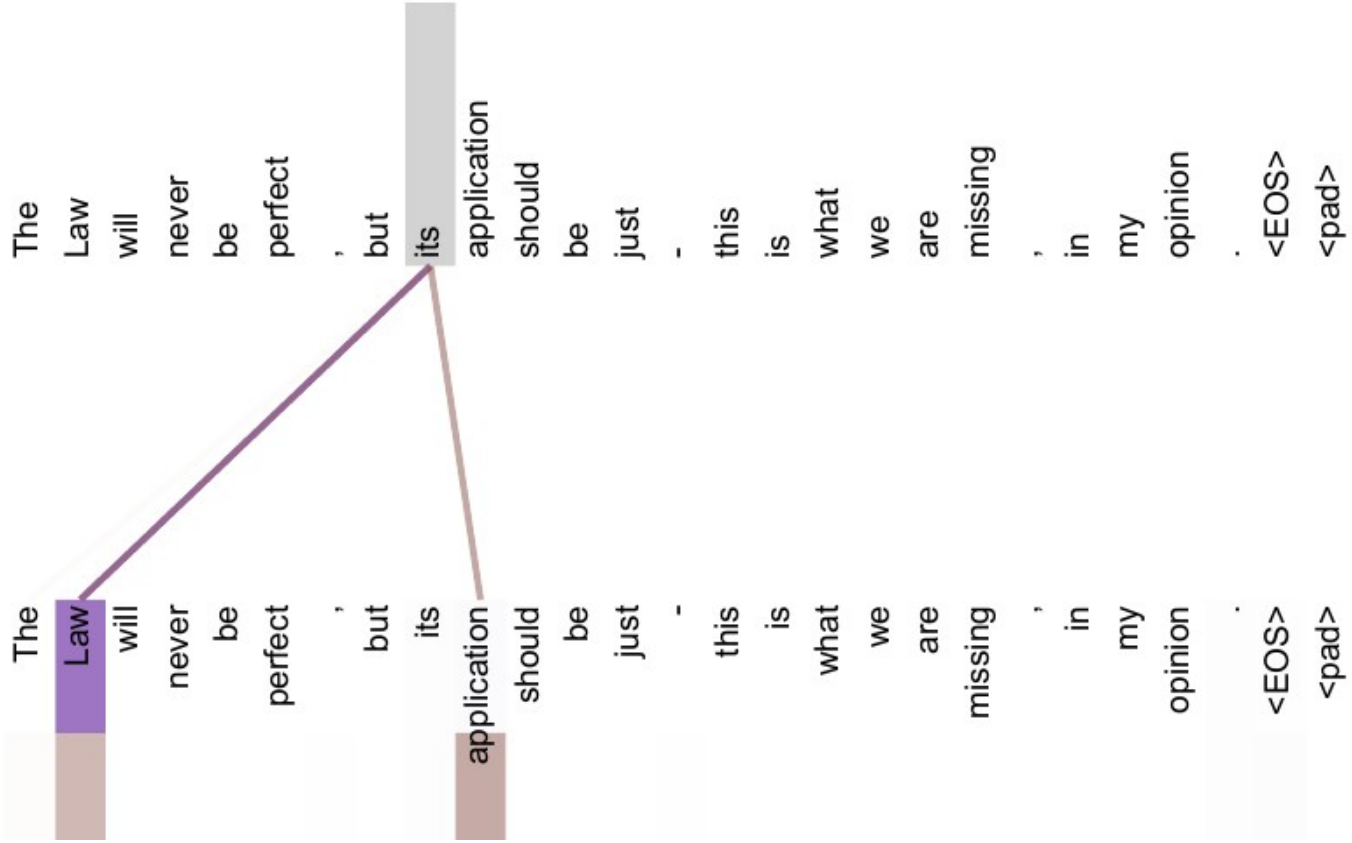
H_h output of head h
$W_O = W^O$ learned output projection

Source: Berkeley CS288

Learned attention patterns

Two heads in a trained translation model

The query is “its”. Different colors show different heads.



Source: Attention Is All You Need

Word order

The same words can describe different events.

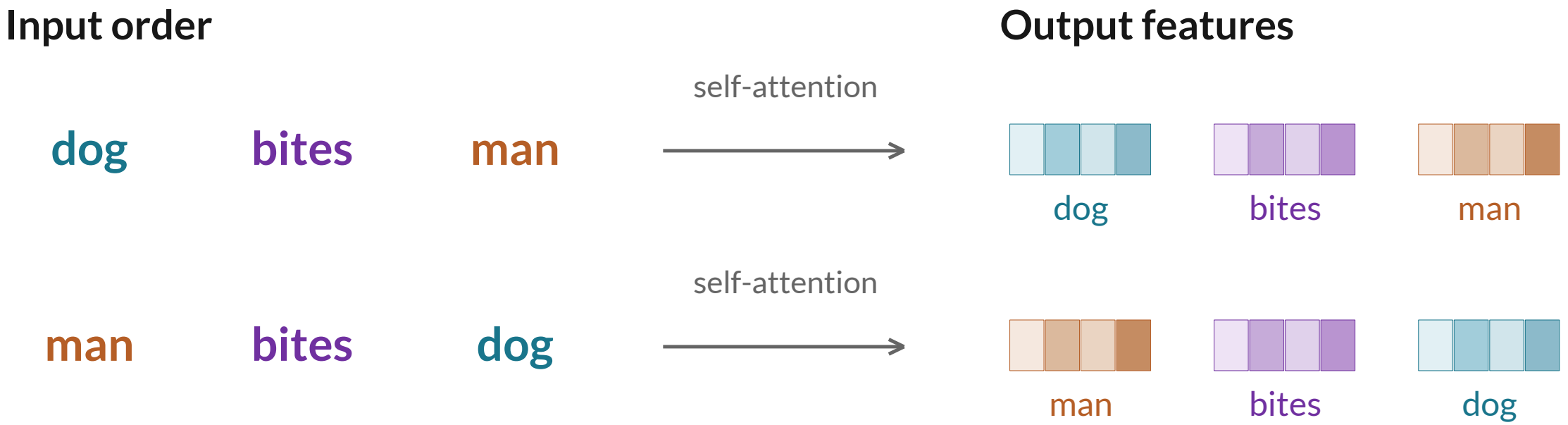
dog	**bites**	**man**	**man**	**bites**	**dog**
does the biting		gets bitten	does the biting		gets bitten

The model must know who acts and who receives the action.

What in self-attention tells it which word came first?

Recap: self-attention and order

No position information. Every token can attend to every token.

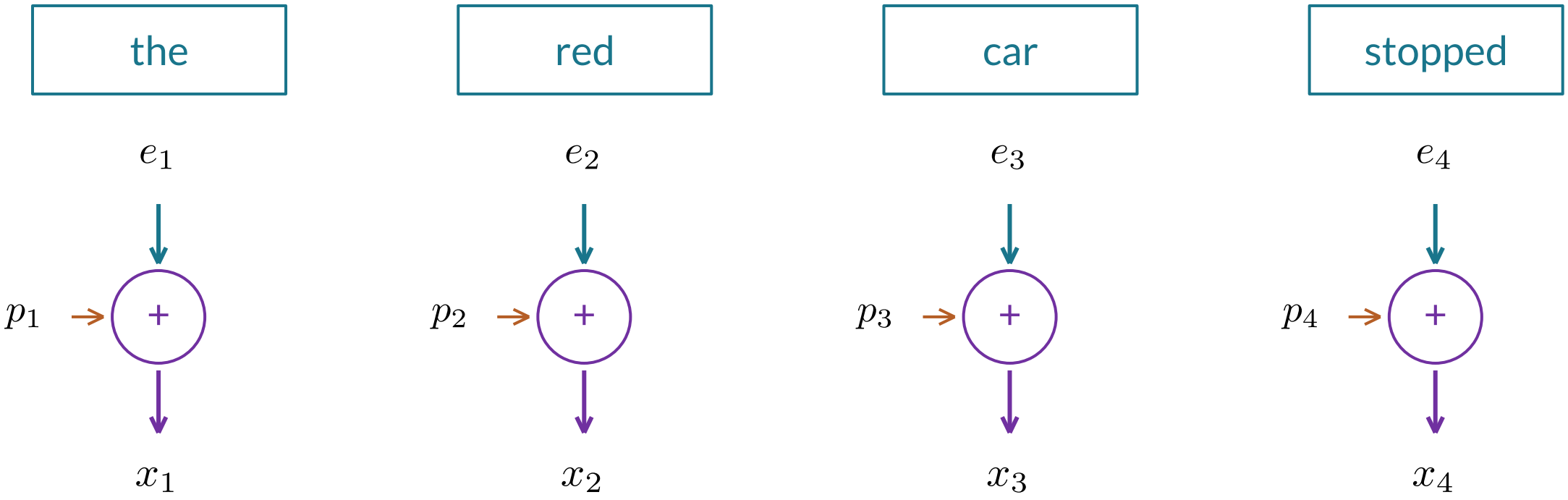


Reordering the inputs only reorders the outputs.

This is permutation equivariance. The model still needs word order.

Positions in the input sequence

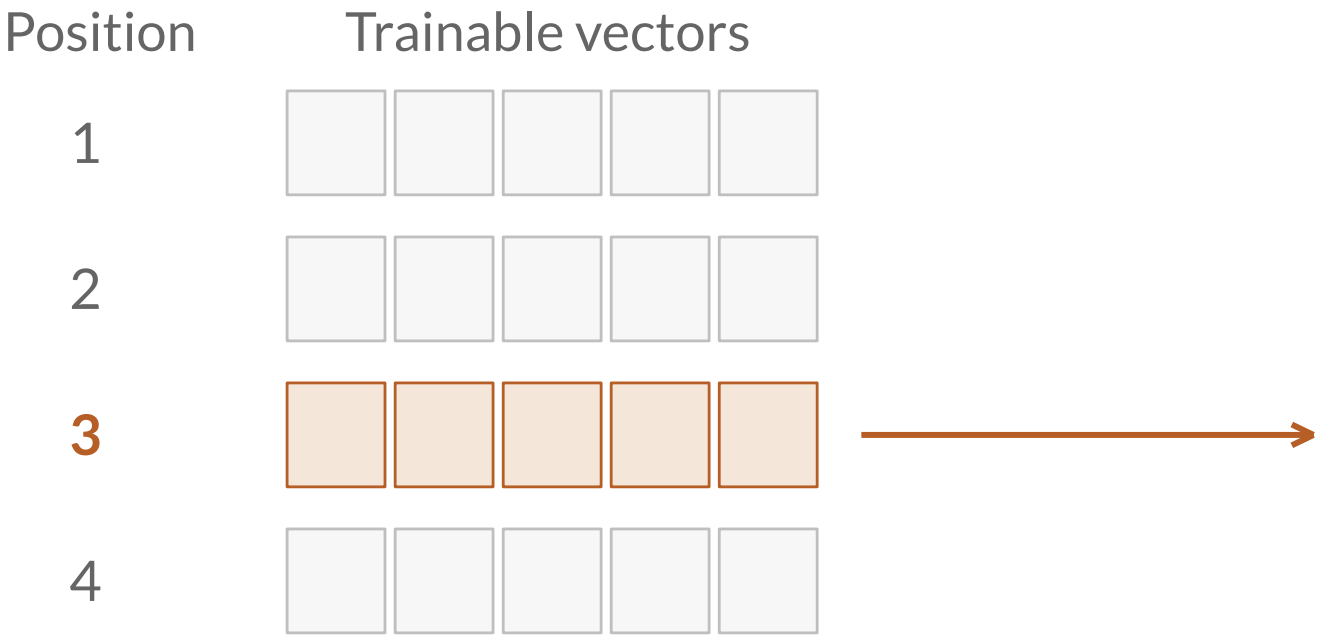
Content and position enter before the attention projections.



e: token embedding p: position vector x: combined input

Learned position embeddings

One learned vector for each position index



p_3

Used at position 3 in every example

Training updates the lookup table along with the model weights.

Sinusoidal position vectors

p_i is a fixed position vector, with d coordinates.

Sine coordinate

$$p_{i,\,2j} = \sin\left(\frac{i}{10000^{2j/d}}\right)$$

Cosine coordinate

$$p_{i,\,2j+1} = \cos\left(\frac{i}{10000^{2j/d}}\right)$$

i: token position

j: pair index, starting at 0

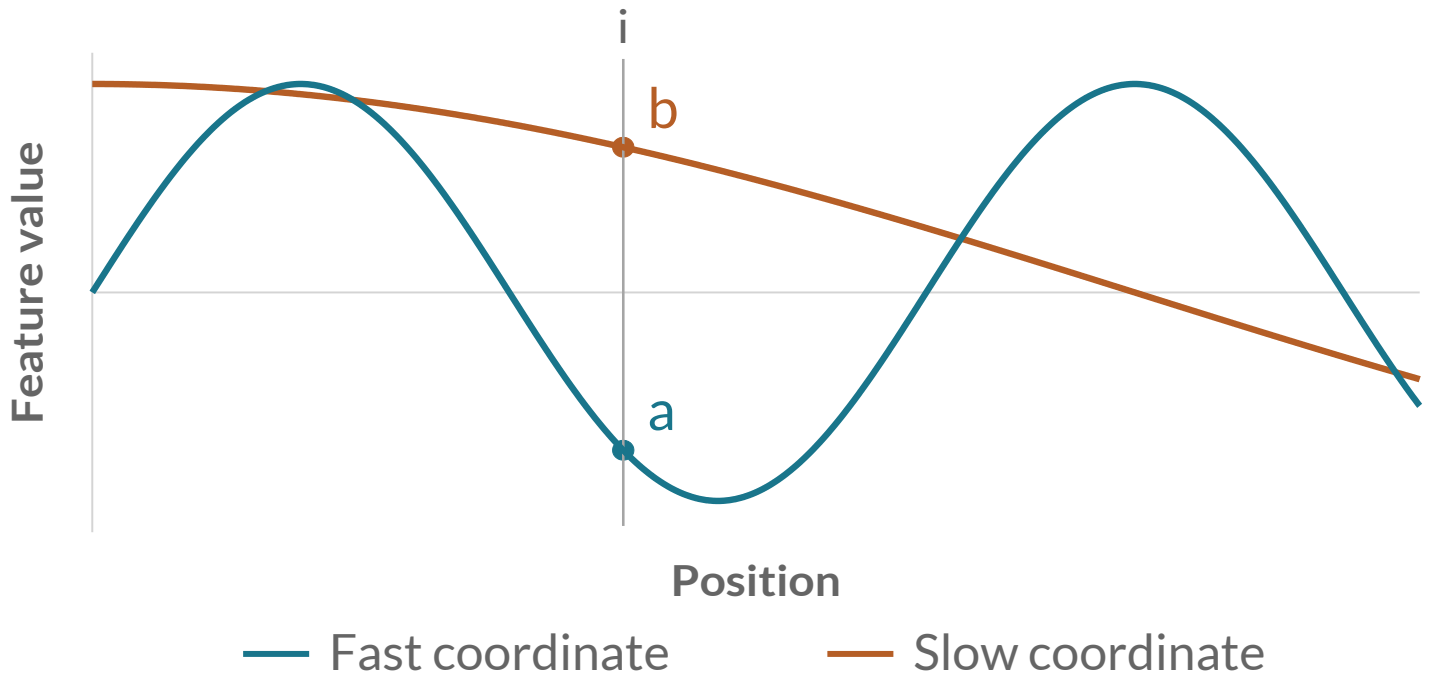
d: even vector width

$$j = 0, 1, \ldots, d/2 - 1$$

Each pair uses one frequency. Different pairs use different frequencies.

Sinusoidal position embeddings

At one position, read a value from each wave to form the position vector.



Position vector

$$p_i = \begin{bmatrix} a \\ \vdots \\ b \\ \vdots \end{bmatrix}$$

The original Transformer uses sine/cosine pairs at many rates.

Structure of sinusoidal positions

Fast pairs distinguish nearby positions.
Slow pairs vary over longer spans.

Each wave repeats, so combine several frequencies.

A fixed position shift has a simple effect on each pair

$$\sin\big(\omega(i+\Delta)\big) = \cos(\omega\Delta)\sin(\omega i) + \sin(\omega\Delta)\cos(\omega i)$$

ω: one pair's frequency Δ: a fixed shift in position

For the same shift, the coefficients are the same at every starting position.

Relative position

The relationship can stay the same when the phrase moves.

I walk **my** **dog** daily

one position apart

Every morning I walk **my** **dog** daily

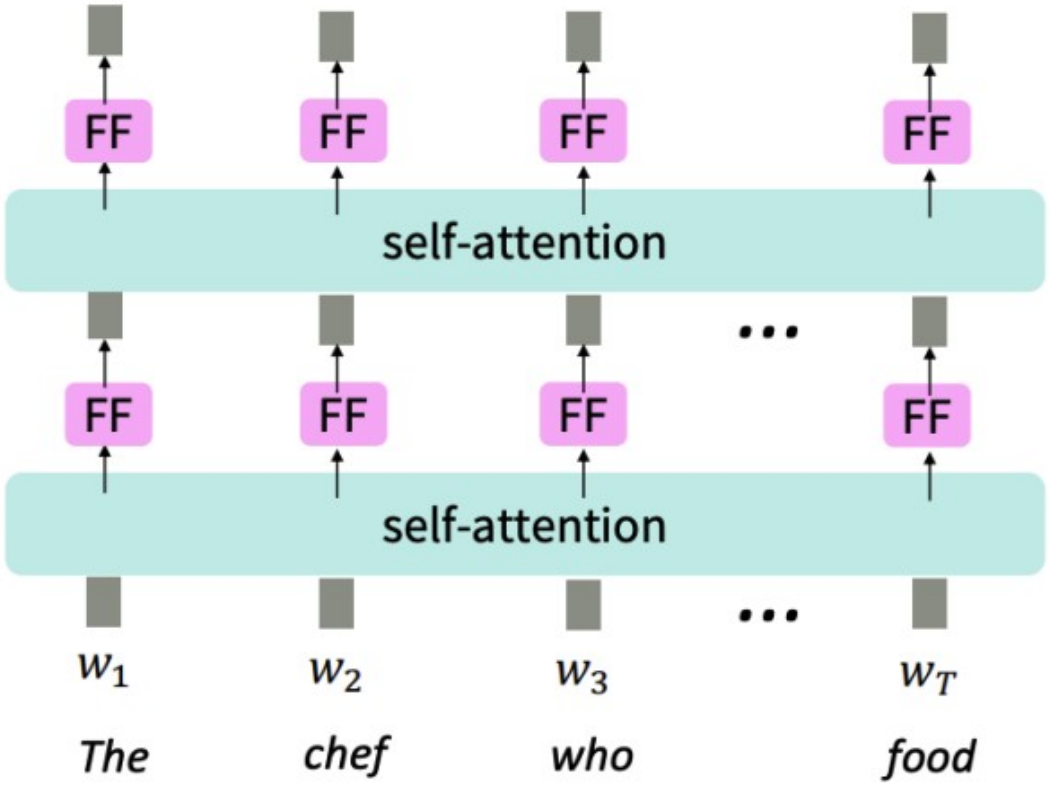
one position apart

$$e_{ij} = \underbrace{\frac{q_i k_j^\mathsf{T}}{\sqrt{d_k}}}_{\text{content match}} + \underbrace{b_{j-i}}_{\text{position bias}}$$

i: query position j: key position b: learned bias for their offset

One approach: add a learned offset bias before softmax.

Token mixing and feature transformation



Self-attention

Combines information across token positions.

Feedforward network (FF)

Builds nonlinear features from the retrieved context.

Source: Berkeley CS288

Position-wise feedforward network

The same learned transformation is applied at every position.

$$\mathrm{FFN}(x) = \underbrace{\sigma(xW_1 + b_1)}_{\text{wider hidden features}} W_2 + b_2$$

x: one contextual token vector σ : activation

Shared parameters

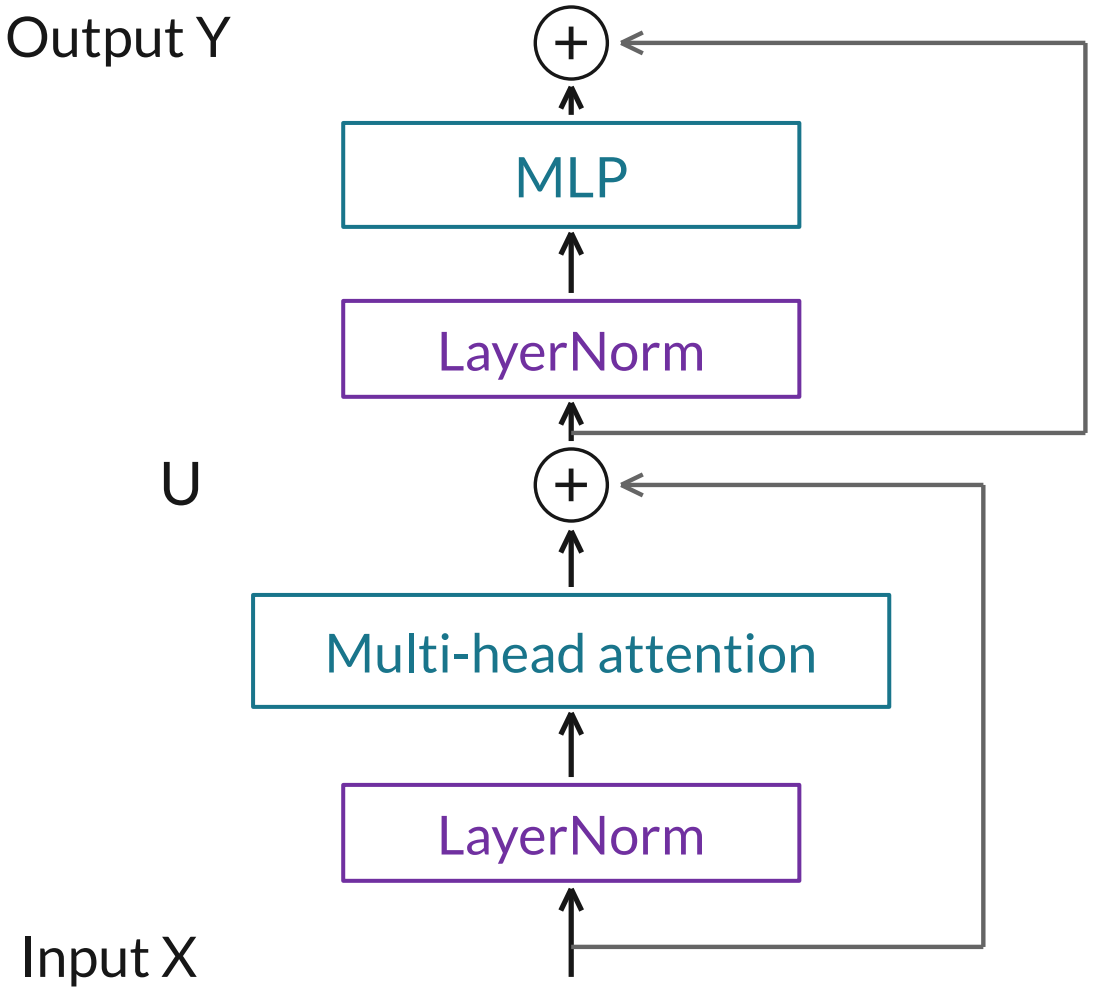
The same MLP at every position.

Separate computation

Each token is transformed independently.

The output returns to the model width for the residual addition.

Transformer encoder block



Pre-LayerNorm block

Attention update

$$U = X + \mathrm{MHA}(\mathrm{LN}(X))$$

Feature update

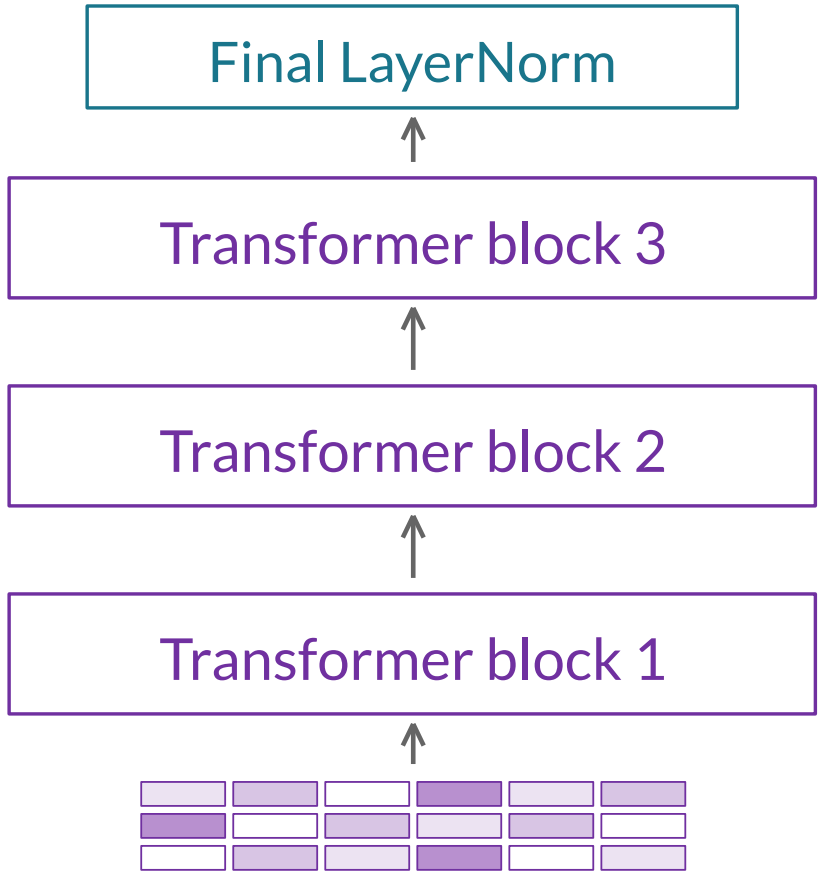
$$Y = U + \mathrm{MLP}(\mathrm{LN}(U))$$

LayerNorm: normalize features within each token.

Residual paths: a direct route through the stack.

Stacking Transformer blocks

Every block has its own learned parameters.



Same number of tokens
Same model width

Each block recomputes queries, keys and values.

Input token vectors

Transformer encoder in PyTorch

```
layer = nn.TransformerEncoderLayer(
    d_model=d, nhead=h,
    dim_feedforward=4*d, activation="gelu",
    batch_first=True,  # axes: batch, tokens, features
    norm_first=True,   # LayerNorm before each sublayer
)
encoder = nn.TransformerEncoder(
    layer, num_layers=L, norm=nn.LayerNorm(d)
)
y = encoder(x)          # same shape as x
```

d: model width h: attention heads L: number of blocks

Lecture outline

01 Recap: RNNs and attention

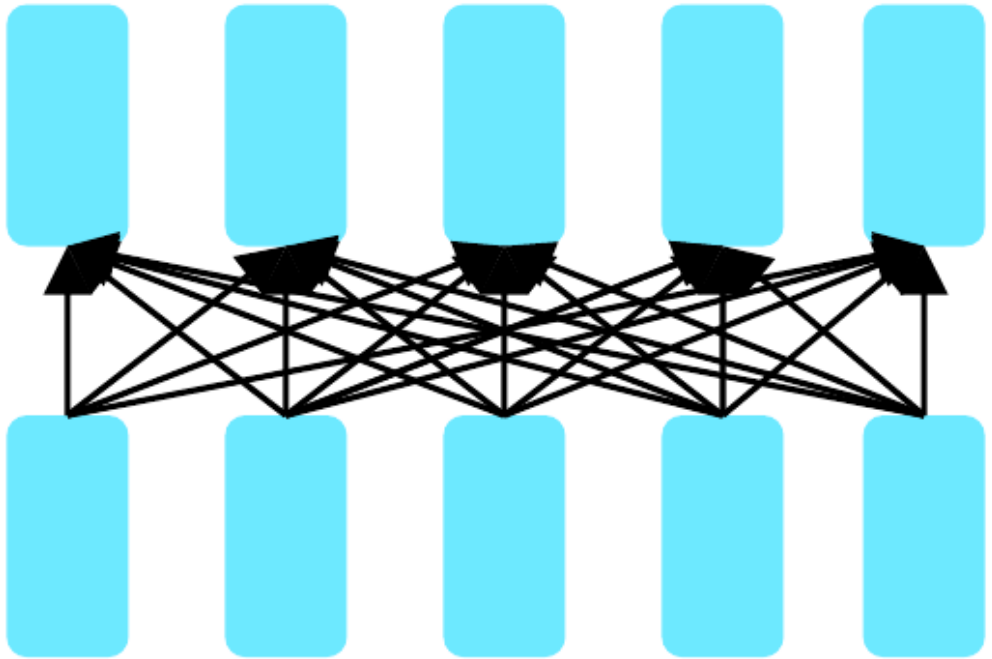
02 Building a Transformer

03 Transformer architectures

04 Transformers for vision

Encoder-only model

Contextual token features



Input token features

Source: Stanford CS224n

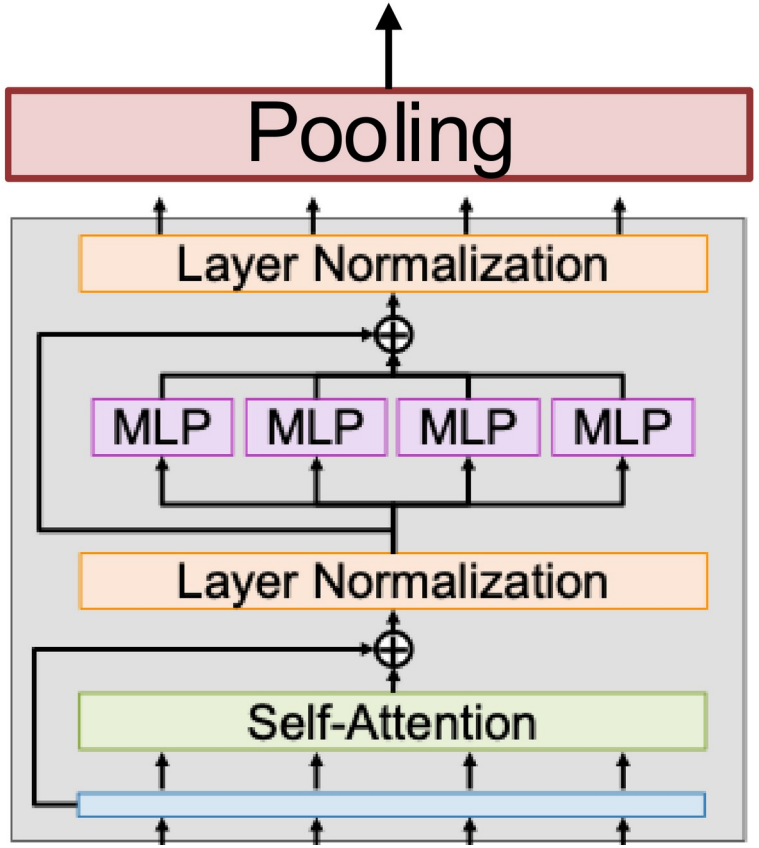
Every token can read the entire input.

This full-context access is repeated at every encoder layer.

BERT uses these contextual features for token or sequence predictions.

Sequence classification

One vector per token. One class prediction for the whole input.



Average the final token features

$$h = \frac{h_1 + \cdots + h_N}{N}$$

h_i: final token feature N: number of tokens

Apply a classification head

$$p = \text{softmax}(Wh + b)$$

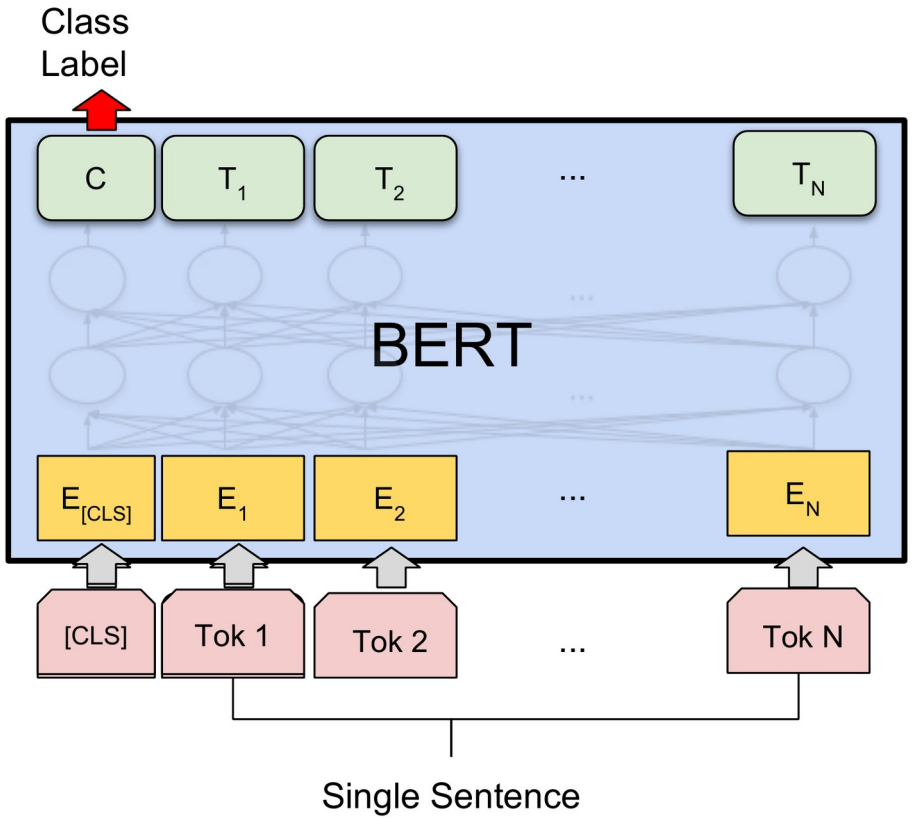
p: class probabilities W, b: classifier parameters

Train the encoder and head with cross-entropy.

Pooling figure: Stanford CS231n

The [CLS] token

[CLS] is a special classification token added before the input.



Learned input

One starting embedding, shared across sentences.

Contextual output

Self-attention gathers information from this input into its final feature C.

$$p = \text{softmax}(WC + b)$$

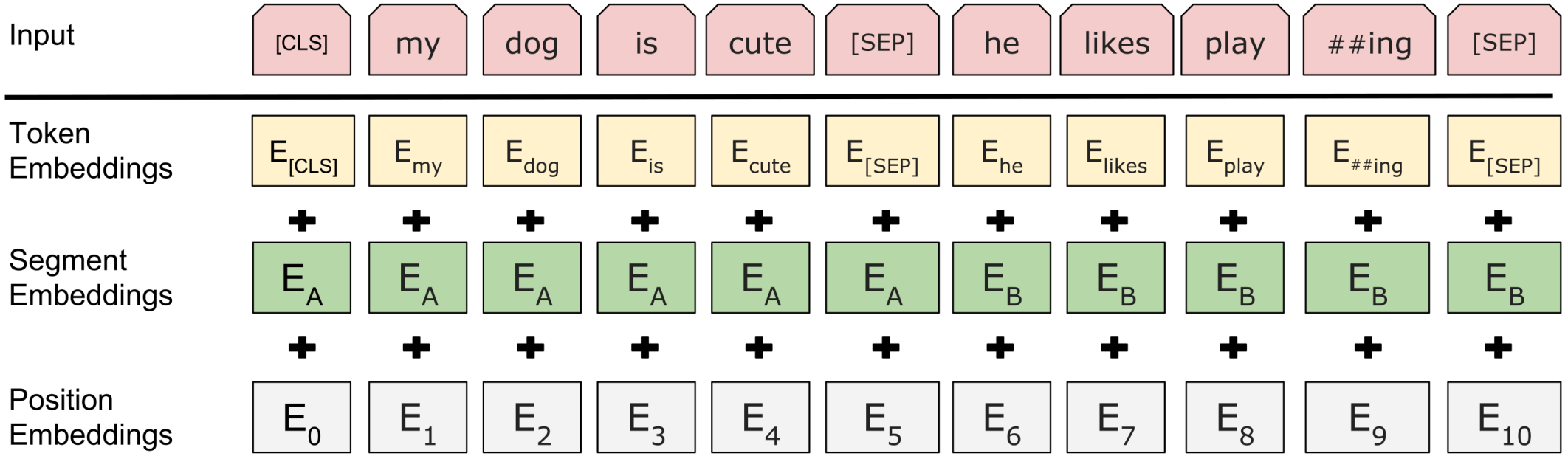
Cross-entropy trains the encoder and classification head together.

Source: BERT, Figure 4

BERT input embeddings

Bidirectional Encoder Representations from Transformers

At each token: add three learned vectors.

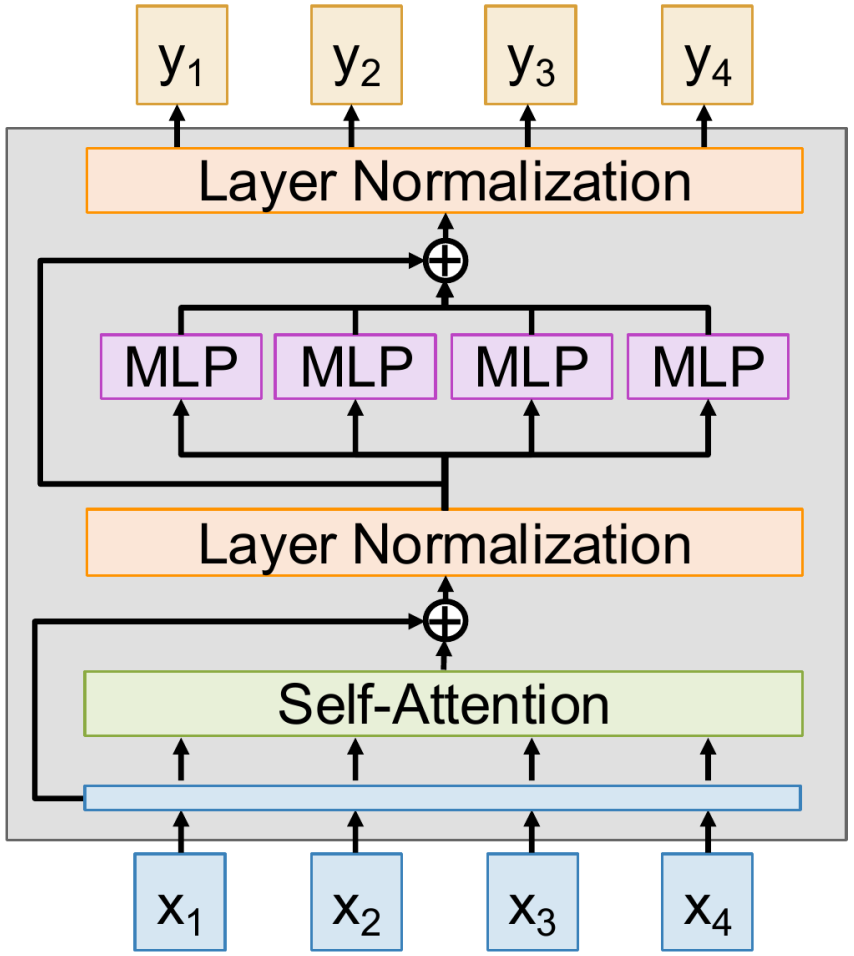


[CLS]: classification token [SEP]: text boundary

A/B identify the two texts. "##ing" continues a word.

Source: BERT, Figure 2

BERT encoder



x: block inputs y: updated token features

Block figure: Stanford CS231n

Input

Token + position + segment vectors, followed by normalization.

Repeated encoder blocks

Full-context attention and a token-wise MLP with GELU activation.

LayerNorm follows each residual addition.

Output

One contextual vector per token.

BERT: learning contextual features

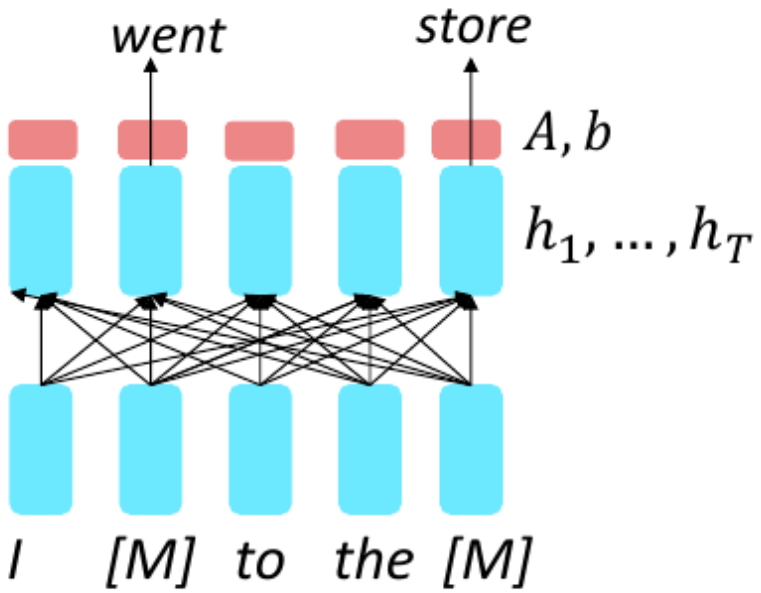
Pretraining: cross-entropy on the hidden tokens.

Encoder-only model

Each token can use visible context on both sides.

Masked language modeling

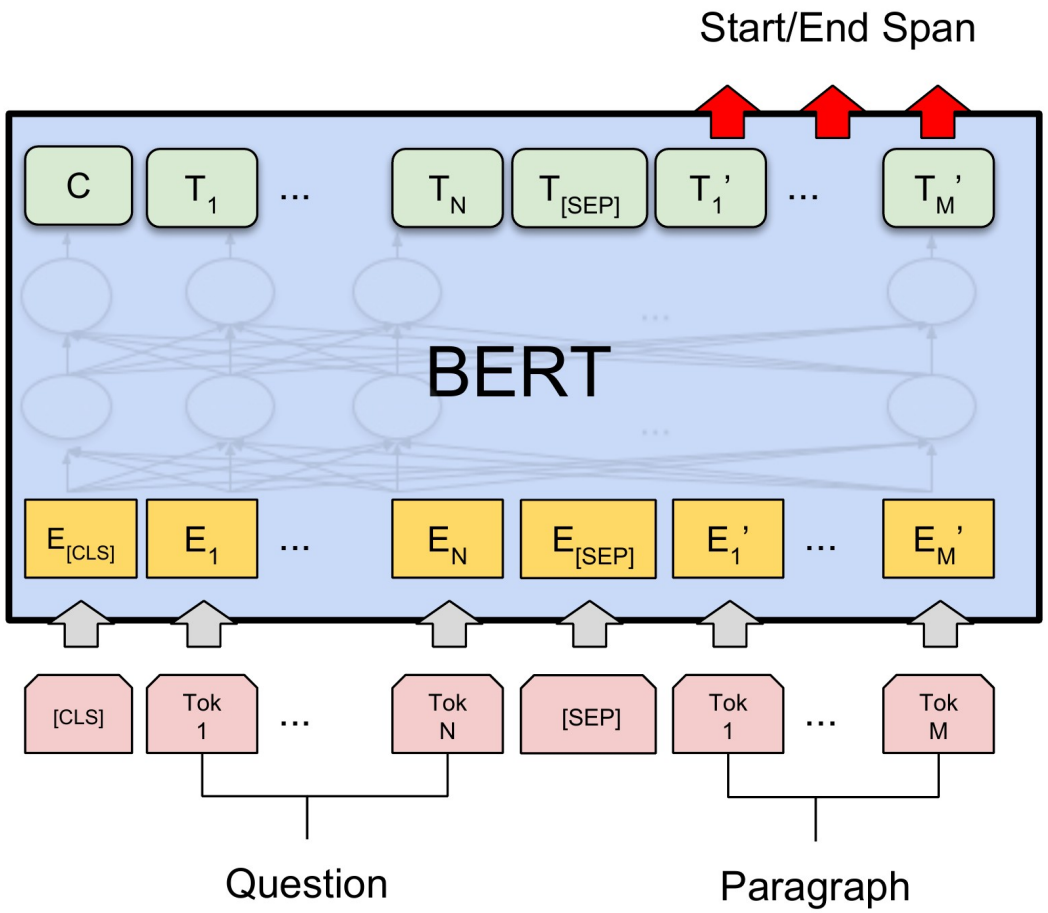
Hide tokens to prevent copying.
Predict them from visible context.



[M]: masked input h: token features
A, b: vocabulary-prediction head

Figure: Stanford CS224n

Extractive question answering



Select an answer from the passage

Input: question + [SEP] + passage.

Two output heads score
the answer's start and end positions.

Return the tokens between
the predicted positions.

E: input embeddings
T: contextual token features

Source: BERT, Figure 4

Task-dependent information access

BERT: classification

“The film was surprisingly good.”

The whole observed sentence is available.

Next-token prediction

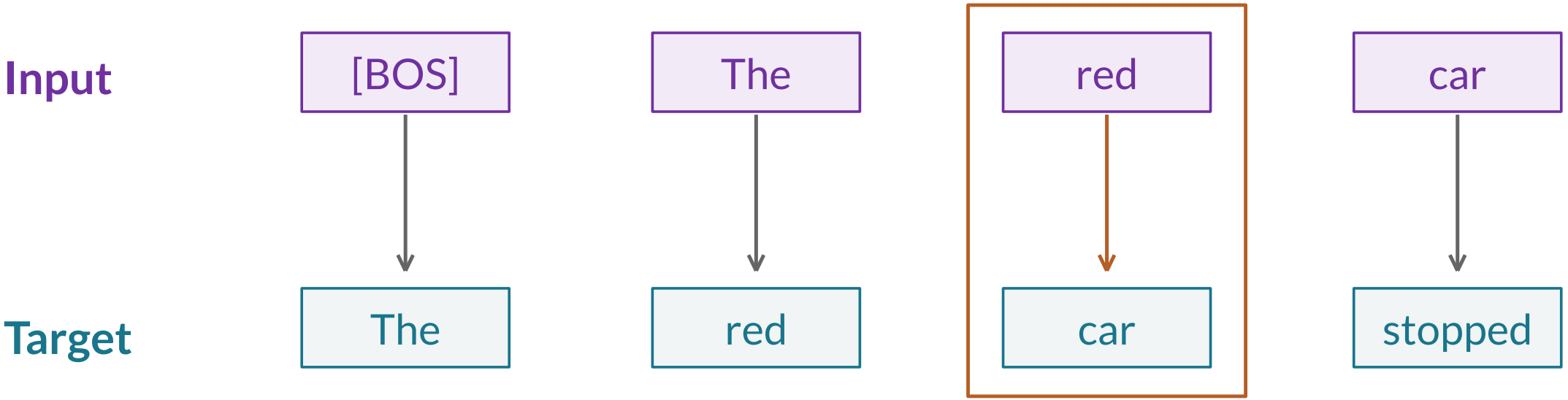
“The film was” Predict the next token

The representation must not read future tokens.

Shifted inputs and targets

Training sequence: The red car stopped

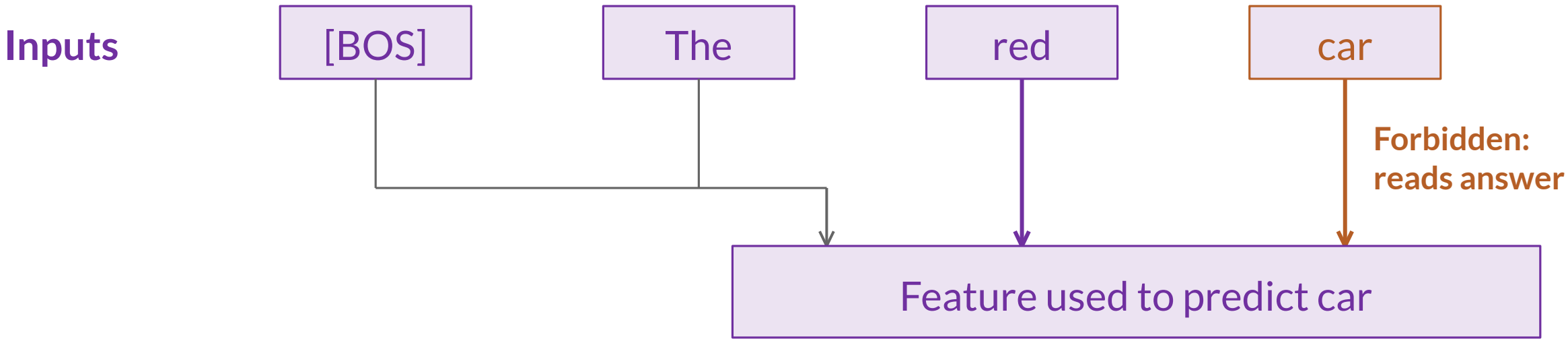
[BOS] = beginning of sequence



At this position: red is known, car is the answer.

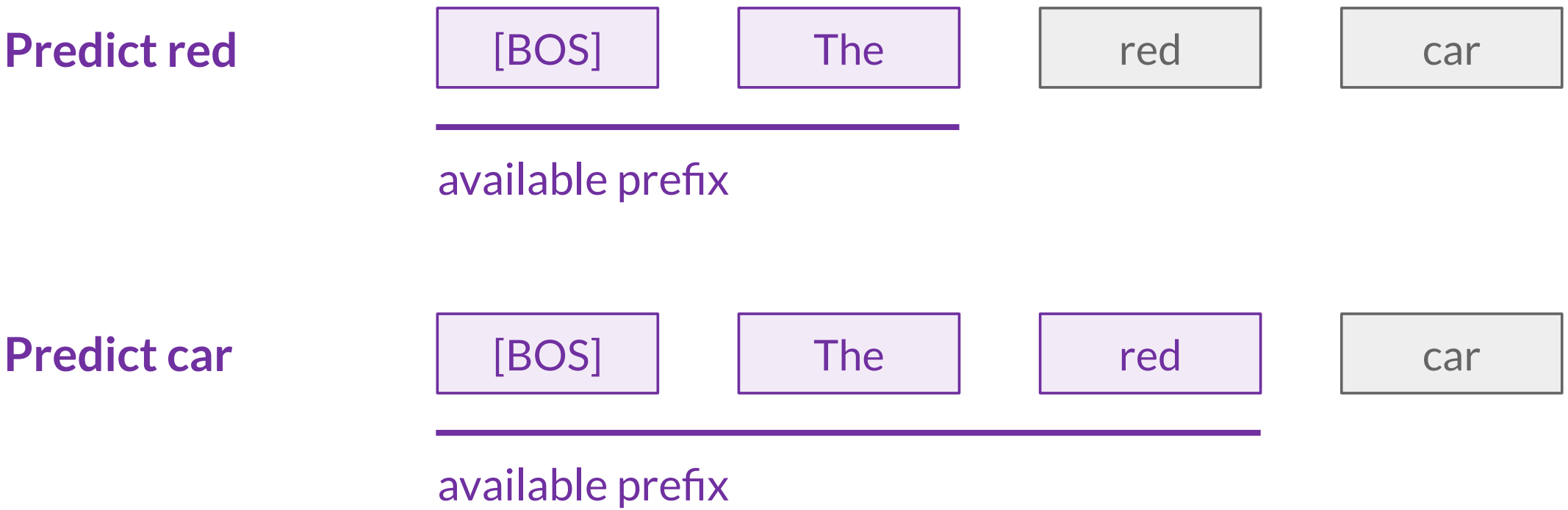
Future-answer leakage

At the third position, the target is car.



Reading the answer can lower training loss.
That information is unavailable during generation.

Permitted prefixes



The current input is allowed. Later inputs are hidden.

Causal attention mask

Key / value positions: inputs being read

Query positions: features being computed

	[BOS]	The	red	car
[BOS]	✓	×	×	×
The	✓	✓	×	×
red	✓	✓	✓	×
car	✓	✓	✓	✓

Allowed: current input and earlier inputs

Masking before softmax

$$A = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right)$$

A: attention weights M: mask added to the scores

Query: red

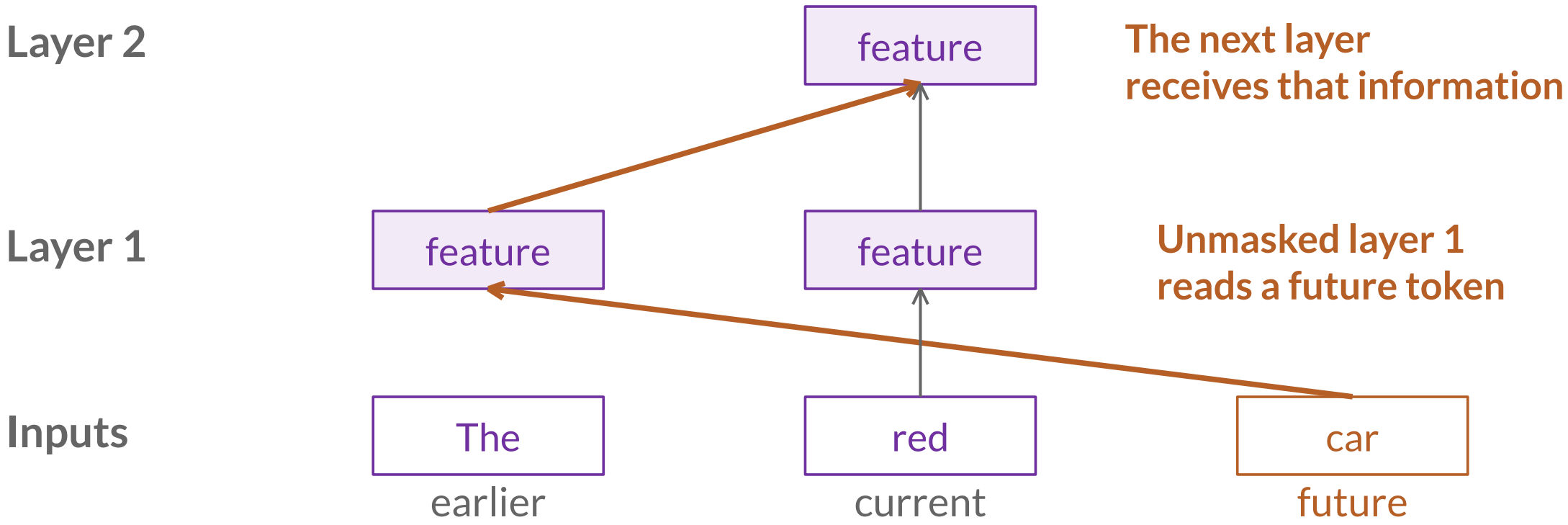
Score row	s1	s2	s3	s4
Mask row	0	0	0	−∞

Allowed scores stay unchanged.

The forbidden position receives zero attention weight.

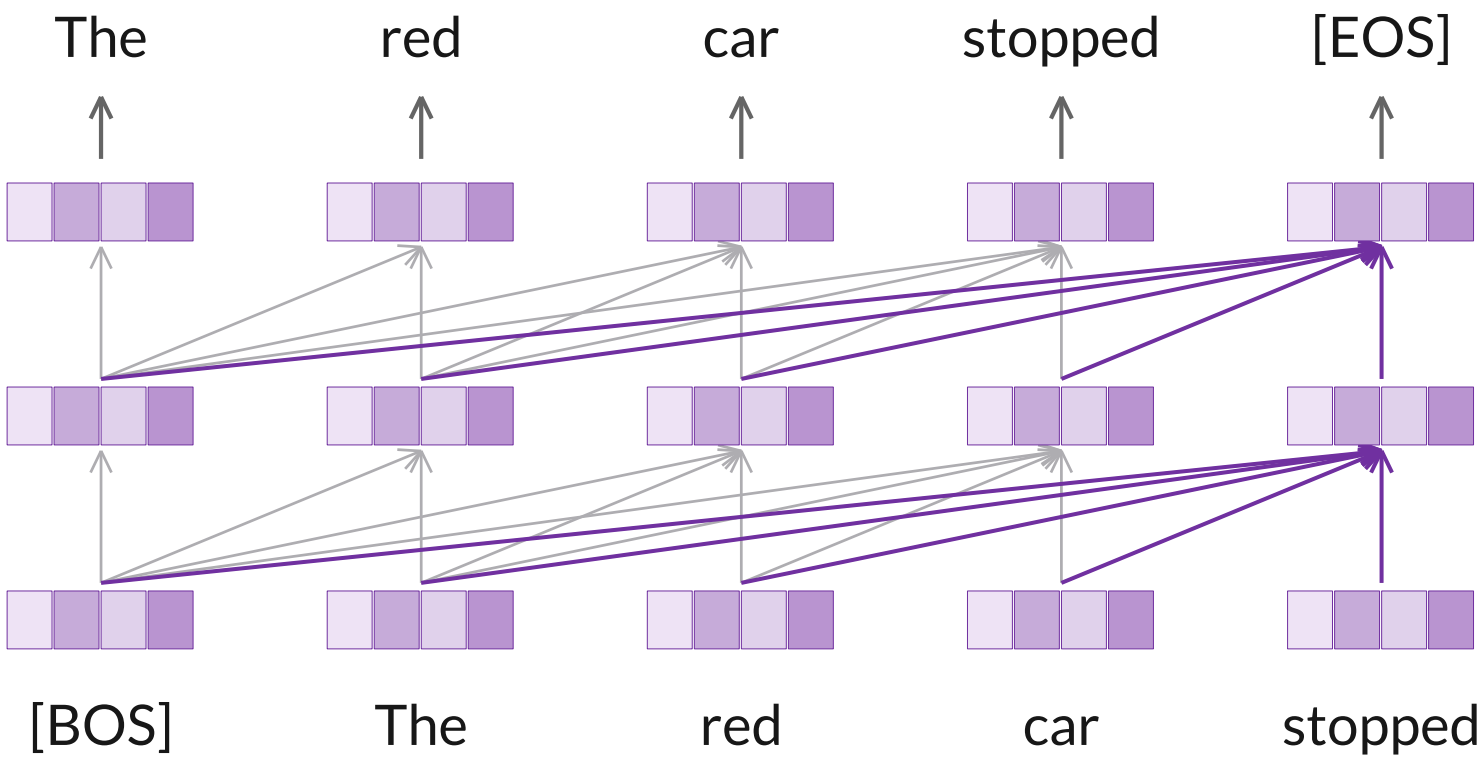
Causal masking in every layer

Masking only the last layer still permits an indirect leak.



Decoder-only model

GPT (Generative Pretrained Transformer) uses a causal decoder.



Training targets

Shared linear layer and softmax

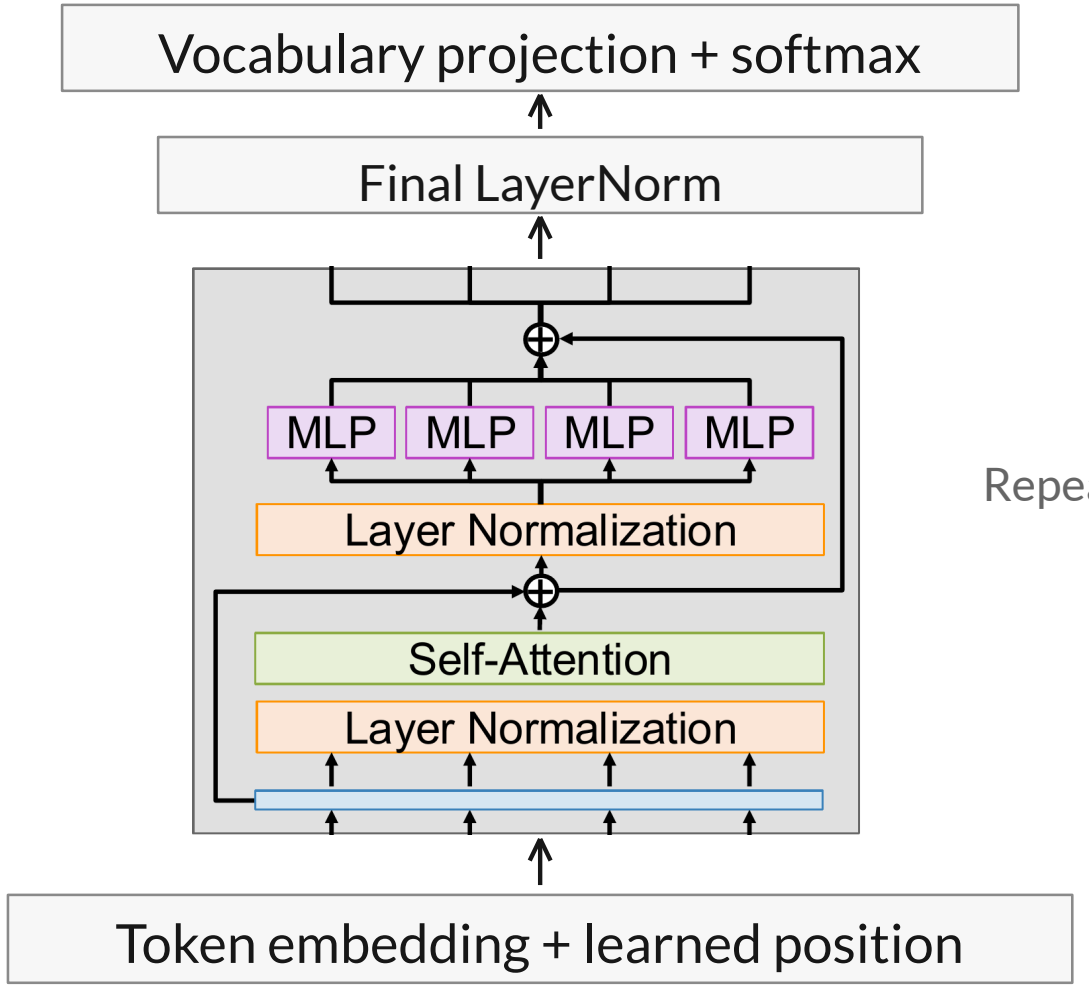
Causal attention in every layer

No separate encoder or cross-attention.

Two layers shown. Each position reads only its prefix.

Connections adapted from Stanford CS224n

GPT-2 architecture



Repeat blocks

Causal self-attention

Each token uses its prefix.
No source encoder or cross-attention.

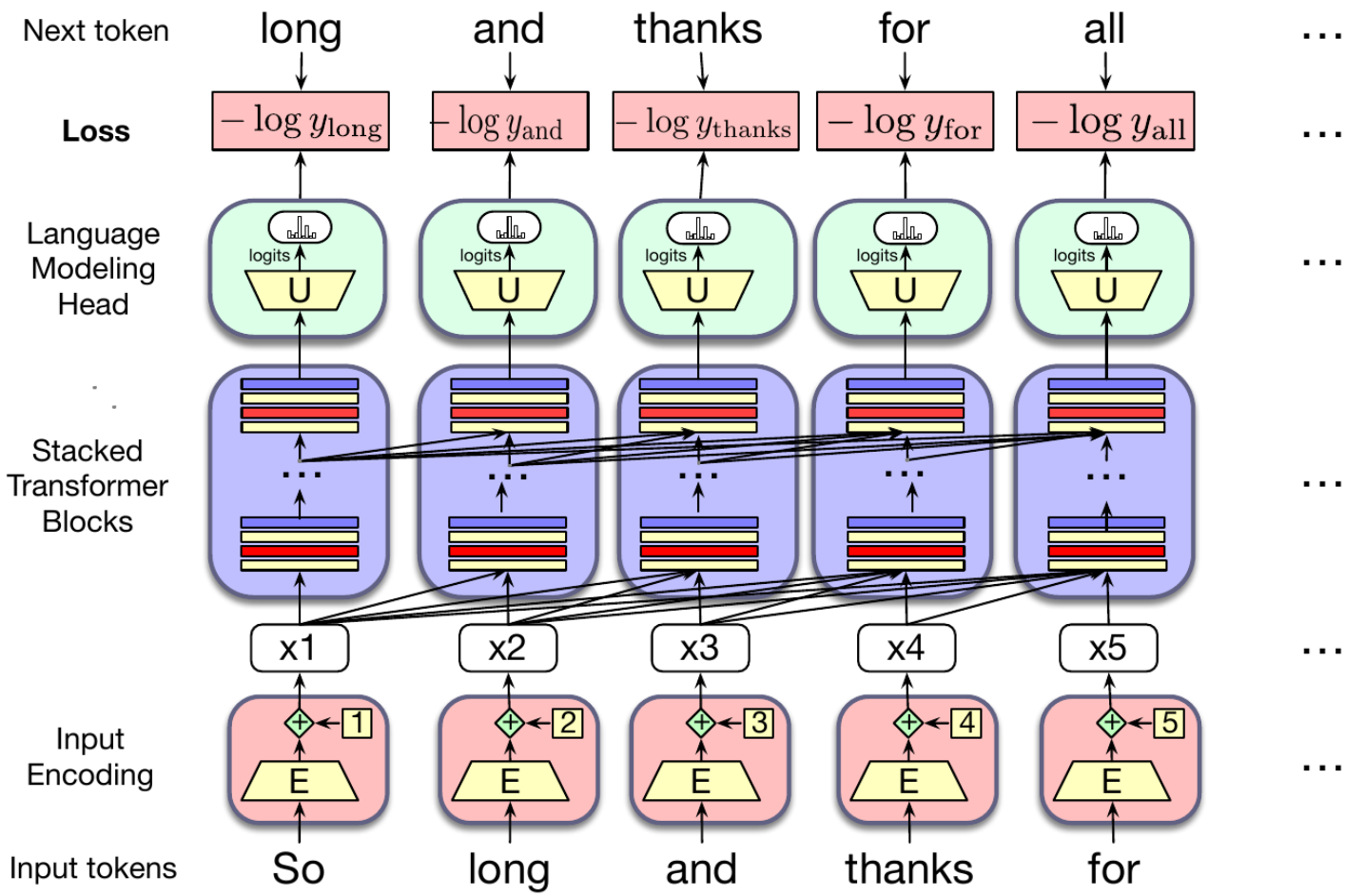
Pre-normalization

LayerNorm sits inside the residual branches, keeping the skip path direct.

GELU in the MLP.
The vocabulary projection reuses the token embedding weights.

Block adapted from Stanford CS231n

Parallel language-model training



$$= \frac{1}{T}\sum_{t=1}^{T} L_{CE}$$

All next-token targets are known during training.

Causal masking keeps each prediction within its prefix.

E: embedding
U: vocabulary projection
x: token feature
$\hat{y}$: token probabilities
T: number of targets
CE: cross-entropy loss

Source: Jurafsky & Martin, Speech and Language Processing

Padding-key masks

Sequences in a batch

Example 1	The	red	car	stopped
Example 2	Birds	fly	[PAD]	[PAD]

Padding keys are excluded from attention.

Padding makes a rectangular batch. It is not sequence content.

Attention masking and loss masking

Attention mask

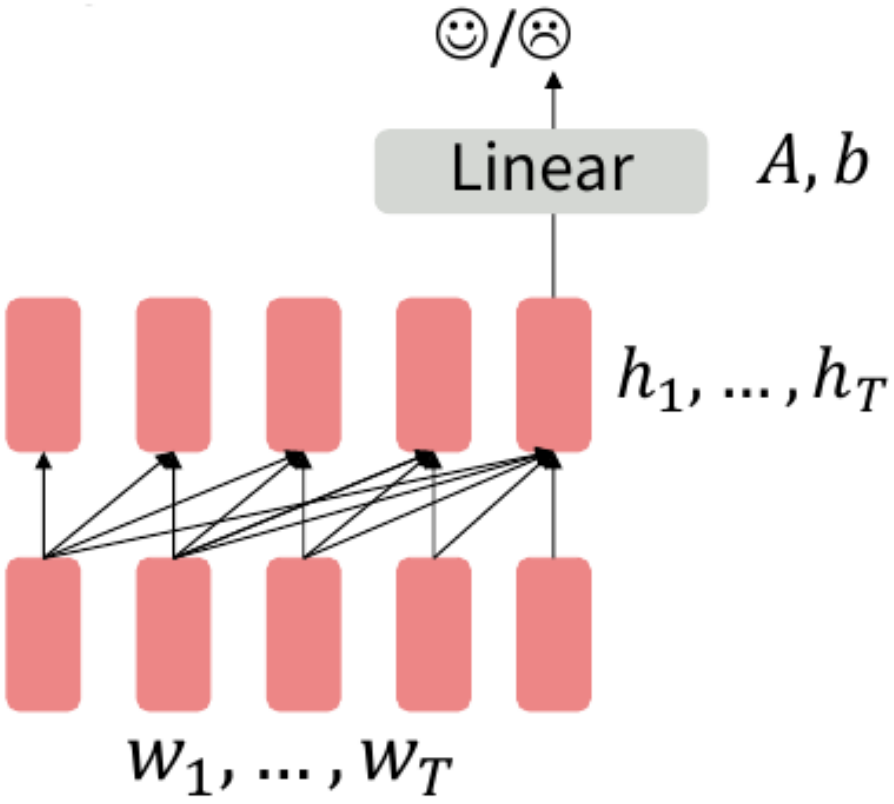
Keys	[BOS]	Birds	fly	[PAD]
Query at fly	**read**	**read**	**read**	**ignore**

Loss mask

Targets	Birds	fly	[EOS]	[PAD]
Loss	**count**	**count**	**count**	**omit**

[EOS] = end of sequence

Original GPT: finetuning



Use the final token feature

The final token can attend to the whole input.

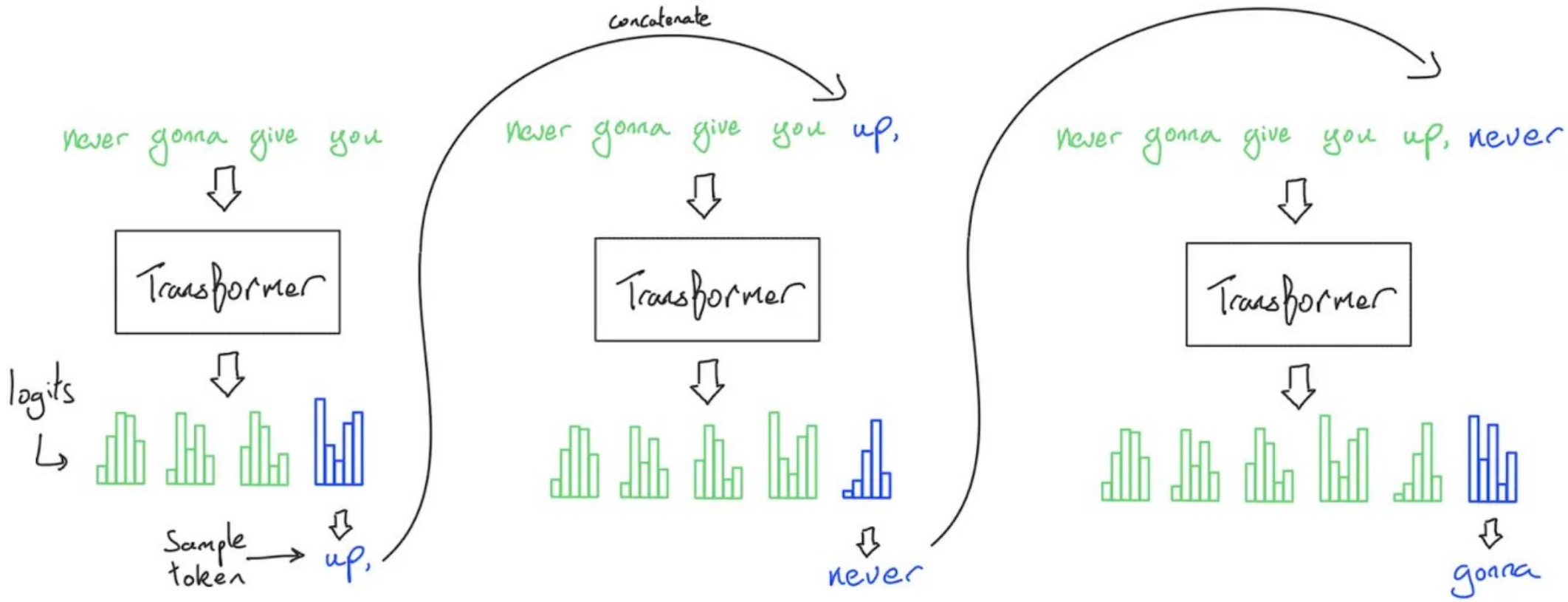
Add a classification head and finetune on labeled data.

w: input tokens h: token features
A, b: classifier parameters

Figure: Stanford CS224n

Repeated prefix computation

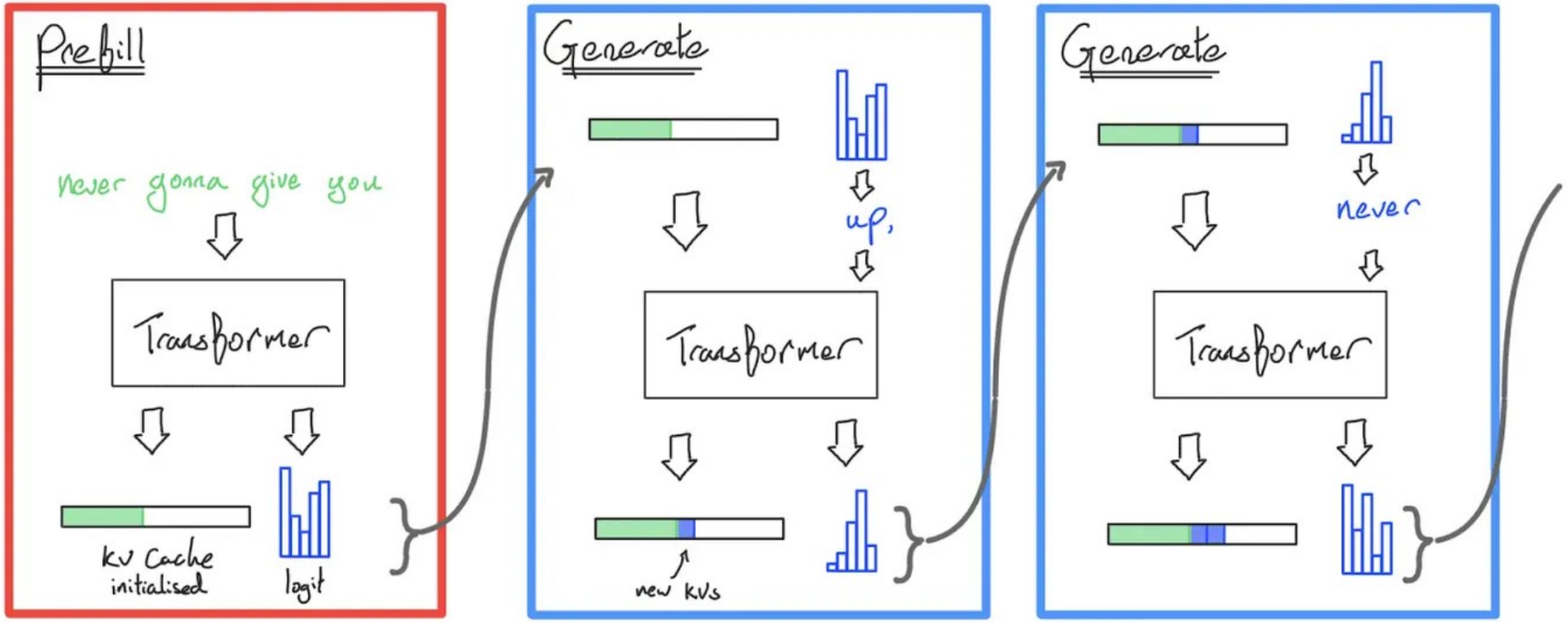
If we rerun the full prefix, earlier token features are computed again.



Source: JAX Scaling Book

Key–value caching

Prefill: process the prompt. KV cache: saved keys and values per layer.



Earlier tokens cannot read new tokens, so their keys and values stay unchanged.

Source: JAX Scaling Book

Recurrent state and retained context

RNN / LSTM

One fixed-size recurrent state summarizes the prefix.

Each new token updates that state.

State storage stays constant.

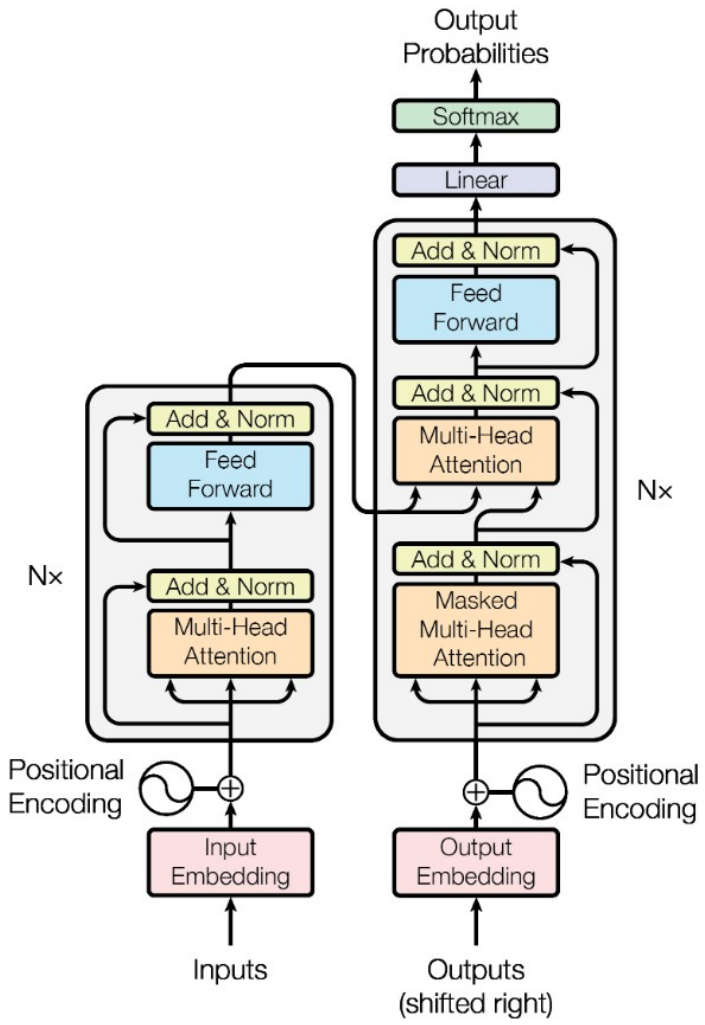
Causal Transformer

Keep keys and values for each earlier token.

New tokens read and extend the cache.

Cache storage grows with the prefix.

Encoder–decoder Transformer



Translation: the whole source is known.
Target tokens are generated one at a time.

Encoder

Source tokens use full-context self-attention.

Decoder

Target tokens use causal self-attention.

Cross-attention

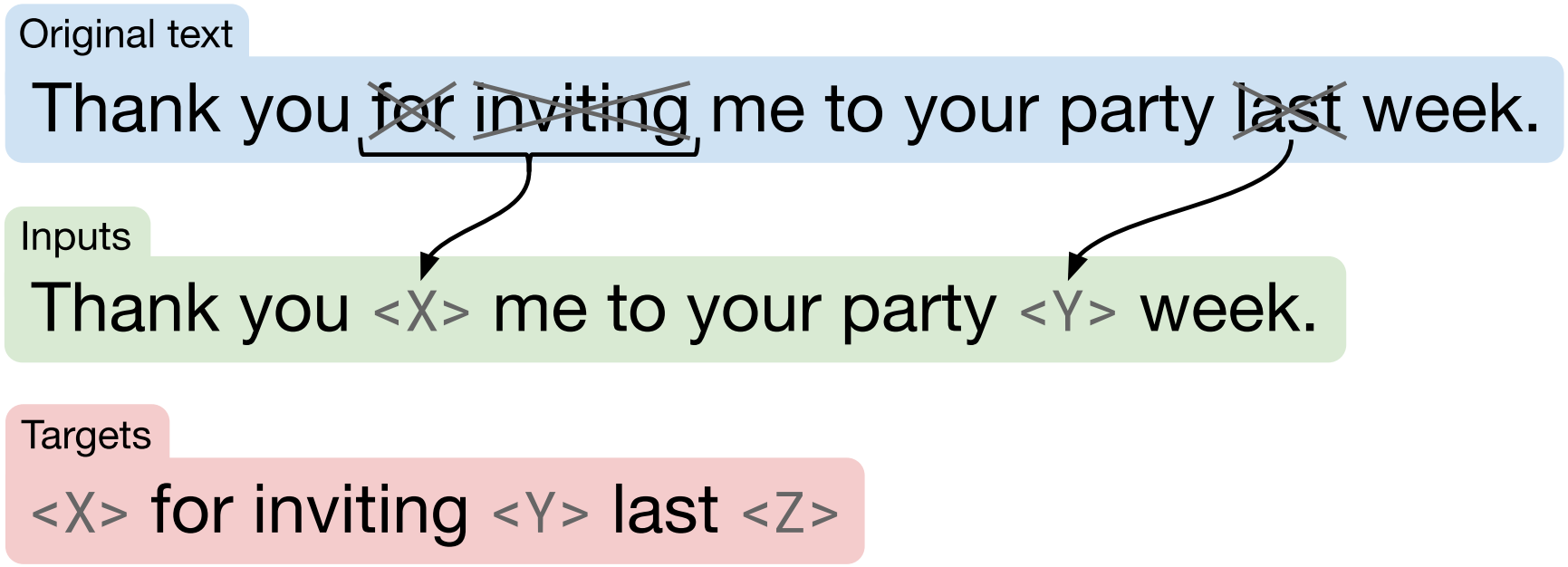
Target queries match source keys and combine source values.

N×: repeated blocks. Norm follows residual addition.

Source: Attention Is All You Need

T5: span reconstruction

Text-to-Text Transfer Transformer · pretraining



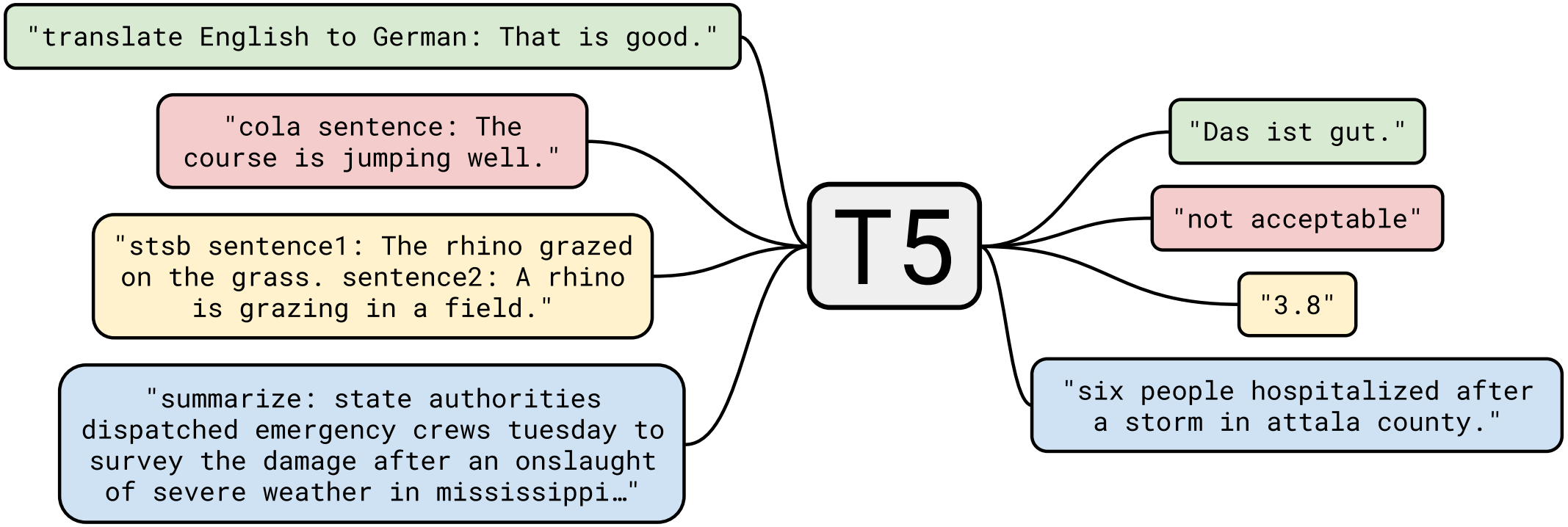
Encoder input: text with missing spans.
Decoder target: the removed spans and their markers.

<X>, <Y>: span markers <Z>: final span marker

Source: T5, Figure 2

T5: text-to-text tasks

Supervised finetuning uses text input–target pairs.



cola: grammatical acceptability stsb: sentence similarity

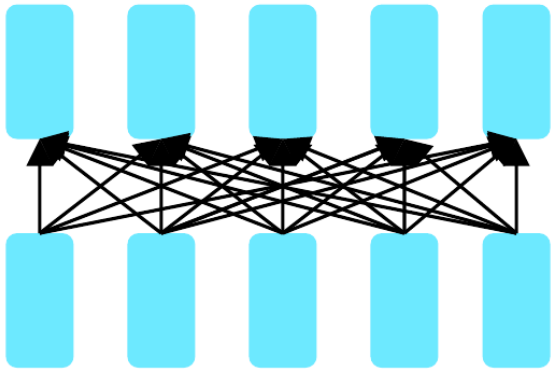
Class labels use the vocabulary output, rather than a separate class head.

Source: T5, Figure 1

Transformer architectures

Encoder-only

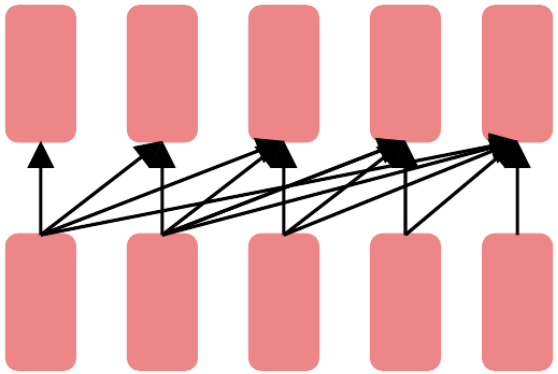
BERT



Full input context

Decoder-only

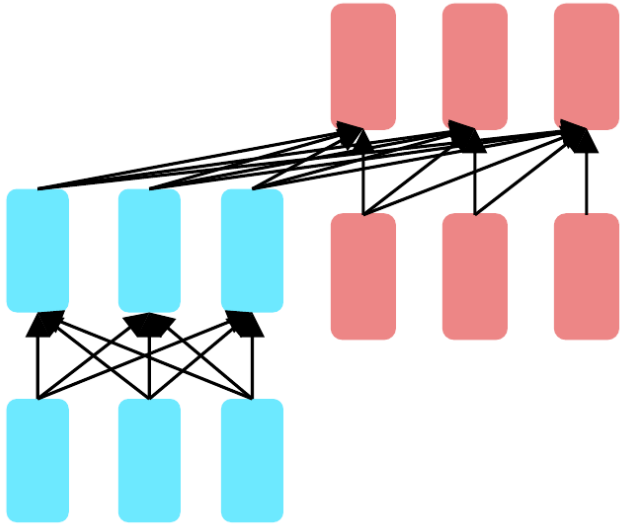
GPT



Prefix only

Encoder–decoder

T5



Source + target prefix

Rectangles: token features. Arrows: information access.

Source: Stanford CS224n

Lecture outline

01 Recap: RNNs and attention

02 Building a Transformer

03 Transformer architectures

04 Transformers for vision

Vision Transformer (ViT)

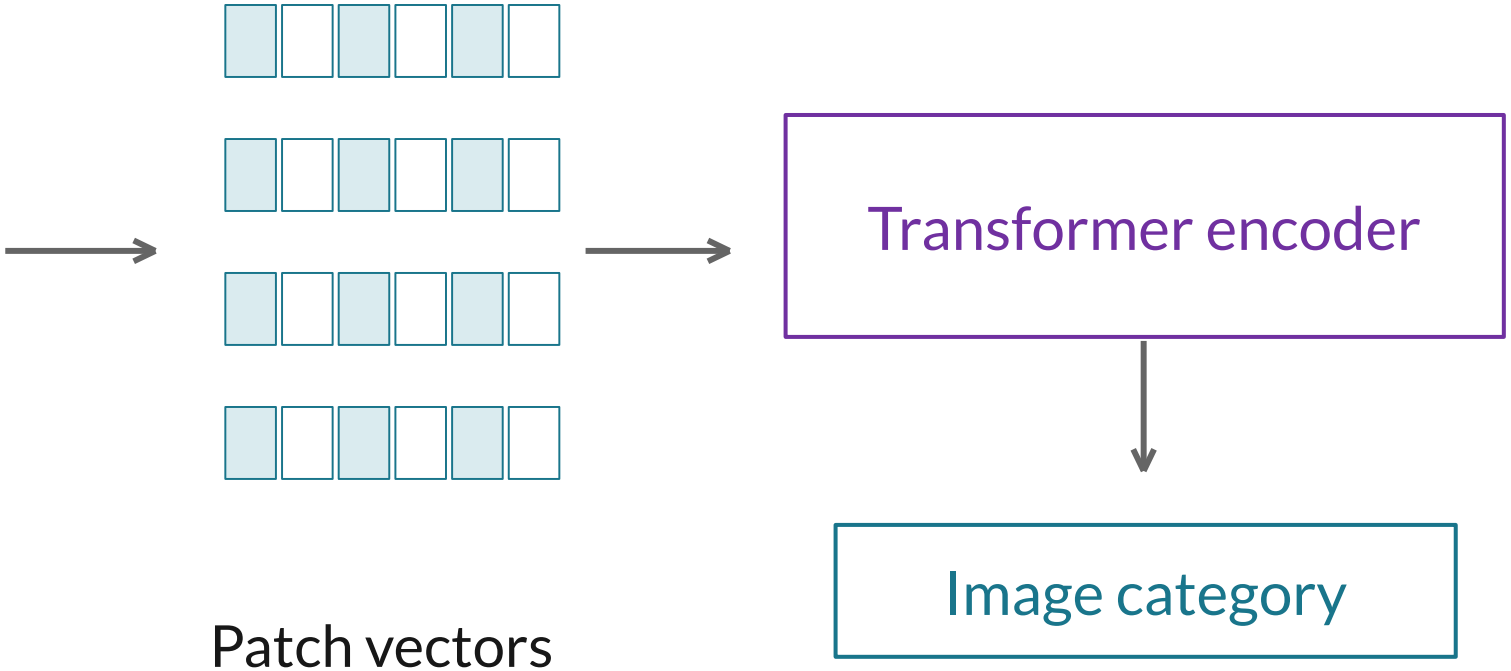


Image classification

Represent image regions as vectors, then let them exchange information.

Source: Photo: Fredo Durand / MIT 6.7960

Pixels and sequence length

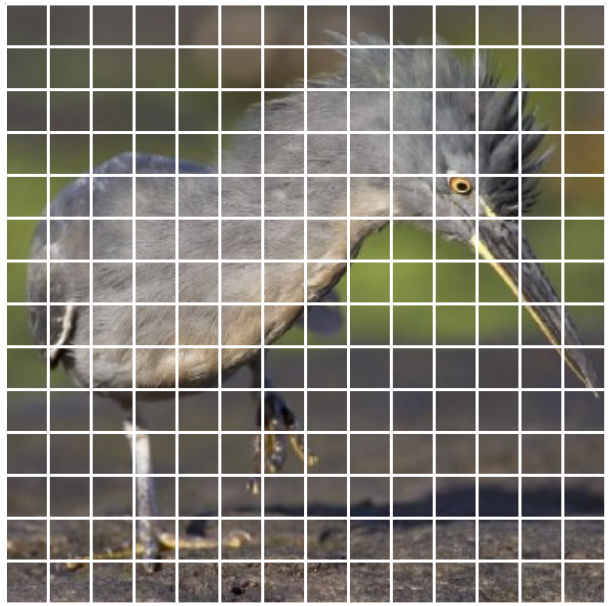
One token per pixel

$224 \times 224 = 50{,}176$

pixel tokens

Group local pixels first

One token per patch



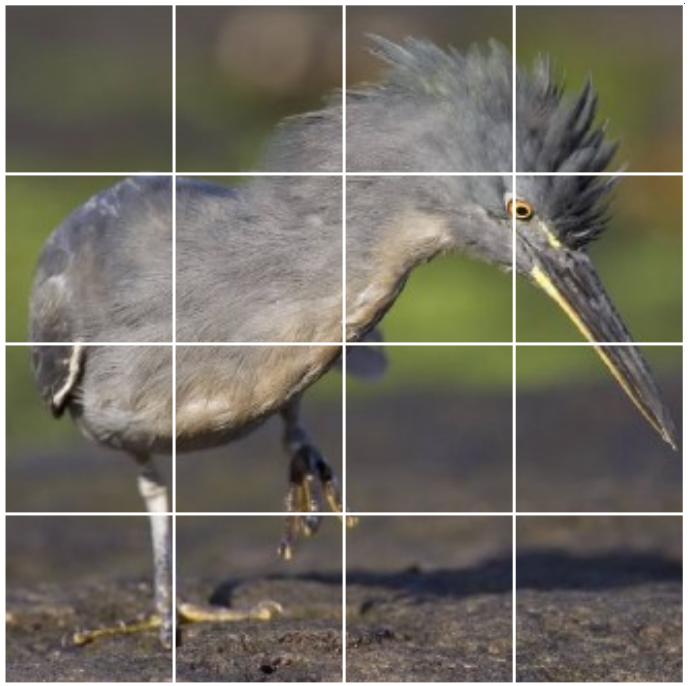
$14 \times 14 = 196$

patch tokens (16 × 16 pixels)

Global attention compares every pair of tokens.

Source: Photo: Fredo Durand / MIT 6.7960

Images as patches



A 4 × 4 patch grid

Store the patches in a fixed row order.

Each patch contains pixel values from a small image region.

Source: Photo: Fredo Durand / MIT 6.7960

Flattening one patch

Flatten

Pixel values in a fixed order

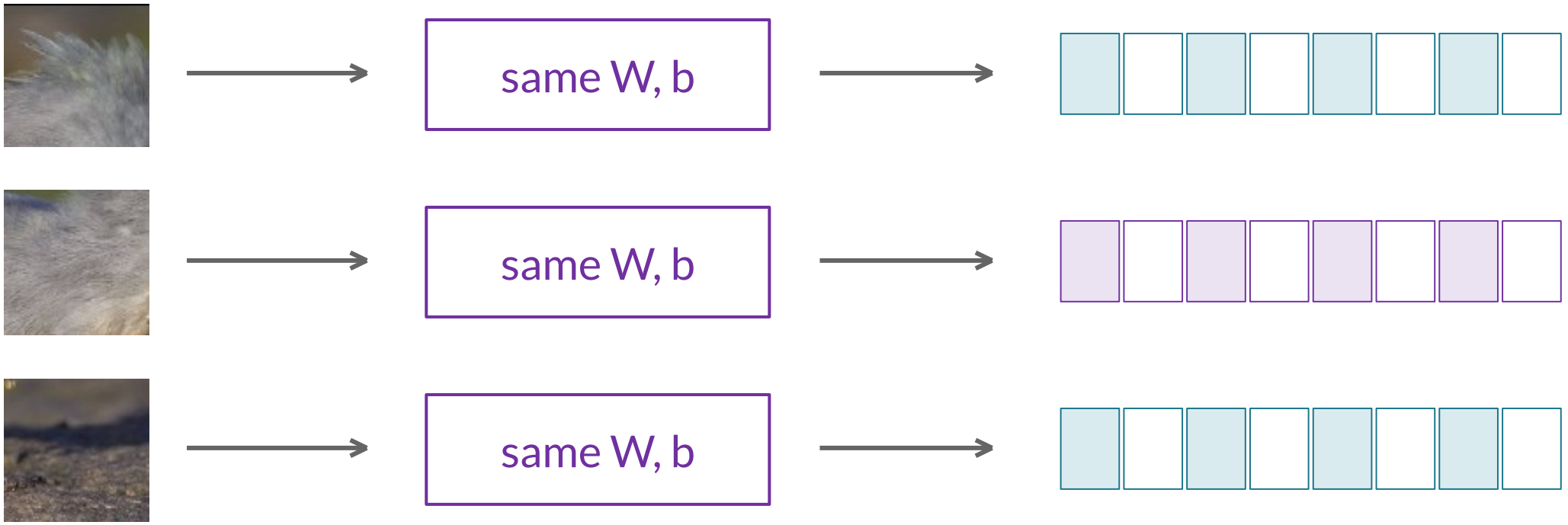
One RGB patch

$$16 \times 16 \times 3 = 768 \text{ values}$$

Flattening rearranges the values. It does not average them.

Source: Photo: Fredo Durand / MIT 6.7960

Patch embeddings



$$\text{embedding} = (\text{flattened patch})\, W + b$$

Patch pixels

Weights W, bias b

Embedding width d

Source: Photo: Fredo Durand / MIT 6.7960

Patch embedding in PyTorch

Input x: a batch of RGB images, shape [B, 3, 224, 224]

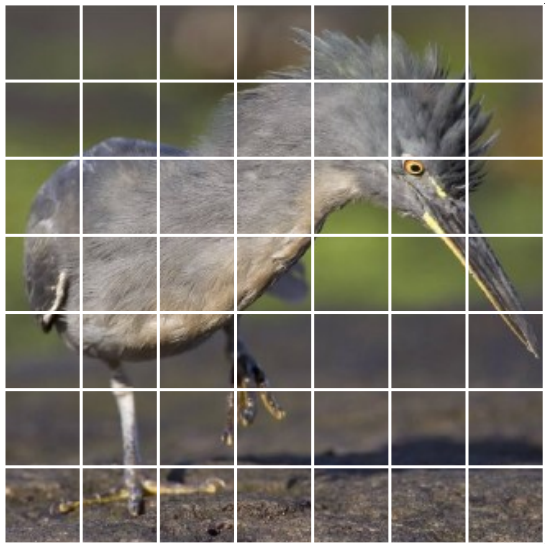
```
patch_embed = nn.Conv2d(3, d, kernel_size=16, stride=16)

z = patch_embed(x)      # [B, d, 14, 14]
z = z.flatten(2)        # [B, d, 196]
z = z.transpose(1, 2)   # [B, 196, d]
```

B = batch size d = model feature width

Shared patch projection = convolution with kernel size = stride = patch size.

Patch size

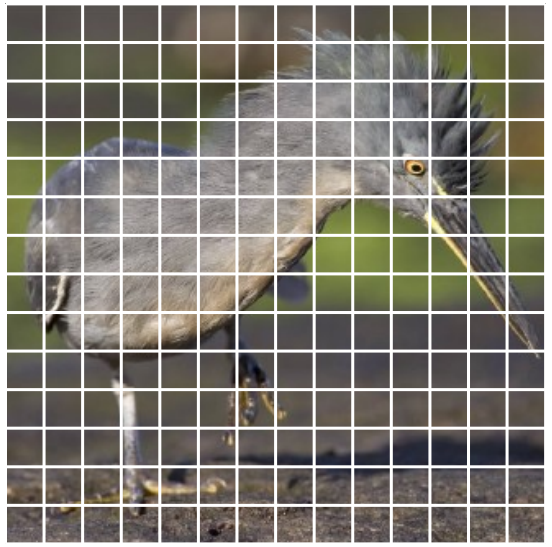


32 × 32 pixel patches

49 patch tokens

Same 224 × 224 image

Finer spatial granularity



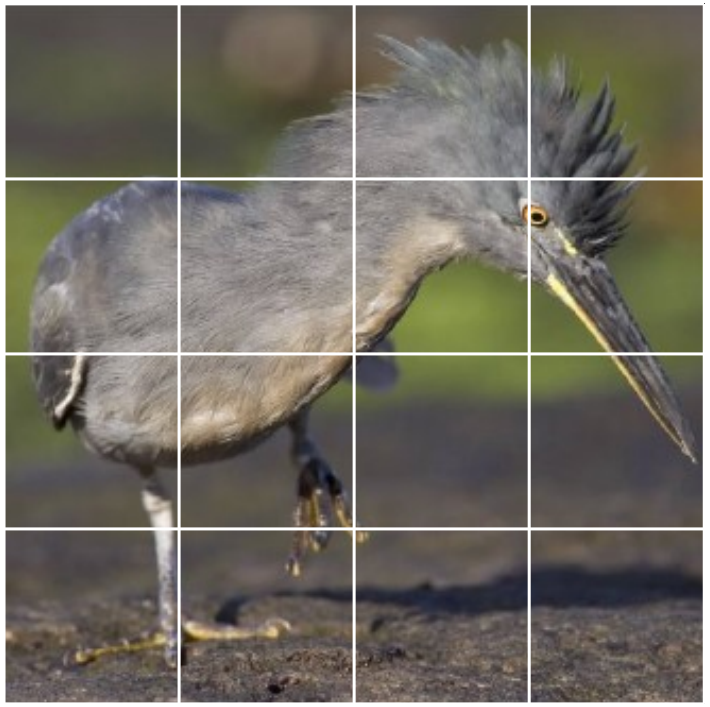
16 × 16 pixel patches

196 patch tokens

4× tokens → 16× patch-to-patch attention scores

Source: Photo: Fredo Durand / MIT 6.7960

Spatial arrangement



Original arrangement

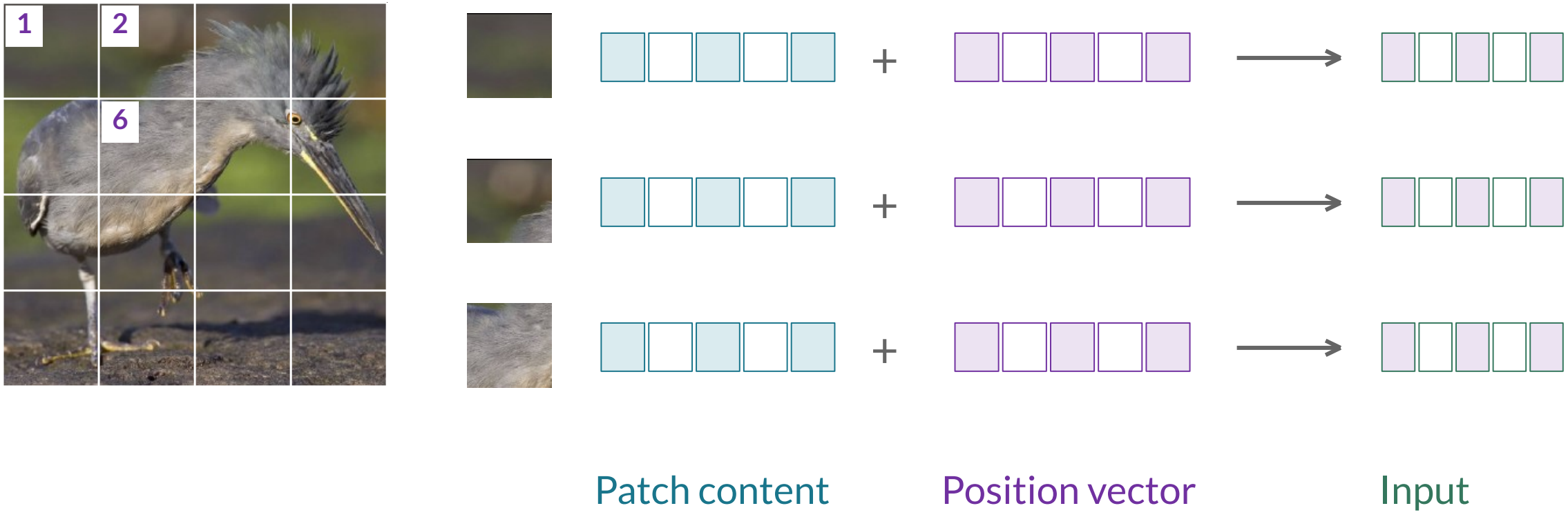
Same patch contents
Different spatial layout

Same patches, rearranged

Patch contents alone do not tell attention where each patch belongs.

Source: Photo: Fredo Durand / MIT 6.7960

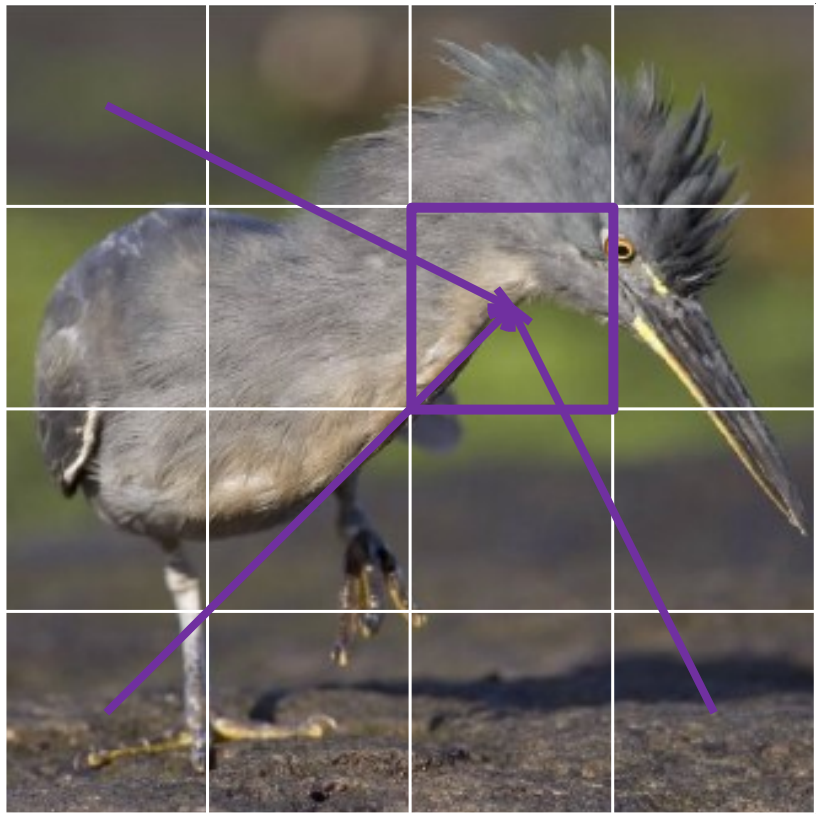
Patch positions



A learned position vector is shared by all images at the same grid location.

Source: Photo: Fredo Durand / MIT 6.7960

Interactions across an image



One patch reads other patches

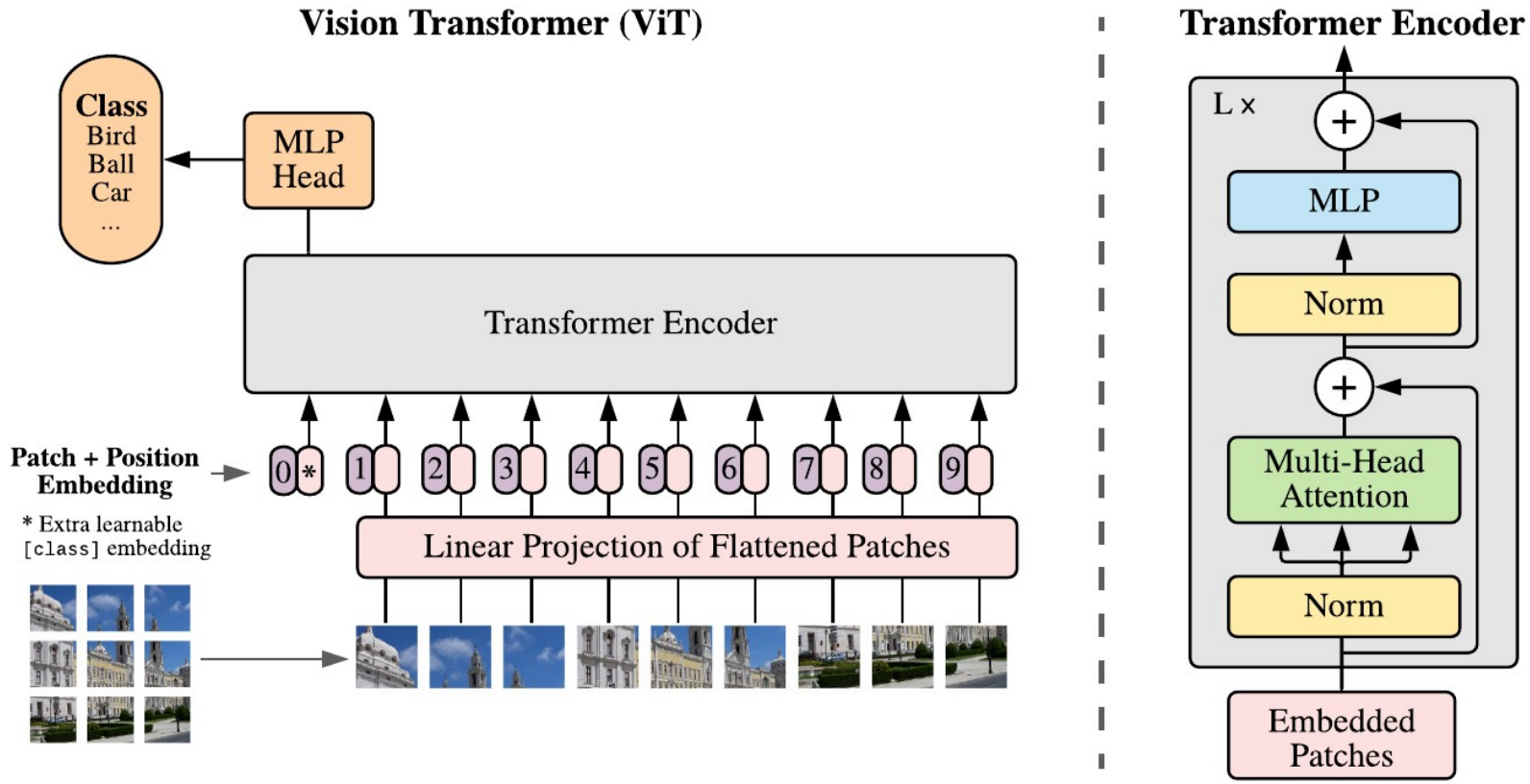
Query: the highlighted patch
Keys and values: all patch tokens

Weights depend on the image
and on the attention head.

The whole image is observed.
No causal mask is needed.

Source: Photo: Fredo Durand / MIT 6.7960

Complete Vision Transformer



[CLS] now gathers information from image patches.
Its final feature feeds the image classifier.

Source: An Image Is Worth 16×16 Words

Convolution and patch attention

Convolution

Local neighborhood
Shared kernel coefficients

Patch self-attention

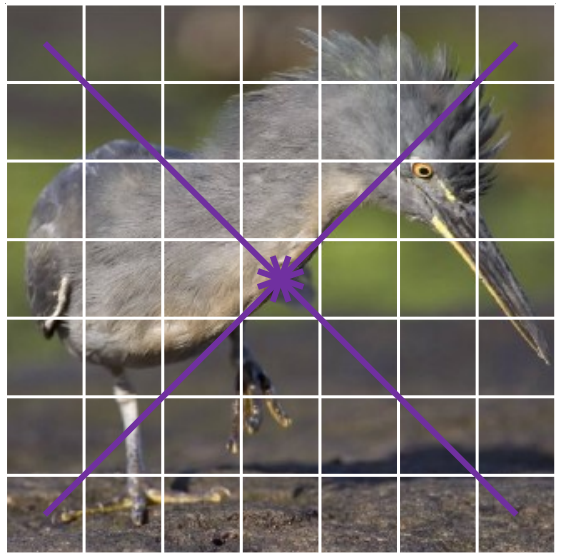


Image-dependent mixing
Global access between patches

Deeper CNN layers also build broad context.

Source: Photo: Fredo Durand / MIT 6.7960

Attention in a trained ViT

Input

Attention summary

Attention rollout

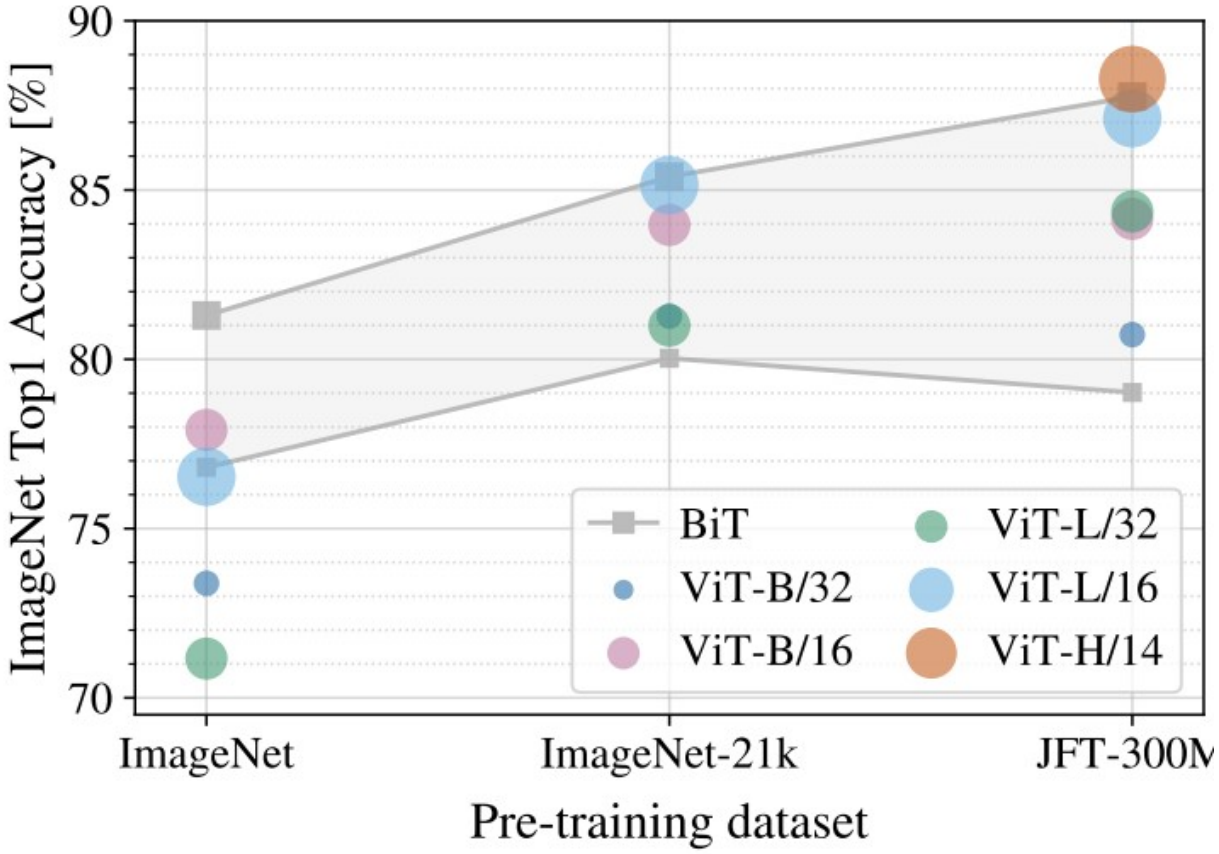
Combines heads and layers to summarize attention from the class output to patches.

Brighter regions receive more weight in this summary.

This is not a segmentation mask.

Source: An Image is Worth 16×16 Words

Pretraining data and recognition



Original ViT experiments

Colored marks: ViT variants
Gray band: ResNet models

ImageNet-21k: a larger ImageNet category collection

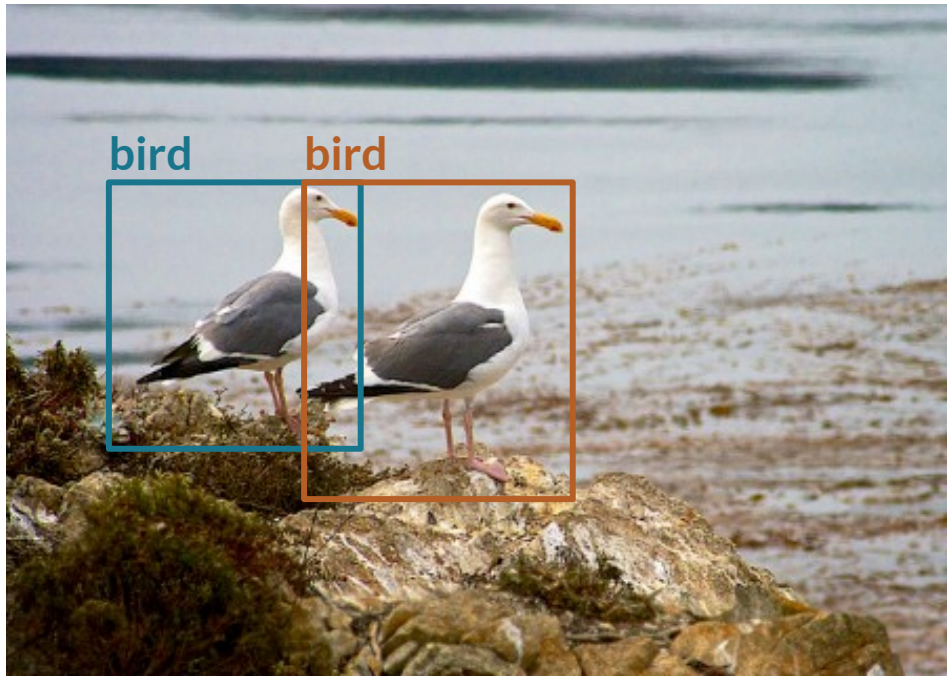
JFT-300M: hundreds of millions of training images

Larger pretraining collections made ViT more competitive.

Source: An Image is Worth 16×16 Words

DETR: object detection with Transformers

Predict a set of boxes directly, without proposals or anchors.



Fixed number of output slots

Each predicts a class distribution + box

No fixed order

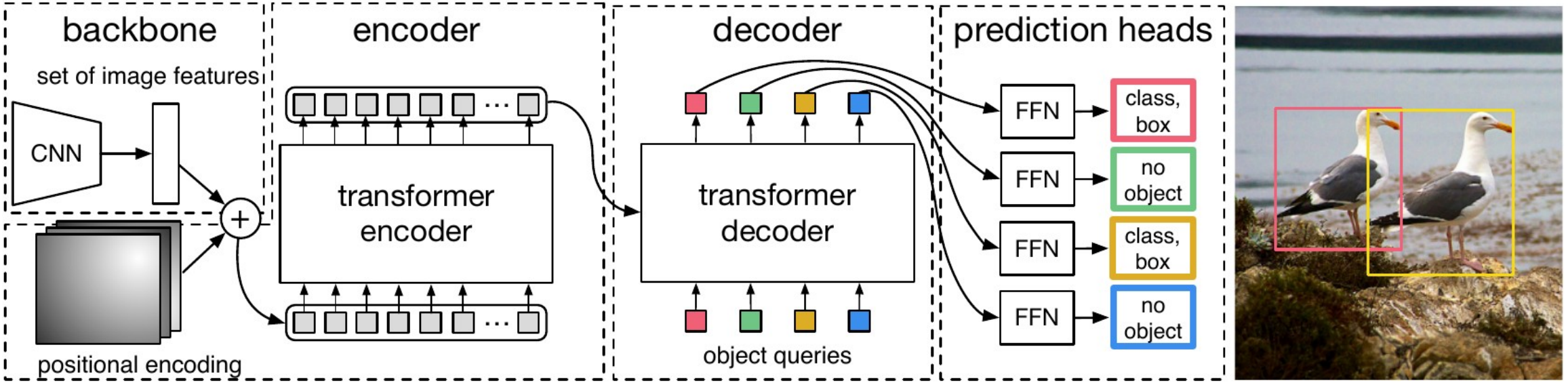
Left bird first or right bird first:
the target set is the same.

Unused slots receive the class “no object”.

Source: DETR

The DETR model

Original DETR: CNN image features + Transformer encoder–decoder



Object queries

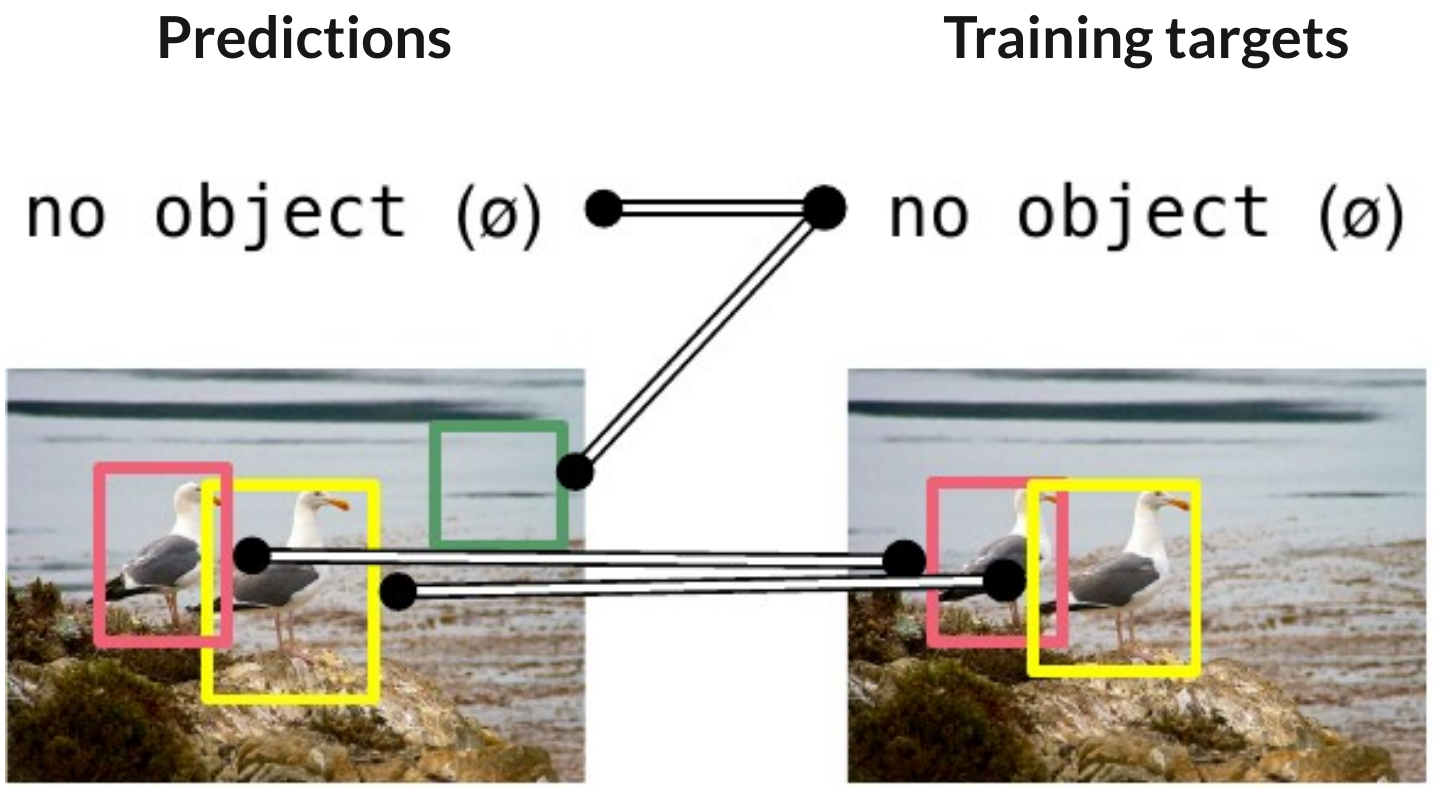
Learned vectors, one per output slot.

Parallel decoding

Slots interact without a causal mask.
Each slot reads encoded image features.

Source: DETR

Assigning training targets



Find a one-to-one match using class and box costs.
Unmatched predictions learn the “no object” class.

Source: DETR

Bounding-box loss

$$\mathcal{L}_{\text{box}} = \lambda_1 \|\hat{b} - b\|_1 + \lambda_G \left[1 - \text{GIoU}(\hat{b}, b)\right]$$

Coordinate error (L1)

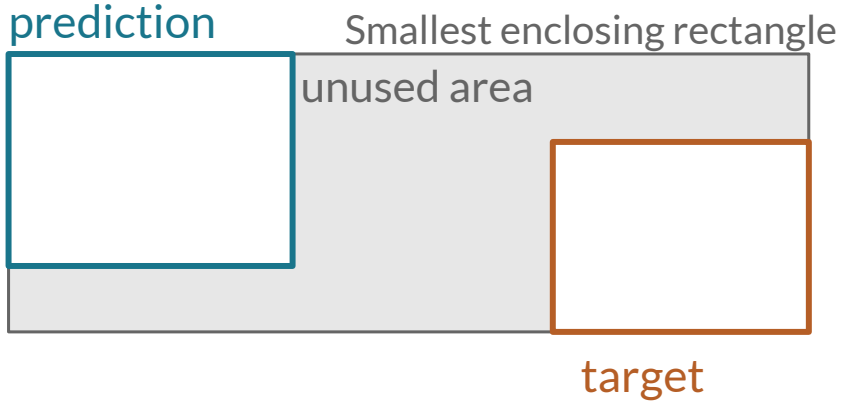
Add the absolute errors in the box center, width and height.

Coordinates are relative to image size.

$\hat{b}$: prediction b : target

The λ weights balance the two terms.

Generalized IoU (GIoU)



$$\text{GIoU} = \text{IoU} - \frac{\text{unused area}}{\text{enclosing area}}$$

Still distinguishes boxes when IoU = 0.

DETR training loss

Cross entropy: all output slots

$$\mathcal{L}_{\text{class}} = \sum_i w_i \, \text{CE}(y_i, p_i)$$

p_i: class probabilities for slot i
y_i: matched category or “no object”

w_i is smaller for “no object”.

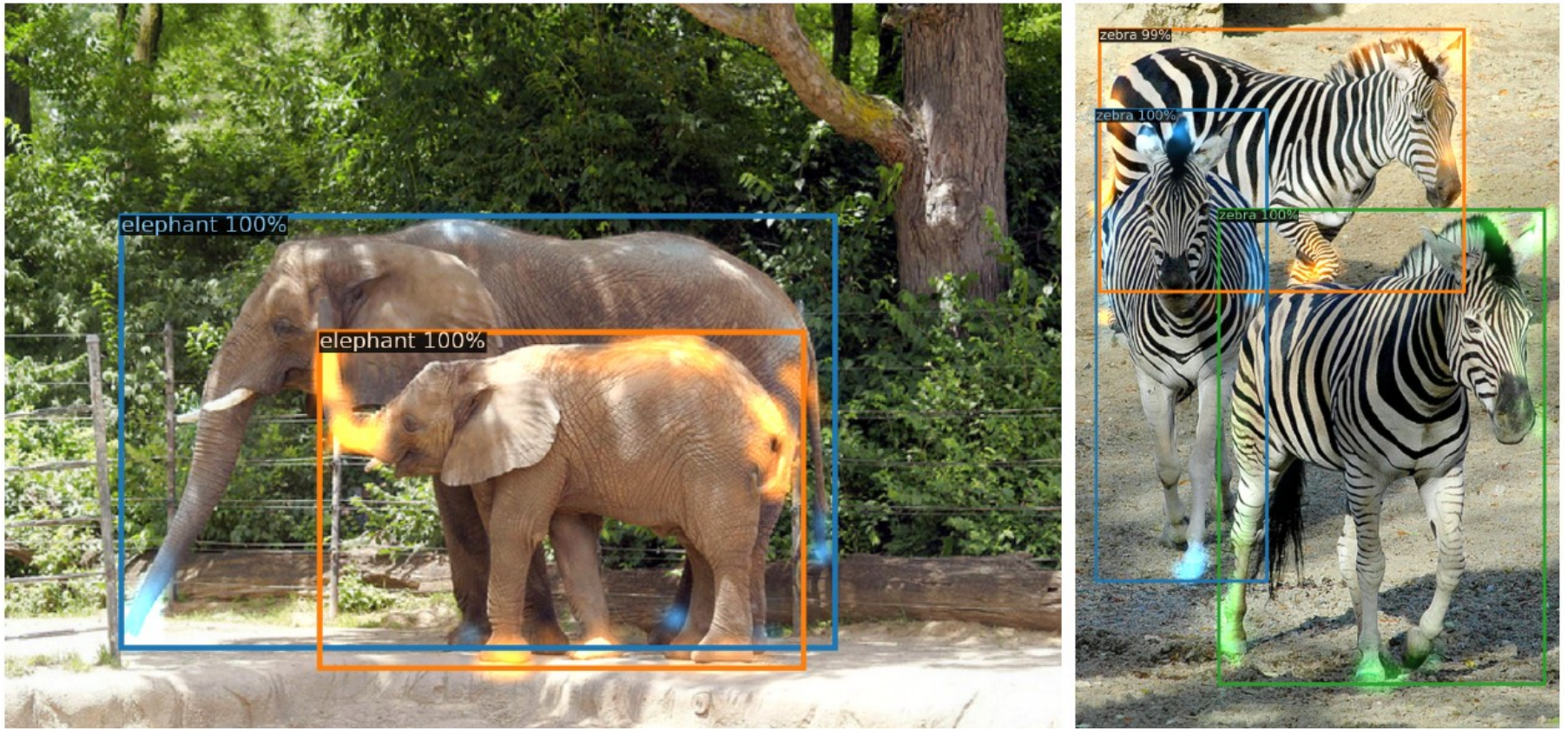
Box loss: matched objects only

$$\mathcal{L} = \mathcal{L}_{\text{class}} + \sum_{\text{matched pairs}} \mathcal{L}_{\text{box}}$$

Matching assigns targets. Backpropagation trains the detector.

DETR detections and attention

Colors show decoder attention for different object predictions.



Prediction needs neither target matching nor NMS.

Source: DETR

Image-and-text models

Question

Cost of the sliced-meat dish with an 18% tip?

Reported answer

CHF 43.07

The image supplies the details.
The question specifies what to answer.

Source: Gemma 3 Technical Report

Video generation

HunyuanVideo 1.5 (2025)

Generated video frames

A Transformer processes video features across space and time.

Transformer architecture, different training and generation procedure.

Source: HunyuanVideo 1.5 · arxiv.org/abs/2511.18870

Why Transformers became widely used

Content-dependent access

Read relevant positions directly, including distant ones.

Parallel computation during training

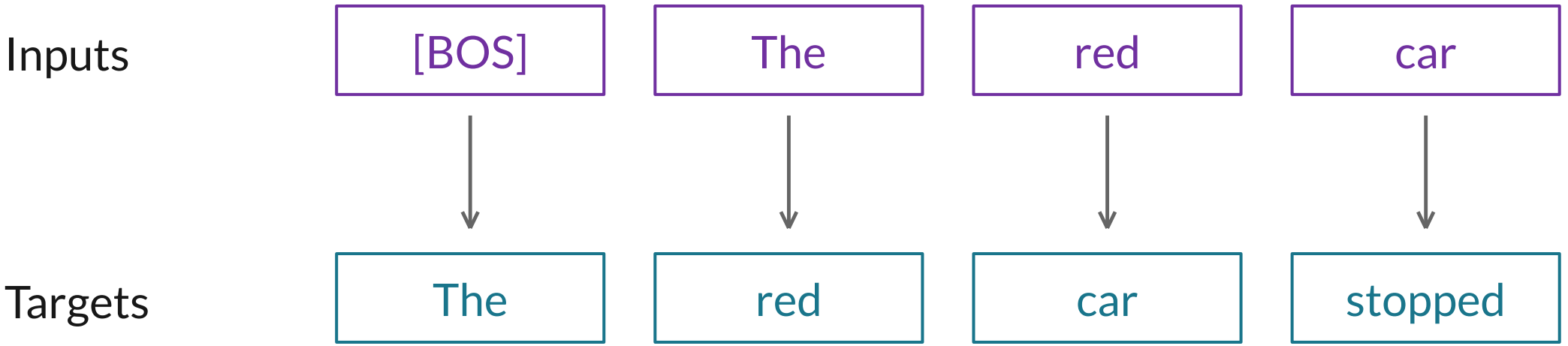
Process known positions together within each layer.

A common input format

Text tokens and image patches both become feature vectors.

Next topic: self-supervised learning

Next meeting: midterm review (L01–L12).



Self-supervised learning

Design prediction tasks that learn useful representations from unlabeled data.