Tiki’s implementation of the self attention mechanism

I implemented the self-attention mechanism, the self-attention mechanism is a fundamental component of transformer models. The self-attention mechanism is widely used in natural language processing and computer vision tasks. The self-attention mechanism allows a model to dynamically focus on different parts of the input sequence when generating a representation with context. Your function should return the self-attention output as a numpy array.

TIKI’S SOLUTION repeated twice

```
import numpy as np #use Numpy for the fast matrix math, like tiny algebra blocks

def compute_qkv(X, W_q, W_k, W_v):
Def compute_qkv(X, W_q, W_k, W_v):
    """
    X:   (seq_len, d_model) input sequence.
    W_q: (d_model, d_k)     query weight matrix.
    W_k: (d_model, d_k)     key weight matrix.
    W_v: (d_model, d_v)     value weight matrix.
    Returns: Q, K, V as NumPy arrays.
    """

    # Q = X * W_q: each token asks "what am I looking for?" in a new space.
#Q = X * W_q: each token asks “what am i looking for?” in a new space.
    Q = X @ W_q
    Q = X @ W_q

    # K = X * W_k: each token stores "what content do I have?" in key space.
    # K = X * W_k: each token stores “what content do I have?” in key space.

    K = X @ W_k
    K = X @ W_k


     # V = X * W_v: each token carries the actual information to be mixed.
     #V = X * W_v: each token carries the actual information to be mixed.
    V = X @ W_v
    V = X @ W_v

    # Return all three so caller can unpack (Q, K, V) without getting None.
    # Return all three so caller can unpack (Q, K,V) without getting None.
    return Q, K, V
    return Q, K, V


def softmax(x, axis=-1):
def softmax(X, axis=-1):
    """
    Numerically stable softmax along given axis.
    Turns raw scores into probabilities that sum to 1.
```

```
    """
    # Subtract max for stability so exp() doesn't blow up.
    #Subtract my max for stability
    x_shifted = x - np.max(x, axis=axis, keepdims=True)
    x_shifted = x - np.max(x, axis=axis, keepsdims=True)

    # Exponentiate shifted scores to make them positive.
    #Exponentiate my raw scores to make them positive
    exp_x = np.exp(x_shifted)
    exp_x = np.exp(x_shifted)

    # Normalize by the sum so they become probabilities.
    # Normalize by the sum so they become probabilities.
    return exp_x / np.sum(exp_x, axis=axis, keepdims=True)
    return exp_x / np.sum(exp_x, axis=axis, keepdims=True)


def self_attention(Q, K, V):
def self_attention(Q, K, V):
    """
    Q: (seq_len, d_k) queries  – "what each position wants."
    K: (seq_len, d_k) keys     – "what each position offers."
    V: (seq_len, d_v) values  – "info carried by each position."
    Returns: (seq_len, d_v) attention output.
    """

    # d_k is the dimension of query/key space. Used to scale scores.
    #D_K IS THE DIMENSION OF THE QUERY/KEY SPACE. USED TO SCALE THE SCORES
    d_k = K.shape[-1]
    d_k= K.shape[-1]

    # Compute raw attention scores: how much each query likes each key.
    #Compute raw attention scores: how much each query likes each key.
    # Q @ K^T: compare every query to every key via dot product.
    # Q @ K^T: compare every query to every key via dot product.
    scores = Q @ K.T / np.sqrt(d_k)  # Scale by sqrt(d_k) so magnitudes stay controlled.
    scores = Q @ K.T / np.sqrt(d_k) #Scale by sqrt(d_k) so magnitudes stay controlled

    # Convert scores into attention weights (probabilities across sequence).
    attn_weights = softmax(scores, axis=-1)  # Each row sums to 1: "how much I attend to others."
    attn_weights = softmax(scores, axis=-1) #Each row sums to 1

    # Weighted sum of values: each position gets a mixture of all V's.
    # Weighted sum of values: each position gets a mixture of all V's.
    output = attn_weights @ V  # Mix information according to attention weights.
    output = attn_weights @ V #Mix info according to attention weights.


    # This is the final contextualized representation of the sequence.
    #This is the final contextualized representation of the sequence.
    return output
    return output
```