

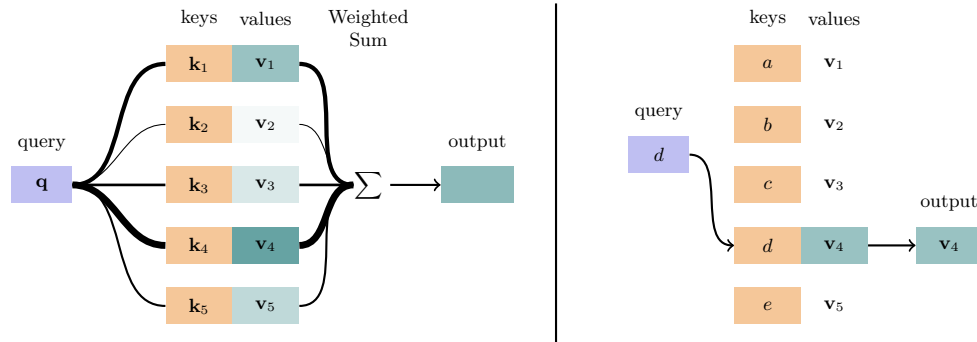


Figure 1: Attention is much like a soft lookup table. Queries are softly matched with keys (by dot product and subsequent softmax) and outputs are the resulting weighted sum of values.

Much of the course so far has been secretly introducing you to the intuitions that led to the most successful neural network architecture so far: the transformer. We know we want an architecture that can make gainful use of lots of parameters. We want residual connections and other optimization-helping tricks, to help train very very deep networks. We want compute efficiency, especially making use of the great parallelization properties of GPUs that we've just learned about. And we have intuitions about building up representations of tokens in a sequence via attention, looking at various places for information useful to the representation at hand.

Here's what we'll do:

- Review attention and introduce its multi-headed variant.
- Discuss future masking again.
- Discuss the other Transformer subcomponents, the feed-forward layer and layer normalization.
- Put together the Transformer block.
- Discuss sampling vs. prefill.
- Introduce variants for improving sampling memory usage, as well as RoPE.
- Maybe we'll get to linear attention, efficient non-materialization of the attention matrix (flash attention) and some other goodies.

Notation and Simple attention (again)

As we have come to know and love, let $x_1, \dots, x_n$ be a sequence over finite vocabulary $\mathcal{V}$. As usual, we embed the sequence as $E\mathbf{x}_1, \dots, E\mathbf{x}_n$. Perhaps we encode our position embeddings as we have before (though as we'll see, we may use a different position representation). That is,

$$E\mathbf{x}_1 + \mathbf{p}_1, \dots, E\mathbf{x}_n + \mathbf{p}_n \quad (1)$$

We'll call this sequence $\mathbf{h}_1, \dots, \mathbf{h}_n$. Recall that before, we were computing self-attention,

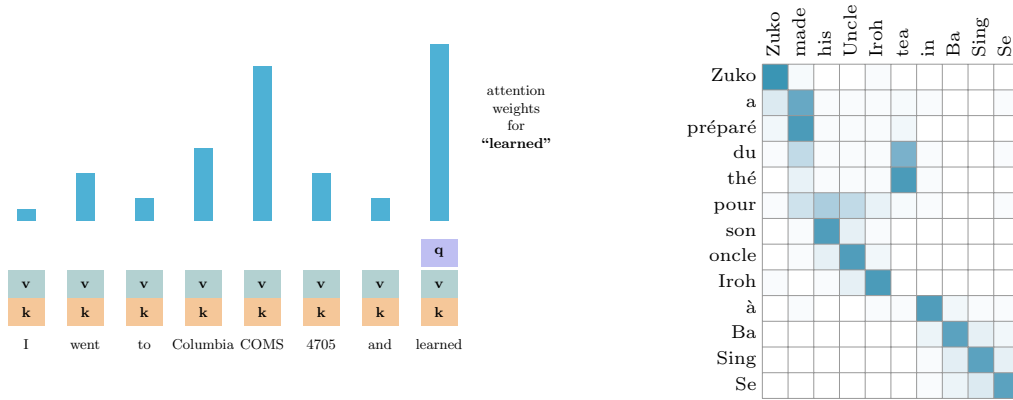


Figure 2: Diagram showing how a query may lead to varying attention weights across the rest of the sequence, and a plausible attention map for translation. At left, higher bars mean more attention. At right, darker blue means more attention.

as follows:

$$\mathbf{o}_i = \sum_{j=1}^i \alpha_{ij} \mathbf{h}_j \quad (2)$$

$$\alpha_{ij} = \frac{\exp(\mathbf{h}_i^\top \mathbf{h}_j)}{\sum_{k=1}^i \exp(\mathbf{h}_i^\top \mathbf{h}_k)} \quad (3)$$

So, this uses the vector $\mathbf{h}$ as three things: First, a **query**: the query is used to search for information in the sequence. Second, a **key**: the key is like a key in a dictionary, determining what types of queries should access the vector. Finally, a **value**: the value is what appears in the weighted sum; it's the information that gets passed.

The simplest thing we change first is to allow the keys, queries, and values to be different by linearly transforming each $\mathbf{h}_i$ separately for the three roles. That is, we set

$$\mathbf{q}_i = W_q \mathbf{h}_i \quad (4)$$

$$\mathbf{k}_i = W_k \mathbf{h}_i \quad (5)$$

$$\mathbf{v}_i = W_v \mathbf{h}_i \quad (6)$$

for linear transformations $W_q, W_k, W_v \in \mathbb{R}^{d \times d}$. So now, we compute:

$$\mathbf{o}_i = \sum_{j=1}^i \alpha_{ij} \mathbf{v}_j \quad (7)$$

$$\alpha_{ij} = \frac{\exp(\mathbf{q}_i^\top \mathbf{k}_j)}{\sum_{k=1}^i \exp(\mathbf{q}_i^\top \mathbf{k}_k)} \quad (8)$$

This allows a representation $\mathbf{h}_i$ of a token to be used differently for its different roles. Intuitively, the reason *why* I might want to look at $\mathbf{h}_i$ (i.e., the key) often times should differ from the **content** I want from $\mathbf{h}_i$ (i.e., the value.) And then both of these certainly might differ from how $\mathbf{h}_i$ might want to pull in information from previous tokens (the query.)

	Zuko	made	his	uncle	tea
Zuko	$\square$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
made	$\square$	$\square$	$-\infty$	$-\infty$	$-\infty$
his	$\square$	$\square$	$\square$	$-\infty$	$-\infty$
uncle	$\square$	$\square$	$\square$	$\square$	$-\infty$
tea	$\square$	$\square$	$\square$	$\square$	$\square$

Figure 3: Diagram of autoregressive future masking in self-attention. Words in each row have words in the future masked out (e.g., “Zuko” can only attend to “Zuko”, while “made” can attend to “Zuko” and “made”).

For a single head, let $\mathbf{h}_{1:n} \in \mathbb{R}^{n \times d}$ be the matrix with $\mathbf{h}_i^\top$ as its rows. Because $\mathbf{q}_i = W_q \mathbf{h}_i$, stacking the queries as rows gives $\mathbf{Q} = \mathbf{h}_{1:n} W_q^\top$ (in code, `H @ W_q.T`), and likewise for keys and values:

$$\mathbf{Q} = \mathbf{h}_{1:n} W_q^\top, \quad \mathbf{K} = \mathbf{h}_{1:n} W_k^\top, \quad \mathbf{V} = \mathbf{h}_{1:n} W_v^\top \in \mathbb{R}^{n \times d}. \quad (9)$$

To compute self-attention, we can compute our weights in matrix operations:

$$\alpha = \text{softmax}(\mathbf{QK}^\top) \in \mathbb{R}^{n \times n} \quad (10)$$

This α is $n \times n$: every query is dotted with every key. In the per-position equations above, the sums ran only up to i . Here those limits have to be enforced explicitly, because nothing in the matrix multiply stops index i from looking at $j > i$.

When performing language modeling like we’ve seen in this course (often called autoregressive modeling), we predict a word given all words so far:

$$\mathbf{w}_t \sim \text{softmax}(f(\mathbf{w}_{1:t-1})). \quad (11)$$

where f is function to map a sequence to a vector in $\mathbb{R}^{|\mathcal{V}|}$.

One crucial aspect of this process is that we can’t look at the future when predicting it—otherwise the problem becomes trivial.

In a Transformer, there’s nothing explicit in the self-attention weight α that says not to look at indices $j > i$ when representing token i . In practice, we enforce this constraint simply adding a large negative constant to the input to the softmax¹

$$e_{ij, \text{masked}} = \begin{cases} e_{ij} & j \leq i \\ -c & \text{otherwise, for } c \text{ very large} / \infty \end{cases} \quad (12)$$

In a diagram, it looks like Figure 3. We’ll write that additive logit mask as M , with $M_{ij} = 0$ if $j \leq i$ and $M_{ij} = -\infty$ (or a large negative constant) otherwise:

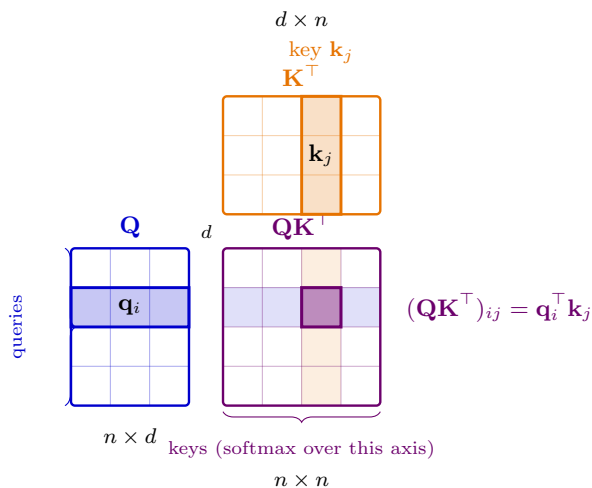
$$\alpha = \text{softmax}(\mathbf{QK}^\top + M) \in \mathbb{R}^{n \times n}. \quad (13)$$

¹It might seem like one should use $-\infty$ as the constant, to “really” ensure that you can’t see the future. However, this is often avoided; a modest constant within even the float range of the ‘float16’ encoding is used instead, like -10^4 . Using infinity can lead to NaNs and it’s sort of undefined how each library should treat infinite inputs, so we tend to avoid using it. And because of finite precision, a large enough negative constant will still set the attention weight to exactly zero.

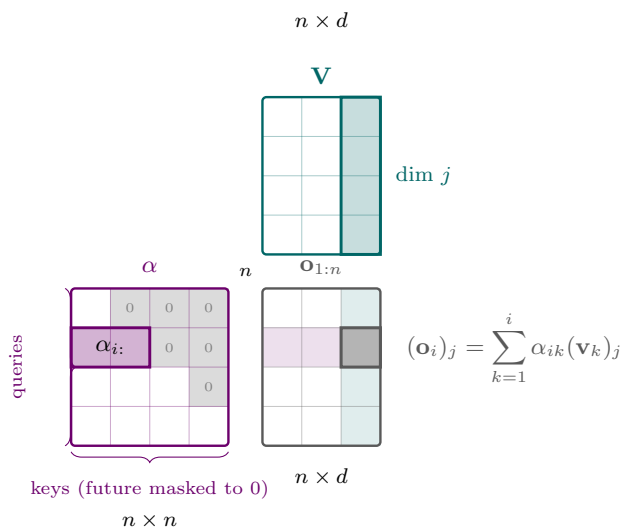
Then the self-attention operation for all positions is:

$$\mathbf{o}_{1:n} = \alpha \mathbf{V} = \text{softmax}(\mathbf{Q}\mathbf{K}^\top + M) \mathbf{V} \in \mathbb{R}^{n \times d}. \quad (14)$$

Here's a pair of diagrams showing the matrix ops, laid out so you can see what gets dotted with what. For the scores, $\mathbf{Q}$ is to the left of $\mathbf{Q}\mathbf{K}^\top$, sharing its n query rows, and $\mathbf{K}^\top$ is above, sharing its n key columns; the inner dimension d meets at the corner and is summed out:



Then for the weighted sum, row i of α (how query i weights every key) dots with column j of $\mathbf{V}$ (dimension j of every value), producing dimension j of the output $\mathbf{o}_i$. Because of the causal mask, entries $j > i$ of that row are already 0, so those value rows do not contribute:



Multi-head Self-Attention

Intuitively, a single call of self-attention is best at picking out a *single* value (on average) from the input value set. It does so softly, by averaging over all of the values, but it requires a balancing game in the key-query dot products in order to carefully average

two or more things. In one of your homeworks, you'll work through a bit of this intuition more carefully. What we'll present now, **multi-head self-attention**, intuitively applies self-attention multiple times at once, each with different key, query, and value transformations of the same input, and then combines the outputs.

For an integer number of heads H , we define per-head maps $W_q^{(\ell)}, W_k^{(\ell)}, W_v^{(\ell)} \in \mathbb{R}^{d_h \times d}$ for ℓ in $\{1, \dots, H\}$, where $d_h = d/H$ is the head dimensionality. (We'll see why we have the dimensionality reduction to d_h soon.) These are the query, key, and value matrices for each head. Correspondingly, we get queries, keys, and values as in single-head self-attention,

$$\mathbf{q}_i^{(\ell)} = W_q^{(\ell)} \mathbf{h}_i \quad (15)$$

$$\mathbf{k}_i^{(\ell)} = W_k^{(\ell)} \mathbf{h}_i \quad (16)$$

$$\mathbf{v}_i^{(\ell)} = W_v^{(\ell)} \mathbf{h}_i, \quad (17)$$

each now in $\mathbb{R}^{d_h}$.

We then perform self-attention with each head:

$$\mathbf{o}_i^{(\ell)} = \sum_{j=1}^i \alpha_{ij}^{(\ell)} \mathbf{v}_j^{(\ell)} \quad (18)$$

$$\alpha_{ij}^{(\ell)} = \frac{\exp(\mathbf{q}_i^{(\ell) \top} \mathbf{k}_j^{(\ell)})}{\sum_{j'=1}^i \exp(\mathbf{q}_i^{(\ell) \top} \mathbf{k}_{j'}^{(\ell)})} \quad (19)$$

Note that the output $\mathbf{o}_i^{(\ell)}$ of each head is in reduced dimension d_h . Finally, we define the output of multi-head self-attention as a linear transformation of the concatenation of the head outputs, letting $W_o \in \mathbb{R}^{d \times d}$:

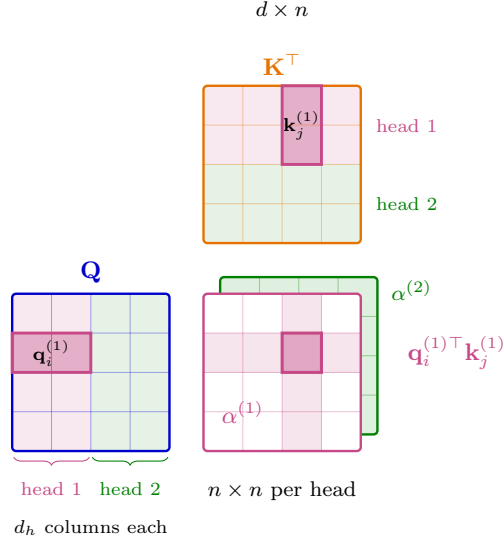
$$\mathbf{o}_i = W_o \left[\mathbf{o}_i^{(1)}; \dots; \mathbf{o}_i^{(H)} \right], \quad (20)$$

where we concatenate the H head outputs, each in $\mathbb{R}^{d_h}$, into a single vector in $\mathbb{R}^d$.

Sequence-tensor form. To understand why we have the reduced dimension of each head output, it's instructive to get a bit closer to how multi-head self-attention is implemented in code. In practice, **multi-head self-attention is no more expensive than single-head** due to the low-rankness of the transformations we apply.

When we perform multi-head self-attention in this matrix form, we pack the per-head maps into a single $W_q \in \mathbb{R}^{d \times d}$ (the vertical concatenation of the $W_q^{(\ell)}$), so $\mathbf{Q} = \mathbf{h}_{1:n} W_q^\top \in \mathbb{R}^{n \times d}$ as in single-head, and likewise for $\mathbf{K}$ and $\mathbf{V}$. We first reshape $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ each into a tensor of shape $\mathbb{R}^{n, H, d_h}$, splitting the model dimensionality into two axes, for the number of heads and the number of dimensions per head. We can then transpose the tensors to $\mathbb{R}^{H, n, d_h}$, which intuitively should look like H sequences of length n and dimensionality d_h . This allows us to perform the batched softmax operation in parallel across the heads, using the number of heads kind of like a **batch** axis (and indeed in practice we'll also have a separate batch axis.) So, the total computation (except the last linear transformation to combine the heads) is the same, just distributed across the (each lower-rank) heads. Here's a diagram like the single-head diagrams, demonstrating how the multi-head operation ends up much like the single-head operation. Each color

denotes one head (here $H = 2$): after the reshape, head ℓ 's slice of a query row only ever dots with head ℓ 's slice of a key column, producing one $n \times n$ attention matrix per head:



Feed-forward (MLP) Layer

Imagine if we were to stack self-attention layers. Would this be sufficient for a replacement for stacked LSTM layers? Intuitively, there's one thing that's missing: the elementwise nonlinearities that we've come to expect in standard deep learning architectures. In fact, if we stack two self-attention layers, we get something that looks a lot like a single self-attention layer:

$$o_i = \sum_{j=1}^n \alpha_{ij}^{(2)} W_v^{(2)} \left(\sum_{k=1}^n \alpha_{jk}^{(1)} W_v^{(1)} \mathbf{x}_k \right) \quad (21)$$

$$= \sum_{k=1}^n \left(\sum_{j=1}^n \alpha_{ij}^{(2)} \alpha_{jk}^{(1)} \right) W_v^{(2)} W_v^{(1)} \mathbf{x}_k \quad (22)$$

$$= \sum_{k=1}^n \alpha_{ik}^* W_v^* \mathbf{x}_k, \quad (23)$$

where $\alpha_{ik}^* = \sum_{j=1}^n \alpha_{ij}^{(2)} \alpha_{jk}^{(1)} = (\alpha^{(2)} \alpha^{(1)})_{ik}$, and $W_v^* = W_v^{(2)} W_v^{(1)}$. So, this is just a linear combination of a linear transformation of the input, much like a single layer of self-attention! Is this good enough?²

In practice, after a layer of self-attention, it's common to apply a feed-forward network independently to each word representation:

$$h_{\text{FF}} = W_2 \text{ReLU}(W_1 h_{\text{self-attention}} + b_1) + b_2, \quad (24)$$

where often, $W_1 \in \mathbb{R}^{5d \times d}$, and $W_2 \in \mathbb{R}^{d \times 5d}$. That is, the hidden dimension of the feed-forward network is substantially larger than the hidden dimension of the network, d —this is done because this matrix multiply is an efficiently parallelizable operation, so it's an efficient place to put a lot of computation and parameters.

²This question ends up having a nuanced answer that's out-of-scope for this note; ask me if you're interested in knowing more!

Layer Norm

One important learning aid in Transformers is *layer normalization* [Ba et al., 2016]. The intuition of layer norm is to reduce uninformative variation in the activations at a layer, providing a more stable input to the next layer. Further work shows that this may be most useful not in normalizing the forward pass, but actually in improving gradients in the backward pass [Xu et al., 2019].

To do this, layer norm (1) computes statistics across the activations at a layer to estimate the mean and variance of the activations, and (2) normalizes the activations with respect to those estimates, while (3) optionally learning (as parameters) an elementwise additive bias and multiplicative gain by which to sort of de-normalize the activations in a predictable way. The third part seems not to be crucial, and may even be harmful [Xu et al., 2019], so we omit it in our presentation.

One question to ask when understanding how layer norm affects a network is, “computing statistics over *what*?” That is, what constitutes a layer? In Transformers, the answer is always that statistics computed independently for a single index into the sequence length (and a single example in the batch) and shared across the d hidden dimensions. Put another way, the statistics for the token at index i won’t affect the token at index $j \neq i$.

So, we compute the statistics for a single index $i \in \{1, \dots, n\}$ as

$$\hat{\mu}_i = \frac{1}{d} \sum_{j=1}^d \mathbf{h}_{ij} \quad \hat{\sigma}_i = \sqrt{\frac{1}{d} \sum_{j=1}^d (\mathbf{h}_{ij} - \hat{\mu}_i)^2}, \quad (25)$$

where (as a reminder), $\hat{\mu}_i$ and $\hat{\sigma}_i$ are scalars, and we compute the layer norm as

$$\text{LN}(\mathbf{h}_i) = \frac{\mathbf{h}_i - \hat{\mu}_i}{\hat{\sigma}_i}, \quad (26)$$

where we’ve broadcasted the $\hat{\mu}_i$ and $\hat{\sigma}_i$ across the d dimensions of $\mathbf{h}_i$. Layer normalization is a great tool to have in your deep learning toolbox more generally.

Residual Connections

Residual connections simply add the *input* of a layer to the *output* of that layer:

$$f_{\text{residual}}(\mathbf{h}_{1:n}) = f(\mathbf{h}_{1:n}) + \mathbf{h}_{1:n}, \quad (27)$$

the intuition being that (1) the gradient flow of the identity function is *great* (the local gradient is 1 everywhere!) so the connection allows for learning much deeper networks, and (2) it is easier to learn the difference of a function from the identity function than it is to learn the function from scratch. As simple as these seem, they’re massively useful in deep learning, not just in Transformers!

Add & Norm. In the Transformer diagrams you’ll see, including Figure 4, the application of layer normalization and residual connection are often combined in a single visual block labeled *Add & Norm*. Such a layer might look like:

$$\mathbf{h}_{\text{pre-norm}} = f(\text{LN}(\mathbf{h})) + \mathbf{h}, \quad (28)$$

where f is either a feed-forward operation or a self-attention operation, (this is known as *pre-normalization*), or like:

$$\mathbf{h}_{\text{post-norm}} = \text{LN}(f(\mathbf{h}) + \mathbf{h}), \quad (29)$$

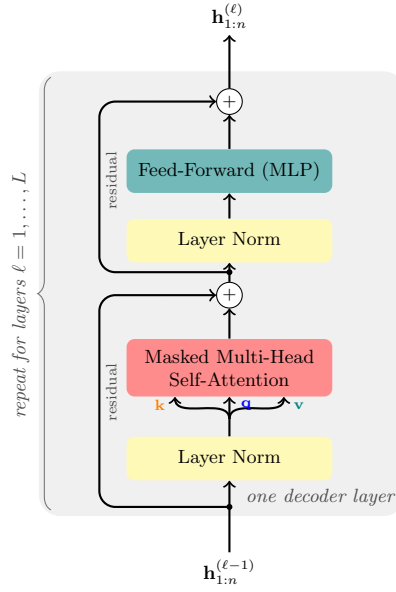


Figure 4: One layer of a Transformer decoder (pre-norm). The input is layer-normed and fed through masked multi-head self-attention (providing the queries, keys, and values), and the result is added back to the residual stream; the same pattern then repeats with the feed-forward network.

which is known as *post-normalization*. It turns out that the gradients of **pre-normalization** are much better at initialization, leading to much faster training [Xiong et al., 2020].

Attention logit scaling

Another trick introduced in [Vaswani et al., 2017] they dub *scaled dot product attention*. The dot product part comes from the fact that we’re computing dot products $\mathbf{q}_i^\top \mathbf{k}_j$. The intuition of scaling is that, when the dimensionality d_h of the vectors we’re dotting grows large, the standard deviation of the dot product of even random vectors (e.g., at initialization) grows roughly as $\sqrt{d_h}$. So, we normalize the dot products by $\sqrt{d_h}$ to stop this scaling:

$$\alpha = \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_h}} \right) \in \mathbb{R}^{n \times n} \quad (30)$$

Stacking layers

Putting the pieces together, Figure 4 shows one decoder layer: layer norm, then masked multi-head self-attention, then a residual connection; then layer norm, then the MLP, then another residual connection. A Transformer decoder is a stack of L of these layers (each with its own parameters), with the output $\mathbf{h}_{1:n}^{(\ell)}$ of layer ℓ serving as the input to layer $\ell + 1$.

Sampling vs prefill

Often in practice, we think of two different ways of running a transformer on a GPU: sampling and prefill. Sampling means generating tokens one at a time (according to

our usual autoregressive process) and prefill means building the representations for a whole sequence of tokens at once. The math is the same in both—the equations we’ve written don’t change. However, the information we have access to differs depending on whether we’re sampling (by our intention, we don’t know the future tokens yet) or prefilling (we know the whole sequence and want to run the model all at once.) As a concrete motivation, you’ll notice that *input* tokens are much cheaper than *output* tokens in common LLM API pricing schemes (like those of OpenAI or Anthropic.) This motivates why.

When performing prefill (say, to compute the log-likelihood of a known sequence,) we get to run our batched attention operation basically as written here. A few very large matrix multiplies. However, when sampling, the lack of knowledge of future tokens means that whenever a new token is generated, we’re constructing the activations *just for that one token* at each layer, and computing attention computations for all previous tokens. We thus need to have cached the activations for previous tokens, in order to go back and reference them when computing attention. In practice, we cache **key and value vectors**, calling it a **KV-cache**. At each layer of the transformer, we thus apply our residual connections and MLPs (those only depend on the current timestep activation) and then use our KV-cache to compute our attention scores over the history. Figure 5 shows the sampling-time picture: keys and values for the prefix live in the KV-cache, the current token’s key $\mathbf{k}_t$ and value $\mathbf{v}_t$ are appended to it, and the current query $\mathbf{q}_t$ is dotted against every key in the cache (including its own, since the mask allows $j \leq t$).

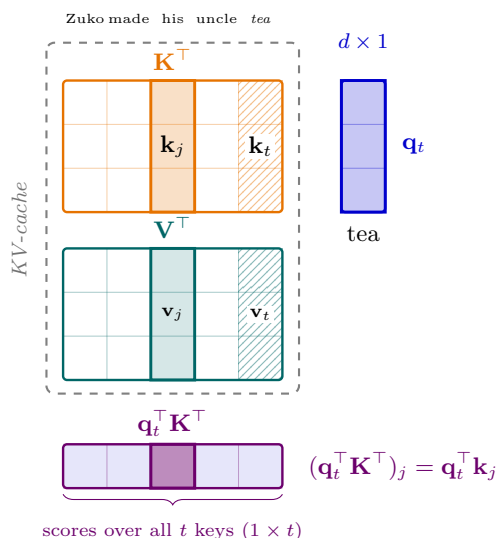


Figure 5: Sampling-time attention scores. The KV-cache holds keys and values for the prefix (here, *Zuko made his uncle*); the current token *tea* computes its own key $\mathbf{k}_t$ and value $\mathbf{v}_t$ and appends them (hatched). The current query $\mathbf{q}_t$ is to the right and is dotted against every key in the cache, including its own, producing a $1 \times t$ row of scores. Keys and values are drawn as columns, so the cached matrices appear transposed.

Grouped Query and Latent Attention

At sampling time, the KV-cache grows over time as we sample. Our sampled sequences may become very long (thousands to millions of tokens,) meaning the GPU memory cost

of keeping the KV cache around becomes prohibitive. There are a ton of systems-level ideas for computing (or approximating) this efficiently while limiting memory usage, but here we'll talk about two popular model architecture tweaks that reduce the total necessary memory for the KV-cache.

The first is **grouped query attention** [Ainslie et al., 2023], which keeps the number of queries as the number of heads H , but computes fewer keys and values, assigning multiple queries to each key/value pair. Intuitively, this approximates full multi-head attention by allowing for a lot of reasons to look up information (queries) but reduces the storage (and thus possible independent types of information.) Figure 6 (left) shows the grouping: pairs of query heads share a key and value, so the KV-cache stores G heads instead of H .

The second is **multi-head latent attention** (MLA) [DeepSeek-AI, 2024]. Instead of explicitly producing and storing keys and values, MLA computes and stores a single memory vector $c = W^{\text{DKV}}h$ for an activation h . The dimensionality d_{dkv} should be much less than the total dimensionalities of keys and values, (heads times dimensions per head times two, $2hd_h$.) At inference time, the memory vector is linearly transformed back up to $2hd_h$ (and reshaped) to be used as keys and values. Cool! Figure 6 (right) shows this: keys and values are not cached; only the compressed latent is.

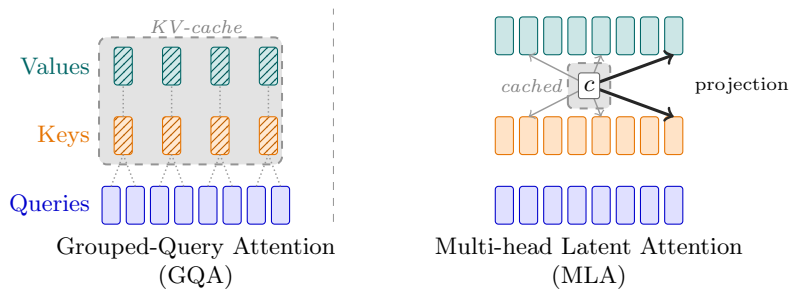


Figure 6: Grouped-query attention (left) caches grouped keys and values. Multi-head latent attention (right) caches a single context vector c per token, then projects it back to keys and values (solid arrows mark the last token). Adapted from Figure 3 of [DeepSeek-AI, 2024].

RoPE: Rotary Position Embeddings

tbd

Recurrence, linear attention

tbd

Efficient (non)-materialization

tbd

References

- [Ainslie et al., 2023] Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. (2023). GQA: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*.

- [Ba et al., 2016] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.
- [Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.
- [Baum and Petrie, 1966] Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563.
- [Bengio et al., 2000] Bengio, Y., Ducharme, R., and Vincent, P. (2000). A neural probabilistic language model. *Advances in neural information processing systems*, 13.
- [Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155.
- [Brown et al., 2020] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.
- [Collobert et al., 2011] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. P. (2011). Natural language processing (almost) from scratch. *CoRR*, abs/1103.0398.
- [Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.
- [DeepSeek-AI, 2024] DeepSeek-AI (2024). DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.
- [Devlin et al., 2019] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.
- [Elman, 1990] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.
- [Fukushima and Miyake, 1982] Fukushima, K. and Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer.
- [Lafferty et al., 2001] Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning, ICML ’01*, page 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.
- [Manning, 2022] Manning, C. D. (2022). Human Language Understanding & Reasoning. *Daedalus*, 151(2):127–138.
- [Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.
- [Press et al., 2022] Press, O., Smith, N., and Lewis, M. (2022). Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*.
- [Radford et al., 2019] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.
- [Raffel et al., 2020] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.
- [Rong, 2014] Rong, X. (2014). word2vec parameter learning explained. *CoRR*, abs/1411.2738.
- [Rumelhart et al., 1985] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.
- [Rumelhart et al., 1988] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA.

- [Schütze, 1992] Schütze, H. (1992). Dimensions of meaning. In *Proceedings of the 1992 ACM/IEEE Conference on Supercomputing*, Supercomputing '92, page 787–796, Washington, DC, USA. IEEE Computer Society Press.
- [Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.
- [Workshop et al., 2022] Workshop, B., :, Scao, T. L., Fan, A., Akiki, C., Pavlick, E., Ilić, S., Hesslow, D., Castagné, R., Luccioni, A. S., Yvon, F., Gallé, M., Tow, J., Rush, A. M., Biderman, S., Webson, A., Ammanamanchi, P. S., Wang, T., Sagot, B., Muennighoff, N., del Moral, A. V., Ruwase, O., Bawden, R., Bekman, S., McMillan-Major, A., Beltagy, I., Nguyen, H., Saulnier, L., Tan, S., Suarez, P. O., Sanh, V., Laurençon, H., Jernite, Y., Launay, J., Mitchell, M., Raffel, C., Gokaslan, A., Simhi, A., Soroa, A., Aji, A. F., Alfassy, A., Rogers, A., Nitzav, A. K., Xu, C., Mou, C., Emezue, C., Klammer, C., Leong, C., van Strien, D., Adelani, D. I., Radev, D., Ponferrada, E. G., Levkovizh, E., Kim, E., Natan, E. B., De Toni, F., Dupont, G., Kruszewski, G., Pistilli, G., Elshahar, H., Benyamina, H., Tran, H., Yu, I., Abdulmumin, I., Johnson, I., Gonzalez-Dios, I., de la Rosa, J., Chim, J., Dodge, J., Zhu, J., Chang, J., Froberg, J., Tobing, J., Bhattacharjee, J., Almubarak, K., Chen, K., Lo, K., Von Werra, L., Weber, L., Phan, L., allal, L. B., Tanguy, L., Dey, M., Muñoz, M. R., Masoud, M., Grandury, M., Šaško, M., Huang, M., Coavoux, M., Singh, M., Jiang, M. T.-J., Vu, M. C., Jauhar, M. A., Ghaleb, M., Subramani, N., Kassner, N., Khamis, N., Nguyen, O., Espejel, O., de Gibert, O., Villegas, P., Henderson, P., Colombo, P., Amuok, P., Lhoest, Q., Harliman, R., Bommasani, R., López, R. L., Ribeiro, R., Osei, S., Pyysalo, S., Nagel, S., Bose, S., Muhammad, S. H., Sharma, S., Longpre, S., Nikpoor, S., Silberberg, S., Pai, S., Zink, S., Torrent, T. T., Schick, T., Thrush, T., Danchev, V., Nikoulina, V., Laippala, V., Lepercq, V., Prabhu, V., Alyafeai, Z., Talat, Z., Raja, A., Heinzerling, B., Si, C., Taşar, D. E., Salesky, E., Mielke, S. J., Lee, W. Y., Sharma, A., Santilli, A., Chaffin, A., Stiegler, A., Datta, D., Szczechla, E., Chhablani, G., Wang, H., Pandey, H., Strobelt, H., Fries, J. A., Rozen, J., Gao, L., Sutawika, L., Bari, M. S., Al-shaibani, M. S., Manica, M., Nayak, N., Teehan, R., Albanie, S., Shen, S., Ben-David, S., Bach, S. H., Kim, T., Bers, T., Fevry, T., Neeraj, T., Thakker, U., Raunak, V., Tang, X., Yong, Z.-X., Sun, Z., Brody, S., Uri, Y., Tojari, H., Roberts, A., Chung, H. W., Tae, J., Phang, J., Press, O., Li, C., Narayanan, D., Bourfoune, H., Casper, J., Rasley, J., Ryabinin, M., Mishra, M., Zhang, M., Shoenberger, M., Peyrounette, M., Patry, N., Tazi, N., Sanseviero, O., von Platen, P., Cornette, P., Lavallée, P. F., Lacroix, R., Rajbhandari, S., Gandhi, S., Smith, S., Reuquena, S., Patil, S., Dettmers, T., Barua, A., Singh, A., Cheveleva, A., Ligozat, A.-L., Subramonian, A., Névol, A., Lovering, C., Garrette, D., Tunuguntla, D., Reiter, E., Taktasheva, E., Voloshina, E., Bogdanov, E., Winata, G. I., Schoelkopf, H., Kalo, J.-C., Novikova, J., Forde, J. Z., Clive, J., Kasai, J., Kawamura, K., Hazan, L., Carpuat, M., Clinciu, M., Kim, N., Cheng, N., Serikov, O., Antverg, O., van der Wal, O., Zhang, R., Zhang, R., Gehrmann, S., Mirkin, S., Pais, S., Shavrina, T., Scialom, T., Yun, T., Limisiewicz, T., Rieser, V., Protasov, V., Mikhailov, V., Punkschatkun, Y., Belinkov, Y., Bamberger, Z., Kasner, Z., Rueda, A., Pestana, A., Feizpour, A., Khan, A., Faranak, A., Santos, A., Hevia, A., Undrea, A., Aghagol, A., Abdollahi, A., Tammour, A., HajiHosseini, A., Behroozi, B., Ajibade, B., Saxena, B., Ferrandis, C. M., Contractor, D., Lansky, D., David, D., Kiela, D., Nguyen, D. A., Tan, E., Baylor, E., Ozoani, E., Mirza, F., Ononiwu, F., Rezanejad, H., Jones, H., Bhattacharya, I., Solaiman, I., Sedenko, I., Nejadgholi, I., Passmore, J., Seltzer, J., Sanz, J. B., Dutra, L., Samagaio, M., Elbadri, M., Mieskes, M., Gerchick, M., Akinlolu, M., McKenna, M., Qiu, M., Ghauri, M., Burynok, M., Abrar, N., Rajani, N., Elkott, N., Fahmy, N., Samuel, O., An, R., Kromann, R., Hao, R., Alizadeh, S., Shubert, S., Wang, S., Roy, S., Viguer, S., Le, T., Oyebade, T., Le, T., Yang, Y., Nguyen, Z., Kashyap, A. R., Palasciano, A., Callahan, A., Shukla, A., Miranda-Escalada, A., Singh, A., Beilharz, B., Wang, B., Brito, C., Zhou, C., Jain, C., Xu, C., Fourrier, C., Perinán, D. L., Molano, D., Yu, D., Manjavacas, E., Barth, F., Fuhrmann, F., Altay, G., Bayrak, G., Burns, G., Vrabec, H. U., Bello, I., Dash, I., Kang, J., Giorgi, J., Golde, J., Posada, J. D., Sivaraman, K. R., Bulchandani, L., Liu, L., Shinzato, L., de Bykhovetz, M. H., Takeuchi, M., Pàmies, M., Castillo, M. A., Nezhurina, M., Sängler, M., Samwald, M., Cullan, M., Weinberg, M., De Wolf, M., Mihaljcic, M., Liu, M., Freidank, M., Kang, M., Seelam, N., Dahlberg, N., Broad, N. M., Muellner, N., Fung, P., Haller, P., Chandrasekhar, R., Eisenberg, R., Martin, R., Canalli, R., Su, R., Su, R., Cahyawijaya, S., Garda, S., Deshmukh, S. S., Mishra, S., Kiblawi, S., Ott, S., Sang-aroonisiri, S., Kumar, S., Schweter, S., Bharati, S., Laud, T., Gigant, T., Kainuma, T., Kusa, W., Labrak, Y., Bajaj, Y. S., Venkatraman, Y., Xu, Y., Xu, Y., Xu, Y., Tan, Z., Xie, X., Xie, Y., Z., Bras, M., Belkada, Y., and Wolf, T. (2022). Bloom: A 176b-parameter open-access multilingual language model.
- [Xiong et al., 2020] Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T. (2020). On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pages 10524–10533. PMLR.
- [Xu et al., 2019] Xu, J., Sun, X., Zhang, Z., Zhao, G., and Lin, J. (2019). Understanding and improving layer normalization. *Advances in Neural Information Processing Systems*, 32.