Lecture 16: More on Large Language Models. RLHF

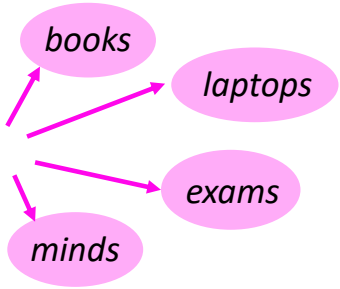
COMP579, Lecture 15

some slides are adapted from: https://www.3blue1brown.com/lessons/gpt

Recall: What is a language model?

- **Language Modeling** is the task of predicting what word comes next

the students opened their _____



- More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

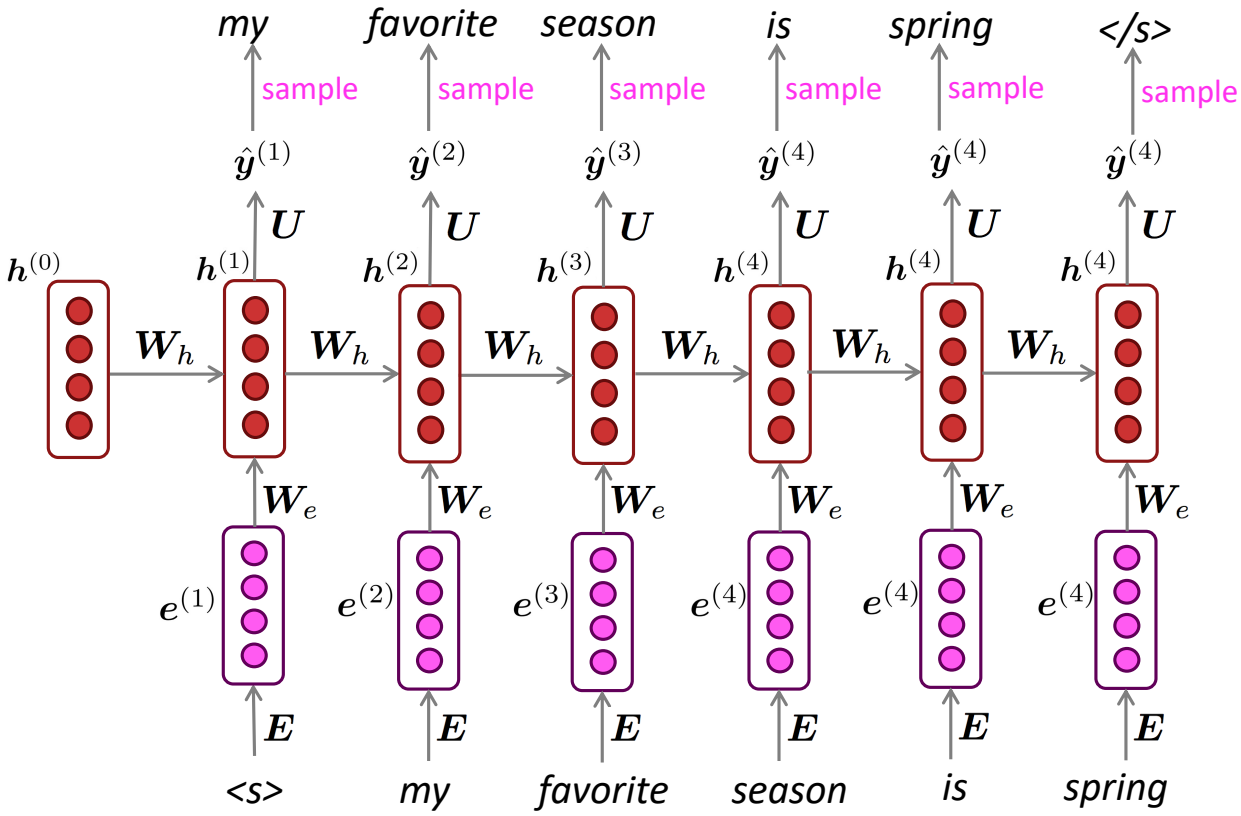
$$P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, ..., \boldsymbol{w}_{|V|}\}$

- A system that does this is called a **Language Model**

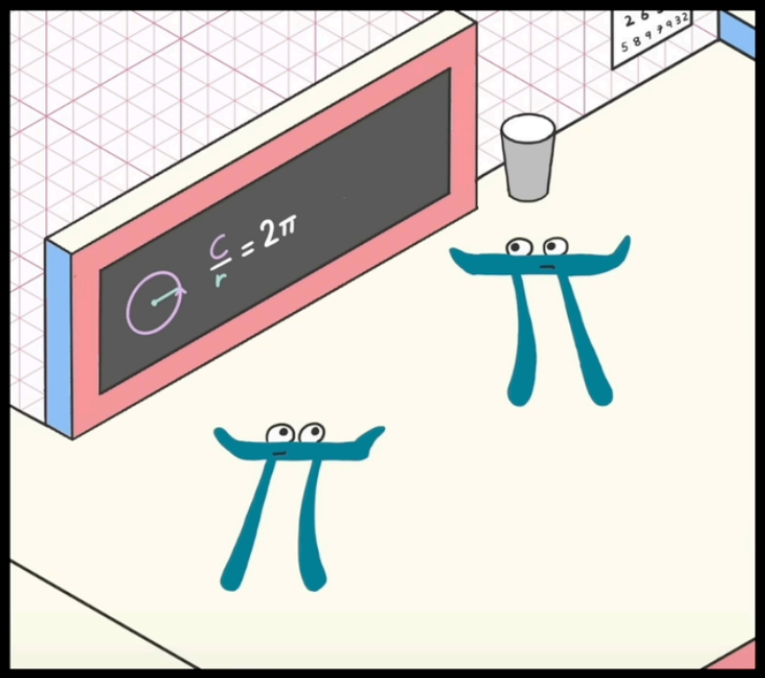
Recall: Generating text with RNNs

Just like an n-gram Language Model, you can use a RNN Language Model to generate text by repeated sampling. Sampled output becomes next step's input.



Transformer

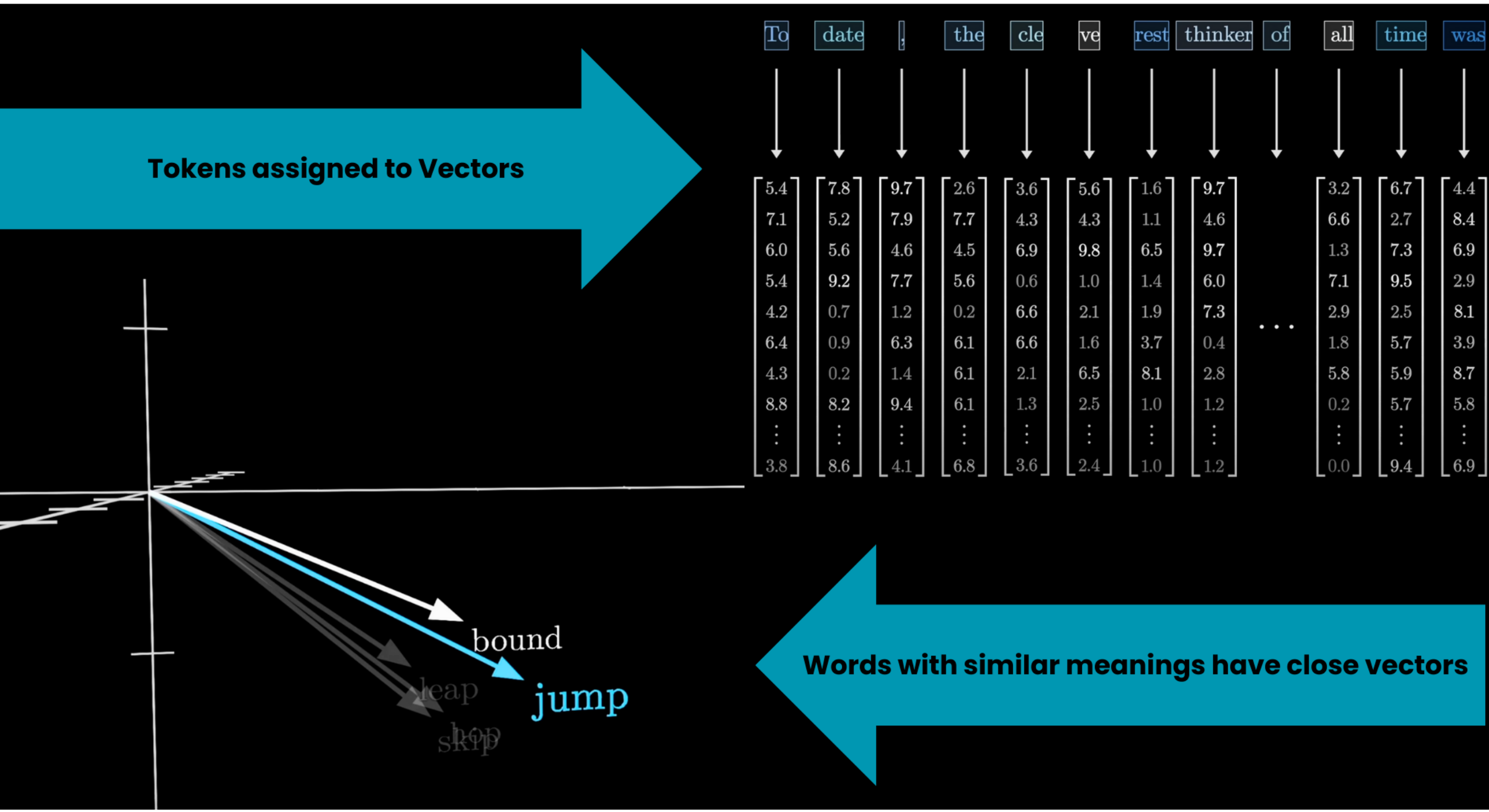
Behold, a wild pi creature,
foraging in its native _____



Transformer

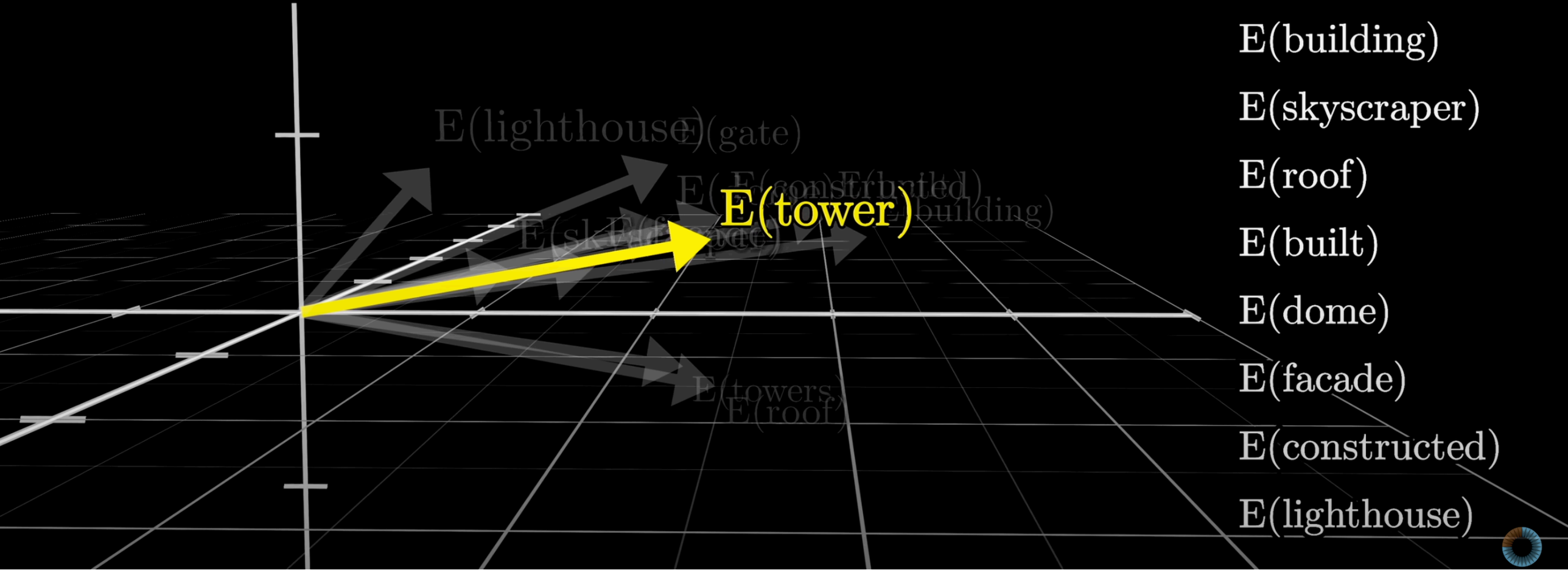
land	22%
forest	9%
country	5%
habitat	4%
forests	4%
soil	4%
territory	2%
woods	2%
lands	1%
waters	1%
woodland	1%
grass	1%
⋮	

Embeddings


Tokens assigned to Vectors
To
date
.
the
cle
ve
rest
thinker
of
all
time
was
5.4 7.1 6.0 5.4 4.2 6.4 4.3 8.8 3.8
7.8 5.2 5.6 9.2 0.7 0.9 0.2 8.2 8.6
9.7 7.9 4.6 7.7 1.2 6.3 1.4 9.4 4.1
2.6 7.7 4.5 5.6 0.2 6.1 6.1 6.1 6.8
3.6 4.3 6.9 0.6 6.6 6.6 2.1 1.3 3.6
5.6 4.3 9.8 1.0 2.1 1.6 6.5 2.5 2.4
1.6 1.1 6.5 1.4 1.9 3.7 8.1 1.0 1.0
9.7 4.6 9.7 6.0 7.3 0.4 2.8 1.2 1.2
3.2 6.6 1.3 7.1 2.9 1.8 5.8 0.2 0.0
6.7 2.7 7.3 9.5 2.5 5.7 5.9 5.7 9.4
4.4 8.4 6.9 2.9 8.1 3.9 8.7 5.8 6.9
bound
jump
leap
hop
skip
Words with similar meanings have close vectors

Embeddings: Similarity

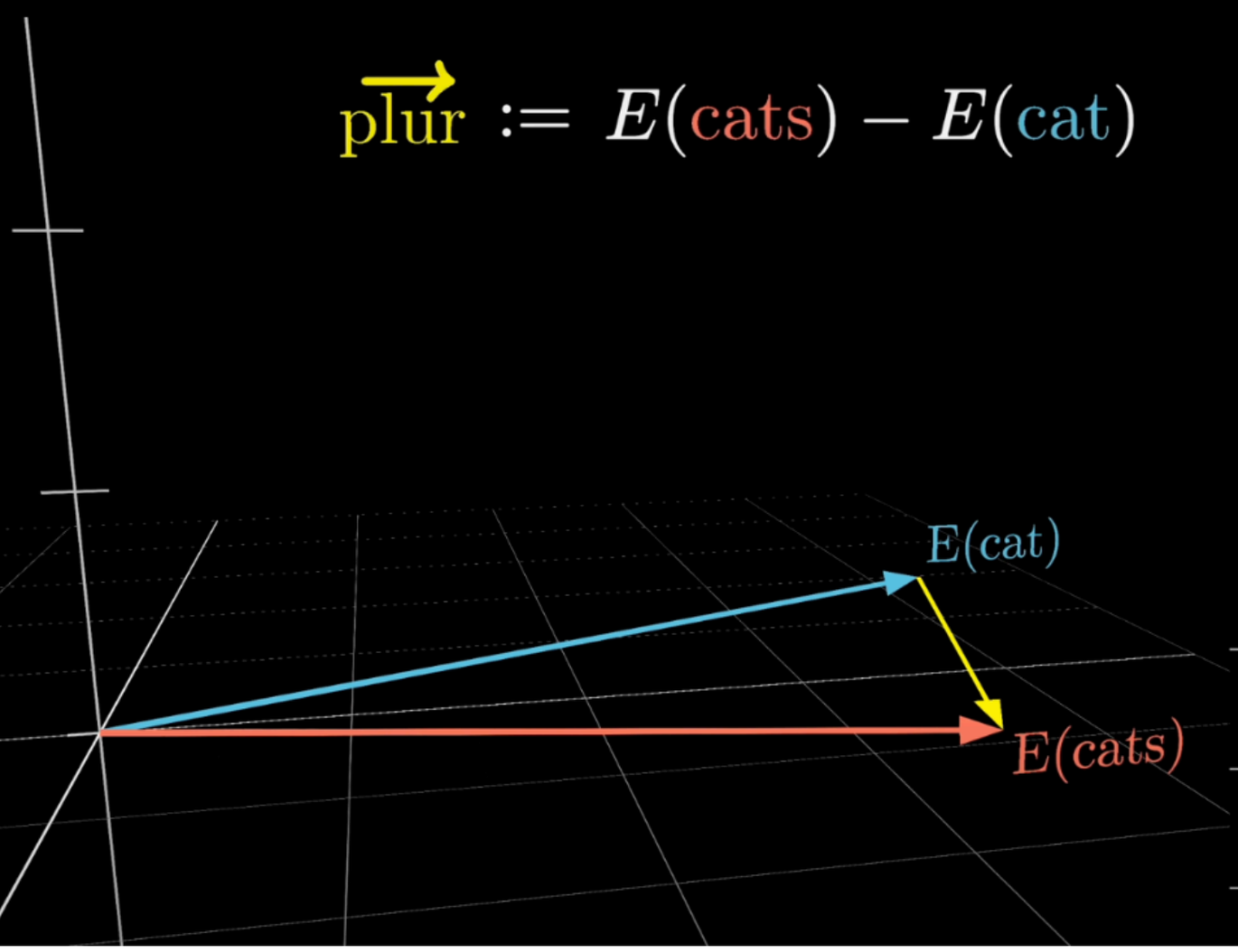
```
In [2]: import gensim.downloader  # You need to pip install gensim

In [3]: model = gensim.downloader.load("glove-wiki-gigaword-50")

In [4]: model["tower"]
```



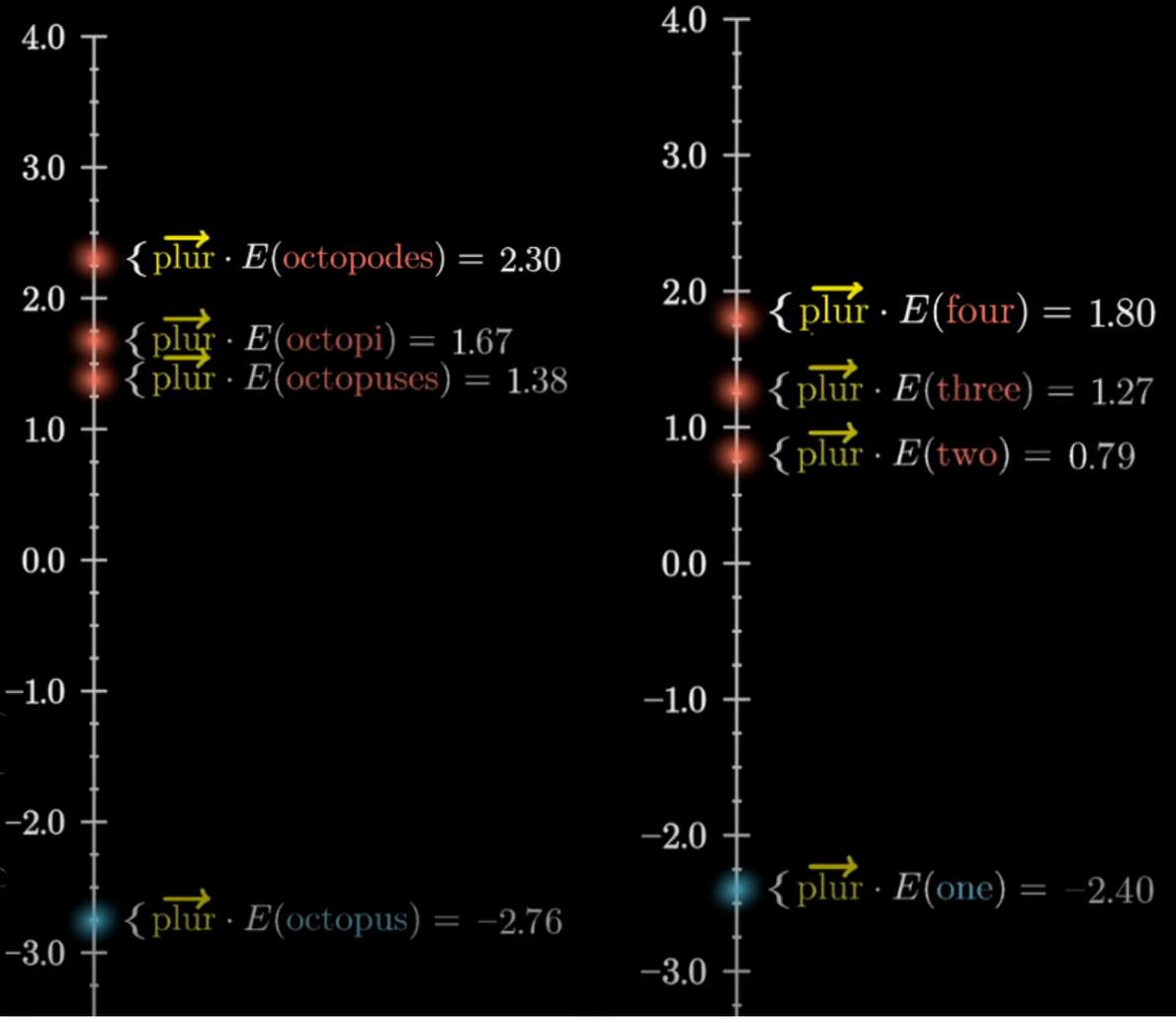
Embeddings: Directions

Plurality Direction:

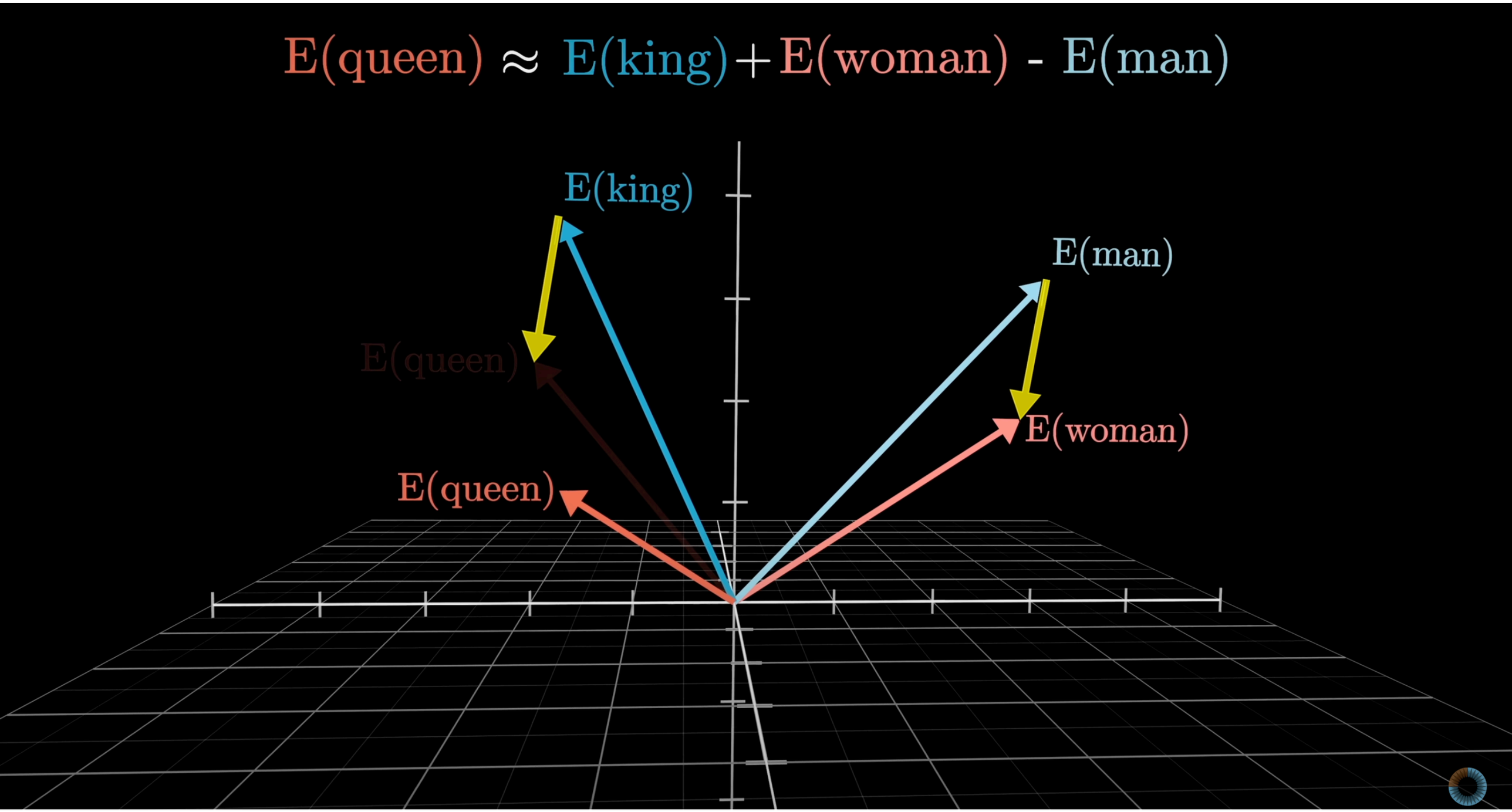
$$\overrightarrow{\text{plur}} := E(\text{cats}) - E(\text{cat})$$



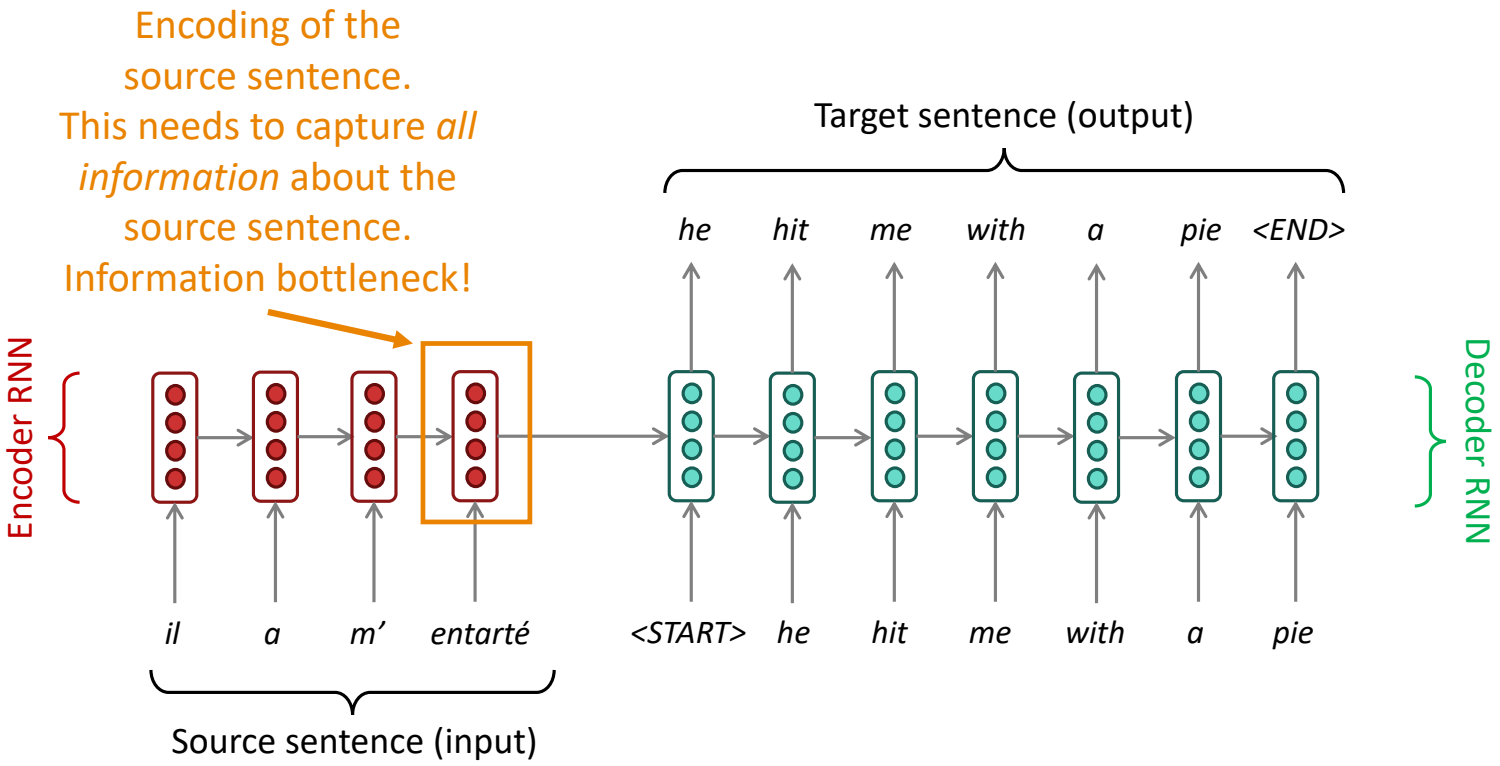
Plurality Dot Products:



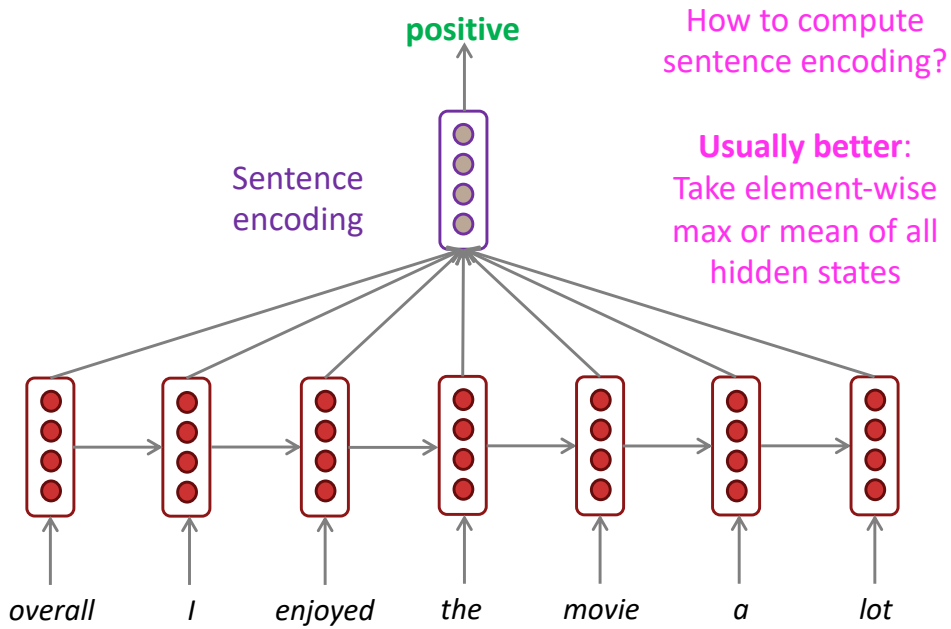
Embeddings: Directions



Recall: RNN Bottleneck problem



Pooling in RNNs



- Starting point: a *very* basic way of 'passing information from the encoder' is to *average*

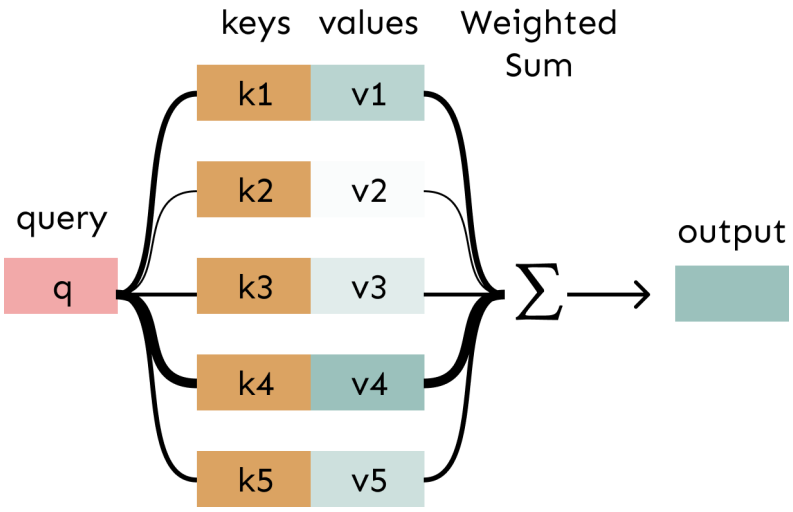
Solution: Attention

- On each step of decoding, use *direct connection to the encoder* to focus on a particular part of the sequence
- A bit like what humans do!
- Attention provides a solution to the bottleneck problem!

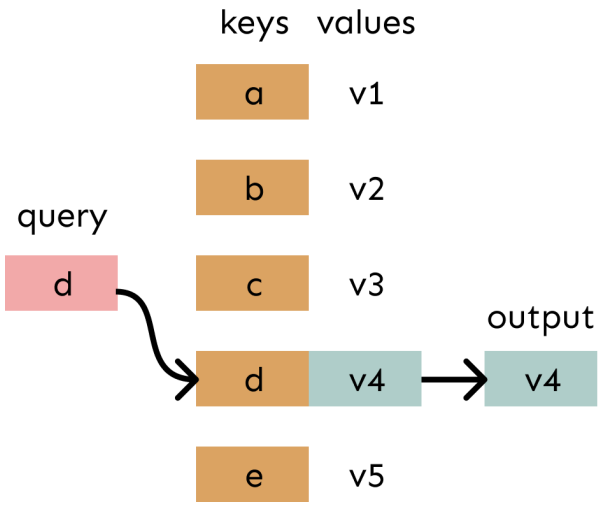
Attention is weighted averaging!

Attention is just a **weighted** average – this is very powerful if the weights are learned!

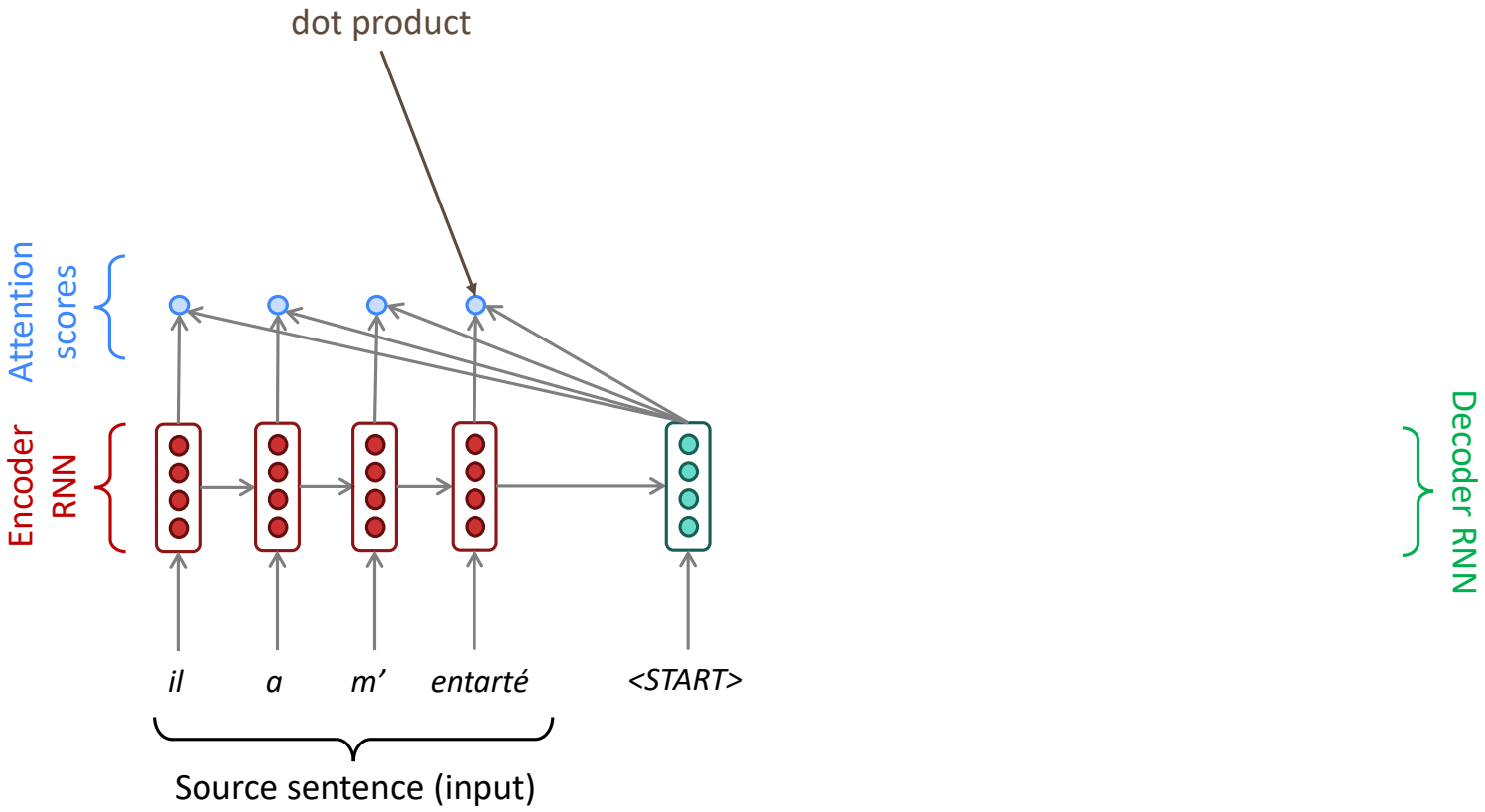
In **attention**, the query matches all keys *softly*, to a weight between 0 and 1. The keys' values are multiplied by the weights and summed.



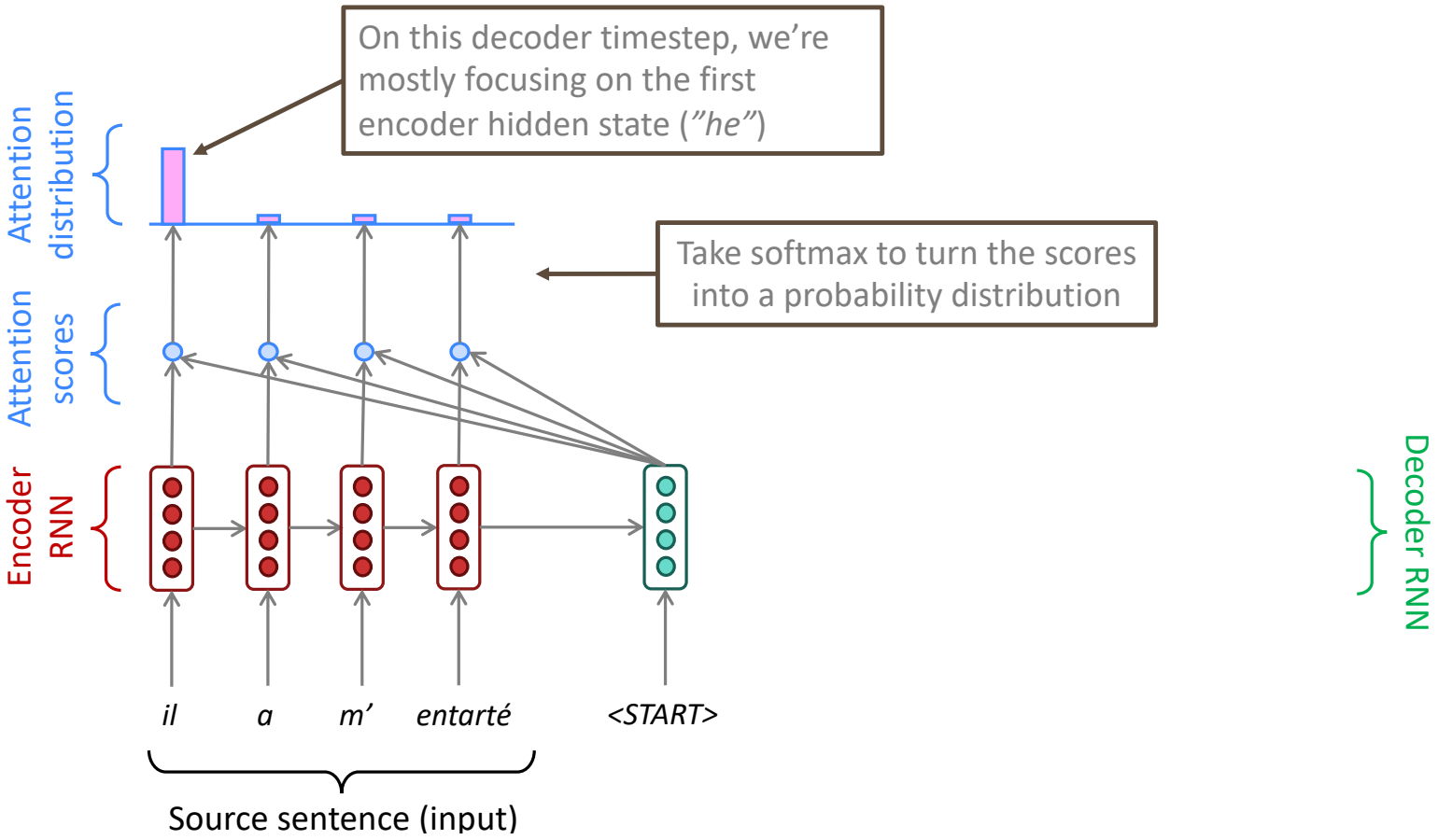
In a **lookup table**, we have a table of keys that map to values. The query matches one of the keys, returning its value.



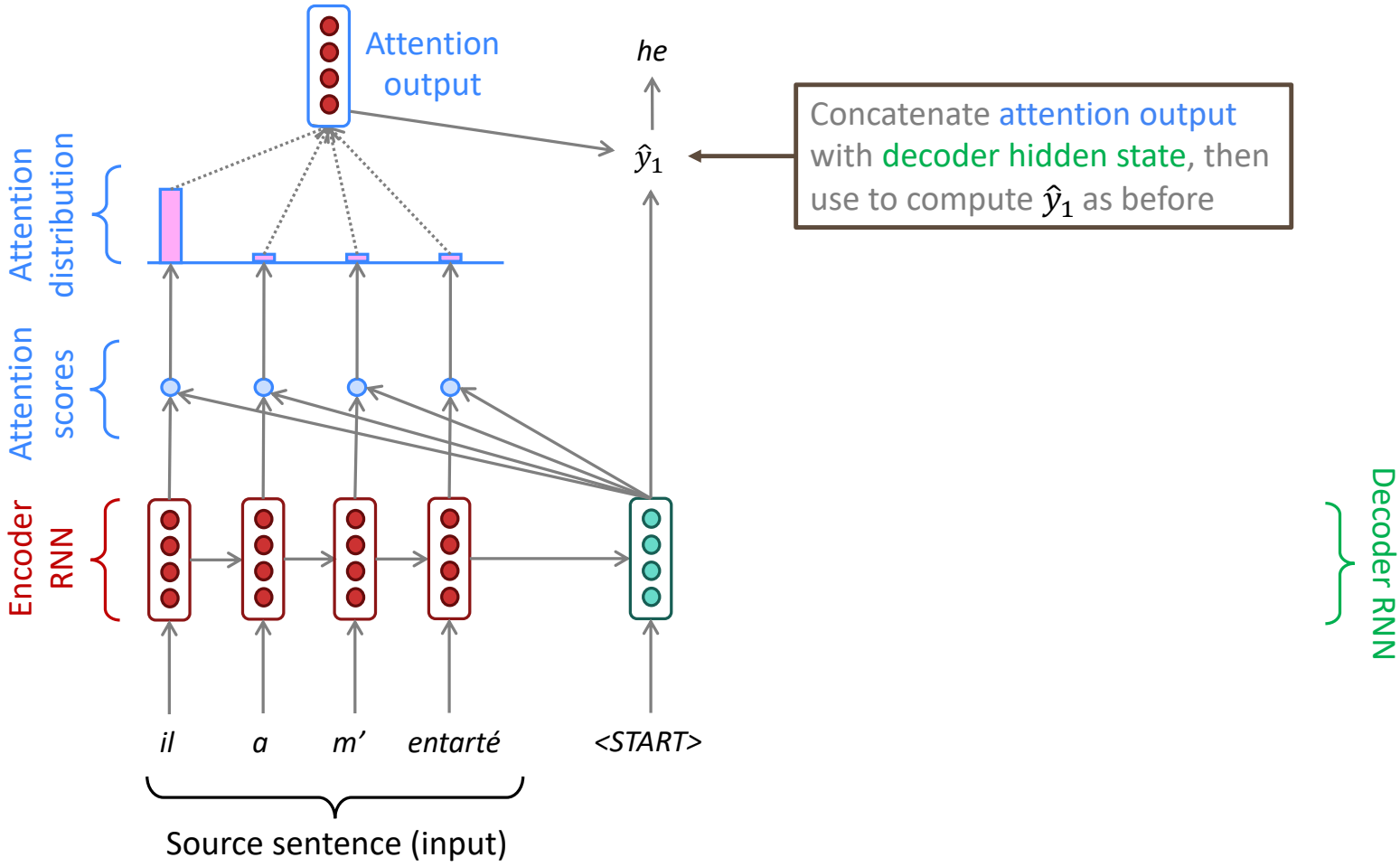
Using dot products



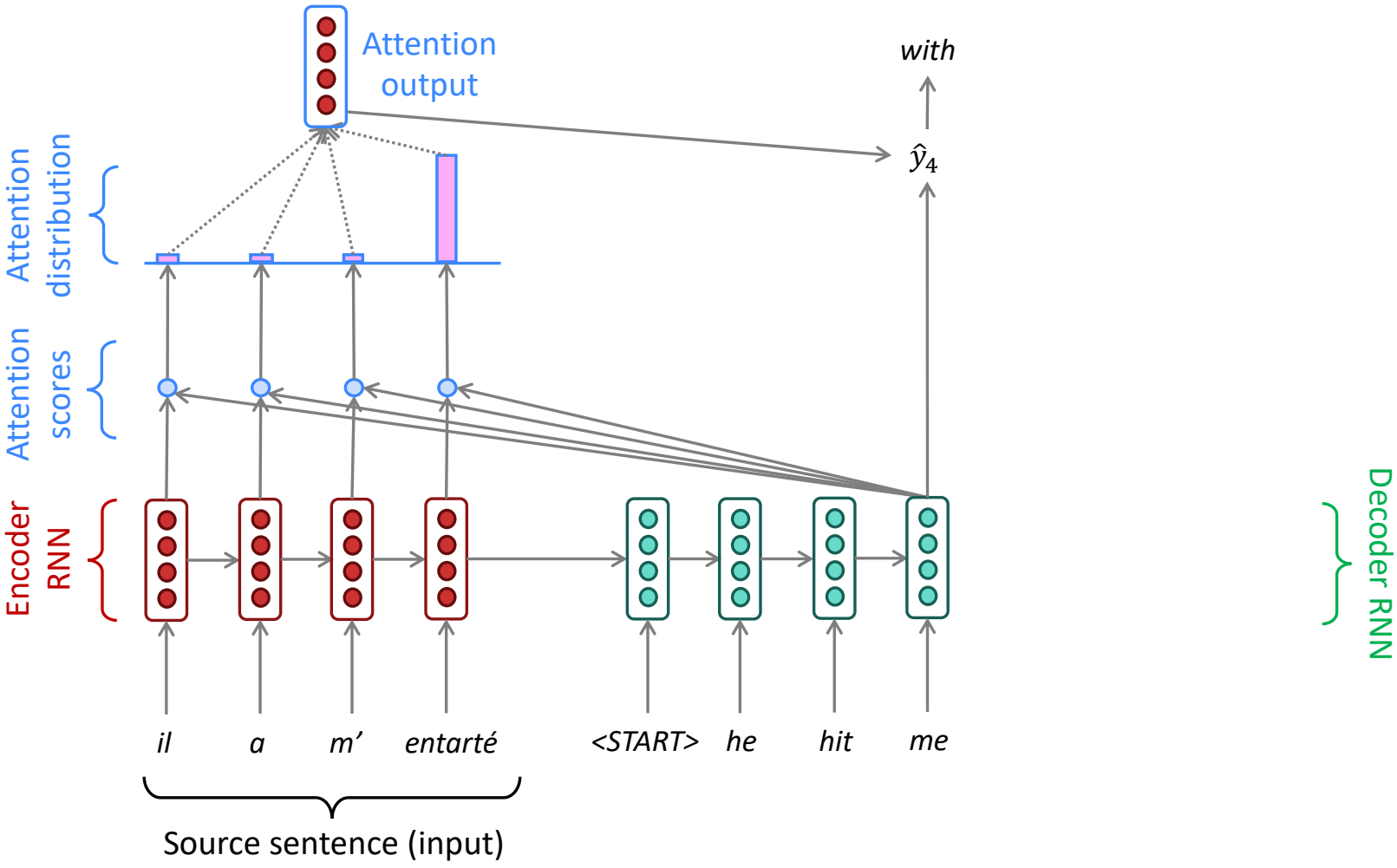
Using softmax for aggregation



Putting it all together



Attention example (continued)



Attention more formally

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores $\boldsymbol{e}^t$ for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output $\boldsymbol{a}_t$ with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

From translation to language generation: Self-attention

Let $\boldsymbol{w}_{1:n}$ be a sequence of words in vocabulary V, like *Zuko made his uncle tea*.

For each $\boldsymbol{w}_i$, let $\boldsymbol{x}_i = E\boldsymbol{w}_i$, where $E \in \mathbb{R}^{d \times |V|}$ is an embedding matrix.

1. Transform each word embedding with weight matrices Q, K, V, each in $\mathbb{R}^{d \times d}$

$\boldsymbol{q}_i = Q\boldsymbol{x}_i$ (queries) $\quad \boldsymbol{k}_i = K\boldsymbol{x}_i$ (keys) $\quad \boldsymbol{v}_i = V\boldsymbol{x}_i$ (values)

2. Compute pairwise similarities between keys and queries; normalize with softmax

$$\boldsymbol{e}_{ij} = \boldsymbol{q}_i^\top \boldsymbol{k}_j \qquad \boldsymbol{\alpha}_{ij} = \frac{\exp(\boldsymbol{e}_{ij})}{\sum_{j'} \exp(\boldsymbol{e}_{ij'})}$$

3. Compute output for each word as weighted sum of values

$$\boldsymbol{o}_i = \sum_j \boldsymbol{\alpha}_{ij} \boldsymbol{v}_i$$

$$\boldsymbol{o}_i = \sum_j \boldsymbol{\alpha}_{ij} \boldsymbol{v}_i$$

Attention blueprint

- We have some *values* $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and a *query* $\boldsymbol{s} \in \mathbb{R}^{d_2}$

- Attention always involves:
  1. Computing the *attention scores* $\boldsymbol{e} \in \mathbb{R}^N$
  2. Taking softmax to get *attention distribution* α:

$$\alpha = \text{softmax}(\boldsymbol{e}) \in \mathbb{R}^N$$

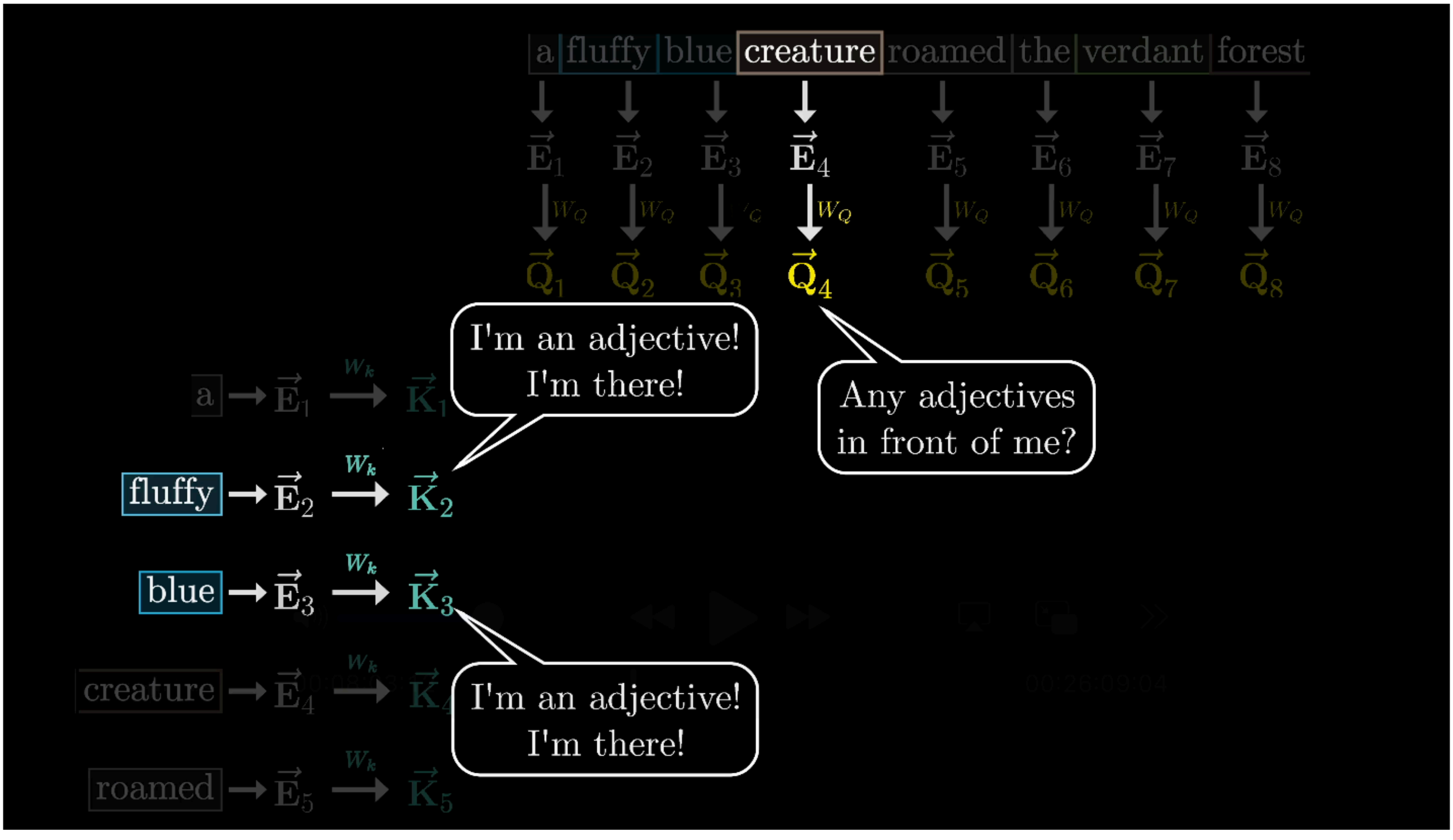
There are multiple ways to do this

  3. Using attention distribution to take weighted sum of values:

$$\boldsymbol{a} = \sum_{i=1}^{N} \alpha_i \boldsymbol{h}_i \in \mathbb{R}^{d_1}$$

  thus obtaining the *attention output* $\boldsymbol{a}$ (sometimes called the *context vector*)

Attention: Query, Key



Attention: Query, Key

	Q_1	Q_2	Q_3	Q_4	Q_5	$\cdots$	Q_n	
K_1	$Q_1 \cdot K_1$	$Q_2 \cdot K_1$	$Q_3 \cdot K_1$	$Q_4 \cdot K_1$	$Q_5 \cdot K_1$	$\cdots$	$Q_n \cdot K_1$	
K_2	$Q_1 \cdot K_2$	$Q_2 \cdot K_2$	$Q_3 \cdot K_2$	$Q_4 \cdot K_2$	$Q_5 \cdot K_2$	$\cdots$	$Q_n \cdot K_2$	
K_3	$Q_1 \cdot K_3$	$Q_2 \cdot K_3$	$Q_3 \cdot K_3$	$Q_4 \cdot K_3$	$Q_5 \cdot K_3$	$\cdots$	$Q_n \cdot K_3$	
K_4	$Q_1 \cdot K_4$	$Q_2 \cdot K_4$	$Q_3 \cdot K_4$	$Q_4 \cdot K_4$	$Q_5 \cdot K_4$	$\cdots$	$Q_n \cdot K_4$	
K_5	$Q_1 \cdot K_5$	$Q_2 \cdot K_5$	$Q_3 \cdot K_5$	$Q_4 \cdot K_5$	$Q_5 \cdot K_5$	$\cdots$	$Q_n \cdot K_5$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	

Unnormalized Attention Pattern

+3.53	+0.80	+1.96	+4.48	+3.74	−1.95
−∞	−0.30	−0.21	+0.82	+0.29	+2.91
−∞	−∞	+0.89	+0.67	+2.99	−0.41
−∞	−∞	−∞	+1.31	+1.73	−1.48
−∞	−∞	−∞	−∞	+3.07	+2.94
−∞	−∞	−∞	−∞	−∞	+0.31

softmax

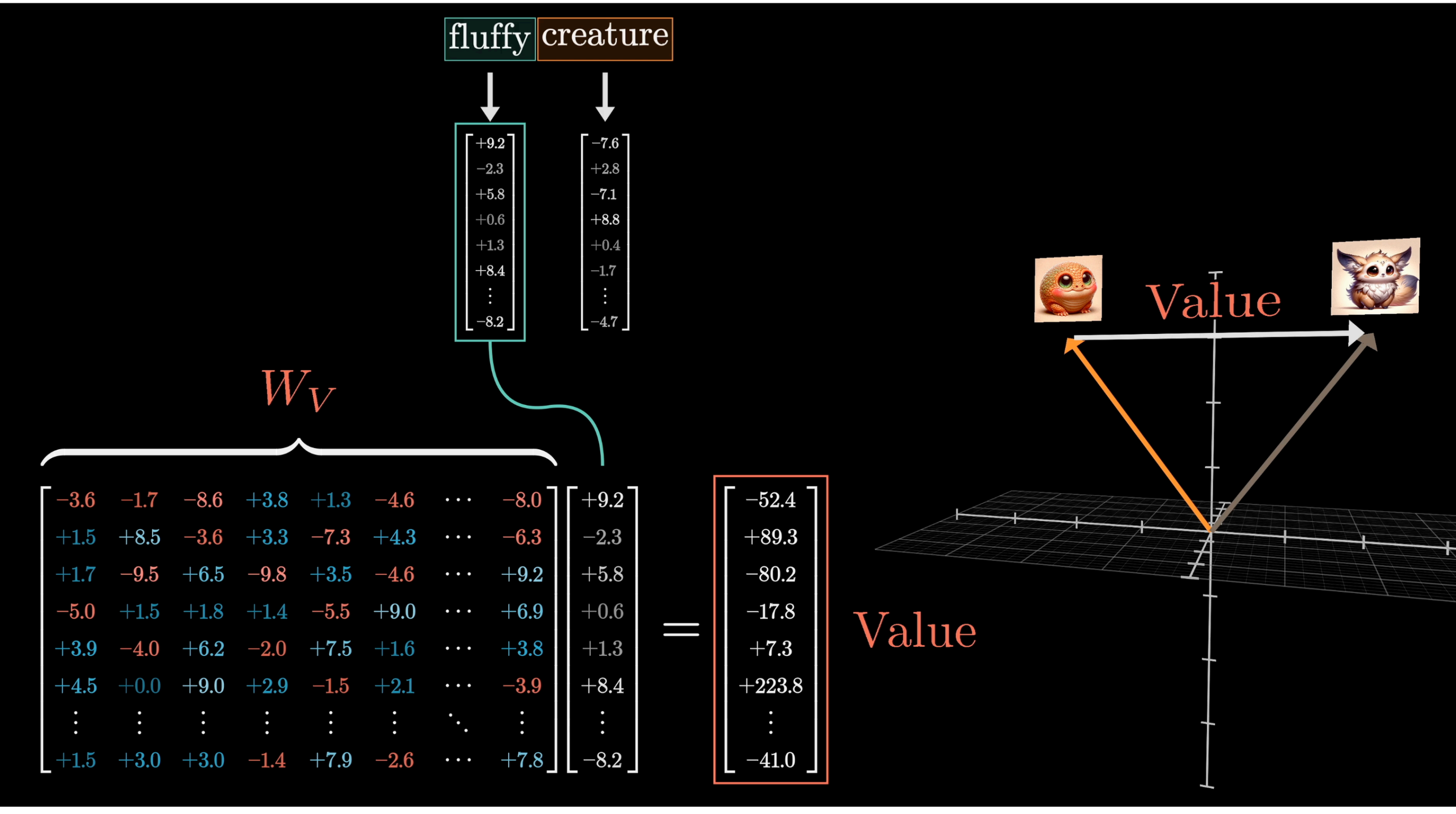
Normalized Attention Pattern

1.00	0.75	0.69	0.92	0.46	0.00
0.00	0.25	0.08	0.02	0.01	0.46
0.00	0.00	0.24	0.02	0.22	0.02
0.00	0.00	0.00	0.04	0.06	0.01
0.00	0.00	0.00	0.00	0.24	0.48
0.00	0.00	0.00	0.00	0.00	0.03

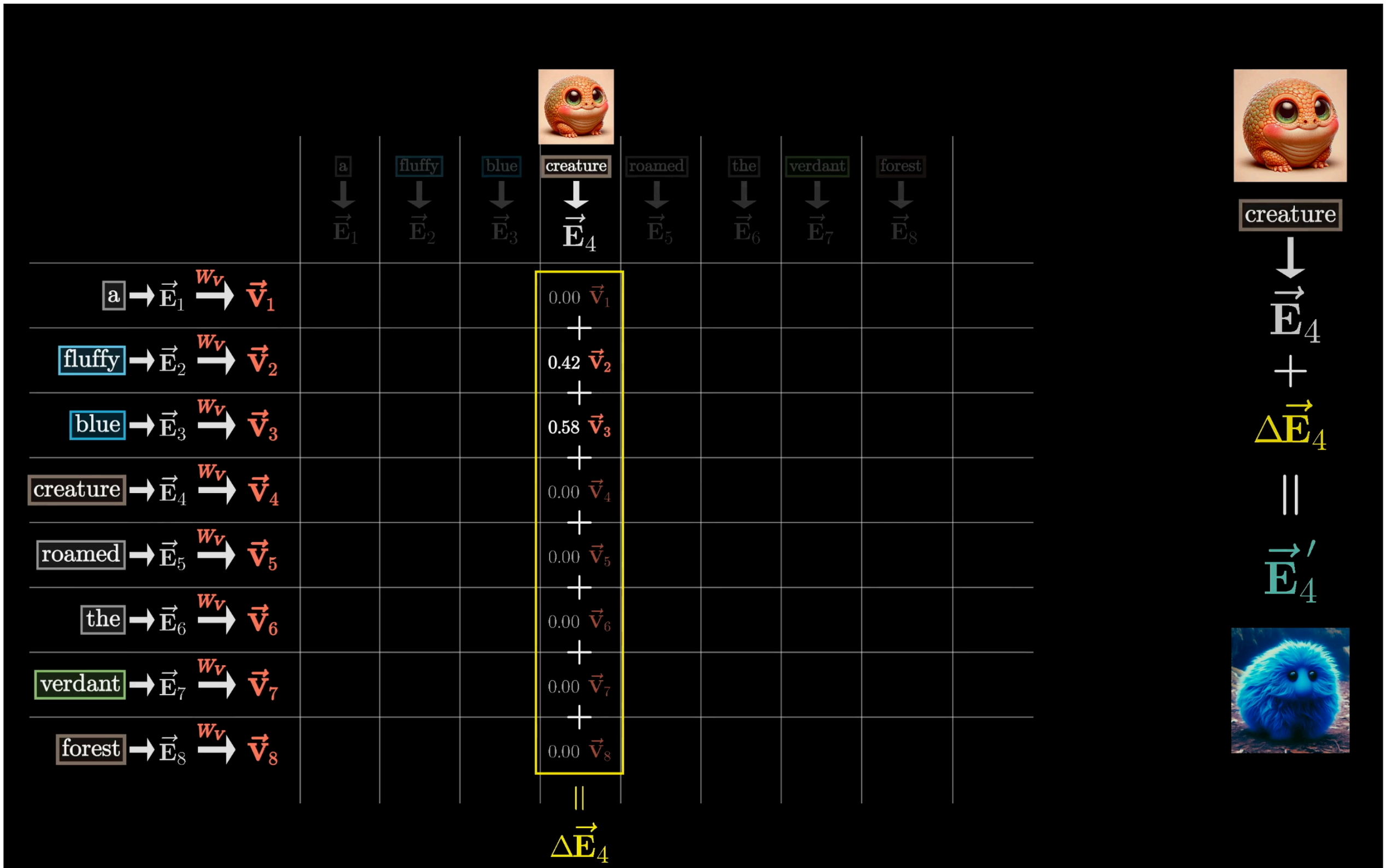
Attention: Query, Key

	a	fluffy	blue	creature	roamed	the	verdant	forest
	$\vec{E}_1$	$\vec{E}_2$	$\vec{E}_3$	$\vec{E}_4$	$\vec{E}_5$	$\vec{E}_6$	$\vec{E}_7$	$\vec{E}_8$
	W_Q	W_Q	W_Q	W_Q	W_Q	W_Q	W_Q	W_Q
	$\vec{Q}_1$	$\vec{Q}_2$	$\vec{Q}_3$	$\vec{Q}_4$	$\vec{Q}_5$	$\vec{Q}_6$	$\vec{Q}_7$	$\vec{Q}_8$
a → $\vec{E}_1$ $\xrightarrow{W_k}$ $\vec{K}_1$	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
fluffy → $\vec{E}_2$ $\xrightarrow{W_k}$ $\vec{K}_2$	0.00	1.00	0.00	0.42	0.00	0.00	0.00	0.00
blue → $\vec{E}_3$ $\xrightarrow{W_k}$ $\vec{K}_3$	0.00	0.00	1.00	0.58	0.00	0.00	0.00	0.00
creature → $\vec{E}_4$ $\xrightarrow{W_k}$ $\vec{K}_4$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
roamed → $\vec{E}_5$ $\xrightarrow{W_k}$ $\vec{K}_5$	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00
the → $\vec{E}_6$ $\xrightarrow{W_k}$ $\vec{K}_6$	0.00	0.00	0.00	0.00	0.99	1.00	0.00	0.00
verdant → $\vec{E}_7$ $\xrightarrow{W_k}$ $\vec{K}_7$	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00
forest → $\vec{E}_8$ $\xrightarrow{W_k}$ $\vec{K}_8$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

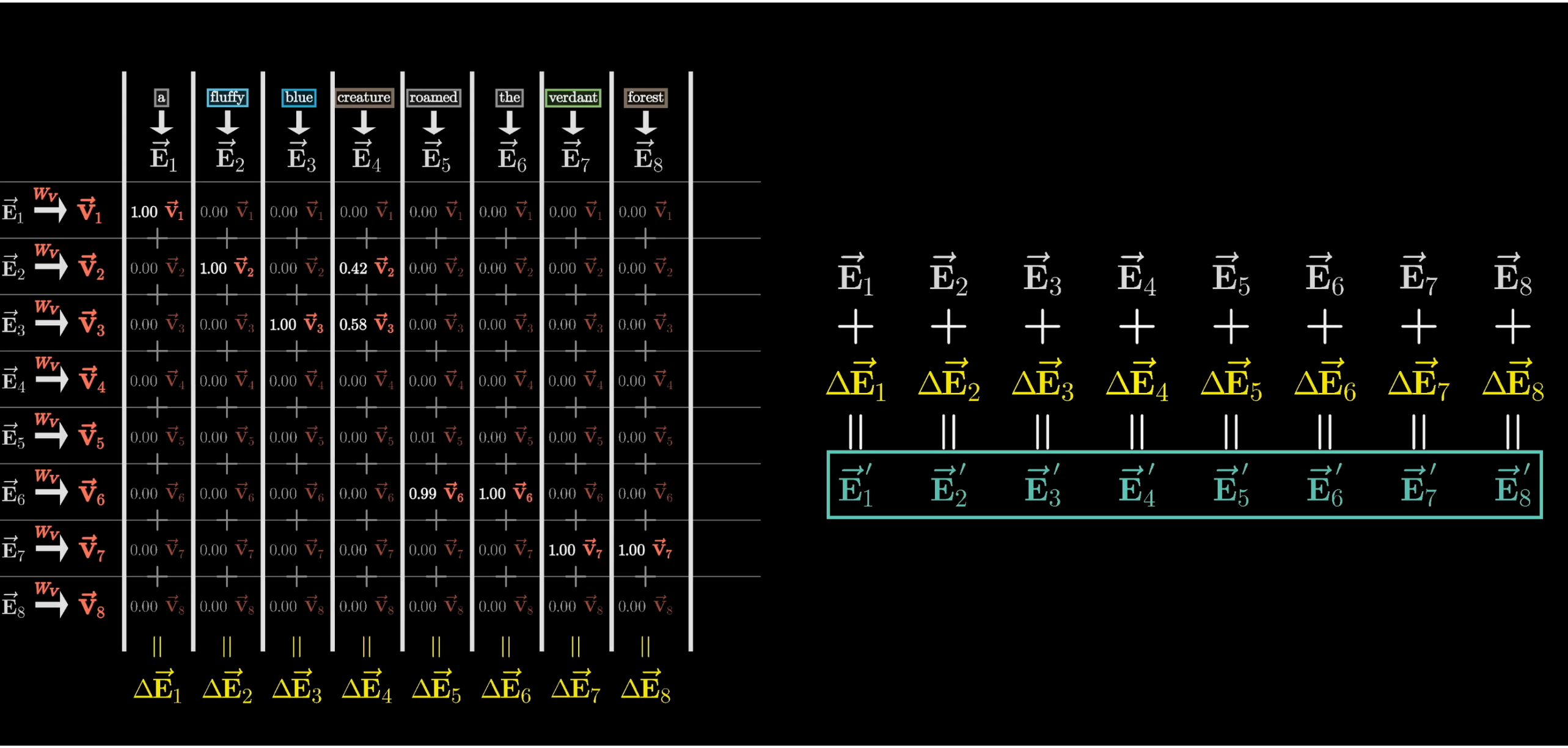
Attention: Value

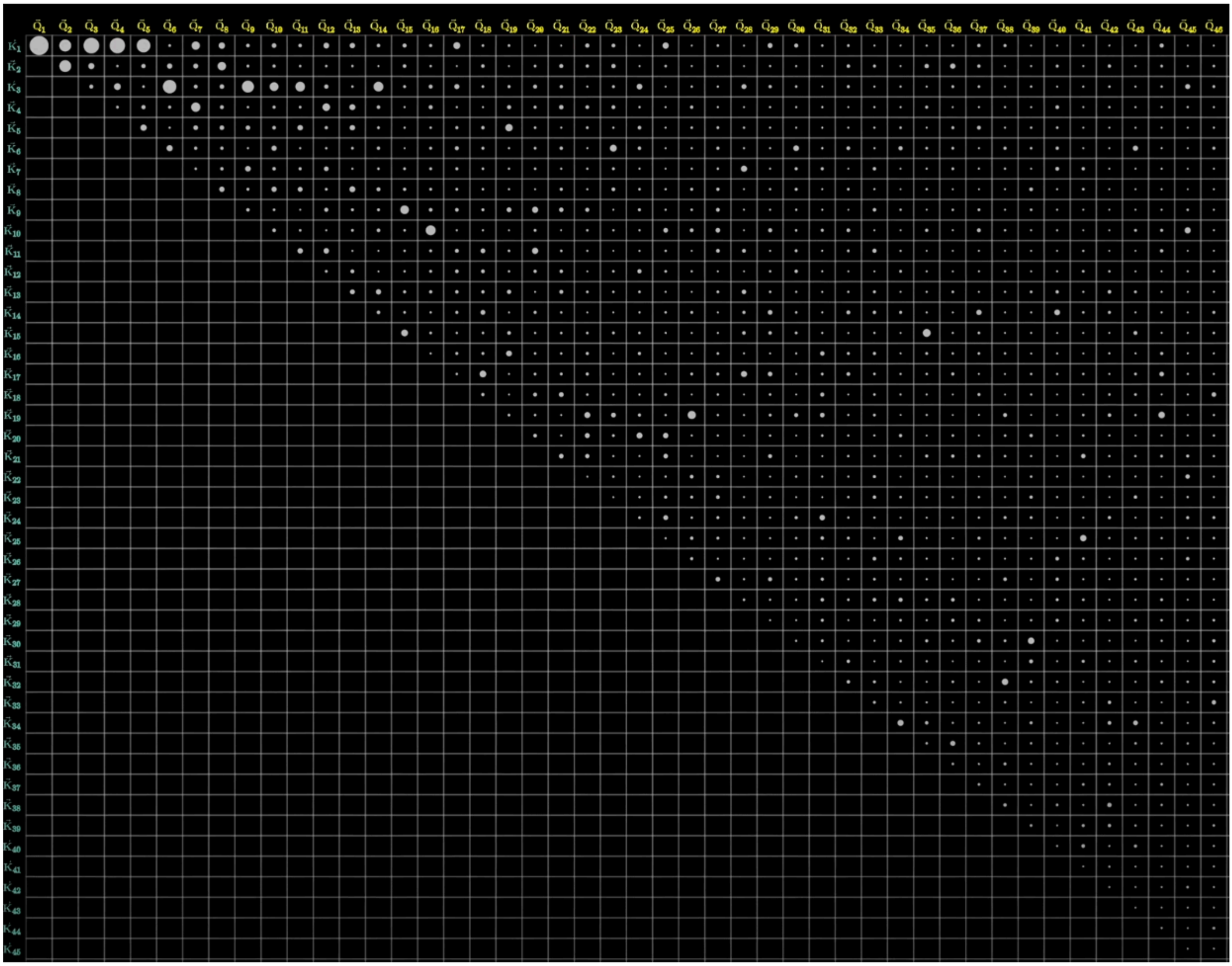


Attention: Value

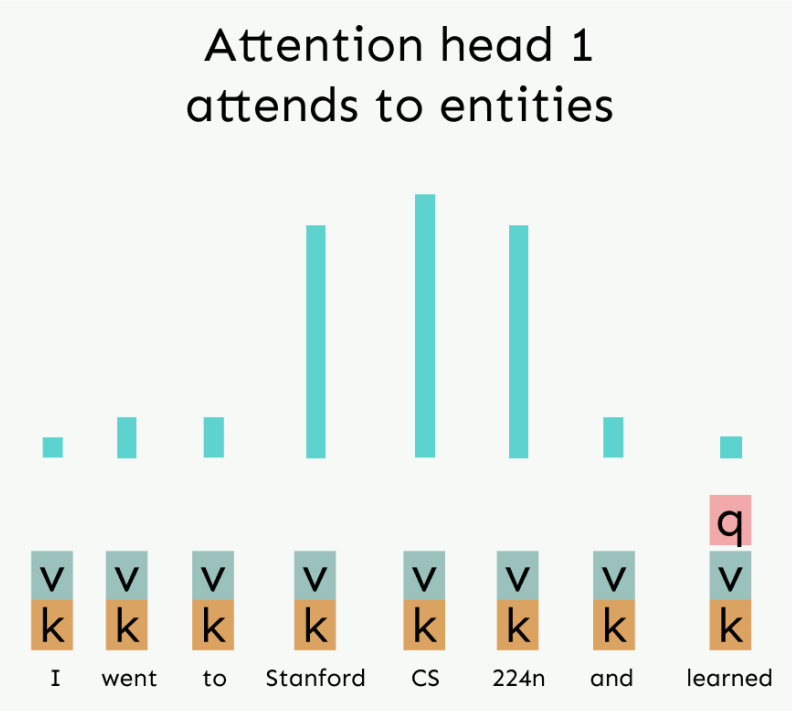


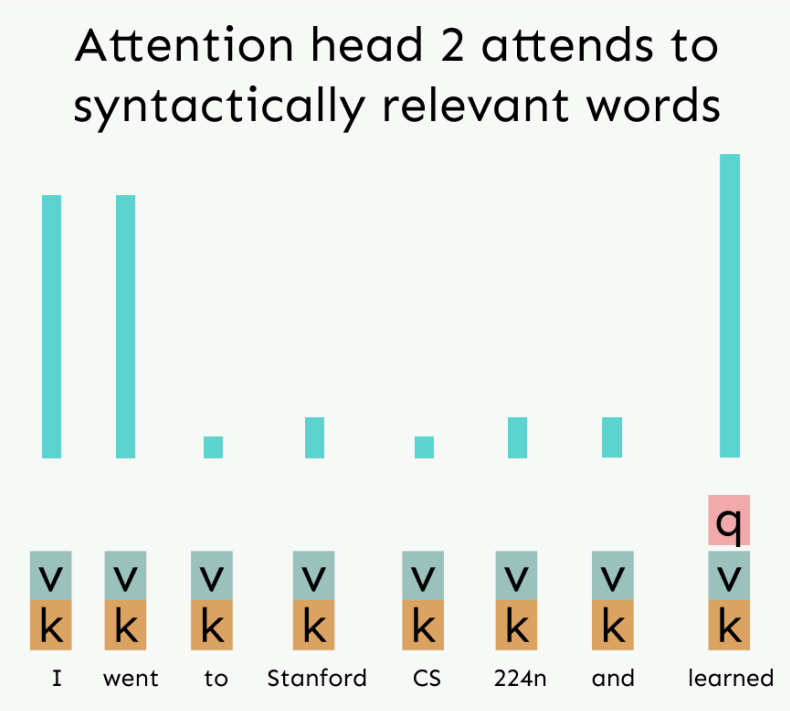
Attention: Value




$\vec{Q}_1$ $\vec{Q}_2$ $\vec{Q}_3$ $\vec{Q}_4$ $\vec{Q}_5$ $\vec{Q}_6$ $\vec{Q}_7$ $\vec{Q}_8$ $\vec{Q}_9$ $\vec{Q}_{10}$ $\vec{Q}_{11}$ $\vec{Q}_{12}$ $\vec{Q}_{13}$ $\vec{Q}_{14}$ $\vec{Q}_{15}$ $\vec{Q}_{16}$ $\vec{Q}_{17}$ $\vec{Q}_{18}$ $\vec{Q}_{19}$ $\vec{Q}_{20}$ $\vec{Q}_{21}$ $\vec{Q}_{22}$ $\vec{Q}_{23}$ $\vec{Q}_{24}$ $\vec{Q}_{25}$ $\vec{Q}_{26}$ $\vec{Q}_{27}$ $\vec{Q}_{28}$ $\vec{Q}_{29}$ $\vec{Q}_{30}$ $\vec{Q}_{31}$ $\vec{Q}_{32}$ $\vec{Q}_{33}$ $\vec{Q}_{34}$ $\vec{Q}_{35}$ $\vec{Q}_{36}$ $\vec{Q}_{37}$ $\vec{Q}_{38}$ $\vec{Q}_{39}$ $\vec{Q}_{40}$ $\vec{Q}_{41}$ $\vec{Q}_{42}$ $\vec{Q}_{43}$ $\vec{Q}_{44}$ $\vec{Q}_{45}$ $\vec{Q}_{46}$
$\vec{K}_1$ $\vec{K}_2$ $\vec{K}_3$ $\vec{K}_4$ $\vec{K}_5$ $\vec{K}_6$ $\vec{K}_7$ $\vec{K}_8$ $\vec{K}_9$ $\vec{K}_{10}$ $\vec{K}_{11}$ $\vec{K}_{12}$ $\vec{K}_{13}$ $\vec{K}_{14}$ $\vec{K}_{15}$ $\vec{K}_{16}$ $\vec{K}_{17}$ $\vec{K}_{18}$ $\vec{K}_{19}$ $\vec{K}_{20}$ $\vec{K}_{21}$ $\vec{K}_{22}$ $\vec{K}_{23}$ $\vec{K}_{24}$ $\vec{K}_{25}$ $\vec{K}_{26}$ $\vec{K}_{27}$ $\vec{K}_{28}$ $\vec{K}_{29}$ $\vec{K}_{30}$ $\vec{K}_{31}$ $\vec{K}_{32}$ $\vec{K}_{33}$ $\vec{K}_{34}$ $\vec{K}_{35}$ $\vec{K}_{36}$ $\vec{K}_{37}$ $\vec{K}_{38}$ $\vec{K}_{39}$ $\vec{K}_{40}$ $\vec{K}_{41}$ $\vec{K}_{42}$ $\vec{K}_{43}$ $\vec{K}_{44}$ $\vec{K}_{45}$

Multihead attention





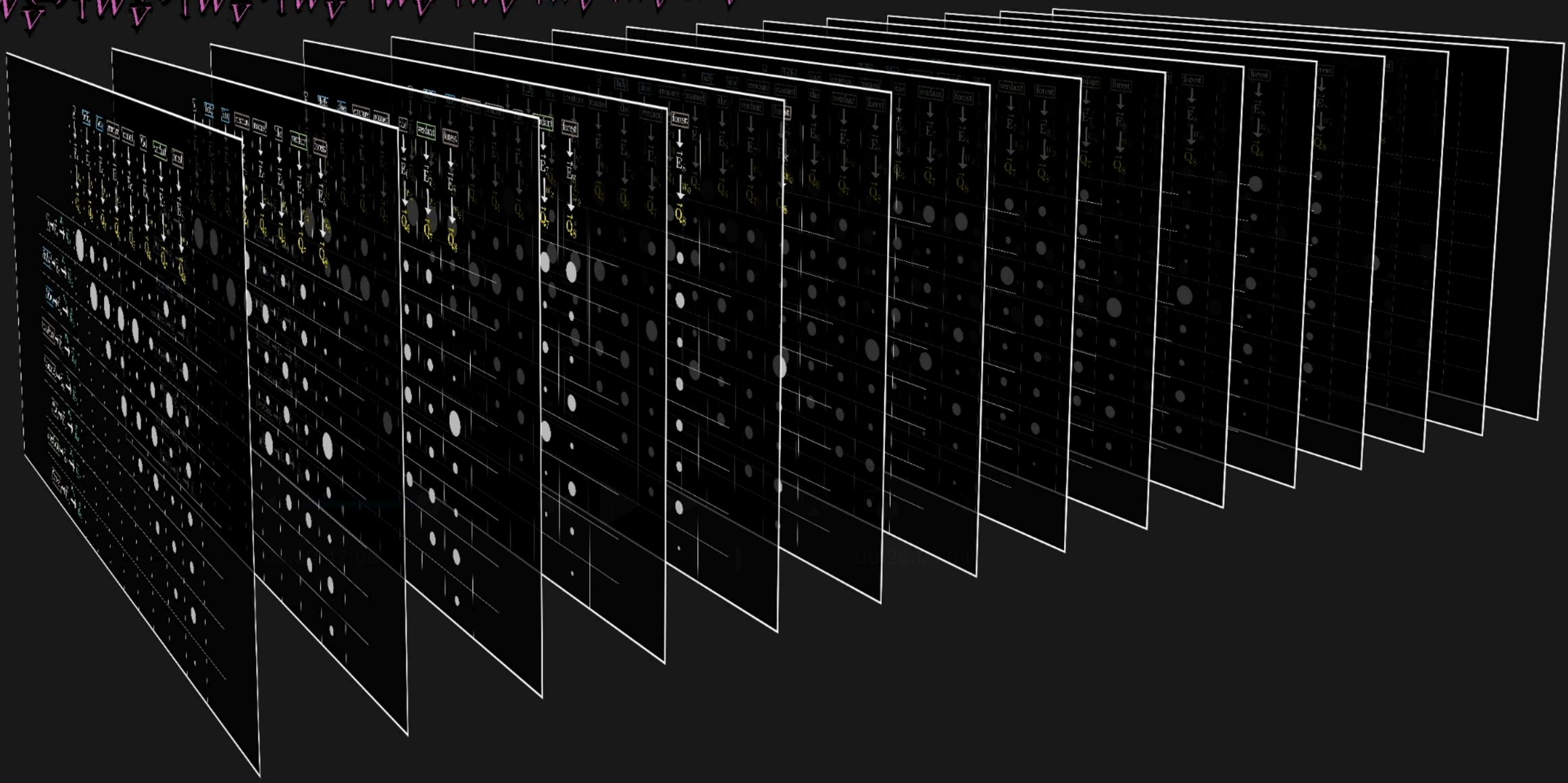
Multi-headed attention

$$W_Q^{(1)}\ W_Q^{(2)}\ W_Q^{(3)}\ W_Q^{(4)}\ W_Q^{(5)}\ W_Q^{(6)}\ W_Q^{(7)}\ W_Q^{(8)}\ W_Q^{(9)}\ \cdots$$

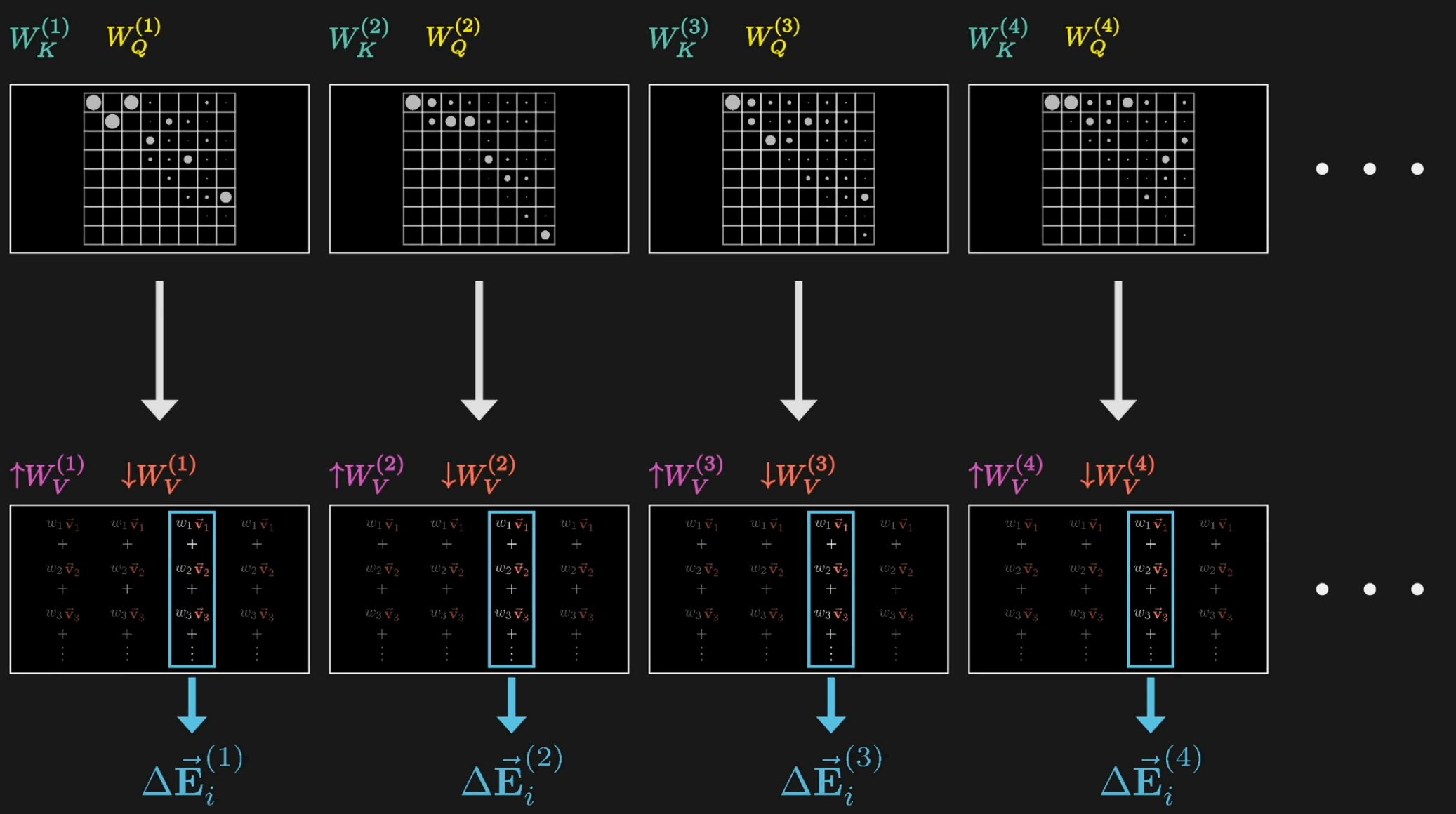
$$W_K^{(1)}\ W_K^{(2)}\ W_K^{(3)}\ W_K^{(4)}\ W_K^{(5)}\ W_K^{(6)}\ W_K^{(7)}\ W_K^{(8)}\ W_K^{(9)}\ \cdots$$

$$\downarrow W_V^{(1)}\ \downarrow W_V^{(2)}\ \downarrow W_V^{(3)}\ \downarrow W_V^{(4)}\ \downarrow W_V^{(5)}\ \downarrow W_V^{(6)}\ \downarrow W_V^{(7)}\ \downarrow W_V^{(8)}\ \downarrow W_V^{(9)}\ \cdots$$

$$\uparrow W_V^{(1)}\ \uparrow W_V^{(2)}\ \uparrow W_V^{(3)}\ \uparrow W_V^{(4)}\ \uparrow W_V^{(5)}\ \uparrow W_V^{(6)}\ \uparrow W_V^{(7)}\ \uparrow W_V^{(8)}\ \uparrow W_V^{(9)}\ \cdots$$



Multi-headed Attention

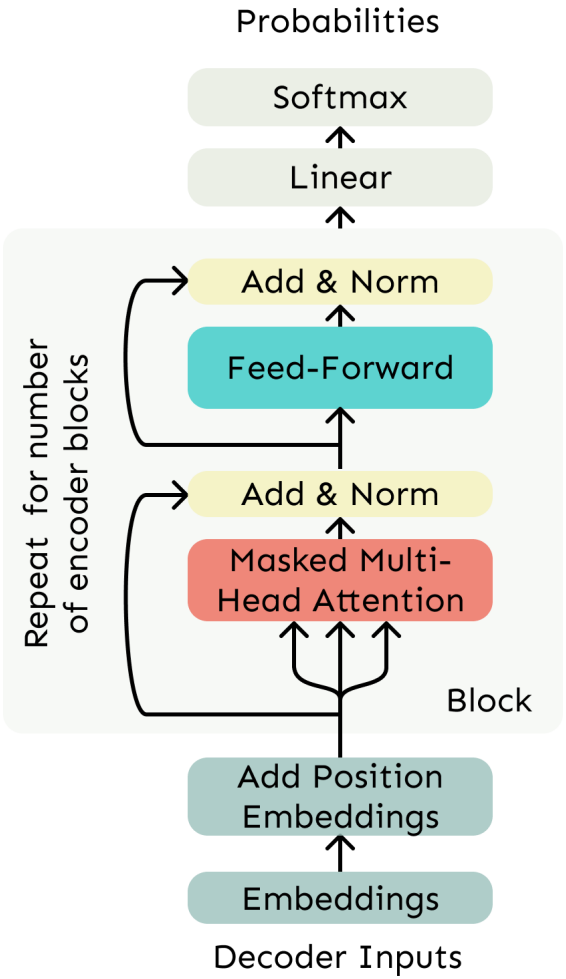


Original embedding $\boxed{\vec{\mathbf{E}}_i} + \Delta\vec{\mathbf{E}}_i^{(1)} + \Delta\vec{\mathbf{E}}_i^{(2)} + \Delta\vec{\mathbf{E}}_i^{(3)} + \Delta\vec{\mathbf{E}}_i^{(4)} + \cdots$

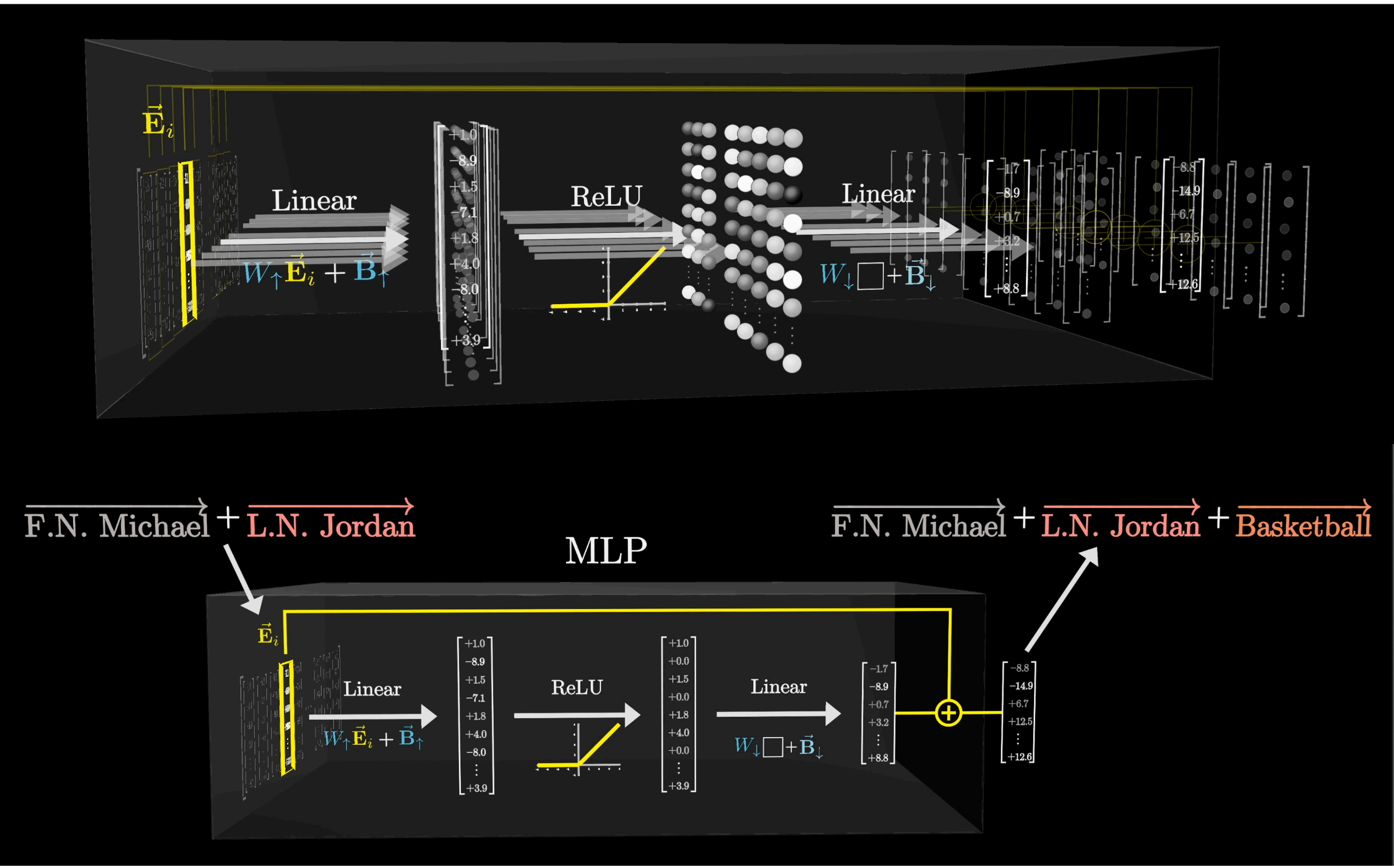
Transformer decoder

The Transformer Decoder

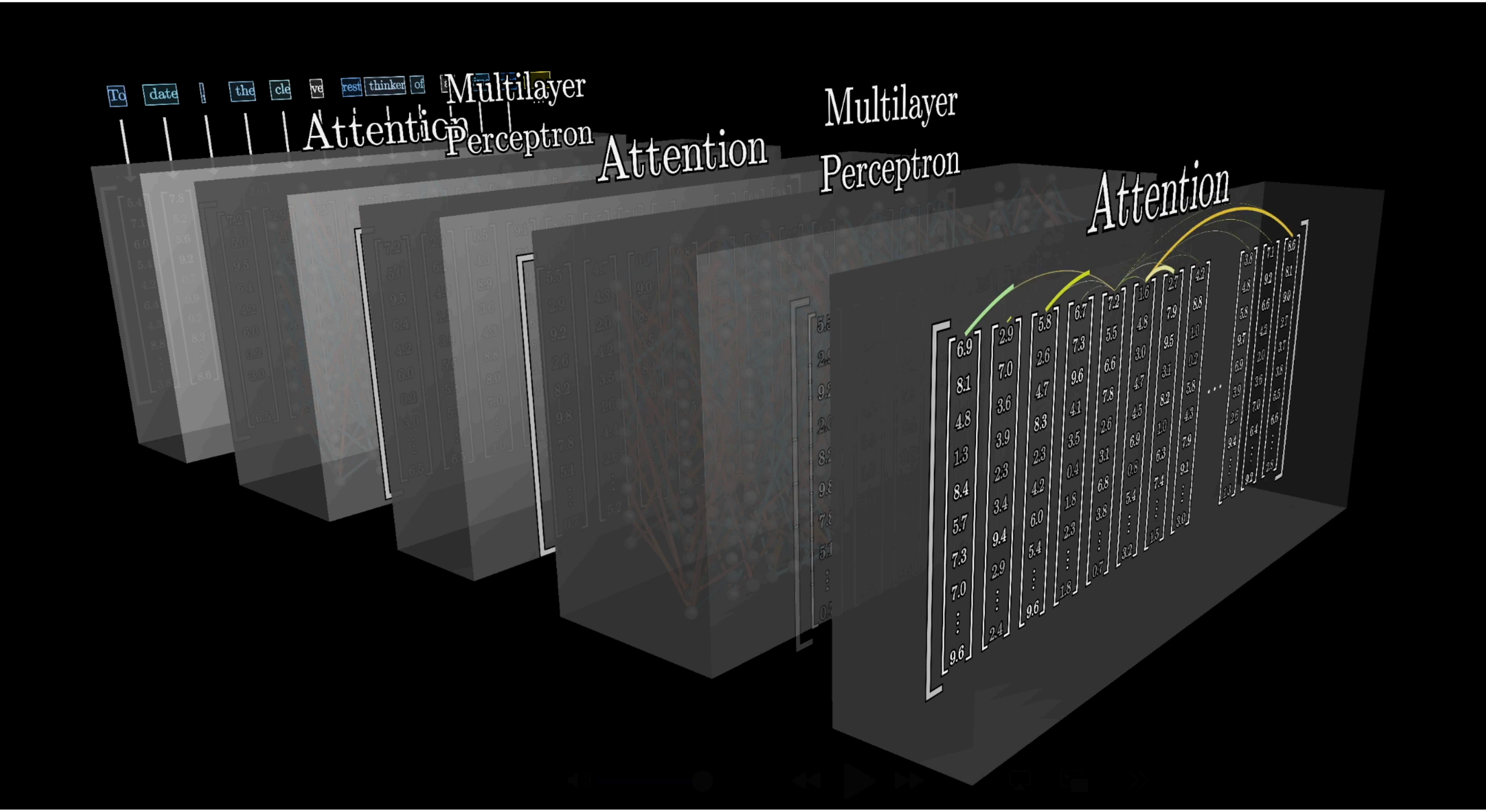
- The Transformer Decoder is a stack of Transformer Decoder **Blocks**.
- Each Block consists of:
  - Self-attention
  - Add & Norm
  - Feed-Forward
  - Add & Norm
- That's it! We've gone through the Transformer Decoder.



Multi-layered Perceptron



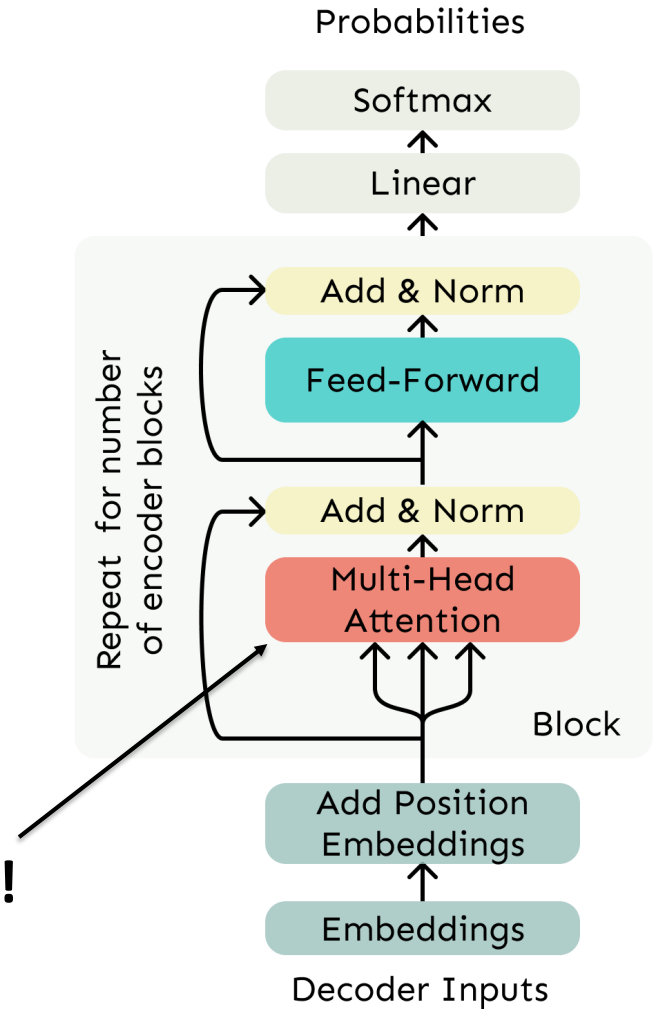
Transformer



Transformer encoder

The Transformer Encoder

- The Transformer Decoder constrains to **unidirectional context**, as for **language models.**
- What if we want **bidirectional context**, like in a bidirectional RNN?
- This is the Transformer Encoder. The only difference is that we **remove the masking** in the self-attention.



No Masking!

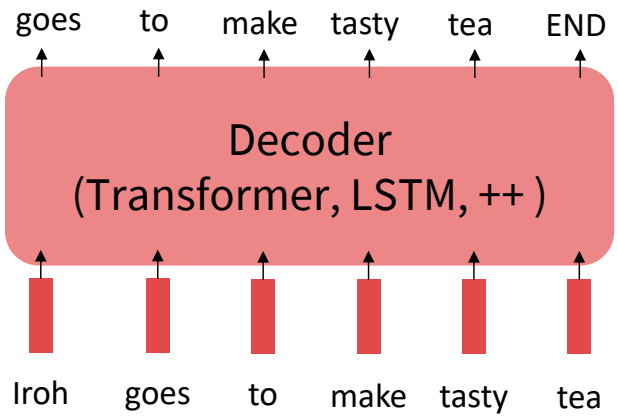
Pretraining

Recall the **language modeling** task:

- Model $p_\theta(w_t|w_{1:t-1})$, the probability distribution over words given their past contexts.
- There's lots of data for this! (In English.)

Pretraining through language modeling:

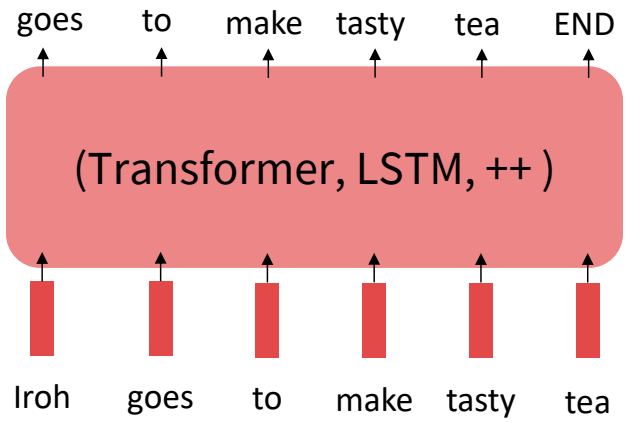
- Train a neural network to perform language modeling on a large amount of text.
- Save the network parameters.



Pretraining / finetuning paradigm

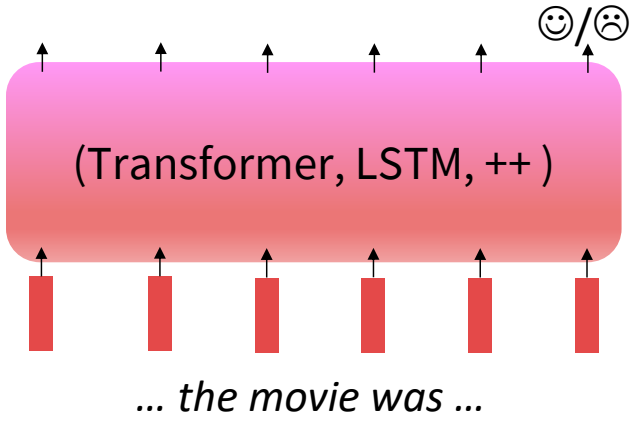
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!

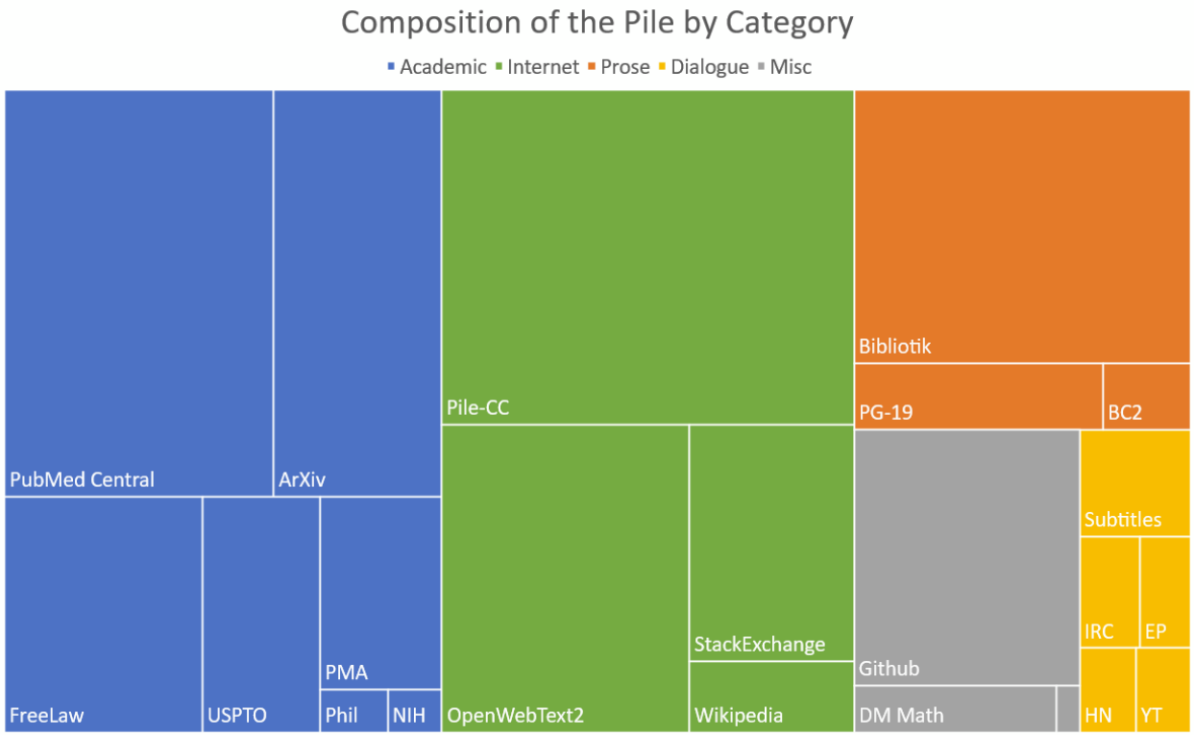


Step 2: Finetune (on your task)

Not many labels; adapt to the task!



What data to use?



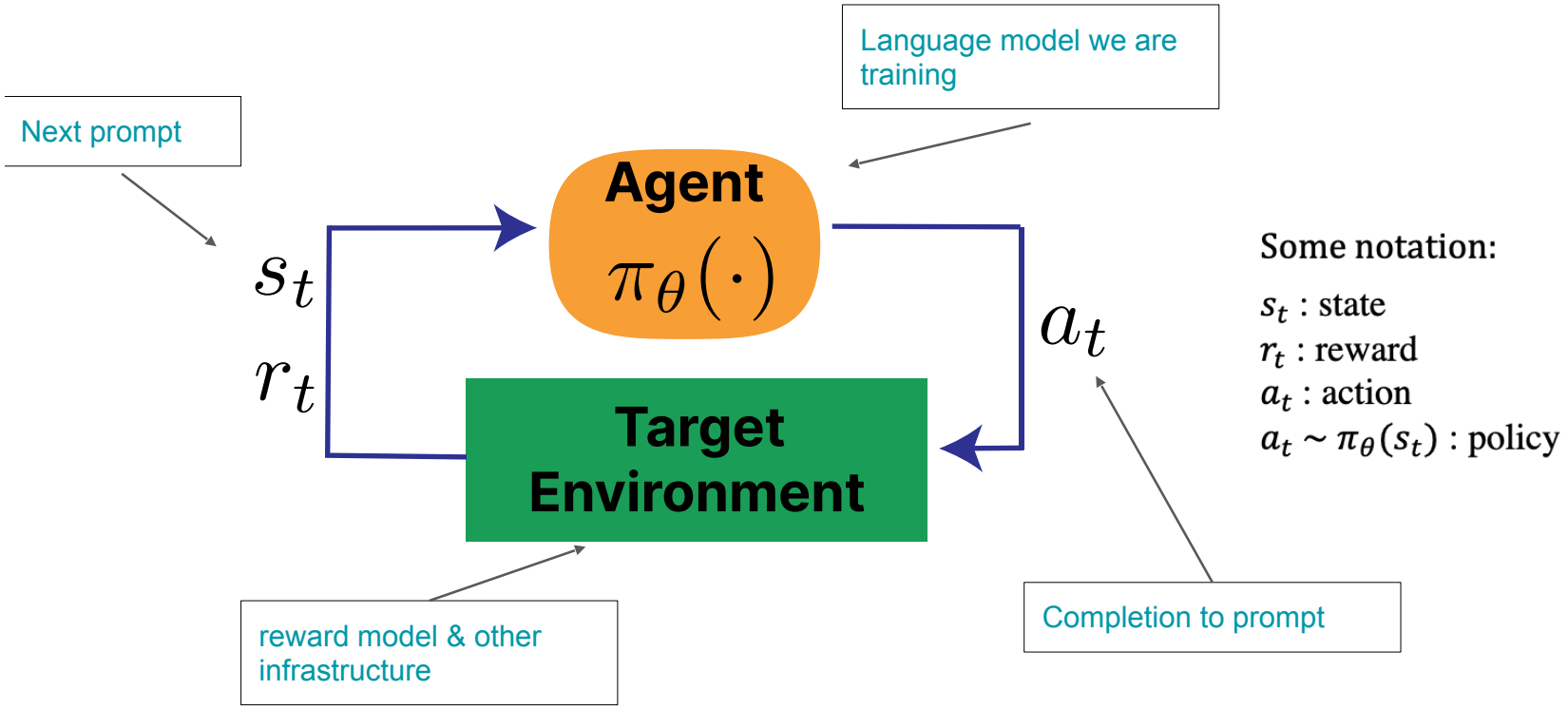
Model	Training Data
BERT	BookCorpus, English Wikipedia
GPT-1	BookCorpus
GPT-3	CommonCrawl, WebText, English Wikipedia, and 2 book databases (“Books 1” and “Books 2”)
GPT-3.5+	Undisclosed

GPT (Devlin et al, 2018)

2018's GPT was a big success in pretraining a decoder!

- Transformer decoder with 12 layers, 117M parameters.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with 40,000 merges
- Trained on BooksCorpus: over 7000 unique books.
  - Contains long spans of contiguous text, for learning long-distance dependencies.
- The acronym "GPT" never showed up in the original paper; it could stand for "Generative PreTraining" or "Generative Pretrained Transformer"

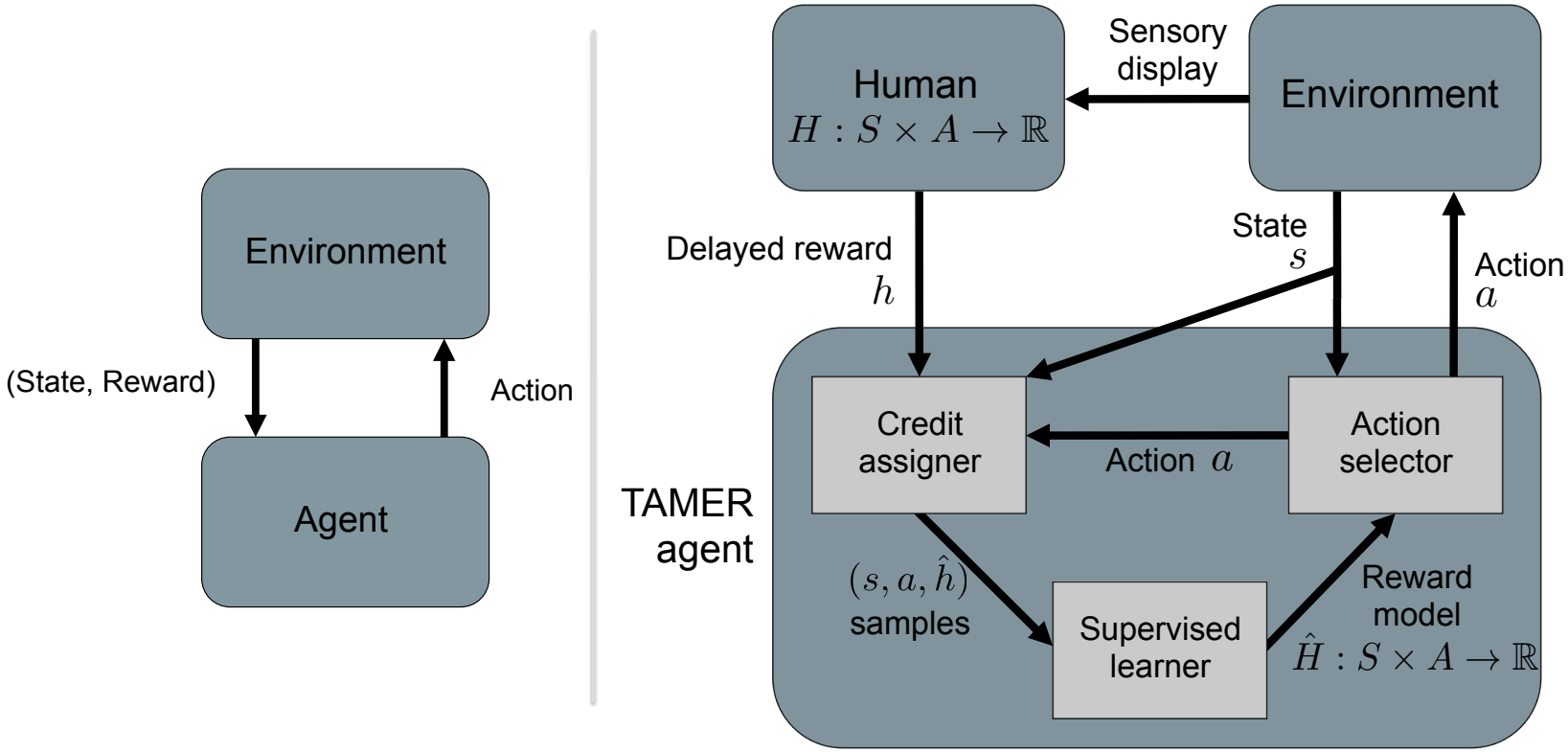
RL comes in the picture!



Some notation:

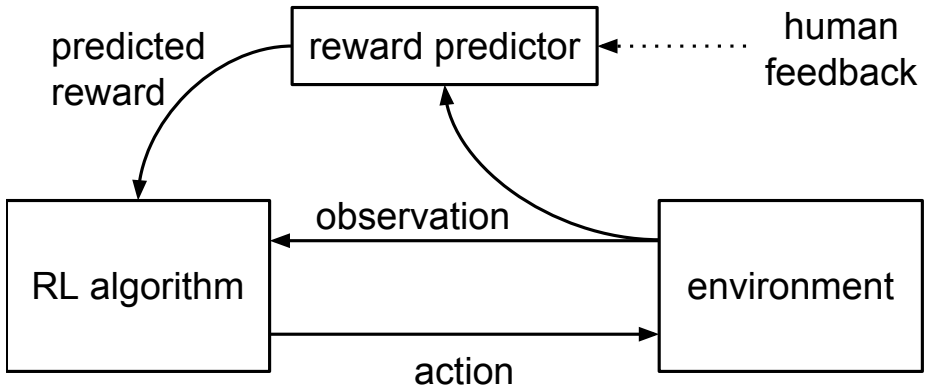
s_t : state
r_t : reward
a_t : action
$a_t \sim \pi_\theta(s_t)$: policy

Learning from Human Feedback (Knox, 2012)



- Numerical reward is a high-variance signal even when learned

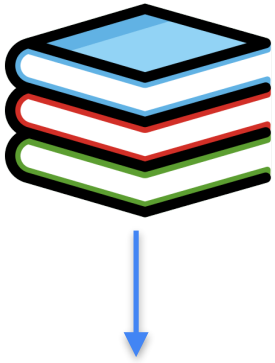
Deep RL from Human Feedback (Christiano et al, 2017)



- People provide a *preference* among two choices
- Assuming there is a latent variable explaining the choice, reward is fit using maximum likelihood (Bradley-Terry model)
- Cf. https://arxiv.org/pdf/1706.03741.pdf

RLHF early attempts

Summarization



"Three pigs defend themselves from a mean wolf"

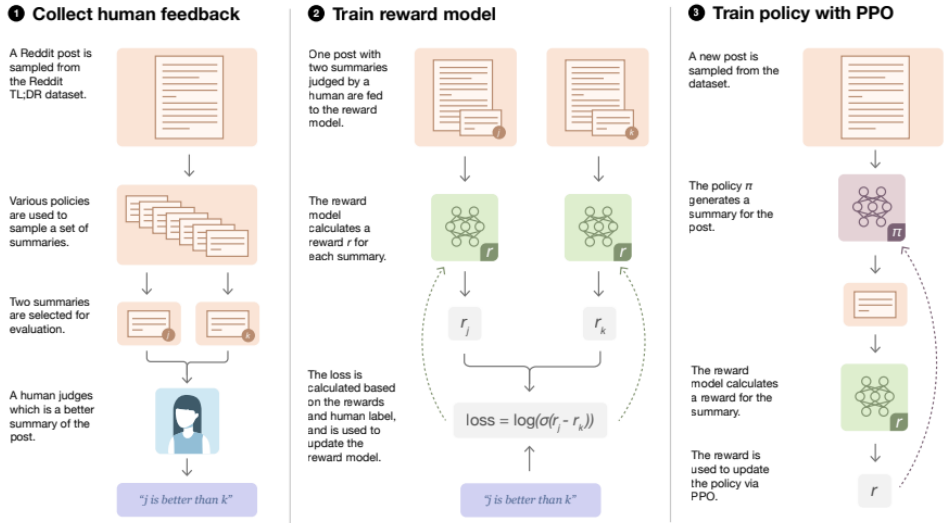
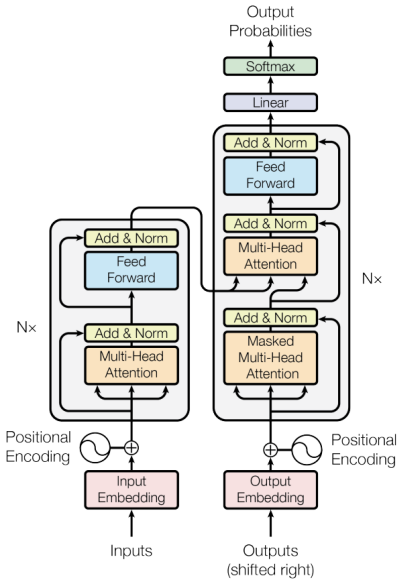


Figure 2: Diagram of our human feedback, reward model training, and policy training procedure.

Stiennon, Nisan, et al. "Learning to summarize with human feedback." *2020.*

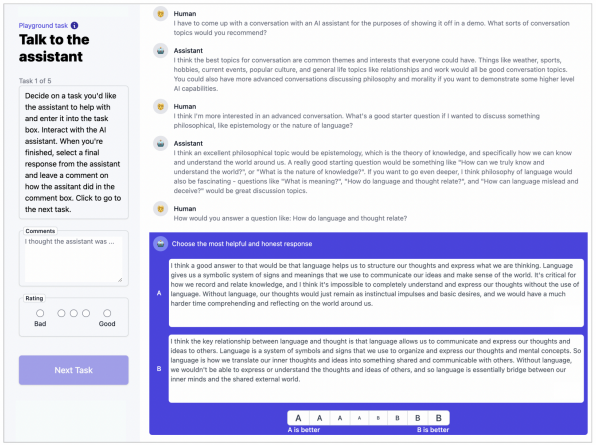
RLHF training phases

base model (instruction, helpful, chatty etc.)

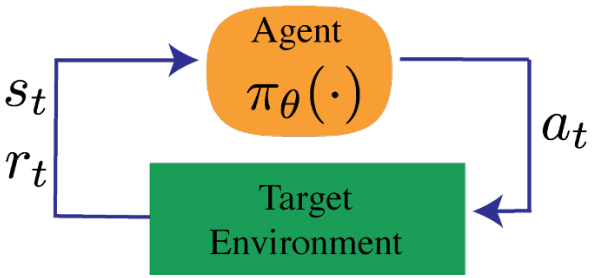


Vaswani et al. 2017

preference collection & training



RL optimization



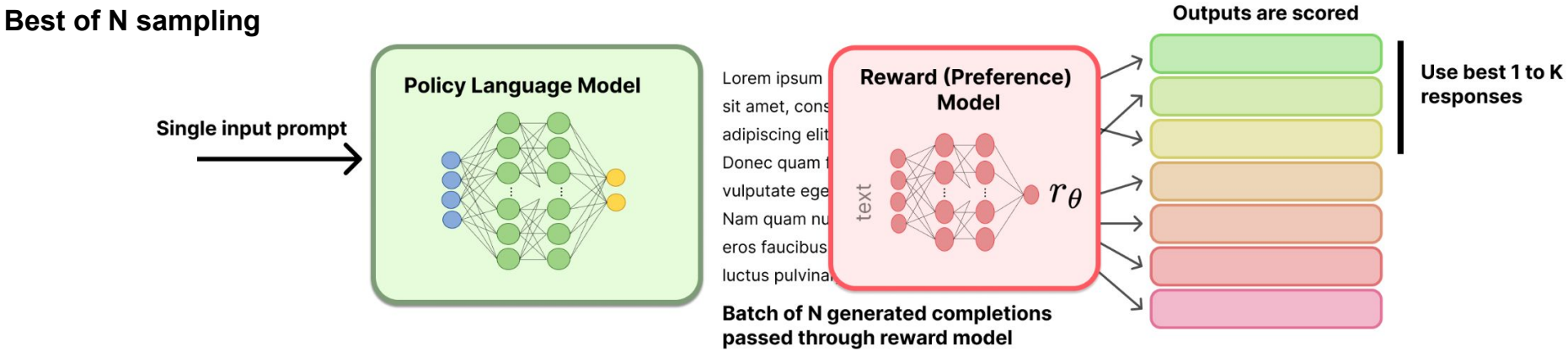
Supervised fine-tuning (SFT): Same as imitation learning!

- We have (s, a) pairs, where s is a prompt and a is a generation corresponding to that prompt (consisting of several tokes
- These are taken from already existing data (eg internet docs, QA, solved problems...)
- Train a policy π_{sft} that maximizes the likelihood of the observed data:

$$J_{\text{sft}}(\theta) = \mathbb{E}_{(s,a)\sim P_{\text{sft}}} \left[\frac{1}{|a|} \sum_{k=1}^{|a|} \log \pi_\theta(a_k | s, a_{i<k}) \right]$$

  Training is done by gradient ascent
- Aka teacher forcing

Better version: Rejection sampling (aka Best-of-N)



- Generate N answers from the model, reinforce the correct/top one(s)
- Train a policy π_{sft} that maximizes the likelihood of the *top data*:

$$J_{\text{rft}}(\theta) = \mathbb{E}_{s\sim P_{\text{sft}}, a\sim\pi_{\text{sft}}(\cdot|s)} \left[\frac{1}{|a|} \mathbf{1}_{a \text{is at the top}} \sum_{k=1}^{|a|} \log \pi_\theta(a_k|s, a_{i<k}) \right]$$

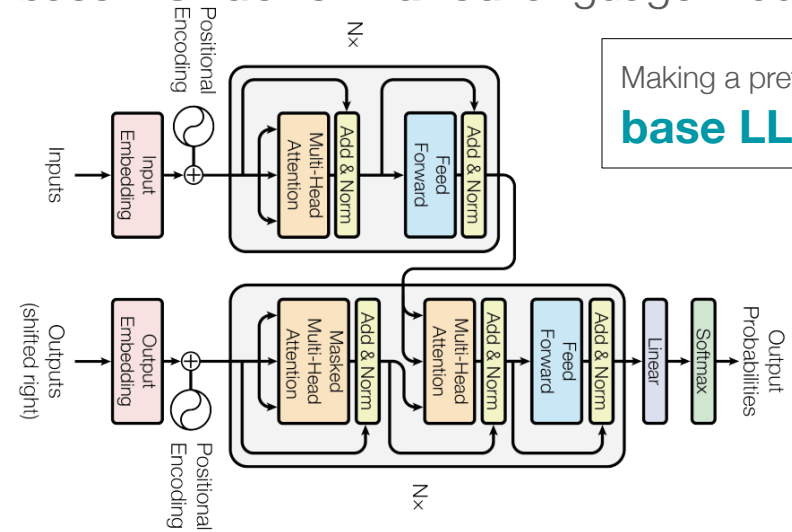
  Training is done by gradient ascent
- *Online* rejection sampling finetuning: $a \sim \pi_\theta$ instead of $a \sim \pi_{\text{sft}}$

Model structure

starting point: a base **instruction-tuned** language model

input:

prompt+completion



The Transformer - Vaswani et al. 2017

Making a preference model:

base LLM with new final layer

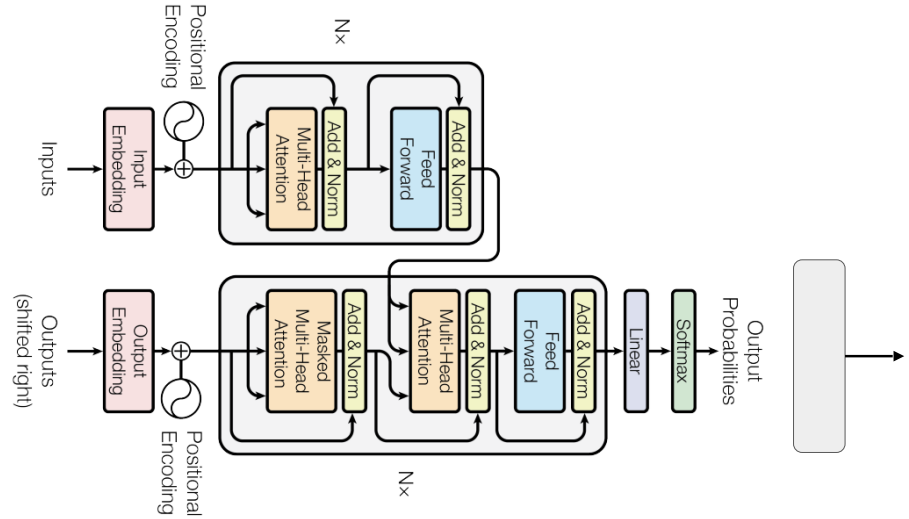
output:
scalar rewards

Training a reward model

input pair:

selected prompt +completion

rejected prompt +completion



The Transformer - Vaswani et al. 2017

output:
scalar rewards

loss: increase difference of predicted reward

Bradely-Terry reward model

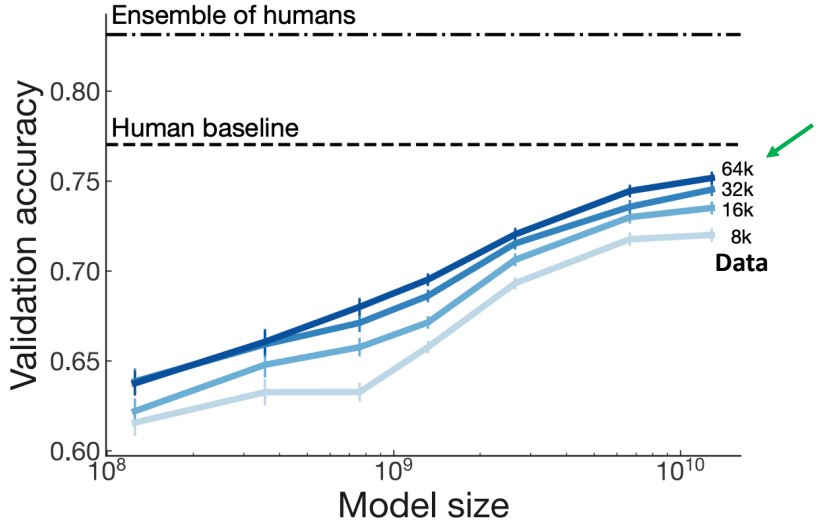
- Collect data from human raters (pairs of y_w, y_l responses to a prompt x)
- Optimize the expected value of:

$$-\log(\sigma(r_\theta(x, y_w) - r_\theta(x, y_l)))$$

  wrt reward parameter vector θ
- Cf. Ouyang et al, InstructGPT
- Corresponds to maximum likelihood fitting of binomial preference function if reward is linear over the variables

Evaluating the reward model

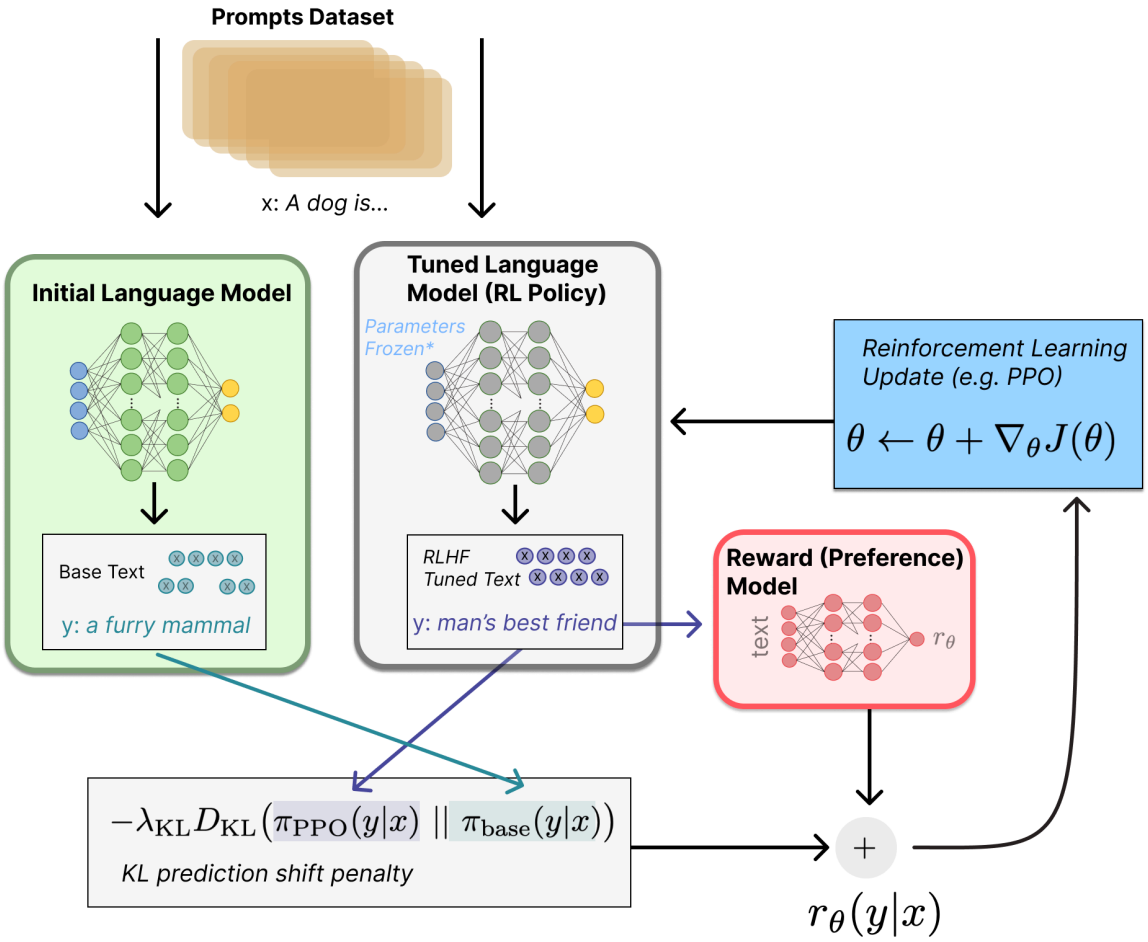
Evaluate RM on predicting outcome of held-out human judgments



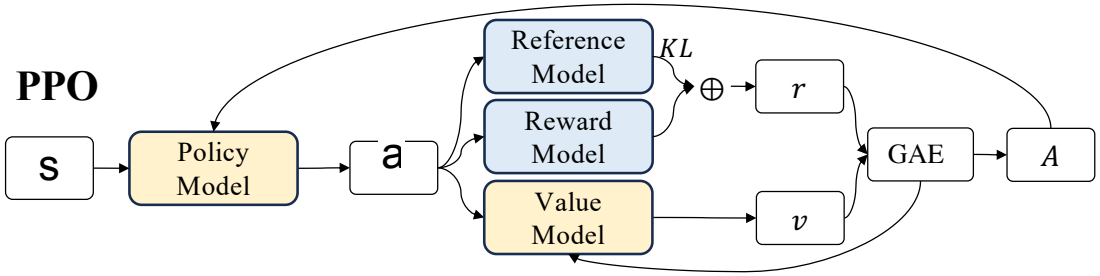
Large enough RM trained on enough data approaching single human perf

[Stiennon et al., 2020]

RLHF finetuning



PPO for RLHF

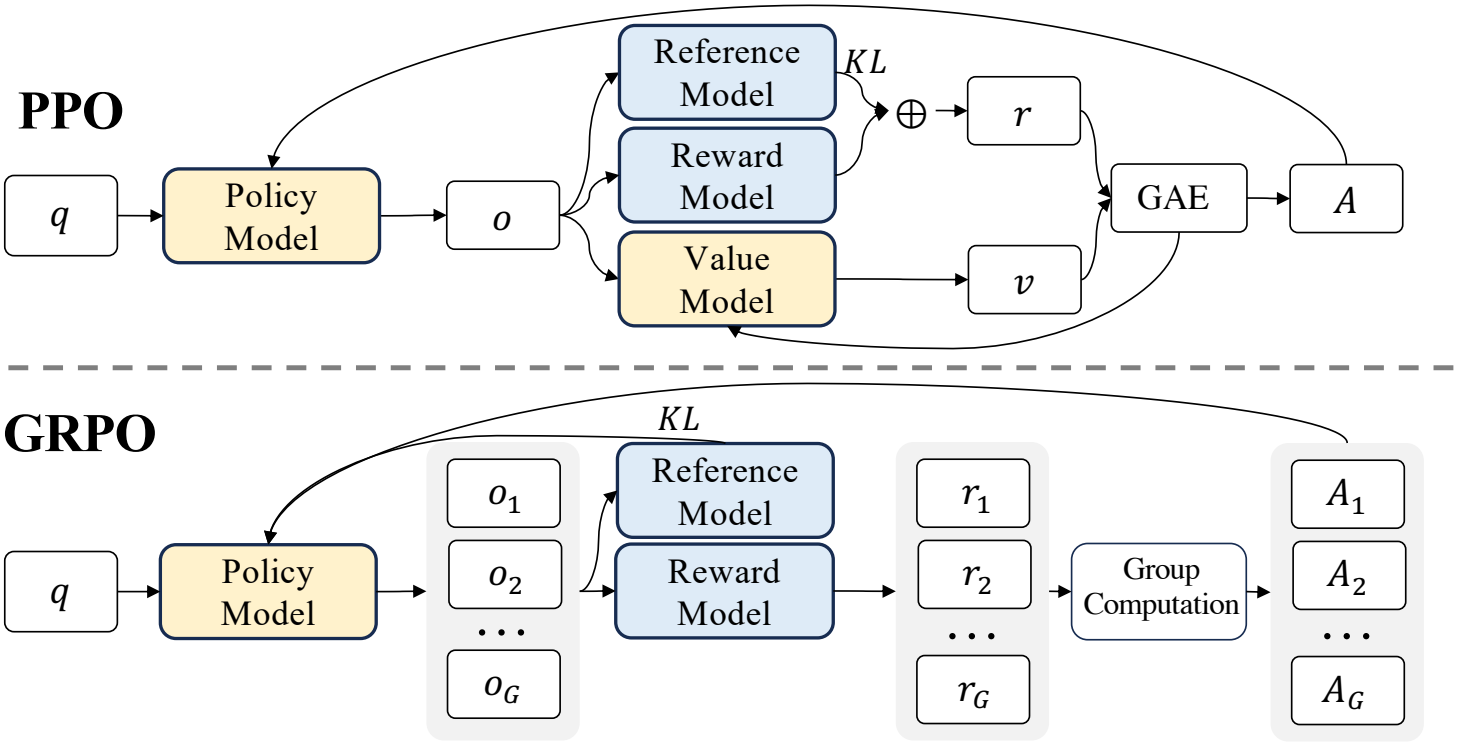


- Train a policy π_θ that maximizes advantage:

$$J_{\text{PPO}}(\theta) = \mathbb{E}_{s\sim P_{\text{sft}}, a\sim\pi_{\theta_{\text{old}}}(\cdot|s)} \left[\frac{1}{|a|} \sum_{k=1}^{|a|} \frac{\pi_\theta(a_k|s, a_{i<k})}{\pi_{\theta_{\text{old}}}(a_k|s, a_{i<k})} A_i \right]$$

  where A_i is the advantage function
- Reward function uses a penalty per token for straying from reference policy: $r_t = r_\phi(s, a_{<t}) - \beta \log \frac{\pi_\theta(a_t|s,a_{<t})}{\pi_{\text{sft}}(a_t|s,a_{<t})}$
- Value function/advantage needs to be estimated!

GRPO (DeepSeek, 2025)



Instead of estimating value, use a group (non-parametric approach)

Notation: $q = s, o = a$

GRPO Objective (DeepSeek, 2025)

- Generate G answers and estimate their reward (no regularization towards reference policy)
- Compute a normalized advantage based on the mean $\bar{r}_t$ and standard deviation of the rewards:

$$\hat{A}_{i,t} = \frac{r_{i,t} - \bar{r}_t}{std(r_{1,t}, \ldots r_{G,t})}$$

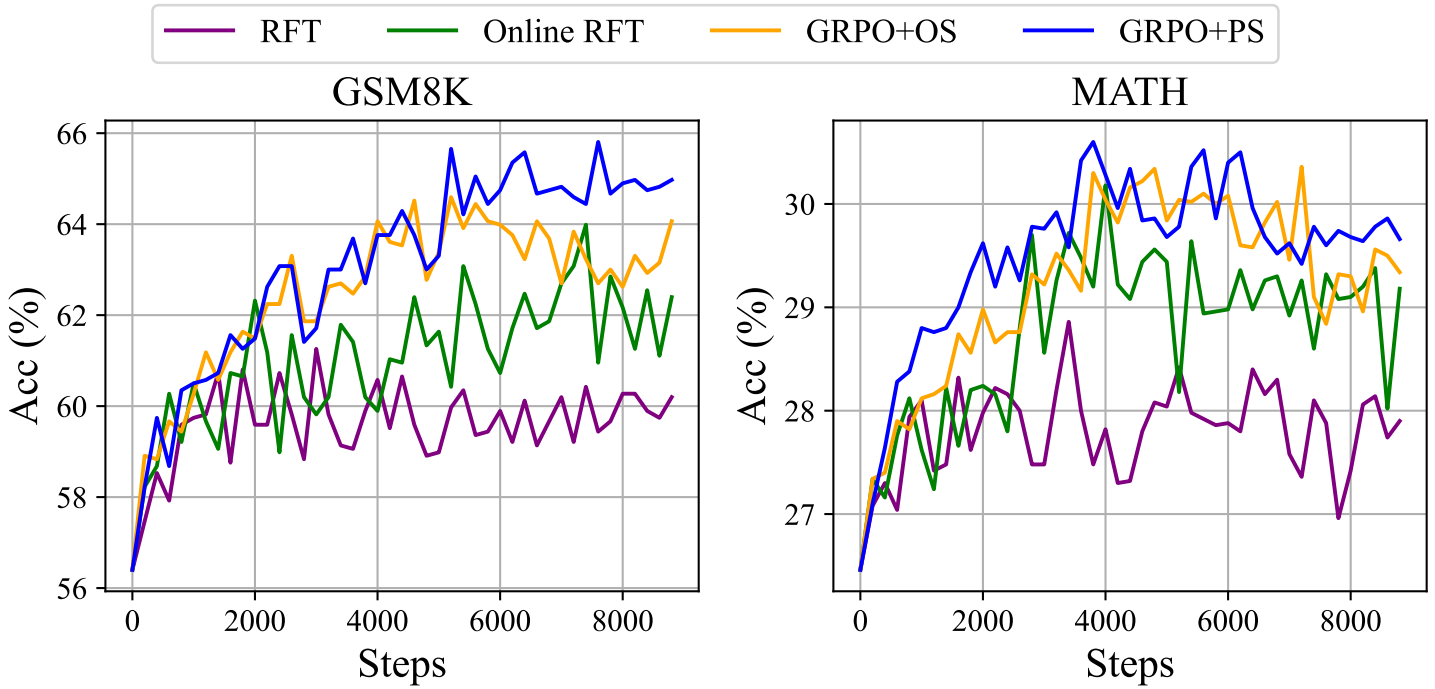
- GRPO objective - very similar to PPO!

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{s\sim P_{\text{sft}}, a_i \sim \pi_{\theta_{\text{old}}}(\cdot|s), i=1,\ldots G} \left[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{|a_i|} \sum_{k=1}^{|a_i|} \frac{\pi_\theta(a_k|s, a_{i<k})}{\pi_{\theta_{\text{old}}}(a_k|s, a_{i<k})} \hat{A}_{i,t} - \beta D_{KL}(\pi_\theta, \pi_{\theta_{\text{old}}}) \right]$$

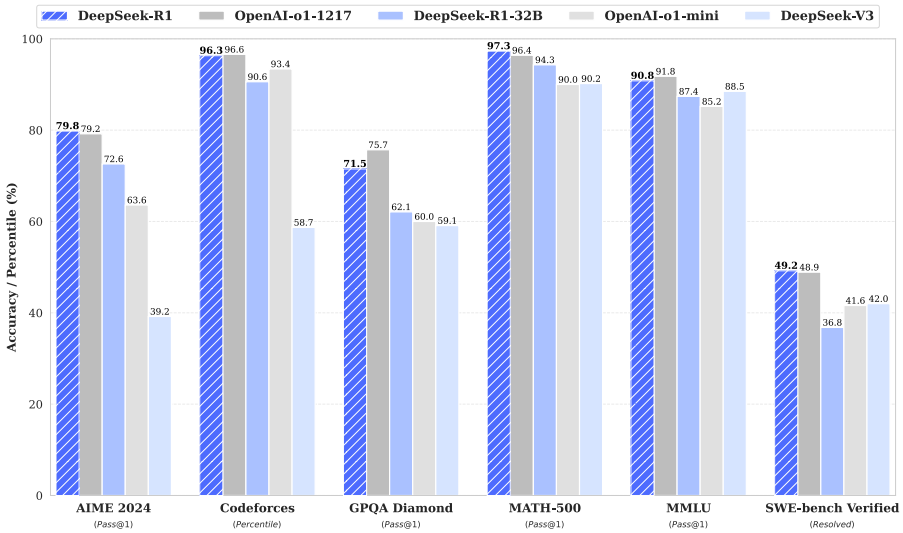
- D_{KL} is also estimated a bit differently (cf Shulman et al, 2020):

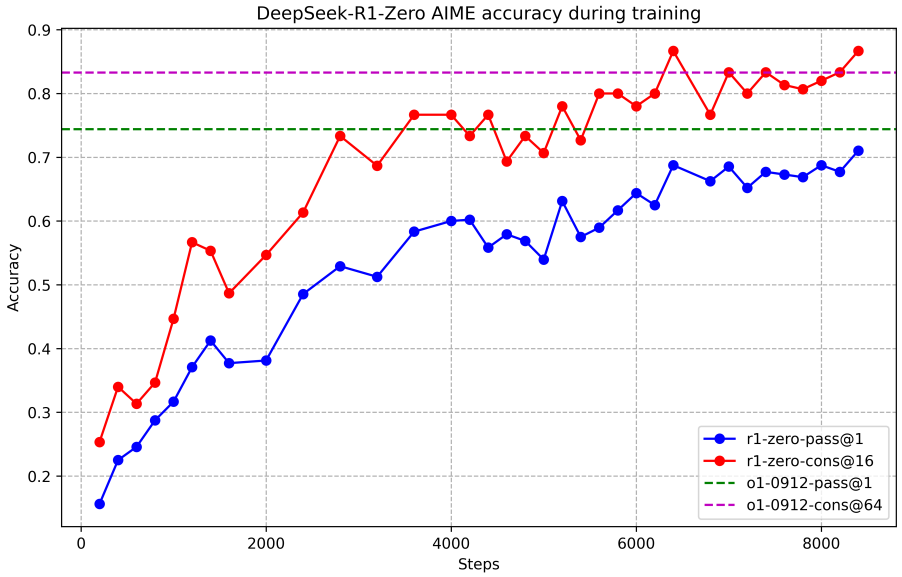
$$D_{KL}(\pi_\theta, \pi_{\theta_{\text{old}}}) = \frac{\pi_{\theta_{\text{old}}}(a_k|s, a_{i<k})}{\pi_\theta(a_k|s, a_{i<k})} - \log \frac{\pi_{\theta_{\text{old}}}(a_k|s, a_{i<k})}{\pi_\theta(a_k|s, a_{i<k})} - 1$$

The Advantage of RL over SFT (DeepSeek, 2025)



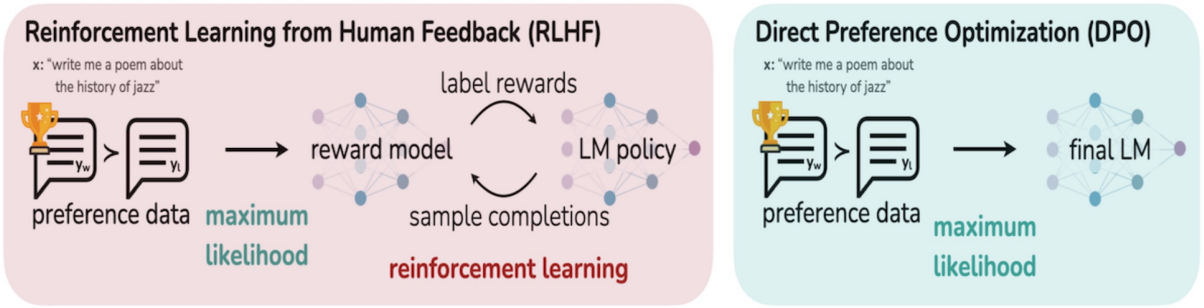
DeepSeek Overall Results (DeepSeek, 2025)





SOTA results back in January 2025

Direct Preference Optimization



$$\nabla_\theta \mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) =$$

$$-\beta \mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\Big[\underbrace{\sigma(\hat{r}_\theta(x,y_l) - \hat{r}_\theta(x,y_w))}_{\text{higher weight when reward estimate is wrong}}\Big[\underbrace{\nabla_\theta \log \pi(y_w \mid x)}_{\text{increase likelihood of } y_w} - \underbrace{\nabla_\theta \log \pi(y_l \mid x)}_{\text{decrease likelihood of } y_l}\Big]\Big],$$

$$\hat{r}_\theta(x,y) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$$

- You can replace the complex RL part with a very simple weighted MLE objective
- Other variants (KTO, IPO) now emerging too

[Rafailov+ 2023]

Learning with non-transitive preferences: NashLLM

- Objective:find a policy π^* which is preferred over any other policy

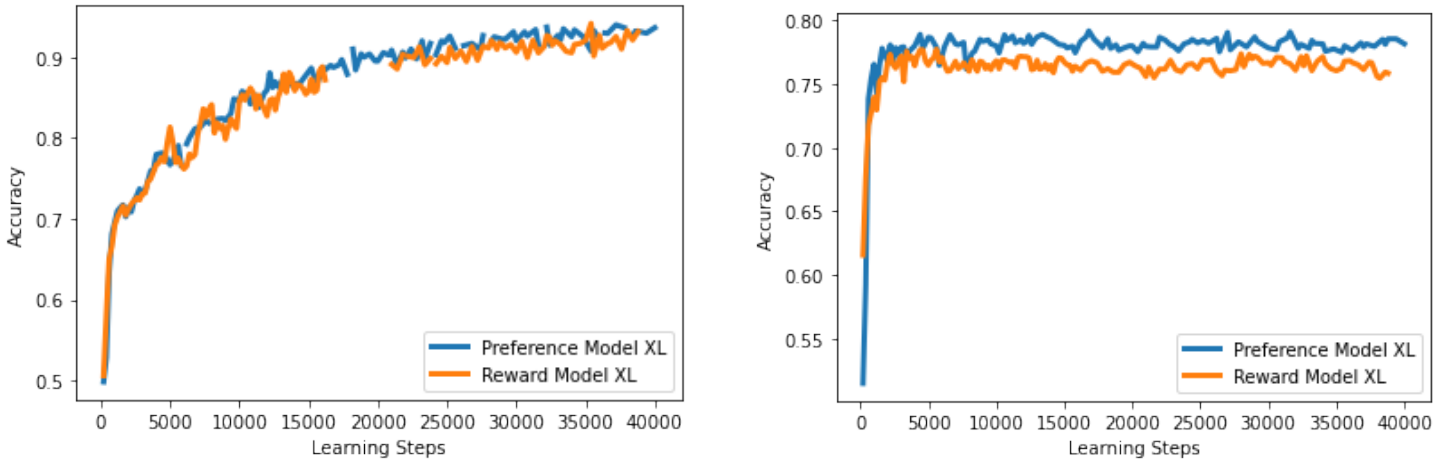
$$\pi^* = \arg\max_{\pi} \min_{\pi'} \mathbb{P}(\pi' \preceq \pi)$$

- Think of this as a game: one player picks π the other picks π'
- When both players use π^* this is a *Nash equilibrium* for the game
- For this game an equilibrium exists (even if eg preferences are not transitive)
- Cf. Munos et al, 2024 (https://arxiv.org/pdf/2312.00886.pdf)

NashLLM-style algorithms

- Fit a *two-argument preference function* by supervised learning
- Decide what is the *set of opponent policies*
- Ideally, the max player should play against a mixture of past policies
- *Optimize* using eg online mirror descent, convex-concave optimization...
- A lot of algorithmic variations to explore!

NashLLM results



Using preferences instead of rewards leads to less overfitting

General blueprint of RLHF training

Finally, we have everything we need:

- A pretrained (possibly instruction-finetuned) LM $p^{PT}(s)$
- A reward model $RM_\phi(s)$ that produces scalar rewards for LM outputs, trained on a dataset of human comparisons
- A method for optimizing LM parameters towards an arbitrary reward function.

Now to do RLHF:

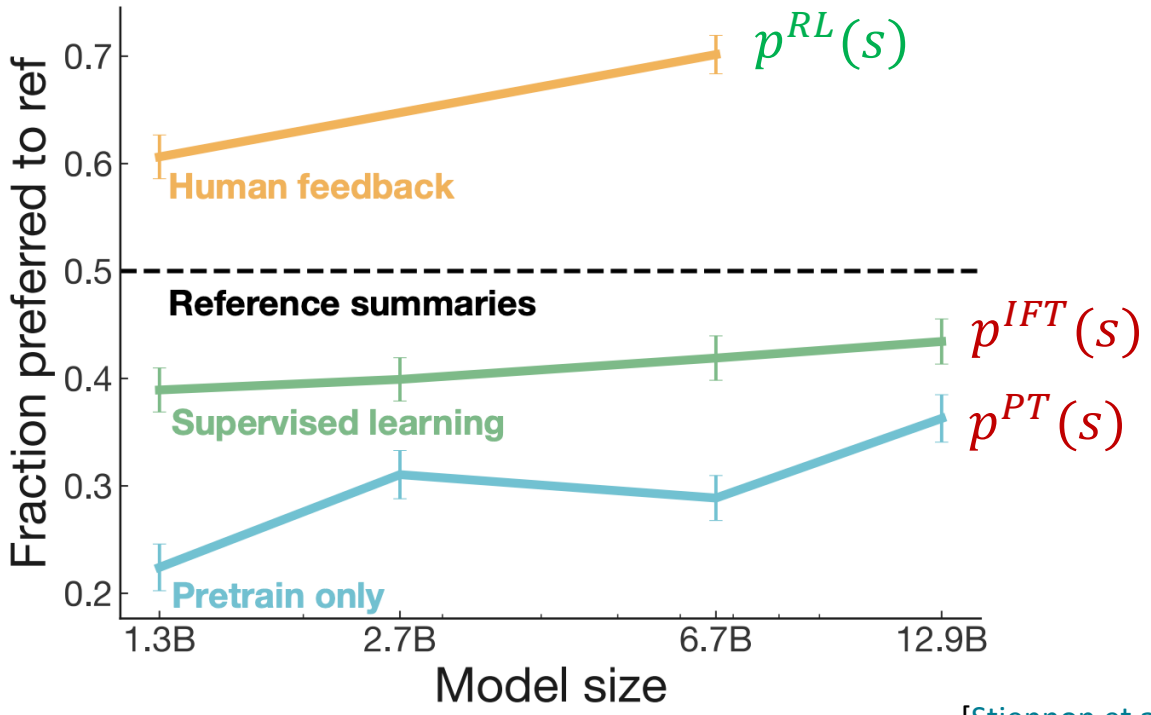
- Initialize a copy of the model $p_\theta^{RL}(s)$, with parameters θ we would like to optimize
- Optimize the following reward with RL:

$$R(s) = RM_\phi(s) - \underbrace{\beta \log \left(\frac{p_\theta^{RL}(s)}{p^{PT}(s)} \right)}$$

Pay a price when $p_\theta^{RL}(s) > p^{PT}(s)$

This is a penalty which prevents us from diverging too far from the pretrained model. In expectation, it is known as the **Kullback-Leibler** (**KL**) divergence between $p_\theta^{RL}(s)$ and $p^{PT}(s)$.

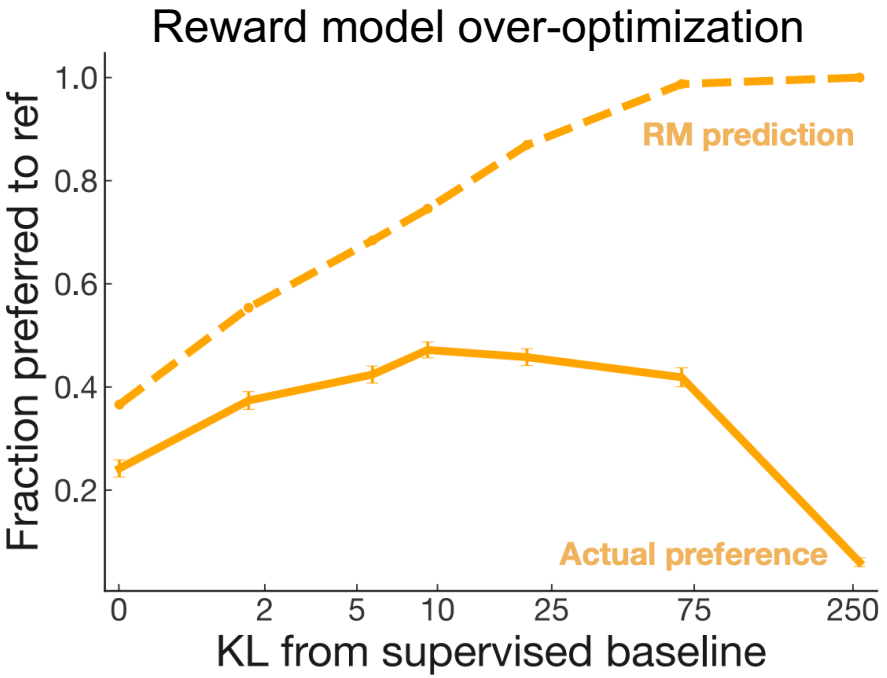
RLHF results



[Stiennon et al., 2020]

Problem: reward hacking

- Human preferences are unreliable!
  - ”Reward hacking” is a common problem in RL
  - Chatbots are rewarded to produce responses that *seem* authoritative and helpful, *regardless of truth*
  - This can result in making up facts + hallucinations
- **Models** of human preferences are *even more* unreliable!



$$R(s) = RM_\phi(s) - \beta \log \left(\frac{p_\theta^{RL}(s)}{p^{PT}(s)} \right)$$

More important methods

- Self-improvement
- Chain-of-thought prompting
- Distillation from large models to small
- Utilizing more inference time using search (cool new work)

More open directions

- Multi-turm
- Exploration
- .....