Training Large Language Models (LLMs)

Prof. Volkan Cevher
volkan.cevher@epfl.ch

Lecture 1: Architectures

Laboratory for Information and Inference Systems (LIONS)
École Polytechnique Fédérale de Lausanne (EPFL)

EE-628 (Spring 2025)

License Information for Training LLMs Slides

- This work is released under a Creative Commons License with the following terms:
- **Attribution**
  - The licensor permits others to copy, distribute, display, and perform the work. In return, licensees must give the original authors credit.
- **Non-Commercial**
  - The licensor permits others to copy, distribute, display, and perform the work. In return, licensees may not use the work for commercial purposes – unless they get the licensor's permission.
- **Share Alike**
  - The licensor permits others to distribute derivative works only under a license identical to the one that governs the licensor's work.
- Full Text of the License

Acknowledgments

- These slides could not have been possible without the help of LIONS group members
  - ▷ Leyla Candogan, Mete Erdogan, Yongtao Wu, Arshia Afzal, Thomas Pethick, Wanyun Xie, Elias Abad Rocamora, Zhenyu Zhu, Pol Puigdemont, Frank Zhengqing Wu, and Francesco Tonin.

Logistics

- **Credits:** 4
- **Lectures:** Thursday 9:00-12:00
- **Prerequisites:** Strong foundations in machine learning, deep learning, and optimization; experience with large-scale models is recommended.
- **Moodle:** My courses > Genie electrique et electronique (EL) > Master > EE-628

syllabus & course outline.

Logistics for online teaching

- **Zoom link for video lectures:**

  `https://epfl.zoom.us/j/64289434614`

- **Mediaspace@EPFL channel for recorded videos:**

  `https://mediaspace.epfl.ch/channel/EE-628+Training+Large+Language+Models/101371`

- **Moodle:**

  `https://moodle.epfl.ch/course/view.php?id=18742`

Outline

- Motivation and basics of LM
- General LLM framework
- 1. Token processing
- 2. Sequence mixing
- 3. Channel processing
- Example architectures

Remark about notation

The LLM literature might use a different notation:

	Our lectures	DL literature
data/sample	$\mathbf{a}$	$\mathbf{x}$
label	b	y
bias	μ	b
weight	$\mathbf{x}, \mathbf{X}$	$\mathbf{w}, \mathbf{W}$
SSM parameters	$\mathbf{X}^A, \mathbf{X}^B, \mathbf{X}^C$	$\mathbf{A}, \mathbf{B}, \mathbf{C}$

A motivation for language models (LMs)

Example

Predict the next word w given the following source sentence S_{source}?

S_{source} : *"On January 1 people usually say happy new [w]."*

A motivation for language models (LMs)

Example

Predict the next word w given the following source sentence S_{source}?

S_{source} : *"On January 1 people usually say happy new [w]."*

Question:

- Why is this important?
  - spelling & grammar correction $\quad \mathrm{p}(\text{year}|S_{\text{source}}) > \mathrm{p}(\text{years}|S_{\text{source}})$
  - machine translation $\quad \mathrm{p}(S_{\text{translation 1}}|S_{\text{source}}) > \mathrm{p}(S_{\text{translation 2}}|S_{\text{source}})$
  - sentence classification $\quad \mathrm{p}(S_{\text{class 1}}|S_{\text{source}}) > \mathrm{p}(S_{\text{class 2}}|S_{\text{source}})$
  - speech recognition $\quad \mathrm{p}(w|S_{\text{source}})$
  - chatbot $\quad \mathrm{p}(w|S_{\text{source}})$
  - (more generally) labeling, automated decisions,...

Basics for language models (LMs) – I

Definition (Language model [35])

Models that assign probabilities to sequences of words are called language models.

Remarks:

- ○ Given a sentence with T words: $S = w_{1:T} = (w_1, \ldots, w_T)$, by the chain rule of probability:

$$\mathsf{p}(S) = \mathsf{p}(w_{1:T}) = \mathsf{p}(w_1)\mathsf{p}(w_2|w_1)\mathsf{p}(w_3|w_{1:2})s\mathsf{p}(w_T|w_{1:T-1}) = \prod_{t=1}^{T} \mathsf{p}(w_t|w_{1:t-1})$$

- ○ Implicitly, we are enforcing a graphical model that takes “time” into account.

Example

If $S = w_{1:3} =$ “happy new year”, then $\mathsf{p}(S) = \mathsf{p}(\text{happy})\mathsf{p}(\text{new}|\text{happy})\mathsf{p}(\text{year}|\text{happy new})$.

Basics for language models (LMs) – II

Question: ○ How can we compute $\mathsf{p}(w_t|w_{1:t-1})$?

Remarks: ○ A trivial solution: Just count the frequency on a large corpus, e.g.,

$$\mathsf{p}(\text{year}|S_{\text{source}}) = \frac{\mathsf{p}(S_{\text{source}} + \text{year})}{\mathsf{p}(S_{\text{source}})} \approx \frac{\#(\text{On January 1 people usually say happy new year})}{\#(\text{On January 1 people usually say happy new})}$$

○ But the language is creative, there are several ways to express the same meaning.

○ The sentence above might even not appear on the corpus.

○ We need better ways to estimate such probabilities!

N-gram LMs

Markov assumption [42]

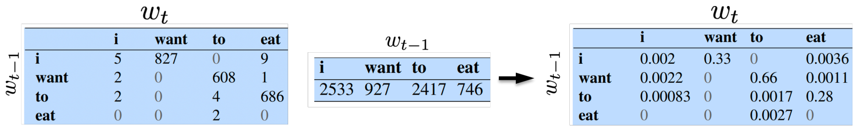
The probability of a word only depends on the last $N-1$ words as

$$\mathsf{p}(w_t|w_{1:t-1}) = \mathsf{p}(w_t|w_{t-N:t-1}) \approx \frac{\#(w_{t-N:t})}{\#(w_{t-N:t-1})}.$$

Markov in 1913 [42] used "Markov chains" to predict whether the upcoming letter would be a vowel or a consonant.

Example

In the bigram LM ($N=2$), we only need to estimate $\mathsf{p}(w_t|w_{t-1}) \approx \frac{\#(w_{t-1:t})}{\#(w_{t-1})}$ to generate text.


w_t
w_{t-1}
w_{t-1}
w_t
w_{t-1}

	i	want	to	eat
i	5	827	0	9
want	2	0	608	1
to	2	0	4	686
eat	0	0	2	0

i	want	to	eat
2533	927	2417	746

	i	want	to	eat
i	0.002	0.33	0	0.0036
want	0.0022	0	0.66	0.0011
to	0.00083	0	0.0017	0.28
eat	0	0	0.0027	0

Figure: Count (Left) and probability $\mathsf{p}(w_t|w_{1:t-1})$ (Right) from the Berkeley Restaurant Project corpus of 9332 sentences [35].

Towards pre-training an N-gram LM

- In natural language processing (NLP), we use tokens to represent words coming from a vocabulary $\mathcal{V}$.

Terminologies:

- A *token* is the smallest unit that can be assigned a meaning to be processed.
  - In English, a token often corresponds to a word.
  - However, a single token can also encode compound words like *New York*.
  - In Chinese or Japanese, there is no space between words.
  - In these languages, sentence segmentation is required before we tokenize.
- We indicate the beginning and the end of sentences with tokens $\langle \mathrm{BOS} \rangle$ and $\langle \mathrm{EOS} \rangle$.
  - S_{source} "$\langle \mathrm{BOS} \rangle$ Happy new year $\langle \mathrm{EOS} \rangle$" has $T = 5$ tokens.
- The size of our vocabulary is denoted as $|\mathcal{V}|$.
- *Pre-training*: building a LM based on a large corpus in a (often) self-supervised manner.
- *Inference*: Using a trained LM to do next word prediction.

N-gram LMs: “Pre-training” & Inference

- The following simplified examples show the difficulty of pre-training and inference with 2-gram LMs.

“Pre-training”
1. Count $\#(w_{t-1})$ and $\#(w_{t-1:t})$ over the corpus.
2. Obtain probability $\mathsf{p}(w_t \| w_{t-1})$ over the corpus.

Inference

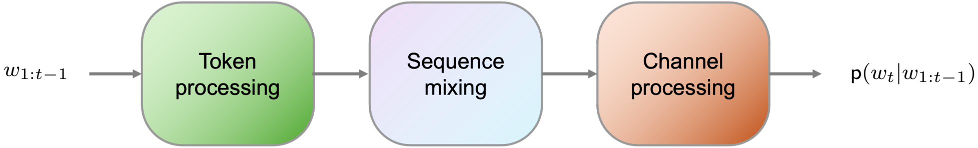
```
1. Set $w_1$ as $\langle\mathrm{BOS}\rangle$, $t = 1$.
2. While True:
    ▶ $w_{t+1} = \arg\max_{w \in \mathcal{V}} \mathsf{p}(w|w_t)$
    ▶ If $w_{t+1}$ is $\langle\mathrm{EOS}\rangle$: break
    ▶ $t = t + 1$
3. Output: $[w_1, s, w_{t+1}]$.
```

Remarks:

- Need to store the probability for all N-gram pairs.
- Language is creative, some new N-gram pairs might not even appear on the corpus.
- Cannot incorporate earlier words than N due to the Markov assumption.

$$\mathsf{p}(\text{two} \mid \text{one plus one equals}) = \mathsf{p}(\text{two} \mid \text{it is wrong that one plus one equals})?$$

A more generalized LLM framework



Token processing

- Converts words into a suitable format.
- Tokenization, embedding, positional encoding...

Sequence mixing

- Captures dependencies across tokens.
- FFN, RNN, Attention, Linear Attention, SSMs...

Channel processing

- Applies transformations within each token representation.
- Normalization, output projection, classification layers...

Token Processing: Word representations

Question:

- ○ How can we numerically represent a word/meaning?

Remarks:

- ○ Osgood et al. 1957 [47] uses 3 numbers to represent a word.
  - ▶ valence: the pleasantness of the stimulus
  - ▶ arousal: the intensity of emotion provoked by the stimulus
  - ▶ dominance: the degree of control exerted by the stimulus

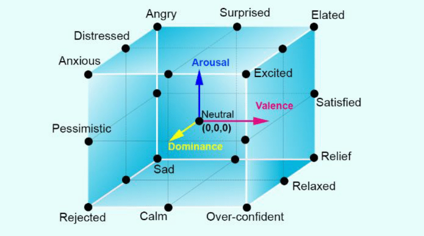
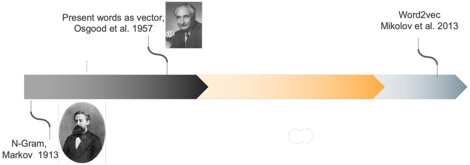


Figure: From [32].

Word embeddings



Definition (Word embeddings [35])

Vectors for representing words are called word embeddings.

- ○ We will briefly introduce two words embeddings:
- ○ One-hot representation: sparse and long word embedding in $\mathbb{R}^{|\mathcal{V}|}$.
  - ▶ Training is not required—trivial to obtain.
  - ▶ Not a good way to capture the underlying meaning—cannot measure similarity.
- ○ Word2vec [43]: a framework to learn dense and concise word embedding.
  - ▶ Training is required.
  - ▶ Better characterization for the meaning of a word, e.g., the similarity can be computed by similarity metrics.
  - ▶ Cosine similarity or inner products work!

Word2vec: Setup

- An illustration of a target word and context words in a ± 2 window size:

$$\ldots\underbrace{\text{people usually}}_{\text{context words}}\ \underbrace{\text{say}}_{\text{target word}}\ \underbrace{\text{happy new}}_{\text{context words}}\ldots$$

- Word2vec uses learnable parameters $\boldsymbol{X}^c$ and $\boldsymbol{X}^t$ to present two embeddings for each word,
  - $\boldsymbol{X}^c$ corresponds to the embedding when it is as a context word.
  - $\boldsymbol{X}^t$ corresponds to the embedding when it is as a target word
  - They satisfy the following relationship:

$$\mathbf{a}_i^t = \boldsymbol{X}^t \boldsymbol{e}_i \in \mathbb{R}^m, \quad \mathbf{a}_i^c = \boldsymbol{X}^c \boldsymbol{e}_i \in \mathbb{R}^m,$$

where $\boldsymbol{e}_i \in \mathbb{R}^{|\mathcal{V}|}$ is the one hot representation for each word, $i \in 1, \ldots |\mathcal{V}|$ and m is the embedding dimension.

Remarks:

- The window size for the context is a hyperparameter.
- The final embedding can be the summation or concatenation of these two embeddings.

Word2vec: Training

- Core idea: Given a pair of words (w_i, w_j), return the probability that w_j is the context word of w_i (i.e., `true`).

A simple approach: $\mathsf{p}(\texttt{true}|(w_t, w_c)) = \sigma(\langle \mathbf{a}_t^t, \mathbf{a}_c^c\rangle) = \dfrac{1}{1+\exp(-\langle \mathbf{a}_t^t, \mathbf{a}_c^c\rangle)}$, where σ is the sigmoid activation.

- Given a tuple (w_t, w_c, w_n), we have the following ingredients
  - w_t is the target word.
  - w_c is one of its context words(positive samples)
  - w_n is not its context word (negative sample)—e.g., chosen via unigram (1-Gram) probability.
  - A loss function:

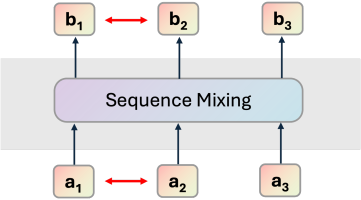
$$
\begin{aligned}
L &= -\log\left(\mathsf{p}(\texttt{true}|(w_t, w_c))\mathsf{p}(\texttt{false}|(w_t, w_n))\right)\\
&= -\log \mathsf{p}(\texttt{true}|(w_t, w_c)) - \log \mathsf{p}(\texttt{false}|(w_t, w_n))\\
&= -\log \sigma(\langle \mathbf{a}_t^t, \mathbf{a}_c^c\rangle) - \log(1 - \sigma(\langle \mathbf{a}_t^t, \mathbf{a}_n^c\rangle))\\
&= -\log \frac{1}{1+\exp(-\langle \boldsymbol{X}^t\boldsymbol{e}_t, \boldsymbol{X}^c\boldsymbol{e}_c\rangle)} - \log\left(1 - \frac{1}{1+\exp(-\langle \boldsymbol{X}^t\boldsymbol{e}_t, \boldsymbol{X}^c\boldsymbol{e}_n\rangle)}\right)
\end{aligned}
$$

- Crawl the corpus to obtain these tuples, and minimize L (e.g., with stochastic gradient descent).

Token processing: Positional embeddings

Question: ○ How can we consider the relative position of each word in the sequence?

Observation: ○ If we switch the order of $\mathbf{a}_1$ and $\mathbf{a}_2$, the output $\mathbf{b}_3$ should not remain the same.



I am happy ≠ Am I happy

Token processing: Positional embeddings

Question:

- ○ How can we consider the relative position of each word in the sequence?

Solution 1?

- ○ Absolute position via trivial concatenation of the word embedding $\mathbf{a}_t$ with its index t.

$$\text{Pos}(\mathbf{a}_t) = \text{Concatenate}[\mathbf{a}_t, t] \,.$$

- ○ As t grows, so do the values.
- ○ Hard to extrapolate on sequence with unseen length.

Token processing: Positional embeddings

Question: ○ How can we consider the relative position of each word in the sequence?

Solution 2 [65]: ○ Absolute position via trigonometric functions of different frequencies. For $t = 1, \ldots, T$:

$$\mathrm{Pos}(\mathbf{a}_t) = \mathbf{a}_t + \begin{pmatrix} \sin\left(t/10000^{2\times 1/m}\right) \\ \cos\left(t/10000^{2\times 1/m}\right) \\ \vdots \\ \sin\left(t/10000^{2\times \frac{m}{2}/m}\right) \\ \cos\left(t/10000^{2\times \frac{m}{2}/m}\right) \end{pmatrix}$$

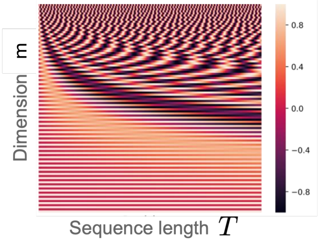


Figure: From [74]

Token processing: Positional embeddings

Question: ○ How can we consider the relative position of each word in the sequence?

Solution 2 [65]: ○ Absolute position via trigonometric functions of different frequencies. For $t = 1, \ldots, T$:

$$\text{Pos}(\mathbf{a}_t) = \mathbf{a}_t + \begin{pmatrix} \sin\left(t/10000^{2\times 1/m}\right) \\ \cos\left(t/10000^{2\times 1/m}\right) \\ \vdots \\ \sin\left(t/10000^{2\times \frac{m}{2}/m}\right) \\ \cos\left(t/10000^{2\times \frac{m}{2}/m}\right) \end{pmatrix}$$

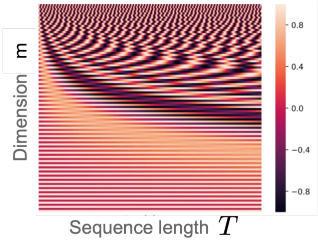


Figure: From [74]

Solution 3: ○ *Rotary position embedding [61]: incorporate both absolute position and relative position.

Token processing: Positional embeddings

Question: ◦ How can we consider the relative position of each word in the sequence?

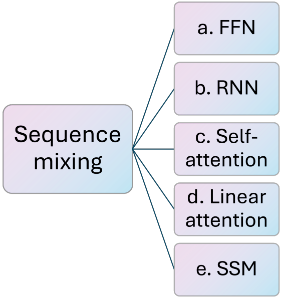
Remarks:

◦ Positional embeddings improve performance in many models (eq. transformers [65]).

◦ They limit the inference sequence length with training sequence length.

◦ Models, such as decay masked linear attentions [62] or SSMs [24, 15], do not need them.

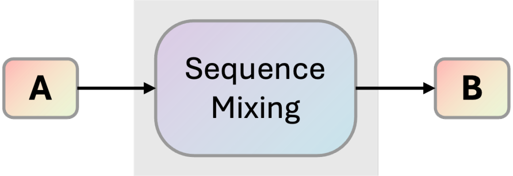
Sequence mixing



- Most important and well-studied part of the LLM framework.
- Captures dependencies across tokens.

Notation

$\mathbf{A} \in \mathbb{R}^{T \times m}, \mathbf{B} \in \mathbb{R}^{T \times d}$ where T is the sequence length and m is the embedding and d is the output dimension.



a. Feed forward neural networks (FFN) as sequence mixers [7]

- Core idea: use most recent N tokens to predict next token (similar to N-gram).
- : $\mathbf{X}_I \in \mathbb{R}^{d\times Nm}$ are learnable parameters, where m is the dimension of the embedding.

Forward pass in pre-training on single sentence
(only use two recent tokens, i.e., $N = 2$)

```
1. Set $\mathbf{a}_0 = \mathbf{0}$, initial loss $L = 0$
2. For $t = 1, \ldots, T$
    ▶ $\mathbf{b}_t = \sigma\left(\mathbf{X}_I \begin{bmatrix}\mathbf{a}_{t-1} \\ \mathbf{a}_t\end{bmatrix}\right)$,    FFN
    ▶ $\mathbf{u}_t = \text{Channel Processing}(\mathbf{b}_t)$,    probability
    ▶ $L+ = \left(\sum_{i=1}^{|\mathcal{V}|} -\hat{\mathbf{u}}_t^{[i]} \log \mathbf{u}_t^{[i]}\right)$,    loss
```

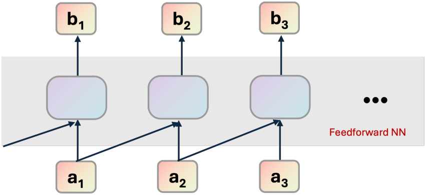


Figure: Feed forward neural network (FFN).

Remarks:
- The model dimension is dependent on N.
- Due to the underlying Markov model, it cannot capture long range dependencies!

b. Recurrent Neural Networks RNN as sequence mixers [44]: Training

Definition (RNN [22])

A recurrent neural network (RNN) is designed to handle sequential data in T steps by maintaining a hidden state $\mathbf{h}_t \in \mathbb{R}^d$ that captures temporal dependencies. At each time step t, we go through the following motions:

$$\mathbf{h}_t = g(\mathbf{a}_t, \mathbf{h}_{t-1}),$$

$$\mathbf{b}_t = f(\mathbf{h}_t),$$

where g and f are learnable functions (e.g., usually FFN layers).

Forward pass in pre-training on a single sentence

```
1. Set initial state $\mathbf{h}_0 = \mathbf{0}$, initial loss $L = 0$
2. For $t = 1, \dots, T$
    ▶ $\mathbf{h}_t = g(\mathbf{a}_t, \mathbf{h}_{t-1})$,
    ▶ $\mathbf{b}_t = f(\mathbf{h}_t)$,                                   $\mathrm{RNN}$
    ▶ $\mathbf{u}_t =$ Channel Processing$(\mathbf{b}_t)$,          probability
    ▶ $L+= \left(\sum_{i=1}^{|\mathcal{V}|} -\hat{\mathbf{u}}_t^{[i]} \log \mathbf{u}_t^{[i]}\right)$,          loss
```

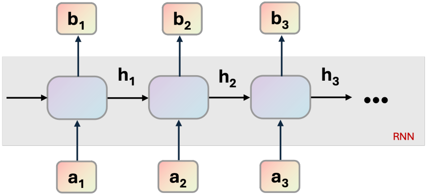


Figure: A recurrent neural network.

b. RNNs as sequence mixers: Inference

- ○ RNN architectures perform auto-regressive inference.

Forward pass in inference

```
1. Set $\mathbf{a}_1$ as the embedding of $\langle \mathrm{BOS} \rangle$, $t = 1$, initial state $\mathbf{h}_0 = \mathbf{0}$.
2. While True:
    ▸ $\mathbf{h}_t = g(\mathbf{a}_t, \mathbf{h}_{t-1})$,
    ▸ $\mathbf{b}_t = f(\mathbf{h}_t)$,
    ▸ $\mathbf{u}_t =$ Channel Processing$(\mathbf{b}_t)$,
    ▸ Set $\mathbf{a}_{t+1}$ as the embedding of the token corresponding to $\arg\max \mathbf{u}_t$.
    ▸ If $\mathbf{a}_{t+1}$ is the embedding of $\langle \mathrm{EOS} \rangle$: break
    ▸ $t{+} = 1$
3. Output: $[\mathbf{a}_1, \ldots, \mathbf{a}_{t+1}]$.
```

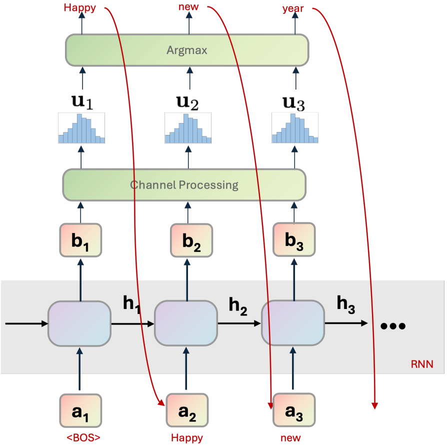


Figure: Auto-regressive inference of RNNs.

b. RNNs as sequence mixers: Take home messages

Remarks:

- RNN architectures *only partially* address long-range dependency problem
- Following problems persist
  - Vanishing or exploding gradients [49],
  - Short-term memory problem [28],
  - Mode collapse (i.e., generating repetitive outputs) [29],
  - Struggle with highly variable input sizes due to limited memory [5].
- Resource considerations:
  - Inference memory: $\mathcal{O}(d)$.
  - Training complexity: $\mathcal{O}(Td)$
  - Training time: no parallelization $\mathcal{O}(T)$ due to non-linearities.
- Many attempts to tackle these problems: LSTM [28], GRUs [11]...

More sophisticated RNNs: LSTM and GRU

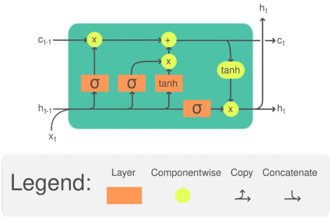


Figure: LSTM. https://en.wikipedia.org/wiki/Long_short-term_memory

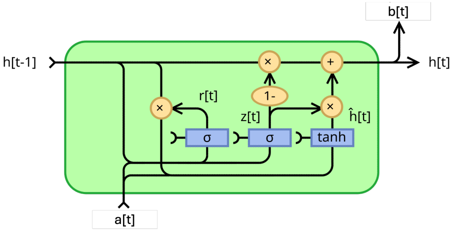


Figure: GRU. https://en.wikipedia.org/wiki/Gated_recurrent_unit

- **Long short-term memory (LSTM)** [28]
  - Aims to mitigate the vanishing gradient problem.
  - A unit is composed of a cell and three gates: an input gate, an output gate and a forget gate.

- **Gated recurrent units (GRUs)** [11]
  - Include mechanisms to gate certain features.
  - Lacks a context vector or output gate, resulting in fewer parameters than LSTM.

c. Self-attention [65] as sequence mixer

- Self-attention can address the short-comings of RNNs but at different training-inference costs trade offs.

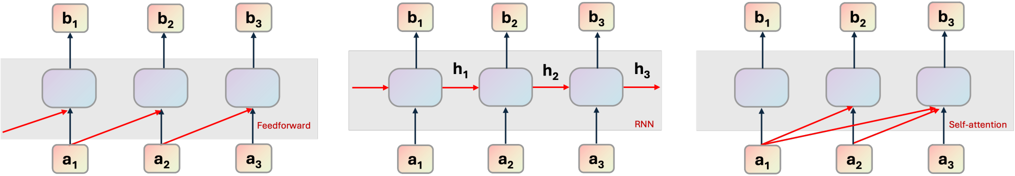
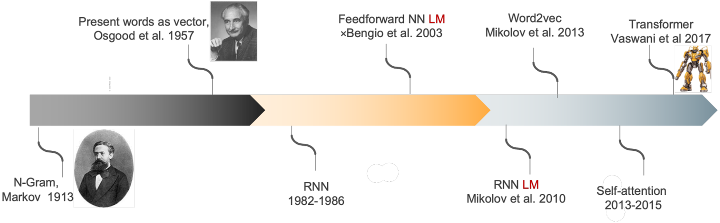


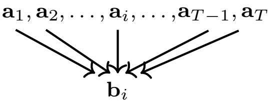
Figure: (**Left panel**) FFN. (**Middle panel**) RNN. (**Right panel**) Self-attention.



Present words as vector, Osgood et al. 1957
Feedforward NN LM ×Bengio et al. 2003
Word2vec Mikolov et al. 2013
Transformer Vaswani et al 2017
N-Gram, Markov 1913
RNN 1982-1986
RNN LM Mikolov et al. 2010
Self-attention 2013-2015

c. Self-attention as sequence mixer: Construction

- Core idea: compare a word of interest to other words based on their relevance.

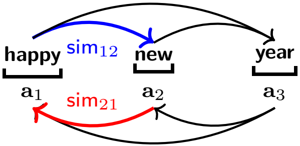

$\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_i, \ldots, \mathbf{a}_{T-1}, \mathbf{a}_T$
$\mathbf{b}_i$

Solution 1:

- Combine information based on their relevance/similarity.

$$\mathbf{b}_t = \sum_{j=1}^{t} \mathsf{sim}_{tj} \mathbf{a}_j$$

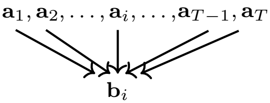
- How do we measure the relevance/similarity of two words?



$\mathsf{sim}_{12} \neq \mathsf{sim}_{21}$

c. Self-attention as sequence mixer: Construction

○ Core idea: compare a word of interest to other words based on their relevance.



$\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_i, \ldots, \mathbf{a}_{T-1}, \mathbf{a}_T$

$\mathbf{b}_i$

Solution 1: ○ Combine information based on their relevance/similarity.

$$\mathbf{b}_t = \sum_{j=1}^{t} \mathsf{sim}_{tj} \mathbf{a}_j$$

○ How do we measure the relevance/similarity of two words?

- We want

$$\mathsf{sim}(\mathbf{a}_k, \mathbf{a}_t) = \mathsf{sim}_{kt} = \begin{cases} 1 & \text{if } k = t, \\ \in (0,1) & \text{otherwise.} \end{cases},$$

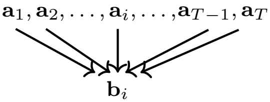
  and

$$\mathsf{sim}_{kt} \neq \mathsf{sim}_{tk}, \text{ for } k \neq t.$$

- One choice of similarity is as follows: $\mathsf{sim}_{kt} = (\mathbf{X}_1\mathbf{a}_k)^T(\mathbf{X}_2\mathbf{a}_t) \neq (\mathbf{X}_1\mathbf{a}_t)^T(\mathbf{X}_2\mathbf{a}_k) = \mathsf{sim}_{tk}$.

c. Self-attention as sequence mixer: Construction

○ Core idea: compare a word of interest to other words based on their relevance.



$\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_i, \dots, \mathbf{a}_{T-1}, \mathbf{a}_T$

$\mathbf{b}_i$

Solution 2: ○ Using parameters $\mathbf{X}_Q, \mathbf{X}_K, \mathbf{X}_V \in \mathbb{R}^{m\times d}$ for each word,

$$\boldsymbol{q}_i = \mathbf{X}_Q\mathbf{a}_i, \;\; \boldsymbol{k}_i = \mathbf{X}_K\mathbf{a}_i, \;\; \boldsymbol{v}_i = \mathbf{X}_V\mathbf{a}_i, \;\; \mathsf{sim}_{tj} = \langle \boldsymbol{q}_t, \boldsymbol{k}_j \rangle$$

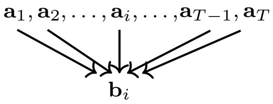
$$\mathbf{b}_t = \sum_{j=1}^{t} \mathsf{sim}_{tj}\boldsymbol{v}_j = \sum_{j=1}^{t} (\boldsymbol{q}_t^T \boldsymbol{k}_j)\boldsymbol{v}_j$$

○ Moreover, we want:

- $\mathsf{sim}_{kt} \geq 0$ (non-negativity),
- $\sum_{t=1}^{T} \mathsf{sim}_{kt} = 1$ (normalization).

c. Self-attention as sequence mixer: Construction

- Core idea: compare a word of interest to other words based on their relevance.



$\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_i, \ldots, \mathbf{a}_{T-1}, \mathbf{a}_T$

$\mathbf{b}_i$

Solution 3:

- Using parameters $\mathbf{X}_Q, \mathbf{X}_K, \mathbf{X}_V \in \mathbb{R}^{m\times d}$ for each word,

$$\boldsymbol{q}_i = \mathbf{X}_Q \mathbf{a}_i, \;\; \boldsymbol{k}_i = \mathbf{X}_K \mathbf{a}_i, \;\; \boldsymbol{v}_i = \mathbf{X}_V \mathbf{a}_i, \;\; \mathsf{sim}_{tj} = \langle \boldsymbol{q}_t, \boldsymbol{k}_j \rangle$$

$$\mathbf{b}_i = \sum_{j=1}^{T} \mathsf{Softmax}([\mathsf{sim}_{i1}, \mathsf{sim}_{i2}, \ldots, \mathsf{sim}_{iT}])_j \mathbf{a}_j$$

$$= \sum_{j=1}^{T} \frac{\exp(\mathsf{sim}_{ij})}{\sum_{\ell=1}^{T} \exp(\mathsf{sim}_{i\ell})} \mathbf{a}_j$$

c. Self-attention as sequence mixer

Definition (Query, Key, Value [65])

Another way to capture how words contribute to each other:

- *Query*: current word measures the relevance with others.
- *Key*: the relevance is measured by other words.
- *Value*: value generalizes the final output.

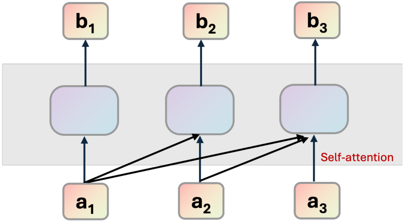


Figure: Self-attention layer.

○ Each word calculates its corresponding query, key, and value with learned parameters $\mathbf{X}_Q, \mathbf{X}_K, \mathbf{X}_V \in \mathbb{R}^{m \times d}$

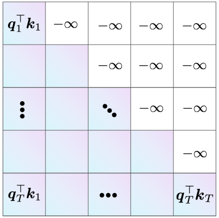
$$\mathbf{Q} \in \mathbb{R}^{T \times d} := \left\{ \mathbf{q}_t = \mathbf{X}_Q \mathbf{a}_t \right\}$$
$$\mathbf{K} \in \mathbb{R}^{T \times d} := \{ \mathbf{k}_t = \mathbf{X}_K \mathbf{a}_t \}$$
$$\mathbf{V} \in \mathbb{R}^{T \times d} := \{ \mathbf{v}_t = \mathbf{X}_V \mathbf{a}_t \}$$

Causal language modeling (CLM)

Causal attention [55]

$$\mathbf{B} = \text{Softmax}((\mathbf{Q}\mathbf{K}^T) \odot \mathbf{M}^C)\mathbf{V} \text{ where } \mathbf{M}^C_{ij} = \begin{cases} 1, & i \geq j, \\ -\infty, & i < j \end{cases}$$

is a lower triangular matrix and $\odot$ is element-wise multiplication.



$\boldsymbol{q}_1^\top \boldsymbol{k}_1$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
		$-\infty$	$-\infty$	$-\infty$
$\vdots$		$\ddots$	$-\infty$	$-\infty$
				$-\infty$
$\boldsymbol{q}_T^\top \boldsymbol{k}_1$		$\cdots$		$\boldsymbol{q}_T^\top \boldsymbol{k}_T$

Figure: Causal attention

Remarks:

- ○ Since self-attention is order invariant, it requires positional embeddings.
- ○ It is necessary to mask scores to prevent “cheating.”
  - ▶ The current word has only seen previous word.
  - ▶ The subsequent word is unknown.
  - ▶ The element $-\infty$ after softmax becomes 0.
- ○ Attention with masking score is usually called “Masked attention” or “Causal attention.”
- ○ This construction enables parallelization whereby improving upon RNNs.

c. Self-attention as sequence mixer: Training

Remarks:

- $\mathbf{A} = [\mathbf{a}_1, ...\mathbf{a}_T]^\top \in \mathbb{R}^{T \times d}$: collections of embeddings of all tokens.
- Learnable parameters: $\mathbf{X}_Q, \mathbf{X}_K, \mathbf{X}_V \in \mathbb{R}^{m \times d}$.

Forward pass in training on a single sentence

1. Set initial loss $L = 0$.

2. $\boldsymbol{Q} = \mathbf{A}\mathbf{X}_Q^\top, \boldsymbol{K} = \mathbf{A}\mathbf{X}_K^\top, \boldsymbol{V} = \mathbf{A}\mathbf{X}_V^\top$, query, key, value.

3. $\mathbf{B} = \text{Softmax}((\mathbf{Q}\mathbf{K}^T) \odot \mathbf{M}^C)\mathbf{V}$, self-attention output

4. $\mathbf{U} := [\mathbf{u}_1, ..., \mathbf{u}_T]^\top = \text{Channel Processing}(\mathbf{B})$, probability

5. $L = L + \left(\sum_{t=1}^{T} \sum_{i=1}^{|\mathcal{V}|} -\hat{\mathbf{u}}_t^{[i]} \log \mathbf{u}_t^{[i]}\right)$, loss

c. Self-attention as sequence mixer: Inference

Forward pass in inference

```
1. Set $\mathbf{a}_1$ as the embedding of $\langle \mathrm{BOS} \rangle$, $t = 1$.
2. While True:
    ▶ $\boldsymbol{q}_t = \mathbf{X}_Q \mathbf{a}_t, \boldsymbol{k}_t = \mathbf{X}_K \mathbf{a}_t, \boldsymbol{v}_t = \mathbf{X}_V \mathbf{a}_t,$        query, key, value
    ▶ $\boldsymbol{s} = [\langle \boldsymbol{q}_t, \boldsymbol{k}_1 \rangle, \ldots, \langle \boldsymbol{q}_t, \boldsymbol{k}_t \rangle]^\top$, calculate score
    ▶ $\mathbf{b}_t = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t]\text{Softmax}(\boldsymbol{s})$
    ▶ $\mathbf{u}_t = \text{Channel Processing}(\mathbf{b}_t)$
    ▶ Set $\mathbf{a}_{t+1}$ as the embedding of the token corresponding to $\arg\max \mathbf{u}_t$.
    ▶ If $\mathbf{a}_{t+1}$ is the embedding of $\langle \mathrm{BOS} \rangle$: break
    ▶ $t += 1$
3. Output: $[\mathbf{a}_1, s, \mathbf{a}_{t+1}]$.
```

Remark: ○ Still non-parallelizable, still auto-regression, the same as RNN and FFN LM.

c. Self-attention as sequence mixer: A key resource trade-off

Remark: ○ Computation and memory of attention scales quadratically $\mathcal{O}(T^2)$ with sequence length T.

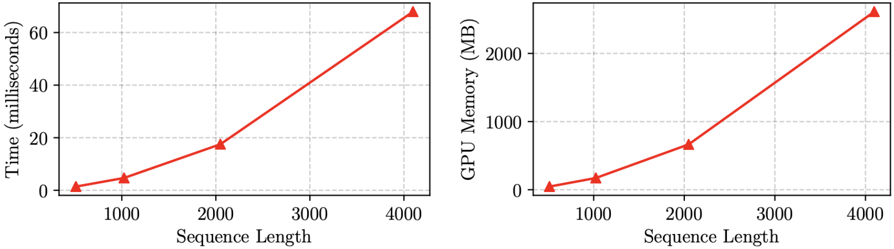


Figure: Scaling of computation and memory of a self-attention with sequence length
From https://angeloskath.github.io/data/linear_transformers_slides.pdf

c. Self-attention as sequence mixer: KV cache

Definition (KV cache [53])

KV Cache (Key-Value Cache) stores computed keys ($\boldsymbol{K}$) and values ($\boldsymbol{V}$) from previous time steps to avoid recomputation in self-attention during autoregressive inference.

- How Does KV Cache Work?
  1. Compute and store $\boldsymbol{k}_1, \boldsymbol{v}_1$.
  2. Retrieve $\boldsymbol{k}_1, \boldsymbol{v}_1$, compute $\boldsymbol{k}_2, \boldsymbol{v}_2$, and append.
  3. Retrieve all cached $\boldsymbol{K}, \boldsymbol{V}$ and compute only for the new token.

Remarks:

- Standard self-attention recomputes all $\boldsymbol{K}$ and $\boldsymbol{V}$ at every step: $\mathcal{O}(T^2 d)$.
- KV Cache stores values and retrieves them, reducing complexity to $\mathcal{O}(Td)$.
- Faster inference.
- Lower memory overhead.
- Enables efficient scaling for LLMs.
- There is a ton of literature in improving the efficiency of KV caches (e.g., with compression, etc.).

c. Self-attention as sequence mixer: KV cache basics

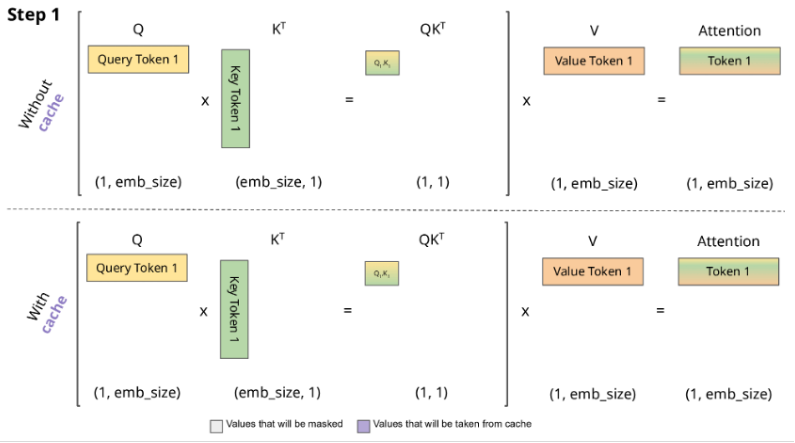


Figure: From `https://medium.com/@joaolages/kv-caching-explained-276520203249`

c. Self-attention as sequence mixer: KV cache basics

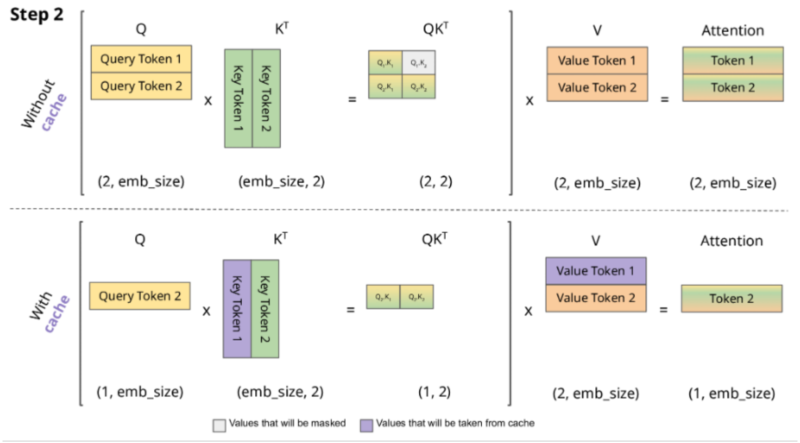


Figure: From `https://medium.com/@joaolages/kv-caching-explained-276520203249`

c. Self-attention as sequence mixer: KV cache basics

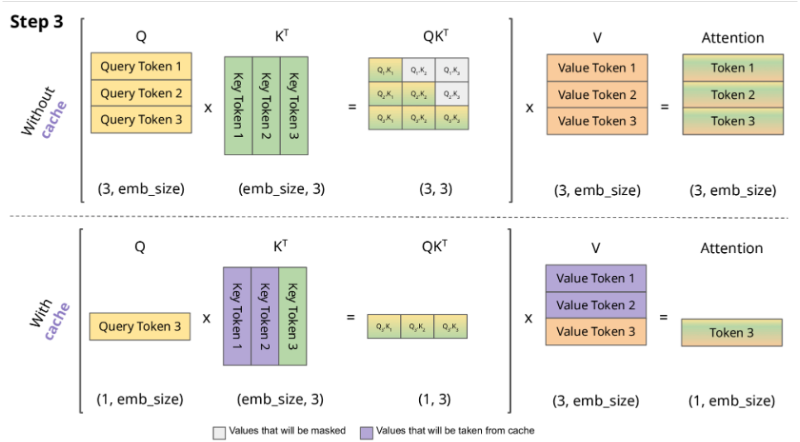


Figure: From `https://medium.com/@joaolages/kv-caching-explained-276520203249`

c. Self-attention as sequence mixer: KV cache basics

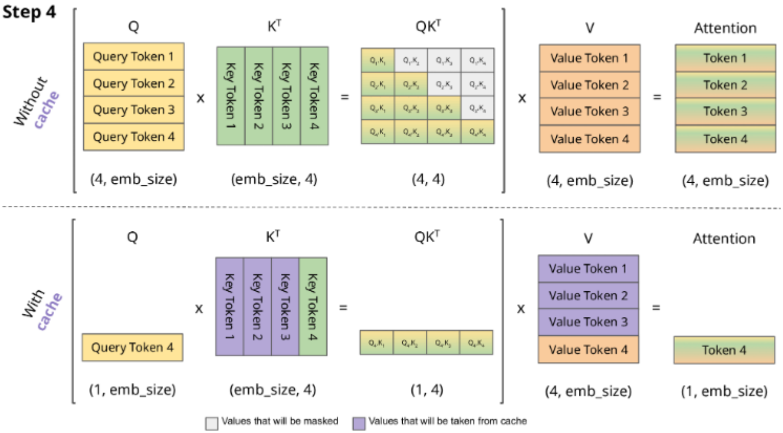


Figure: From `https://medium.com/@joaolages/kv-caching-explained-276520203249`

Another key idea in the same vein: Multihead attention

Multihead Attention [65]

Instead of having one attention, it is possible to have h attention heads in parallel such that

$$\text{MultiHead}(\mathbf{A}) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})$$
$$\text{where head}_\text{i} = \text{Attention}(\mathbf{A}\mathbf{X}_i^Q, \mathbf{A}\mathbf{X}_i^K, \mathbf{A}\mathbf{X}_i^V),$$

where the projections are parameter matrices $\mathbf{X}_i^Q \in \mathbb{R}^{m \times d_h}$, $\mathbf{X}_i^K \in \mathbb{R}^{m \times d_h}$, $\mathbf{X}_i^V \in \mathbb{R}^{m \times d_h}$ and $d_h = d/h$.

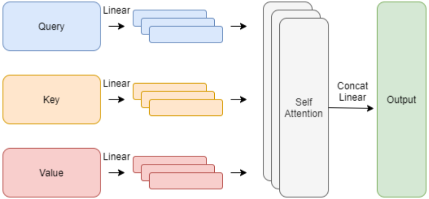


Figure: Multi-head self-attention mechanism, where the embedding dimension is split across multiple heads, each capturing different contextual features before aggregation with concatenation [46].

Remarks: ○ Dividing hidden dimension to heads allows parallelization while keeping the computational cost similar to single-head attention [65]. .

○ Allows each head to focus on different aspects of the input, capturing a wide range of features and relationships (e.g., subject-verb agreement, syntax, semantics) [31].

○ Enhances the model's ability to capture diverse dependencies.

○ Another similar idea to reduce cost is called Grouped-Query Attention (GQA) [2] (see supplementary material).

Efficient self-attention: FlashAttention [14]

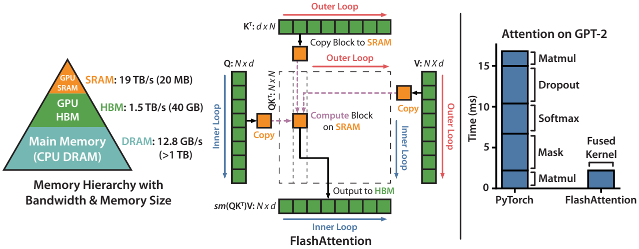


Figure: Visualization of inner and outer loops of FlashAttention with the locations in memory hierarchy [14].

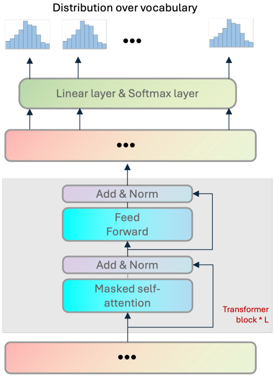
- Core idea: Reduce memory usage and improve speed using tiling and recomputation.
  - Fused kernel for matrix multiplication, softmax and masking.
  - Processes in small blocks instead of full sequence.
  - Uses fast on-chip memory (SRAM) to minimize memory traffic.
  - Up to 2–$4\times$ **faster** than standard attention with less memory footprint.
  - Has $\mathcal{O}(TCd)$ complexity where C is the block size.

Transformer as LM

- A Transformer block= [self-attention layer + layer normalization + feedforward layer + layer normalization].
- We stack $\mathcal{L}$ Transformer blocks to form an LM, e.g., $\mathcal{L} = 12$ in [56].

Forward pass in pre-training on single sentence

```
1. Set initial loss $L = 0$, denote by $\mathbf{B}_0 = \mathbf{A}$ the input to the first block.
2. For $l = 1, \dots, \mathcal{L}$
    ▶ $\boldsymbol{Q}_l = \mathbf{B}_{l-1}\mathbf{X}_{Q,l}^\top, \boldsymbol{K}_l = \mathbf{B}_{l-1}\mathbf{X}_{K,l}^\top, \boldsymbol{V}_l = \mathbf{B}_{l-1}\mathbf{X}_{V,l}^\top,$    query, key, value.
    ▶ $\boldsymbol{S}_l = \text{Mask}(\boldsymbol{Q}_l\boldsymbol{K}_l^\top)$, calculate score and mask score.
    ▶ $\mathbf{B}_l = \text{Row-wise-Softmax}(\boldsymbol{S}_l)\boldsymbol{V}_l$
    ▶ $\mathbf{B}_l + = \mathbf{B}_{l-1},$    "add" in the figure, motivated by ResNet [27]
    ▶ $\mathbf{B}_l = \text{Layernorm}(\mathbf{B}_l)$
    ▶ $\mathbf{B}_{\text{shortcut}} = \mathbf{B}_l$
    ▶ $\mathbf{B}_l = \sigma(\mathbf{X}_{F,l}\mathbf{B}_l),$    feedforward
    ▶ $\mathbf{B}_l + = \mathbf{B}_{\text{shortcut}},$    "add"
    ▶ $\mathbf{B}_l = \text{Layernorm}(\mathbf{B}_l)$    output of each Transformer block
3. $\mathbf{U} := [\mathbf{u}_1, \dots, \mathbf{u}_T]^\top = \text{Row-wise-Softmax}(\mathbf{B}_L\mathbf{X}_O^\top),$    probability
4. $L + = \left(\sum_{t=1}^{T}\sum_{i=1}^{|\mathcal{V}|} -\hat{\mathbf{u}}_t^{[i]} \log \mathbf{u}_t^{[i]}\right),$    loss
```



Remarks:
- Original Transformer is proposed with encoder and decoder for neural machine translation [65].
- The Transformer decoder is sufficient as an LM.

Batch and layer normalization [4]

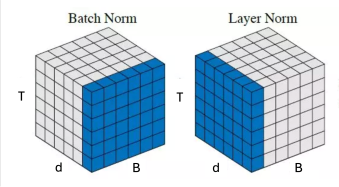


Figure: Batch and layer normalization [10].

- ○ For an input $\mathbf{A} \in \mathbb{R}^{B \times T \times d}$, we use the following notation:
  - ▶ B is batch size
  - ▶ T is sequence length
  - ▶ d is embedding dimension
- ○ The normalization layers enable the following
  - ▶ Forward view: distribution stability [4].
  - ▶ Backward view: normalization for the backward gradient [66].

Batch normalization

$$\boldsymbol{\mu}_B = \frac{1}{B} \sum_{i=1}^{B} \mathbf{A}_i, \quad \boldsymbol{\sigma}_B^2 = \frac{1}{B} \sum_{i=1}^{B} (\mathbf{A}_i - \boldsymbol{\mu}_B)^2$$

$$\hat{\mathbf{A}}_i = \frac{\mathbf{A}_i - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \epsilon}}$$

Layer normalization

$$\boldsymbol{\mu}_d = \frac{1}{d} \sum_{j=1}^{d} \mathbf{A}_{::j} \quad \boldsymbol{\sigma}_d^2 = \frac{1}{d} \sum_{j=1}^{d} (\mathbf{A}_{::j} - \boldsymbol{\mu}_d)^2$$

$$\hat{\mathbf{A}}_{::j} = \frac{\mathbf{A}_{::j} - \boldsymbol{\mu}_d}{\sqrt{\boldsymbol{\sigma}_d^2 + \epsilon}}$$

Remark: ○ ϵ is a small value to ensure stability.

d. Linear-attention [38] as sequence mixer

Layer Type	Inference memory	Training time	Computational complexity
Recurrent	$\boldsymbol{\mathcal{O}(d)}$	$\mathcal{O}(T)$	$\boldsymbol{\mathcal{O}(Td)}$
Self-Attention	$\mathcal{O}(T^2)$	$\boldsymbol{\mathcal{O}(1)}$	$\mathcal{O}(T^2 d)$

Question: ◦ Can we have the best of both worlds?

d. Linear-attention [38] as sequence mixer

Observation: ○ Softmax is the bottleneck for both training and infering self-attention.

Solution: ○ Aproximated softmax with linear dot product of feature maps [8, 57, 59].

Full softmax attention

$$\text{Softmax}((\mathbf{Q}\mathbf{K}^T))\mathbf{V} = \frac{\sum_{j=1}^{N} \text{sim}(\mathbf{q}_i, \mathbf{k}_j)\mathbf{v}_j}{\sum_{j=1}^{N} \text{sim}(\mathbf{q}_i, \mathbf{k}_j)}.$$

Linearized full attention

$$\left(\phi(\mathbf{Q})\phi(\mathbf{K})^T\right)\mathbf{V} = \phi(\mathbf{Q})\left(\phi(\mathbf{K})^T\mathbf{V}\right) = \frac{\phi(\mathbf{q}_i)^T \sum_{j=1}^{N} \phi(\mathbf{k}_j)\mathbf{v}_j^T}{\phi(\mathbf{q}_i)^T \sum_{j=1}^{N} \phi(\mathbf{k}_j)}.$$

d. Linear-attention as sequence mixer: CLM

Causal softmax attention

$$\mathsf{Softmax}((\mathbf{Q}\mathbf{K}^T) \odot \mathbf{M}^C)\mathbf{V} = \frac{\sum_{j=1}^{N} \mathsf{sim}(\mathbf{q}_i, \mathbf{k}_j)\mathbf{v}_j}{\sum_{j=1}^{i} \mathsf{sim}(\mathbf{q}_i, \mathbf{k}_j)}.$$

Linearized causal attention

$$\frac{\phi(\mathbf{q}_i)^T \sum_{j=1}^{N} \phi(\mathbf{k}_j)\mathbf{v}_j^T}{\phi(\mathbf{q}_i)^T \sum_{j=1}^{i} \phi(\mathbf{k}_j)} = \frac{\phi(\mathbf{q}_i)^T \boldsymbol{S}_i}{\phi(\mathbf{q}_i)^T \mathbf{z}_i}$$

where

$$\boldsymbol{S}_i = \sum_{j=1}^{i} \phi(\mathbf{k}_j)\mathbf{v}_j^T,$$

$$\mathbf{z}_i = \sum_{j=1}^{i} \phi(\mathbf{k}_j).$$

Remarks:

- ○ One common choice of nonlinearity is $\phi(a) = \mathsf{elu}(a) + 1$.
- ○ Note that the state $\S \in \mathbb{R}^{d\times d}$.

d. Linear-attention as sequence mixer: Training

Forward pass in training on a single sentence

1. Set initial loss $L = 0$.
2. $\boldsymbol{Q} = \phi(\mathbf{A}\mathbf{X}_Q^\top), \boldsymbol{K} = \phi(\mathbf{A}\mathbf{X}_K^\top), \boldsymbol{V} = \mathbf{A}\mathbf{X}_V^\top,$ query, key, value.
3. $\mathbf{B} = ((\mathbf{Q}\mathbf{K}^T) \odot \mathbf{M}^C)\mathbf{V},$ linear attention output
4. $\mathbf{U} := [\mathbf{u}_1, ..., \mathbf{u}_T]^\top =$ Channel Processing$(\mathbf{B})$, probability
5. $L = L + \left(\sum_{t=1}^{T} \sum_{i=1}^{|\mathcal{V}|} -\hat{\mathbf{u}}_t^{[i]} \log \mathbf{u}_t^{[i]}\right),$ loss

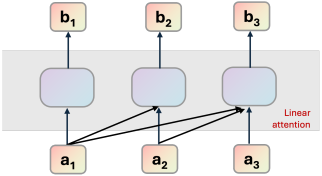


Figure: Linear attention layer.

Remarks:

- ∘ It can be trained like a self-attention.
- ∘ $\mathbf{A} = [\mathbf{a}_1, ...\mathbf{a}_T]^\top \in \mathbb{R}^{T \times d}$: collections of embeddings of all tokens.
- ∘ Learnable parameters: $\mathbf{X}_Q, \mathbf{X}_K, \mathbf{X}_V \in \mathbb{R}^{m \times d}$.
- ∘ With parallelized training, the time is $\mathcal{O}(1)$.
- ∘ Due to lack of softmax, $\mathbf{M}_{ij}^C = \begin{cases} 1, & i \geq j, \\ 0, & i < j \end{cases}$

d. Linear-attention as sequence mixer: Inference

Forward pass in inference

```
1. Set $\mathbf{a}_1$ as the embedding of $\langle\mathrm{BOS}\rangle$, $t = 1$, initial state $\boldsymbol{S}_0 = \mathbf{0}, \boldsymbol{z}_0 = \mathbf{0}$.
2. While True:
    ▶ $\boldsymbol{q}_t = \phi(\mathbf{X}_Q \mathbf{a}_t), \boldsymbol{k}_t = \phi(\mathbf{X}_K \mathbf{a}_t), \boldsymbol{v}_t = \mathbf{X}_V \mathbf{a}_t$,
    ▶ $\boldsymbol{S}_t + = \mathbf{k}_t \mathbf{v}_t^T, \mathbf{z}_t + = \mathbf{k}_t$,
    ▶ $\mathbf{b}_t = \frac{\phi(\mathbf{q}_t)^T \boldsymbol{S}_t}{\phi(\mathbf{q}_t)^T \mathbf{z}_t}$,
    ▶ $\mathbf{u}_t =$ Channel Processing$(\mathbf{b}_t)$,
    ▶ Set $\mathbf{a}_{t+1}$ as the embedding of the token corresponding to $\arg\max \mathbf{u}_t$.
    ▶ If $\mathbf{a}_{t+1}$ is the embedding of $\langle\mathrm{EOS}\rangle$: break
    ▶ $t + = 1$
3. Output: $[\mathbf{a}_1, s, \mathbf{a}_{t+1}]$.
```

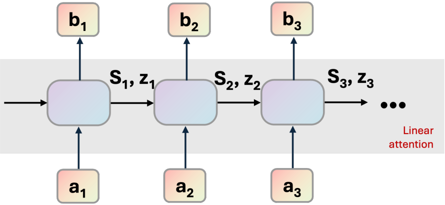


Figure: Auto-regressive inference of linear attention.

Remarks:

- ○ Linear attention can perform auto-regressive inference.
- ○ $\boldsymbol{S}_i$ and $\boldsymbol{z}_i$ can be computed from $\boldsymbol{S}_{i-1}$ and $\boldsymbol{z}_{i-1}$ in constant time.
- ○ The memory for inference is $\mathcal{O}(d^2)$.

d. Linear-attention as sequence mixer

Layer Type	Inference memory	Training time	Computational complexity
Recurrent	$\boldsymbol{\mathcal{O}(d)}$	$\mathcal{O}(T)$	$\boldsymbol{\mathcal{O}(Td)}$
Self-Attention	$\mathcal{O}(T^2)$	$\boldsymbol{\mathcal{O}(1)}$	$\mathcal{O}(T^2 d)$
KV Cache	$\mathcal{O}(Td)$	$\boldsymbol{\mathcal{O}(1)}$	$\mathcal{O}(Td + d^2)$
Linear-Attention	$\mathcal{O}(d^2)$	$\boldsymbol{\mathcal{O}(1)}$	$\mathcal{O}(Td^2)$

Remarks:

- Note that d is the corresponding embedding dimension for the network.
- Still requires positional embeddings.
- Significantly underperforms softmax based attention.
- Due to the cumulative sum, the state value could explode.

Solution:

- Embed the positional information in the state updates.

d. Linear-attention as sequence mixer: To position embed or not?

- Embeding the positional information in the state updates provides:
  - Implicit positional information,
  - Numerical stability,
  - Better performance.
- We can implicitly encode time via a variable λ as follows:

$$\boldsymbol{S}_t = \lambda_t \boldsymbol{S}_{t-1} + \mathbf{k}_t \mathbf{v}_t^T \quad \boldsymbol{z}_t = \lambda_t \boldsymbol{z}_{t-1} + \mathbf{k}_t.$$

  - For ease of notation, we use $\boldsymbol{q}_j = \phi(\mathbf{X}_Q \mathbf{a}_j), \boldsymbol{k}_j = \phi(\mathbf{X}_K \mathbf{a}_j), \boldsymbol{v}_j = \mathbf{X}_V \mathbf{a}_j$ in the sequel.
- One choice of λ for recurrent computation is an exponential decay factor $\lambda_t = \gamma$ where $0 < \gamma < 1$ [62]

$$\begin{aligned}
\boldsymbol{S}_0 &= 0 \\
\boldsymbol{S}_1 &= \mathbf{k}_1 \mathbf{v}_1^T, \\
\boldsymbol{S}_2 &= \gamma \boldsymbol{S}_1 + \mathbf{k}_2 \mathbf{v}_2^T = \gamma(\mathbf{k}_1 \mathbf{v}_1^T) + \mathbf{k}_2 \mathbf{v}_2^T, \\
\boldsymbol{S}_3 &= \gamma \boldsymbol{S}_2 + \mathbf{k}_3 \mathbf{v}_3^T = \gamma^2(\mathbf{k}_1 \mathbf{v}_1^T) + \gamma(\mathbf{k}_2 \mathbf{v}_2^T) + \mathbf{k}_3 \mathbf{v}_3^T \ldots
\end{aligned}$$

d. Linear-attention as sequence mixer: Train like a self-attention, infer like an RNN

1	0	0	0	0
γ	1	0	0	0
$\vdots$		$\ddots$	0	0
γ^{T-2}			1	0
γ^{T-1}	γ^{T-2}	$\cdots$	γ	1

Figure: Decay mask $\mathbf{M}^D$

Training

$\mathbf{B} = ((\mathbf{Q}\mathbf{K}^T) \odot \mathbf{M}^D)\mathbf{V}$, where $\mathbf{M}^D_{ij} =$

$$\begin{cases} \gamma^{i-j}, & i > j, \\ 1, & i = j, \\ 0, & \text{otherwise.} \end{cases}$$

Inference

$$S_t = \sum_{j=1}^{t} \gamma^{t-j} \mathbf{k}_j \mathbf{v}_j^T \qquad S_t = \gamma S_{t-1} + \mathbf{k}_t \mathbf{v}_t^T$$

$$\mathbf{z}_t = \sum_{j=1}^{t} \gamma^{t-j} \mathbf{k}_j \qquad \mathbf{z}_t = \gamma \mathbf{z}_{t-1} + \mathbf{k}_t$$

$$s_{\text{out},t} = \frac{\mathbf{q}_t^T S_t}{\mathbf{q}_t^T \mathbf{z}_t}$$

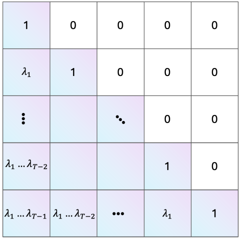
Observation:

- ○ Competitive performance with softmax based attention [62].
- ○ It lacks input-dependency (i.e., selectivity).

d. Linear-attention as sequence mixer: Selectivity

- How can we also incorporate input dependency along with position information?

Solution:
- λ_{ij} is data dependent decay term with $0 < \lambda_{ij} < 1$.
- Many examples in the literature with competitive/better performances [40, 6, 52, 71]...



1	0	0	0	0
λ_1	1	0	0	0
$\vdots$		$\ddots$	0	0
$\lambda_1 \dots \lambda_{T-2}$			1	0
$\lambda_1 \dots \lambda_{T-1}$	$\lambda_1 \dots \lambda_{T-2}$	$\cdots$	λ_1	1

Figure: Selective mask $\mathbf{M}^S$

Training

$$\mathbf{B} = ((\mathbf{QK}^T) \odot \mathbf{M}^S)\mathbf{V}, \text{where}$$

$$\mathbf{M}^S_{ij} = \begin{cases} \prod_{m=i+1}^{j} \lambda_m, & i > j, \\ 1, & i = j, \\ 0, & \text{otherwise.} \end{cases}$$

Inference

$$\boldsymbol{S}_t = \lambda_t \boldsymbol{S}_{t-1} + \mathbf{k}_t \mathbf{v}_t^T,$$
$$\boldsymbol{z}_t = \lambda_t \boldsymbol{z}_{t-1} + \mathbf{k}_t,$$
$$\mathbf{b}_t = \frac{\mathbf{q}_t^T \boldsymbol{S}_t}{\mathbf{q}_t^T \mathbf{z}_t}.$$

e. SSMs as sequence mixers

Definition (Continuous state space representation)

S4 (structured state space sequence) models [25] in continuous domain are defined using 4 parameters $(\Delta, \mathbf{X}^A, \mathbf{X}^B, \mathbf{X}^C)$ such as

$$\begin{aligned} \mathbf{S}'_{(t)} &= \mathbf{X}^A_{(t)} \mathbf{S}_{(t)} + \mathbf{X}^B_{(t)} \mathbf{a}_{(t)}, \\ \mathbf{b}_{(t)} &= \mathbf{X}^C_{(t)} \mathbf{S}_{(t)}. \end{aligned}$$

Definition (Discrete state space representation)

Using zero-order hold (ZOH) [36] approximation (see supplementary material), it is possible to discretize them as

$$\begin{aligned} \mathbf{S}_i &= \overline{\mathbf{X}^A}_i \mathbf{S}_{i-1} + \overline{\mathbf{X}^B}_i \mathbf{a}_i, \\ \mathbf{b}_i &= \mathbf{X}^C_i \mathbf{S}_i, \end{aligned}$$

where $\overline{\mathbf{X}^A} = \exp(\Delta \mathbf{X}^A), \overline{\mathbf{X}^B} = (\Delta \mathbf{X}^A)^{-1}(\exp(\Delta \mathbf{X}^A) - I)\Delta \mathbf{X}^B$.

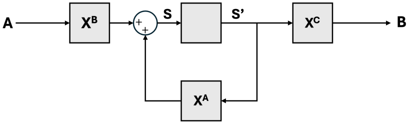


Figure: State space model.

e. SSMs as sequence mixer: Training

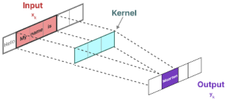


Figure: SSM convolutional kernel usage for efficient training.
From `https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state`

- ○ For efficient training, sequential convolutional operations are used.

Training
$\overline{\boldsymbol{K}} = (\mathbf{X}^C\overline{\mathbf{X}^B}, \mathbf{X}^C\overline{\mathbf{X}^A\mathbf{X}^B}, \dots, \mathbf{X}^C\overline{\mathbf{X}^A}^k\overline{\mathbf{X}^B}, \dots),$
$\mathbf{B} = \mathbf{A} * \overline{\boldsymbol{K}}$

Remarks:

- ○ S4 had all parameters input independent.
- ○ Mamba1 [24] introduced selective Δ_t.
- ○ Selectivity enabled higher performance and ability to apply to CLM task.

e. SSMs as sequence mixer: Inference

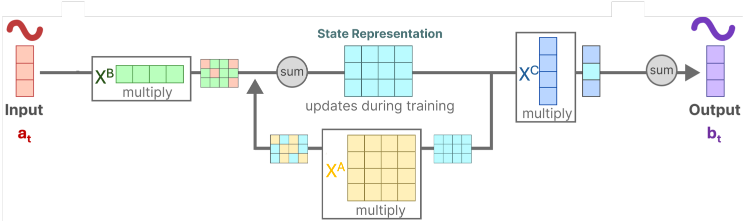


Figure: SSM model. From https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

- Inference can be performed similar to the recurrent sequence modeling.

Inference

$$\mathbf{S}_t = \overline{\mathbf{X}^A}_t \mathbf{S}_{t-1} + \overline{\mathbf{X}^B}_t \mathbf{a}_t,$$

$$\mathbf{b}_i = \mathbf{X}_t^C \mathbf{S}_t$$

Remarks:

- Many sota SSMs include additional design elements such as
  - Hippo [26] initialization, gating mechanism, convolutional layers,
  - multi head layers and state expansion [24, 15].

e. SSMs as sequence mixer: Towards linear attention and state-space duality

Linear-attention inference

$$\boldsymbol{S}_t = \lambda_t \boldsymbol{S}_{t-1} + \mathbf{k}_t \mathbf{v}_t^T, \quad \boldsymbol{z}_t = \lambda_t \boldsymbol{z}_{t-1} + \mathbf{k}_t,$$

$$\mathbf{b}_t = \frac{\mathbf{q}_t^T \boldsymbol{S}_t}{\mathbf{q}_t^T \mathbf{z}_t}.$$

$$\longleftrightarrow$$

SSM inference

$$\mathbf{S}_t = \overline{\mathbf{X}^A}_t \mathbf{S}_{t-1} + \overline{\mathbf{X}^B}_t \mathbf{a}_t,$$

$$\mathbf{b}_i = \mathbf{X}_t^C \mathbf{S}_t$$

- Using state-space duality [15], it is possible to rename parameters as $(\mathbf{X}^C, \mathbf{X}^B, \mathbf{A}) \to (\mathbf{Q}, \mathbf{K}, \mathbf{V})$.
- Following the linear attention formulation, we can write the inference of Mamba1 [24] as follows

$$\boldsymbol{S}_t = \boldsymbol{G}_t \odot \boldsymbol{S}_{t-1} + \mathbf{k}_t \mathbf{v}_t^T,$$
$$\mathbf{b}_t = \mathbf{q}_t^T \boldsymbol{S}_t,$$

with $\boldsymbol{G}_t = \exp(-(\Delta_t \mathbf{1}^T) \odot \exp(\mathbf{X}^A))$.

Remarks:

- $\mathbf{X}^A \in \mathbb{R}^{d \times d}$ is data independent.
- $\Delta_t \in \mathbb{R}^d$ is data dependent [68].
- Mamba2 uses a parameter instead of the selective diagonal matrix $\boldsymbol{G}_t$.
- It is more efficient, scalable and closer to linear attention.

Duality of linear attention and SSMs

Table: Overview of recent linear recurrent models. Matrix state values $\mathbf{S}_t \in \mathbb{R}^{d\times n}$,$\mathbf{S}_t^{\boldsymbol{k}}$, $\mathbf{S}_t^{\boldsymbol{v}}$, $\odot$ is the Hadamard product, additional linear RNN with hidden state vector $\boldsymbol{z}_t$, which used to normalized the query vector $\boldsymbol{q}_t$. Variables with the subscript t are potentially non-linear functions of the current input $\mathbf{a}_t$. Taken from [71].

Model	Recurrence	Memory read-out
Linear Attention [38, 37]	$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^\intercal$	$\mathbf{b}_t = \mathbf{S}_t \boldsymbol{q}_t$
+ Kernel	$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t \phi(\boldsymbol{k}_t)^\intercal$	$\mathbf{b}_t = \mathbf{S}_t \phi(\boldsymbol{q}_t)$
+ Normalization	$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t \phi(\boldsymbol{k}_t)^\intercal, \ \ \boldsymbol{z}_t = \boldsymbol{z}_{t-1} + \phi(\boldsymbol{k}_t)$	$\mathbf{b}_t = \mathbf{S}_t \phi(\boldsymbol{q}_t) / (\boldsymbol{z}_t^\intercal \phi(\boldsymbol{q}_t))$
DeltaNet [71]	$\mathbf{S}_t = \mathbf{S}_{t-1}(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\intercal) + \beta_t \boldsymbol{v}_t \boldsymbol{k}_t^\intercal$	$\mathbf{b}_t = \mathbf{S}_t \boldsymbol{q}_t$
Gated RFA [52]	$\mathbf{S}_t = g_t \mathbf{S}_{t-1} + (1 - g_t) \boldsymbol{v}_t \boldsymbol{k}_t^\intercal, \ \ \boldsymbol{z}_t = g_t \boldsymbol{z}_{t-1} + (1 - g_t) \boldsymbol{k}_t$	$\mathbf{b}_t = \mathbf{S}_t \boldsymbol{q}_t / (\boldsymbol{z}_t^\intercal \boldsymbol{q}_t)$
S4 [25, 60]	$\mathbf{S}_t = \mathbf{S}_{t-1} \odot \exp(-(\boldsymbol{\alpha} \mathbf{1}^\intercal) \odot \exp(\mathbf{X}^A)) + \mathbf{X}^B \odot (\boldsymbol{v}_t \mathbf{1}^\intercal)$	$\mathbf{b}_t = (\mathbf{S}_t \odot \mathbf{X}^C) \mathbf{1} + \boldsymbol{d} \odot \boldsymbol{v}_t$
ABC [51]	$\mathbf{S}_t^{\boldsymbol{k}} = \mathbf{S}_{t-1}^{\boldsymbol{k}} + \boldsymbol{k}_t \boldsymbol{\phi}_t^\intercal, \ \ \mathbf{S}_t^{\boldsymbol{v}} = \mathbf{S}_{t-1}^{\boldsymbol{v}} + \boldsymbol{v}_t \boldsymbol{\phi}_{\boldsymbol{t}}^\intercal$	$\mathbf{b}_t = \mathbf{S}_t^{\boldsymbol{v}} \operatorname{softmax}\left(\mathbf{S}_t^{\boldsymbol{k}} \boldsymbol{q}_t\right)$
DFW [41]	$\mathbf{S}_t = \mathbf{S}_{t-1} \odot (\boldsymbol{\beta}_t \boldsymbol{\alpha}_t^\intercal) + \boldsymbol{v}_t \boldsymbol{k}_t^\intercal$	$\mathbf{b}_t = \mathbf{S}_t \boldsymbol{q}_t$
RetNet [62]	$\mathbf{S}_t = \gamma \mathbf{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^\intercal$	$\mathbf{b}_t = \mathbf{S}_t \boldsymbol{q}_t$
Mamba [24]	$\mathbf{S}_t = \mathbf{S}_{t-1} \odot \exp(-(\boldsymbol{\alpha}_t \mathbf{1}^\intercal) \odot \exp(\mathbf{X}^A)) + (\boldsymbol{\alpha}_t \odot \boldsymbol{v}_t) \boldsymbol{k}_t^\intercal$	$\mathbf{b}_t = \mathbf{S}_t \boldsymbol{q}_t + \boldsymbol{d} \odot \boldsymbol{v}_t$
GLA [70]	$\mathbf{S}_t = \mathbf{S}_{t-1} \odot (\mathbf{1} \boldsymbol{\alpha}_t^\intercal) + \boldsymbol{v}_t \boldsymbol{k}_t^\intercal = \mathbf{S}_{t-1} \mathsf{Diag}(\boldsymbol{\alpha}_t) + \boldsymbol{v}_t \boldsymbol{k}_t^\intercal$	$\mathbf{b}_t = \mathbf{S}_t \boldsymbol{q}_t$
RWKV-6 [50]	$\mathbf{S}_t = \mathbf{S}_{t-1} \mathsf{Diag}(\boldsymbol{\alpha}_t) + \boldsymbol{v}_t \boldsymbol{k}_t^\intercal$	$\mathbf{b}_t = (\mathbf{S}_{t-1} + (\boldsymbol{d} \odot \boldsymbol{v}_t) \boldsymbol{k}_t^\intercal) \boldsymbol{q}_t$
HGRN-2 [54]	$\mathbf{S}_t = \mathbf{S}_{t-1} \mathsf{Diag}(\boldsymbol{\alpha}_t) + \boldsymbol{v}_t (\mathbf{1} - \boldsymbol{\alpha}_t)^\intercal$	$\mathbf{b}_t = \mathbf{S}_t \boldsymbol{q}_t$
mLSTM [40]	$\mathbf{S}_t = f_t \mathbf{S}_{t-1} + i_t \boldsymbol{v}_t \boldsymbol{k}_t^\intercal, \ \ \boldsymbol{z}_t = f_t \boldsymbol{z}_{t-1} + i_t \boldsymbol{k}_t$	$\mathbf{b}_t = \mathbf{S}_t \boldsymbol{q}_t / \max\{1, \lvert \boldsymbol{z}_t^\intercal \boldsymbol{q}_t \rvert\}$
Mamba-2 [15]	$\mathbf{S}_t = \gamma_t \mathbf{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^\intercal$	$\mathbf{b}_t = \mathbf{S}_t \boldsymbol{q}_t$
GSA [73]	$\mathbf{S}_t^{\boldsymbol{k}} = \mathbf{S}_{t-1}^{\boldsymbol{k}} \operatorname{Diag}(\boldsymbol{\alpha}_t) + \boldsymbol{k}_t \boldsymbol{\phi}_{\boldsymbol{t}}^\intercal, \ \ \mathbf{S}_t^{\boldsymbol{v}} = \mathbf{S}_{t-1}^{\boldsymbol{v}} \operatorname{Diag}(\boldsymbol{\alpha}_t) + \boldsymbol{v}_t \boldsymbol{\phi}_t^\intercal$	$\mathbf{b}_t = \mathbf{S}_t^{\boldsymbol{v}} \operatorname{softmax}\left(\mathbf{S}_t^{\boldsymbol{k}} \boldsymbol{q}_t\right)$
Gated DeltaNet [69]	$\mathbf{S}_t = \mathbf{S}_{t-1} \left(\alpha_t (\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\intercal)\right) + \beta_t \boldsymbol{v}_t \boldsymbol{k}_t^\intercal$	$\mathbf{b}_t = \mathbf{S}_t \boldsymbol{q}_t$

Efficient linear attention/SSMs: FlashLinearAttention [72]

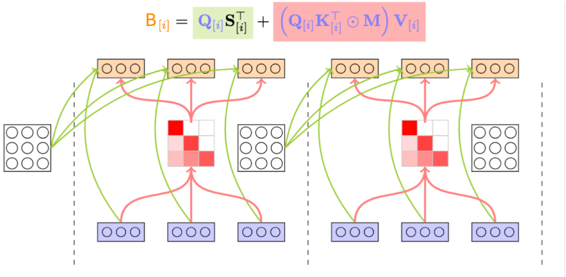

$\mathsf{B}_{[i]} = \mathbf{Q}_{[i]}\mathbf{S}_{[i]}^\top + \left(\mathbf{Q}_{[i]}\mathbf{K}_{[i]}^\top \odot \mathbf{M}\right)\mathbf{V}_{[i]}$

Figure: FlashLinearAttention [68].

Remarks:

- ◦ IO-aware, fast and efficient linear attention calculation algorithm.
- ◦ Supports many linear attention variations with decay factor.
- ◦ Has $\mathcal{O}(Td^2 + TdC)$ complexity where C is the chunk size.

$$\mathbf{B}_{[t]} = \underbrace{\mathbf{Q}_{[t]}\mathbf{S}_{[t]}^\top}_{\substack{\mathbb{R}^{C\times d}\quad \mathbb{R}^{d\times d}\\ \text{inter-chunk:}\mathbf{O}_{[t]}^{\text{inter}}}} + \underbrace{(\mathbf{Q}_{[t]}\mathbf{K}_{[t]}^\top \circ \mathbf{M})\mathbf{V}_{[t]}}_{\substack{\mathbb{R}^{C\times C}\quad \mathbb{R}^{C\times d}\\ \text{intra-chunk:}\mathbf{O}_{[t]}^{\text{intra}}}} \in \mathbb{R}^{C\times d}$$

Efficient linear attention/SSMs: SSD algorithm [15]

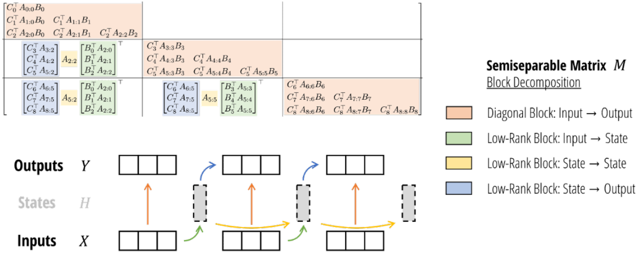

$C_0^\top A_{0:0} B_0$
$C_1^\top A_{1:0} B_0$ $C_1^\top A_{1:1} B_1$
$C_2^\top A_{2:0} B_0$ $C_2^\top A_{2:1} B_1$ $C_2^\top A_{2:2} B_2$
$\begin{bmatrix} C_3^\top A_{3:2} \\ C_4^\top A_{4:2} \\ C_5^\top A_{5:2} \end{bmatrix} A_{2:2} \begin{bmatrix} B_0^\top A_{2:0} \\ B_1^\top A_{2:1} \\ B_2^\top A_{2:2} \end{bmatrix}^\top$
$C_3^\top A_{3:3} B_3$
$C_4^\top A_{4:3} B_3$ $C_4^\top A_{4:4} B_4$
$C_5^\top A_{5:3} B_3$ $C_5^\top A_{5:4} B_4$ $C_5^\top A_{5:5} B_5$
$\begin{bmatrix} C_6^\top A_{6:5} \\ C_7^\top A_{7:5} \\ C_8^\top A_{8:5} \end{bmatrix} A_{5:2} \begin{bmatrix} B_0^\top A_{2:0} \\ B_1^\top A_{2:1} \\ B_2^\top A_{2:2} \end{bmatrix}^\top$
$\begin{bmatrix} C_6^\top A_{6:5} \\ C_7^\top A_{7:5} \\ C_8^\top A_{8:5} \end{bmatrix} A_{5:5} \begin{bmatrix} B_3^\top A_{5:3} \\ B_4^\top A_{5:4} \\ B_5^\top A_{5:5} \end{bmatrix}^\top$
$C_6^\top A_{6:6} B_6$
$C_7^\top A_{7:6} B_6$ $C_7^\top A_{7:7} B_7$
$C_8^\top A_{8:6} B_6$ $C_8^\top A_{8:7} B_7$ $C_8^\top A_{8:8} B_8$
Semiseparable Matrix M
Block Decomposition
🟧 Diagonal Block: Input → Output
🟩 Low-Rank Block: Input → State
🟨 Low-Rank Block: State → State
🟦 Low-Rank Block: State → Output
Outputs Y
States H
Inputs X

Figure: SSD algorithm of Mamba2 [15].

- **Block Matrix Decomposition:**
  - Divide the SSM matrix into $C \times C$ blocks.
  - Compute diagonal blocks using a quadratic (attention-like) form.
  - Factorize and compute off-diagonal blocks using batched matrix multiplications.
  - Process sequentially using modified A factors.
- **Chunking & State Passing:**
  - Split input into chunks of size C.
  - Compute local outputs in parallel (assuming zero initial state).
  - Compute final states of chunks in parallel.
  - Propagate states using a parallel or sequential scan.
  - Adjust outputs using true initial states.

Bidirectional sequence modeling

Question:
- What if all past and future tokens are available at the beginning?
- For instance, image classification, masked language modeling...

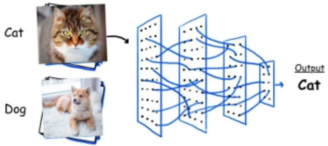


Figure: Image classification task

- Bidirectional transformers. Ex: ViT, BERT [19, 18].
- Bidirectional SSMs. Ex: Hydra, Vision Mamba [30, 75].
- Bidirectional RNNs. Ex: Vision-LSTM [3].
- Bidirectional linear attention. Ex: ???.

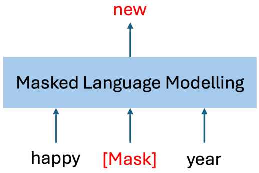


Figure: Masked language modelling

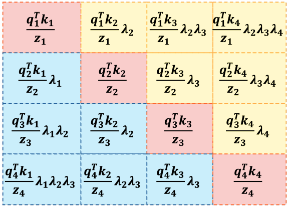
LION: LInear attentiON for bidirectional sequence modeling

Training

$$\mathbf{B} = ((\mathbf{Q}\mathbf{K}^T) \odot \mathbf{M}^{\text{LION}})\mathbf{V},$$

$$\mathbf{M}_{ij}^{\text{LION}} = \begin{cases} \Pi_{k=j+1}^{i} \lambda_k, & i > j \\ 1 & i = j \\ \Pi_{k=i+1}^{j} \lambda_k, & i < j. \end{cases}$$

- Three stable choices of decay parameter with LION
  - LION-LIT for $\lambda_i = 1$ which is bi-directional form of LinearTrans [38].
  - LION-D for $\lambda_i = \lambda$ fixed decay, and bi-directional form of RetNet [62].
  - LION-S for $\lambda_i = \sigma(\mathbf{W}\mathbf{x_i})$ being input dependent, and bi-directional Linear Transformer inspired by selectivity of Mamba2 [15].



$$\begin{array}{cccc} \frac{q_1^T k_1}{z_1} & \frac{q_1^T k_2}{z_1}\lambda_2 & \frac{q_1^T k_3}{z_1}\lambda_2\lambda_3 & \frac{q_1^T k_4}{z_1}\lambda_2\lambda_3\lambda_4 \\ \frac{q_2^T k_1}{z_2}\lambda_1 & \frac{q_2^T k_2}{z_2} & \frac{q_2^T k_3}{z_2}\lambda_3 & \frac{q_2^T k_4}{z_2}\lambda_3\lambda_4 \\ \frac{q_3^T k_1}{z_3}\lambda_1\lambda_2 & \frac{q_3^T k_2}{z_3}\lambda_2 & \frac{q_3^T k_3}{z_3} & \frac{q_3^T k_4}{z_3}\lambda_4 \\ \frac{q_4^T k_1}{z_4}\lambda_1\lambda_2\lambda_3 & \frac{q_4^T k_2}{z_4}\lambda_2\lambda_3 & \frac{q_4^T k_3}{z_4}\lambda_3 & \frac{q_4^T k_4}{z_4} \end{array}$$

Figure: LION linear attention

LION: linear attention for bidirectional sequence modeling

Recurrent Inference

$$
\begin{aligned}
\mathbf{S}_i^{F/B} &= \lambda_i \mathbf{S}_{i-1}^{F/B} + \mathbf{k}_i \mathbf{v}_i^\top, \\
\mathbf{z}_i^{F/B} &= \lambda_i \mathbf{z}_{i-1}^{F/B} + \mathbf{k}_i, \\
c_i^{F/B} &= \mathbf{q}_i{}^\top \mathbf{z}_i^{F/B} - \frac{\mathbf{q}_i{}^\top \mathbf{k}_i}{2}, \\
\mathbf{b}_i^{F/B} &= \mathbf{q}_i{}^\top \mathbf{S}_i^{F/B} - \frac{\mathbf{q}_i{}^\top \mathbf{k}_i}{2} \mathbf{v}_i, \\
\mathbf{b}_i &= \frac{\mathbf{b}_i^F + \mathbf{b}_i^B}{c_i^F + c_i^B}.
\end{aligned}
$$

Chunked Inference

$$
\begin{aligned}
\mathbf{P}_{[ij]} &= \mathbf{Q}_{[i]} \mathbf{K}_{[j]}^\top \odot \mathbf{M}_{[ij]}, \\
\mathbf{C}_{[ij]} &= \mathbf{C}_{[ij-1]} + \mathrm{Sum}(\mathbf{P}_{[ij]}), \\
\mathbf{S}_{[ij]} &= \mathbf{S}_{[ij-1]} + \mathbf{P}_{[ij]} \mathbf{V}_{[j]}, \\
\mathbf{B}_{[i]} &= \frac{\mathbf{S}_{[iN]}}{\mathbf{C}_{[iN]}}
\end{aligned}
$$

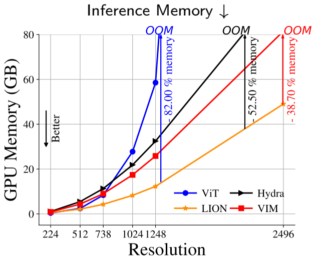


Figure: Inference memory resources of LION framework and other sota models

Training Times (relative to Transformer) ↓

Task	LION-LIT	LION-D	LION-S	Hydra	Vim
Vision	×0.73	×1.39	×1.46	×2.51	×10.86
MLM	×0.95	×1.10	×1.32	×3.13	×

Table: Existing bidirectional models employ more than ×2 the training time of a Transformer while LION maintains the transformer training speed of Transformers.

3. FFN as channel processing [48]

Observation: ○ $2/3$ of all the parameters of a transformer comes from FFNs.

Question: ○ Why do models need residual connections and an FFN as the last layer?

1. Individual token information:
   - Sequence mixer captures temporal relations.
   - It is also necessary to capture information that lies within each word.
2. Token uniformity [67] in the high dimensional embedding space:
   - Without the additional FFN, token representations can become very similar.
   - Model struggles to distinguish between tokens.

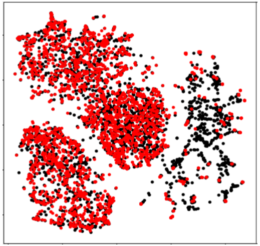


Figure: Token uniformity. t-SNE plot of the token embeddings of BERT model is visualized. Tokens exhibit clear clusters, indicating token uniformity [67].

3. FFN as channel processing

Question: ○ Why do models need residual connections and an FFN as the last layer?

○ The role of skip connections

- Motivated by ResNet [27].
- Prevents vanishing gradients by allowing gradients to flow directly from deeper layers to earlier layers.
- Smooths the loss surface.
- Preserves original information.

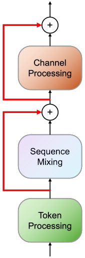


Figure: Residual connections

○ The role of FFN

- Introduces additional nonlinearity.
- Enhances the representation diversity by mapping to high dimentional space $d_{\text{ffn}} = 4d$.

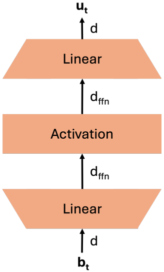


Figure: FFN as channel processing

Mixture of Experts (MoE)

- What is an MoE? [9, 58]
  - A modular approach where experts specialize in different input regions.
  - A gating function selects relevant experts per input, reducing compute cost.

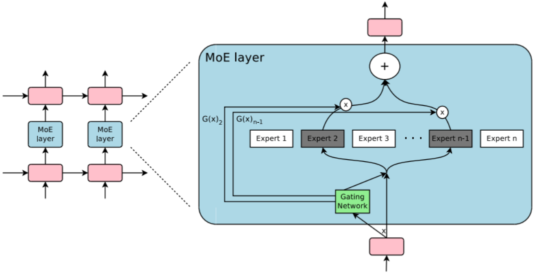


Figure: Sparse MoE with top-k expert selection [58].

- Mathematical formulation:

$$F_{\text{MoE}}(\mathbf{a}) = \sum_{i=1}^{K} G_i(\mathbf{a}) f_i(\mathbf{a}) \qquad (1)$$

$$G_i(\mathbf{a}) = \frac{\exp(g_i(\mathbf{a}))}{\sum_{j=1}^{K} \exp(g_j(\mathbf{a}))} \qquad (2)$$

  - $f_i(\mathbf{a})$ is the output of i^{th} expert.
  - $g_i(\mathbf{a})$ is the raw gating scores.
  - The softmax gate $G_i(\mathbf{a})$ ensures a probabilistic selection of experts.
- Cost changes form $\mathcal{O}(p)$ to $\mathcal{O}(kp/K)$.
  - p is the number of parameters

Dense vs Sparse MoE

- In a dense MoE, all experts are used for each input, making it computationally expensive.
- In a sparse MoE, only a subset (top-k) of experts is activated for each token, where

$$G_i(\mathbf{a}) = \mathrm{softmax}(\mathrm{Top}_k(g(\mathbf{a}) + R_{\text{noise}}, k)) \tag{3}$$

$$\mathrm{Top}_k(g(\mathbf{a}), k)_i = \begin{cases} g_i(\mathbf{a}), & \text{if } g_i(\mathbf{a}) \text{ is in the top-k elements of } g(\mathbf{a}), \\ -\infty, & \text{otherwise.} \end{cases} \tag{4}$$

- Adding noise $R_{\text{noise}} \in \mathbb{R}^K$ to a sparsely-gated MoE layer promotes expert exploration and stabilizes training.
- This sparsification reduces computational cost significantly, while scaling the model.

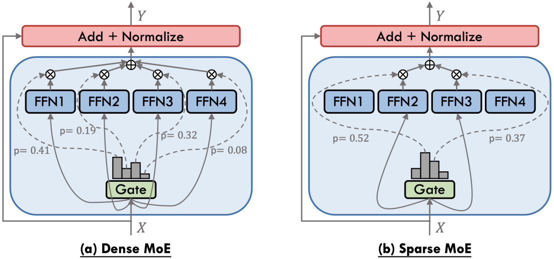


Figure: The MoE layer selects experts per input: (a) Dense MoE uses all, while (b) Sparse MoE activates the top-k [9].

Better scaling with MoEs

- Why do we use MoEs Instead of a larger model?
  - Compute cost of a dense model:
    - A standard FFN with p parameters uses all parameters in every forward pass.
    - Computational cost scales as $\mathcal{O}(p)$.
  - Sparse activation in an MoE:
    - MoE has K experts, each with $\frac{p}{K}$ parameters.
    - Only k experts $(k \ll K)$ are selected per input with $\mathcal{O}(k \cdot \frac{p}{K})$ cost during inference.

Key result:

MoE enables better scaling to large models without a proportional increase in compute cost.

- MoEs replace dense FFN layers with experts while preserving self-attention, enabling better scaling [9].

Examples:

- Mixtral-8x7B [34] - 8 experts, top-2 activation.
- DeepSeekMoE [13] - 16 experts, top-2/top-16 activation.
- DBRX [16] - Fine-grained expert segmentation.
- Qwen1.5-MoE [63] - Shared expert configurations.

Some example LLM architectures: Llama3 [20]

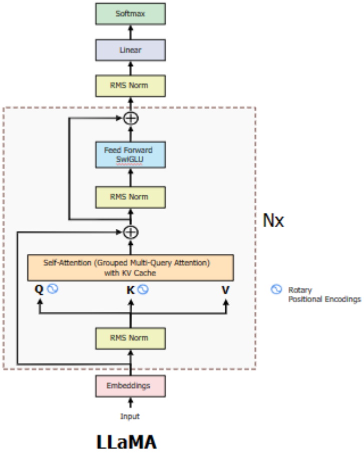


Figure: Llama3 architecture [39]

- Decoder-only transformer architecture.
- Tokenizer with a vocabulary of 128,000 tokens.
- Trained on sequences up to 8,192 tokens in length.
- Uses Grouped-Query Attention (GQA).
- Post-training:
  - Fine-tuned using supervised fine-tuning
  - Aligned with reinforcement learning with human feedback
- Three sizes:
  - 8B Model: 32 transformer layers,
  - 70B Model: 80 transformer layers,
  - 405B Model: 126 transformer layers.

Some example LLM architectures: Llama3 [20]

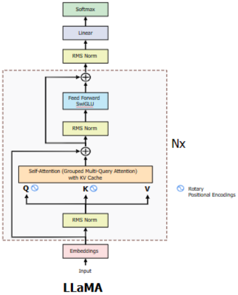

Softmax
Linear
RMS Norm
Feed Forward SwiGLU
RMS Norm
Nx
Self-Attention (Grouped Multi-Query Attention) with KV Cache
Q
K
V
Rotary Positional Encodings
RMS Norm
Embeddings
Input
LLaMA

Figure: Llama3 architecture [39]

○ RMS normalization

$$\mathrm{RMS}(\mathbf{A}) = \sqrt{\frac{1}{d}\sum_{j=1}^{d} \mathbf{A}_{::j}^2}$$

$$\hat{\mathbf{A}}_{::j} = \frac{\mathbf{A}_{::j}}{\mathrm{RMS}(\mathbf{A}) + \epsilon}$$

$$\hat{\mathbf{A}}_{::j} = \gamma_j \hat{\mathbf{A}}_{::j}$$

- γ_j is a learnable scaling parameter.
- ϵ is a small constant for numerical stability.

Some example LLM architectures: Mamba2 [15]

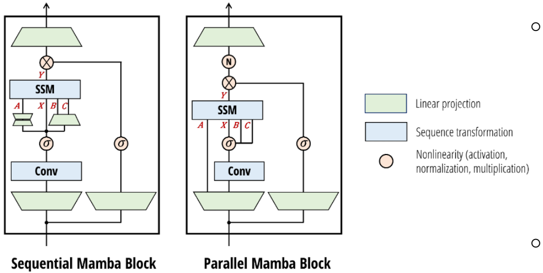


Figure: Mamba2 architecture [15]

- ○ Flexibility in configuration.
- ○ Specific implementations may vary.
- ○ 8B Mamba-2:
  - ▶ It has hidden dimension of 4096 and 56 layers.
  - ▶ Each Mamba-2 layer had an internal state dimension of 128, organized into eight groups.
  - ▶ It employs a head dimension of 64.
  - ▶ It has an expansion factor of two, and a convolution window size of four.
- ○ Capability to handle extremely long contexts
  - ▶ The passkey retrieval task [45]
  - ▶ 370M Mamba-2 achieves near-perfect accuracy
  - ▶ a context length of 256,000 tokens!

Some example LLM architectures: DeepSeek v3 [17]

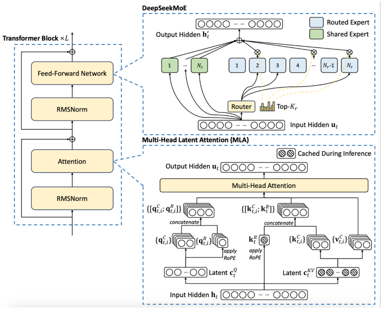


Figure: DeepSeek v3 architecture [1]

○ Mixture-of-Experts (MoE) Framework:
  - Dynamically activates subsets of model parameters.
  - Reduces computational cost while maintaining high performance.

○ Multi-head Latent Attention (MLA):
  - Compresses Key-Value (KV) cache into latent vectors.
  - Supports extended context lengths (up to 128,000 tokens).
  - Improves memory efficiency during inference.

Some example LLM architectures: DeepSeek v3 [17]

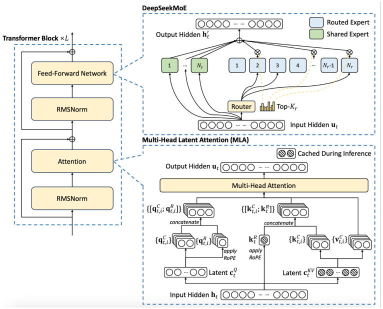


Figure: DeepSeek v3 architecture [1]

- Multi-Token Prediction:
  - Allows simultaneous generation of multiple tokens.
  - Enhances decoding speed without sacrificing accuracy.
- Training Methodology:
  - Trained on a multilingual corpus (primarily English and Chinese).
  - Fine-tuned with a focus on reasoning-intensive tasks like mathematics and programming.
- Reduces the dependency on large-scale GPU resources.

Some example LLM architectures: Recurrent depth approach [23]

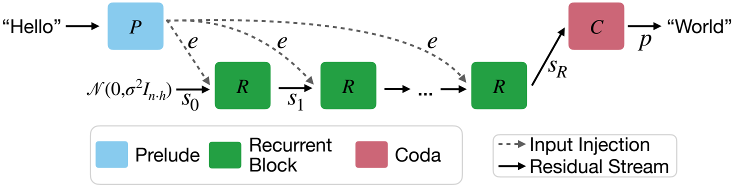


Figure: A visualization of the architecture. Each block consists of a number of sub-layers. The blue prelude block embeds the inputs into latent space, where the green shared recurrent block is a block of layers that is repeated to compute the final latent state, which is decoded by the layers of the red coda block. [23]

- Traditional models increase reasoning capacity by generating more tokens.
- An architecture that scales computation at test time using a Recurrent Depth Approach.
- Recurrent block iteration: A core recurrent block is iterated multiple times to refine reasoning.

Some example LLM architectures: Recurrent depth approach [23]

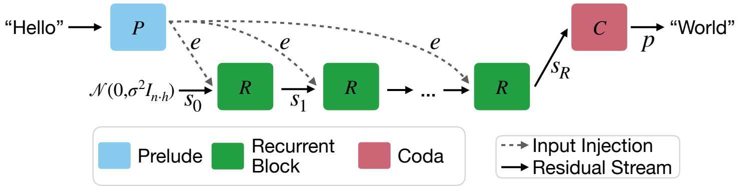


Figure: A visualization of the architecture. Each block consists of a number of sub-layers. The blue prelude block embeds the inputs into latent space, where the green shared recurrent block is a block of layers that is repeated to compute the final latent state, which is decoded by the layers of the red coda block. [23]

- Dynamic test-time computation: Computational depth can be increased as needed during inference.
- Latent space processing: Reasoning is performed internally, minimizing unnecessary token generation.
- Inference computation can be scaled without changing model parameters.

Wrap up!

- Lecture 2 about Optimization next Thursday!

Supplementary Material

⋆ Rotary position embedding in self-attention

Solution 3 [61]

- ○ Rotary position encoding: incorporate both absolute position and relative position.
- ○ Given $\boldsymbol{q}_t$ and $\boldsymbol{k}_{t'}$, we want to find a position encoding function Pos($\dot{}$) such that:

$$\langle \mathsf{Pos}(\boldsymbol{q}_t), \mathsf{Pos}(\boldsymbol{k}_{t'}) \rangle = \mathsf{SomeFunction}(\boldsymbol{q}_t, \boldsymbol{k}_{t'}, \color{red}{t - t'}).$$

- ○ Assume $m = 2$ (can be generalized to $m > 2$): by the derivation in [61], one can use

$$\mathsf{Pos}(\boldsymbol{q}_t) := \begin{bmatrix} \cos t, & -\sin t \\ \sin t, & \cos t \end{bmatrix} \boldsymbol{q}_t, \qquad \mathsf{Pos}(\boldsymbol{k}_{t'}) := \begin{bmatrix} \cos t', & -\sin t' \\ \sin t', & \cos t' \end{bmatrix} \boldsymbol{k}_{t'}.$$

- ○ Achieve better performance on various long text tasks.
- ○ Being employed in several recent LLMs [12, 64].

⋆ Grouped-query attention

Grouped-Query Attention (GQA) [2]

Reduces computational cost by sharing key-value pairs across multiple queries, for the group g:

$$\mathrm{g\_head_i} = \text{Attention}(Q\mathbf{X}_i^Q, K\mathbf{X}_g^K, V\mathbf{X}_g^V).$$

Used in large-scale models (e.g., LLaMA-3.1 [21], Mistral [33]) for faster inference and reduced memory usage.

KV Cache Efficiency

GQA improves KV caching by reducing memory overhead since fewer key-value pairs need to be stored, leading to faster auto-regressive decoding.

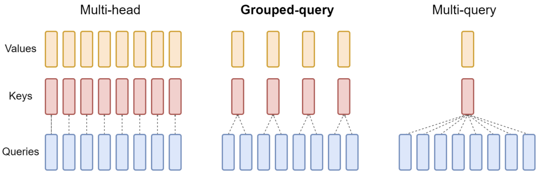


Figure: GQA: Interpolates between multi-head and multi-query attention by sharing key-value heads in query groups [2].

⋆ Zero-order hold discretization

Below, the zero-order hold discretization derived by [36] is explained. An LTI system can be represented with the equation:

$$\mathbf{h}'(t) = A\mathbf{h}(t) + B\mathbf{x}(t), \tag{5}$$

which can be rearranged to isolate $\mathbf{h}(t)$:

$$\mathbf{h}'(t) - A\mathbf{h}(t) = B\mathbf{x}(t). \tag{6}$$

By multiplying the equation by e^{-At}, we get

$$e^{-At}\mathbf{h}'(t) - e^{-At}A\mathbf{h}(t) = e^{-At}B\mathbf{x}(t) \tag{7}$$

Since $\frac{\partial}{\partial t}e^{At} = Ae^{At} = e^{At}A$, (3) can be written as:

$$\frac{\partial}{\partial t}\left(e^{-At}\mathbf{h}(t)\right) = e^{-At}B\mathbf{x}(t). \tag{8}$$

After integrating both sides and simplifications, we get

$$e^{-At}\mathbf{h}(t) = \int_0^t e^{-A\tau}B\mathbf{x}(\tau)\,d\tau + \mathbf{h}(0). \tag{9}$$

⋆ Zero-order hold discretization

After multiplying by $e^{\mathbf{A}T}$ and rearranging we get

$$e^{\mathbf{A}(k+1)T}\mathbf{h}(0) = e^{\mathbf{A}T}\mathbf{h}_k - e^{\mathbf{A}(k+1)T}\int_0^{kT} e^{-\mathbf{A}\tau}\mathbf{B}\mathbf{x}(\tau)\,d\tau. \tag{15}$$

Plugging this expression for $\mathbf{x}_{k+1}$ in (10) yields to

$$\mathbf{h}_{k+1} = e^{\mathbf{A}T}\mathbf{h}_k - e^{\mathbf{A}(k+1)T}\left(\int_0^{kT} e^{-\mathbf{A}\tau}\mathbf{B}\mathbf{x}(\tau)\,d\tau + \int_0^{(k+1)T} e^{-\mathbf{A}\tau}\mathbf{B}\mathbf{x}(\tau)\,d\tau\right), \tag{16}$$

which can be further simplified to

$$\mathbf{h}_{k+1} = e^{\mathbf{A}T}\mathbf{h}_k - e^{\mathbf{A}(k+1)T}\int_{kT}^{(k+1)T} e^{-\mathbf{A}\tau}\mathbf{B}\mathbf{x}(\tau)\,d\tau. \tag{17}$$

Now, assuming that $\mathbf{x}(t)$ is constant on the interval $[kT,(k+1)T)$, which allows us to take $\mathbf{B}\mathbf{x}(t)$ outside the integral. Moreover, by bringing the $e^{\mathbf{A}(k+1)T}$ term inside the integral we have

$$\mathbf{h}_{k+1} = e^{\mathbf{A}T}\mathbf{h}_k - \int_{kT}^{(k+1)T} e^{\mathbf{A}((k+1)T-\tau)}\,d\tau\,\mathbf{B}\mathbf{x}_k. \tag{18}$$

⋆ Zero-order hold discretization

Using a change of variables $v = (k+1)T - \tau$, with $d\tau = -dv$, and reversing the integration bounds results in

$$\mathbf{h}_{k+1} = e^{\mathbf{A}T}\mathbf{h}_k + \int_0^T e^{\mathbf{A}v}\, dv\, \mathbf{B}\mathbf{x}_k. \tag{19}$$

Finally, if we evaluate the integral by noting that $\frac{d}{dt}e^{\mathbf{A}t} = \mathbf{A}e^{\mathbf{A}t}$ and assuming $\mathbf{A}$ is invertible, we get

$$\mathbf{h}_{k+1} = e^{\mathbf{A}T}\mathbf{h}_k + \mathbf{A}^{-1}\left(e^{\mathbf{A}T} - \mathbf{I}\right)\mathbf{B}\mathbf{x}_k. \tag{20}$$

Thus, we find the discrete-time state and input matrices:

$$\tilde{\mathbf{A}} = e^{\mathbf{A}T} \tag{21}$$

$$\tilde{\mathbf{B}} = \mathbf{A}^{-1}\left(e^{\mathbf{A}T} - \mathbf{I}\right)\mathbf{B}. \tag{22}$$

And the final discrete state space representation is:

$$\mathbf{h}_{\mathbf{k}} = e^{\mathbf{A}T}\mathbf{h}_{k-1} + \mathbf{A}^{-1}\left(e^{\mathbf{A}T} - \mathbf{I}\right)\mathbf{B}_k\mathbf{x}_k. \tag{23}$$

References I

[1] Epoch AI.
How has deepseek improved the transformer architecture?, 2024.
Accessed: 2025-02-16.
(Cited on pages 77 and 78.)

[2] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai.
Gqa: Training generalized multi-query transformer models from multi-head checkpoints.
arXiv preprint arXiv:2305.13245, 2023.
(Cited on pages 46 and 84.)

[3] Benedikt Alkin, Maximilian Beck, Korbinian Pöppel, Sepp Hochreiter, and Johannes Brandstetter.
Vision-LSTM: xLSTM as generic vision backbone.
arXiv preprint arXiv:2406.04303, 2024.
(Cited on page 66.)

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton.
Layer normalization.
arXiv preprint arXiv:1607.06450, 2016.
(Cited on page 49.)

References II

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio.
Neural machine translation by jointly learning to align and translate, 2016.
(Cited on page 29.)

[6] Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter.
xlstm: Extended long short-term memory.
arXiv preprint arXiv:2405.04517, 2024.
(Cited on page 58.)

[7] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent.
A neural probabilistic language model.
Advances in neural information processing systems, 13, 2000.
(Cited on page 26.)

[8] Guillaume Blanc and Steffen Rendle.
Adaptive sampled softmax with kernel based sampling.
arXiv preprint arXiv:1712.00527, 2017.
(Cited on page 51.)

References III

[9] Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang.
A survey on mixture of experts.
arXiv preprint arXiv:2407.06204, 2024.
(Cited on pages 71, 72, and 73.)

[10] Vitaliy Chiley, Ilya Sharapov, Atli Kosson, Urs Koster, Ryan Reece, Sofia Samaniego de la Fuente, Vishal Subbiah, and Michael James.
Online normalization for training neural networks.
In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
(Cited on page 49.)

[11] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio.
Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.
(Cited on pages 29 and 30.)

[12] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al.
Palm: Scaling language modeling with pathways.
Journal of Machine Learning Research, 2022.
(Cited on page 83.)

References IV

[13] Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y Wu, et al.
Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models.
arXiv preprint arXiv:2401.06066, 2024.
(Cited on page 73.)

[14] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré.
Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022.
(Cited on page 47.)

[15] Tri Dao and Albert Gu.
Transformers are ssms: Generalized models and efficient algorithms through structured state space duality.
In *Forty-first International Conference on Machine Learning*, 2024.
(Cited on pages 24, 61, 62, 63, 65, 67, and 76.)

[16] Databricks.
Introducing DBRX: A New State-of-the-Art Open LLM, March 2024.
(Cited on page 73.)

References V

[17] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang,

References VI

Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan.
Deepseek-v3 technical report, 2024.
(Cited on pages 77 and 78.)

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova.
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)*, 2019.
(Cited on page 66.)

[19] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby.
An image is worth 16x16 words: Transformers for image recognition at scale, 2021.
(Cited on page 66.)

References VII

[20] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, A. Goyal, A. Hartshorn, et al.
Llama 3: Open foundation and instruction-tuned models.
arXiv preprint arXiv:2407.21783, 2024.
(Cited on pages 74 and 75.)

[21] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al.
The llama 3 herd of models.
arXiv preprint arXiv:2407.21783, 2024.
(Cited on page 84.)

[22] Jeffrey L. Elman.
Finding structure in time.
Cognitive Science, 14(2):179–211, 1990.
(Cited on page 27.)

[23] Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein.
Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach.
arxiv:2502.05171[cs], February 2025.
(Cited on pages 79 and 80.)

References VIII

[24] Albert Gu and Tri Dao.
Mamba: Linear-time sequence modeling with selective state spaces.
In *Conference on Learning and Modeling (COLM 2024)*, 2024.
(Cited on pages 24, 60, 61, 62, and 63.)

[25] Albert Gu, Karan Goel, and Christopher Ré.
Efficiently modeling long sequences with structured state spaces, 2022.
(Cited on pages 59 and 63.)

[26] Albert Gu, Isys Johnson, Aman Timalsina, Atri Rudra, and Christopher Ré.
How to train your hippo: State space models with generalized orthogonal basis projections, 2022.
(Cited on page 61.)

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.
Deep residual learning for image recognition.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
(Cited on pages 48 and 70.)

[28] Sepp Hochreiter and Jürgen Schmidhuber.
Long short-term memory.
Neural Computation, 9(8):1735–1780, 11 1997.
(Cited on pages 29 and 30.)

References IX

[29] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi.
The curious case of neural text degeneration.
In *International Conference on Learning Representations*, 2020.
(Cited on page 29.)

[30] Sukjun Hwang, Aakash Lahoti, Tri Dao, and Albert Gu.
Hydra: Bidirectional state space models through generalized matrix mixers, 2024.
(Cited on page 66.)

[31] Hassaan Idrees.
Exploring multi-head attention: Why more heads are better than one, 2024.
Accessed: 2025-02-19.
(Cited on page 46.)

[32] Md Rabiul Islam, Mohammad Ali Moni, Md Milon Islam, Md Rashed-Al-Mahfuz, Md Saiful Islam, Md Kamrul Hasan, Md Sabir Hossain, Mohiuddin Ahmad, Shahadat Uddin, Akm Azad, et al.
Emotion recognition from eeg signal focusing on deep learning and shallow learning techniques.
IEEE Access, 9:94601–94624, 2021.
(Cited on page 16.)

References X

[33] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al.
Mistral 7b.
arXiv preprint arXiv:2310.06825, 2023.
(Cited on page 84.)

[34] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al.
Mixtral of experts.
arXiv preprint arXiv:2401.04088, 2024.
(Cited on page 73.)

[35] Dan Jurafsky and James H. Martin.
Speech and Language Processing (3rd ed. draft).
draft, third edition, 2023.
(Cited on pages 10, 12, and 17.)

[36] Rudolph Emil Kalman.
A new approach to linear filtering and prediction problems.
1960.
(Cited on pages 59 and 85.)

References XI

[37] Jungo Kasai, Hao Peng, Yizhe Zhang, Dani Yogatama, Gabriel Ilharco, Nikolaos Pappas, Yi Mao, Weizhu Chen, and Noah A. Smith.
Finetuning pretrained transformers into RNNs.
In Marie-Francine Moens, Xuanjing Huang, and Scott Wen-tau Specia, Lucia andz Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10630–10643, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics.
(Cited on page 63.)

[38] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret.
Transformers are rnns: fast autoregressive transformers with linear attention.
In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.
(Cited on pages 50, 51, 63, and 67.)

[39] Pranjal Khadka.
LLaMA Explained!
Towards AI, April 2024.
Accessed: 2025-02-16.
(Cited on pages 74 and 75.)

[40] Ben Krause, Liang Lu, Iain Murray, and Steve Renals.
Multiplicative lstm for sequence modelling, 2017.
(Cited on pages 58 and 63.)

References XII

[41] Huanru Henry Mao.
Fine-tuning pre-trained transformers into decaying fast weights.
In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 10236–10242, Abu Dhabi, United Arab Emirates, 2022. Association for Computational Linguistics.
(Cited on page 63.)

[42] Andrey Andreyevich Markov.
Essai d'une recherche statistique sur le texte du roman.
Eugene Onegin" illustrant la liaison des epreuve en chain ('Example of a statistical investigation of the text of "Eugene Onegin" illustrating the dependence between samples in chain')". In: Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg). 6th ser, 7:153–162, 1913.
(Cited on page 12.)

[43] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean.
Efficient estimation of word representations in vector space.
In *International Conference on Learning Representations*, 2013.
(Cited on page 17.)

References XIII

[44] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur.
Recurrent neural network based language model.
In *Interspeech*, 2010.
(Cited on page 27.)

[45] Amirkeivan Mohtashami and Martin Jaggi.
Landmark attention: Random-access infinite context length for transformers.
arXiv preprint arXiv:2305.16300, 2023.
(Cited on page 76.)

[46] Changwon Ok, Geonseok Lee, and Kichun Lee.
Informative language encoding by variational autoencoders using transformer.
Applied Sciences, 12(16), 2022.
(Cited on page 46.)

[47] Charles Egerton Osgood, George J Suci, and Percy H Tannenbaum.
The measurement of meaning.
University of Illinois press, 1957.
(Cited on page 16.)

References XIV

[48] Marvelous Catawba Otter.
Understand the role of ffns in transformers.
`https://medium.com/@marvelous_catawba_otter_200/understand-the-role-of-ffns-in-transformers-51606a56e654`, 2023.
Accessed: 2025-02-16.
(Cited on page 69.)

[49] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio.
On the difficulty of training recurrent neural networks.
In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.
(Cited on page 29.)

[50] Bo Peng, Daniel Goldstein, Quentin Anthony, Alon Albalak, Eric Alcaide, Stella Biderman, Eugene Cheah, Xingjian Du, Teddy Ferdinan, Haowen Hou, Przemysław Kazienko, Kranthi Kiran GV, Jan Kocoń, Bartłomiej Koptyra, Satyapriya Krishna, Ronald McClelland Jr., Niklas Muennighoff, Fares Obeid, Atsushi Saito, Guangyu Song, Haoqin Tu, Stanisław Woźniak, Ruichong Zhang, Bingchen Zhao, Qihang Zhao, Peng Zhou, Jian Zhu, and Rui-Jie Zhu.
Eagle and Finch: RWKV with Matrix-Valued States and Dynamic Recurrence, 2024.
(Cited on page 63.)

References XV

[51] Hao Peng, Jungo Kasai, Nikolaos Pappas, Dani Yogatama, Zhaofeng Wu, Lingpeng Kong, Roy Schwartz, and Noah A. Smith.
ABC: Attention with bounded-memory control.
In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7469–7483, Dublin, Ireland, 2022. Association for Computational Linguistics.
(Cited on page 63.)

[52] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong.
Random feature attention.
In *International Conference on Learning Representations*, 2021.
(Cited on pages 58 and 63.)

[53] Ofir Press, Noah A. Smith, and Mike Lewis.
Train short, test long: Attention with linear biases enables input length extrapolation.
arXiv preprint arXiv:2108.12409, 2021.
(Cited on page 41.)

[54] Zhen Qin, Songlin Yang, Weixuan Sun, Xuyang Shen, Dong Li, Weigao Sun, and Yiran Zhong.
HGRN2: Gated Linear RNNs with State Expansion.
2024.
(Cited on page 63.)

References XVI

[55] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever.
Improving language understanding by generative pre-training.
Technical report, OpenAI, 2018.
(Cited on page 37.)

[56] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever.
Improving language understanding with unsupervised learning.
Technical report, OpenAI, 2018.
(Cited on page 48.)

[57] Ananda S. Rawat, Jiecao Chen, Felix X. X. Yu, Ananda Theertha Suresh, and Sanjiv Kumar.
Sampled softmax with random fourier features.
In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 13834–13844, 2019.
(Cited on page 51.)

[58] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean.
Outrageously large neural networks: The sparsely-gated mixture-of-experts layer.
arXiv preprint arXiv:1701.06538, 2017.
(Cited on page 71.)

References XVII

[59] Zhifan Shen, Ming Zhang, Haiyu Zhao, Shuai Yi, and Hongsheng Li.
Efficient attention: Attention with linear complexities.
arXiv preprint arXiv:1812.01243, 2020.
(Cited on page 51.)

[60] Jimmy T. H. Smith, Andrew Warrington, and Scott W. Linderman.
Simplified state space layers for sequence modeling.
In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.
(Cited on page 63.)

[61] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu.
Roformer: Enhanced transformer with rotary position embedding.
Neurocomputing, page 127063, 2023.
(Cited on pages 23 and 83.)

[62] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei.
Retentive network: A successor to transformer for large language models.
arXiv preprint arXiv:2307.08621, 2023.
(Cited on pages 24, 56, 57, 63, and 67.)

References XVIII

[63] Qwen Team.
Qwen1.5-MoE: Matching 7B Model Performance with 1/3 Activated Parameters", February 2024.
(Cited on page 73.)

[64] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al.
Llama 2: Open foundation and fine-tuned chat models.
arXiv preprint arXiv:2307.09288, 2023.
(Cited on page 83.)

[65] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin.
Attention is all you need.
In *Advances in Neural Information Processing Systems*, 2017.
(Cited on pages 22, 23, 24, 31, 36, 46, and 48.)

[66] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin.
Understanding and improving layer normalization.
Advances in Neural Information Processing Systems, 32, 2019.
(Cited on page 49.)

References XIX

[67] Hanqi Yan, Lin Gui, Wenjie Li, and Yulan He.
Addressing token uniformity in transformers via singular value transformation.
In *Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2022.
(Cited on page 69.)

[68] Songlin Yang.
What's next for mamba? towards more expressive recurrent update rules.
`https://sustcsonglin.github.io/assets/pdf/talk_250117.pdf`, January 2025.
Presentation slides.
(Cited on pages 62 and 64.)

[69] Songlin Yang, Jan Kautz, and Ali Hatamizadeh.
Gated delta networks: Improving mamba2 with delta rule, 2024.
(Cited on page 63.)

[70] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim.
Gated linear attention transformers with hardware-efficient training.
ArXiv preprint, abs/2312.06635, 2023.
(Cited on page 63.)

References XX

[71] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim.
Parallelizing linear transformers with the delta rule over sequence length.
arXiv preprint arXiv:2406.06484, 2024.
(Cited on pages 58 and 63.)

[72] Songlin Yang and Yu Zhang.
Fla: A triton-based library for hardware-efficient implementations of linear attention mechanism, 2024.
Version 0.0.1.
(Cited on page 64.)

[73] Yu Zhang, Songlin Yang, Ruijie Zhu, Yue Zhang, Leyang Cui, Yiqiao Wang, Bolun Wang, Freda Shi, Bailin Wang, Wei Bi, Peng Zhou, and Guohong Fu.
Gated slot attention for efficient linear-time sequence modeling.
In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.
(Cited on page 63.)

[74] Beitong Zhou, Cheng Cheng, Guijun Ma, and Yong Zhang.
Remaining useful life prediction of lithium-ion battery based on attention mechanism with positional encoding.
In *IOP Conference Series: Materials Science and Engineering*, volume 895, page 012006. IOP Publishing, 2020.
(Cited on pages 22 and 23.)

References XXI

[75] Lianghui Zhu, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang.
Vision mamba: Efficient visual representation learning with bidirectional state space model, 2024.
(Cited on page 66.)