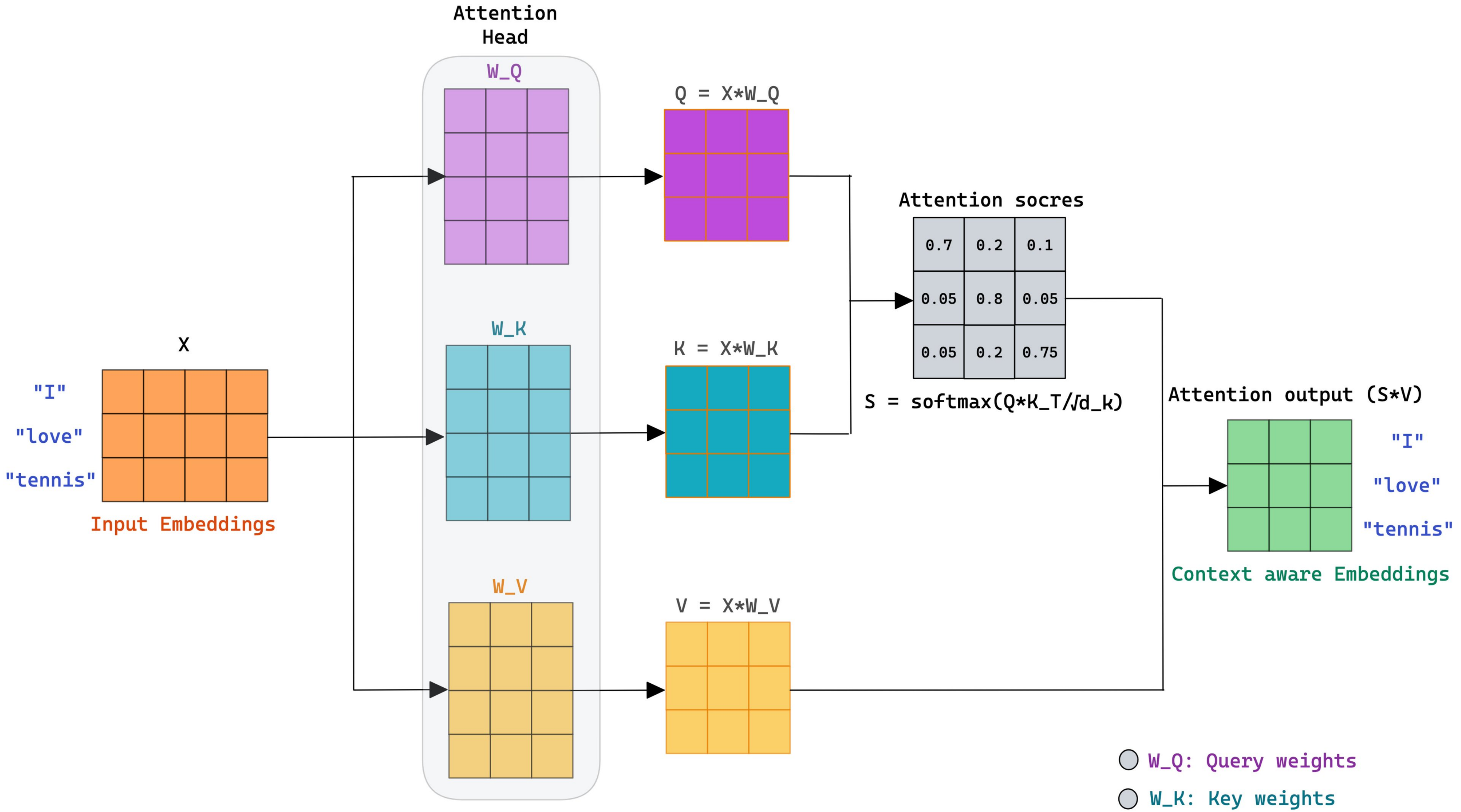
Self Attention Clearly Explained!



Attention socres

0.7	0.2	0.1
0.05	0.8	0.05
0.05	0.2	0.75

@akshay_pachaar

- ○ W_Q: Query weights
- ○ W_K: Key weights
- ○ W_V: Value weights
- ○ d_k: attention head size

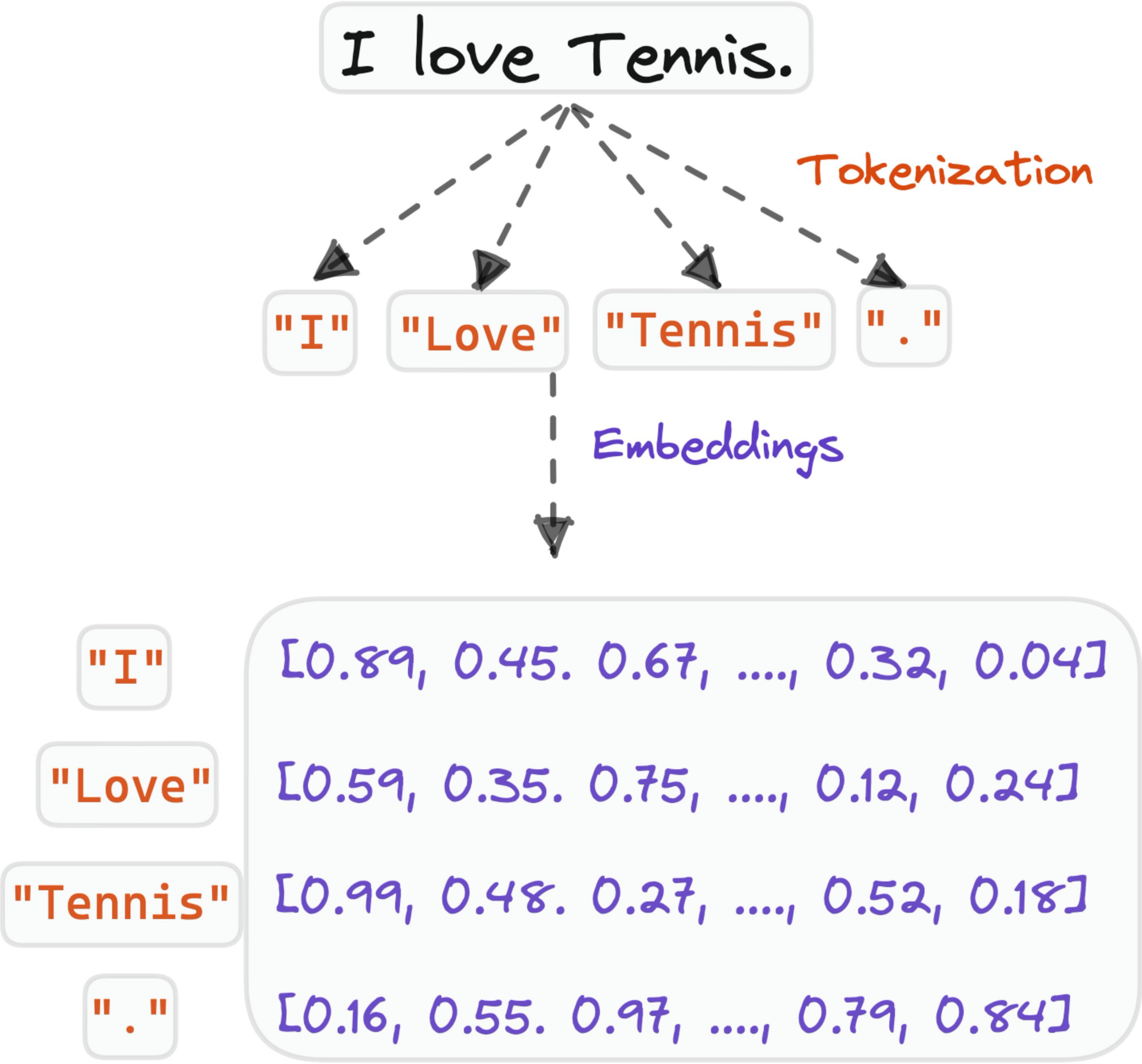
Computers are good with numbers ❗

In NLP we convert the sequence of words into token & then token to embeddings.

You can think of embedding as a meaningful representation of each token using a bunch of numbers.

Swipe 👉

@akshay_pachaar



I love Tennis.

Tokenization

"I" "Love" "Tennis" "."

Embeddings

"I" [0.89, 0.45. 0.67, ..., 0.32, 0.04]

"Love" [0.59, 0.35. 0.75, ..., 0.12, 0.24]

"Tennis" [0.99, 0.48. 0.27, ..., 0.52, 0.18]

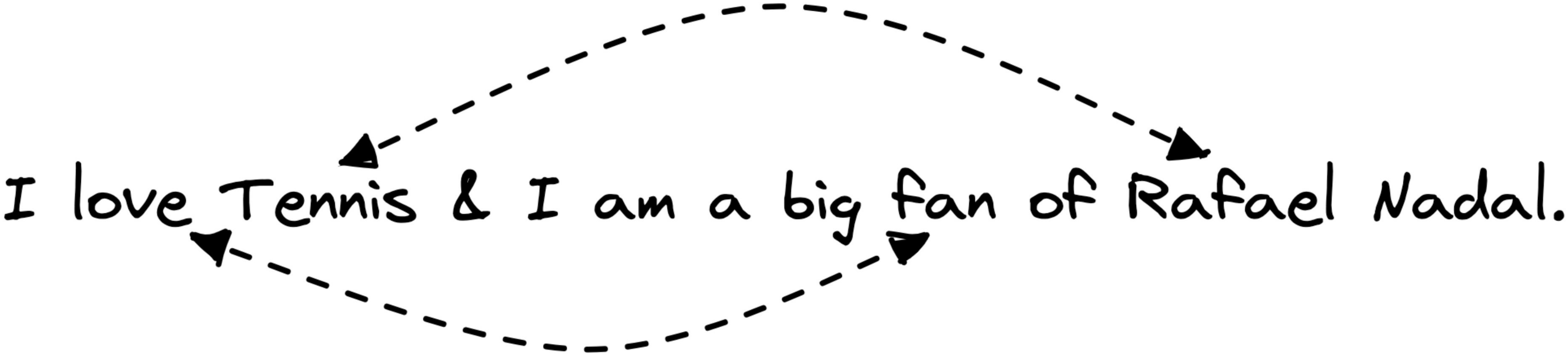
"." [0.16, 0.55. 0.97, ..., 0.79, 0.84]

@akshay_pachaar

Now, for a language model to perform at a human level, it's not sufficient for it to process these tokens independently.

It's also important to understand the relationship between them!

Swipe 👉



I love Tennis & I am a big fan of Rafael Nadal.

A language model must see the entire context,
It should be aware of the relative positions and
relationships among the tokens.

Let's see how it's done 🧵

In the self-attention, relationships between tokens are expressed as probability scores.

Each token assigns the highest score to itself and additional scores to other tokens based on their relevance.

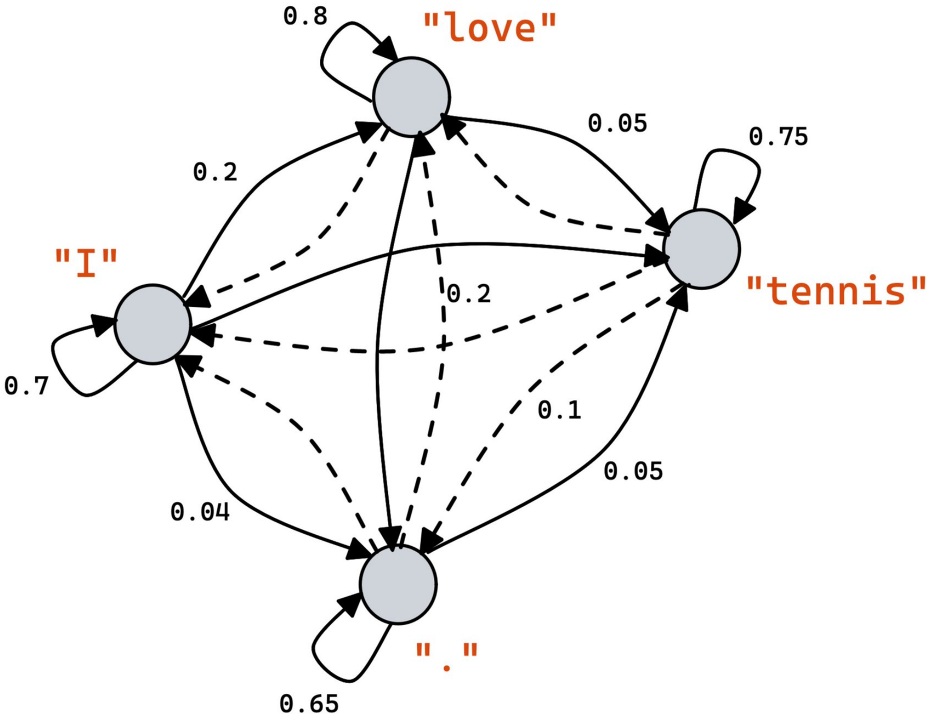
Swipe 👉

Attention: A communication mechanism

Attention probability scores:
how much a token should pay attention
to itself & the neighboring token

	"I"	"love"	"tennis"	"."
"I"	0.7	0.2	0.06	0.04
"love"	0.1	0.8	0.05	0.05
"tennis"	0.05	0.1	0.75	0.1
"."	0.1	0.2	0.05	0.65

Visualizing attention as a directed graph



Wondering where these numbers come from⁉️🤔
Continue reading ... 📖

@akshay_pachaar

To understand how self-attention works we first need to understand 3 terms:

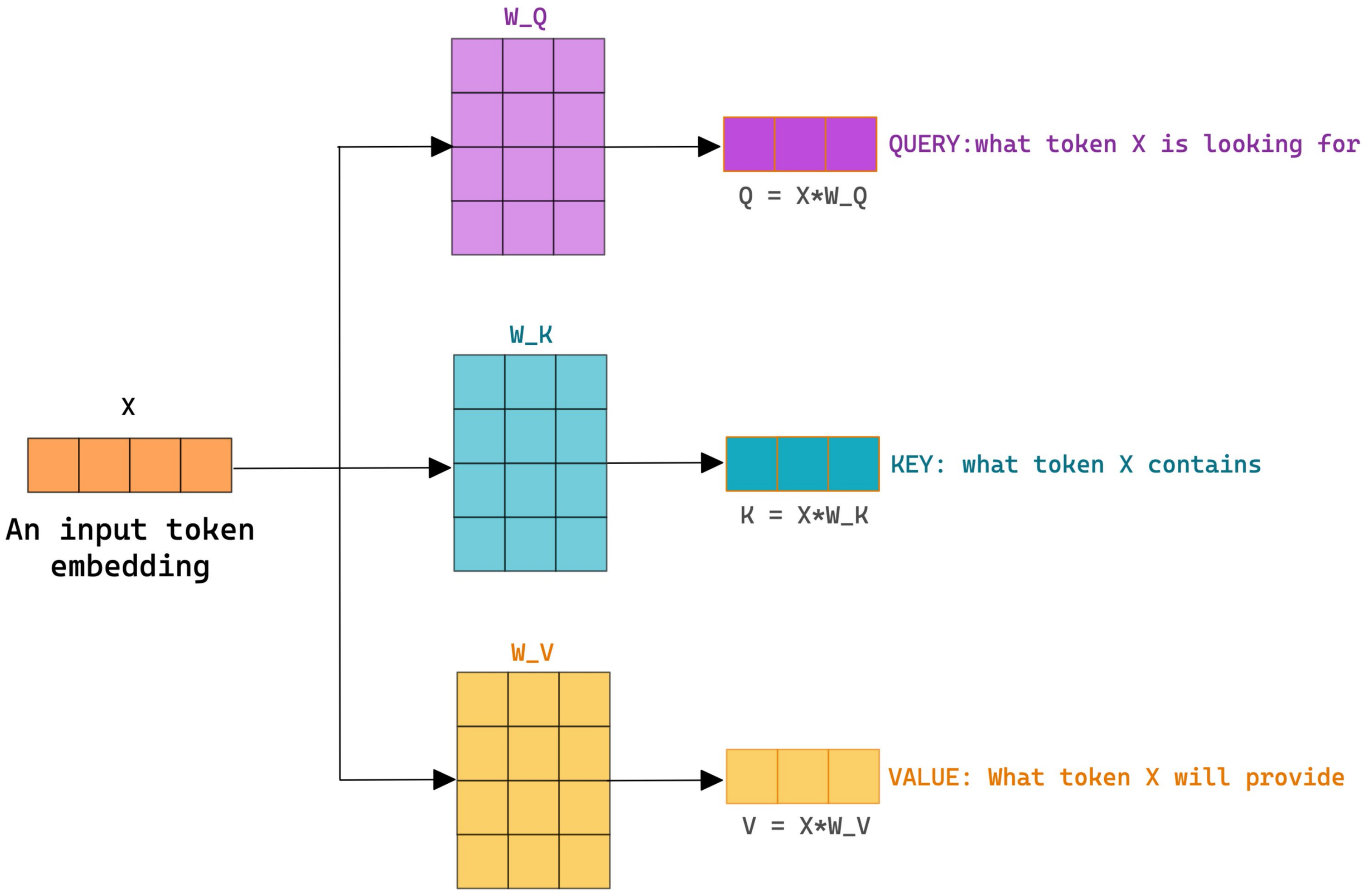
- Query Vector
- Key Vector
- Value Vector

These vectors are created by multiplying the input embedding by three weight matrices that are trainable.

Swipe 👉

Understanding Keys, Queries & Values

W_Q , W_K & W_V are Trainable weight matrices.


W_Q
QUERY:what token X is looking for
Q = X*W_Q
W_K
X
KEY: what token X contains
K = X*W_K
An input token embedding
W_V
VALUE: What token X will provide
V = X*W_V

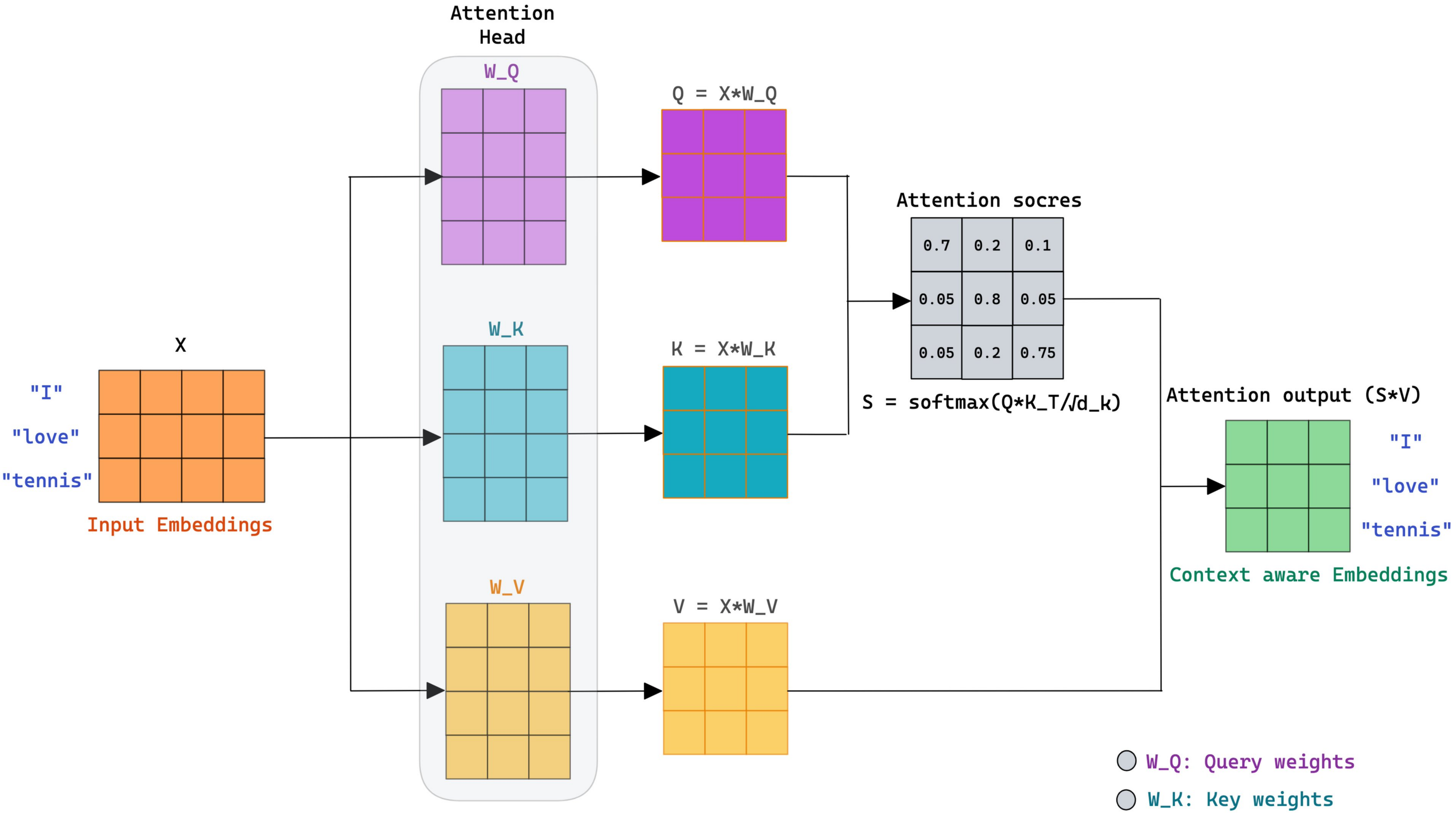
@akshay_pachaar

Self-attention allows models to learn long-range dependencies between different parts of a sequence.

After acquiring keys, queries, and values, we merge them to create a new set of context-aware embeddings.

Swipe 👉

Self Attention Clearly Explained!



Attention socres

0.7	0.2	0.1
0.05	0.8	0.05
0.05	0.2	0.75

- ○ W_Q: Query weights
- ○ W_K: Key weights
- ○ W_V: Value weights
- ○ d_k: attention head size

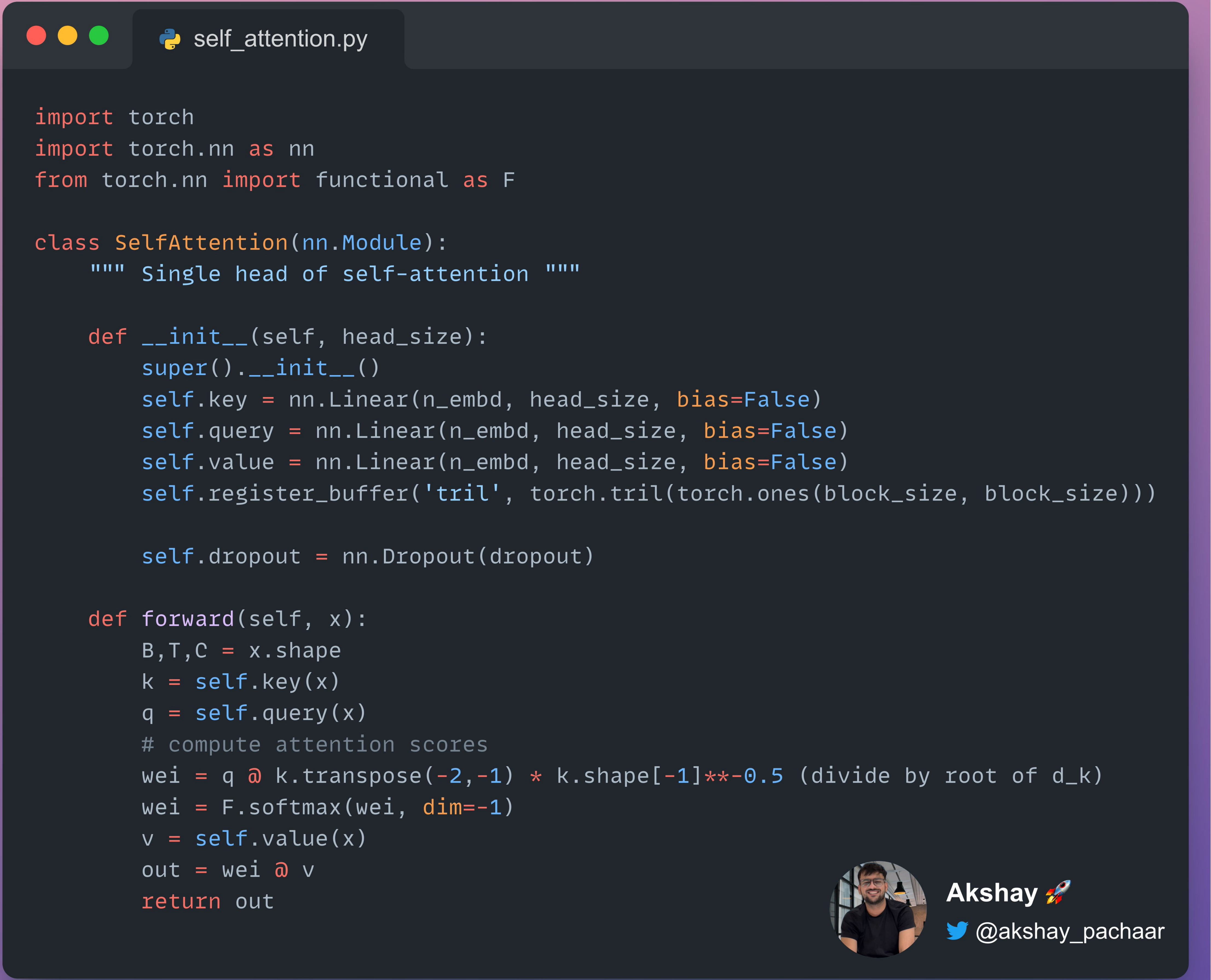
@akshay_pachaar

Implementing self-attention using PyTorch, doesn't get easier! 🚀

It's very intuitive! 💡

Swipe 👉

@akshay_pachaar



self_attention.py

```python
import torch
import torch.nn as nn
from torch.nn import functional as F

class SelfAttention(nn.Module):
    """ Single head of self-attention """

    def __init__(self, head_size):
        super().__init__()
        self.key = nn.Linear(n_embd, head_size, bias=False)
        self.query = nn.Linear(n_embd, head_size, bias=False)
        self.value = nn.Linear(n_embd, head_size, bias=False)
        self.register_buffer('tril', torch.tril(torch.ones(block_size, block_size)))

        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        B,T,C = x.shape
        k = self.key(x)
        q = self.query(x)
        # compute attention scores
        wei = q @ k.transpose(-2,-1) * k.shape[-1]**-0.5 (divide by root of d_k)
        wei = F.softmax(wei, dim=-1)
        v = self.value(x)
        out = wei @ v
        return out
```

Akshay 🚀

@akshay_pachaar

That's a wrap!

If you interested in:

- Python 🐍
- Data Science 📈
- Machine Learning 🤖
- MLOps 🛠️
- NLP 🗣️
- Computer Vision 🎥
- LLMs 🧠

Follow me on LinkedIn ✔️
Everyday, I share tutorials on above topics!

Cheers!! 😊

@akshay_pachaar