



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

SoSe 2026

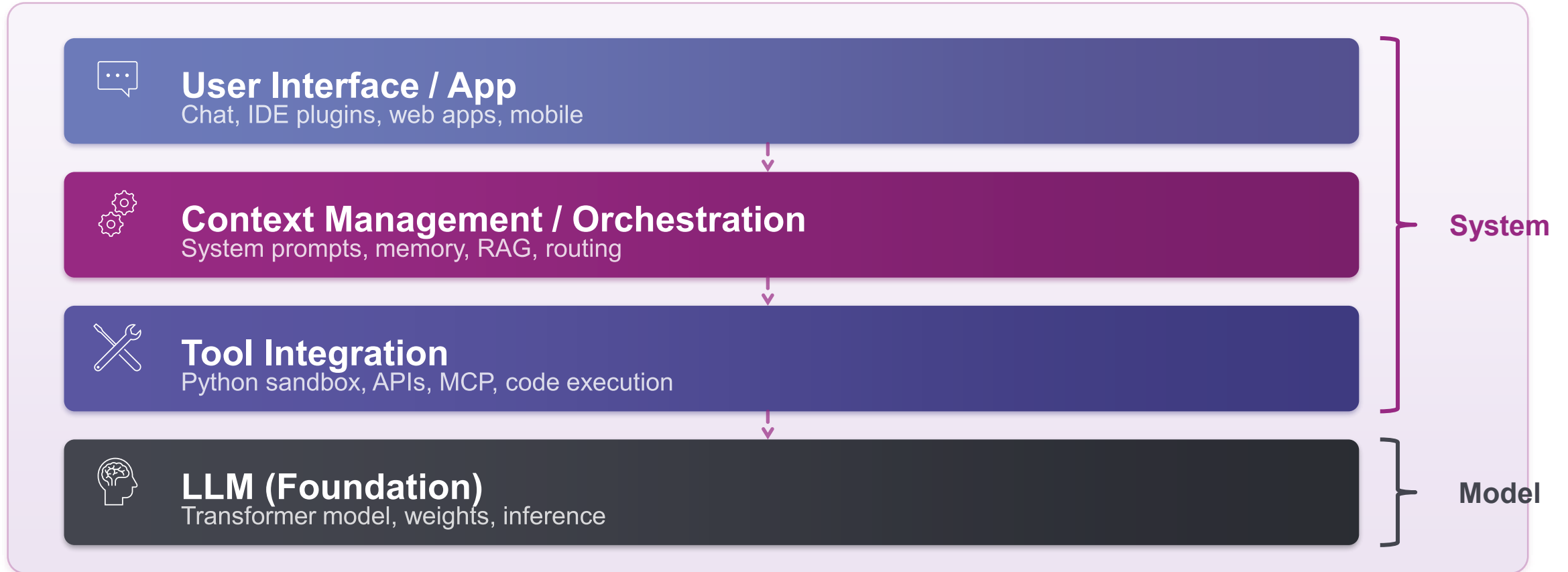
AI Supported Software Development

Week 5 – Inside the Transformer

Prof. Dr. Dominik Müller

AI Systems in a nutshell

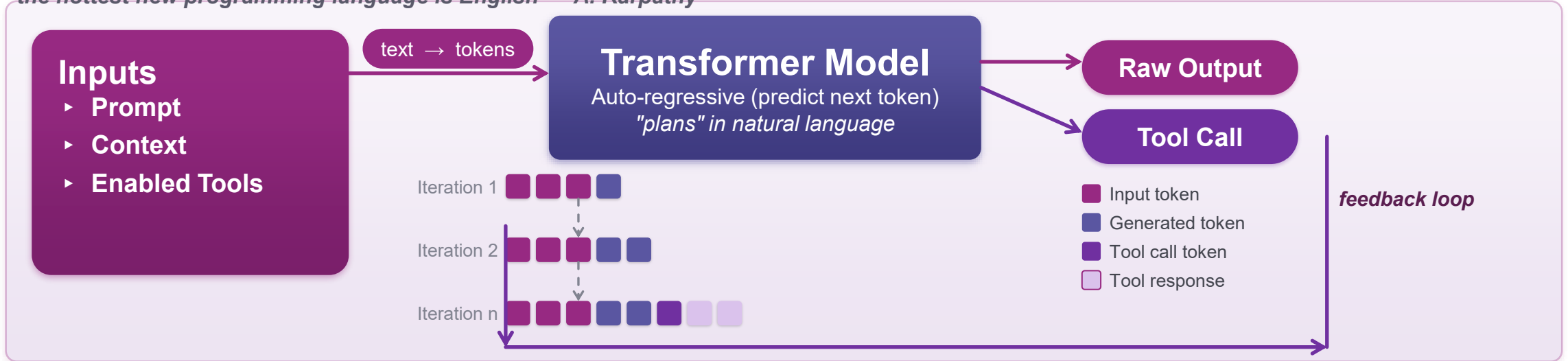
The model is only a (small) part



The LLM provides the **intelligence**, the system provides the **capability**

LLM in one picture

“the hottest new programming language is English” – A. Karpathy



Glossary

- **Prompt** (user provided input instructions), **Context** (extra text the model sees)
- **Token** (sub-word unit, model operates on tokens)
- **MCP** (generic interface / protocol to make any external tool available to an LLM)

Tools / Connectors

- External actions (HTTP, email, git, database, ...) **exposed to the model**
- On the fly **code generation & execution**

Four-Act Structure

Act I Experience Weeks 1–2

“The Wow and the Oops”

We USE AI tools before understanding them. We experience both the power and the failures firsthand, creating the cognitive tension that motivates everything that follows.

Act II Understand Weeks 3–7

“How Does This Actually Work?”

We learn foundational concepts: tokenization, embeddings, attention, context windows, inference, hallucination mechanisms. Prompting is taught as the applied side of these fundamentals.

Act III Master Weeks 8–10

“Wielding It Professionally”

We learn to evaluate AI output quality rigorously, integrate AI into real development workflows (IDE assistants, agentic pipelines, CI/CD), and measure productivity impact.

Act IV Govern Weeks 11–13

“The Bigger Picture”

We confront risks, security, organizational impact, and the economics of AI adoption. The course ends with you defending your own AI integration strategy.

AI AUTONOMY

Chat Assistant

Knowledge Partner

Workflow Integrator

Autonomous Agent

From Practice to Understanding

Act II — Understand the model behavior

In **Act I**, you used AI tools and got impressive results — and puzzling failures.

Act II: Understand the Model Behavior

- Four weeks to open the "black box" layer by layer
- No deep math — just enough to explain what you observed
- Every mysterious failure from Act I will have an answer

Today's layer: Embeddings and Entering the Transformer

LLMs are Next Token Predictors

1

The LLM forward path

START

Prompt = Raw Text

<BOS> dragon drinks

$V = 10$, Vocabulary

Tokenizer

$T = 3$, tokens

Token IDs

[0, 5, 6]

$d = 6$, embedding dim.

Embedding Lookup

Embedding table $E \in \mathbb{R}^{V \times d} = \mathbb{R}^{10 \times 6}$

	d_1	d_2	d_3	d_4	d_5	d_6
0: <BOS>	0.12	-0.34	0.56	-0.78	0.91	-0.23
1: wizard	-0.45	0.23	0.67	-0.12	0.34	-0.56
2: mixes	0.78	-0.91	0.11	0.45	-0.67	0.89
3: glowing	0.33	0.52	-0.41	0.19	-0.83	0.07
4: potion	-0.62	0.15	0.88	-0.27	0.46	-0.71
5: dragon	0.67	-0.11	0.43	-0.52	0.28	0.85
6: drinks	-0.39	0.74	-0.18	0.61	-0.45	0.33
7: fire	0.54	-0.68	0.22	0.37	-0.14	0.5
8: troll	-0.21	0.47	-0.83	0.66	0.09	-0.55
9: smashes	0.41	-0.36	0.73	-0.58	0.82	-0.17

$d = 6$ columns (embedding dimension)

TRANSFORMER INPUT

<BOS>	0.12	-0.34	0.56	-0.78	0.91	-0.23
dragon	0.67	-0.11	0.43	-0.52	0.28	0.85
drinks	-0.39	0.74	-0.18	0.61	-0.45	0.33

3 tokens $\times$ 6 dims

Transformer

+ Position — encode word order

Layer $\times$ N

Attention — tokens look at each other
FFN — process & refine

Inside:
W
trained weights

12 $\times$ (GPT-2) ... 96 $\times$ (GPT-3)

TRANSFORMER OUTPUT

<BOS>	0.08	0.15	-0.42	0.31	0.67	-0.19
dragon	0.55	-0.28	0.71	-0.63	0.44	0.92
drinks	0.83	0.47	-0.55	0.39	-0.12	0.68

still 3 $\times$ 6 — same shape, new values

LAST TOKEN: "drinks"

[0.83, 0.47, -0.55, 0.39, -0.12, 0.68]

multiply by
output weights
(10 $\times$ 6 matrix)

10 LOGITS (ONE PER VOCAB TOKEN)

[-1.2, 0.5, -0.3, 1.8, 4.2, 0.1, -0.8, 3.1, 0.9, 1.1]

softmax

NEXT TOKEN

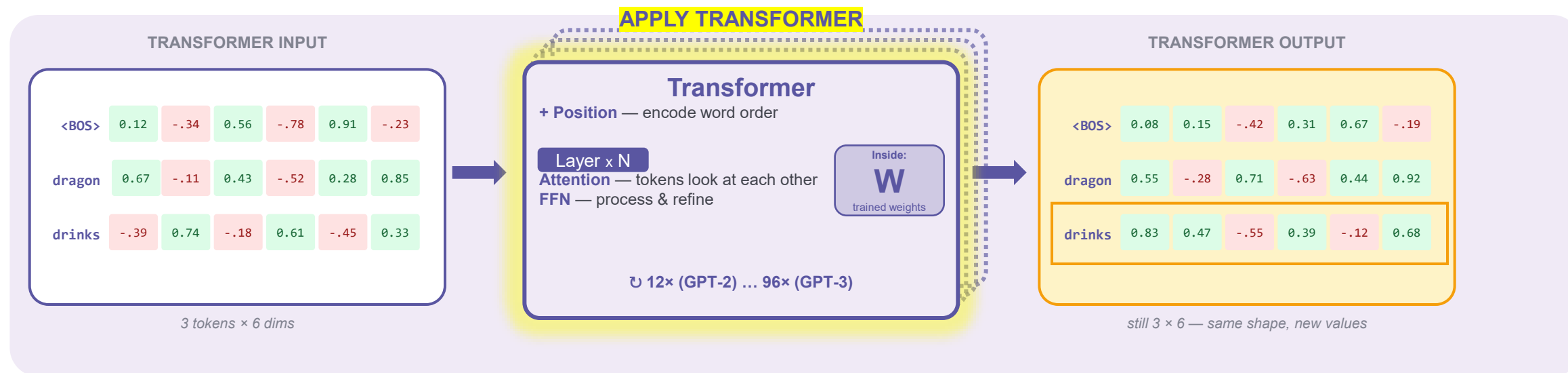
PROBABILITIES
potion = 62%

<BOS> dragon drinks potion

Inside the Transformer

2

Inside the Transformer

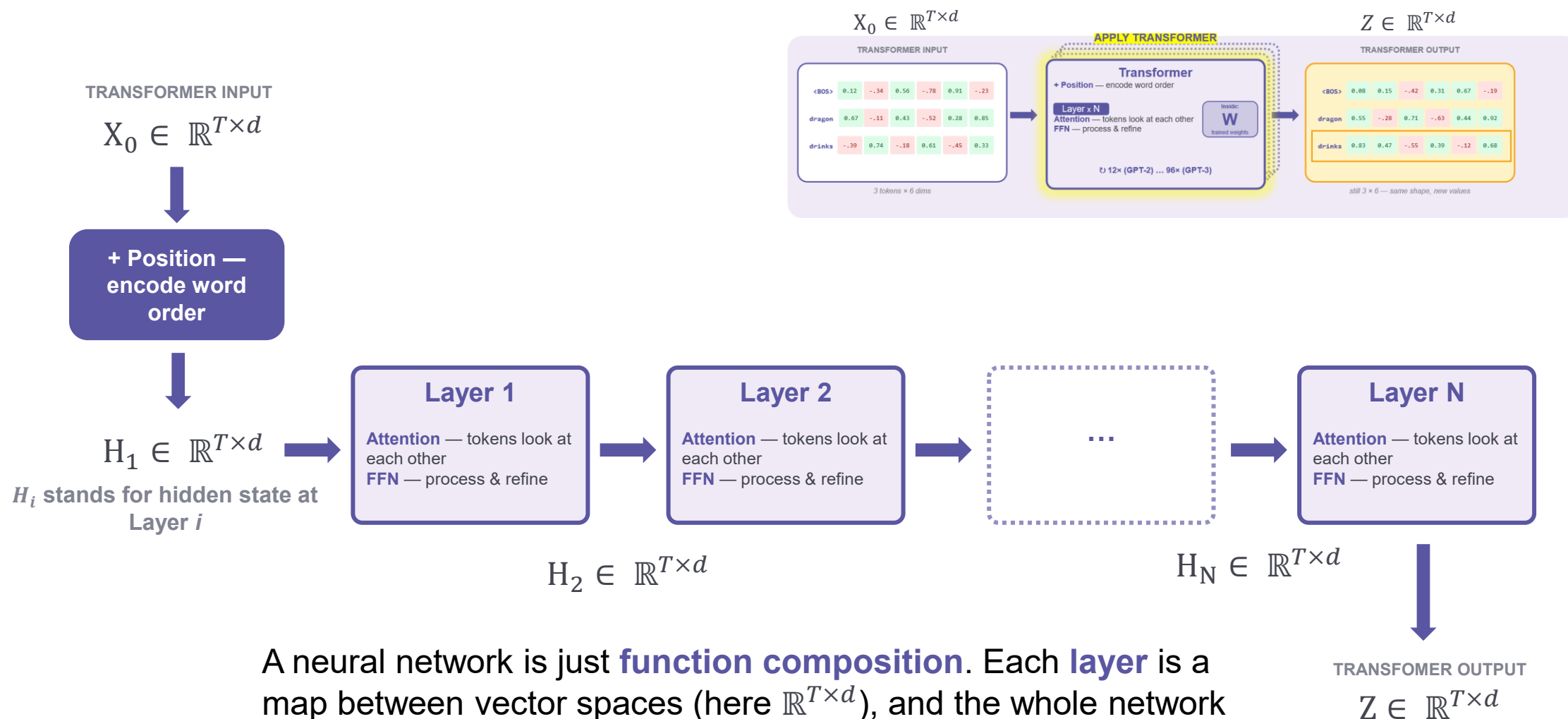


$$X_0 = \begin{bmatrix} x_1^T \\ x_2^T \\ x_3^T \end{bmatrix} \in \mathbb{R}^{T \times d} = \mathbb{R}^{3 \times 6}$$

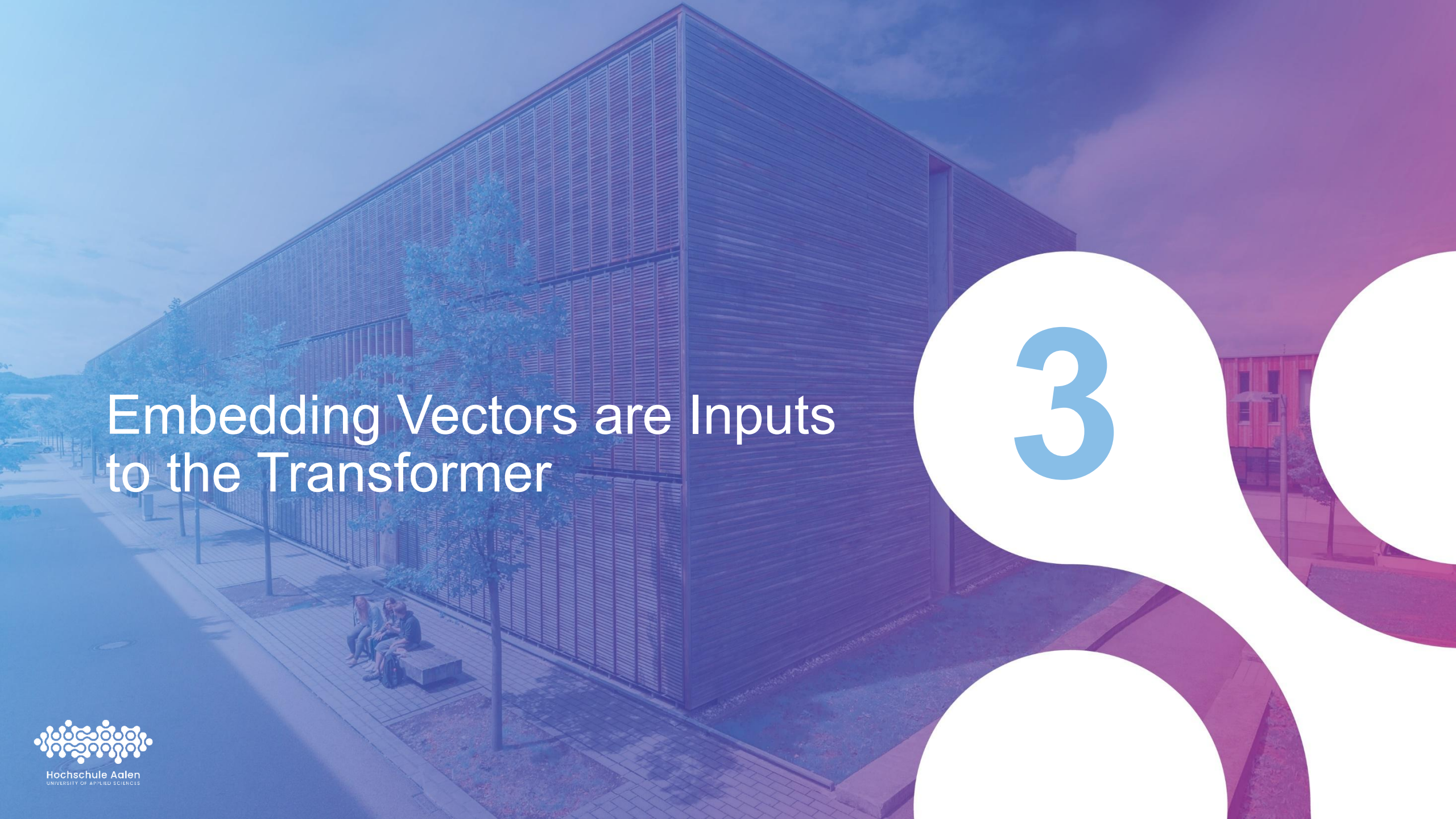
Introduce **matrix notation** — a compact, precise description of the layers that brings linear algebra to bear and lets us snap back to the visuals whenever we need to.

$$Z = \begin{bmatrix} z_1^T \\ z_2^T \\ z_3^T \end{bmatrix} \in \mathbb{R}^{T \times d} = \mathbb{R}^{3 \times 6}$$

Inside the Transformer



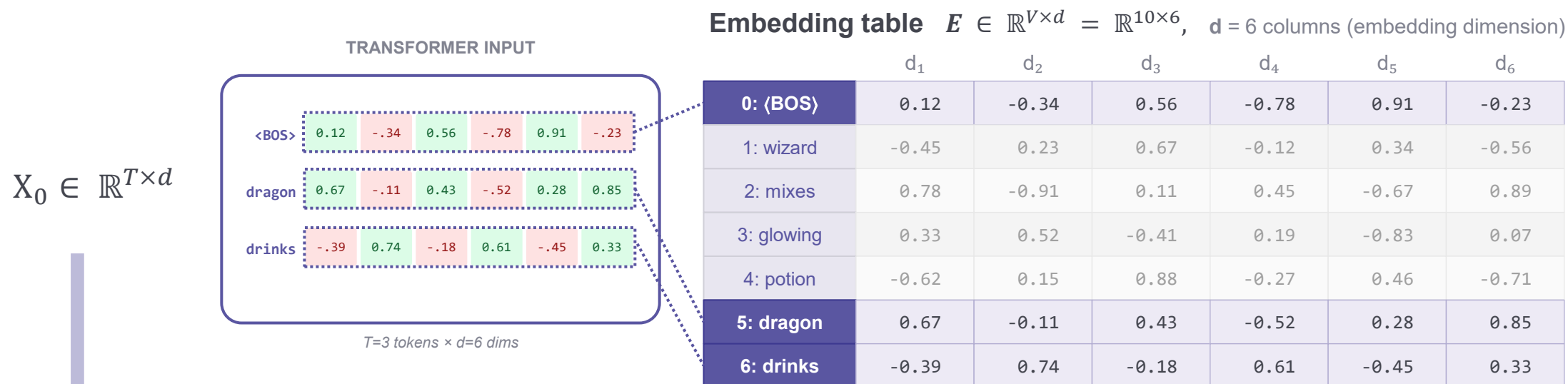
A neural network is just **function composition**. Each **layer** is a map between vector spaces (here $\mathbb{R}^{T \times d}$), and the whole network is $f = f_N \circ \dots \circ f_1$. Stack enough layers and this composed function becomes enormous — that's the model.



Embedding Vectors are Inputs to the Transformer

3

Embeddings



The Embedding Dimension d — a Hyperparameter

- d defines the size of the vector used to represent each token.
- Each token is mapped to a vector in $\mathbb{R}^d$.
- Information is encoded in a **distributed** way across these d dimensions.
- Individual dimensions are usually **not directly human-interpretable**.
- Larger d allows richer representations but increases parameters and computation.

Embeddings

$$X_0 \in \mathbb{R}^{T \times d}$$

+ Position —
encode word
order

TRANSFORMER INPUT

<BOS>	0.12	-.34	0.56	-.78	0.91	-.23
dragon	0.67	-.11	0.43	-.52	0.28	0.85
drinks	-.39	0.74	-.18	0.61	-.45	0.33

$T=3$ tokens $\times$ $d=6$ dims

$d = 6$ columns (embedding dimension)

TRANSFORMER OUTPUT

<BOS>	0.08	0.15	-.42	0.31	0.67	-.19
dragon	0.55	-.28	0.71	-.63	0.44	0.92
drinks	0.83	0.47	-.55	0.39	-.12	0.68

still $T=3 \times d=6$ — same shape, new values

$$z_T \in \mathbb{R}^d$$

The transformer keeps the shape $T \times d$ but updates the values using **context**.

At the last position T , the output vector $z_T \in \mathbb{R}^d$ is a **contextual representation** of the sequence so far.

This vector is then used to **predict** the **next token**.

The Embedding Dimension d — a Hyperparameter

- d defines the size of the vector used to represent each token.
- Each token is mapped to a vector in $\mathbb{R}^d$.
- Information is encoded in a **distributed** way across these d dimensions.
- Individual dimensions are usually **not directly human-interpretable**.
- Larger d allows richer representations but increases parameters and computation.

Embeddings

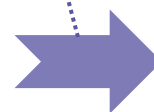
$$X_0 \in \mathbb{R}^{T \times d}$$

TRANSFORMER INPUT

<BOS>	0.12	-.34	0.56	-.78	0.91	-.23
dragon	0.67	-.11	0.43	-.52	0.28	0.85
drinks	-.39	0.74	-.18	0.61	-.45	0.33

$T=3$ tokens $\times$ $d=6$ dims

$d = 6$ columns (embedding dimension)



TRANSFORMER OUTPUT

<BOS>	0.08	0.15	-.42	0.31	0.67	-.19
dragon	0.55	-.28	0.71	-.63	0.44	0.92
drinks	0.83	0.47	-.55	0.39	-.12	0.68

still $T=3 \times d=6$ — same shape, new values

$$z_T \in \mathbb{R}^d$$

The transformer keeps the shape $T \times d$ but updates the values using **context**.

At the last position T , the output vector $z_T \in \mathbb{R}^d$ is a **contextual representation** of the sequence so far.

This vector is then used to **predict** the **next token**.

+ Position —
encode word
order

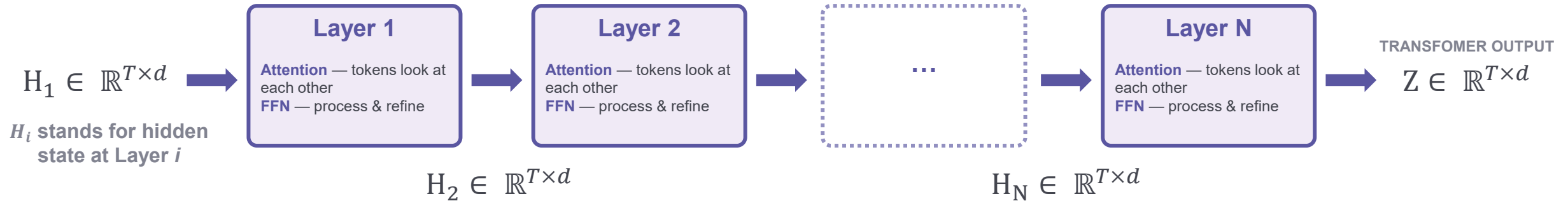
	Toy	GPT-2	GPT-3	DeepSeek-V3	Llama 4
V = Vocabulary	10	50,257	50,257	129,280	202,048
d = Embedding dim	6	768	12,288	7,168	5,120
V x d = Embedding params	60	38M	618M	926M	1,034M
Ratio = d / V	0,6	0,015	0,244	0,055	0,025

Because next-token prediction means **choosing one token out of V candidates**, the model needs a **d -dimensional representation rich enough to encode** the context and separate the correct token from all others.

Some Linear Algebra

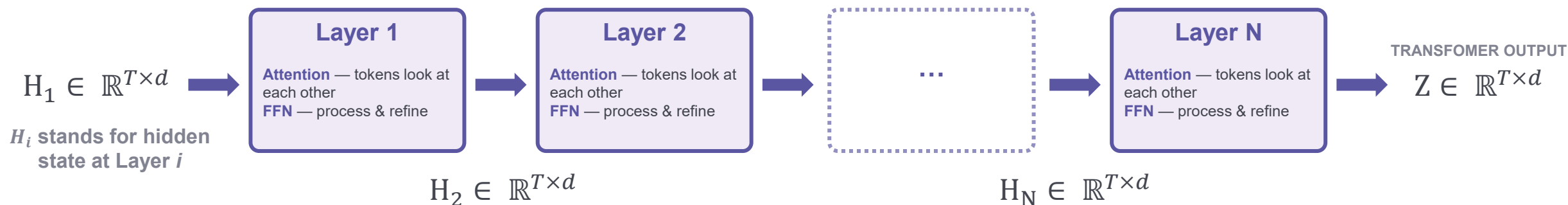
4

Inside the Layers we will do advanced Linear Algebra



- Columns are transformed by left multiplication / Rows are transformed by right multiplication.
- Ax is a linear combination of columns of A / $x^T A$ is a linear combination of rows of A .
- $XWW^T X^T$ stores all query-key dot products.
- Cosine similarity compares direction; dot product compares direction and magnitude.
- Self-attention is linear algebra plus softmax.

Inside the Layers we will keep track of the Tokens



$$H_1 = \begin{bmatrix} h_1^T \\ \vdots \\ h_{20}^T \\ \vdots \\ h_T^T \end{bmatrix}$$

TOKEN		d_1	d_2	d_3	$d = d_{12,288}$	
2234: the	h_1^T	-0.39	0.74	-0.18	...	-0.45
...	...	-0.62	0.15	0.38	...	0.46
4225: cat	h_{20}^T	0.67	-0.11	0.43	...	0.28
5561: and	h_{21}^T	-0.39	0.74	-0.18	...	-0.45
...	...	-0.62	0.15	0.87	...	0.46
31622: hat	h_T^T	0.67	-0.11	0.43	...	0.28

$\in \mathbb{R}^{T \times d}$

Keep track of which token each row belongs to —
across every matrix — so later we can tell which tokens
are talking to which.

Inside the Layers we will keep track of the Tokens

$$H_1 = \begin{bmatrix} h_1^T \\ \vdots \\ h_{20}^T \\ \vdots \\ h_T^T \end{bmatrix}$$

TOKEN		d_1	d_2	d_3	$d = d_{12,288}$	
2234: king	h_1^T	-0.39	0.74	-0.18	...	-0.45
...	...	-0.62	0.15	0.38	...	0.46
4225: man	h_{20}^T	0.67	-0.11	0.43	...	0.28
5561: woman	h_{21}^T	-0.39	0.74	-0.18	...	-0.45
...	...	-0.62	0.15	0.87	...	0.46
31622: queen	h_T^T	0.67	-0.11	0.43	...	0.28

$\in \mathbb{R}^{T \times d}$

Keep track of which token each row belongs to — across every matrix — so later we can tell which tokens are talking to which.

Every upcoming operation will preserve the row order: **row i** always corresponds to **token i** . Attention mixes information across rows, but the mapping stays — which is how we later trace who attended to whom.



Vectors You Can Reason About

5

Meaning Lives in Direction

The canonical embedding-space result:

king – man + woman $\approx$ queen

What it means

- Semantic relationships become **directions** in embedding space.
- Gender, tense, plurality, analogy — all encoded as **offsets**.
- Nobody designed this — it emerges from training on "which words appear near which"

Vocabulary $V = 50,257$, Embedding Dimension $d = 12,288$

	d_1	d_2	d_3	...	$d_{12,288}$
0: space	0.12	-0.34	0.56	...	0.91
...	-0.45	0.23	0.67	...	0.34
2234: king	-0.39	0.74	-0.18	...	-0.45
2235: function	0.33	0.52	-0.41	...	-0.83
...	-0.62	0.15	0.38	...	0.46
4225: man	0.67	-0.11	0.43	...	0.28
4226: woman	-0.39	0.74	-0.18	...	-0.45
...	-0.62	0.15	0.87	...	0.46
11255: python	0.33	0.52	-0.41	...	-0.83
31622: queen	0.67	-0.11	0.43	...	0.28
...	0.12	0.35	-0.78	...	0.36

Notebook: Open [§2 in Colab](#) — inspect nearest neighbors

Meaning Lives in Direction

Model contains the loaded embeddings.

`most_similar` with `topn=5` returns the embeddings with the largest cosine similarity to the given `word`.

```
if HAS_EMBEDDINGS:
    for word in ['python', 'king', 'paris', 'function']:
        neighbours = model.most_similar(word, topn=5)
        print(f'{word}:')
        for w, s in neighbours:
            print(f'    {w:15s} (sim={s:.3f})')
        print()
```

```
python:
    reticulated    (sim=0.692)
    spamalot      (sim=0.664)
    php           (sim=0.641)
    owl         (sim=0.630)
    mouse         (sim=0.628)
```

```
king:
    prince        (sim=0.824)
    queen         (sim=0.784)
    ii            (sim=0.775)
    emperor       (sim=0.774)
    son           (sim=0.767)
```

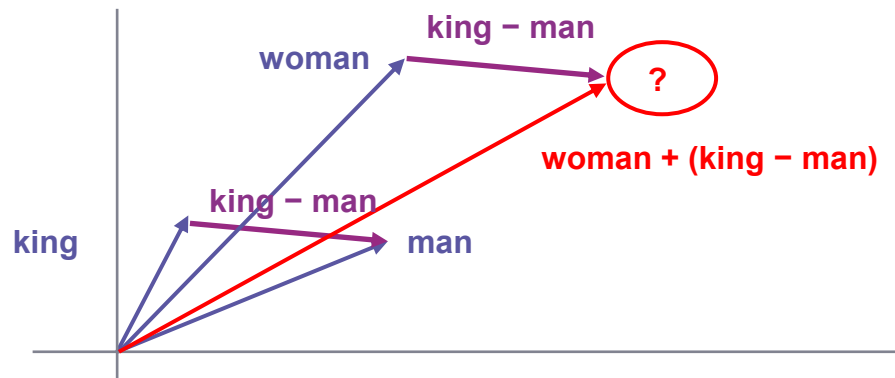
```
paris:
    prohertrib    (sim=0.861)
    france        (sim=0.803)
    brussels      (sim=0.780)
    amsterdam     (sim=0.777)
    french        (sim=0.774)
```

```
function:
    functions     (sim=0.933)
    i.e.          (sim=0.846)
    functional    (sim=0.827)
    hence         (sim=0.823)
    parameters    (sim=0.822)
```

Meaning Lives in Direction

king - man + woman $\approx$ queen

If we add the embedding vector of woman to the delta vector of the embedding vectors of king and man, we find that the closest embedding vector in this vocabulary (w.r.t cosine similarity) is queen.



```
▶ if HAS_EMBEDDINGS:
    for a, b, c in [
        ('king', 'man', 'woman'),
        ('paris', 'france', 'germany'),
        ('walked', 'walking', 'running'),
        ('bigger', 'big', 'small'),
    ]:
        # compute model[a] - model[b] + model[c] and find the closest word
        results = model.most_similar(positive=[a, c], negative=[b], topn=3)
        print(f'{a} - {b} + {c} ≈')
        for w, s in results:
            print(f'    {w:15s} (sim={s:.3f})')
        print()
```

```
... king - man + woman ≈
    queen      (sim=0.852)
    throne     (sim=0.766)
    prince     (sim=0.759)
```

```
paris - france + germany ≈
    berlin     (sim=0.920)
    frankfurt  (sim=0.820)
    vienna     (sim=0.818)
```

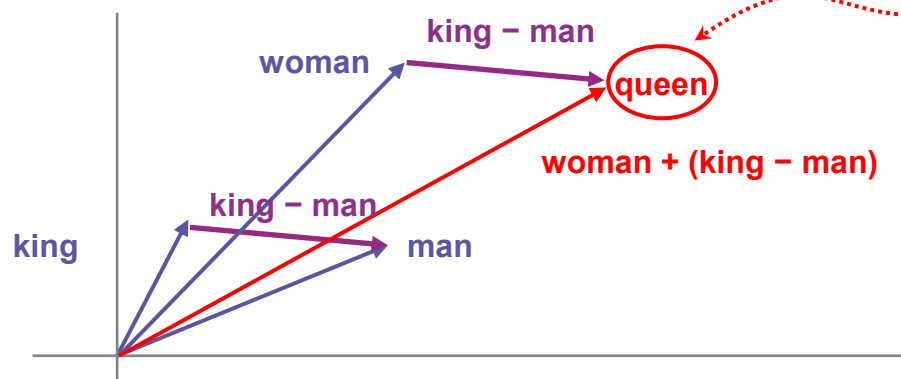
```
walked - walking + running ≈
    run        (sim=0.795)
    ran        (sim=0.795)
    picked     (sim=0.762)
```

```
bigger - big + small ≈
    larger     (sim=0.924)
    smaller    (sim=0.885)
    large      (sim=0.884)
```

Meaning Lives in Direction

$\text{king} - \text{man} + \text{woman} \approx \text{queen}$

If we add the embedding vector of woman to the delta vector of the embedding vectors of king and man, we find that the closest embedding vector in this vocabulary (w.r.t cosine similarity) is queen.



```
if HAS_EMBEDDINGS:
    for a, b, c in [
        ('king', 'man', 'woman'),
        ('paris', 'france', 'germany'),
        ('walked', 'walking', 'running'),
        ('bigger', 'big', 'small'),
    ]:
        # compute model[a] - model[b] + model[c] and find the closest word
        results = model.most_similar(positive=[a, c], negative=[b], topn=3)
        print(f'{a} - {b} + {c} ≈')
        for w, s in results:
            print(f'    {w:15s} (sim={s:.3f})')
        print()
```

```
... king - man + woman ≈
    queen      (sim=0.852)
    throne     (sim=0.766)
    prince     (sim=0.759)
```

```
paris - france + germany ≈
    berlin     (sim=0.920)
    frankfurt  (sim=0.820)
    vienna     (sim=0.818)
```

```
walked - walking + running ≈
    run        (sim=0.795)
    ran        (sim=0.795)
    picked     (sim=0.762)
```

```
bigger - big + small ≈
    larger     (sim=0.924)
    smaller    (sim=0.885)
    large      (sim=0.884)
```

Latent Space — t-SNE and UMAP

Embedding vectors live in ~768 to ~16,000 dimensions. **Unreadable.**

Two standard tools to project them to 2D / 3D for humans:

- **t-SNE** — preserves local structure. Good for "what clusters together?"
- **UMAP** — preserves both local and global structure. Faster. Current default.

You will run both in the lab notebook:

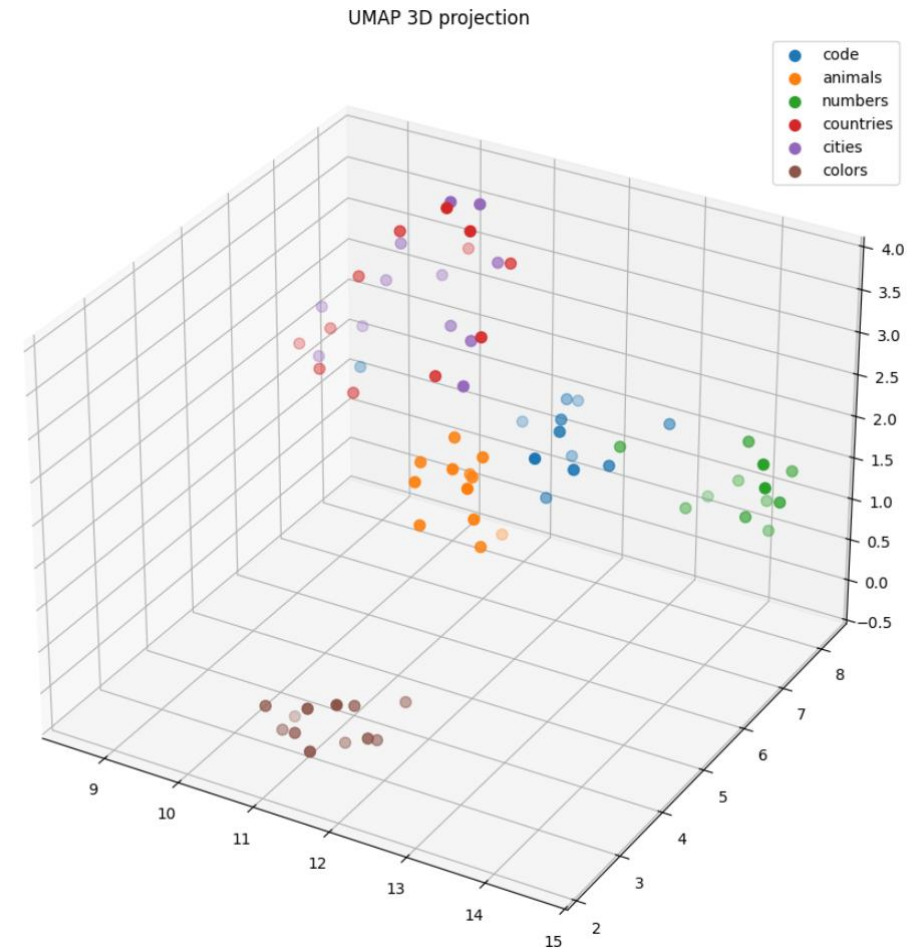
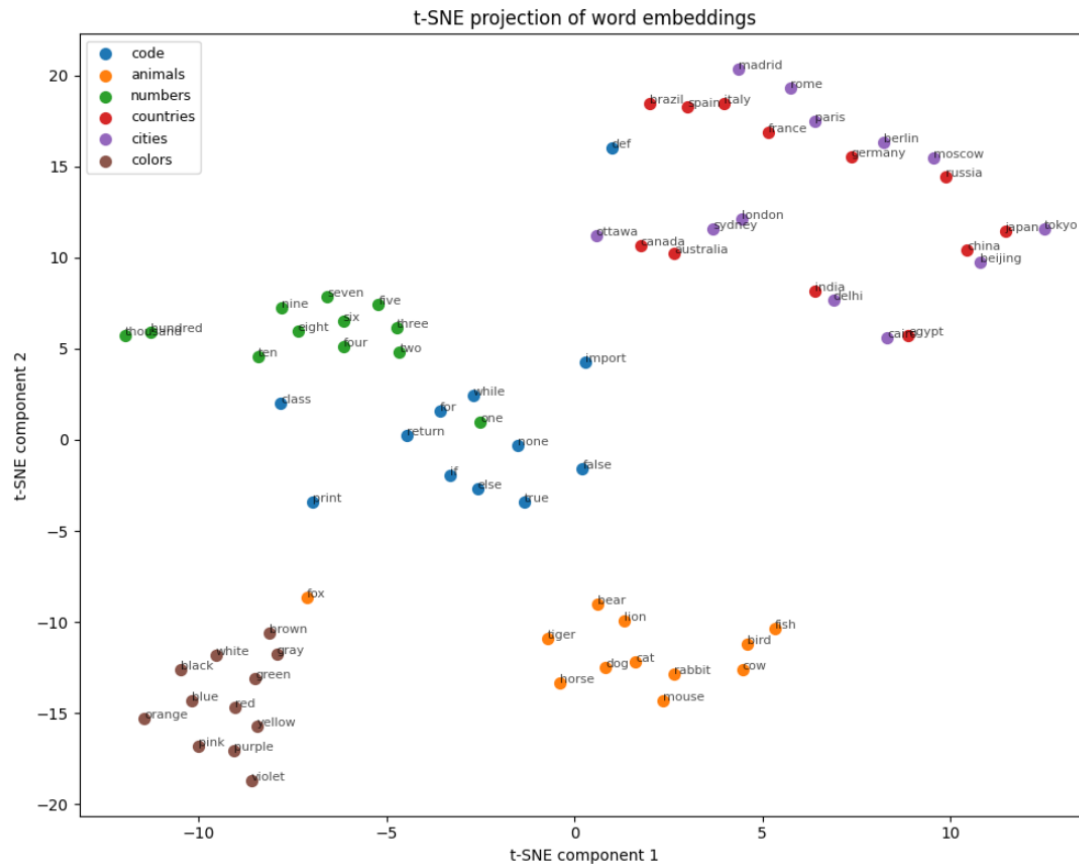
embed {king, queen, man, woman, Paris, France, Berlin, Germany, cat, dog, tree, forest, ...} ,

project to 2D, look at what ends up next to what.

The point: these are lossy pictures. Nearness in the plot suggests nearness in the real space — but the real space always has the final say.

Notebook: Open [§2 in Colab](#) — latent-space visualisation

Latent Space — t-SNE and UMAP





Inside the Layer

6

Inside the Transformer

TRANSFORMER INPUT

$$X_0 \in \mathbb{R}^{T \times d}$$

+ Position —
encode word
order

$$H_1 \in \mathbb{R}^{T \times d}$$

We skip **positional encoding** for now and clarify this step in detail later

Layer 1

Attention — tokens look at each other
FFN — process & refine

Layer 2

Attention — tokens look at each other
FFN — process & refine

...

Layer N

Attention — tokens look at each other
FFN — process & refine

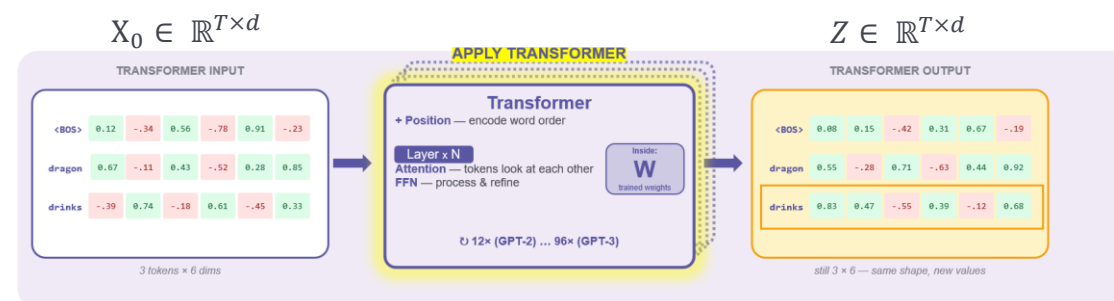
$$H_2 \in \mathbb{R}^{T \times d}$$

H_i stands for hidden state at Layer i

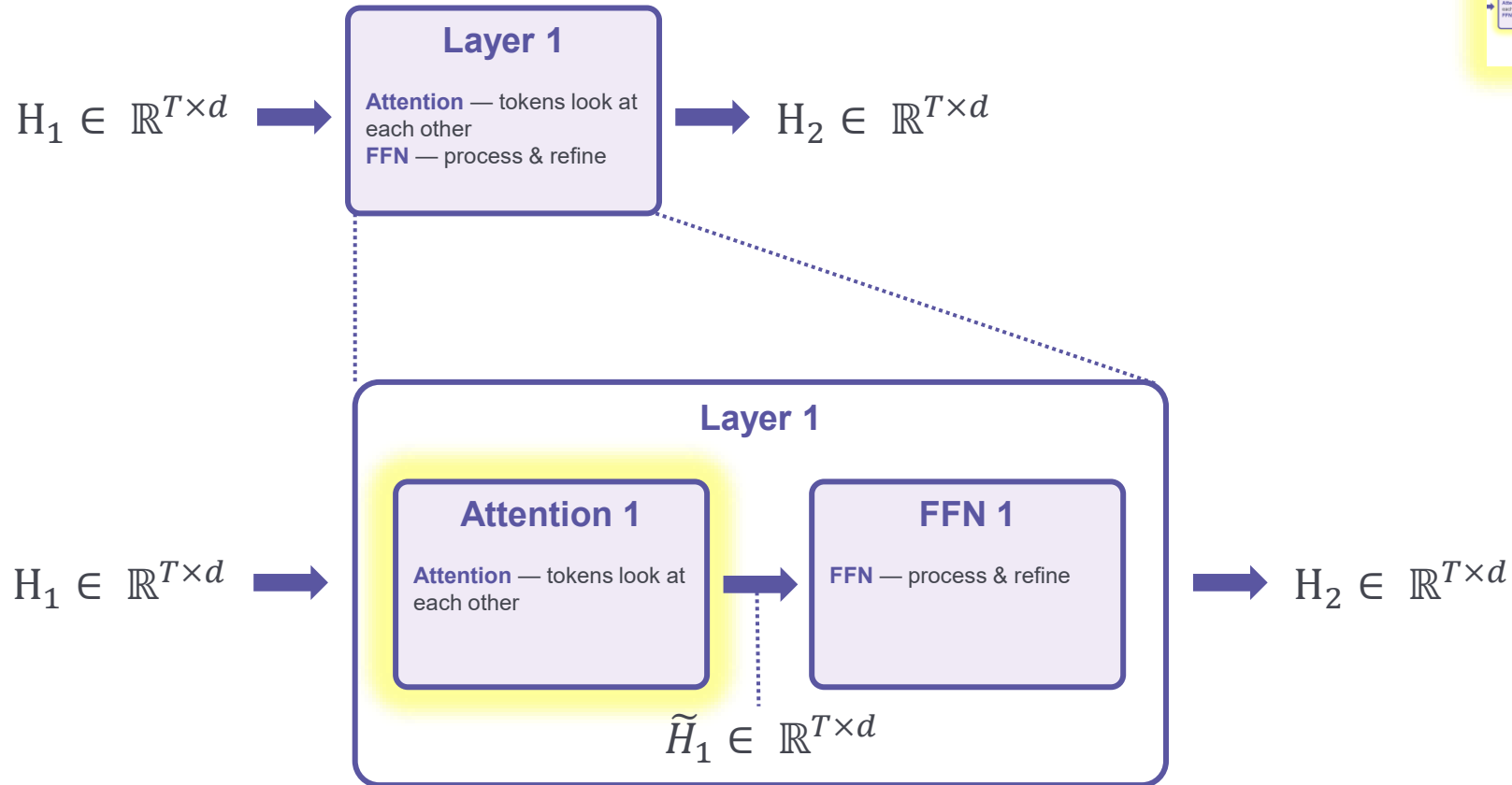
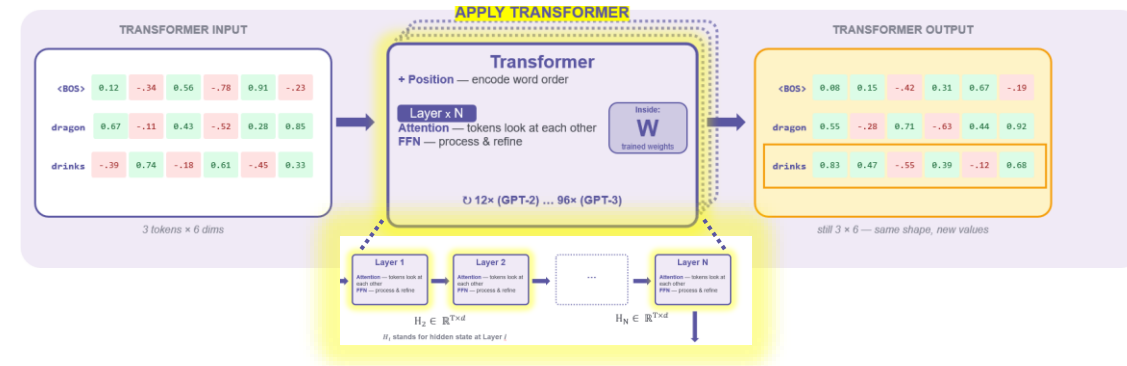
$$H_N \in \mathbb{R}^{T \times d}$$

TRANSFORMER OUTPUT

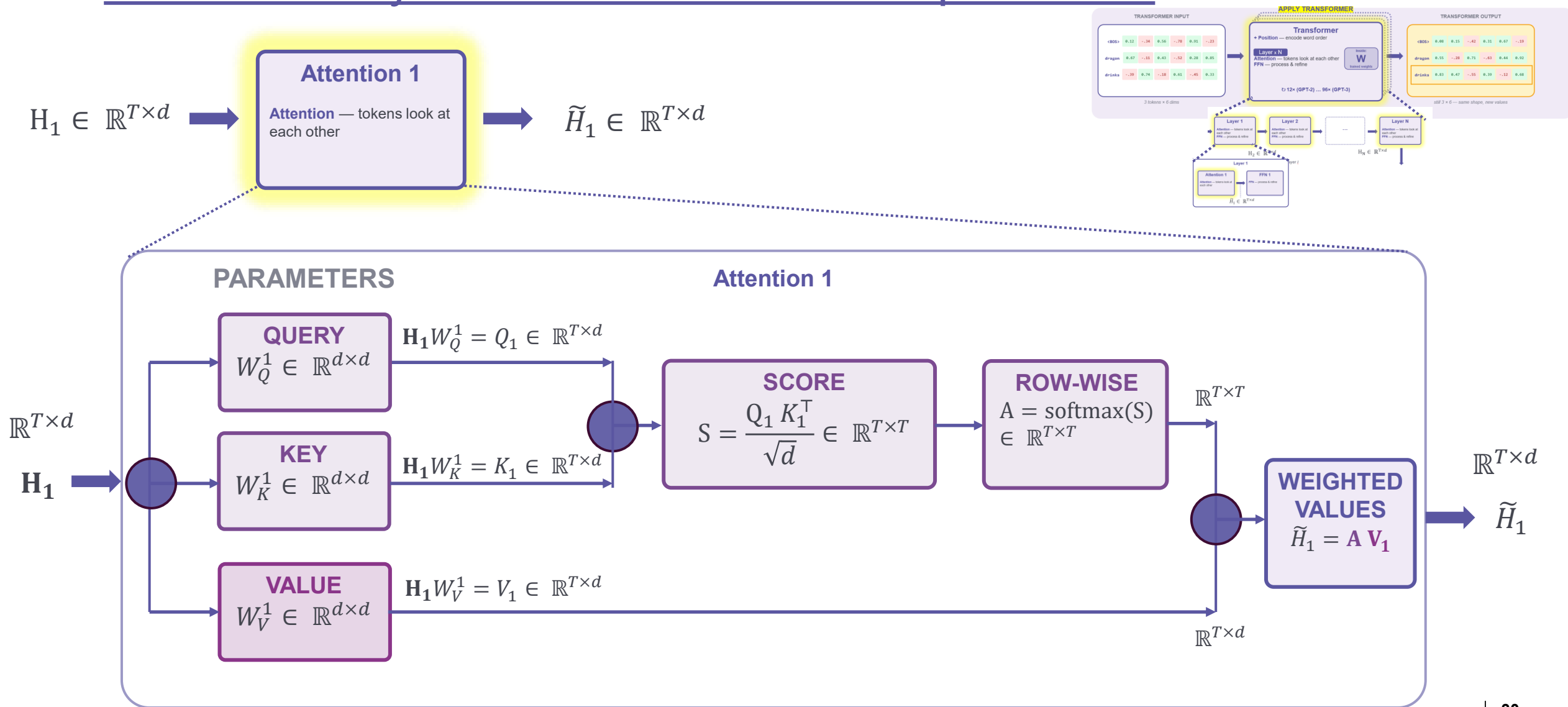
$$Z \in \mathbb{R}^{T \times d}$$



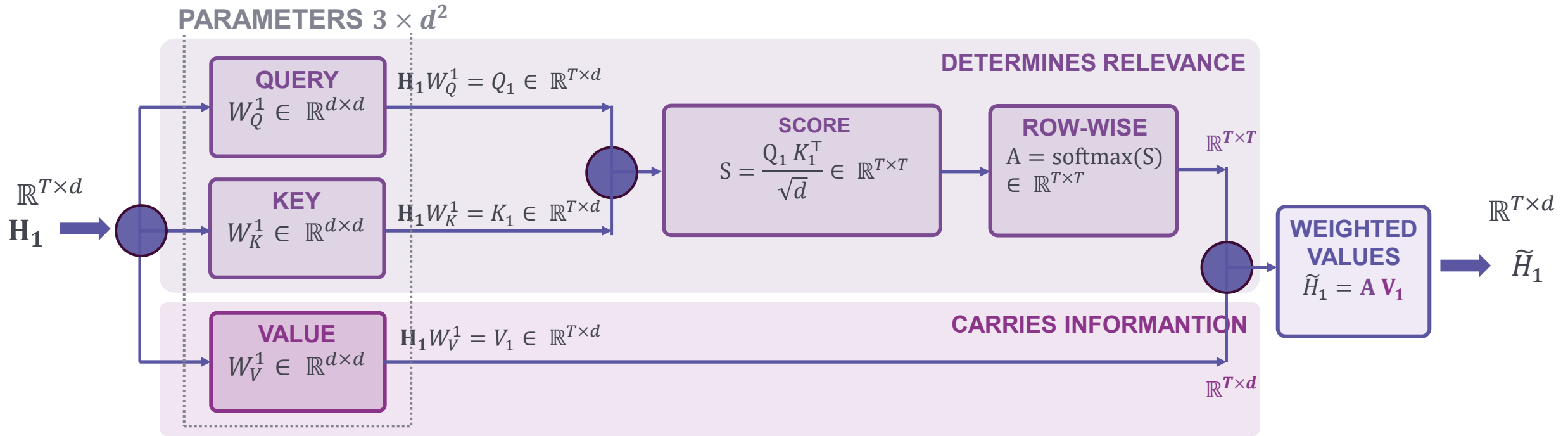
Inside the Layers



Inside the Layers – Attention in its simplest form



Inside the Layers – Attention in its simplest form



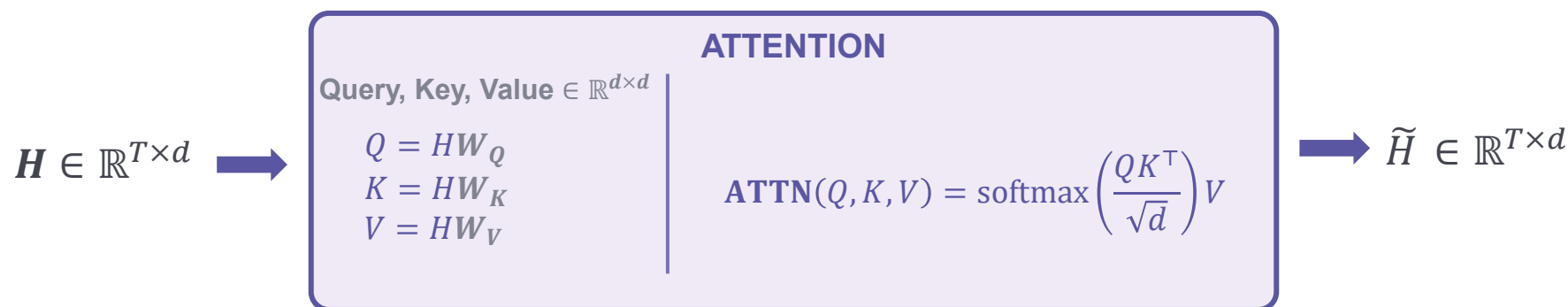
$$Q_1 = H_1 W_Q^1, \quad K_1 = H_1 W_K^1, \quad V_1 = H_1 W_V^1 \in \mathbb{R}^{T \times d}$$

$$\tilde{H}_1 = \underbrace{\text{softmax}\left(\frac{Q_1 K_1^T}{\sqrt{d}}\right)}_{\mathbb{R}^{T \times T}} \underbrace{V_1}_{\mathbb{R}^{T \times d}} \in \mathbb{R}^{T \times d}$$

Understanding Attention

7

Inside the Layers – Attention in its simplest form

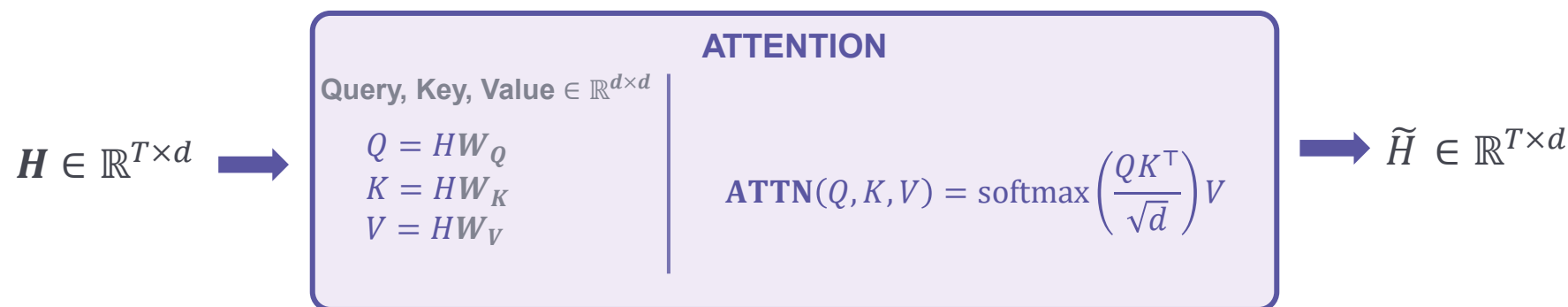


$$Q = HW_Q, \quad K = HW_K, \quad V = HW_V \in \mathbb{R}^{d \times d}$$

$$\tilde{H} = \underbrace{\text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)}_{\mathbb{R}^{T \times T}} \underbrace{V}_{\mathbb{R}^{T \times d}} \in \mathbb{R}^{T \times d}$$

Linear algebra collapses all the per-token computations into **a single formula** you can memorize — and that's the whole attention mechanism.

Inside the Layers – Attention in its simplest form



$$Q = HW_Q, \quad K = HW_K, \quad V = HW_V \quad \in \mathbb{R}^{d \times d}$$

$$\tilde{H} = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V = \underbrace{\text{softmax}\left(\frac{HW_QW_V^\top H^\top}{\sqrt{d}}\right)}_{W(H)} HW_V = [W(H)]HW_V \in \mathbb{R}^{T \times d}$$

$W(H)$ This is a weight matrix that **depends on the input H itself**.

A new kind of nonlinearity

Attention is nonlinear — but not in the way you've seen before

- In **FFNs, CNNs, RNNs**: nonlinearity comes from e.g. ReLU or tanh applied element-wise on top of **fixed** weight matrices.
- In **attention**: the mixing matrix $W(H)$ **depends on the input itself**. The weights change with every input.
- Once $W(H)$ is fixed, applying it to H is linear — but computing $W(H)$ from H is not. The full map is a **data-dependent linear map** — a pattern you'll meet again (hypernetworks, gating, state-space models).

Layers in classical nets

$$\tilde{H} = \sigma(H \cdot W)$$

W is fixed; σ is element-wise

Attention Layer

$$\tilde{H} = W(H) \cdot H W_v$$

$W(H)$ depends on the input H

Self-Attention: Projections (Row-wise)

Let $H \in \mathbb{R}^{T \times d}$ be the input to an attention layer. With the projection matrices (layer parameter)

$$W_Q \in \mathbb{R}^{d \times d}, \quad W_K \in \mathbb{R}^{d \times d}, \quad W_V \in \mathbb{R}^{d \times d}$$

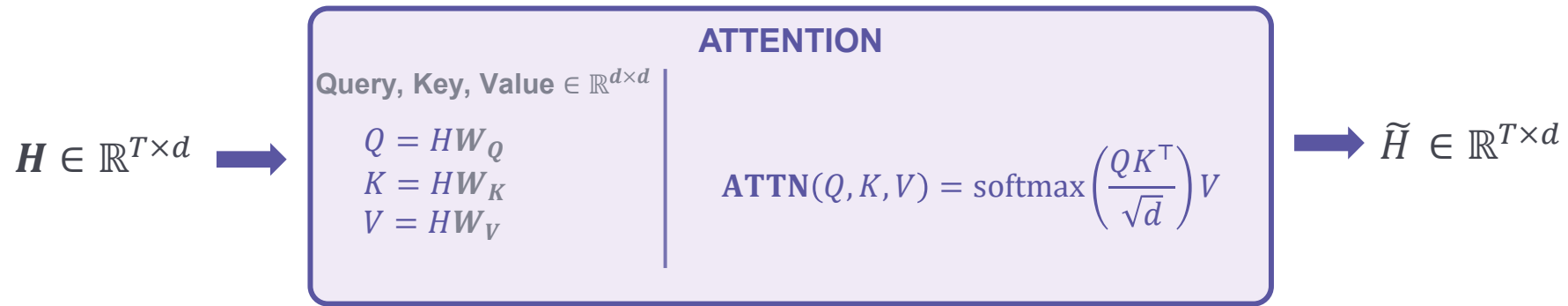
Define

$$Q = H W_Q, \quad K = H W_K, \quad V = H W_V$$

Right multiplication transforms every **token row** into a query / key / value row. So, we have $Q, K, V \in \mathbb{R}^{T \times d}$ and the output of the layer is given as

$$\tilde{H} = \text{softmax} \left(\frac{Q K^\top}{\sqrt{d}} \right) V = \text{softmax} \left(\frac{H W_Q W_V^\top H^\top}{\sqrt{d}} \right) H W_V = [W(H)] H W_V \in \mathbb{R}^{T \times d}$$

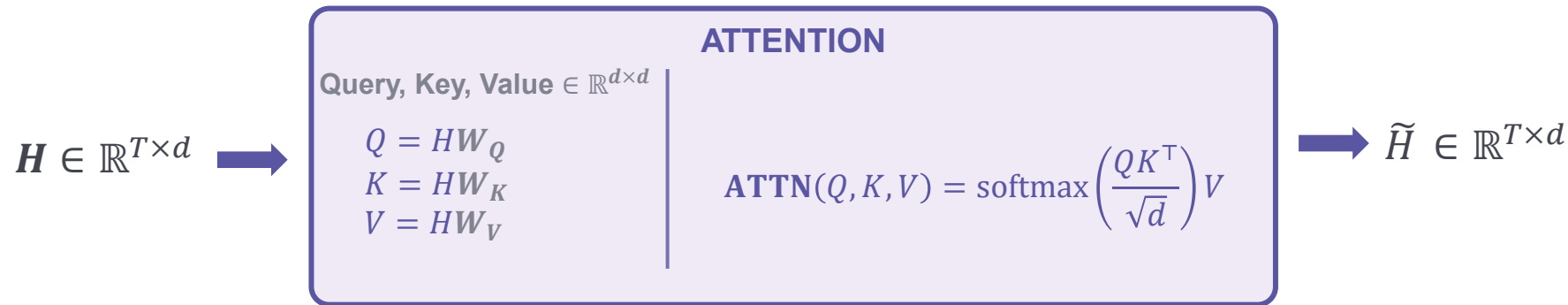
Giving meaning to Attention



$$K = HW_K = \begin{bmatrix} h_1^\top W_K \\ \vdots \\ h_T^\top W_K \end{bmatrix}, \quad [W_K^\top h_1, \dots, W_K^\top h_T] = W_K^\top H = K^\top$$

$$Q = HW_Q = \begin{bmatrix} h_1^\top W_Q \\ \vdots \\ h_T^\top W_Q \end{bmatrix}$$

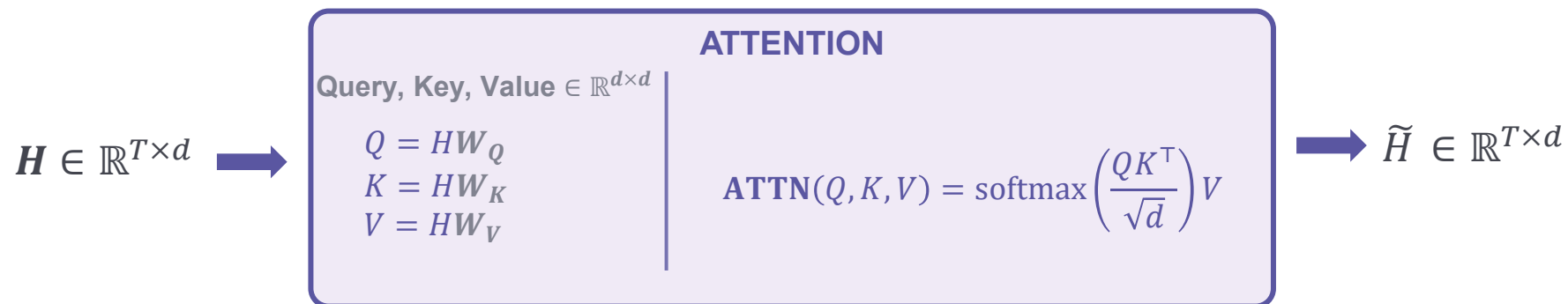
Giving meaning to Attention



$Q K^\top$	$[W_K^\top h_1, \quad \dots, \quad W_K^\top h_T] = W_K^\top H = K^\top$
$Q = HW_Q = \begin{bmatrix} h_1^\top W_Q \\ \vdots \\ h_T^\top W_Q \end{bmatrix}$	$\begin{bmatrix} h_1^\top W_Q W_K^\top h_1 & \dots & h_1^\top W_Q W_K^\top h_T \\ \vdots & \ddots & \vdots \\ h_T^\top W_Q W_K^\top h_1 & \dots & h_T^\top W_Q W_K^\top h_T \end{bmatrix}$ $\underbrace{\hspace{10em}}$

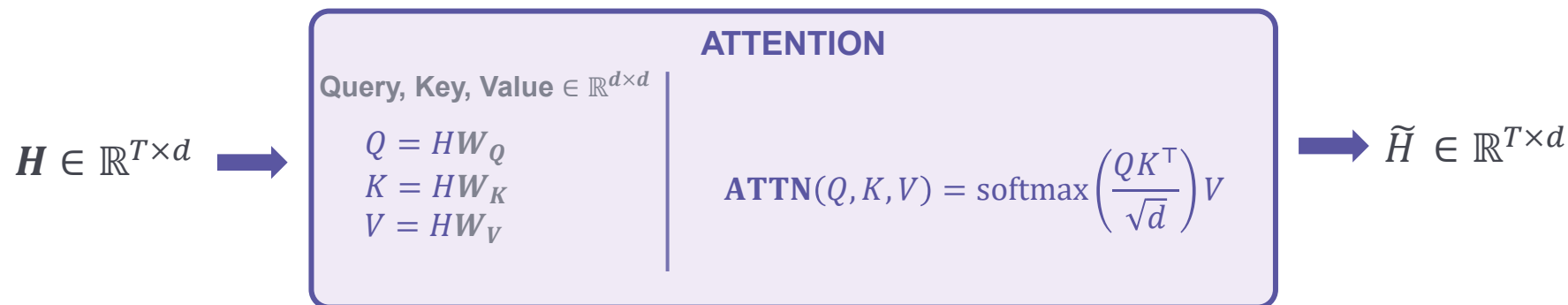
The Score matrix **S**

Giving meaning to Attention



$Q K^T$	$[W_K^T h_1, \dots, W_K^T h_T] = K^T$ $[k_1, \dots, k_T]$ the key vectors
$Q = \begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_T^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ \vdots \\ q_T^T \end{bmatrix}$ the query vectors	$\begin{bmatrix} q_1^T k_1 & \dots & q_1^T k_T \\ \vdots & \ddots & \vdots \\ q_T^T k_1 & \dots & q_T^T k_T \end{bmatrix}$ $\underbrace{\hspace{10em}}$ The Score matrix S

Giving meaning to Attention



$Q K^\top$	$[W_K^\top h_1, \dots, W_K^\top h_T] = K^\top$ $[k_1, \dots, k_T]$ the key vectors
$Q = \begin{bmatrix} h_1^\top W_Q \\ \vdots \\ h_T^\top W_Q \end{bmatrix} = \begin{bmatrix} q_1^\top \\ \vdots \\ q_T^\top \end{bmatrix}$ the query vectors	$\begin{bmatrix} \langle q_1, k_1 \rangle & \dots & \langle q_1, k_T \rangle \\ \vdots & \ddots & \vdots \\ \langle q_T, k_1 \rangle & \dots & \langle q_T, k_T \rangle \end{bmatrix}$ $\underbrace{\hspace{10em}}$ The Score matrix S

Score Entry as an Inner Product

Rows of the projected matrices are

$$\text{row } i \text{ of } Q = h_i^\top W_Q, \quad \text{row } j \text{ of } K = h_j^\top W_K.$$

The decomposition

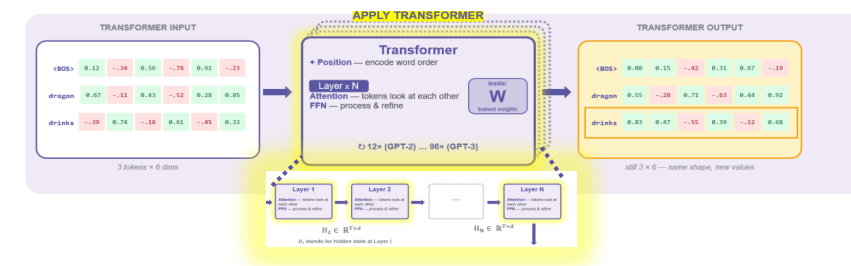
$$S_{ij} = q_i^\top k_j = (h_i^\top W_Q)(h_j^\top W_K)^\top = h_i^\top W_Q W_K^\top h_j = \langle W_Q^\top h_i, W_K^\top h_j \rangle.$$

A **scalar product** between the *learned query vector* $W_Q^\top h_i$ and the *learned key vector* $W_K^\top h_j$, both in $\mathbb{R}^d$.

(The middle step uses the transpose rule $(AB)^\top = B^\top A^\top$.)

“Learned” is explained on the next slide.

Callback: Inside the Layers



These **parameters** need to be determined in a practice called training the neural network. Where they will be learned from real data. The layout of the function blocks (i.e. the layers) just determine the model which has the chance by finding good values for these parameters to solve the actual task we have in mind. Here this was next token prediction.

Parameter: $3 \times d^2$

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

$$H_1 \in \mathbb{R}^{T \times d}$$

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$\begin{aligned} Q &= H_1 W_Q \\ K &= H_1 W_K \\ V &= H_1 W_V \end{aligned}$$

Attention 1

$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

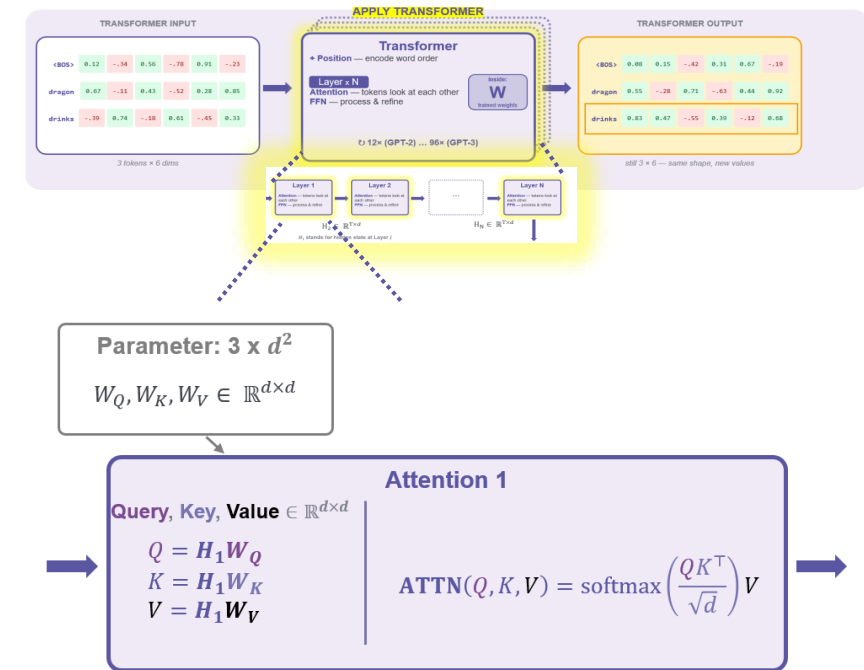
$$H_2 \in \mathbb{R}^{T \times d}$$

Aside: What 'learned' means

So far we've described the forward pass as a fixed mathematical function. But that function has many free matrices — W_Q, W_K, W_V , and many more — whose values we have not specified.

The **architecture** — the layout of the function blocks, the Q/K/V split, where each matrix sits — defines *which* parameters even exist. Without that design there would be no “learned query vector” to look for.

The remaining job is to **fit those parameters to data** so the model performs its task — for GPT-style models, **next-token prediction**. This step is called **training**; we'll return to *how* it works.



Geometric Reading of the Score

In the query/key subspace $\mathbb{R}^d$,

$$q_i^\top k_j = \|q_i\|_2 \|k_j\|_2 \cos \theta_{ij}.$$

Two geometric readings

1. **Projection.** $q_i^\top k_j$ is the length of the projection of q_i onto the direction of k_j , scaled by $\|k_j\|_2$.
2. **Magnitude × alignment.** Large iff *both* vectors are long *and* they point similarly.

“Token i attends to token j ” means: in the learned Q/K subspace, the query of i has a large component in the direction of the key of j .

Scores as a Learned Bilinear Form

Set $M = W_Q W_K^\top \in \mathbb{R}^{d \times d}$. Then

$$S_{ij} = h_i^\top M h_j.$$

The score matrix S is a **learned bilinear form** on the embedding space.

- **Asymmetry.** In general, $M \neq M^\top$, so $S_{ij} \neq S_{ji}$ — attention is *directional*. (“*The cat chased the mouse*”: the relation between *cat* and *mouse* is not symmetric.)
- **Low rank.** $\text{rank}(M) \leq d_k$. Factoring M into two $d \times d_k$ pieces is an explicit rank bottleneck.
- **Why two matrices, not one.** W_Q and W_K can learn *different* embeddings for “asking” and “being asked about” — the asymmetry is a **feature**, not a cost.

Why Dot Product, Not Cosine?

Self-attention uses $q_i^\top k_j$, *not* cosine similarity — and the reason is not just computational cheapness.

Cosine throws away a learned degree of freedom

W_Q and W_K learn *both* the **direction** and **magnitude** of q_i and k_j . Forcing unit norm discards that.

Norms can meaningfully encode:

- $\|q_i\|$ — how strongly token i **wants to attend** somewhere
- $\|k_j\|$ — how loudly token j **advertises itself**

Numerical stability of the raw dot product is handled by softmax plus the $\frac{1}{\sqrt{d}}$ scaling — cosine normalization is not needed for that.



Interpretation of Attention

7

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

$$V = HW_V$$

$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

TOKEN		d_1	d_2	$d_{12,288}$	
2234: the	h_1^T	-0.39	0.74	...	-0.45
5561: cat	h_2^T	-0.92	0.31	...	-1.25
4225: sat	h_3^T	0.27	-0.01	...	0.81
5561: on	h_4^T	-0.39	0.74	...	-0.45
7162: it	h_5^T	0.67	-0.11	...	0.28

Each **token** needs to understand the **context** in which it is surrounded.

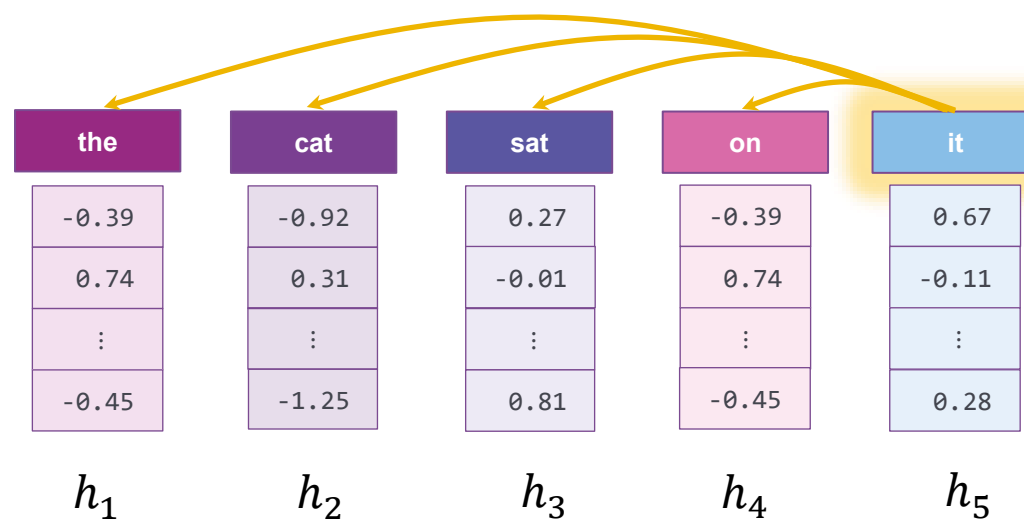
$Q K^T$

$[W_K^T h_1, \dots, W_K^T h_5]$
 $[k_1, \dots, k_T]$ key vectors

$$\begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ \vdots \\ q_T^T \end{bmatrix}$$

query vectors

$$\begin{bmatrix} \langle q_1, k_1 \rangle & \dots & \langle q_1, k_5 \rangle \\ \vdots & \ddots & \vdots \\ \langle q_5, k_1 \rangle & \dots & \langle q_5, k_5 \rangle \end{bmatrix}$$



How much should each token listen?

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

$$V = HW_V$$

$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$Q K^T$

$[W_K^T h_1, \dots, W_K^T h_5]$
 $[k_1, \dots, k_5]$ key vectors

$$\begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ \vdots \\ q_5^T \end{bmatrix}$$

query vectors

$$\begin{bmatrix} \langle q_1, k_1 \rangle & \dots & \langle q_1, k_5 \rangle \\ \vdots & \ddots & \vdots \\ \langle q_5, k_1 \rangle & \dots & \langle q_5, k_5 \rangle \end{bmatrix}$$

TOKEN		d_1	d_2	$d_{12,288}$	
2234: the	$h_1^T W_Q$	0.21	-0.14	...	-0.12
5561: cat	$h_2^T W_Q$	-0.14	-0.22	...	2.35
4225: sat	$h_3^T W_Q$	3.27	-0.01	...	-0.81
5561: on	$h_4^T W_Q$	-1.29	-0.77	...	2.45
7162: it	$h_5^T W_Q$	-1.67	0.31	...	-2.82

the	cat	sat	on	it
0.21	-0.14	3.27	-1.29	-1.67
-0.14	-0.22	-0.01	-0.77	0.31
⋮	⋮	⋮	⋮	⋮
-0.12	2.35	-0.81	2.45	-2.82

q_1

q_2

q_3

q_4

q_5

Query (Q). What am I looking for? — the question a token is asking about its context.

First, we compute the **learned query vectors** for each token (in parallel using matrix multiplication HW_Q)

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

$$V = HW_V$$

$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$Q K^T$

$[W_K^T h_1, \dots, W_K^T h_5]$
 $[k_1, \dots, k_5]$ key vectors

$$\begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ \vdots \\ q_5^T \end{bmatrix}$$

query vectors

$$\begin{bmatrix} \langle q_1, k_1 \rangle & \dots & \langle q_1, k_5 \rangle \\ \vdots & \ddots & \vdots \\ \langle q_5, k_1 \rangle & \dots & \langle q_5, k_5 \rangle \end{bmatrix}$$

TOKEN		d_1	d_2	$d_{12,288}$	
2234: the	$h_1^T W_K$	-1.11	-2.01	...	-1.23
5561: cat	$h_2^T W_K$	0.33	-1.52	...	3.11
4225: sat	$h_3^T W_K$	1.39	-2.02	...	0.01
5561: on	$h_4^T W_K$	-0.34	0.28	...	-2.21
7162: it	$h_5^T W_K$	-1.03	2.87	...	-1.24

the	cat	sat	on	it
-1.11	0.33	1.39	-0.34	-1.03
-2.01	-1.52	-2.02	0.28	2.87
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
-1.23	3.11	0.01	-2.21	-1.24
k_1	k_2	k_3	k_4	k_5

Key (K). *What do I contain?* — the advertisement a token publishes so that others can find it.

Then, we compute the **learned key vectors** for each token (in parallel using matrix multiplication HW_K)

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

$$V = HW_V$$

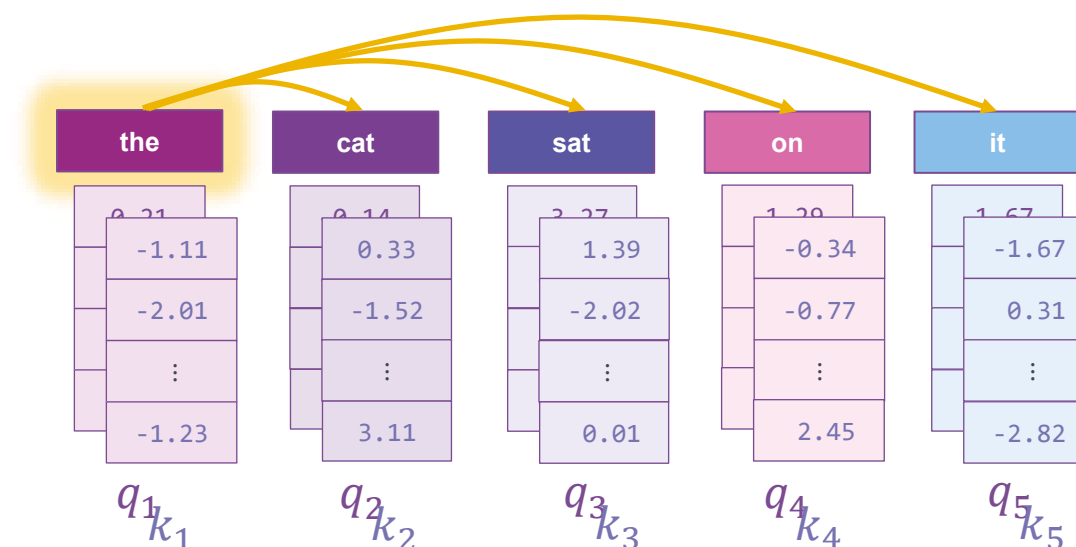
$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

TOKEN		d_1	d_2	$d_{12,288}$
2234: the	$h_1^T W_K$	0.30	0.74	0.45
5561: cat	$h_2^T W_K$	-1.11	-2.01	-1.23
4225: sat	$h_3^T W_K$	0.33	-1.52	3.11
5561: on	$h_4^T W_K$	1.39	-2.02	0.01
7162: it	$h_5^T W_K$	-0.34	0.28	-2.21
7162: it	$h_5^T W_K$	-1.03	2.87	-1.24

Query (Q). What am I looking for?

Key (K). What do I contain?

$$\begin{array}{c|ccccc}
 Q & K^T & [W_K^T h_1, & \dots & W_K^T h_5] \\
 & & [k_1 & k_2 & k_3 & k_4 & k_5] & \text{key vectors} \\
 \hline
 \begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix} & \text{query vectors} & \begin{bmatrix} \langle q_1, k_1 \rangle & \langle q_1, k_2 \rangle & \langle q_1, k_3 \rangle & \langle q_1, k_4 \rangle & \langle q_1, k_5 \rangle \\ \langle q_2, k_1 \rangle & \langle q_2, k_2 \rangle & \langle q_2, k_3 \rangle & \langle q_2, k_4 \rangle & \langle q_2, k_5 \rangle \\ \langle q_3, k_1 \rangle & \langle q_3, k_2 \rangle & \langle q_3, k_3 \rangle & \langle q_3, k_4 \rangle & \langle q_3, k_5 \rangle \\ \langle q_4, k_1 \rangle & \langle q_4, k_2 \rangle & \langle q_4, k_3 \rangle & \langle q_4, k_4 \rangle & \langle q_4, k_5 \rangle \\ \langle q_5, k_1 \rangle & \langle q_5, k_2 \rangle & \langle q_5, k_3 \rangle & \langle q_5, k_4 \rangle & \langle q_5, k_5 \rangle \end{bmatrix}
 \end{array}$$



Each **token** needs to understand the **context** in which it is surrounded.

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

$$V = HW_V$$

$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$QK^T = \begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix}$$

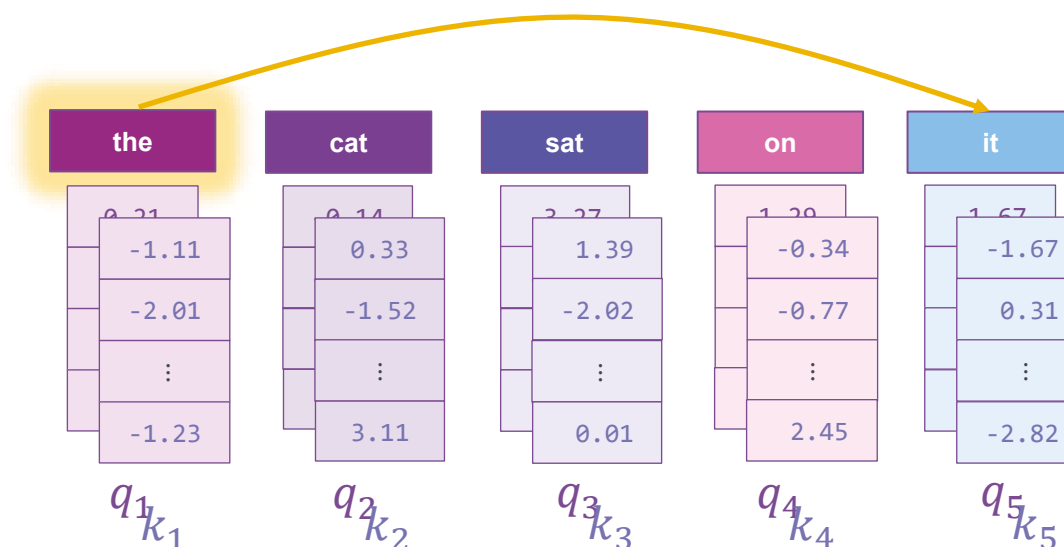
query vectors

$$[W_K^T h_1, \dots, W_K^T h_5] = \begin{bmatrix} k_1 & k_2 & k_3 & k_4 & k_5 \end{bmatrix}$$

key vectors

$$\begin{bmatrix} \langle q_1, k_1 \rangle & \langle q_1, k_2 \rangle & \langle q_1, k_3 \rangle & \langle q_1, k_4 \rangle & \langle q_1, k_5 \rangle \\ \langle q_2, k_1 \rangle & \langle q_2, k_2 \rangle & \langle q_2, k_3 \rangle & \langle q_2, k_4 \rangle & \langle q_2, k_5 \rangle \\ \langle q_3, k_1 \rangle & \langle q_3, k_2 \rangle & \langle q_3, k_3 \rangle & \langle q_3, k_4 \rangle & \langle q_3, k_5 \rangle \\ \langle q_4, k_1 \rangle & \langle q_4, k_2 \rangle & \langle q_4, k_3 \rangle & \langle q_4, k_4 \rangle & \langle q_4, k_5 \rangle \\ \langle q_5, k_1 \rangle & \langle q_5, k_2 \rangle & \langle q_5, k_3 \rangle & \langle q_5, k_4 \rangle & \langle q_5, k_5 \rangle \end{bmatrix}$$

TOKEN		d_1	d_2	$d_{12,288}$
2234: the	$h_1^T W_K$	-1.11	-2.01	...
5561: cat	$h_2^T W_K$	0.33	-1.52	...
4225: sat	$h_3^T W_K$	1.39	-2.02	...
5561: on	$h_4^T W_K$	-0.34	0.28	...
7162: it	$h_5^T W_K$	-1.03	2.87	...



Query (Q). What am I looking for?
Key (K). What do I contain?

“Token 1 attends to token 5” means: the query of 1 – q_1 has a large component in the direction of the key of 5 – k_5 .

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

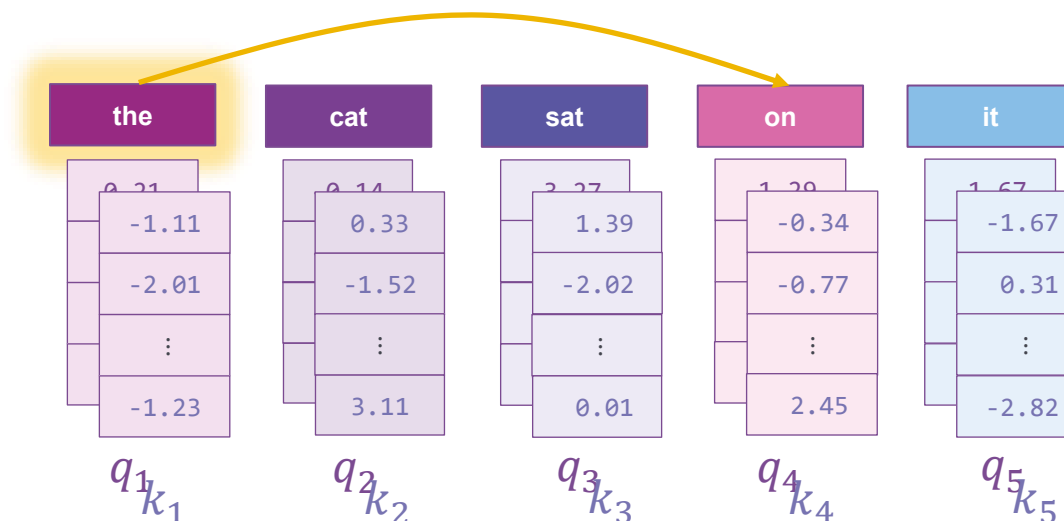
$$V = HW_V$$

$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$QK^T = \begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix} \quad \begin{array}{c|c} [W_K^T h_1, & \dots & W_K^T h_5] \\ [k_1 & k_2 & k_3 & k_4 & k_5] \text{ key vectors} \end{array}$$

$$\begin{bmatrix} \langle q_1, k_1 \rangle & \langle q_1, k_2 \rangle & \langle q_1, k_3 \rangle & \langle q_1, k_4 \rangle & \langle q_1, k_5 \rangle \\ \langle q_2, k_1 \rangle & \langle q_2, k_2 \rangle & \langle q_2, k_3 \rangle & \langle q_2, k_4 \rangle & \langle q_2, k_5 \rangle \\ \langle q_3, k_1 \rangle & \langle q_3, k_2 \rangle & \langle q_3, k_3 \rangle & \langle q_3, k_4 \rangle & \langle q_3, k_5 \rangle \\ \langle q_4, k_1 \rangle & \langle q_4, k_2 \rangle & \langle q_4, k_3 \rangle & \langle q_4, k_4 \rangle & \langle q_4, k_5 \rangle \\ \langle q_5, k_1 \rangle & \langle q_5, k_2 \rangle & \langle q_5, k_3 \rangle & \langle q_5, k_4 \rangle & \langle q_5, k_5 \rangle \end{bmatrix}$$

TOKEN		d_1	d_2	$d_{12,288}$	
2234: the	$h_1^T W_K$	-1.11	-2.01	...	-1.23
5561: cat	$h_2^T W_K$	0.33	-1.52	...	3.11
4225: sat	$h_3^T W_K$	1.39	-2.02	...	0.01
5561: on	$h_4^T W_K$	-0.34	0.28	...	-2.21
7162: it	$h_5^T W_K$	-1.03	2.87	...	-1.24



Query (Q). What am I looking for?
Key (K). What do I contain?

“Token 1 attends to token 4” means: the **query of 1** – q_1 has a large component in the direction of the **key of 4** – k_4 .

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

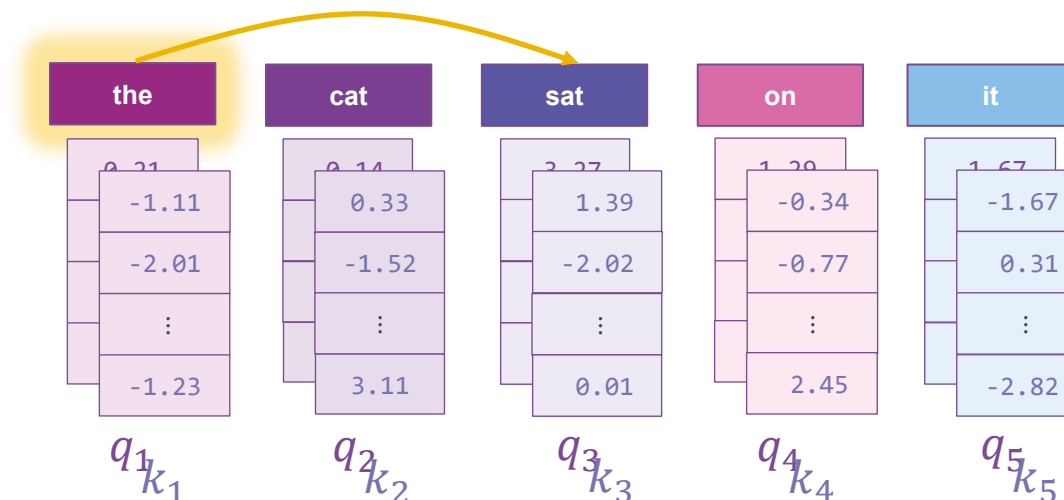
$$V = HW_V$$

$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$QK^T = \begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix} \quad \begin{array}{c|c} [W_K^T h_1, & \dots & W_K^T h_5] \\ [k_1 & k_2 & k_3 & k_4 & k_5] \text{ key vectors} \end{array}$$

$$\begin{bmatrix} \langle q_1, k_1 \rangle & \langle q_1, k_2 \rangle & \langle q_1, k_3 \rangle & \langle q_1, k_4 \rangle & \langle q_1, k_5 \rangle \\ \langle q_2, k_1 \rangle & \langle q_2, k_2 \rangle & \langle q_2, k_3 \rangle & \langle q_2, k_4 \rangle & \langle q_2, k_5 \rangle \\ \langle q_3, k_1 \rangle & \langle q_3, k_2 \rangle & \langle q_3, k_3 \rangle & \langle q_3, k_4 \rangle & \langle q_3, k_5 \rangle \\ \langle q_4, k_1 \rangle & \langle q_4, k_2 \rangle & \langle q_4, k_3 \rangle & \langle q_4, k_4 \rangle & \langle q_4, k_5 \rangle \\ \langle q_5, k_1 \rangle & \langle q_5, k_2 \rangle & \langle q_5, k_3 \rangle & \langle q_5, k_4 \rangle & \langle q_5, k_5 \rangle \end{bmatrix}$$

TOKEN		d_1	d_2	$d_{12,288}$	
2234: the	$h_1^T W_K$	-1.11	-2.01	...	-1.23
5561: cat	$h_2^T W_K$	0.33	-1.52	...	3.11
4225: sat	$h_3^T W_K$	1.39	-2.02	...	0.01
5561: on	$h_4^T W_K$	-0.34	0.28	...	-2.21
7162: it	$h_5^T W_K$	-1.03	2.87	...	-1.24



Query (Q). What am I looking for?
Key (K). What do I contain?

"Token 1 attends to token 3" means: the query of 1 – q_1 has a large component in the direction of the key of 3 – k_3 .

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

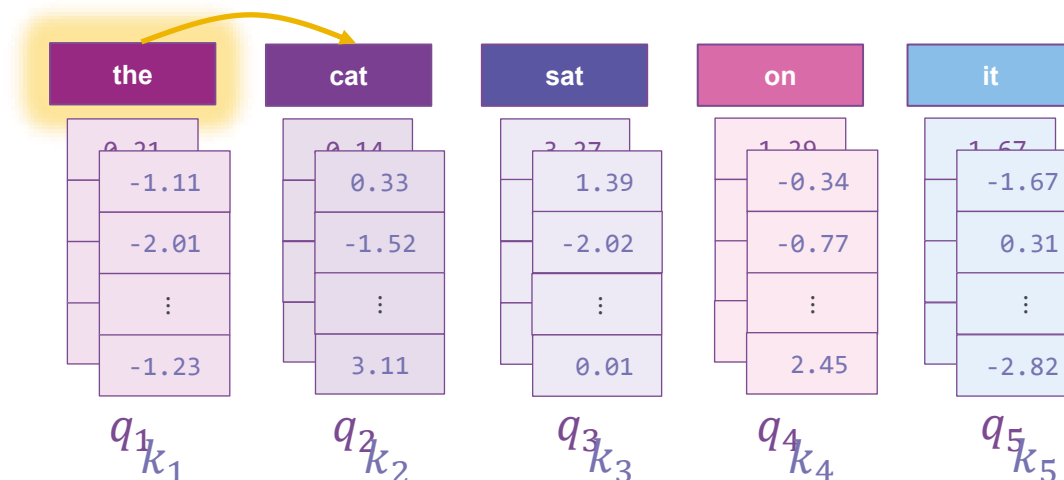
$$V = HW_V$$

$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$QK^T = \begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix} \quad \begin{array}{c|c} [W_K^T h_1, & \dots & W_K^T h_5] \\ [k_1 & k_2 & k_3 & k_4 & k_5] \text{ key vectors} \end{array}$$

$$\begin{bmatrix} \langle q_1, k_1 \rangle & \langle q_1, k_2 \rangle & \langle q_1, k_3 \rangle & \langle q_1, k_4 \rangle & \langle q_1, k_5 \rangle \\ \langle q_2, k_1 \rangle & \langle q_2, k_2 \rangle & \langle q_2, k_3 \rangle & \langle q_2, k_4 \rangle & \langle q_2, k_5 \rangle \\ \langle q_3, k_1 \rangle & \langle q_3, k_2 \rangle & \langle q_3, k_3 \rangle & \langle q_3, k_4 \rangle & \langle q_3, k_5 \rangle \\ \langle q_4, k_1 \rangle & \langle q_4, k_2 \rangle & \langle q_4, k_3 \rangle & \langle q_4, k_4 \rangle & \langle q_4, k_5 \rangle \\ \langle q_5, k_1 \rangle & \langle q_5, k_2 \rangle & \langle q_5, k_3 \rangle & \langle q_5, k_4 \rangle & \langle q_5, k_5 \rangle \end{bmatrix}$$

TOKEN		d_1	d_2	$d_{12,288}$	
2234: the	$h_1^T W_K$	-1.11	-2.01	...	-1.23
5561: cat	$h_2^T W_K$	0.33	-1.52	...	3.11
4225: sat	$h_3^T W_K$	1.39	-2.02	...	0.01
5561: on	$h_4^T W_K$	-0.34	0.28	...	-2.21
7162: it	$h_5^T W_K$	-1.03	2.87	...	-1.24



Query (Q). What am I looking for?
Key (K). What do I contain?

“Token 1 attends to token 2” means: the query of 1 – q_1 has a large component in the direction of the key of 2 – k_2 .

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

$$V = HW_V$$

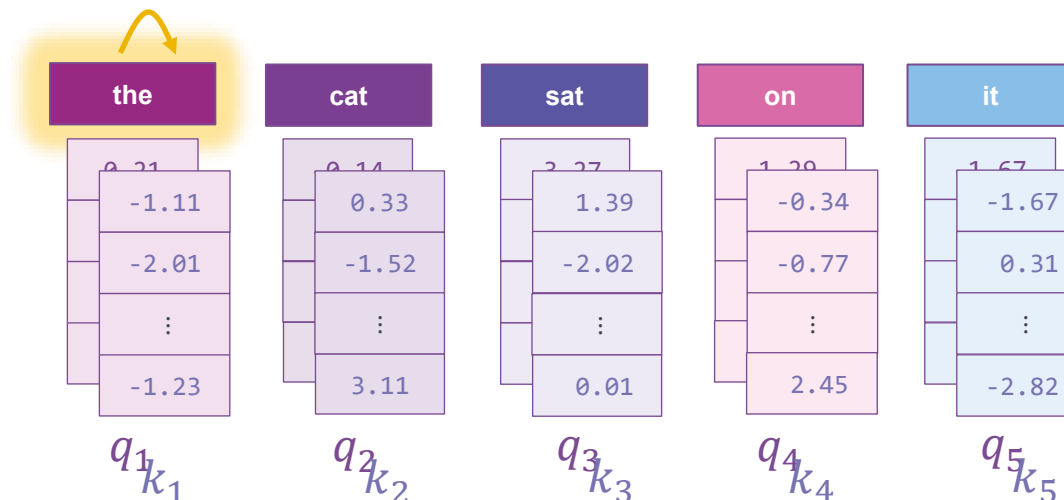
$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$QK^T = \begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix} \quad \begin{array}{c|c} [W_K^T h_1, & \dots & W_K^T h_5] \\ [k_1 & k_2 & k_3 & k_4 & k_5] \text{ key vectors} \end{array}$$

$$\begin{bmatrix} \langle q_1, k_1 \rangle & \langle q_1, k_2 \rangle & \langle q_1, k_3 \rangle & \langle q_1, k_4 \rangle & \langle q_1, k_5 \rangle \\ \langle q_2, k_1 \rangle & \langle q_2, k_2 \rangle & \langle q_2, k_3 \rangle & \langle q_2, k_4 \rangle & \langle q_2, k_5 \rangle \\ \langle q_3, k_1 \rangle & \langle q_3, k_2 \rangle & \langle q_3, k_3 \rangle & \langle q_3, k_4 \rangle & \langle q_3, k_5 \rangle \\ \langle q_4, k_1 \rangle & \langle q_4, k_2 \rangle & \langle q_4, k_3 \rangle & \langle q_4, k_4 \rangle & \langle q_4, k_5 \rangle \\ \langle q_5, k_1 \rangle & \langle q_5, k_2 \rangle & \langle q_5, k_3 \rangle & \langle q_5, k_4 \rangle & \langle q_5, k_5 \rangle \end{bmatrix}$$

query vectors

TOKEN		d_1	d_2	$d_{12,288}$	
2234: the	$h_1^T W_K$	-1.11	-2.01	...	-1.23
5561: cat	$h_2^T W_K$	0.33	-1.52	...	3.11
4225: sat	$h_3^T W_K$	1.39	-2.02	...	0.01
5561: on	$h_4^T W_K$	-0.34	0.28	...	-2.21
7162: it	$h_5^T W_K$	-1.03	2.87	...	-1.24



Query (Q). What am I looking for?
Key (K). What do I contain?

“Token 1 attends to token 1” means: the query of 1 – q_1 has a large component in the direction of the key of 1 – k_1 .

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

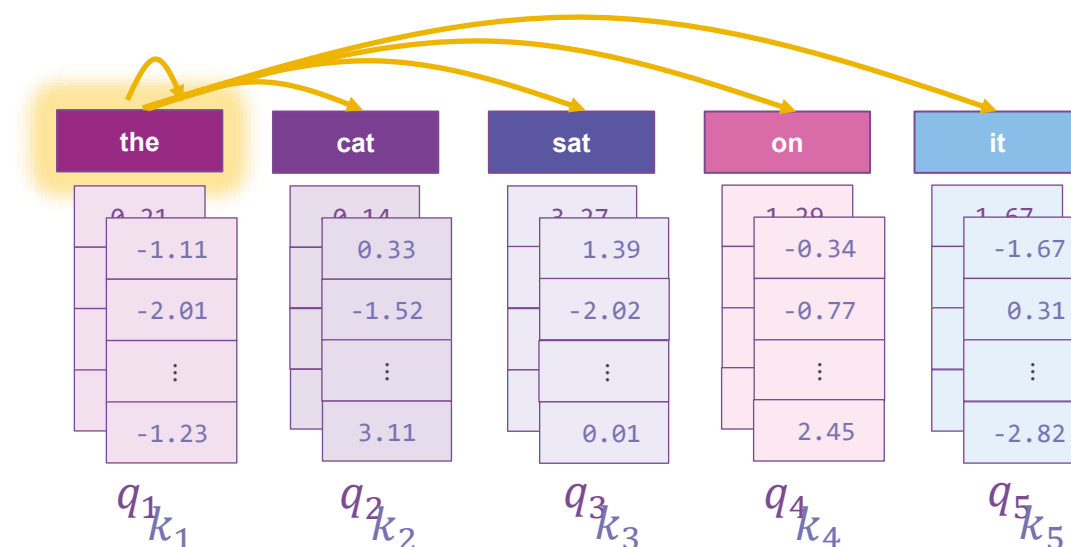
$$V = HW_V$$

$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

TOKEN		d_1	d_2	$d_{12,288}$	
2234: the	$h_1^T W_K$	-1.11	-2.01	...	-1.23
5561: cat	$h_2^T W_K$	0.33	-1.52	...	3.11
4225: sat	$h_3^T W_K$	1.39	-2.02	...	0.01
5561: on	$h_4^T W_K$	-0.34	0.28	...	-2.21
7162: it	$h_5^T W_K$	-1.03	2.87	...	-1.24

Query (Q). What am I looking for?
Key (K). What do I contain?

$$\begin{array}{c|ccccc}
 Q & K^T & [W_K^T h_1, & \dots & W_K^T h_5] \\
 & & [k_1 & k_2 & k_3 & k_4 & k_5] \text{ key vectors} \\
 \hline
 \begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix} \text{ query vectors} & & \begin{bmatrix} \langle q_1, k_1 \rangle & \langle q_1, k_2 \rangle & \langle q_1, k_3 \rangle & \langle q_1, k_4 \rangle & \langle q_1, k_5 \rangle \\ \langle q_2, k_1 \rangle & \langle q_2, k_2 \rangle & \langle q_2, k_3 \rangle & \langle q_2, k_4 \rangle & \langle q_2, k_5 \rangle \\ \langle q_3, k_1 \rangle & \langle q_3, k_2 \rangle & \langle q_3, k_3 \rangle & \langle q_3, k_4 \rangle & \langle q_3, k_5 \rangle \\ \langle q_4, k_1 \rangle & \langle q_4, k_2 \rangle & \langle q_4, k_3 \rangle & \langle q_4, k_4 \rangle & \langle q_4, k_5 \rangle \\ \langle q_5, k_1 \rangle & \langle q_5, k_2 \rangle & \langle q_5, k_3 \rangle & \langle q_5, k_4 \rangle & \langle q_5, k_5 \rangle \end{bmatrix}
 \end{array}$$



“Token 1 attends to token j” means: the **query** of 1 – q_1 has a large component in the direction of the **key** of j – k_j .

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

$$V = HW_V$$

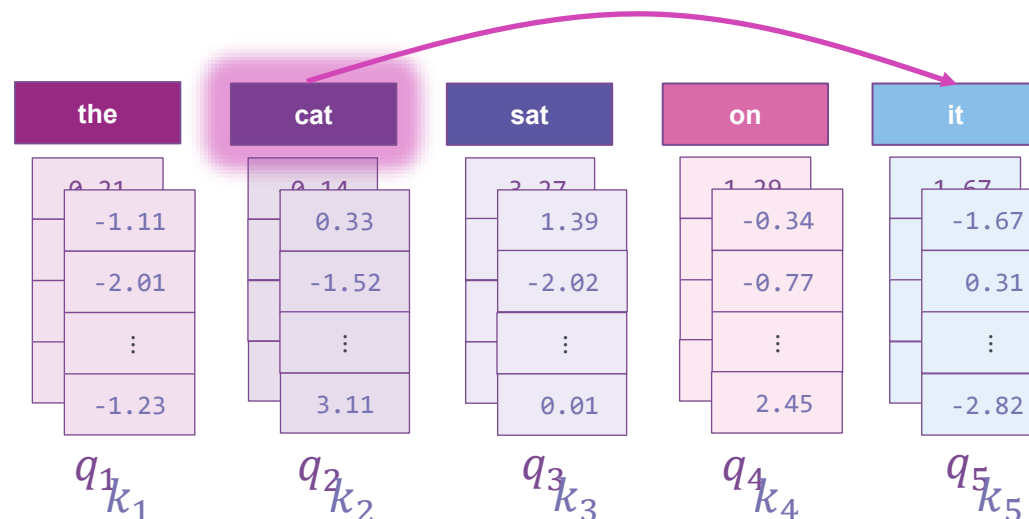
$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$QK^T = \begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix} \quad \begin{array}{c} [W_K^T h_1, \\ [k_1 \quad k_2 \quad k_3 \quad k_4 \quad k_5] \text{ key vectors} \\ W_K^T h_5] \end{array}$$

$$\begin{bmatrix} \langle q_1, k_1 \rangle & \langle q_1, k_2 \rangle & \langle q_1, k_3 \rangle & \langle q_1, k_4 \rangle & \langle q_1, k_5 \rangle \\ \langle q_2, k_1 \rangle & \langle q_2, k_2 \rangle & \langle q_2, k_3 \rangle & \langle q_2, k_4 \rangle & \langle q_2, k_5 \rangle \\ \langle q_3, k_1 \rangle & \langle q_3, k_2 \rangle & \langle q_3, k_3 \rangle & \langle q_3, k_4 \rangle & \langle q_3, k_5 \rangle \\ \langle q_4, k_1 \rangle & \langle q_4, k_2 \rangle & \langle q_4, k_3 \rangle & \langle q_4, k_4 \rangle & \langle q_4, k_5 \rangle \\ \langle q_5, k_1 \rangle & \langle q_5, k_2 \rangle & \langle q_5, k_3 \rangle & \langle q_5, k_4 \rangle & \langle q_5, k_5 \rangle \end{bmatrix}$$

query vectors

TOKEN		d_1	d_2	$d_{12,288}$
2234: the	$h_1^T W_K$	0.30	0.74	0.45
5561: cat	$h_2^T W_K$	-1.11	-2.01	-1.23
4225: sat	$h_3^T W_K$	0.33	-1.52	3.11
5561: on	$h_4^T W_K$	1.39	-2.02	0.01
7162: it	$h_5^T W_K$	-0.34	0.28	-2.21
7162: it	$h_5^T W_K$	-1.03	2.87	-1.24



Query (Q). What am I looking for?
Key (K). What do I contain?

“Token 2 attends to token 5” means: the query of 2 – q_2 has a large component in the direction of the key of 5 – k_5 .

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

$$V = HW_V$$

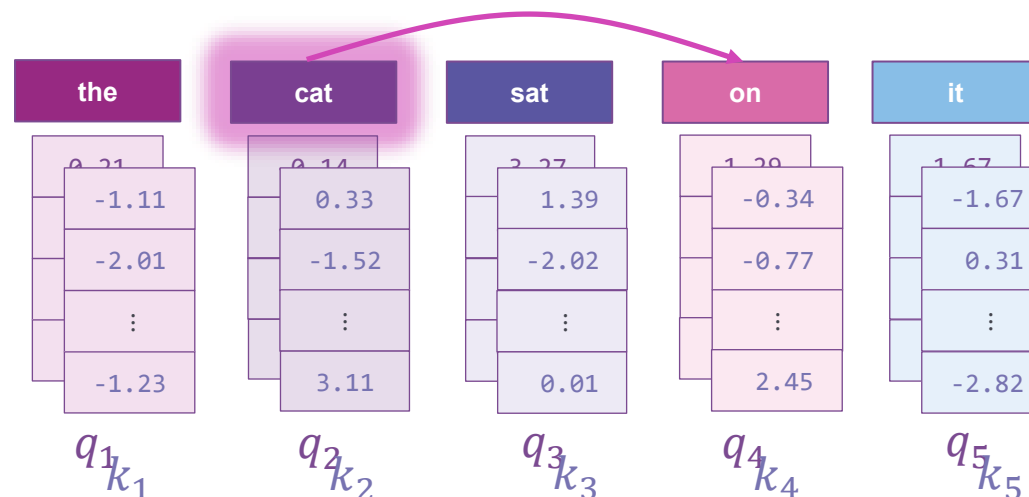
$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$QK^T = \begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix} \quad \text{query vectors}$$

$$[W_K^T h_1, \dots, W_K^T h_5] = \begin{bmatrix} k_1 & k_2 & k_3 & k_4 & k_5 \end{bmatrix} \quad \text{key vectors}$$

$$\begin{bmatrix} \langle q_1, k_1 \rangle & \langle q_1, k_2 \rangle & \langle q_1, k_3 \rangle & \langle q_1, k_4 \rangle & \langle q_1, k_5 \rangle \\ \langle q_2, k_1 \rangle & \langle q_2, k_2 \rangle & \langle q_2, k_3 \rangle & \langle q_2, k_4 \rangle & \langle q_2, k_5 \rangle \\ \langle q_3, k_1 \rangle & \langle q_3, k_2 \rangle & \langle q_3, k_3 \rangle & \langle q_3, k_4 \rangle & \langle q_3, k_5 \rangle \\ \langle q_4, k_1 \rangle & \langle q_4, k_2 \rangle & \langle q_4, k_3 \rangle & \langle q_4, k_4 \rangle & \langle q_4, k_5 \rangle \\ \langle q_5, k_1 \rangle & \langle q_5, k_2 \rangle & \langle q_5, k_3 \rangle & \langle q_5, k_4 \rangle & \langle q_5, k_5 \rangle \end{bmatrix}$$

TOKEN		d_1	d_2	$d_{12,288}$	
2234: the	$h_1^T W_K$	-1.11	-2.01	...	-1.23
5561: cat	$h_2^T W_K$	0.33	-1.52	...	3.11
4225: sat	$h_3^T W_K$	1.39	-2.02	...	0.01
5561: on	$h_4^T W_K$	-0.34	0.28	...	-2.21
7162: it	$h_5^T W_K$	-1.03	2.87	...	-1.24



Query (Q). What am I looking for?
Key (K). What do I contain?

“Token 2 attends to token 4” means: the query of 2 – q_2 has a large component in the direction of the key of 4 – k_4 .

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

$$V = HW_V$$

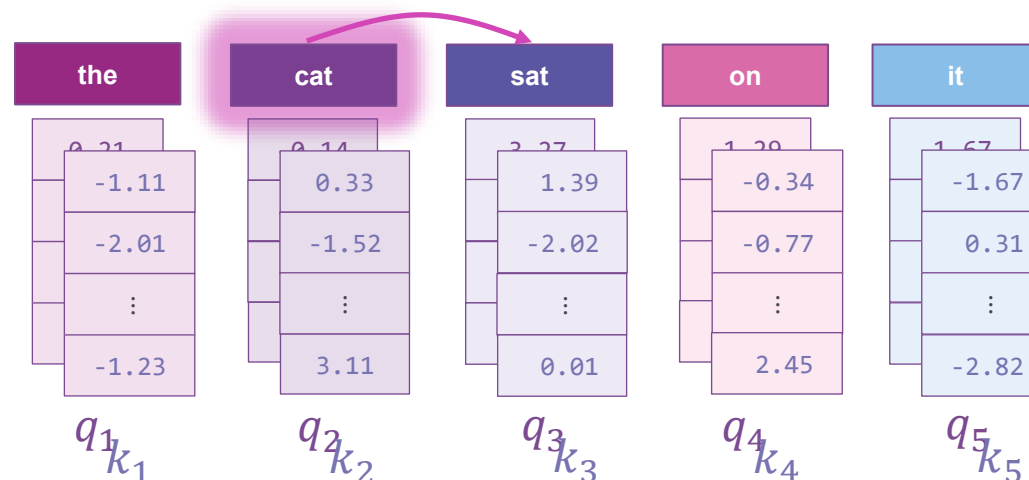
$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$QK^T = \begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix} \quad \text{query vectors}$$

$$[W_K^T h_1, \dots, W_K^T h_5] = \begin{bmatrix} k_1 & k_2 & k_3 & k_4 & k_5 \end{bmatrix} \quad \text{key vectors}$$

$$\begin{bmatrix} \langle q_1, k_1 \rangle & \langle q_1, k_2 \rangle & \langle q_1, k_3 \rangle & \langle q_1, k_4 \rangle & \langle q_1, k_5 \rangle \\ \langle q_2, k_1 \rangle & \langle q_2, k_2 \rangle & \langle q_2, k_3 \rangle & \langle q_2, k_4 \rangle & \langle q_2, k_5 \rangle \\ \langle q_3, k_1 \rangle & \langle q_3, k_2 \rangle & \langle q_3, k_3 \rangle & \langle q_3, k_4 \rangle & \langle q_3, k_5 \rangle \\ \langle q_4, k_1 \rangle & \langle q_4, k_2 \rangle & \langle q_4, k_3 \rangle & \langle q_4, k_4 \rangle & \langle q_4, k_5 \rangle \\ \langle q_5, k_1 \rangle & \langle q_5, k_2 \rangle & \langle q_5, k_3 \rangle & \langle q_5, k_4 \rangle & \langle q_5, k_5 \rangle \end{bmatrix}$$

TOKEN		d_1	d_2	$d_{12,288}$	
2234: the	$h_1^T W_K$	-1.11	-2.01	...	-1.23
5561: cat	$h_2^T W_K$	0.33	-1.52	...	3.11
4225: sat	$h_3^T W_K$	1.39	-2.02	...	0.01
5561: on	$h_4^T W_K$	-0.34	0.28	...	-2.21
7162: it	$h_5^T W_K$	-1.03	2.87	...	-1.24



Query (Q). What am I looking for?
Key (K). What do I contain?

“Token 2 attends to token 3” means: the query of 2 – q_2 has a large component in the direction of the key of 3 – k_3 .

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

$$V = HW_V$$

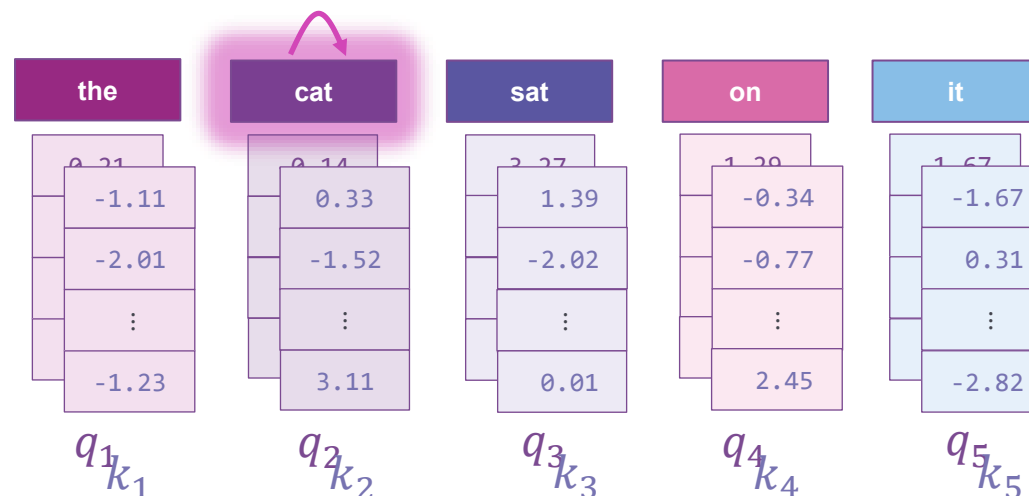
$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$QK^T = \begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix} \quad \begin{array}{c|c} [W_K^T h_1, & \dots & W_K^T h_5] \\ [k_1 & k_2 & k_3 & k_4 & k_5] \text{ key vectors} \end{array}$$

$$\begin{bmatrix} \langle q_1, k_1 \rangle & \langle q_1, k_2 \rangle & \langle q_1, k_3 \rangle & \langle q_1, k_4 \rangle & \langle q_1, k_5 \rangle \\ \langle q_2, k_1 \rangle & \langle q_2, k_2 \rangle & \langle q_2, k_3 \rangle & \langle q_2, k_4 \rangle & \langle q_2, k_5 \rangle \\ \langle q_3, k_1 \rangle & \langle q_3, k_2 \rangle & \langle q_3, k_3 \rangle & \langle q_3, k_4 \rangle & \langle q_3, k_5 \rangle \\ \langle q_4, k_1 \rangle & \langle q_4, k_2 \rangle & \langle q_4, k_3 \rangle & \langle q_4, k_4 \rangle & \langle q_4, k_5 \rangle \\ \langle q_5, k_1 \rangle & \langle q_5, k_2 \rangle & \langle q_5, k_3 \rangle & \langle q_5, k_4 \rangle & \langle q_5, k_5 \rangle \end{bmatrix}$$

query vectors

TOKEN		d_1	d_2	$d_{12,288}$	
2234: the	$h_1^T W_K$	-1.11	-2.01	...	-1.23
5561: cat	$h_2^T W_K$	0.33	-1.52	...	3.11
4225: sat	$h_3^T W_K$	1.39	-2.02	...	0.01
5561: on	$h_4^T W_K$	-0.34	0.28	...	-2.21
7162: it	$h_5^T W_K$	-1.03	2.87	...	-1.24



Query (Q). What am I looking for?
Key (K). What do I contain?

“Token 2 attends to token 2” means: the query of 2 – q_2 has a large component in the direction of the key of 2 – k_2 .

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

$$V = HW_V$$

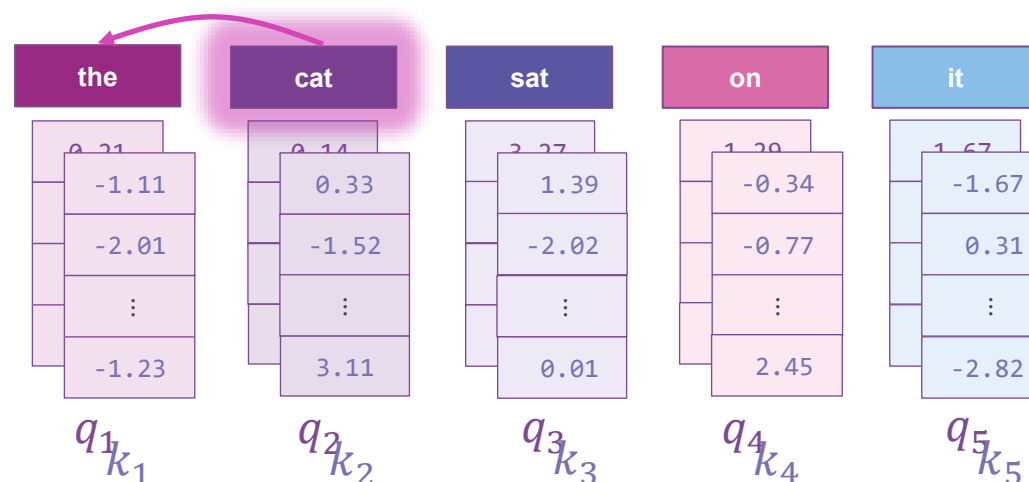
$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$QK^T = \begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix} \quad \text{query vectors}$$

$$[W_K^T h_1, \dots, W_K^T h_5] = \begin{bmatrix} k_1 & k_2 & k_3 & k_4 & k_5 \end{bmatrix} \quad \text{key vectors}$$

$$\begin{bmatrix} \langle q_1, k_1 \rangle & \langle q_1, k_2 \rangle & \langle q_1, k_3 \rangle & \langle q_1, k_4 \rangle & \langle q_1, k_5 \rangle \\ \langle q_2, k_1 \rangle & \langle q_2, k_2 \rangle & \langle q_2, k_3 \rangle & \langle q_2, k_4 \rangle & \langle q_2, k_5 \rangle \\ \langle q_3, k_1 \rangle & \langle q_3, k_2 \rangle & \langle q_3, k_3 \rangle & \langle q_3, k_4 \rangle & \langle q_3, k_5 \rangle \\ \langle q_4, k_1 \rangle & \langle q_4, k_2 \rangle & \langle q_4, k_3 \rangle & \langle q_4, k_4 \rangle & \langle q_4, k_5 \rangle \\ \langle q_5, k_1 \rangle & \langle q_5, k_2 \rangle & \langle q_5, k_3 \rangle & \langle q_5, k_4 \rangle & \langle q_5, k_5 \rangle \end{bmatrix}$$

TOKEN		d_1	d_2	$d_{12,288}$	
2234: the	$h_1^T W_K$	-1.11	-2.01	...	-1.23
5561: cat	$h_2^T W_K$	0.33	-1.52	...	3.11
4225: sat	$h_3^T W_K$	1.39	-2.02	...	0.01
5561: on	$h_4^T W_K$	-0.34	0.28	...	-2.21
7162: it	$h_5^T W_K$	-1.03	2.87	...	-1.24



Query (Q). What am I looking for?
Key (K). What do I contain?

“Token 2 attends to token 1” means: the query of 2 – q_2 has a large component in the direction of the key of 1 – k_1 .

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

$$V = HW_V$$

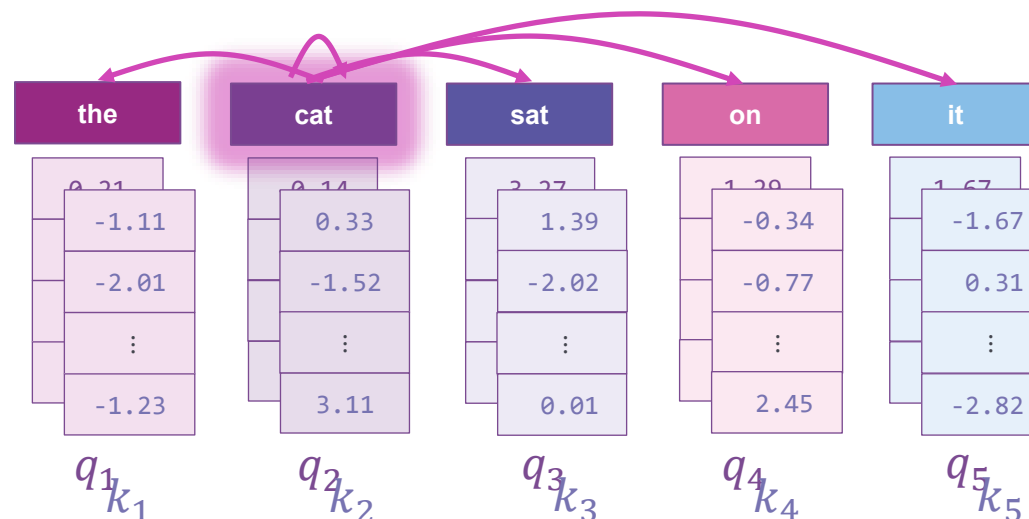
$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

TOKEN	d_1		d_2	$d_{12,288}$	
2234: the	$h_1^T W_K$	-1.11	-2.01	...	-1.23
5561: cat	$h_2^T W_K$	0.33	-1.52	...	3.11
4225: sat	$h_3^T W_K$	1.39	-2.02	...	0.01
5561: on	$h_4^T W_K$	-0.34	0.28	...	-2.21
7162: it	$h_5^T W_K$	-1.03	2.87	...	-1.24

Query (Q). What am I looking for?
Key (K). What do I contain?

$$\begin{array}{c|ccccc}
 Q K^T & [W_K^T h_1, & & \dots & W_K^T h_5] \\
 & [k_1 & k_2 & k_3 & k_4 & k_5] \text{ key vectors} \\
 \hline
 \begin{array}{c} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{array} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix} & \begin{bmatrix} \langle q_1, k_1 \rangle & \langle q_1, k_2 \rangle & \langle q_1, k_3 \rangle & \langle q_1, k_4 \rangle & \langle q_1, k_5 \rangle \\ \langle q_2, k_1 \rangle & \langle q_2, k_2 \rangle & \langle q_2, k_3 \rangle & \langle q_2, k_4 \rangle & \langle q_2, k_5 \rangle \\ \langle q_3, k_1 \rangle & \langle q_3, k_2 \rangle & \langle q_3, k_3 \rangle & \langle q_3, k_4 \rangle & \langle q_3, k_5 \rangle \\ \langle q_4, k_1 \rangle & \langle q_4, k_2 \rangle & \langle q_4, k_3 \rangle & \langle q_4, k_4 \rangle & \langle q_4, k_5 \rangle \\ \langle q_5, k_1 \rangle & \langle q_5, k_2 \rangle & \langle q_5, k_3 \rangle & \langle q_5, k_4 \rangle & \langle q_5, k_5 \rangle \end{bmatrix}
 \end{array}$$

query vectors



“Token 2 attends to token j” means: the query of 2 – q_2 has a large component in the direction of the key of j – k_j .

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

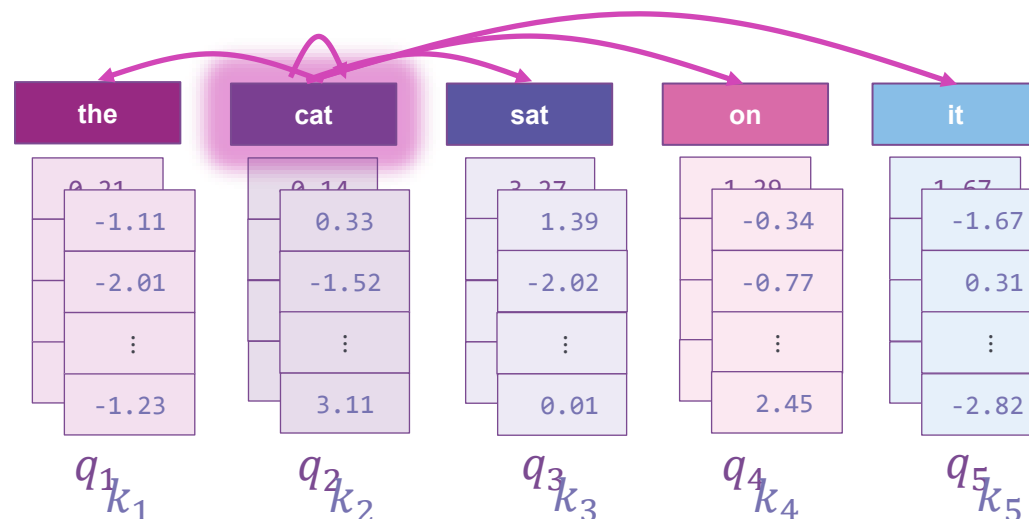
$$V = HW_V$$

$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

TOKEN	d_1		d_2	$d_{12,288}$	
2234: the	$h_1^T W_K$	-1.11	-2.01	...	-1.23
5561: cat	$h_2^T W_K$	0.33	-1.52	...	3.11
4225: sat	$h_3^T W_K$	1.39	-2.02	...	0.01
5561: on	$h_4^T W_K$	-0.34	0.28	...	-2.21
7162: it	$h_5^T W_K$	-1.03	2.87	...	-1.24

Query (Q). What am I looking for?
Key (K). What do I contain?

$$\begin{array}{c|ccccc}
 Q K^T & [W_K^T h_1, & \dots & W_K^T h_5] & \\
 & [k_1 & k_2 & k_3 & k_4 & k_5] \text{ key vectors} \\
 \hline
 \begin{array}{c} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{array} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix} & \begin{bmatrix} \langle q_1, k_1 \rangle & \langle q_1, k_2 \rangle & \langle q_1, k_3 \rangle & \langle q_1, k_4 \rangle & \langle q_1, k_5 \rangle \\ \langle q_2, k_1 \rangle & \langle q_2, k_2 \rangle & \langle q_2, k_3 \rangle & \langle q_2, k_4 \rangle & \langle q_2, k_5 \rangle \\ \langle q_3, k_1 \rangle & \langle q_3, k_2 \rangle & \langle q_3, k_3 \rangle & \langle q_3, k_4 \rangle & \langle q_3, k_5 \rangle \\ \langle q_4, k_1 \rangle & \langle q_4, k_2 \rangle & \langle q_4, k_3 \rangle & \langle q_4, k_4 \rangle & \langle q_4, k_5 \rangle \\ \langle q_5, k_1 \rangle & \langle q_5, k_2 \rangle & \langle q_5, k_3 \rangle & \langle q_5, k_4 \rangle & \langle q_5, k_5 \rangle \end{bmatrix} \\
 \text{query vectors} & & & &
 \end{array}$$



“Token 2 attends to token j” means: the **query** of 2 – q_2 has a large component in the direction of the **key** of j – k_j .

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

$$V = HW_V$$

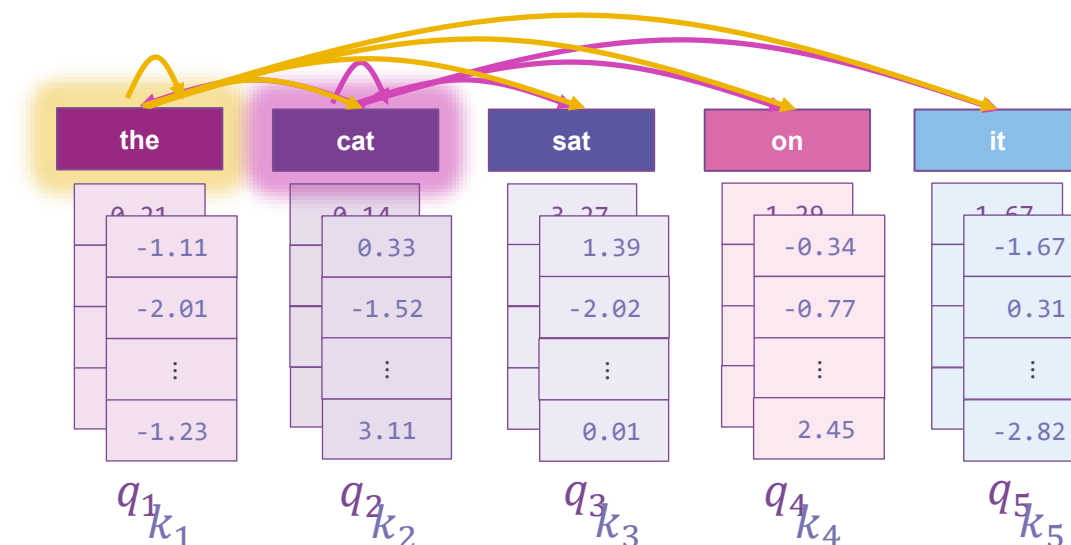
$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

TOKEN		d_1	d_2	$d_{12,288}$	
2234: the	$h_1^T W_K$	-1.11	-2.01	...	-1.23
5561: cat	$h_2^T W_K$	0.33	-1.52	...	3.11
4225: sat	$h_3^T W_K$	1.39	-2.02	...	0.01
5561: on	$h_4^T W_K$	-0.34	0.28	...	-2.21
7162: it	$h_5^T W_K$	-1.03	2.87	...	-1.24

Query (Q). What am I looking for?

Key (K). What do I contain?

$Q K^T$	$[W_K^T h_1,$ $[k_1$	k_2	$\dots$ k_3	k_4	$W_K^T h_5]$ $k_5]$	key vectors
$\begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix}$ query vectors	$\langle q_1, k_1 \rangle$	$\langle q_1, k_2 \rangle$	$\langle q_1, k_3 \rangle$	$\langle q_1, k_4 \rangle$	$\langle q_1, k_5 \rangle$	
	$\langle q_2, k_1 \rangle$	$\langle q_2, k_2 \rangle$	$\langle q_2, k_3 \rangle$	$\langle q_2, k_4 \rangle$	$\langle q_2, k_5 \rangle$	
	$\langle q_3, k_1 \rangle$	$\langle q_3, k_2 \rangle$	$\langle q_3, k_3 \rangle$	$\langle q_3, k_4 \rangle$	$\langle q_3, k_5 \rangle$	
	$\langle q_4, k_1 \rangle$	$\langle q_4, k_2 \rangle$	$\langle q_4, k_3 \rangle$	$\langle q_4, k_4 \rangle$	$\langle q_4, k_5 \rangle$	
	$\langle q_5, k_1 \rangle$	$\langle q_5, k_2 \rangle$	$\langle q_5, k_3 \rangle$	$\langle q_5, k_4 \rangle$	$\langle q_5, k_5 \rangle$	



"Token 1 attends to token j" – First row of QK^T

"Token 2 attends to token j" – Second row of QK^T

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

$$V = HW_V$$

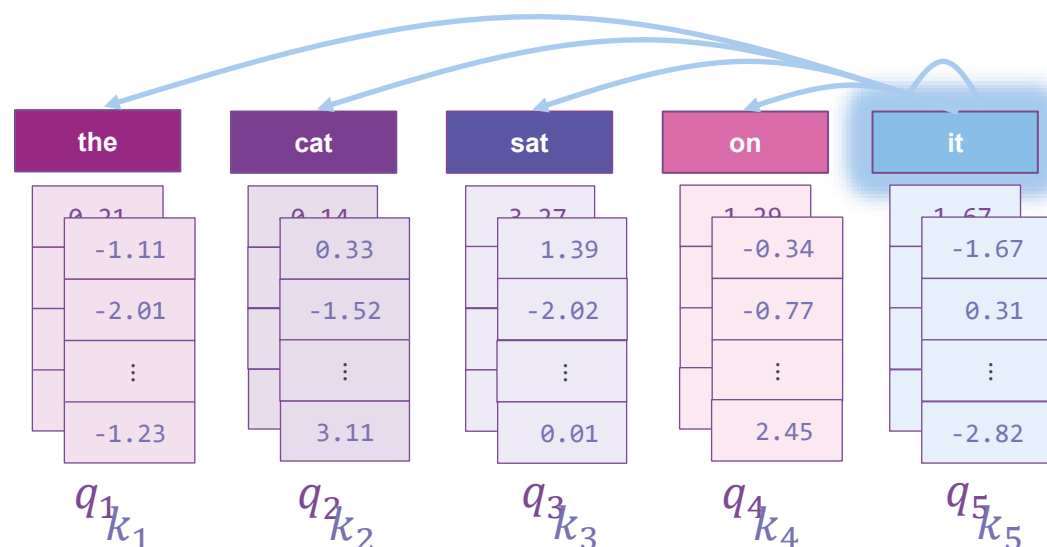
$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

TOKEN		d_1	d_2	$d_{12,288}$
2234: the	$h_1^T W_K$	0.30	0.74	0.45
5561: cat	$h_2^T W_K$	-1.11	-2.01	-1.23
4225: sat	$h_3^T W_K$	0.33	-1.52	3.11
5561: on	$h_4^T W_K$	1.39	-2.02	0.01
7162: it	$h_5^T W_K$	-0.34	0.28	-2.21
		-1.03	2.87	-1.24

Query (Q). What am I looking for?

Key (K). What do I contain?

$Q K^T$	$[W_K^T h_1,$ $[k_1$	k_2	k_3	k_4	$W_K^T h_5]$ $k_5]$ key vectors
$\begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix}$ query vectors	$\langle q_1, k_1 \rangle$ $\langle q_2, k_1 \rangle$ $\langle q_3, k_1 \rangle$ $\langle q_4, k_1 \rangle$ $\langle q_5, k_1 \rangle$	$\langle q_1, k_2 \rangle$ $\langle q_2, k_2 \rangle$ $\langle q_3, k_2 \rangle$ $\langle q_4, k_2 \rangle$ $\langle q_5, k_2 \rangle$	$\langle q_1, k_3 \rangle$ $\langle q_2, k_3 \rangle$ $\langle q_3, k_3 \rangle$ $\langle q_4, k_3 \rangle$ $\langle q_5, k_3 \rangle$	$\langle q_1, k_4 \rangle$ $\langle q_2, k_4 \rangle$ $\langle q_3, k_4 \rangle$ $\langle q_4, k_4 \rangle$ $\langle q_5, k_4 \rangle$	$\langle q_1, k_5 \rangle$ $\langle q_2, k_5 \rangle$ $\langle q_3, k_5 \rangle$ $\langle q_4, k_5 \rangle$ $\langle q_5, k_5 \rangle$



“Token 5 attends to token j” – Fifth row of QK^T

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

$$V = HW_V$$

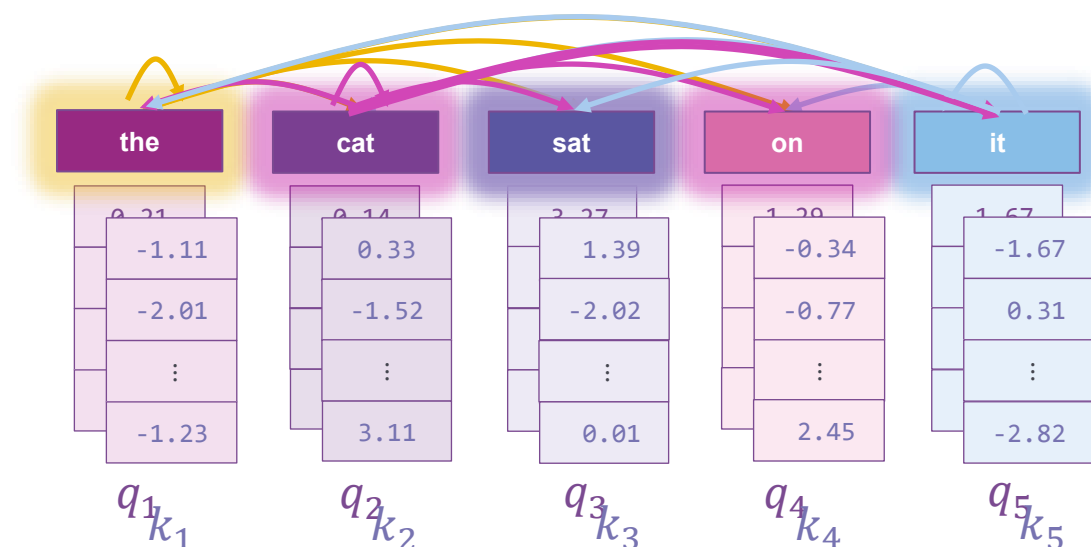
$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

TOKEN		d_1	d_2	$d_{12,288}$	
2234: the	$h_1^T W_K$	-1.11	-2.01	...	-1.23
5561: cat	$h_2^T W_K$	0.33	-1.52	...	3.11
4225: sat	$h_3^T W_K$	1.39	-2.02	...	0.01
5561: on	$h_4^T W_K$	-0.34	0.28	...	-2.21
7162: it	$h_5^T W_K$	-1.03	2.87	...	-1.24

Query (Q). What am I looking for?

Key (K). What do I contain?

$Q K^T$	$[W_K^T h_1,$ $[k_1$	k_2	$\dots$ k_3	k_4	$W_K^T h_5]$ $k_5]$	key vectors
$\begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix}$ query vectors	$\langle q_1, k_1 \rangle$	$\langle q_1, k_2 \rangle$	$\langle q_1, k_3 \rangle$	$\langle q_1, k_4 \rangle$	$\langle q_1, k_5 \rangle$	
	$\langle q_2, k_1 \rangle$	$\langle q_2, k_2 \rangle$	$\langle q_2, k_3 \rangle$	$\langle q_2, k_4 \rangle$	$\langle q_2, k_5 \rangle$	
	$\langle q_3, k_1 \rangle$	$\langle q_3, k_2 \rangle$	$\langle q_3, k_3 \rangle$	$\langle q_3, k_4 \rangle$	$\langle q_3, k_5 \rangle$	
	$\langle q_4, k_1 \rangle$	$\langle q_4, k_2 \rangle$	$\langle q_4, k_3 \rangle$	$\langle q_4, k_4 \rangle$	$\langle q_4, k_5 \rangle$	
	$\langle q_5, k_1 \rangle$	$\langle q_5, k_2 \rangle$	$\langle q_5, k_3 \rangle$	$\langle q_5, k_4 \rangle$	$\langle q_5, k_5 \rangle$	



"Token i attends to all other token" – i th row of QK^T

From bidirectional to causal attention

- So far, we computed scores for **all $T \times T$ pairs** — every token attends to every other (bidirectional, like BERT).
- But our goal is a **next-token predictor** (GPT, decoder-only): a function that takes **only the past** as input and predicts the next token.
- **Example** — from “**The cat sat on it**”, after seeing {**The, cat, sat**} the model must predict the **next** token — ideally “**on**”. So “**on**” and “**it**” are exactly the answers we are trying to produce.
- **Why not just use everything available?** Because at **inference** the future does not exist yet — the model has to work from past tokens alone. During **training** we *do* have the whole sentence in memory — but if we let the model peek at it, we would learn a **different function** than the one we are forced to use at inference. The model would also trivially copy the answer (“**on**”) from its own input and the loss would collapse.
- So, we deliberately **handicap training to match inference**: hide every token strictly to the right of the current position, so the model is trained on exactly the same inputs it will see when generating.

→ **We need to constrain attention to be causal.**

Masked (causal) self-attention

CAUSAL ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

$$V = HW_V$$

$$\text{ATTN}_M(Q, K, V) = \text{softmax}\left(\frac{QK^T + M}{\sqrt{d}}\right)V$$

TOKEN		d_1	d_2	$d_{12,288}$
2234: the	$h_1^T W_K$	-1.11	-2.01	...
5561: cat	$h_2^T W_K$	0.33	-1.52	...
4225: sat	$h_3^T W_K$	1.39	-2.02	...
5561: on	$h_4^T W_K$	-0.34	0.28	...
7162: it	$h_5^T W_K$	-1.03	2.87	...

Query (Q). What am I looking for?

Key (K). What do I contain?

$Q K^T$	$[W_K^T h_1,$ $[k_1$	k_2	$\dots$ k_3	k_4	$W_K^T h_5]$ $k_5]$	key vectors
$\begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix}$ query vectors	$\langle q_1, k_1 \rangle$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	
	$\langle q_2, k_1 \rangle$	$\langle q_2, k_2 \rangle$	$-\infty$	$-\infty$	$-\infty$	
	$\langle q_3, k_1 \rangle$	$\langle q_3, k_2 \rangle$	$\langle q_3, k_3 \rangle$	$\langle q_3, k_4 \rangle$	$\langle q_3, k_5 \rangle$	
	$\langle q_4, k_1 \rangle$	$\langle q_4, k_2 \rangle$	$\langle q_4, k_3 \rangle$	$\langle q_4, k_4 \rangle$	$\langle q_4, k_5 \rangle$	
	$\langle q_5, k_1 \rangle$	$\langle q_5, k_2 \rangle$	$\langle q_5, k_3 \rangle$	$\langle q_5, k_4 \rangle$	$\langle q_5, k_5 \rangle$	



Causal: Position 2 (cat) can only attend to positions 1 and 2 (the, cat).

Masked (causal) self-attention

$$\text{ATTN}(Q, K, V) = \text{softmax} \left(\frac{QK^T + M}{\sqrt{d}} \right) V,$$

$$M_{ij} = -\infty, \quad j > i, \\ M_{ij} = 0, \quad \text{else}$$

- Add a **mask M** to the score matrix **before** the softmax.
- Upper-triangular entries become $-\infty \rightarrow$ softmax sends them to **0**.
- Position i can only attend to positions **1, ..., i**.
- Shapes of Q, K, V and the **$O(T^2)$ cost** are unchanged — only the score matrix is masked.

	1-The	2-cat	3-sat	4-on	5-it
1-The	S	$-\infty$	$-\infty$	$-\infty$	$-\infty$
2-cat	S	S	$-\infty$	$-\infty$	$-\infty$
3-sat	S	S	S	$-\infty$	$-\infty$
4-on	S	S	S	S	$-\infty$
5-it	S	S	S	S	S

Score matrix S with causal mask M applied

$$S \rightarrow QK^T + M$$

$\langle q_1, k_1 \rangle$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$\langle q_2, k_1 \rangle$	$\langle q_2, k_2 \rangle$	$-\infty$	$-\infty$	$-\infty$
$\langle q_3, k_1 \rangle$	$\langle q_3, k_2 \rangle$	$\langle q_3, k_3 \rangle$	$-\infty$	$-\infty$
$\langle q_4, k_1 \rangle$	$\langle q_4, k_2 \rangle$	$\langle q_4, k_3 \rangle$	$\langle q_4, k_4 \rangle$	$-\infty$
$\langle q_5, k_1 \rangle$	$\langle q_5, k_2 \rangle$	$\langle q_5, k_3 \rangle$	$\langle q_5, k_4 \rangle$	$\langle q_5, k_5 \rangle$

Why upper-triangular? Convention vs. real structure

The convention part

The choice of “**upper triangle = future**” comes from how we lay out the score matrix on paper:

- Rows indexed top $\rightarrow$ bottom in **time order** (row 1 = first token, row T = last token)
- Columns indexed left $\rightarrow$ right in the **same time order** (col 1 = first token, col T = last token)
- Entry $S[i][j]$ = how much query i attends to key j

Given this layout, $j > i$ — keys that come after the query in time — lands in the **upper-right triangle**. Transpose the matrix, or read tokens right-to-left, and the masked half lands in a different corner. The math is identical; only the visual orientation changes.

The non-convention part

What is **not** convention is the **asymmetry itself**. Causality treats the two indices differently:

- The **query index i** is “the position currently making a prediction”
- The **key/value index j** is “the position whose information might be used”
- The constraint $j \leq i$ is a statement about **time**, not about matrix geometry

So the **existence** of a triangular mask is a real semantic property of the architecture — information flows forward in time only. Drawing it as the upper-right corner is just the same left-to-right, top-to-bottom convention that makes English read that way and timelines run that way in textbooks.

In code

Every framework (PyTorch, JAX, TF) builds the mask as `torch.triu(ones, diagonal=1) * -inf` — “upper triangle above the diagonal”. Standard enough that papers and library APIs all agree, but a **notational standard**, not a mathematical necessity.

When is the mask applied? Training vs. inference

Training (teacher forcing)

- All T tokens fed in **parallel**.
- Mask prevents position i from **cheating** by looking at the ground-truth token $i+1$ in its own input.
- Without it, the loss would **collapse to zero** — the model just learns the identity.

Inference — prompt prefill

- All T prompt tokens pushed through in **one pass**.
- Mask still required so every position is **causally consistent** with how the model was trained.
- Removing it would leak future prompt context into earlier positions — **out-of-distribution**.

Inference — decoding

- Generation is **autoregressive**: at step $t+1$ the model has produced tokens $1, \dots, t$ and must predict the next.
- The new query (length 1) attends over positions $1, \dots, t$ — exactly **the past**.
- Causality is **the same constraint** as in training and prefill — the mask is part of the architecture and applied in every forward pass. The only difference at decoding is that there are no future tokens *in this particular input*, so the upper triangle of the new row is empty.

Giving meaning to Attention – Keys, Queries and Values

ATTENTION

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$

$$K = HW_K$$

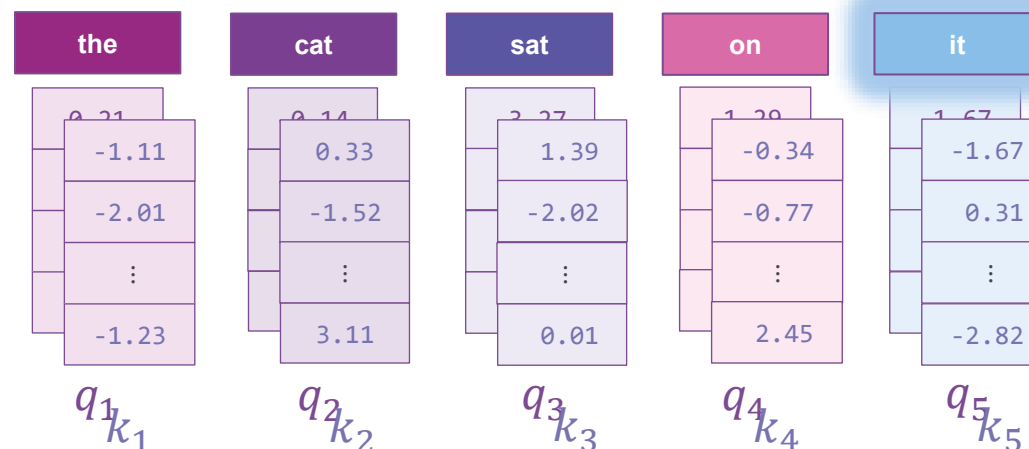
$$V = HW_V$$

DETERMINES RELEVANCE

$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$Q K^T$	$[W_K^T h_1,$	$\dots$	$W_K^T h_5]$		
	$[k_1$	k_2	k_3	k_4	$k_5]$ key vectors
$\begin{bmatrix} h_1^T W_Q \\ \vdots \\ h_5^T W_Q \end{bmatrix} = \begin{bmatrix} q_1^T \\ q_2^T \\ q_3^T \\ q_4^T \\ q_5^T \end{bmatrix}$	$\begin{bmatrix} \langle q_1, k_1 \rangle \\ \langle q_2, k_1 \rangle \\ \langle q_3, k_1 \rangle \\ \langle q_4, k_1 \rangle \\ \langle q_5, k_1 \rangle \end{bmatrix}$	$\begin{bmatrix} \langle q_1, k_2 \rangle \\ \langle q_2, k_2 \rangle \\ \langle q_3, k_2 \rangle \\ \langle q_4, k_2 \rangle \\ \langle q_5, k_2 \rangle \end{bmatrix}$	$\begin{bmatrix} \langle q_1, k_3 \rangle \\ \langle q_2, k_3 \rangle \\ \langle q_3, k_3 \rangle \\ \langle q_4, k_3 \rangle \\ \langle q_5, k_3 \rangle \end{bmatrix}$	$\begin{bmatrix} \langle q_1, k_4 \rangle \\ \langle q_2, k_4 \rangle \\ \langle q_3, k_4 \rangle \\ \langle q_4, k_4 \rangle \\ \langle q_5, k_4 \rangle \end{bmatrix}$	$\begin{bmatrix} \langle q_1, k_5 \rangle \\ \langle q_2, k_5 \rangle \\ \langle q_3, k_5 \rangle \\ \langle q_4, k_5 \rangle \\ \langle q_5, k_5 \rangle \end{bmatrix}$
query vectors					

For query **it**, let's look at the complete context and what will be attended the most too.



Self-Attention: Scores, Scaling and Softmax

For query **it**, let's look at the complete context and what will be attended the most too.

Example $\langle q_{it}, k_{cat} \rangle = 12.8$

Fifth row of QK^T — raw scores for query it

the	cat	sat	on	it
3.2	12.8	5.6	2.4	7.2

highest → cat

Promising — but raw scores are not weights yet: unbounded, can be negative, do not sum to anything meaningful. Two more steps polish them.

Self-Attention: Scaling by $\sqrt{d}$

Before turning the scores into weights, divide by $\sqrt{d}$. With $d = 64$, $\sqrt{d} = 8$.

$$S' = \frac{QK^T}{\sqrt{d}}$$

	the	cat	sat	on	it
raw (S)	3.2	12.8	5.6	2.4	7.2
	$\div \sqrt{d}$				
scaled (S')	0.40	1.60	0.70	0.30	0.90

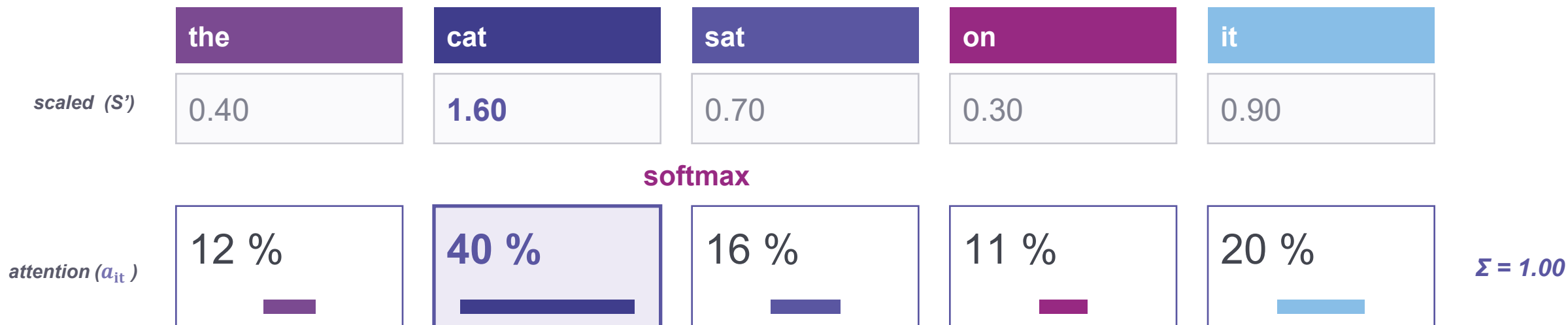
Ranking preserved. cat is still the highest — same factor on every score.

Spread shrinks. max-min: 10.4 $\rightarrow$ 1.3. The maximum range should be kept as small as possible for numerical stability.

Self-Attention – with Queries and Keys determine Relevance

Five reals are not weights. Apply softmax: exponentiate (everything positive, big numbers get disproportionately bigger), then normalise so the total is 1.

$$A = \text{softmax} \left(\frac{QK^T}{\sqrt{d}} \right)$$



No rules, no dictionary, no parse tree. Just dot products and softmax — and the model puts 40 % of it's attention onto cat, the largest share of any word.

Self-Attention – with Queries and Keys determine Relevance

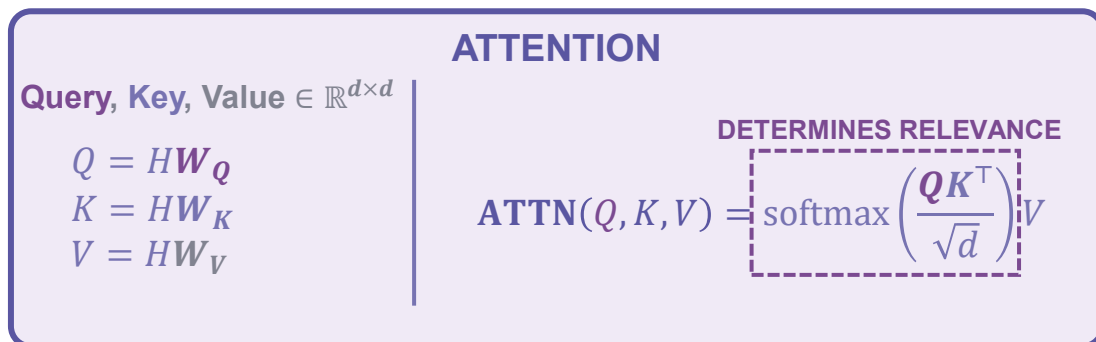
Five reals are not weights. Apply softmax: exponentiate (everything positive, big numbers get disproportionately bigger), then normalize so the total is 1.

$$A = \text{softmax} \left(\frac{QK^T}{\sqrt{d}} \right)$$

the cat sat on it

$Q K^T$	$[k_1$	k_2	k_3	k_4	$k_5]$	$\text{softmax}\left(\frac{Q K^T}{\sqrt{d}}\right)$	$[$	the	cat	sat	on	it]
$\begin{bmatrix} q_1^\top \\ q_2^\top \\ q_3^\top \\ q_4^\top \\ q_5^\top \end{bmatrix}$	$\begin{bmatrix} \langle q_1, k_1 \rangle \\ \langle q_2, k_1 \rangle \\ \langle q_3, k_1 \rangle \\ \langle q_4, k_1 \rangle \\ \langle q_5, k_1 \rangle \end{bmatrix}$	$\begin{bmatrix} \langle q_1, k_2 \rangle \\ \langle q_2, k_2 \rangle \\ \langle q_3, k_2 \rangle \\ \langle q_4, k_2 \rangle \\ \langle q_5, k_2 \rangle \end{bmatrix}$	$\begin{bmatrix} \langle q_1, k_3 \rangle \\ \langle q_2, k_3 \rangle \\ \langle q_3, k_3 \rangle \\ \langle q_4, k_3 \rangle \\ \langle q_5, k_3 \rangle \end{bmatrix}$	$\begin{bmatrix} \langle q_1, k_4 \rangle \\ \langle q_2, k_4 \rangle \\ \langle q_3, k_4 \rangle \\ \langle q_4, k_4 \rangle \\ \langle q_5, k_4 \rangle \end{bmatrix}$	$\begin{bmatrix} \langle q_1, k_5 \rangle \\ \langle q_2, k_5 \rangle \\ \langle q_3, k_5 \rangle \\ \langle q_4, k_5 \rangle \\ \langle q_5, k_5 \rangle \end{bmatrix}$	$\begin{bmatrix} \text{the} \\ \text{cat} \\ \text{sat} \\ \text{on} \\ \text{it} \end{bmatrix}$	$\begin{bmatrix} 0.39 & 0.18 & 0.16 & 0.13 & 0.14 \\ 0.16 & 0.36 & 0.24 & 0.11 & 0.12 \\ 0.12 & 0.29 & 0.32 & 0.17 & 0.11 \\ 0.12 & 0.14 & 0.26 & 0.31 & 0.17 \\ 0.12 & 0.40 & 0.16 & 0.11 & 0.20 \end{bmatrix}$					

Self-Attention – with Queries and Keys determine Relevance

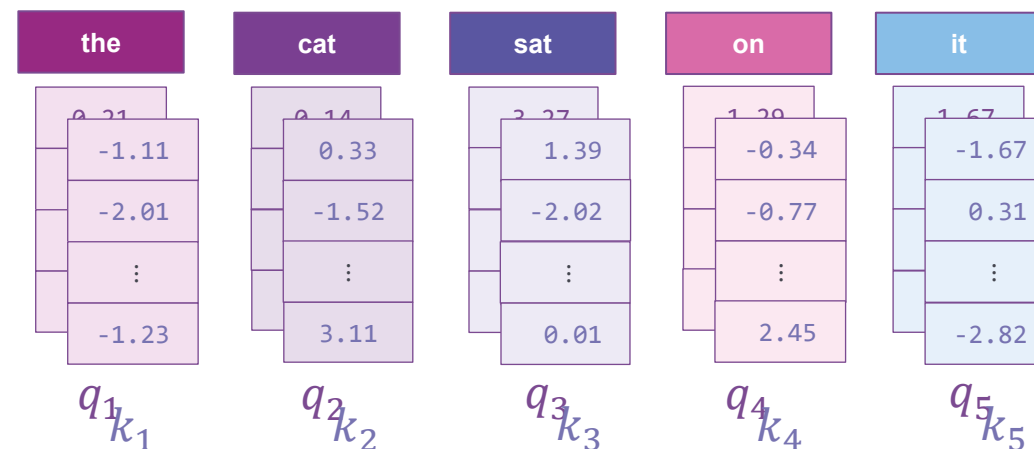


Each **token** needs to understand the **context** in which it is surrounded.

Query (Q). *What am I looking for?* — the question a token is asking about its context.

Key (K). *What do I contain?* — the advertisement a token publishes so that others can find it.

The **attention matrix A** contains how much each token listens to every other — one distribution per row.

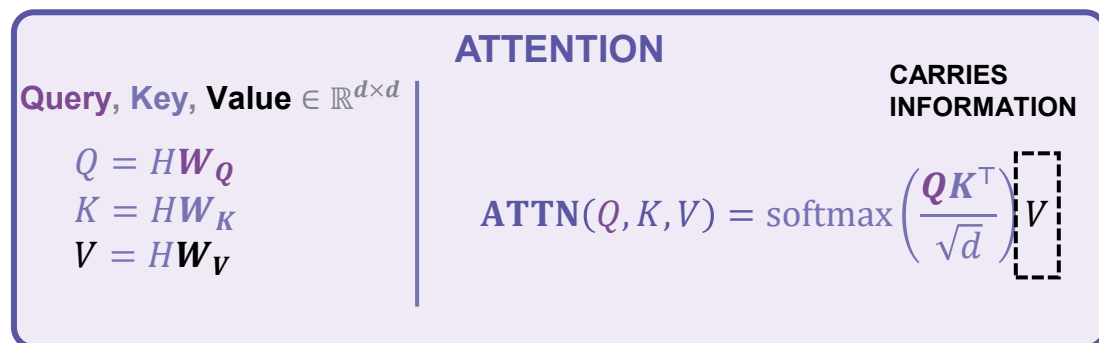


↓

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)$$

	the	cat	sat	on	it
the	0.39	0.18	0.16	0.13	0.14
cat	0.16	0.36	0.24	0.11	0.12
sat	0.12	0.29	0.32	0.17	0.11
on	0.12	0.14	0.26	0.31	0.17
it	0.12	0.40	0.16	0.11	0.20

Self-Attention – with Values we carry Information



$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)$$

	[the	cat	sat	on	it]
[the	0.39	0.18	0.16	0.13	0.14
[cat	0.16	0.36	0.24	0.11	0.12
[sat	0.12	0.29	0.32	0.17	0.11
[on	0.12	0.14	0.26	0.31	0.17
[it	0.12	0.40	0.16	0.11	0.20

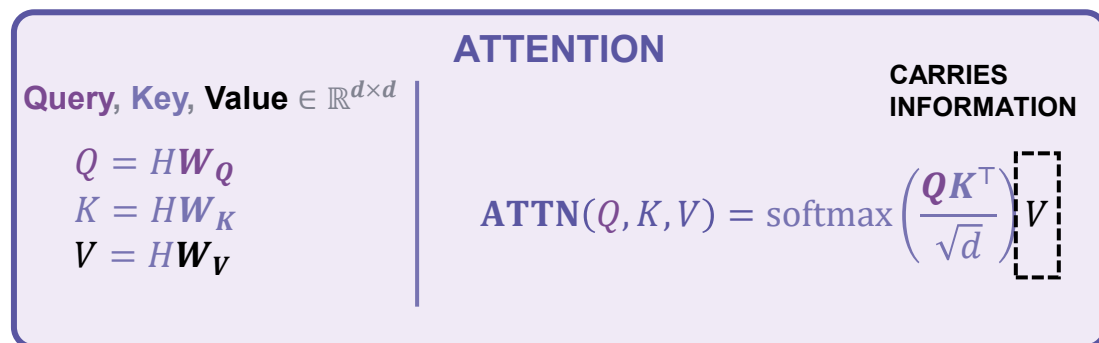
Value (V). *What should I transmit, if I am selected?* — the actual payload of information the token offers when another token has decided to listen to it.

Unpacking $Z = AV$ row by row. To see why each output row z_i is a convex combination of value vectors, expand the matrix product one row at a time. By the definition of matrix multiplication,

$$(AV)[i:] = \sum_{j=1}^T A[i,j] \cdot V[j:] = \sum_{j=1}^T A[i,j] \cdot v_j,$$

The attention matrix A contains how much each token listens to every other — one distribution per row.

Self-Attention – with Values we carry Information



Value (V). *What should I transmit, if I am selected?* — the actual payload of information the token offers when another token has decided to listen to it.

Unpacking $Z = AV$ row by row. Expand the product one row at a time. By the definition of matrix multiplication,

$$(AV)[i:] = \sum_{j=1}^T A[ij] \cdot V[j:] = \sum_{j=1}^T A[ij] \cdot v_j,$$

the	cat	sat	on	it
0.21	-0.14	3.27	-1.29	-1.67
-0.14	-0.22	-0.01	-0.77	0.31
⋮	⋮	⋮	⋮	⋮
-0.12	2.35	-0.81	2.45	-2.82
v_1	v_2	v_3	v_4	v_5

Because $A[i:]$ came out of a row-wise softmax, its entries are non-negative and sum to exactly 1. So, each output row z_i lies in the convex hull of $\{v_1, \dots, v_T\}$ — a weighted average of the T value vectors, with weights read off the i -th row of the attention matrix.

A note on conventions. Math texts treat the value vector of token i as a column: $v_i = W_V^T h_i$, with $h_i, v_i \in \mathbb{R}^d$ as columns. Here we use row-major layout (matching PyTorch): h_i is the i -th row of H , v_i is the i -th row of V , and the projection becomes the right-multiplication $v_i = h_i W_V$ with no transposes. Same linear map, different bookkeeping.

Self-Attention – with Values we carry Information

Query, Key, Value $\in \mathbb{R}^{d \times d}$

$$Q = HW_Q$$
$$K = HW_K$$
$$V = HW_V$$

ATTENTION

$$\text{ATTN}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) \boxed{V}$$

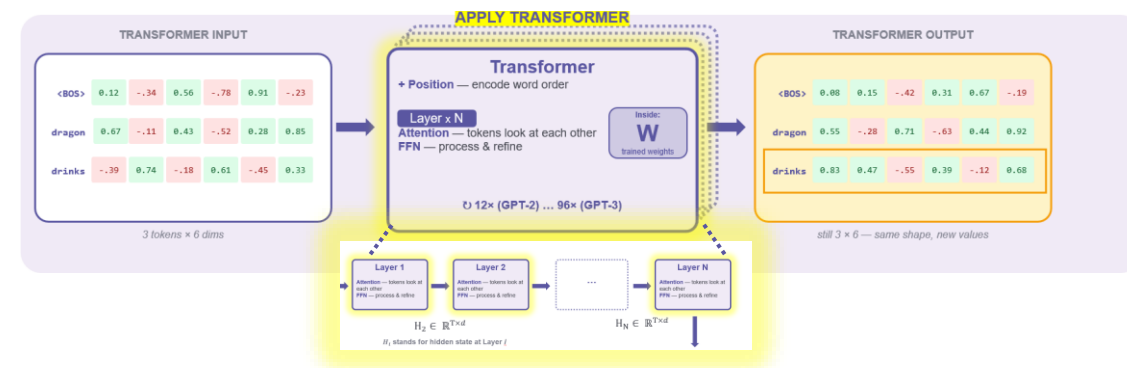
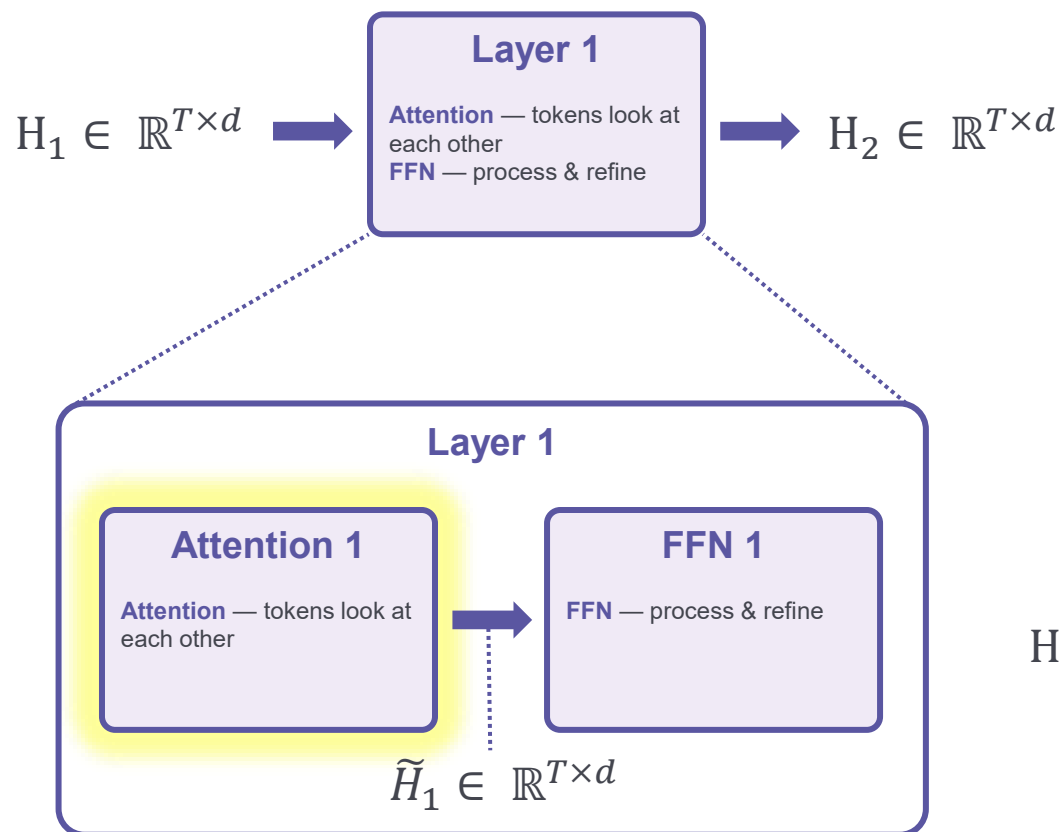
CARRIES
INFORMATION

Value (V). *What should I transmit, if I am selected?* — the actual payload of information the token offers when another token has decided to listen to it.

In the library analogy: you walk in with a **question** (**query**). Every book on the shelf has a **title** (**key**). You match your question against each title to decide which books are relevant. Then you *read* the **content** (**value**) of the most relevant books and summarize. Three different artefacts, each doing a different job.

Notice the careful separation: query and key together determine **who listens to whom**; value determines **what actually gets sent**. A token can advertise its relevance for one reason (its key) and carry useful information for another (its value). That decoupling is what gives attention its expressive power.

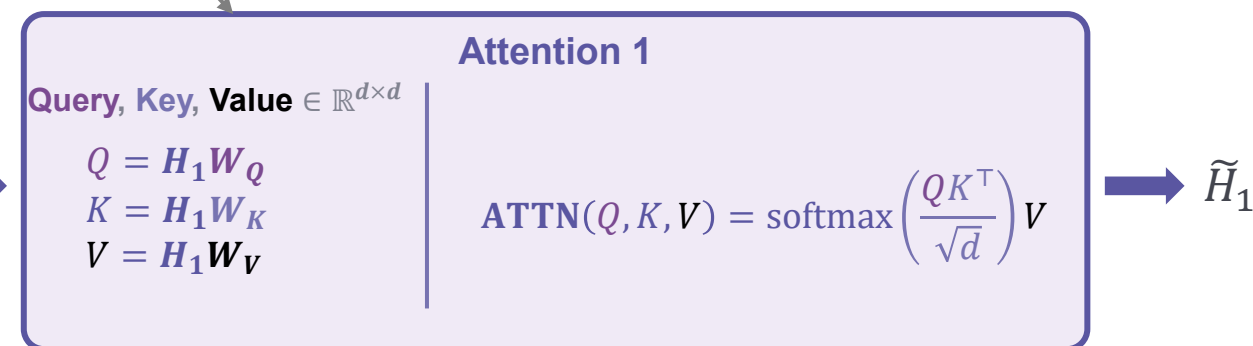
Callback: Inside the Layers



Parameter: $3 \times d^2$

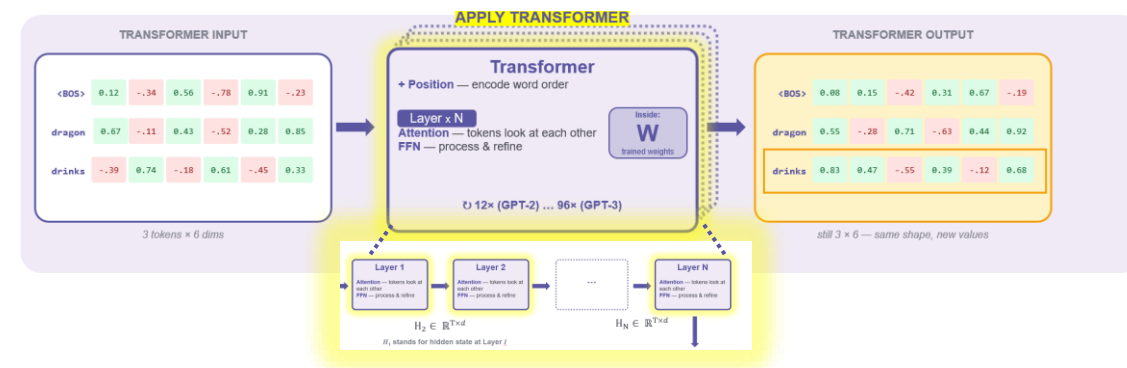
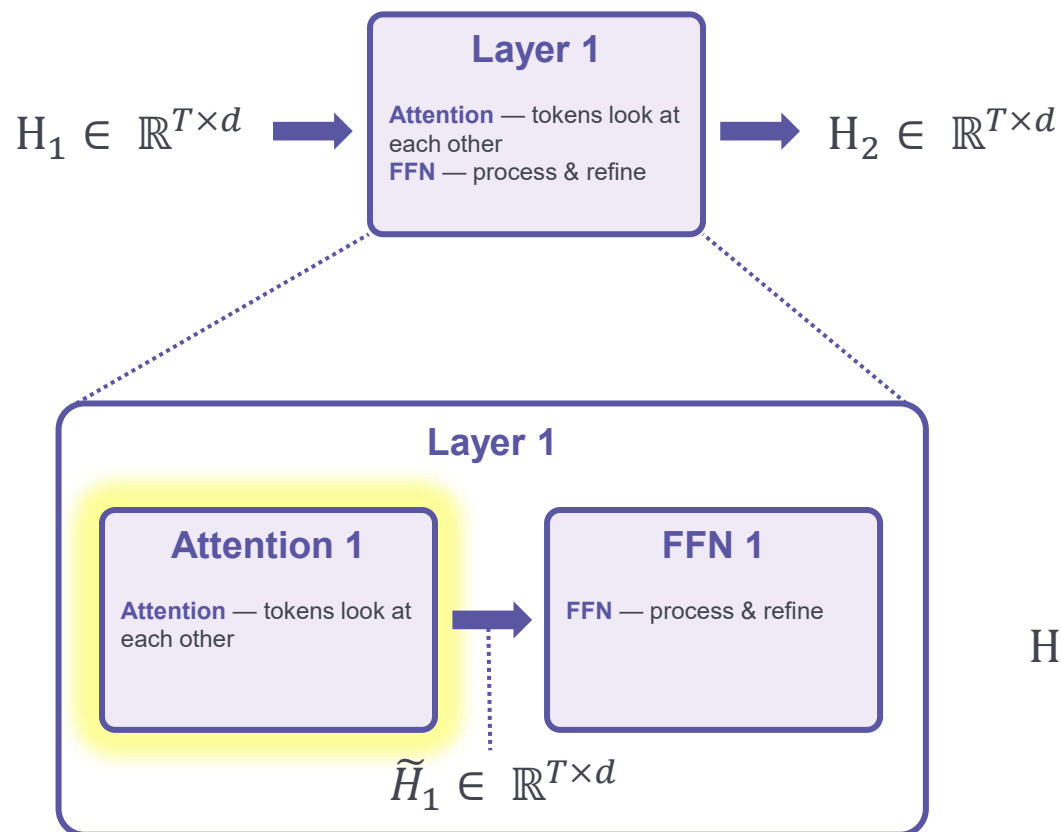
$$W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$$

$$H_1 \rightarrow$$



The **attention block** is rich in meaning, simple in math — one formula does it all.

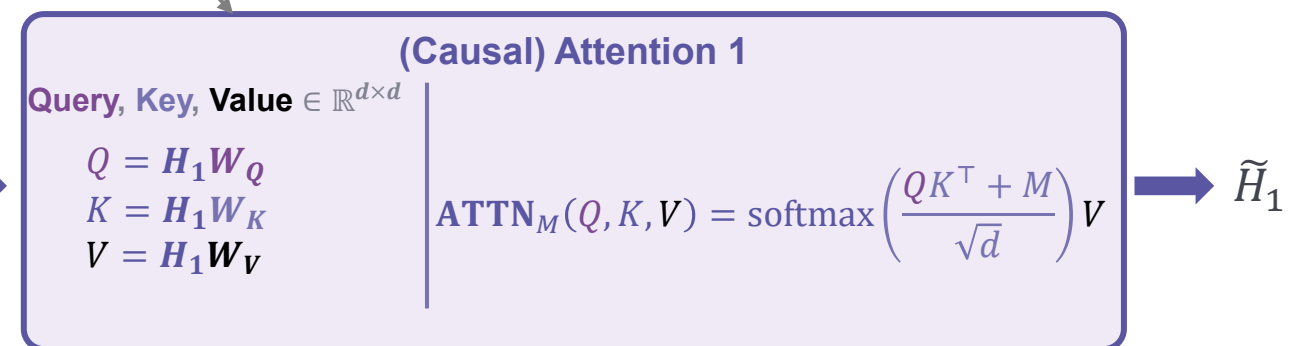
Callback: Inside the Layers



Parameter: $3 \times d^2$

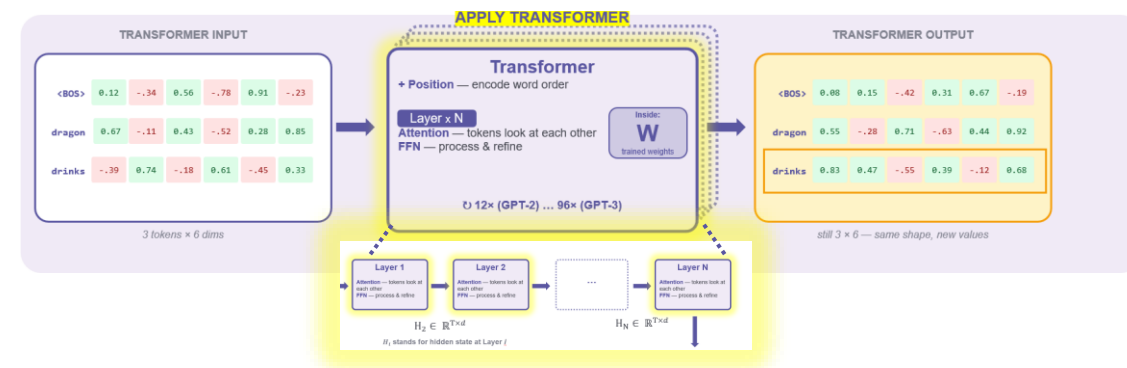
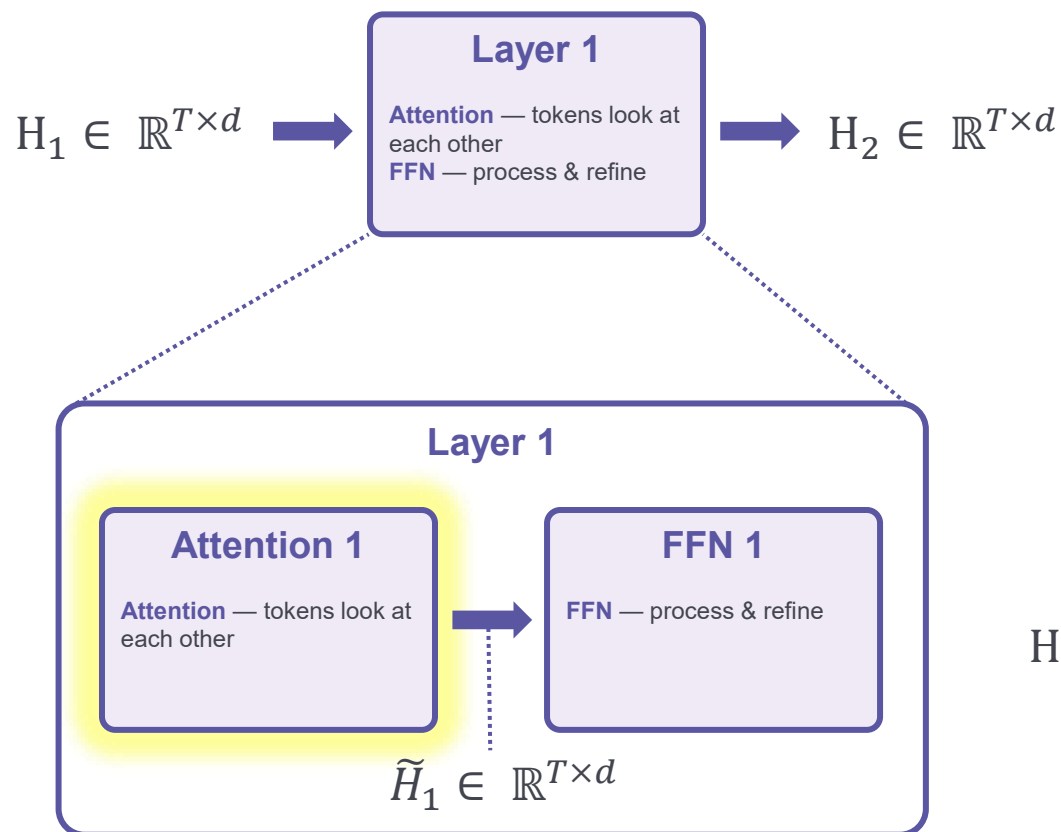
$$W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$$

$$H_1 \rightarrow$$



The **attention block** is rich in meaning, simple in math — one formula does it all.

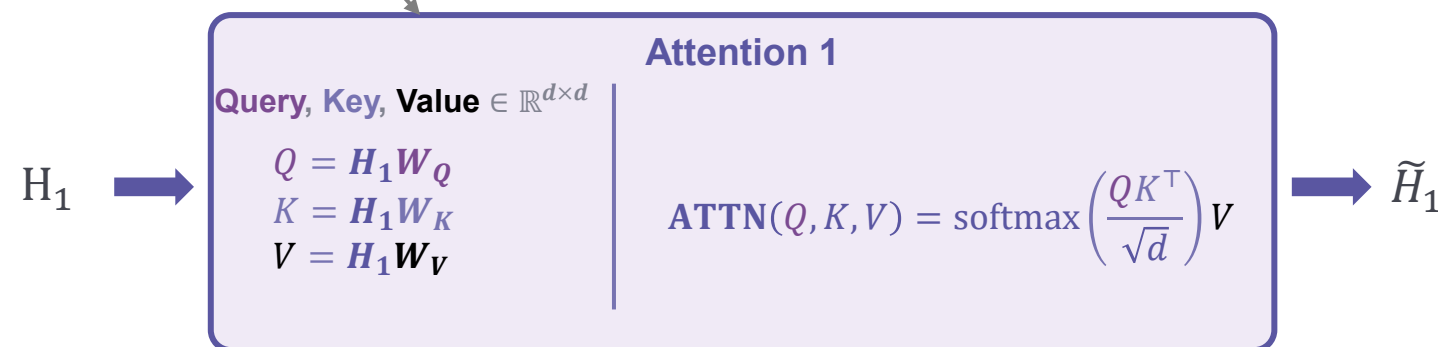
Callback: Inside the Layers



Parameter: $3 \times d^2$

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

For large embedding dimensions this becomes quite huge, e. g. $d = 12\,288 \approx 10^5$ becomes $3 \cdot 10^{10}$



The **attention block** is rich in meaning, simple in math — one formula does it all.

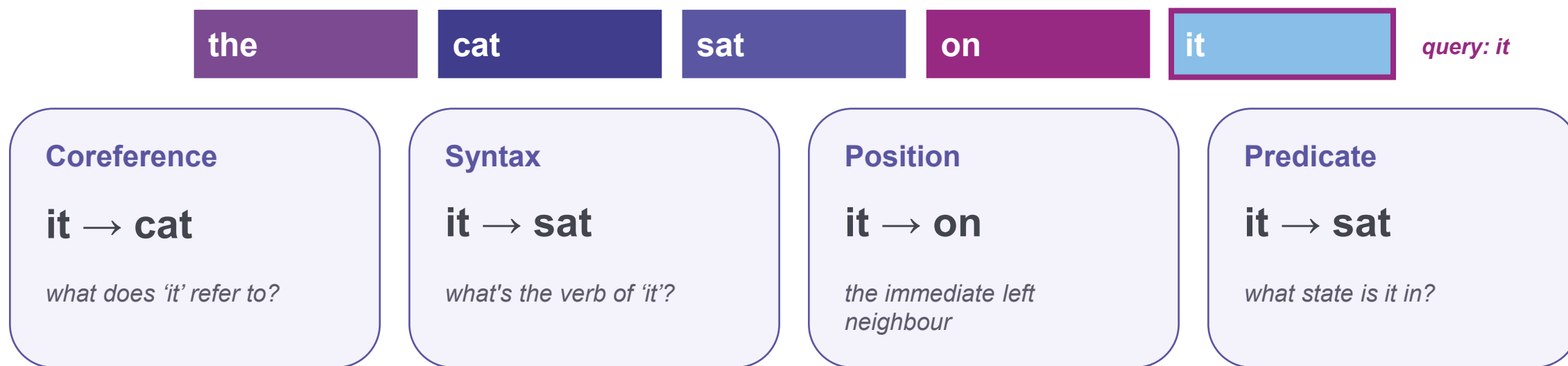
Multi-Head Attention

8

One Head, One Perspective

Self-attention answers one question per token: “which other tokens are most relevant to me?”

But “relevant” has many flavours.



A single head (i.e., a set of query, key matrices) can only learn one of these.
Force it to choose, and the others get dropped on the floor.

And we pay quite a high price in term of parameters already, e. g. $d = 10^5$
becomes $3 \cdot 10^{10}$

From One Head to Many — Multi-Head Attention

Multi-head attention (MHA) is **not a fundamentally new mechanism**. It is the simple attention from above in parallel on *disjoint slices of the embedding space*, with a final mixing projection that recomposes the slices back into a d -dimensional output.

Three structural changes from single-head

1. **Slice the embedding space** into *heads* chunks of size $d_h = d/\text{heads}$.
2. **Per-head Q/K/V projections**, each of shape $d \times d_h$.
3. **New output projection** $W_O \in \mathbb{R}^{d \times d}$ that mixes head outputs.

Concretely (GPT-2 small): $d = 768$, $\text{heads} = 12$, $d_h = 64$.

Embedding dimension d											
d_h	d_h	d_h	d_h	d_h	d_h	d_h	d_h	d_h	d_h	d_h	d_h

$$d = 768, \quad d = \text{heads} \cdot d_h$$

$$d_h = 64, \quad \text{heads} = 12$$

Multi-Head Attention: Running heads in Parallel

GPT-2 small:

Embedding dimension d										
d_h	d_h	d_h	d_h	d_h	d_h	d_h	d_h	d_h	d_h	d_h

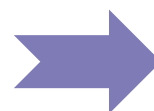
$d = 768$

$d_h = 64, \text{ heads} = 12$

Single-Head Attention

Parameter: $3 \times d^2$

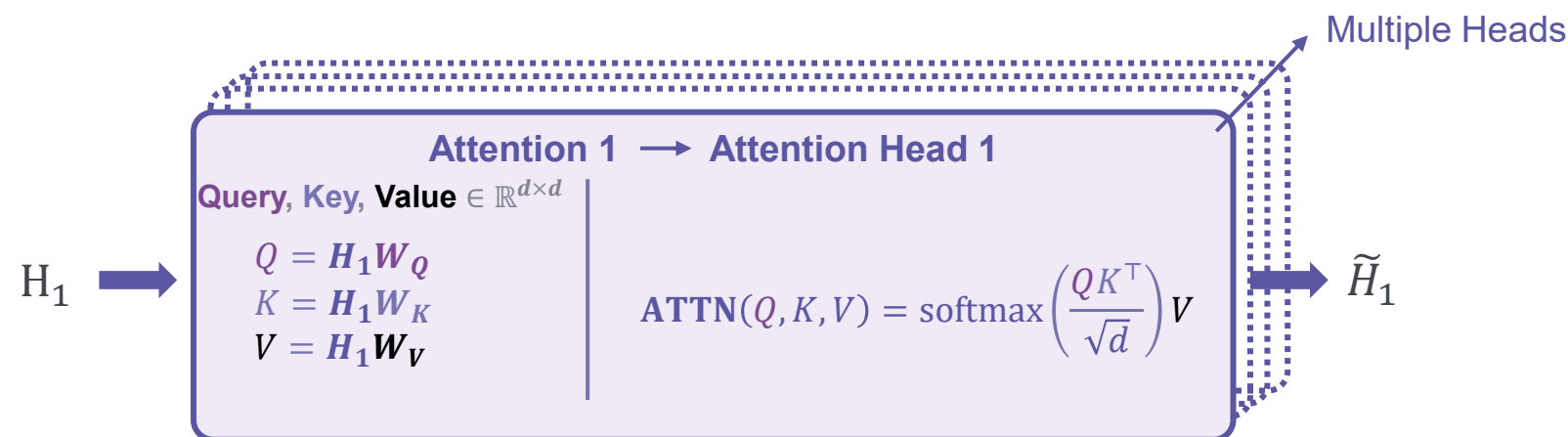
$W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$



Multi-Head Attention

Parameter: $3 \times \text{heads} \times d \times d_h$

$W_Q^1, W_K^1, W_V^1, \quad W_Q^2, W_K^2, W_V^2, \quad \dots, \quad W_Q^{\text{heads}}, W_K^{\text{heads}}, W_V^{\text{heads}} \in \mathbb{R}^{d \times d_h}$



Multi-Head Attention: Running heads in Parallel

Single-Head Attention

Parameter: $3 \times d^2$

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

Multi-Head Attention

Parameter: $3 \times \text{heads} \times d_h \times d + d^2$

$$W_Q^1, W_K^1, W_V^1, \quad W_Q^2, W_K^2, W_V^2, \quad \dots, \quad W_Q^{\text{heads}}, W_K^{\text{heads}}, W_V^{\text{heads}} \in \mathbb{R}^{d \times d_h}$$

Attention Head 1

Query, Key, Value $\in \mathbb{R}^{d \times d_h}$

$$Q^1 = H_1 W_Q^1$$

$$K^1 = H_1 W_K^1$$

$$V^1 = H_1 W_V^1$$

$$\text{ATTN}(Q^1, K^1, V^1) = \text{softmax}\left(\frac{Q^1 K^{1\top}}{\sqrt{d}}\right) V^1$$

$$d = \text{heads} \cdot d_h$$

$$H_1^1 \in \mathbb{R}^{T \times d_h}$$

heads = 12
many
matrices

$$H_1^{12} \in \mathbb{R}^{T \times d_h}$$

concatenate

$$[H_1^1 | \dots | H_1^{12}]$$

$$\in \mathbb{R}^{T \times d}$$

New output projection

$$W_O \in \mathbb{R}^{d \times d}$$

that **mixes** head outputs.

$$\tilde{H}_1 = [H_1^1 | \dots | H_1^{12}] W_O$$

$$\mathbb{R}^{T \times d}$$

Multi-Head Attention: Running heads in Parallel

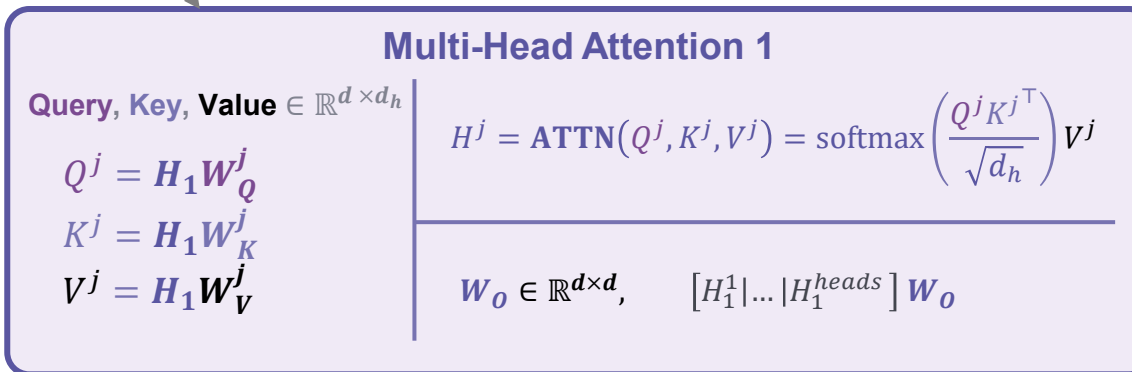
Multi-Head Attention

Parameter: $3 \times \text{heads} \times d_h \times d + d^2 = 4 d^2$

$$W_Q^1, W_K^1, W_V^1, \quad W_Q^2, W_K^2, W_V^2, \quad \dots, \quad W_Q^{\text{heads}}, W_K^{\text{heads}}, W_V^{\text{heads}} \in \mathbb{R}^{d \times d_h}, \quad W_O \in \mathbb{R}^{d \times d}$$

The number of heads of the Layer is a new **Hyperparameter**

$$H_1 \in \mathbb{R}^{T \times d}$$



$$\tilde{H}_1 \in \mathbb{R}^{T \times d}$$

Slicing and Per-Head Projections

Every head $j = 1, \dots, \text{heads}$ gets its own *narrower* projection matrices:

$$W_Q^j, W_K^j, W_V^j \in \mathbb{R}^{d \times d_h}.$$

Q/K/V parameter budget is unchanged

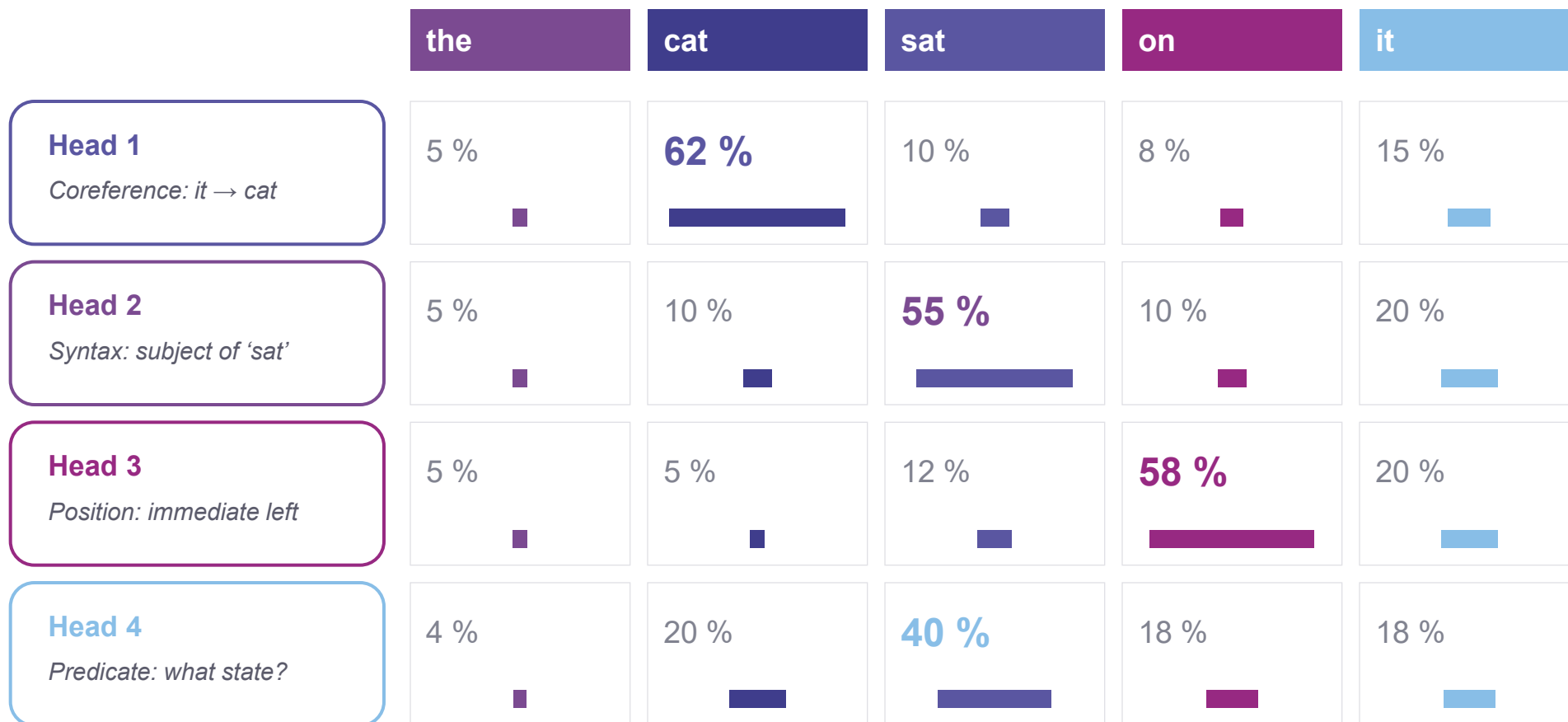
$$3 \cdot \text{heads} \cdot d \cdot d_h = 3 \cdot \text{heads} \cdot d \cdot (d/\text{heads}) = 3d^2.$$

Multi-head is **not** “more expensive per projection” — it is the *same* parameter budget rearranged into H parallel subspaces.

The scaling denominator changes from $\sqrt{d_h}$ already used per head — each head now works in a d_h -dimensional subspace, not the full d .

Four Heads, Four Perspectives — Same Sentence, Same Query

Query = it. Each head produces its own attention distribution — below are four plausible specializations.

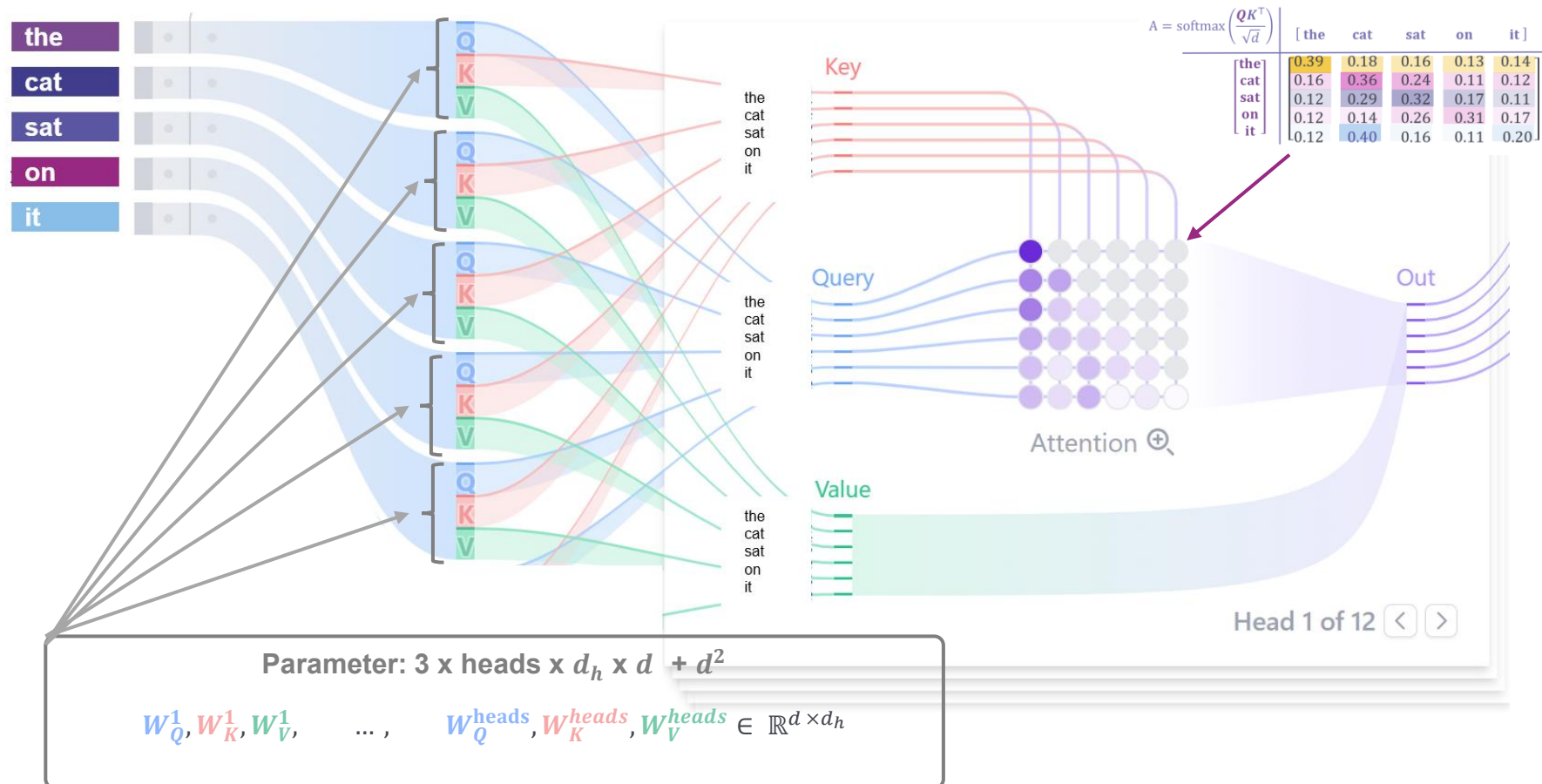


Same input, same query — four distinct attention patterns.

W_O reads from all four and decides how much each contributes to the output.

Multi-Head Attention: Interactive visualization can help

Transformer Explainer: LLM Transformer Model Visually Explained



What Heads Actually Learn — Findings From Interpretability

When researchers probe trained models, they find heads that specialize in surprisingly clean patterns. A small selection:

Previous-token

attends almost entirely to position $i-1$.

Builds local context: bigrams, prefixes.

Coreference

links pronouns to their antecedents.

“it” → “cat”, “she” → a person mentioned earlier.

Bracket-matching

pairs (with), { with }, opening tag with closing.

Critical for code, JSON, math expressions.

Syntactic-role

verbs attend to their subjects, adjectives to nouns.

Recovers grammar without explicit supervision.

Positional

fixed offsets: “two tokens back”, “start of sentence”.

Provides absolute positional anchors.

Induction

if [A B] appeared earlier, copy B after seeing A again.

Backbone of in-context learning.

Nobody programmed these patterns. Specialization emerges from training — each head finds a pattern that helps reduce next-token loss.

Why This Matters: Heads as Routing for Code

Code is rich in long-range structure: matching brackets, scope, variable bindings. Each is a routing pattern — ideal work for a head.

Example

```
def total_price(items, tax_rate):  
    subtotal = sum(item.price for item in items)  
    tax = subtotal * tax_rate  
    return subtotal + tax
```

Each highlighted relationship below is a different head's job.

tax_rate → **tax_rate**

variable binding (def → use)

subtotal → **subtotal**

scope: respect the def line

(→)

bracket matching

4 spaces → **4 spaces**

indentation, block structure

items, tax_rate → ...

function-signature awareness

One head per pattern, in parallel. This is why code models work: many simple routing rules combine into structural understanding.

Multi-Head Attention: The Real Layer in Use

Multi-Head Attention

Parameter: $3 \times \text{heads} \times d_h \times d + d^2 = 4 d^2$

$$W_Q^1, W_K^1, W_V^1, \quad W_Q^2, W_K^2, W_V^2, \quad \dots, \quad W_Q^{\text{heads}}, W_K^{\text{heads}}, W_V^{\text{heads}} \in \mathbb{R}^{d \times d_h}, \quad W_O \in \mathbb{R}^{d \times d}$$

The number of heads of the Layer is a new **Hyperparameter**

Multi-Head Attention 1

Query, Key, Value $\in \mathbb{R}^{d \times d_h}$

$$Q^j = H_1 W_Q^j$$

$$K^j = H_1 W_K^j$$

$$V^j = H_1 W_V^j$$

$$H_1^j = \text{ATTN}(Q^j, K^j, V^j) = \text{softmax}\left(\frac{Q^j K^{j\top}}{\sqrt{d_h}}\right) V^j$$

$$W_O \in \mathbb{R}^{d \times d}, \quad [H_1^1 | \dots | H_1^{\text{heads}}] W_O$$

$$H_1 \in \mathbb{R}^{T \times d}$$

$$\tilde{H}_1 \in \mathbb{R}^{T \times d}$$

Multi-Head Attention: The Real Layer in Use

Multi-Head Attention

Parameter: $3 \times \text{heads} \times d_h \times d + d^2 = 4 d^2$

$$W_Q^1, W_K^1, W_V^1, \quad W_Q^2, W_K^2, W_V^2, \quad \dots, \quad W_Q^{\text{heads}}, W_K^{\text{heads}}, W_V^{\text{heads}} \in \mathbb{R}^{d \times d_h}, \quad W_O \in \mathbb{R}^{d \times d}$$

The number of heads of the Layer is a new **Hyperparameter**

$$H_1 \in \mathbb{R}^{T \times d}$$



(Causal) Multi-Head Attention 1

Query, Key, Value $\in \mathbb{R}^{d \times d_h}$

$$Q^j = H_1 W_Q^j$$

$$K^j = H_1 W_K^j$$

$$V^j = H_1 W_V^j$$

$$H_1^j = \text{ATTN}_M(Q^j, K^j, V^j) = \text{softmax}\left(\frac{Q^j K^{j\top} + M}{\sqrt{d_h}}\right) V^j$$

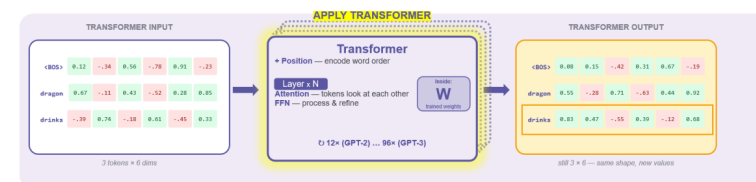
$$W_O \in \mathbb{R}^{d \times d}, \quad [H_1^1 | \dots | H_1^{\text{heads}}] W_O$$

$$\tilde{H}_1 \in \mathbb{R}^{T \times d}$$

Positional Encoding

9

Understanding Positional Encoding



TRANSFORMER INPUT

$$X_0 \in \mathbb{R}^{T \times d}$$

+ Position —
encode word
order

$$H_1 \in \mathbb{R}^{T \times d}$$

We skip **positional encoding** for now and clarify this step in detail later

Now it is time to clarify why **positional encoding** is so important for the next token prediction task

Layer 1

Attention — tokens look at each other
FFN — process & refine

Layer 2

Attention — tokens look at each other
FFN — process & refine

...

Layer N

Attention — tokens look at each other
FFN — process & refine

$$H_2 \in \mathbb{R}^{T \times d}$$

H_i stands for hidden state at Layer i

$$H_N \in \mathbb{R}^{T \times d}$$

TRANSFORMER OUTPUT

$$Z \in \mathbb{R}^{T \times d}$$

The Blind Spot — Attention Sees a Bag of Words

Take the sentence `the cat sat on the mat` and scramble the word order,
e. g. `the cat mat on the sat`.

What changes in attention?

Nothing structurally. The set of pairwise scores $q_i^\top k_j$ is *identical*, just relabeled. Each score depends on **which two tokens** are involved — never on **where they sit**.

- Attention's input is, mathematically, an *unordered set* of token vectors.
- Without help, the transformer is just a very expensive bag-of-words model.

We will now make this precise as a lemma — and then fix it.

Permutation Matrices — A One-Slide Refresher

A permutation matrix $P \in \{0,1\}^{T \times T}$ has exactly one 1 per row and per column.

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \quad P \begin{pmatrix} x_1^\top \\ x_2^\top \\ x_3^\top \end{pmatrix} = \begin{pmatrix} x_2^\top \\ x_3^\top \\ x_1^\top \end{pmatrix}$$

Two facts we need

- PX permutes the rows of X .
- $P^\top P = I$ (permutation matrices are orthogonal).

These are the only two facts the proof on the next slide will use.

Self-Attention is Permutation Equivariant

Attention as a function. Self-attention is the function

$$\text{Attention}: \mathbb{R}^{T \times d} \rightarrow \mathbb{R}^{T \times d}$$

parametrized by fixed matrices $W_Q, W_K, W_V \in \mathbb{R}^{T \times d}$, defined by

$$\text{Attention}(X) = \text{softmax}\left(\frac{(XW_Q)(XW_K)^T}{\sqrt{d_k}}\right)XW_V.$$

Lemma (permutation equivariance)

For every permutation matrix $P \in \{0,1\}^{T \times T}$ and every $X \in \mathbb{R}^{T \times d}$,

$$\text{Attention}(PX) = P \text{Attention}(X) \in \mathbb{R}^{T \times d}.$$

Doesn't the causal mask already encode order?

The causal mask says: at position i , only attend to **positions 1, ..., i** . So, order matters now — right? Not quite. The mask restricts **which set** of past tokens is visible — not **their order within that set**.

A pair of sequences with the same multiset

- **A:** the cat sat on the mat **B:** the mat sat on the cat

Predict the **7th token**. The query at position 6 attends to its causal prefix:

- For **A**: {the, cat, sat, on, the, mat}
- For **B**: {the, mat, sat, on, the, cat} — **identical multiset**.

What the model sees — and what it misses

- The query token differs (**mat** vs **cat**), so outputs differ — but only because the **query identity** changed, not because the model can tell its past apart.
 - The model **cannot distinguish** "cat sat on mat" from "mat sat on cat" inside its context. Both look like the same bag.
- **Causal mask gives "only the past" — not "the past in order". Positional encodings still required.**

Causal Attention — A Weaker Equivariance

The identity

For causal attention with **mask** M :

$$\text{Attention}_M(PX) = P \text{Attention}_M(X) \text{ iff } PMP^T = M$$

— i.e. equivariance survives **only for permutations that commute with the mask**.

Consequence — equivariance collapses to the identity

- M is strictly lower-triangular; PMP^T reorders both rows and columns by the same perturbation.
- The only such perturbation is the **identity** — so full-sequence permutation equivariance is **broken**, as expected: the mask is order-aware.

But a weaker symmetry survives — prefix invariance

Lemma (causal prefix invariance). For each $i = 1, \dots, T$ and every prefix permutation $P_i = \text{diag}(Q, I_{T-i})$ with $Q \in \{0, 1\}^{T \times T}$ a permutation matrix:

$$\mathbf{e}_i^T \cdot \text{Attention}_M(P_i X) = \mathbf{e}_i^T \cdot \text{Attention}_M(X)$$

What Equivariance Means for Position

Each output row is a function only of which tokens are present, never of their order.

Token cat's enriched representation

Identical content whether the sequence is

the cat sat, sat the cat, cat sat the

— only its row index in the output matrix changes.

- Each token's output depends only on the multiset of other tokens.
- Layer-Norm and FFN are position-wise, so the whole block is permutation equivariant.
- Stacking blocks does not fix this — equivariance composes.

Consequence: with no extra signal, the model literally cannot tell “John loves Mary” from “Mary loves John”

The Fix — Inject Position Into the Tokens

We don't want to change attention itself — equivariance is what gives us parallelism, length generalization, and no recurrence.

The Trick (Positional Encoding)

Change the **input**, so that the same token at different positions becomes a different vector:

$$X \rightarrow X + E_{\text{pos}}, \quad E_{\text{pos}} \in \mathbb{R}^{T \times d}.$$

Now in general

$$\text{Attention}(P(X + E_{\text{pos}})) \neq P \text{ Attention}(X + E_{\text{pos}})$$

because $PE_{\text{pos}} \neq E_{\text{pos}}$. The equivariance is **broken on purpose**.

How do we choose E_{pos} ? — next section.

Positional Encoding — Three Standard Approaches

Let $X \in \mathbb{R}^{T \times d}$ be the token-embedding matrix.

Method	Where it acts	Params	Used by
Sinusoidal	$H_0 = X + P_{sin}$ (computed)	0	“Attention Is All You Need”
Learned absolute	$H_0 = X + E_{pos}[0:T]$	$n_{max} \cdot d$	GPT-2, GPT-3
RoPE*	rotate Q, K inside attention	0	LLaMA, frontier models

The framing matters

Two of these modify the **input** ($H_0 = X + P$); RoPE does *not* — it leaves $H_0 = X$ untouched and injects position later, inside the attention computation. There is no single “ P matrix” that covers all three.

Learned Absolute (GPT-2 / GPT-3 Style)

A second lookup table, keyed on **position** instead of token identity:

$$E_{\text{pos}} \in \mathbb{R}^{n_{\text{max}} \times d}$$

For input length $T \leq n_{\text{max}}$:

$$H_0 = X + E_{\text{pos}}[0:T, :] \in \mathbb{R}^{T \times d}.$$

The one n_{max} in the entire architecture

This is the *only* weight whose shape depends on the **maximum sequence length** n_{max} .

- Parameter count: $n_{\text{max}} \cdot d$.
- Shorter input: prefix of rows; the rest get zero gradient on that pass.
- Longer input: **no row to look up** — the model literally cannot process it.

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

Don't add anything to X . Instead, rotate Q and K inside attention by a position-dependent angle. For each consecutive 2-coordinate slice $(2k, 2k + 1)$ of row q_t (and k_t) apply the 2D rotation

$$R(t\omega_k) = \begin{pmatrix} \cos(t\omega_k) & -\sin(t\omega_k) \\ \sin(t\omega_k) & \cos(t\omega_k) \end{pmatrix}, \quad \omega_k = \frac{1}{10000^{2k/d}}.$$

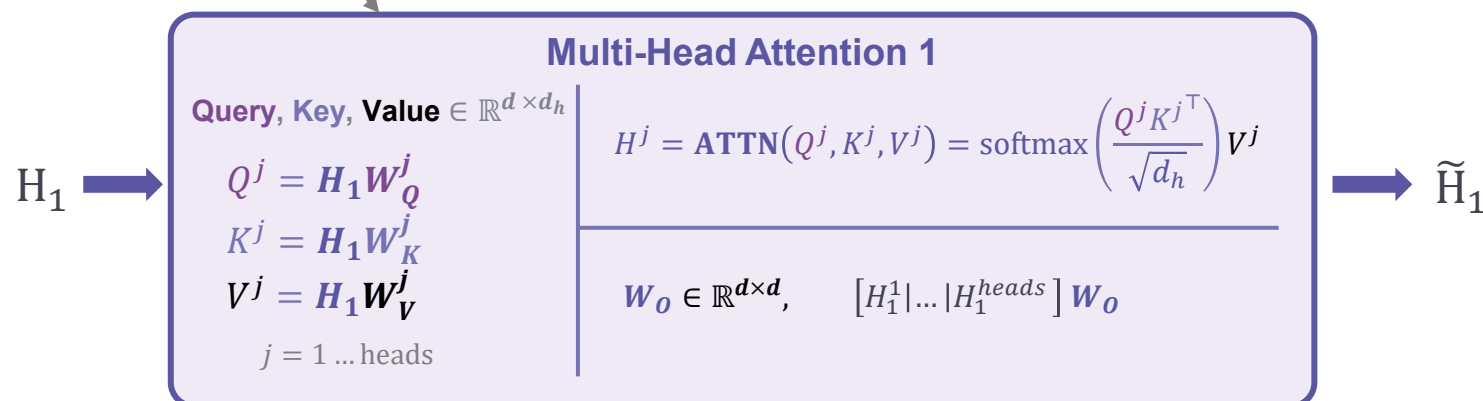
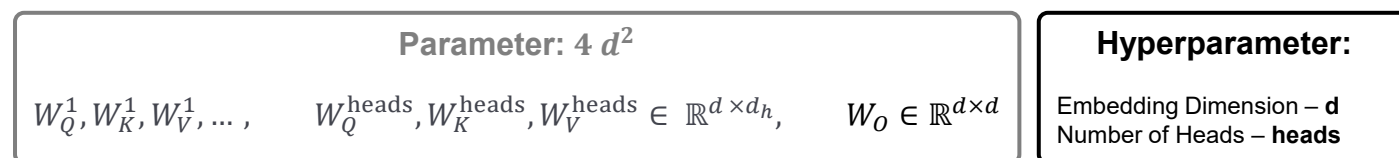
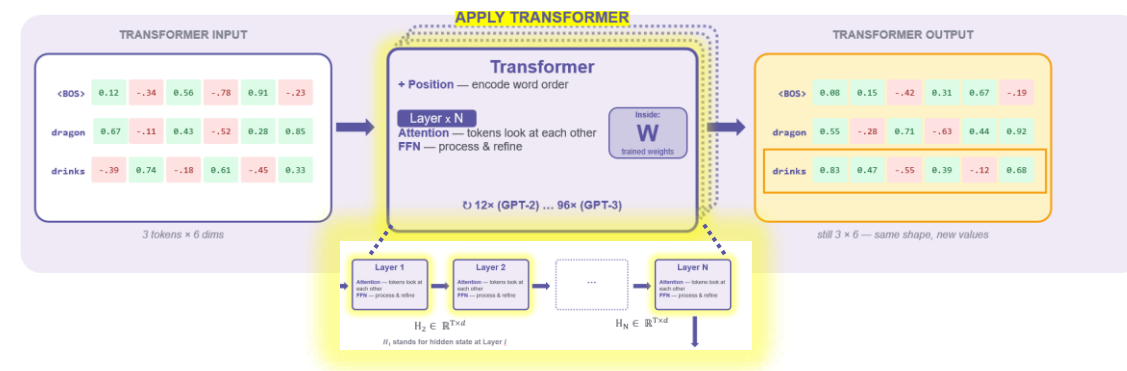
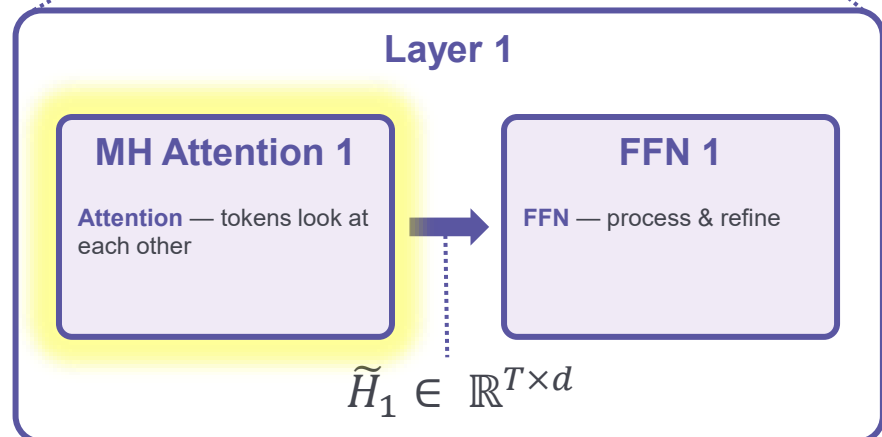
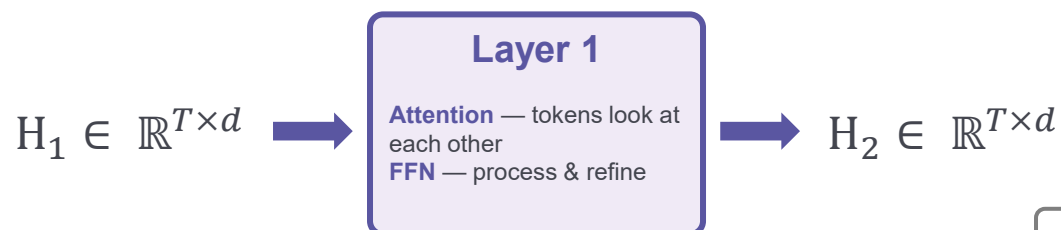
Why it became the modern default

- **Zero learned parameters**, no n_{max} -shaped matrix.
- The score $q_t^T k_s$ depends only on the relative offset $t - s$ — relative encoding, not absolute
- Extrapolates beyond the training length (with degrading quality; mitigated by NTK /YaRN scaling)
- Used by LLaMA, Mistral, Qwen, and most current frontier models

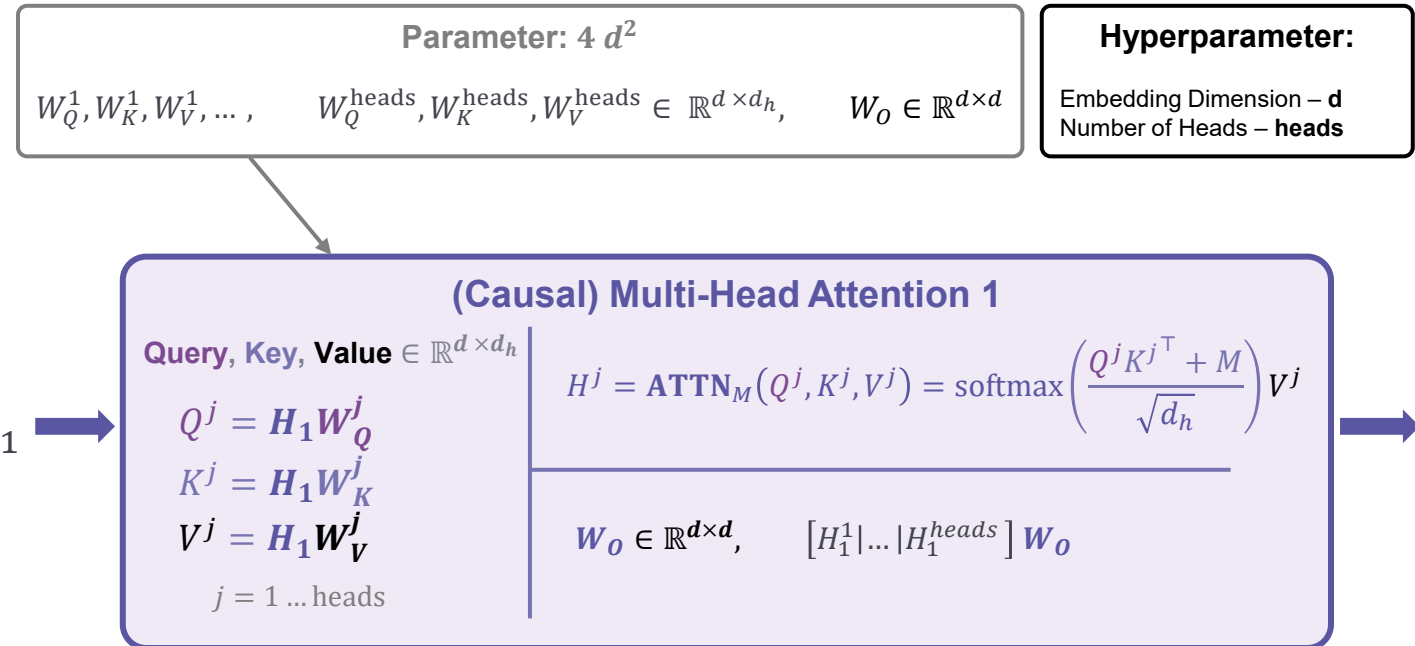
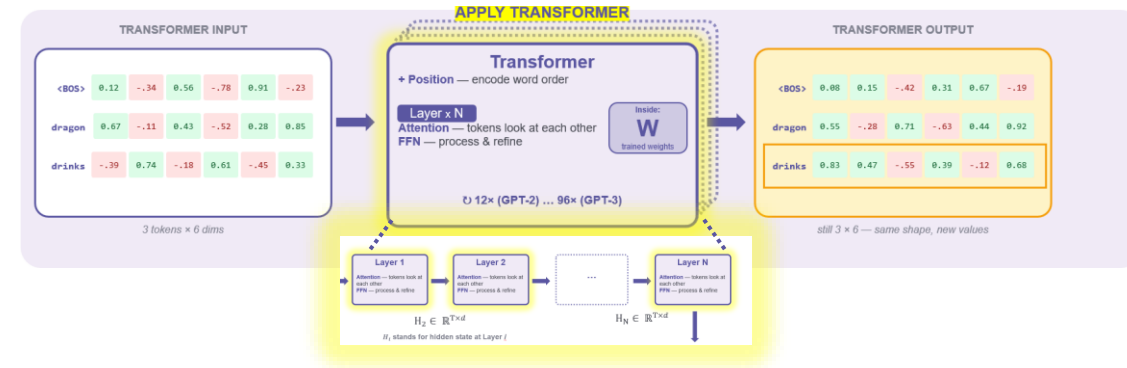
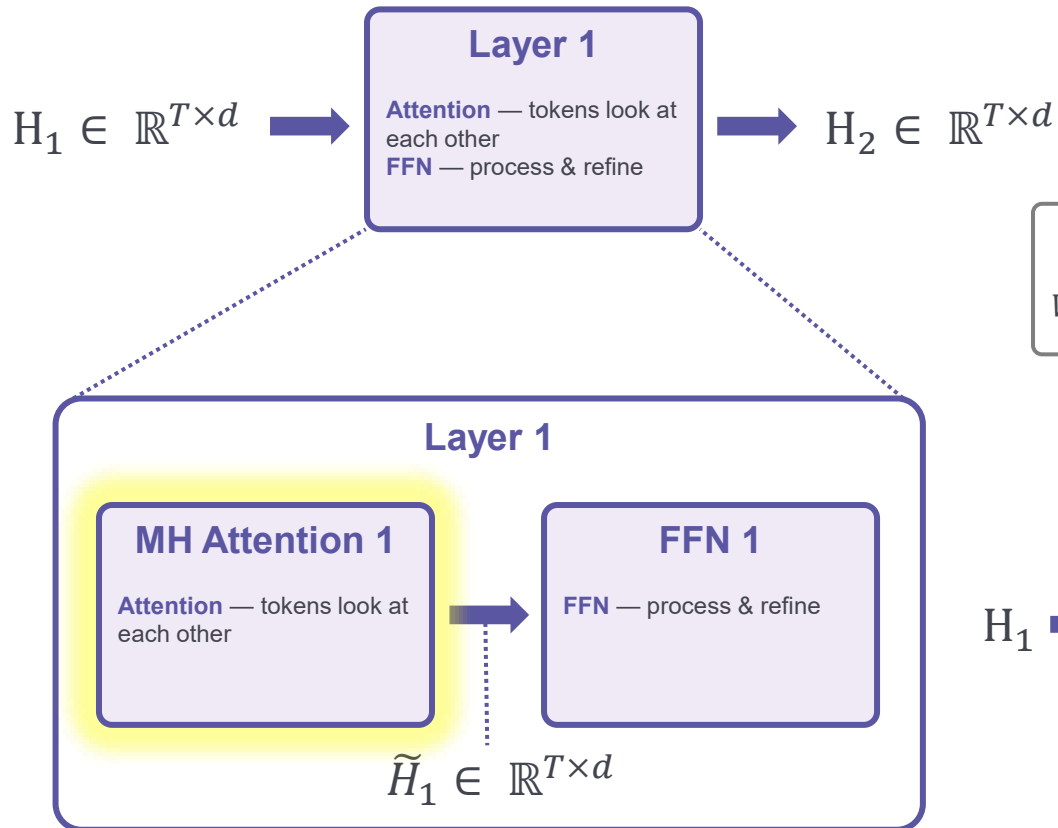
Feed-Forward Network

10

Back to the Layers



Back to the Layers



The Feed-Forward Block — Where Knowledge Lives

After multi-head attention, each token has absorbed information from other tokens. Attention has done its job of gathering — but it has not yet done much thinking.

All attention does is produce weighted sums of existing vectors. Turning a pronoun-that-has-absorbed-context into the concept of the animal it refers to, enriched with facts about animals — that is the job of the FFN.

Attention vs. FFN, in one breath

Attention **gathers information** across positions.
FFN **processes it**, per token, in isolation.

Most of what we call "the model's knowledge" lives in FFN weights.

FFN: Parameters and the Expansion Factor

A two-layer MLP (Multi-Layer Perceptron) applied independently to each token. For each block ℓ :

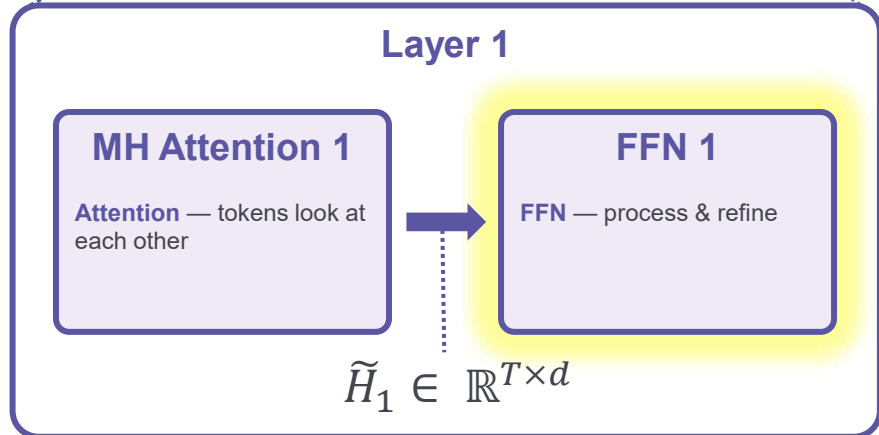
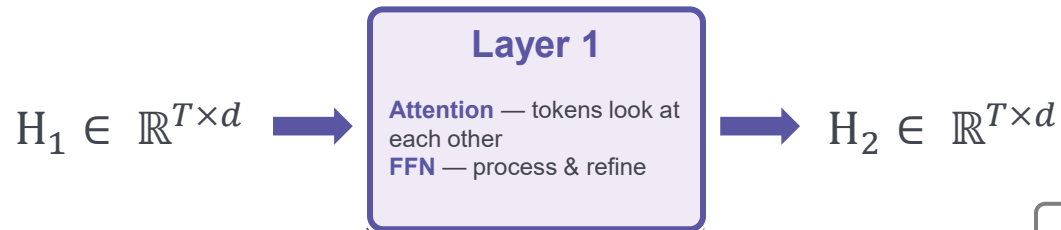
$$W_1^{(\ell)} \in \mathbb{R}^{d \times 4d}, \quad b_1^{(\ell)} \in \mathbb{R}^{4d}, \quad W_2^{(\ell)} \in \mathbb{R}^{4d \times d}, \quad b_2^{(\ell)} \in \mathbb{R}^d$$

The expansion factor

The first matrix expands the dimension by a factor of 4 (sometimes 8 in newer architectures): blow the token up to 4× its size, apply a non-linearity, then compress back down.

GPT-3 ($d = 12,288$): hidden dim inside the FFN is $4d = 49,152$.

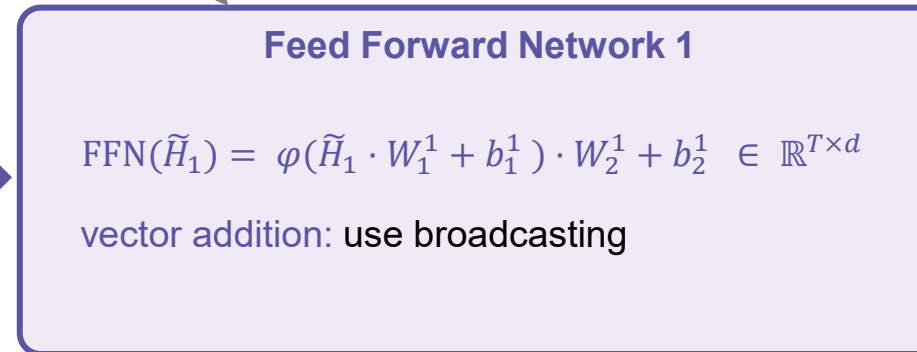
FFN Layer



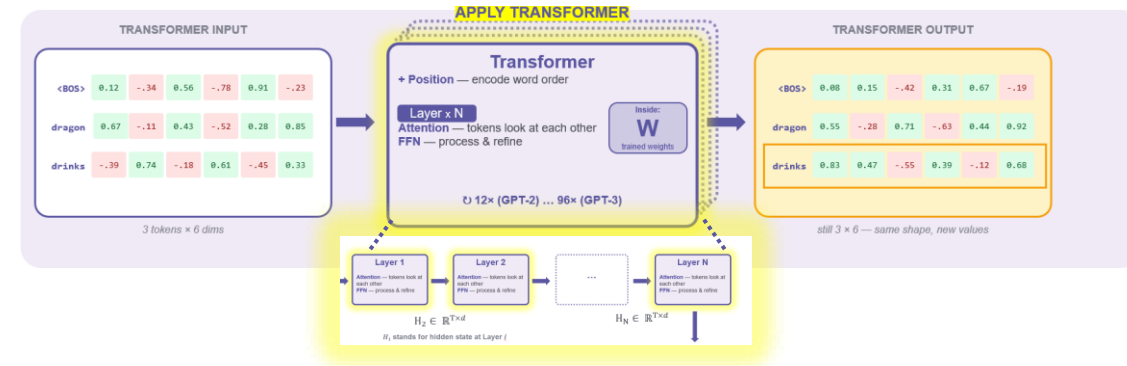
$\tilde{H}_1$

Parameter: $4d^2 + 4d + 4d^2 + d = 8d^2 + 5d$

$W_1^1 \in \mathbb{R}^{d \times 4d}, \quad b_1^1 \in \mathbb{R}^{4d}, \quad W_2^1 \in \mathbb{R}^{4d \times d}, \quad b_2^1 \in \mathbb{R}^d$



→ H_2



Hyperparameter:

Embedding Dimension — d
 Nonlinearity — φ (ReLU, GeLU, SwiGLU)

FFN: Forward Pass

Applied **per row**. For each token $h_t \in \mathbb{R}^d$:

$$\text{FFN}(h_t) = \varphi(h_t^\top W_1 + b_1) W_2 + b_2 \in \mathbb{R}^d$$

where φ is a pointwise non-linearity (ReLU, GeLU, SwiGLU — the choice does not affect shapes).
On the whole matrix (broadcasting b_1, b_2 across rows):

$$\text{FFN}(H) = \varphi(H W_1 + b_1) W_2 + b_2 \in \mathbb{R}^{T \times d}$$

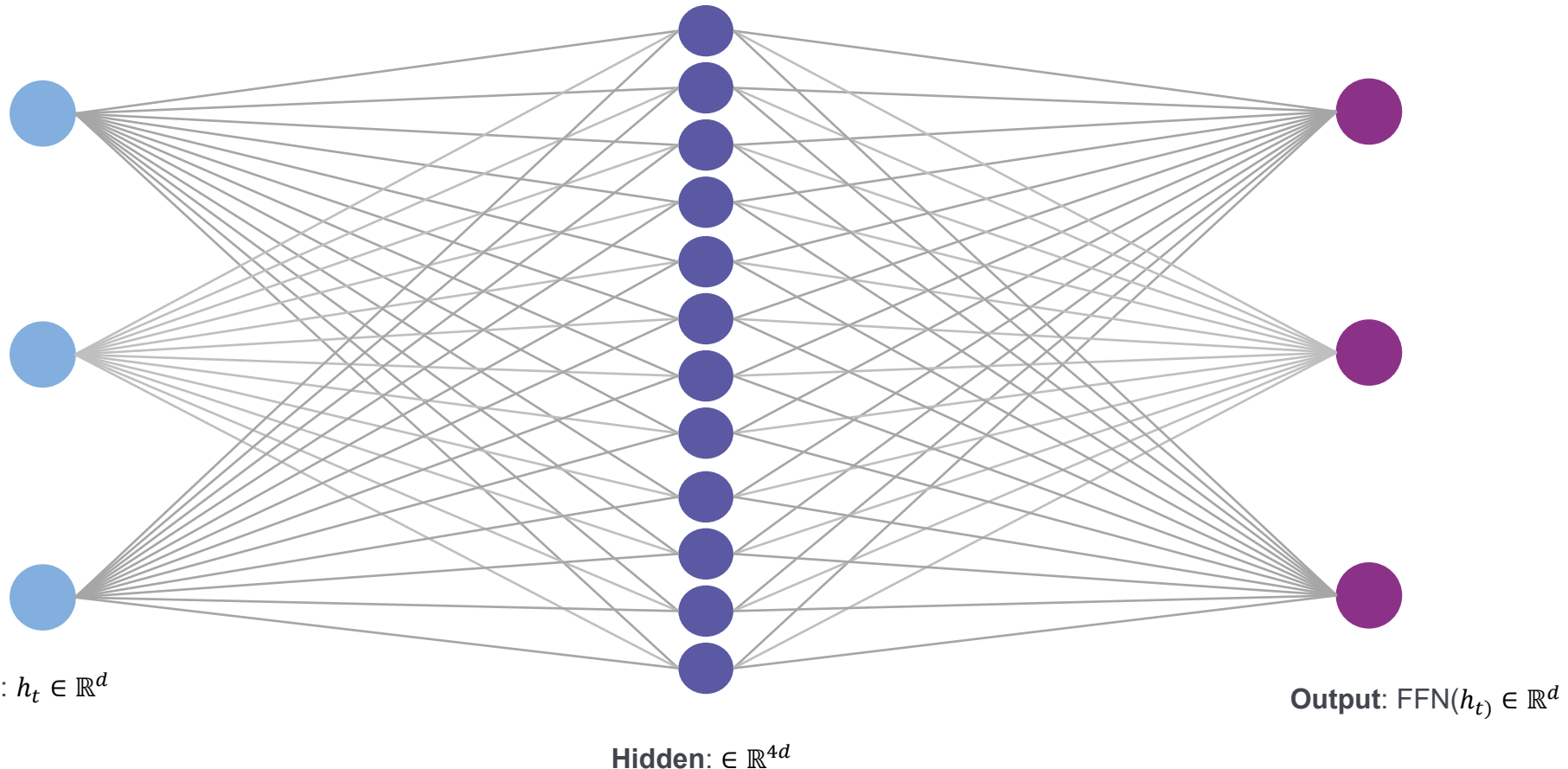
Shape check.

- $H \cdot W_1: (T \times d) \cdot (d \times 4d) \rightarrow (T \times 4d)$
- add b_1 , apply φ : same shape
- multiply by $W_2: (T \times 4d) \cdot (4d \times d) \rightarrow (T \times d) \checkmark$

FFN: Dense 2-Layer NN on the Rows

For each vector associated to the token, $h_t \in \mathbb{R}^d$: $\text{FFN}(h_t) = \varphi(h_t^\top W_1 + b_1) W_2 + b_2 \in \mathbb{R}^d$

- $h_t^\top W_1 : \mathbb{R}^d \times \mathbb{R}^{d \times 4d} \rightarrow \mathbb{R}^{4d}$, $\cdot W_2 : \mathbb{R}^{4d} \times \mathbb{R}^{4d \times d} \rightarrow \mathbb{R}^d$, in total: $\mathbb{R}^d \rightarrow \mathbb{R}^d$



Where the Parameters Live

Per transformer block, the FFN holds roughly $8d^2$ parameters — about two-thirds of the block's total weights.

Component	Shape	Parameters
W_1 (expand)	$d \times 4d$	$4d^2$
W_2 (project)	$4d \times d$	$4d^2$
Total per FFN	—	$\approx 8d^2$

Why this matters at scale

Attention has $4d^2$ parameters per block (Q, K, V, O projections). FFN has $\approx 8d^2$ — twice as many. Most of the parameter budget — and most of the model's stored knowledge — sits in the FFN.

FFN as Key-Value Memory

A useful mental model from interpretability research: read the FFN as a **key–value lookup**, where W_1 stores **keys** and W_2 stores **values**.

Reading the matmuls as memory access

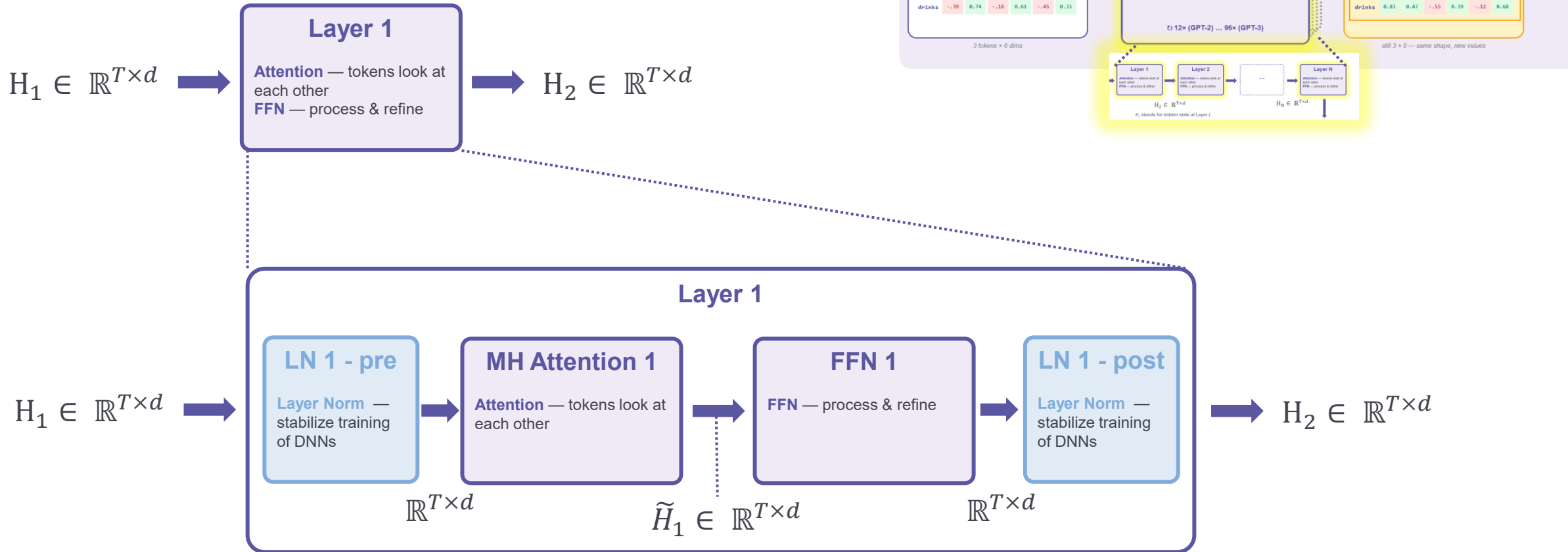
- **Rows of W_1 act as keys**: each row pattern-matches a feature in the input.
- $\text{GeLU}(xW_1 + b_1)$ is the **activation pattern** — which keys fired, and how strongly.
- **Columns of W_2 act as values**: a weighted sum produces the output to write back.
- Hidden-layer neurons behave like **detectors**: “this is a French verb”, “this is a country in Africa”, “this is a question about cooking”.

This view explains why FFNs are where much of a model’s factual knowledge appears to be stored.

Complete Forward Path

11

Inside the Layers – with Training Details



Layer Norm – Row wise

Parameter: $d + d = 2d$

$$\gamma \in \mathbb{R}^d, \quad \beta \in \mathbb{R}^d$$

Row-wise Layer Norm rescales each token's d -vector to **zero mean and unit variance**, then applies a learned scale and shift. Three reasons it sits before MHA and FFN:

- **Stable activations across depth.** Without it, residual sums grow with L ; normalising each token before each sub-layer keeps inputs in a fixed range.
- **Healthier gradients.** Bounded inputs avoid softmax saturation in attention and exploding/vanishing gradients in deep stacks — Pre-LN trains reliably without learning-rate warmup.
- **Per-token, not per-batch.** Each row is normalised independently, so behaviour is identical at training and inference and unaffected by batch size or sequence length T .

$$H = \begin{bmatrix} h_1^\top \\ \vdots \\ h_T^\top \end{bmatrix} \in \mathbb{R}^{T \times d} \rightarrow$$

Layer Norm

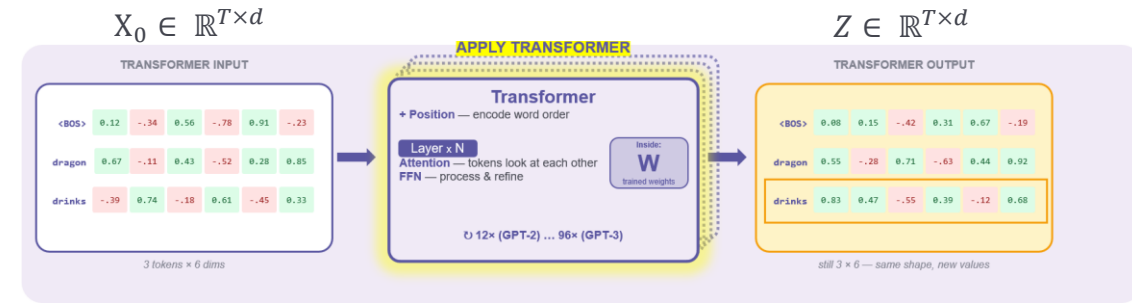
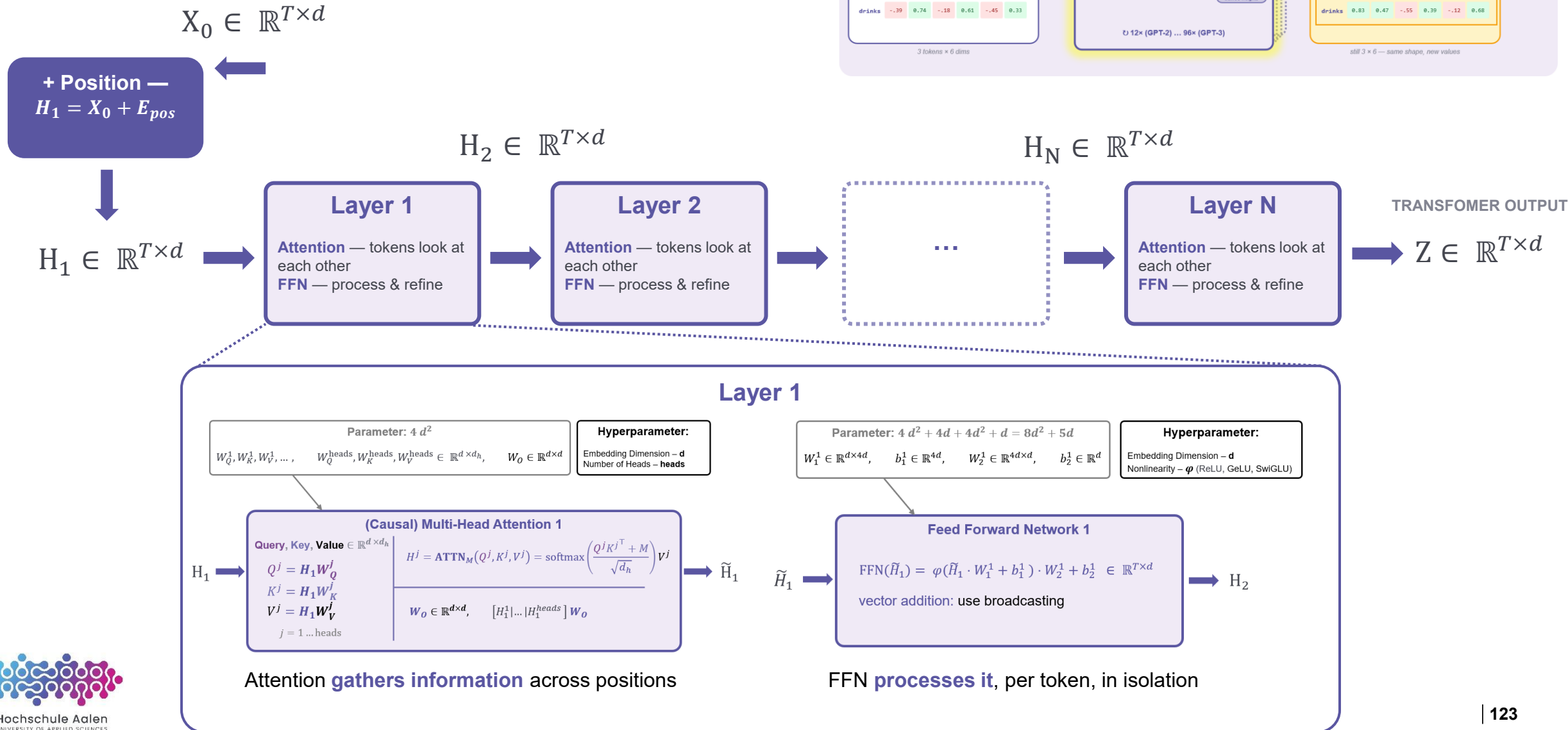
$$\mu_t = \frac{1}{d} \sum_{j=1}^d h_{t,j} = \text{mean}(h_t)$$
$$\sigma_t^2 = \frac{1}{d} \sum_{j=1}^d (h_{t,j} - \mu_t)^2 = \text{var}(h_t)$$

$$\text{LN}(h_t) = \gamma \odot \frac{h_t - \mu_t}{\sigma_t} + \beta \in \mathbb{R}^d,$$

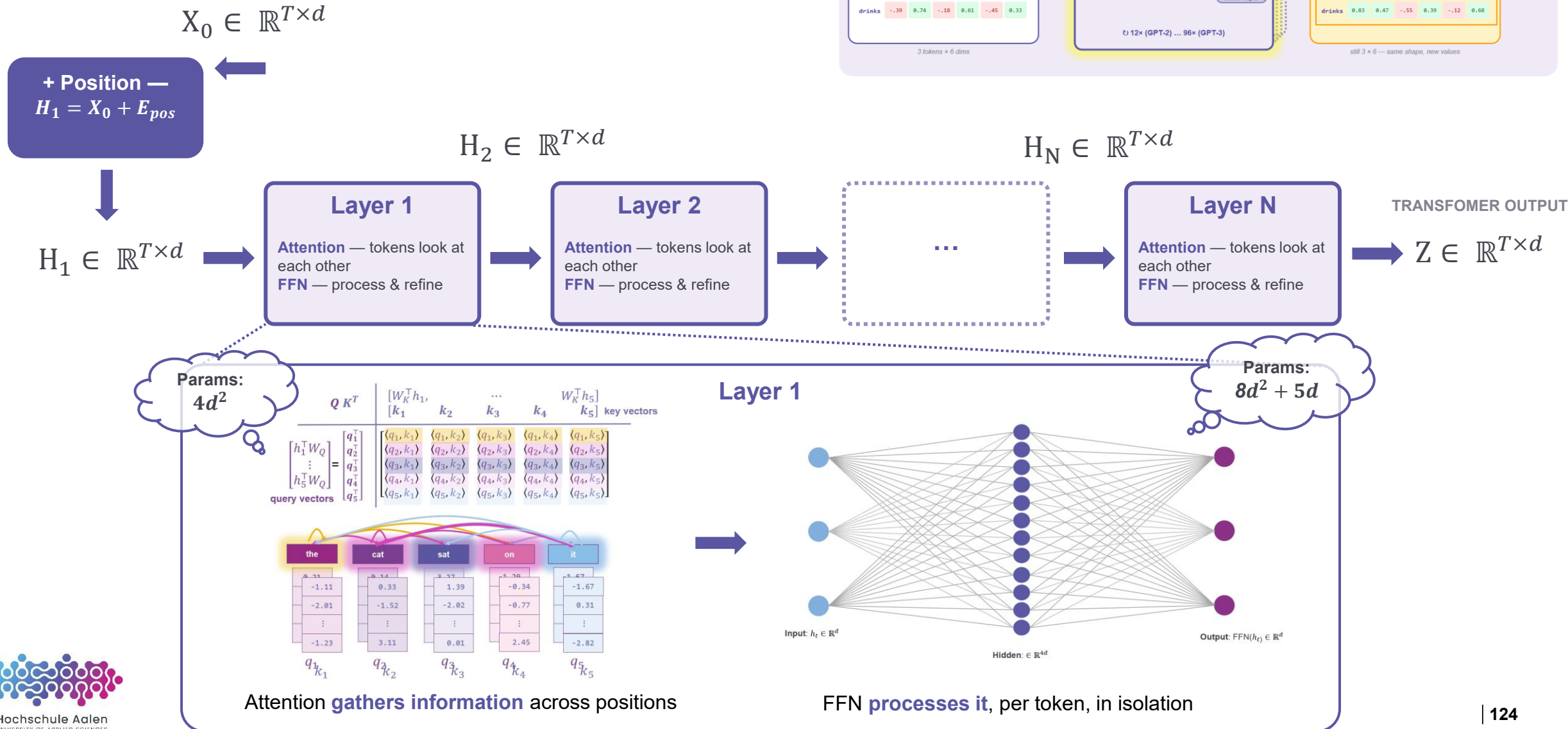
for all rows $t = 1, \dots, T$

$$\rightarrow \text{LN}(H) \in \mathbb{R}^{T \times d}$$

Putting the Blocks Together



Putting the Blocks Together



Putting the Blocks Together

$$X_0 \in \mathbb{R}^{T \times d}$$

+ Position —
 $H_1 = X_0 + E_{pos}$

$$H_1 \in \mathbb{R}^{T \times d}$$

Layer 1

Attention — tokens look at each other
 FFN — process & refine

$$H_2 \in \mathbb{R}^{T \times d}$$

Layer 2

Attention — tokens look at each other
 FFN — process & refine

...

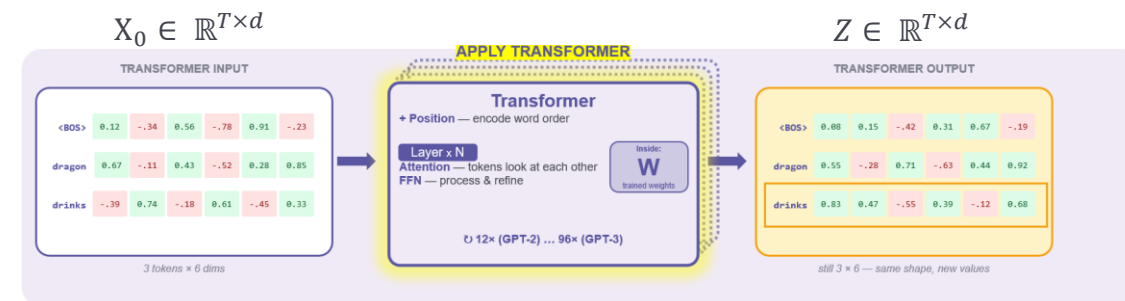
$$H_N \in \mathbb{R}^{T \times d}$$

Layer N

Attention — tokens look at each other
 FFN — process & refine

TRANSFORMER OUTPUT

$$Z \in \mathbb{R}^{T \times d}$$



Layer 1

Parameter: $d + d = 2d$
 $\gamma \in \mathbb{R}^d, \beta \in \mathbb{R}^d$

Layer Norm

$$\mu_t = \frac{1}{d} \sum_{j=1}^d h_{t,j} = \text{mean}(h_t)$$

$$\sigma_t^2 = \frac{1}{d} \sum_{j=1}^d (h_{t,j} - \mu_t)^2 = \text{var}(h_t)$$

$$\text{LN}(h_t) = \gamma \odot \frac{h_t - \mu_t}{\sigma_t} + \beta \in \mathbb{R}^d$$

for all rows $t = 1, \dots, T$

Parameter: $4d^2$
 $W_Q^1, W_K^1, W_V^1, \dots, W_Q^{\text{heads}}, W_K^{\text{heads}}, W_V^{\text{heads}} \in \mathbb{R}^{d \times d_h}, W_O \in \mathbb{R}^{d \times d}$

(Causal) Multi-Head Attention 1

Query, Key, Value $\in \mathbb{R}^{d \times d_h}$

$$Q^j = H_1 W_Q^j, K^j = H_1 W_K^j, V^j = H_1 W_V^j, j = 1 \dots \text{heads}$$

$$H^j = \text{ATTN}_M(Q^j, K^j, V^j) = \text{softmax}\left(\frac{Q^j K^{j\top} + M}{\sqrt{d_h}}\right) V^j$$

$$W_O \in \mathbb{R}^{d \times d}, [H_1^1 \dots H_1^{\text{heads}}] W_O$$

Attention **gathers information** across positions

Hyperparameter:
 Embedding Dimension — d
 Number of Heads — heads

Parameter: $4d^2 + 4d + 4d^2 + d = 8d^2 + 5d$
 $W_1^1 \in \mathbb{R}^{d \times 4d}, b_1^1 \in \mathbb{R}^{4d}, W_2^1 \in \mathbb{R}^{4d \times d}, b_2^1 \in \mathbb{R}^d$

Feed Forward Network 1

$$\text{FFN}(\tilde{H}_1) = \varphi(\tilde{H}_1 \cdot W_1^1 + b_1^1) \cdot W_2^1 + b_2^1 \in \mathbb{R}^{T \times d}$$

vector addition: use broadcasting

FFN **processes it**, per token, in isolation

Hyperparameter:
 Embedding Dimension — d
 Nonlinearity — φ (ReLU, GeLU, SwiGLU)

Parameter: $d + d = 2d$
 $\gamma \in \mathbb{R}^d, \beta \in \mathbb{R}^d$

Layer Norm

$$\mu_t = \frac{1}{d} \sum_{j=1}^d h_{t,j} = \text{mean}(h_t)$$

$$\sigma_t^2 = \frac{1}{d} \sum_{j=1}^d (h_{t,j} - \mu_t)^2 = \text{var}(h_t)$$

$$\text{LN}(h_t) = \gamma \odot \frac{h_t - \mu_t}{\sigma_t} + \beta \in \mathbb{R}^d$$

for all rows $t = 1, \dots, T$

Tracking the Parameter

Layer Parameter: $12d^2 + 9d$

Attention Parameter: $4d^2$

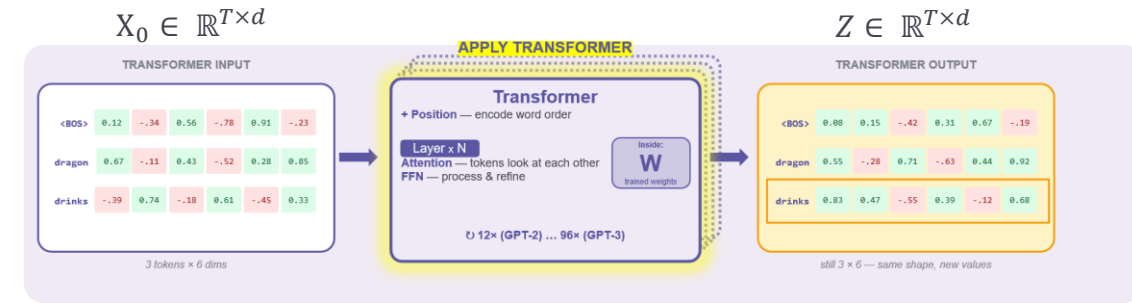
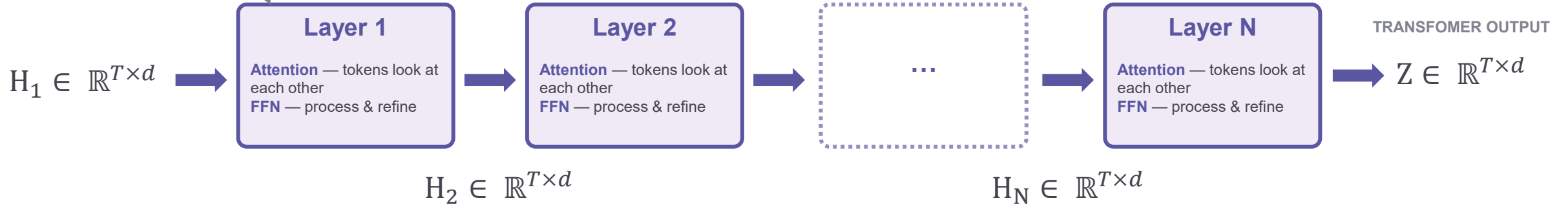
$$W_Q^1, W_K^1, W_V^1, \dots, W_Q^{\text{heads}}, W_K^{\text{heads}}, W_V^{\text{heads}} \in \mathbb{R}^{d \times d_h}, W_O \in \mathbb{R}^{d \times d}$$

FFN Parameter: $8d^2 + 5d$

$$W_1^1 \in \mathbb{R}^{d \times 4d}, \quad b_1^1 \in \mathbb{R}^{4d}, \quad W_2^1 \in \mathbb{R}^{4d \times d}, \quad b_2^1 \in \mathbb{R}^d$$

Layer Norm Parameter: $4d$

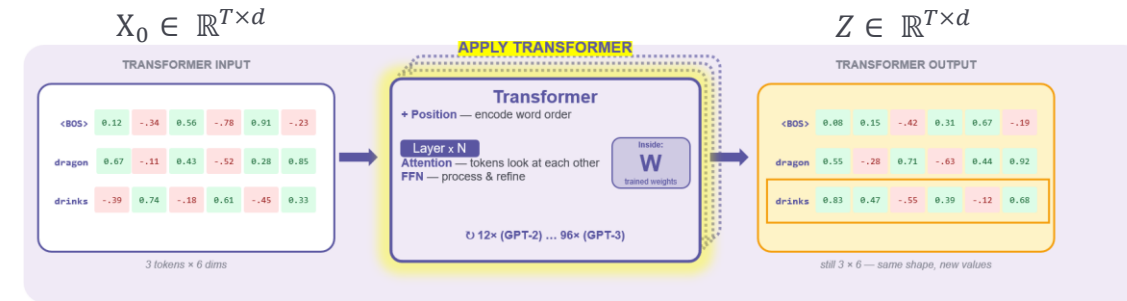
$$\gamma^{\text{pre}} \in \mathbb{R}^d, \quad \beta^{\text{pre}} \in \mathbb{R}^d, \quad \gamma^{\text{post}} \in \mathbb{R}^d, \quad \beta^{\text{post}} \in \mathbb{R}^d$$



Hyperparameter:

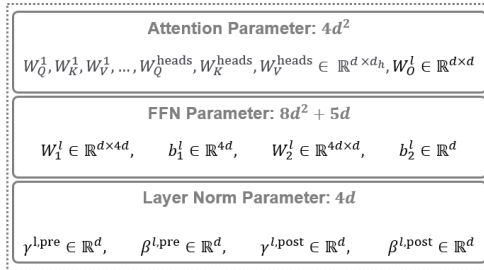
- Vocabulary Size – **V**
- Embedding Dimension – **d**
- Number of Layers – **N**
- Number of Heads – **heads**
- Nonlinearity – ϕ (ReLU, GeLU, SwiGLU)

Tracking the Parameter



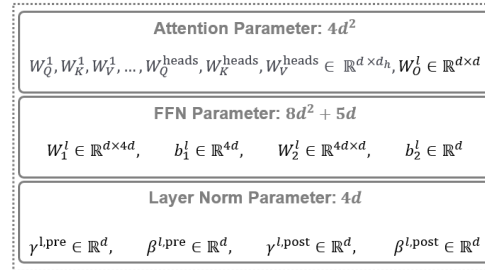
$\ell = 1$

Layer Parameter: $12d^2 + 9d$



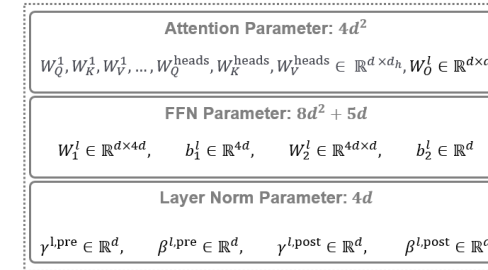
$\ell = 2$

Layer Parameter: $12d^2 + 9d$



$\ell = N$

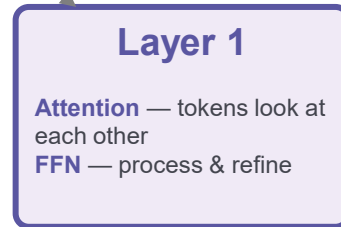
Layer Parameter: $12d^2 + 9d$



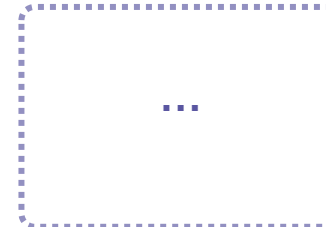
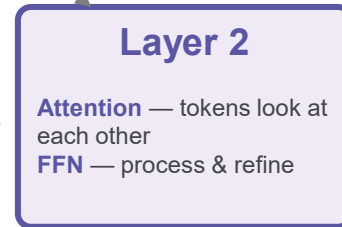
Total number of learnable Parameter inside the Transformer

$$N(12d^2 + 9d)$$

$$H_1 \in \mathbb{R}^{T \times d}$$



$$H_2 \in \mathbb{R}^{T \times d}$$



$$H_N \in \mathbb{R}^{T \times d}$$



TRANSFORMER OUTPUT

$$Z \in \mathbb{R}^{T \times d}$$

From Last Hidden State to Vocabulary — The Output Head

START

Prompt = Raw Text

<BOS> dragon drinks

$V = 10$, Vocabulary

Tokenizer

$T = 3$, tokens

Token IDs

[0, 5, 6]

$d = 6$, embedding dim.

Embedding Lookup

TRANSFORMER INPUT

<BOS>	0.12	-.34	0.56	-.78	0.91	-.23
dragon	0.67	-.11	0.43	-.52	0.28	0.85
drinks	-.39	0.74	-.18	0.61	-.45	0.33

3 tokens $\times$ 6 dims

Transformer

+ Position — encode word order

Layer $\times$ N

Attention — tokens look at each other
FFN — process & refine

Inside:

W

trained weights

12 $\times$ (GPT-2) ... 96 $\times$ (GPT-3)

TRANSFORMER OUTPUT

<BOS>	0.08	0.15	-.42	0.31	0.67	-.19
dragon	0.55	-.28	0.71	-.63	0.44	0.92
drinks	0.83	0.47	-.55	0.39	-.12	0.68

still 3 $\times$ 6 — same shape, new values

LAST TOKEN: "drinks"

[0.83, 0.47, -0.55, 0.39, -0.12, 0.68]

multiply by
output weights
(10 $\times$ 6 matrix)

10 LOGITS (ONE PER VOCAB TOKEN)

[-1.2, 0.5, -0.3, 1.8, **4.2**, 0.1, -0.8, 3.1, 0.9, 1.1]

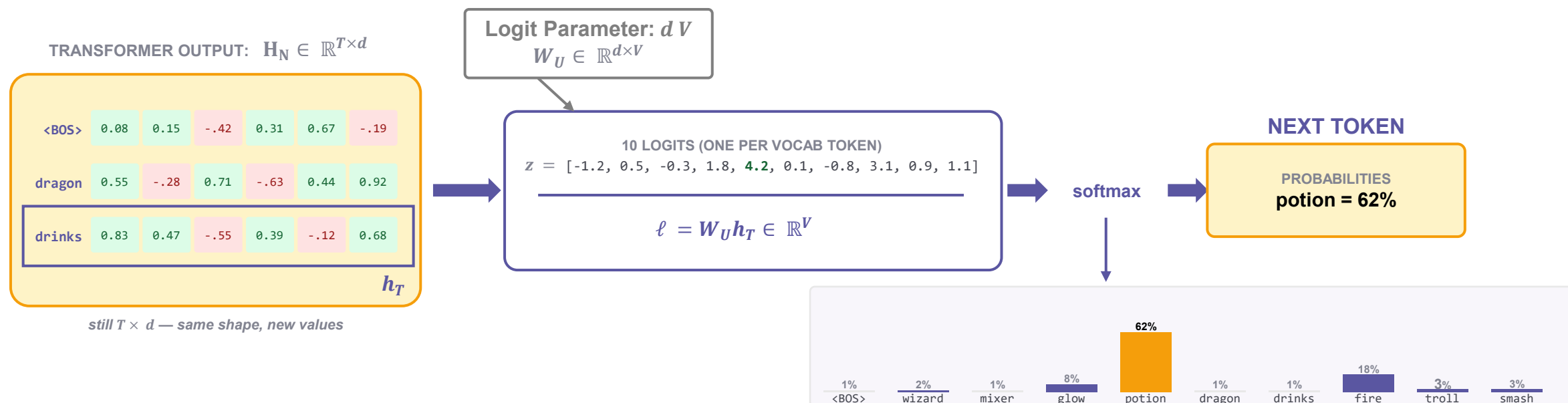
softmax

NEXT TOKEN

PROBABILITIES
potion = 62%

<BOS> dragon drinks **potion**

From Last Hidden State to Vocabulary — The Output Head

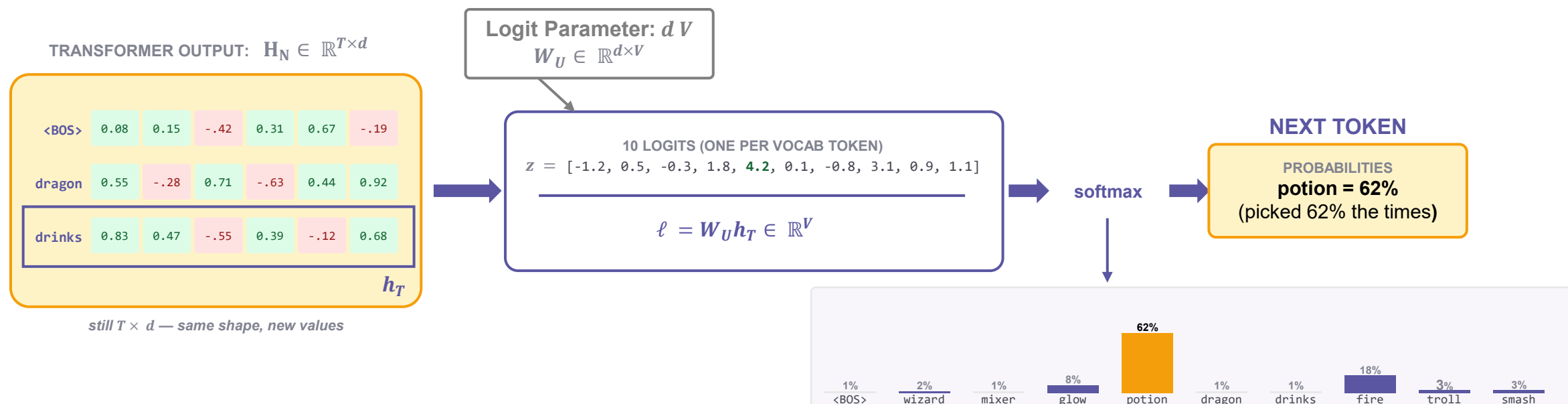


Step 1 — pick the last token's hidden vector: $h_T = H_N[T, :] \in \mathbb{R}^d$

Step 2 — project to vocabulary **logits**: $\ell = W_U h_T \in \mathbb{R}^V$, where $W_U \in \mathbb{R}^{d \times V}$.

Step 3 — softmax to get next-token distribution: $p = \text{softmax}(\ell) \in [0, 1]^V$

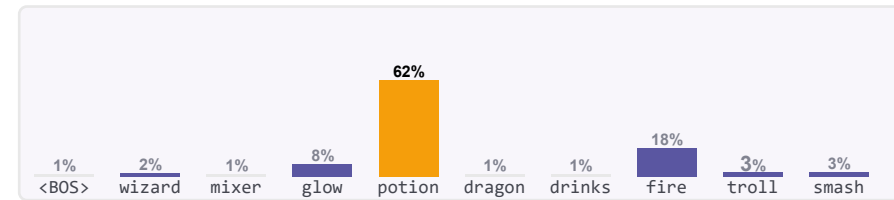
Softmax — from logits to probabilities



A subtle but important point about what happens next. **The model does not just pick the highest-probability token.** It *draws a random token* from this distribution, where each token's chance of being drawn equals its probability.

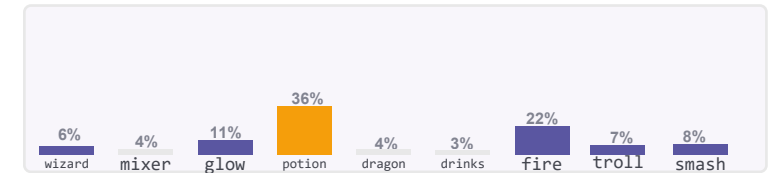
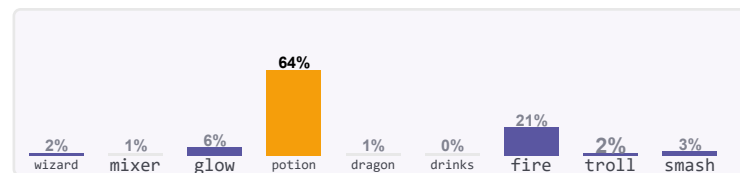
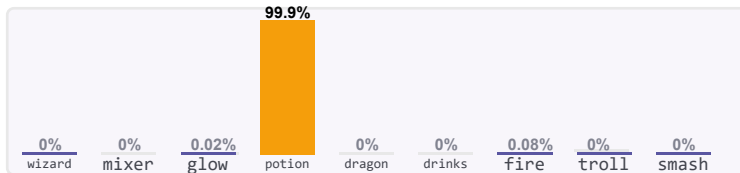
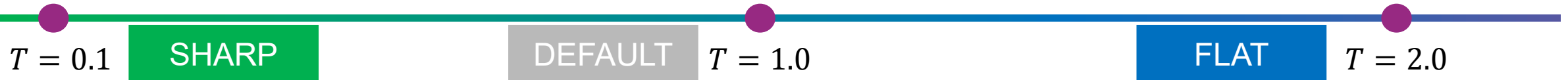
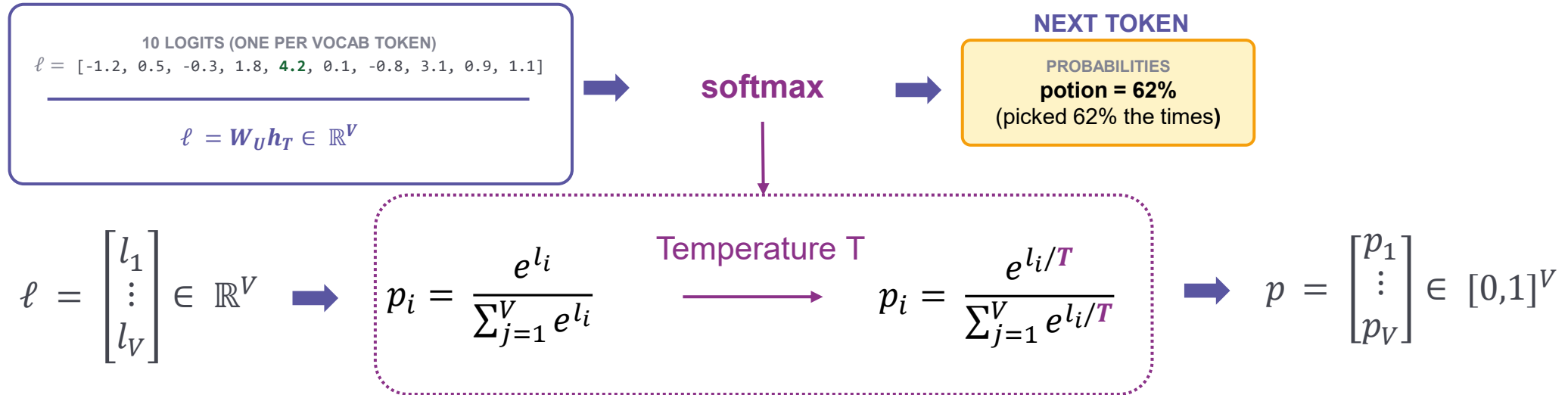
Softmax — from logits to probabilities

Why bother with the random draw at all?



- **Use the full distribution.** Argmax keeps only the rank and throws away the model's uncertainty.
- **Enable variation.** Argmax is deterministic — fatal for creative or open-ended generation.
- **Escape greedy failure modes.** Looping, repetition, and first-token cascades; a little stochasticity breaks them.

Temperature – controlling how peaked the distribution is



Temperature – controlling how peaked the distribution is

$T = 0.1$

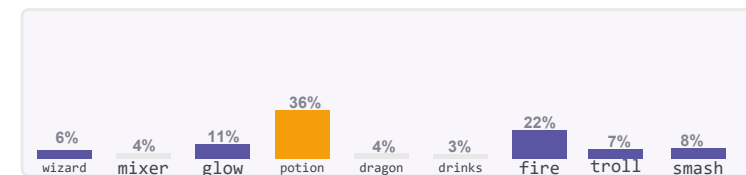
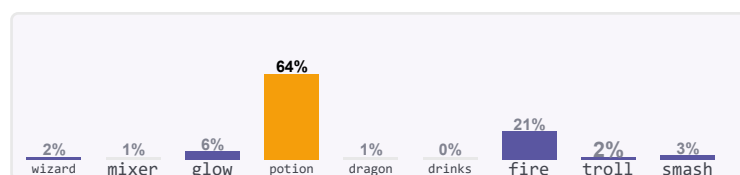
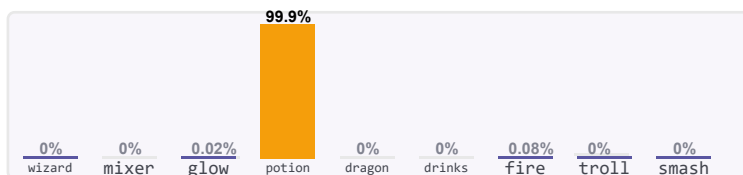
SHARP

DEFAULT

$T = 1.0$

FLAT

$T = 2.0$



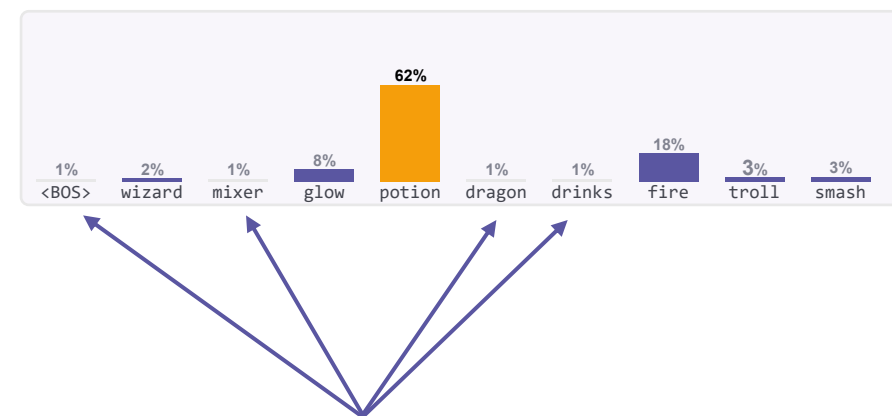
- **$T \rightarrow 0$ (greedy).** The output collapses onto the highest-logit token with probability 1. The model becomes deterministic: “always pick the best”. This is what you use when you want the safest, most predictable completion — generating code that compiles or answering a factual question.
- **$T = 0.3$ (low).** The distribution is sharpened but not collapsed. Top tokens dominate but occasionally a runner-up wins. Good for code that needs to vary slightly across runs (e.g. name generation, unit-test synthesis).
- **$T = 0.7$ (typical).** The default for most chat-style APIs. A balance: top tokens are strongly preferred, but there is enough breadth that outputs feel natural and not robotic.
- **$T = 1.0$ (unscaled).** The distribution that actually came out of the model’s head, unmodified. Often what researchers report.
- **$T = 1.5$ or higher.** The distribution is flattened; improbable tokens get meaningful probability mass. Creative, weird, occasionally incoherent. Useful for brainstorming. Dangerous for code.

Top-k and top-p (nucleus) sampling

Temperature alone is not enough

Two filtering techniques tame the tail.

- **Top-k sampling.** Keep the k highest-probability tokens, zero the rest, renormalise.
 - Typical k: 40, 50, 100. Fixed cut — does not adapt to confidence.
- **Top-p (nucleus) sampling.** Keep the smallest set whose cumulative probability reaches p, then renormalise.
 - Typical p: 0.9–0.95. Adaptive — narrow when confident, wide when uncertain.



Even at moderate temperatures, softmax assigns **some probability** to **pathologically rare tokens** — the **long tail** of the vocabulary that almost never makes sense in any context. Sampling from it occasionally produces genuinely broken output.

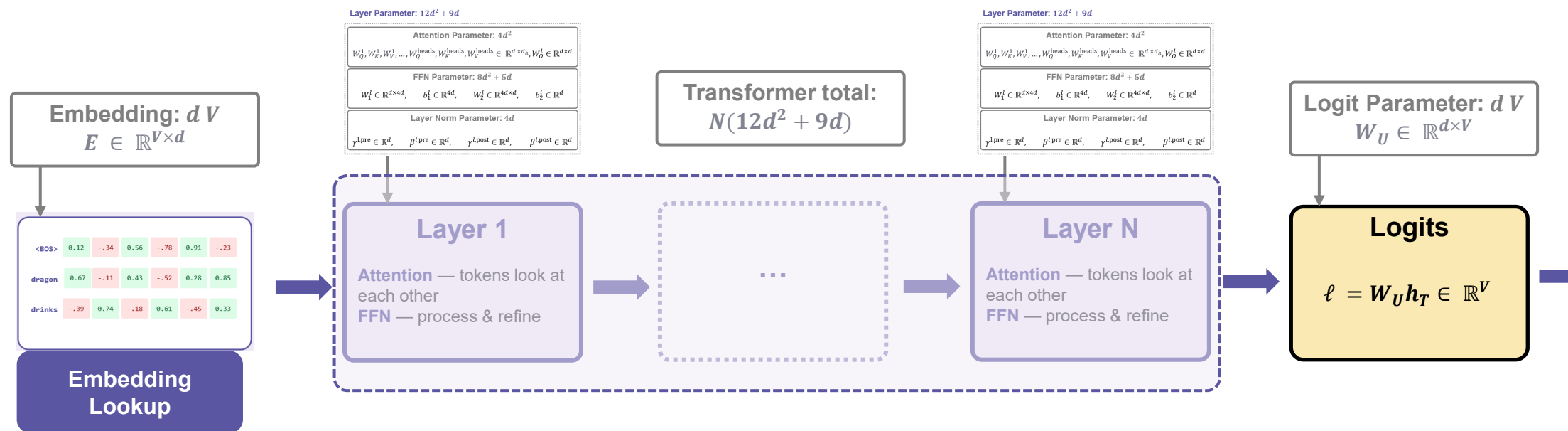
Tracking all the Parameter

Decoding Parameter:

Temperature – T
top-p – p
top k – k

5 – Hyperparameter:

Vocabulary Size – V
Embedding Dimension – d
Number of Layers – N
Number of Heads – **heads**
Nonlinearity – ϕ (ReLU, GeLU, SwiGLU)



Total number of parameter:

$$12Nd^2 + (2V + 9)d$$

The Key Insight

All these dimensions — V , d , N , H — are **baked into the parameters**.

*Once chosen, they are **fixed for the lifetime of the model**.*

*The sequence length T is **not** — same parameters handles any T .*

*This is what makes a Transformer **one model that generalizes across sequence lengths** — not a different model per length.*



Decoder-only LLM parameter accounting

From GPT-2 / GPT-3 to
Llama, Mistral, Qwen, Mixtral

12

What changes in the parameter budget?

- **Classic GPT-style** dense blocks converge to $\sim 1/3$ attention and $\sim 2/3$ FFN at large scale.
- Modern open LLMs shift parameters away from attention via **GQA/MQA (Grouped Query Attention / Multi-Query Attention)** and toward **SwiGLU** FFNs.
- Embeddings become material again when the vocabulary expands (e.g., Llama 3, Qwen2.5).
- **MoE (Mixture of Experts) models** separate total parameters from active parameters per token.



Goal: compare how many parameters sit in embeddings / logits vs attention vs FFN across representative model families.

Classic GPT block vs modern LLaMA-style block

Notation: d = embedding dimension, V = vocabulary, d_{ff} = FFN width, h_q = query heads, h_{kv} = KV heads, E = experts, k = active experts

Classic dense decoder (GPT-2 / GPT-3 style)

- **Layer norms** $\approx 4d$ per layer ($2 \times [\gamma, \beta]$)
- **Attention** $= 4d^2$ (Q, K, V, O all $d \rightarrow d$)
- **FFN (GELU)** $= 8d^2 + 5d$
- **Token embedding** $= Vd$
- **Logits head** $= Vd$
- **Positional embeddings** $= n_{\text{max}} d$ (learned absolute positions)
- **Typical split at scale** attention $\approx 1/3$, FFN $\approx 2/3$

Modern open LLM block (LLaMA / Mistral / Qwen)

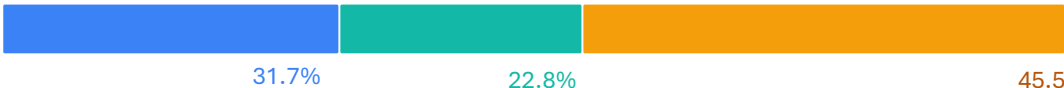
- **RMSNorm** $\approx 2d$ per layer
- **GQA / MQA attention** $= d^2(2 + 2h_{kv} / h_q)$
- **SwiGLU FFN** $= 3dd_{\text{ff}}$
- **Token embedding** $= Vd$
- **Logits head** $= Vd$
- **RoPE** $= 0$ trainable parameters
- **MoE FFN total** $= E \cdot 3dd_{\text{ff}}$, active $= k \cdot 3dd_{\text{ff}}$
- **Typical split at scale** attention $\approx 16\text{--}18\%$, FFN $\approx 78\text{--}82\%$

Main driver of the shift: reduced K/V projections + larger SwiGLU FFNs + untied large-vocab heads.

Representative breakdowns: dense decoder families

Bars show share of trainable parameters in embeddings + logits vs attention vs FFN. Layer norms and tiny routing terms are omitted from the bars.

■ Emb + logits ■ Attention ■ FFN / MoE

Model	Total params	Parameter share by component	Architecture note
GPT-2 117M	0.124B		tied logits
GPT-2 XL	1.56B		tied logits
GPT-3 175B*	174.6B		GPT-2-style count
LLaMA-1 7B	6.74B		SwiGLU / RoPE

Takeaway: once width and depth grow, dense GPT-style models are overwhelmingly an attention + FFN story, with FFN dominating by $\sim 2\times$ over attention.

Representative breakdowns: modern open LLMs

Bars show share of trainable parameters in embeddings + logits vs attention vs FFN. Layer norms and tiny routing terms are omitted from the bars.

■ Emb + logits ■ Attention ■ FFN / MoE

Model	Total params	Parameter share by component	Architecture note
Llama-2 70B	69.0B	 0.8% 17.5% 81.7%	GQA 64q / 8kv
Mistral 7B	7.24B	 3.6% 18.5% 77.8%	GQA + SWA
Llama-3 8B	8.03B	 13.1% 16.7% 70.2%	128k vocab
Qwen2.5 72B	72.7B	 3.4% 16.6% 80.0%	152k vocab
Mixtral 8x7B	46.7B total / 12.9B active	 0.6% 2.9% 96.5%	8 experts, top-2