# Key-Value Cache Notes → Youtube: Efficient NLP The KV cache: Memory usage in transformers

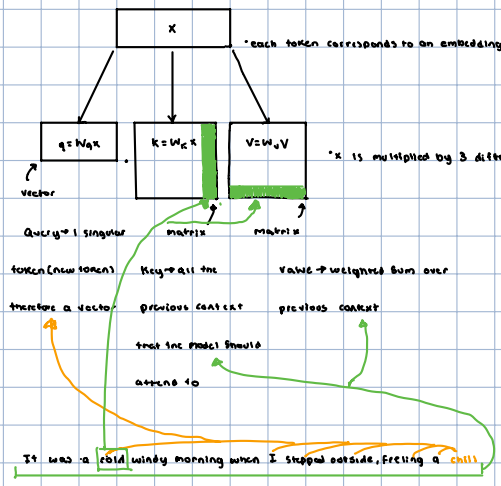
- as more and more text is generated you will use up more and more gpu memory

Why is this?

- most of the memory usage is taken up by the KV cache

Self-attention mechanism

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




- each token corresponds to an embedding vector x
- x is multiplied by 3 different matrices ($W_Q$, $W_V$, $W_K$) are all learned from data

- in auto-regressive decoding we are generating one token at a time given all of the previous context
- as we generate a sequence 1 token at a time the K and V matrices don't change very much
- cold the token here corresponds to 1 column in the weight matrix and a row of the x-matrix
- once we compute the embedding for this word it is not going to change again no matter how many words we generate
- the model still has to compute the key and the value matrix for all subsequent steps (resulting in a quadratic number of matrix-vector multiplications, which is really slow)

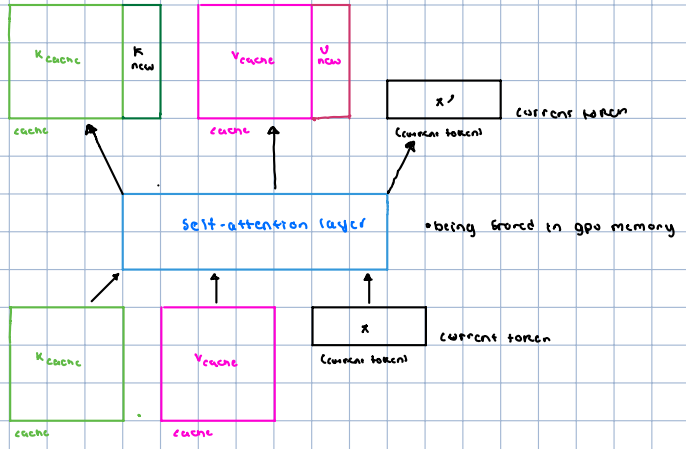
imagine you had a model generating a sentence 1 word at a time

- It was a cold windy morning when I stepped outside, feeling a chill crawl down my spine. The gusts of wind whipped → through
- It was a cold windy morning when I stepped outside, feeling a chill crawl down my spine. The gusts of wind whipped through → the
  - imagine for each word you write you have to read every word you have written before and you have to use that information to generate the next word
  - this is EXTREMELY INEFFICIENT

- when a model reads a new word - it generates a query vector


- instead we only have to compute 1 new column for the key matrix and 1 new column for the value matrix
- we then proceed with the dot product and softmax as usual

- we cache the previous values for the key and value matrices so we no longer have to compute these vectors for the previous context




Memory usage to store KV cache

2 x precision x $n_{layers}$ x $d_{model}$ x $\text{seqlen}$ x batch

2 = two matrices for K and V

precision = # of bytes per parameter batch = batch size

$n_{layers}$ = # of layers in a model

$d_{model}$ = dimensions of embeddings

seqlen = length of context in tokens