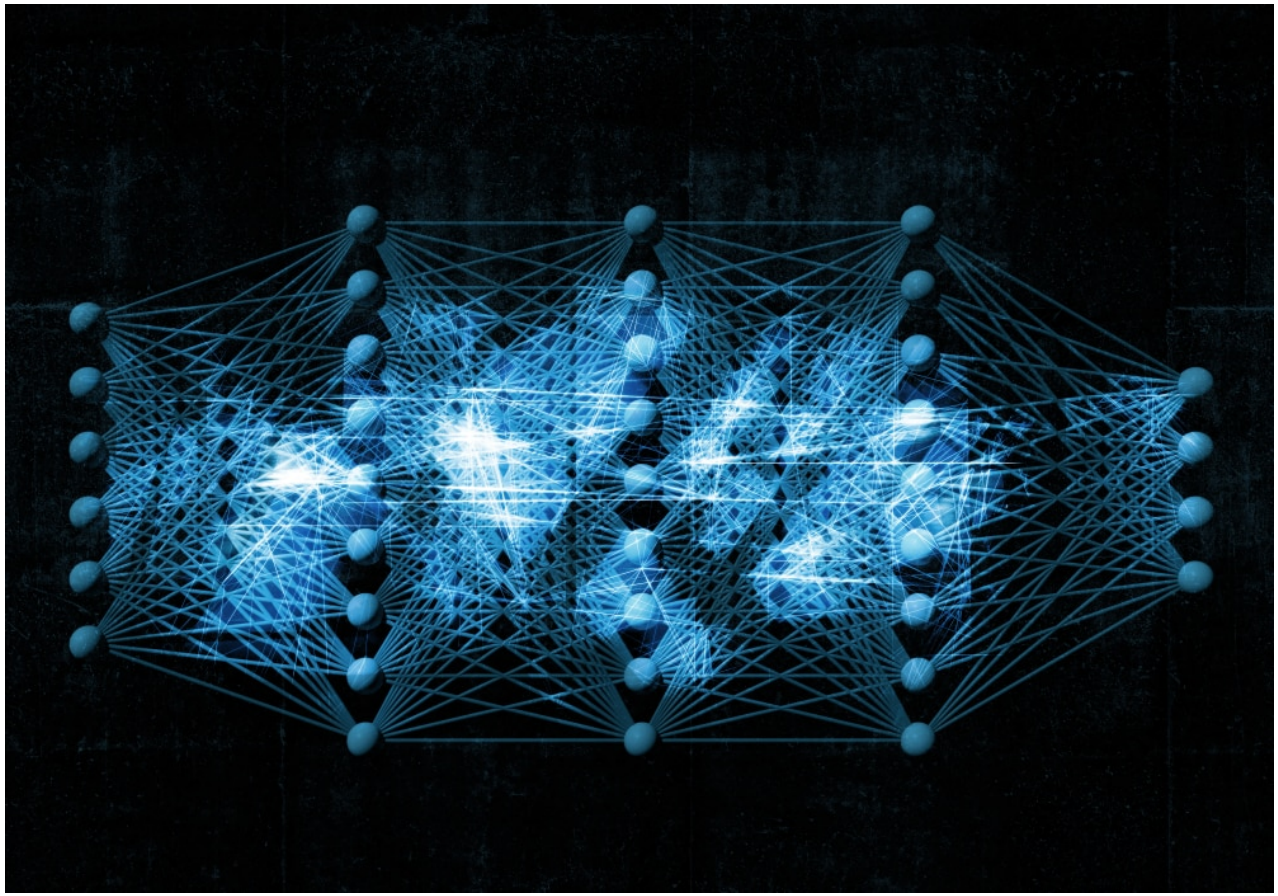
Natural Language Processing

Lecture : Large Language Models & Inference

Master Degree in Computer Engineering
University of Padua
Lecturer : Giorgio Satta

Inference



©Shutterstock

Preliminary notation:

- $\mathbf{x}$: the input token sequence $x_0 \cdots x_m$, called **prompt**, where x_0 is a special start symbol
- $\mathbf{y}$: the output token sequence $y_1 \cdots y_n$, called **response**
- $\mathbf{y}_{<i}$: the output tokens $y_1 \cdots y_{i-1}$
- $P(\mathbf{y} \mid \mathbf{x})$: the **conditional** probability of generating $\mathbf{y}$ given $\mathbf{x}$
- $[\mathbf{x}; \mathbf{y}]$: the **concatenation** of sequences $\mathbf{x}$ and $\mathbf{y}$
- $P([\mathbf{x}; \mathbf{y}])$: the **joint** probability of sequence $[\mathbf{x}; \mathbf{y}]$
- $P(y_i \mid \mathbf{x}, \mathbf{y}_{<i})$ is a shortcut for $P(y_i \mid [\mathbf{x}; \mathbf{y}_{<i}])$

Inference

Once we have pre-trained and fine-tuned an LLM, we can apply it to generate responses given a prompt.

This process is called **inference** and is generally expressed in the following form

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{x})}{\operatorname{argmax}} \; P(\mathbf{y} \mid \mathbf{x})$$

where $\mathcal{Y}(\mathbf{x})$ represents the search space.

In machine learning, this is called a **structured prediction problem**, to distinguish it from a classification problem.

This is one of the most widely encountered problems in NLP.

Inference

LLM inference can be decomposed in two tasks

- **model computation**: we need to efficiently compute $P(\mathbf{y} \mid \mathbf{x})$ with our LLM
- **search**: we need to find the optimal (or sub-optimal) output sequence in $\mathcal{Y}(\mathbf{x})$

We need to make a trade-off between accuracy and efficiency, by carefully combining techniques for the two tasks.

Considering the log-scale of probability $P(\mathbf{y} \mid \mathbf{x})$ we can write

$$\log P(\mathbf{y} \mid \mathbf{x}) \quad = \quad \log P([\mathbf{x}; \mathbf{y}]) - \log P(\mathbf{x})$$

We can calculate the probability of the two sequences above using the **chain rule**. For instance

$$\begin{aligned}
\log P(\mathbf{x}) &= \log P(x_0 \cdots x_m) \\
&= \log[P(x_0)P(x_1 \mid x_0) \cdots P(x_m \mid x_0 \cdots x_{m-1})] \\
&= \underbrace{\log P(x_0)}_{=0} + \sum_{j=1}^{m} \log P(x_j \mid \mathbf{x}_{<j}) \\
&= \sum_{j=1}^{m} \log P(x_j \mid \mathbf{x}_{<j})
\end{aligned}$$

Model computation

In common implementations of LLM, however, we do not compute the log of the **joint** probability of these two sequences.

Instead, we use the LLM to directly compute the log of the **conditional** probability of the output sequence $\log P(\mathbf{y} \mid \mathbf{x})$ in the following form

$$\log P(\mathbf{y} \mid \mathbf{x}) = \sum_{i=1}^{n} \log P(y_i \mid \mathbf{x}, \mathbf{y}_{<i})$$

Model computation

Model computation is based on a pre-trained, decoder-only transformer, which outputs a probability distribution over vocabulary V for the token y_i after $[\mathbf{x}, \mathbf{y}_{<i}]$

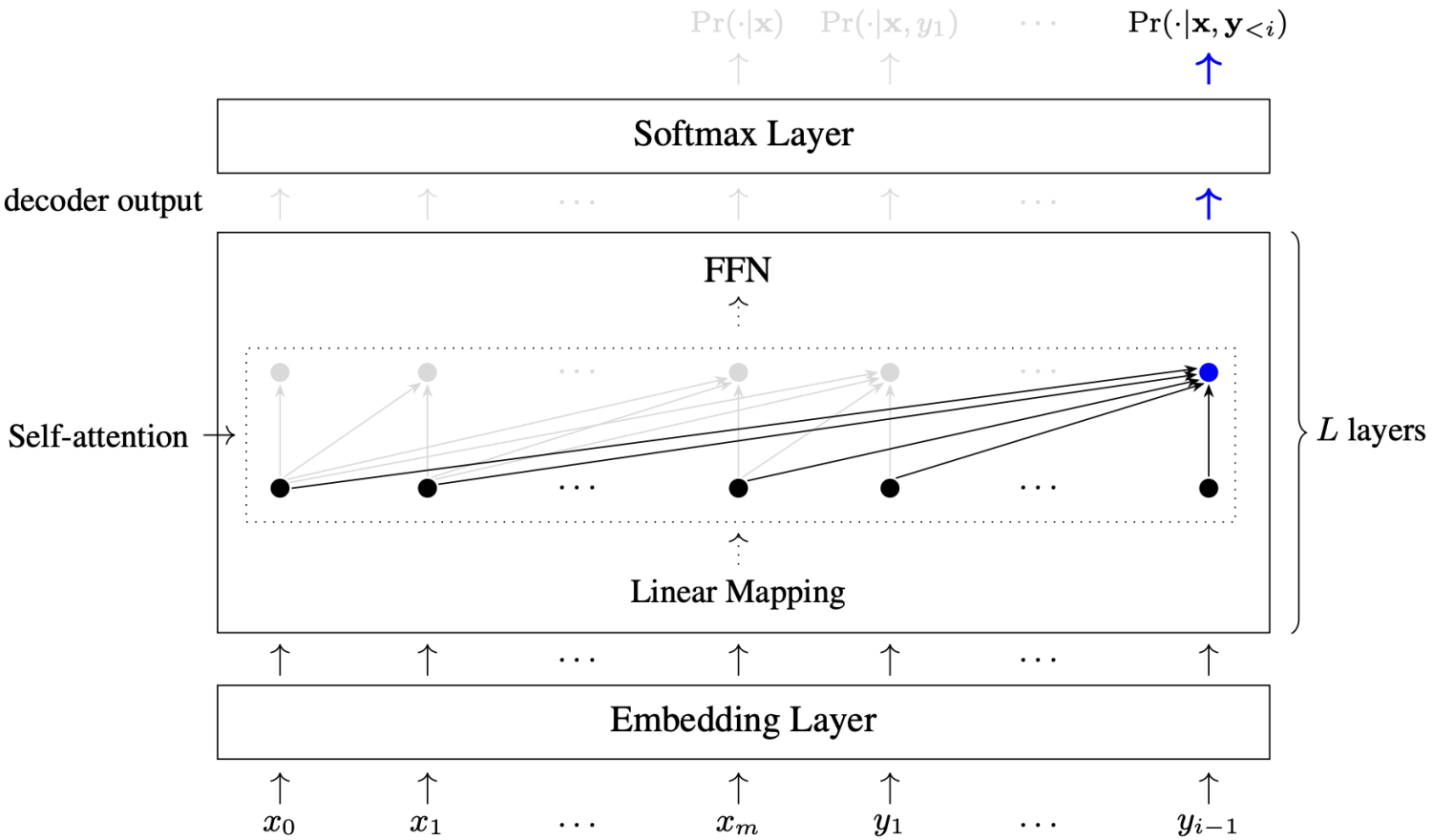
Recall that the prompt has length $m+1$ and the response generated so far has length $i-1$, so that $[\mathbf{x}, \mathbf{y}_{<i}]$ has lenght $m+i$.

$$\begin{aligned} \log P(\cdot \mid \mathbf{x}, \mathbf{y}_{<i}) &= \text{softmax}(\mathbf{H}\mathbf{W}^o)_{m+i} \\ \mathbf{H} &= \text{decoder}([\mathbf{x}, \mathbf{y}_{<i}]) \end{aligned}$$

where

- $\mathbf{H} \in \mathbb{R}^{(m+i)\times d}$, d the hidden vector dimension
- $\mathbf{W}^o \in \mathbb{R}^{d\times|V|}$
- softmax is performed only at position $m+i$

Model computation



Model computation

The transformer decoding network consists of an embedding network and a number of stacked self-attention and FFN networks.

The difficulty of inference is in part from the use of the **self-attention** mechanism

$$\text{Att}(\mathbf{q}_{m+i}, \mathbf{K}, \mathbf{V}) \quad = \quad \text{softmax}\left(\frac{\mathbf{q}_{m+i}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V}$$

where

- $\mathbf{q}_{m+i} \in \mathbb{R}^{(1\times d)}$ is the query for response token y_i
- $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{(m+i)\times d}$ are the keys and values up to $m+i$, respectively

Model computation

At each inference step the model attends position $m + i$ to all previous positions, which results in

- $m + i$ vector products for $\mathbf{q}_{m+i}$ and $\mathbf{K}^\top$
- $m + i$ vector products for softmax $\left(\frac{\mathbf{q}_{m+i}\mathbf{K}^\top}{\sqrt{d}}\right)$ and $\mathbf{V}$

for a total of $2(m + i)$ vector products.

Generating a sequence of length ℓ through a number L of layers has a time complexity of $\mathcal{O}(L \cdot \ell^2)$ for the self-attention network.

Alternative models have focused on efficient methods that are faster than this quadratic time complexity.

Model computation

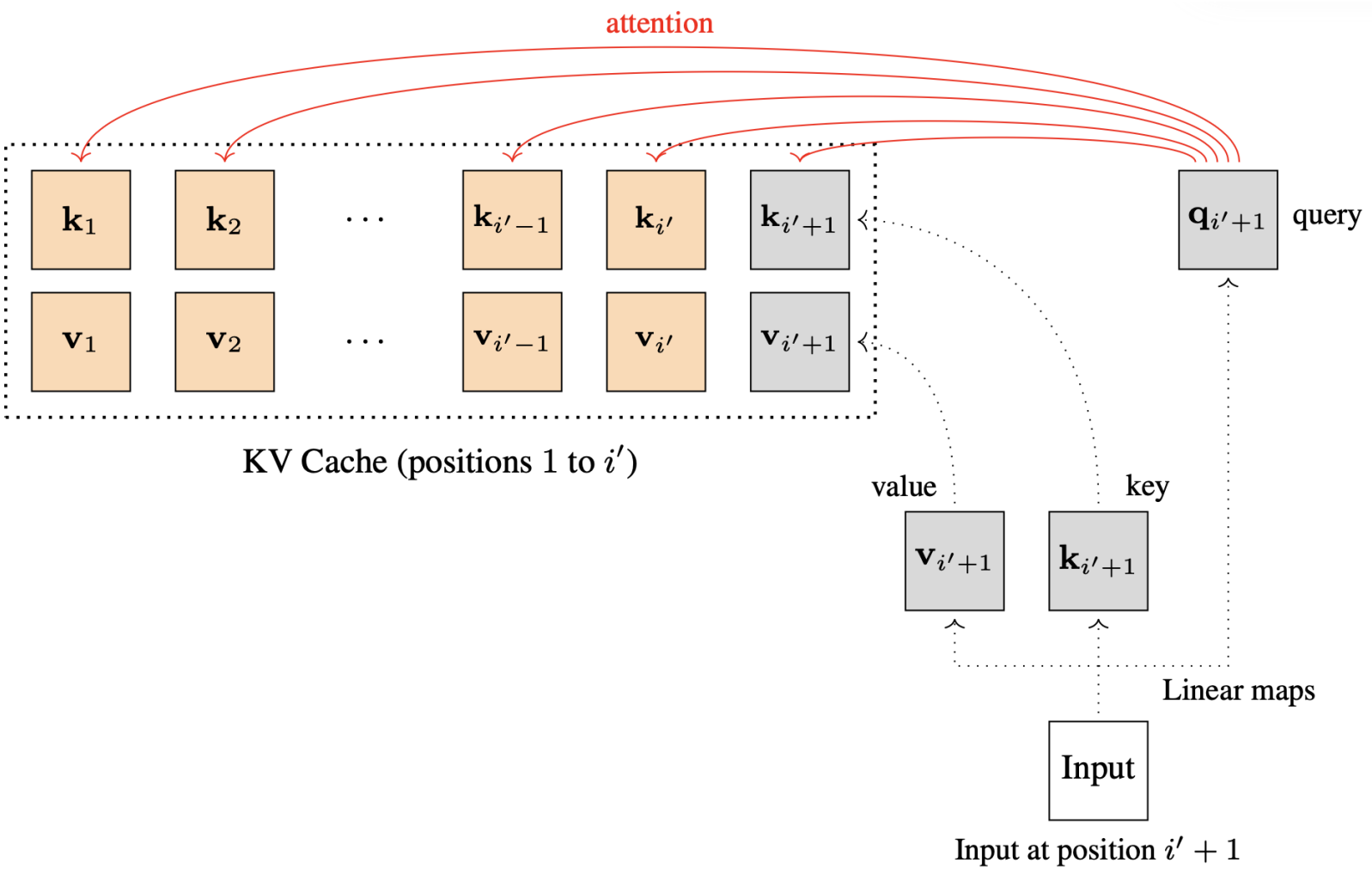
Once a new key-value pair is generated, it is stored in a structure called the **KV cache**.

Let $(\mathbf{K}, \mathbf{V})$ be the current KV cache. This cache is updated as follows

$$\begin{aligned} \mathbf{K} &\leftarrow \text{append}(\mathbf{K}, \mathbf{k}_{m+i}) \\ \mathbf{V} &\leftarrow \text{append}(\mathbf{V}, \mathbf{v}_{m+i}) \end{aligned}$$

where $(\mathbf{k}_{m+i}, \mathbf{v}_{m+i})$ is the newly generated key-value pair at position $m + i$. In the next figure: $i' = m + i$.

Model computation



Model computation

It is natural to divide inference into two phases.

The **prefilling** phase computes the KV cache for the input sequence $\mathbf{x}$. Since $\mathbf{x}$ is given, this can be done in one single step.

The **decoding** phase continues generating tokens based on the KV cache. This needs to be carried out at one token per step.

We specify and compare these two phases in the next slides.

Prefilling can be considered an encoding process, since we do not need to generate tokens.

Unlike BERT, which generates bidirectional sequence representations, prefilling is based on standard language modeling tasks, and is thus **unidirectional**.

Prefilling

Prefilling is carried out using a **masking** array, to ensure that each token only attends to itself and the tokens in the past.

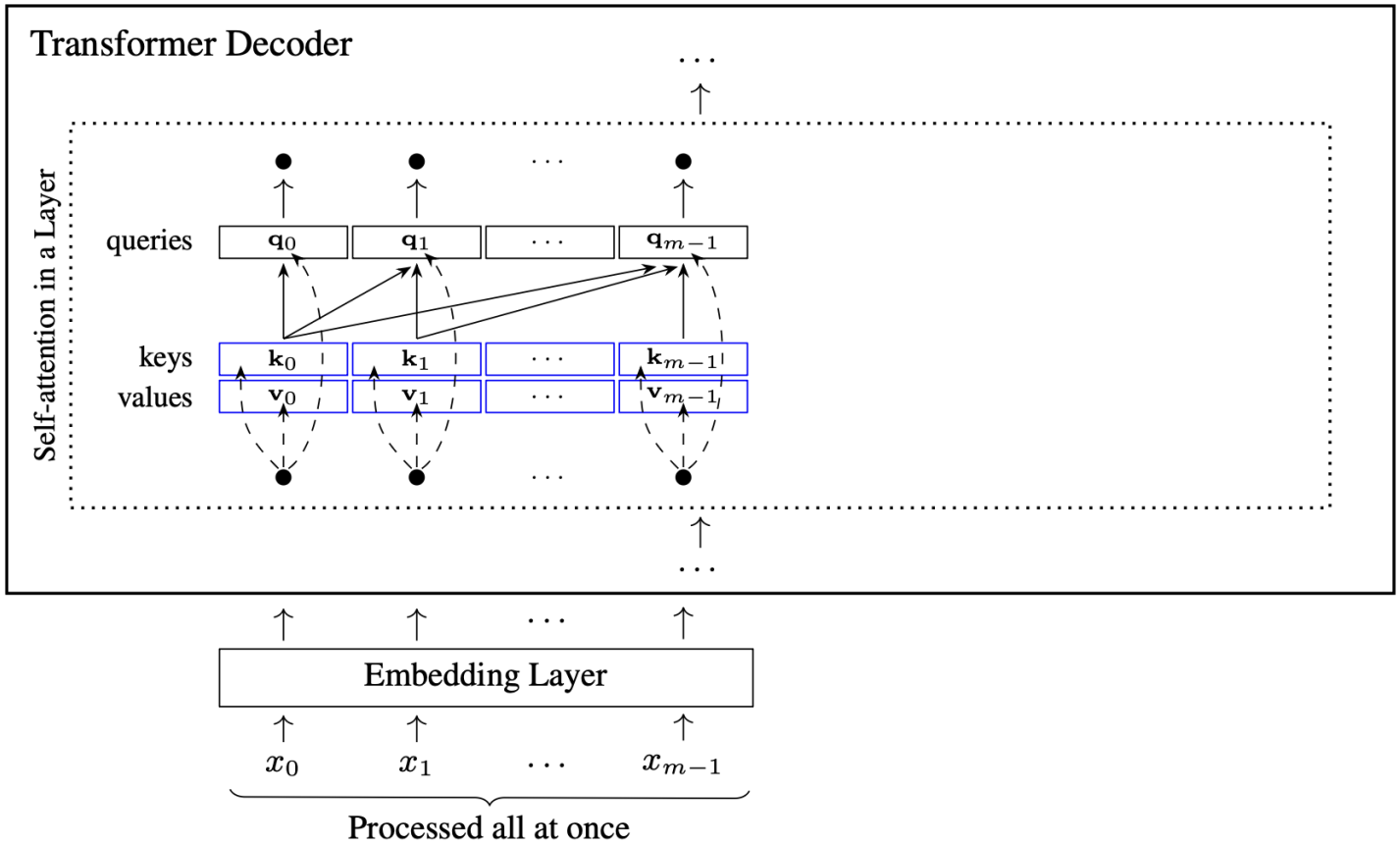
$$\text{Att}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \quad = \quad \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}} + \mathbf{M}\right)\mathbf{V}$$

where

- $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{(m+1)\times d}$
- $\mathbf{M} \in \mathbb{R}^{(m+1)\times(m+1)}$

Values of $\mathbf{M}$ corresponding to future tokens, that is, query q_i and key k_j with $i < j$, are set to $-\infty$.

Prefilling

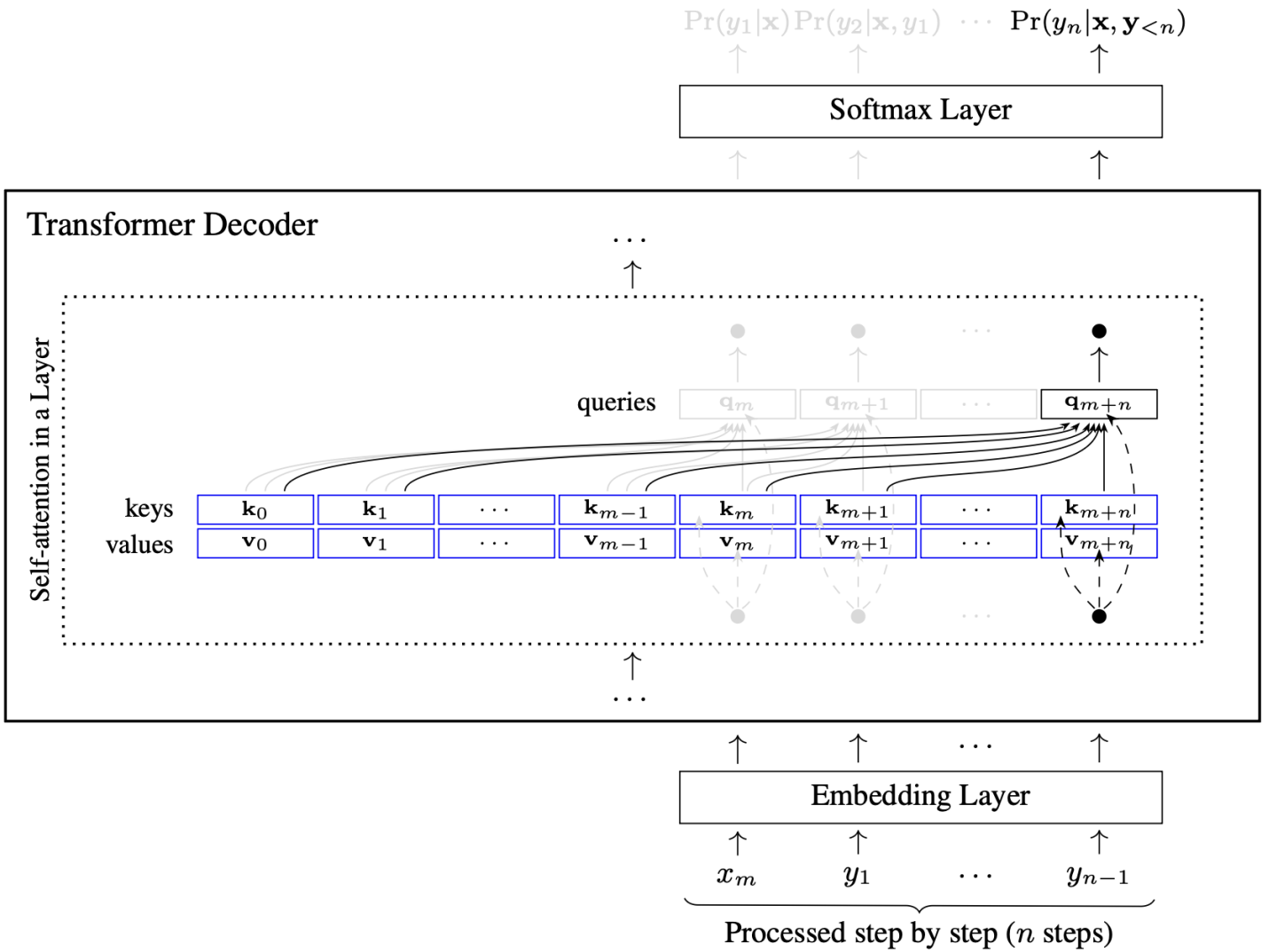

Transformer Decoder
Self-attention in a Layer
queries
keys
values
$\mathbf{q}_0$ $\mathbf{q}_1$ $\cdots$ $\mathbf{q}_{m-1}$
$\mathbf{k}_0$ $\mathbf{k}_1$ $\cdots$ $\mathbf{k}_{m-1}$
$\mathbf{v}_0$ $\mathbf{v}_1$ $\cdots$ $\mathbf{v}_{m-1}$
Embedding Layer
x_0 x_1 $\cdots$ x_{m-1}
Processed all at once

Prefilling

Prefilling process is considered **compute-bound**, meaning that the bottleneck is in the computational capacity rather than in the memory bandwidth.

This is because merging multiple computational operations into one operation reduces the number of memory requirements and data transfers.

Decoding


$\Pr(y_1|\mathbf{x})\Pr(y_2|\mathbf{x},y_1)\ \cdots\ \Pr(y_n|\mathbf{x},\mathbf{y}_{<n})$
Softmax Layer
Transformer Decoder
Self-attention in a Layer
queries
keys
values
$\mathbf{q}_m$ $\mathbf{q}_{m+1}$ $\cdots$ $\mathbf{q}_{m+n}$
$\mathbf{k}_0$ $\mathbf{k}_1$ $\cdots$ $\mathbf{k}_{m-1}$ $\mathbf{k}_m$ $\mathbf{k}_{m+1}$ $\cdots$ $\mathbf{k}_{m+n}$
$\mathbf{v}_0$ $\mathbf{v}_1$ $\cdots$ $\mathbf{v}_{m-1}$ $\mathbf{v}_m$ $\mathbf{v}_{m+1}$ $\cdots$ $\mathbf{v}_{m+n}$
Embedding Layer
x_m y_1 $\cdots$ y_{n-1}
Processed step by step (n steps)

Decoding

Decoding generates the response part one token at a time.

The decoding process is **memory-bound**, due to its frequent access to the KV cache.

The cost of decoding grows significantly as more tokens are generated. In most cases, decoding is computationally more expensive than prefilling.

Decoding

Comparison between prefilling and decoding phases.

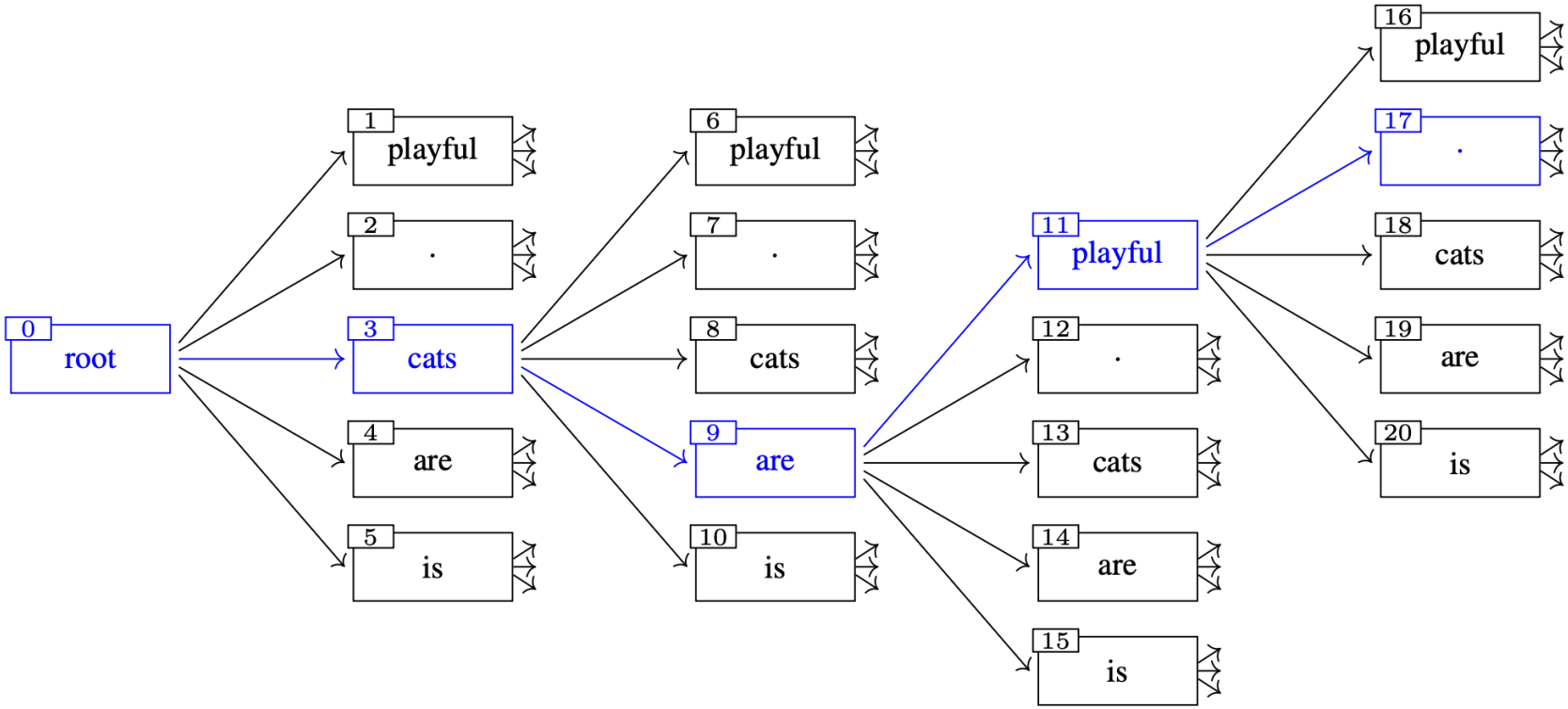
	Prefilling	Decoding
Goal	Set up initial context $\mathbf{x}$.	Continue generating tokens $\mathbf{y}$ after the initial input.
All-at-once Visibility	Tokens in $\mathbf{x}$ are presented all at once.	Tokens in $\mathbf{y}$ are presented sequentially, that is, predicting a token requires waiting for the previous tokens to be predicted first.
Context Use	Build the context or encoded representation of the input.	Use the cached key-value pairs (from prefilling) to generate further tokens.
Resource Limitation	Compute-bound	Memory-bound
Computational Cost	High	Very High

The search problem for LLM inference can be re-expressed as

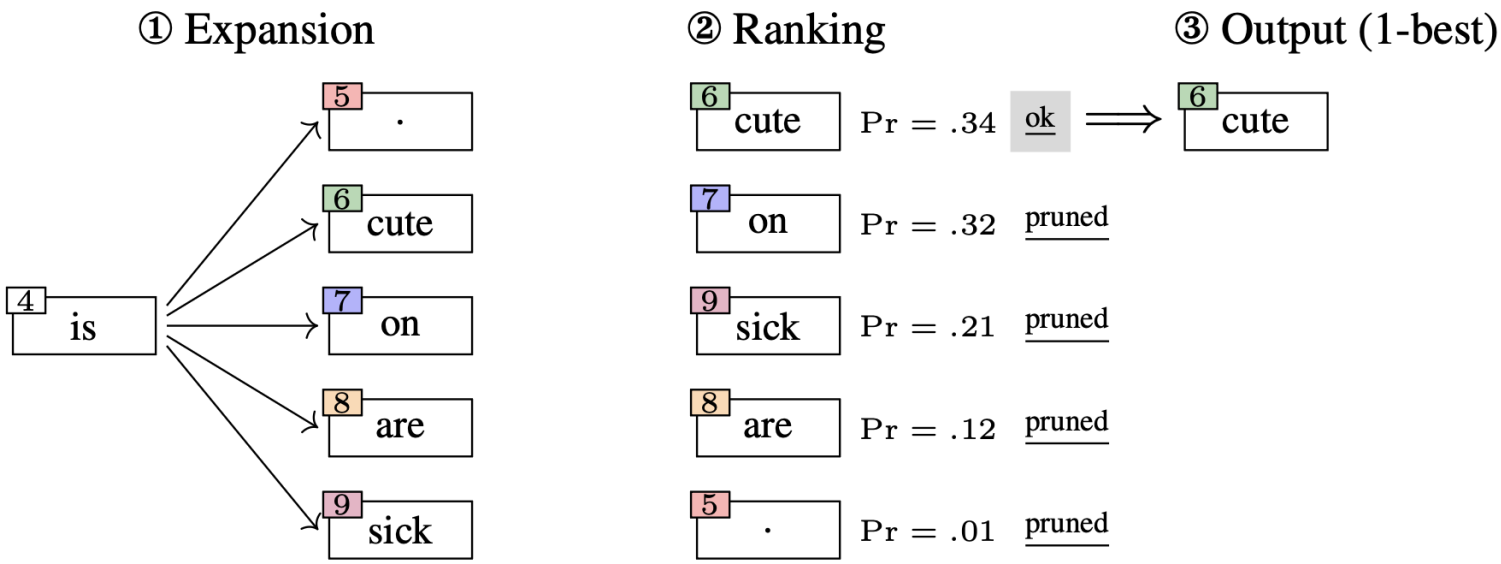
$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{x})}{\operatorname{argmax}} \; P(\mathbf{y} \mid \mathbf{x})$$

where $\mathcal{Y}$ is commonly represented in a **tree** data structure to facilitate search.

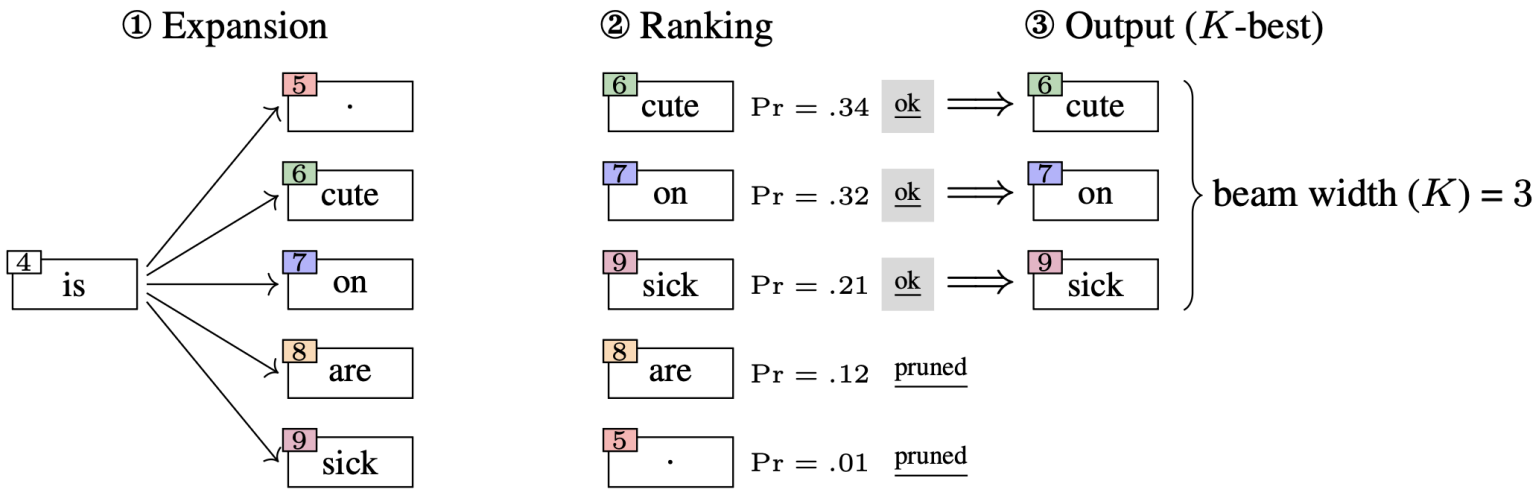
Search


0
root
1
playful
2
.
3
cats
4
are
5
is
6
playful
7
.
8
cats
9
are
10
is
11
playful
12
.
13
cats
14
are
15
is
16
playful
17
.
18
cats
19
are
20
is

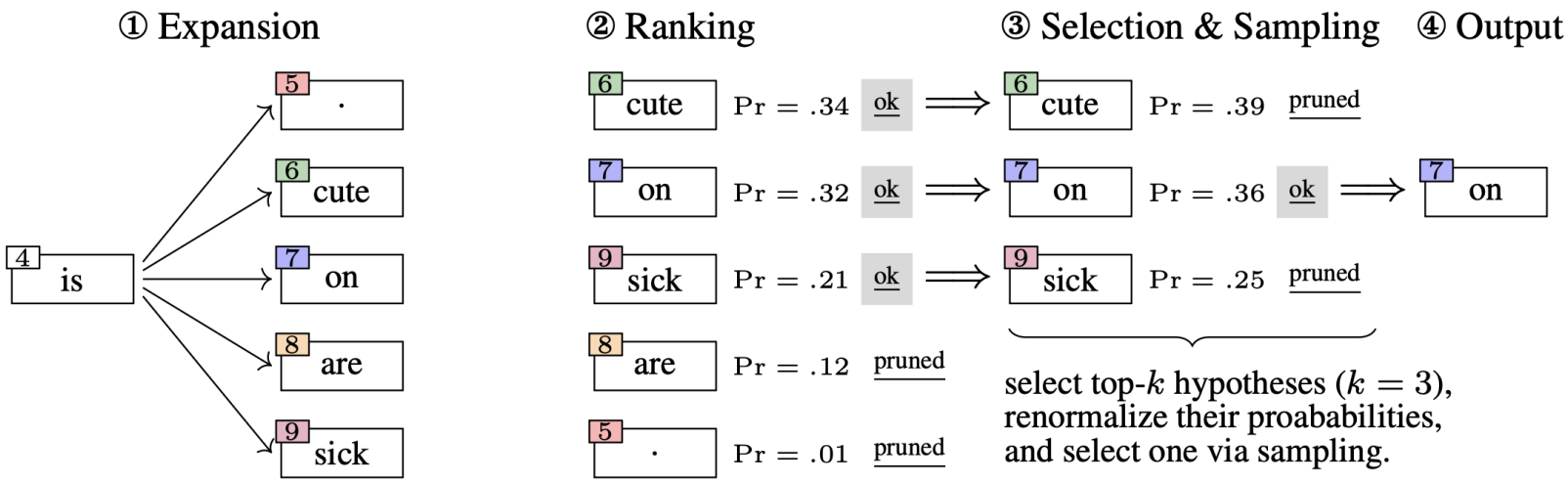
Example : Greedy search.



Example : Beam search.



Example : Top-k search.



Example : Top-p search.