Recent neural architectures for language

AI602: Recent Advances in Deep Learning

Lecture 1

Jinwoo Shin

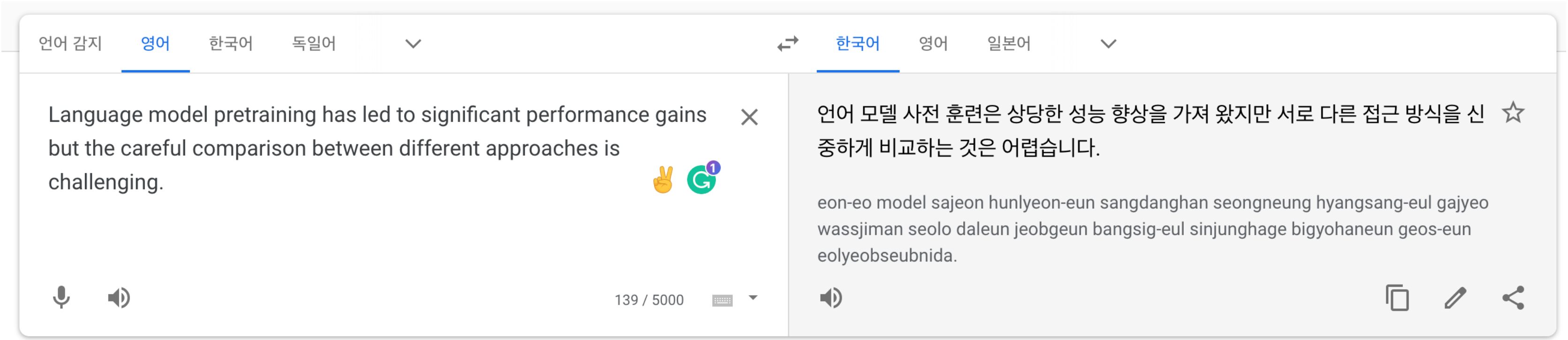
KAIST AI

Motivation: Temporal Data in Real World

- Many real-world data has a **temporal structure** intrinsically
    - Natural language

“Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was __.” ⟶ *terrible*

Language modeling



Language model pretraining has led to significant performance gains but the careful comparison between different approaches is challenging.

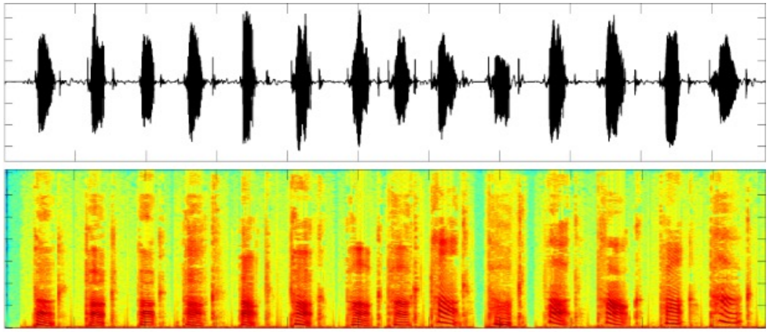
언어 모델 사전 훈련은 상당한 성능 향상을 가져 왔지만 서로 다른 접근 방식을 신중하게 비교하는 것은 어렵습니다.

eon-eo model sajeon hunlyeon-eun sangdanghan seongneung hyangsang-eul gajyeo wassjiman seolo daleun jeobgeun bangsig-eul sinjunghage bigyohaneun geos-eun eolyeobseubnida.

Translation

Motivation: Temporal Data in Real World

- Many real-world data has a **temporal structure** intrinsically
  - Natural language
  - Speech



Motivation: Temporal Data in Real World

- Many real-world data has a **temporal structure** intrinsically
    - Natural language
    - Speech
    - Video

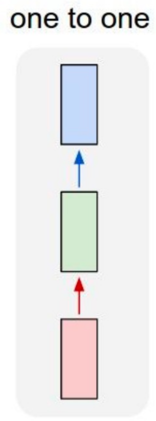
Motivation: Temporal Data in Real World

- Many real-world data has a **temporal structure** intrinsically
  - Natural language
  - Speech
  - Video
  - Stock prices, and etc…

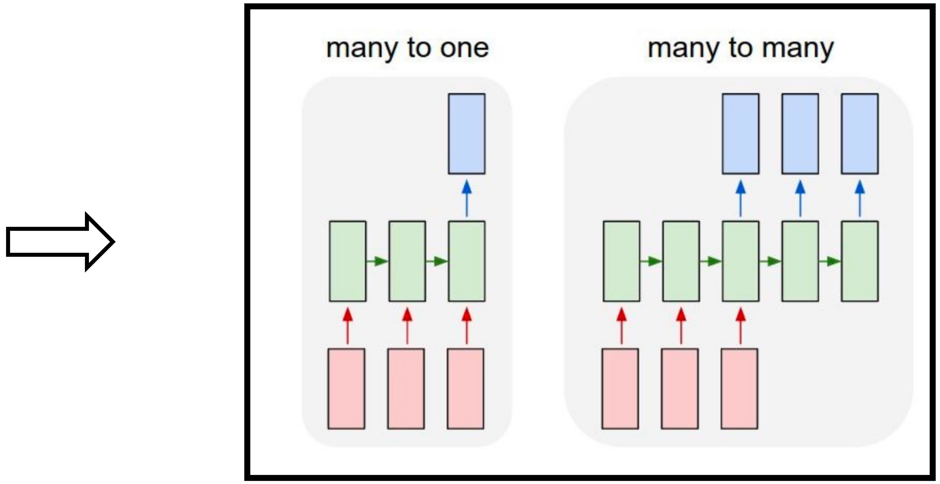
Motivation: Temporal Data in Real World

- Many real-world data has a **temporal structure** intrinsically
  - "**Natural language**"
  - Speech
  - Video
  - Stock prices, and etc…

- In order to solve much complicated real-world problems, we need a **better architecture to capture temporal dependency** in the data
  - Specifically, we will focus on the recent models for natural language in this lecture



Vanilla neural network



Overview

Part 1. Basics

- RNN to LSTM
- Sequence-to-sequence Model
- Attention-based NLP Model

Part 2. Transformers and Large Language Models

- Transformer (self-attention)
- Pre-training of Transformers and Language Models

Part 3. Advanced Topics

- Handling long inputs with Transformers
- Techniques for improving efficiency
- State-Space Models

Part 4. Summary

Overview

Part 1. Basics

- RNN to LSTM
- Sequence-to-sequence Model
- Attention-based NLP Model

Part 2. Transformers and Large Language Models

- Transformer (self-attention)
- Pre-training of Transformers and Language Models

Part 3. Advanced Topics

- Handling long inputs with Transformers
- Techniques for improving efficiency
- State-Space Models

Part 4. Summary

Vanilla RNN

- Process a sequence of vectors by applying **recurrence formula** at every time step :

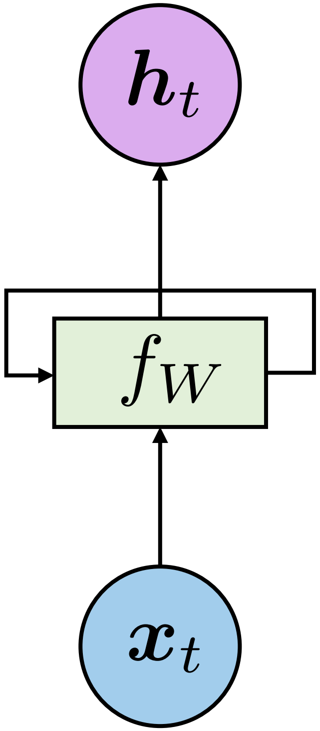
$$\boxed{\boldsymbol{h}_t} = \boxed{f_W}(\boxed{\boldsymbol{h}_{t-1}}, \boxed{\boldsymbol{x}_t})$$

New state

Old state

Input vector at time step t

Function parameterized by learnable W



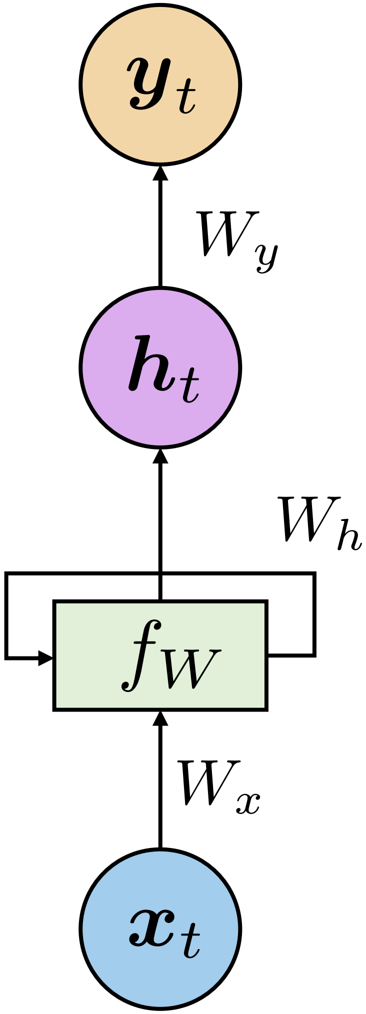
Vanilla RNN

- Vanilla RNN (or sometimes called **Elman RNN**)
  - The state consists of a single "hidden" vector $\mathbf{h}_t$

$$\boldsymbol{h}_t = f_W(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$$

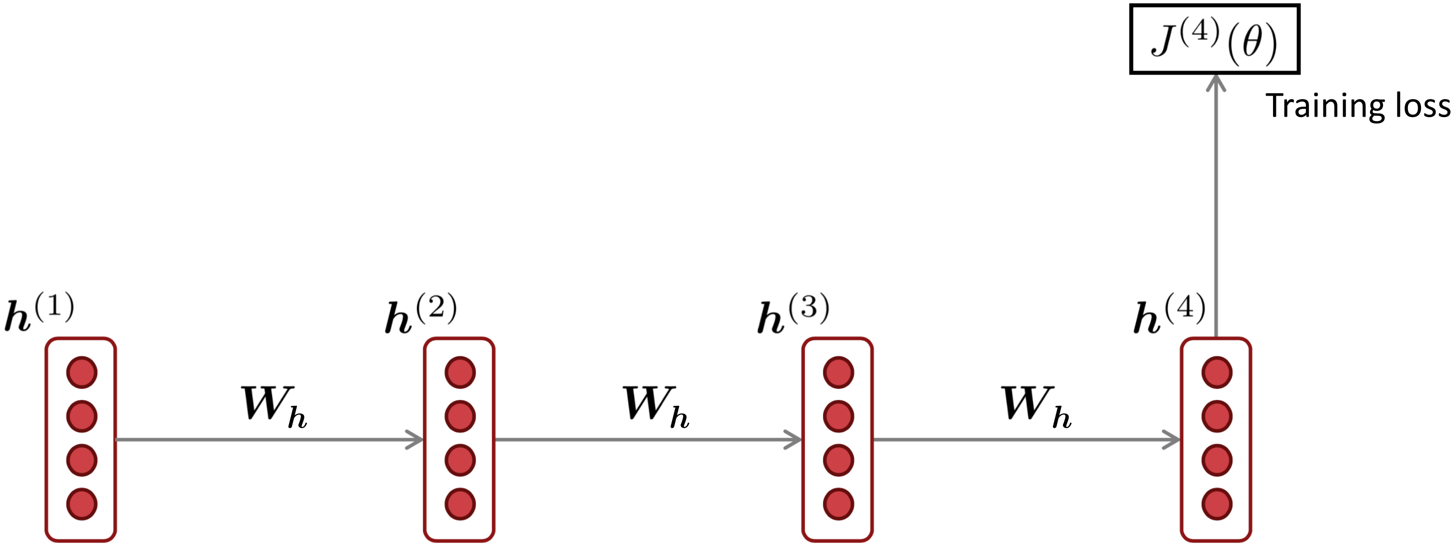
$$\downarrow$$

$$\boxed{\begin{aligned} \boldsymbol{h}_t &= \tanh(W_h \boldsymbol{h}_{t-1} + W_x \boldsymbol{x}_t) \\ \boldsymbol{y}_t &= W_y \boldsymbol{h}_t \end{aligned}}$$



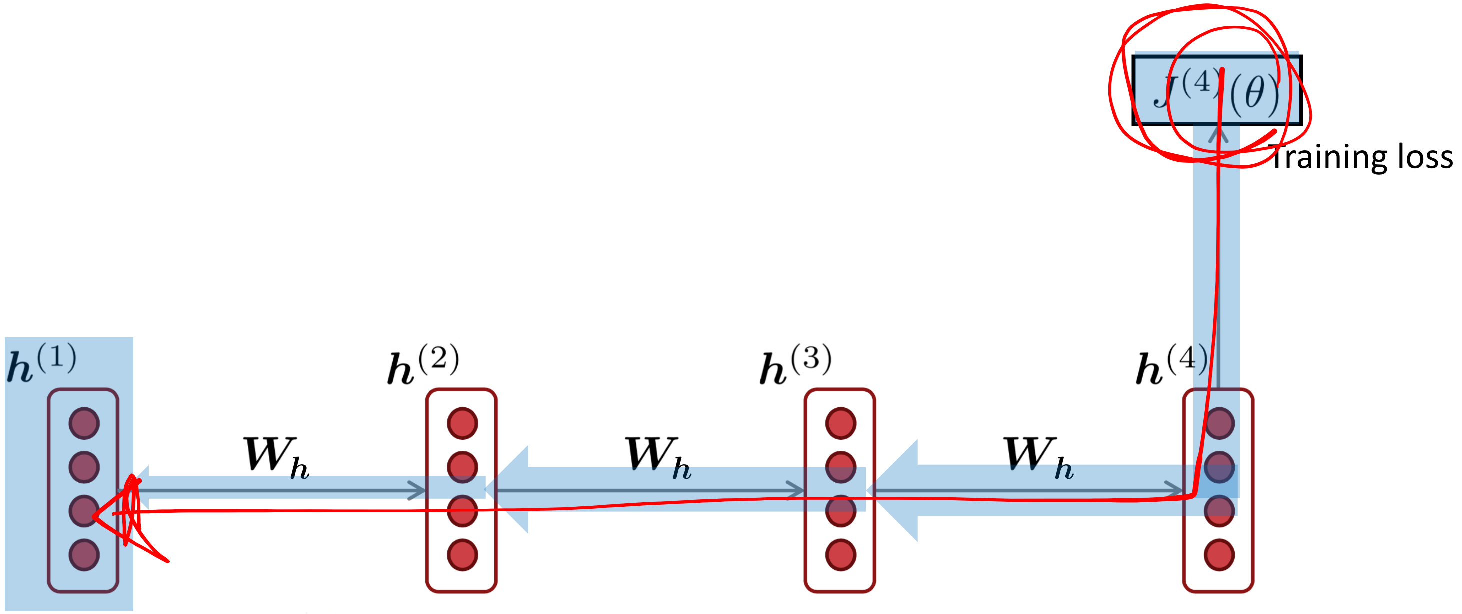
Why Do We Need to Develop RNN Architectures?

- E.g., RNN with a sequence of length 4



Why Do We Need to Develop RNN Architectures?

- E.g., RNN with a sequence of length 4
  - Consider a gradient from the first state $\boldsymbol{h}^{(1)}$



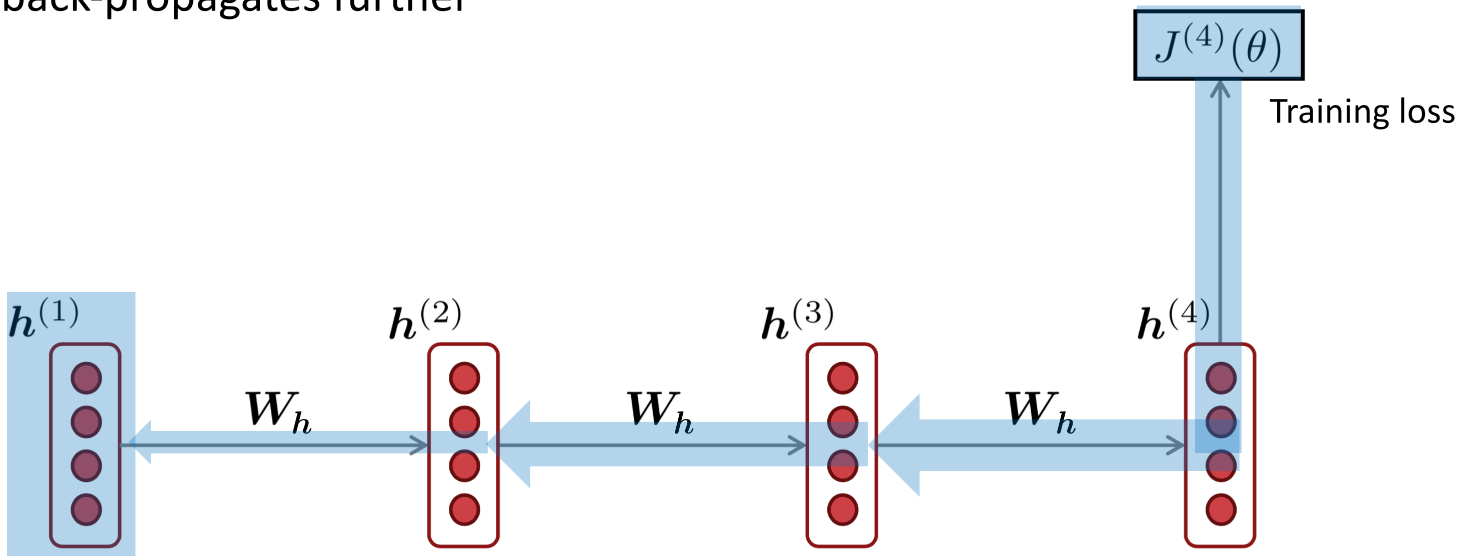
$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

Chain rule!

Why Do We Need to Develop RNN Architectures?

- E.g., RNN with a sequence of length 4
  - Consider a gradient from the first state $\boldsymbol{h}^{(1)}$

- What happens if $\frac{\partial \boldsymbol{h}^{(i+1)}}{\partial \boldsymbol{h}^{(i)}}$ are **too small**? ⇨ **Vanishing gradient problem**
  - When these are small, the gradient signal gets smaller and smaller as it back-propagates further



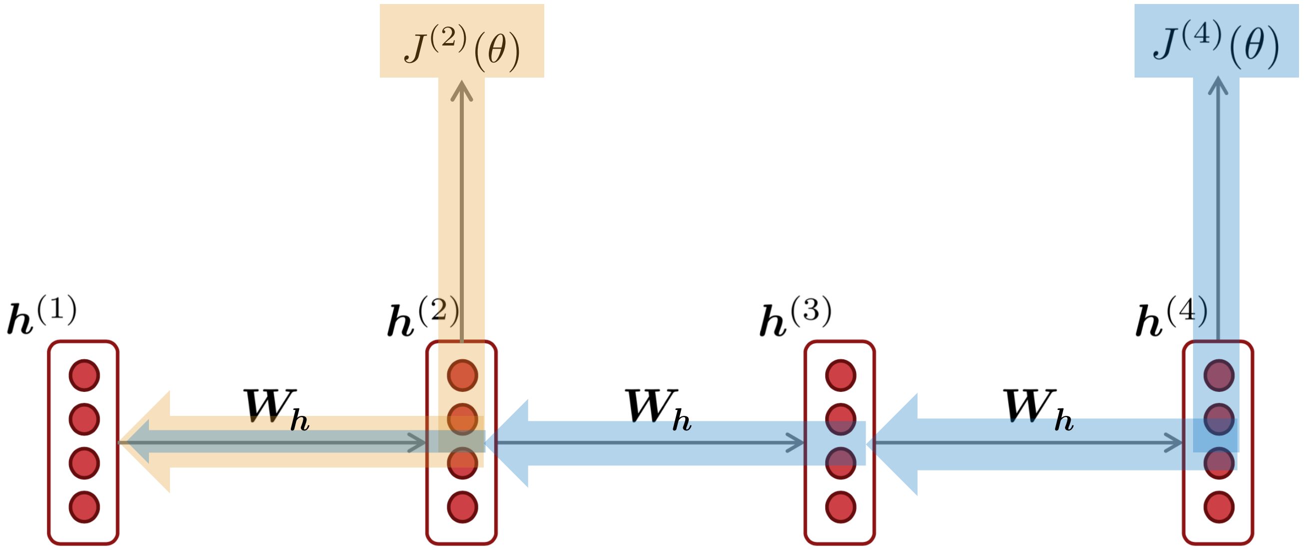
$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \boxed{\frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

Chain rule!

Why Do We Need to Develop RNN Architectures?

- E.g., RNN with a sequence of length 4
  - Consider a gradient from the first state $\boldsymbol{h}^{(1)}$

- What happens if $\frac{\partial \boldsymbol{h}^{(i+1)}}{\partial \boldsymbol{h}^{(i)}}$ are **too small**? ⇒ **Vanishing gradient problem**
  - When these are small, the gradient signal gets smaller and smaller as it back-propagates further
  - So, model weight are updated only with respect to near effects, **not** long-term effects.



Why Do We Need to Develop RNN Architectures?

- E.g., RNN with a sequence of length 4
  - Consider a gradient from the first state $\boldsymbol{h}^{(1)}$

- What happens if $\frac{\partial \boldsymbol{h}^{(i+1)}}{\partial \boldsymbol{h}^{(i)}}$ are **too small**? ⇨ **Vanishing gradient problem**
  - When these are small, the gradient signal gets smaller and smaller as it back-propagates further
  - So, model weight are updated only with respect to near effects, **not** long-term effects.

- What happens if $\frac{\partial \boldsymbol{h}^{(i+1)}}{\partial \boldsymbol{h}^{(i)}}$ are **too large**? ⇨ **Exploding gradient problem**

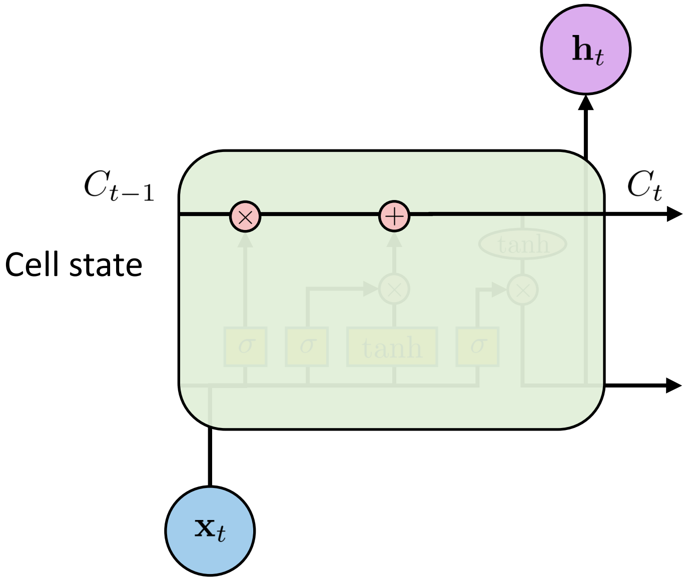
$$\theta^{\text{new}} = \theta^{\text{old}} - \alpha \nabla_\theta J(\theta)$$

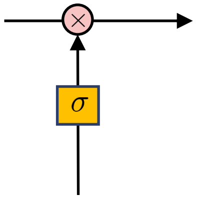
  - This can cause bad updates as the update step of parameters becomes too big
  - In the worst case, this will result in divergence of your network
  - In practice, with a gradient clipping, exploding gradient is **relatively easy to solve**

RNN Architectures: LSTM

- **Long Short-Term Memory (LSTM)** [Hochreiter and Schmidhuber, 1997]
  - A special type of RNN unit, i.e., LSTM networks = RNN composed of LSTM units
  - Explicitly designed RNN to
    - Capture **long-term dependency** ⟹ more robust to vanishing gradient problem

- Core idea behind LSTM
  - With **cell state** (**memory**), it controls **how much to remove or add information**
    - Only linear interactions from the output of each “**gates**” (**prevent vanishing gradient**)



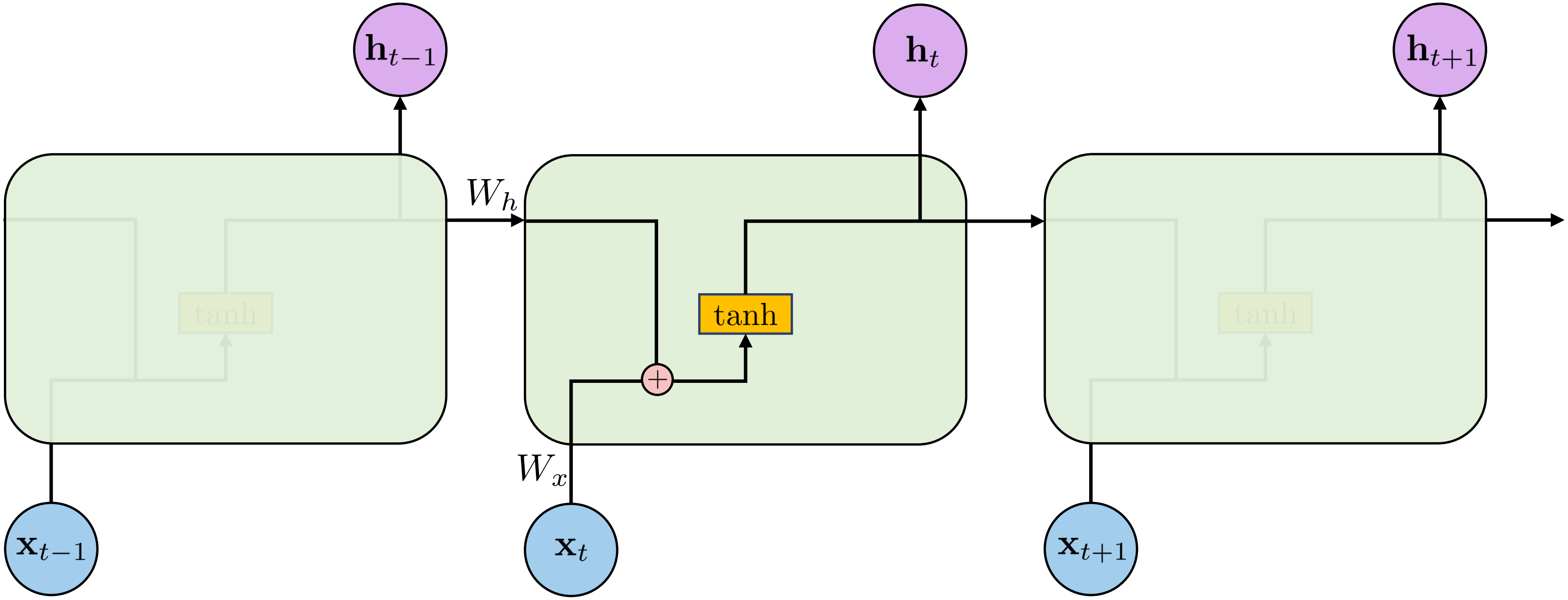


Gates : Way to optionally let information through

RNN Architectures: Vanilla RNN

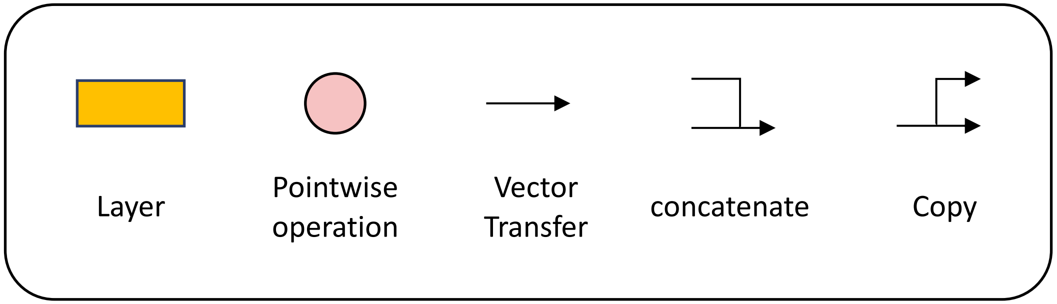
- Repeating modules in **Vanilla RNN** contains a **single layer**

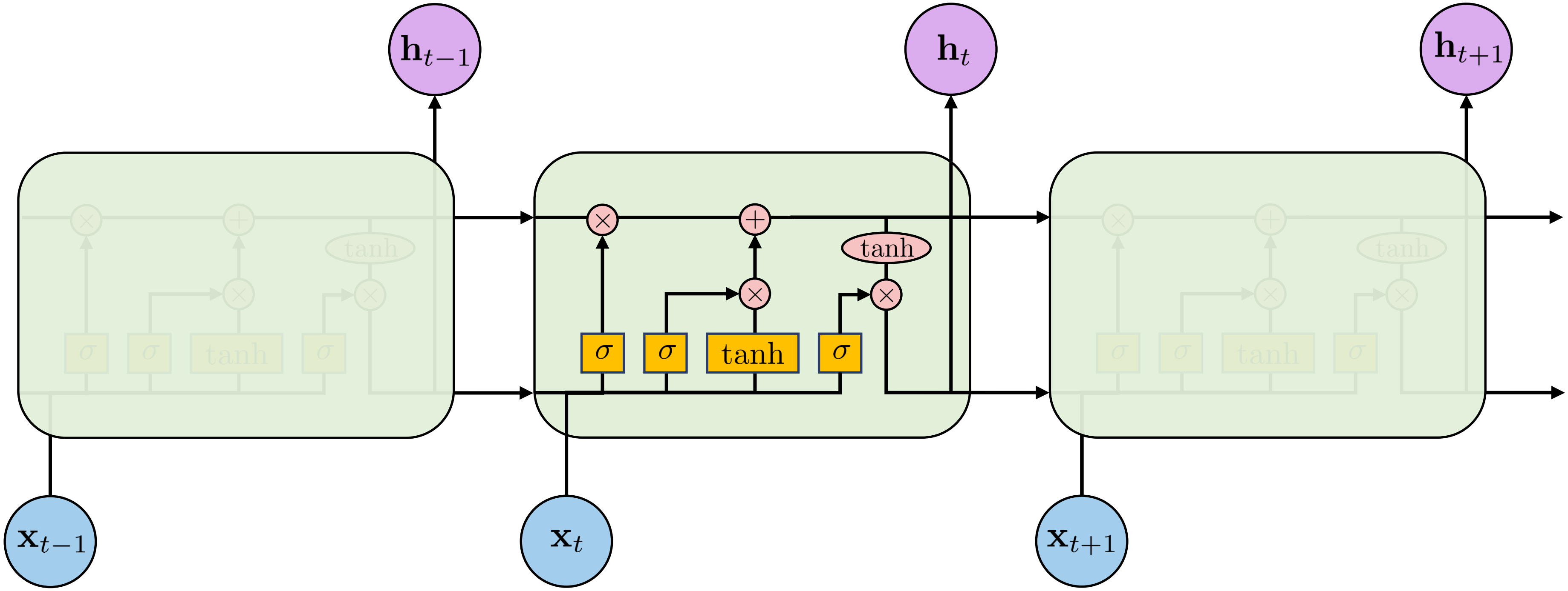
$$\boldsymbol{h}_t = \tanh(W_h \boldsymbol{h}_{t-1} + W_x \boldsymbol{x}_t)$$



RNN Architectures: LSTM

- Repeating modules in **LSTM**



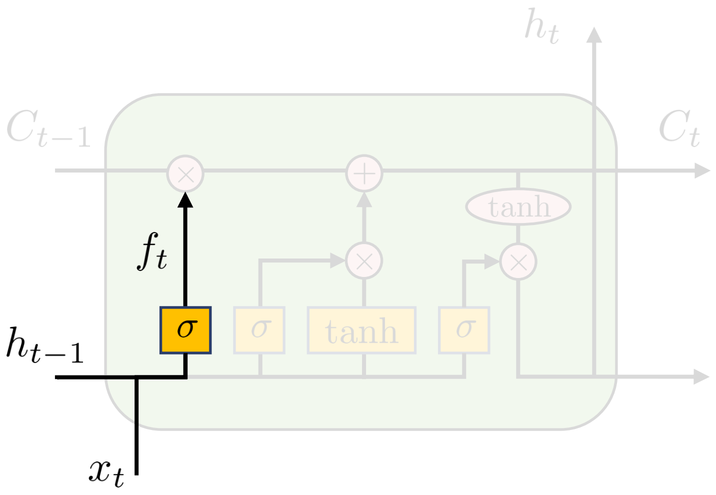


RNN Architectures: LSTM

Step 1: Decide what **information** we're going to **throw away** from the **cell state**

- A sigmoid layer called "**Forget gate**" f_t
- Looks at h_{t-1}, x_t and outputs a number between 0 and 1 for each cell state C_{t-1}
    - If 1: completely keep, if 0: completely remove

- E.g., language model trying to **predict the next word** based on all previous ones
    - The cell state might include the gender of the present subject so that the correct pronouns can be used
    - When we see a new subject, we want to forget the gender of the old subject

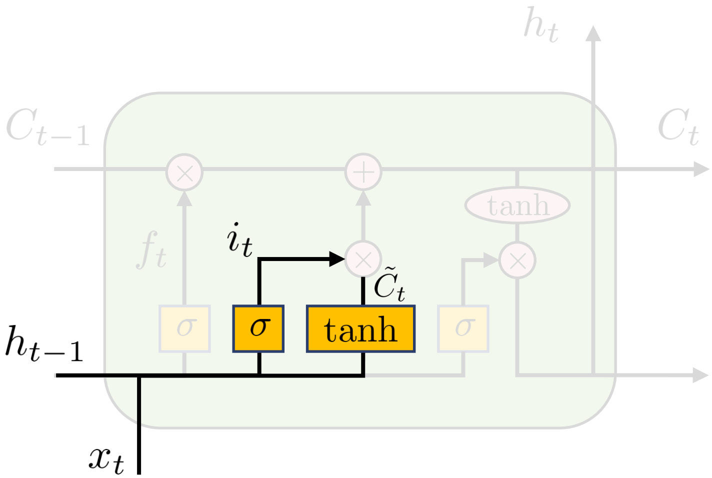


$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

RNN Architectures: LSTM

Step 2: Decide what **information** we're going to **store** in the cell state and **update**

- First, a sigmoid layer called the "**Input gate**" i_t decides which values to update
- Next, a tanh layer creates a **new content** $\tilde{C}_t$ to be written to the



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

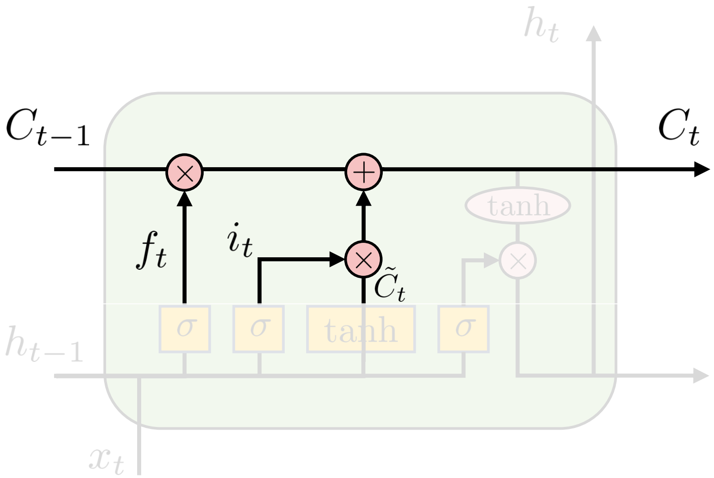
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

RNN Architectures: LSTM

Step 2: Decide what **information** we're going to **store** in the cell state and **update**

- First, a sigmoid layer called the "**Input gate**" i_t decides which values to update
- Next, a tanh layer creates a **new content** $\tilde{C}_t$ to be written to the

- Then, **update** the old cell state C_{t-1} into the **new cell state** C_t
  - Multiply the old state by f_t (forget gate)
  - Add $i_t * \tilde{C}_t$, new content scaled by how much to update (input gate)



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

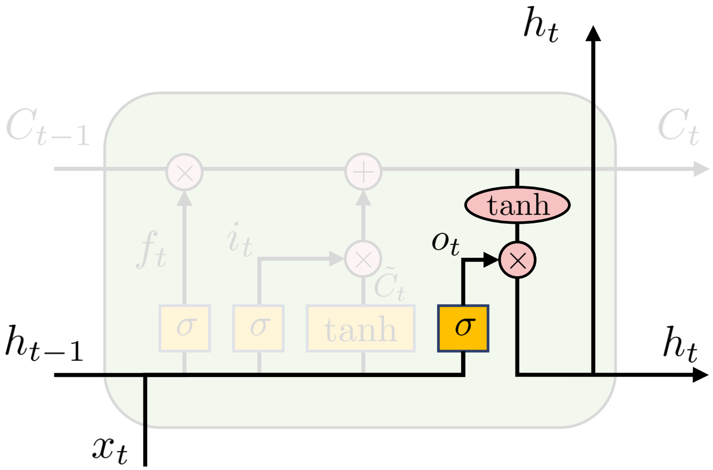
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$$\boxed{C_t = f_t * C_{t-1} + i_t * \tilde{C}_t}$$

RNN Architectures: LSTM

Step 3: Decide what **information** we're going to **output**

- A sigmoid layer called "**Output gate**" o_t
- First, go through o_t which decides **what parts** of the cell state **to output**
- Then, put the cell state C_t through tanh and multiply it by o_t for hidden state h_t



$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

RNN Architectures: LSTM

- Overall LSTM operations

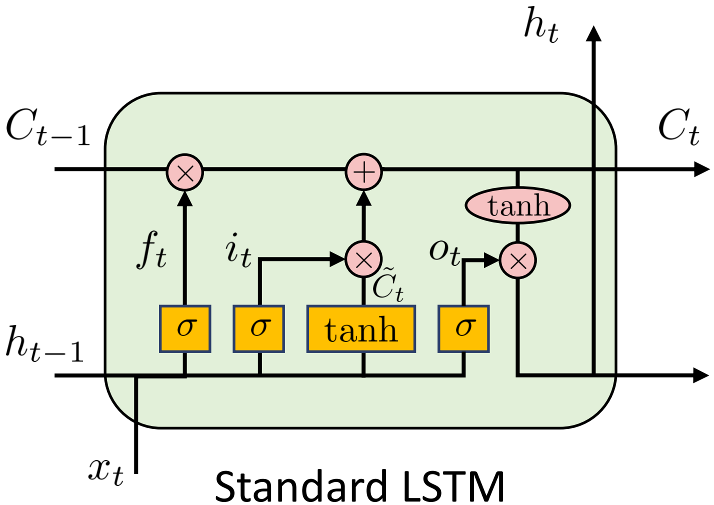
Forget gate: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$ Input gate: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

Previous cell state: C_{t-1} New cell content: $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$

Updated cell state: $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$

Output gate: $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$

Hidden state: $h_t = o_t * \tanh(C_t)$



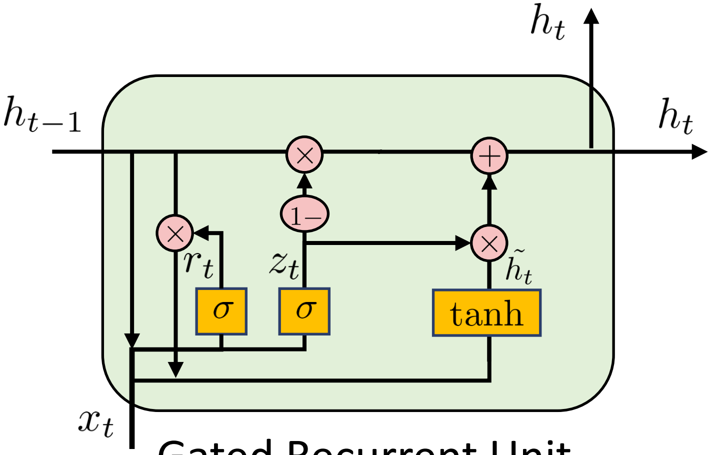
Standard LSTM

RNN Architectures: GRU

- **Gated Recurrent Unit (GRU)** [Cho et.al, 2014]
  - Combines the forget and input gates into a single "**update gate**" z_t
    - Controls the **ratio of information to keep** between previous state and new state
  - **Reset gate** r_t controls how much information to forget when create a new content
  - **Merges** the cell state C_t and hidden state h_t
  - **(+)** Resulting in **simpler model (less weights)** than standard LSTM

Reset gate: $r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$ New content: $\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$

Update gate: $z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$ Hidden state: $h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$



Gated Recurrent Unit

Overview

Part 1. Basics

- RNN to LSTM
- Sequence-to-sequence Model
- Attention-based NLP Model

Part 2. Transformers and Large Language Models

- Transformer (self-attention)
- Pre-training of Transformers and Language Models

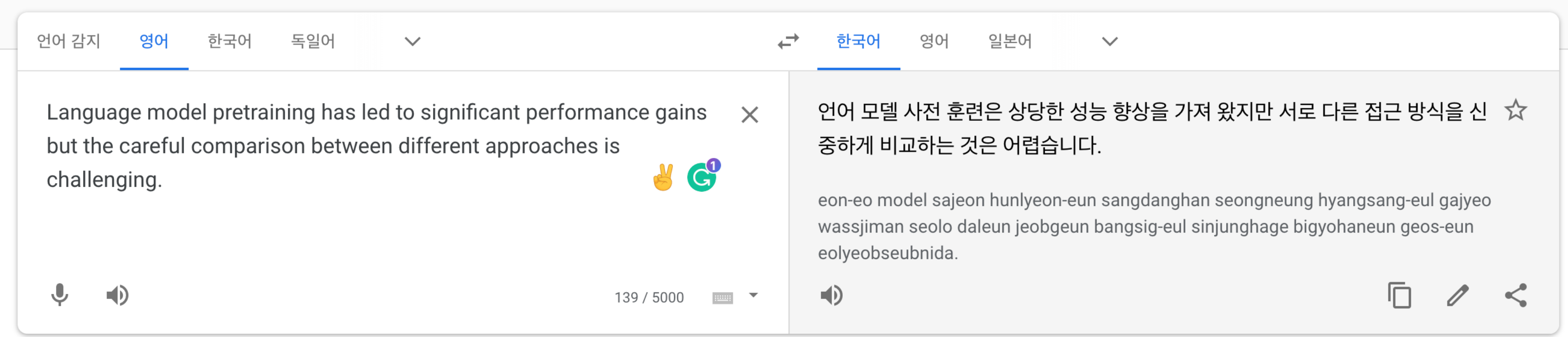
Part 3. Advanced Topics

- Handling long inputs with Transformers
- Techniques for improving efficiency
- State-Space Models

Part 4. Summary

Motivation: Natural Language Processing and Sequence-to-sequence Modeling

- Many natural language processing (NLP) tasks are **Sequence-to-sequence**
  - Given an input sequence, turn it into an output sequence
  - Example: **Translation**



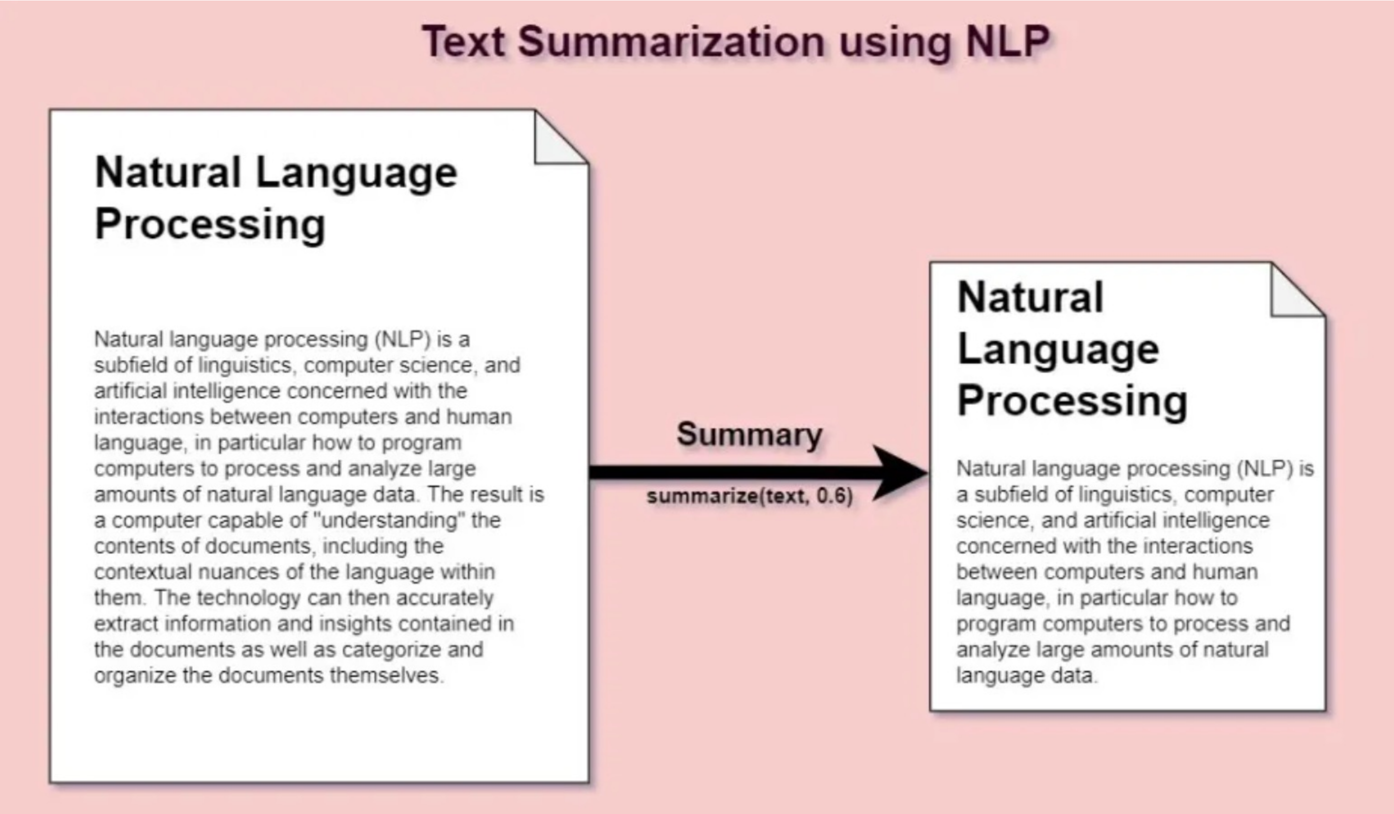
Language model pretraining has led to significant performance gains but the careful comparison between different approaches is challenging.

언어 모델 사전 훈련은 상당한 성능 향상을 가져 왔지만 서로 다른 접근 방식을 신중하게 비교하는 것은 어렵습니다.

eon-eo model sajeon hunlyeon-eun sangdanghan seongneung hyangsang-eul gajyeo wassjiman seolo daleun jeobgeun bangsig-eul sinjunghage bigyohaneun geos-eun eolyeobseubnida.

Motivation: Natural Language Processing and Sequence-to-sequence Modeling

- Many natural language processing (NLP) tasks are **Sequence-to-sequence**
    - Given an input sequence, turn it into an output sequence
    - Example: **Text Summarization**



Natural Language Processing

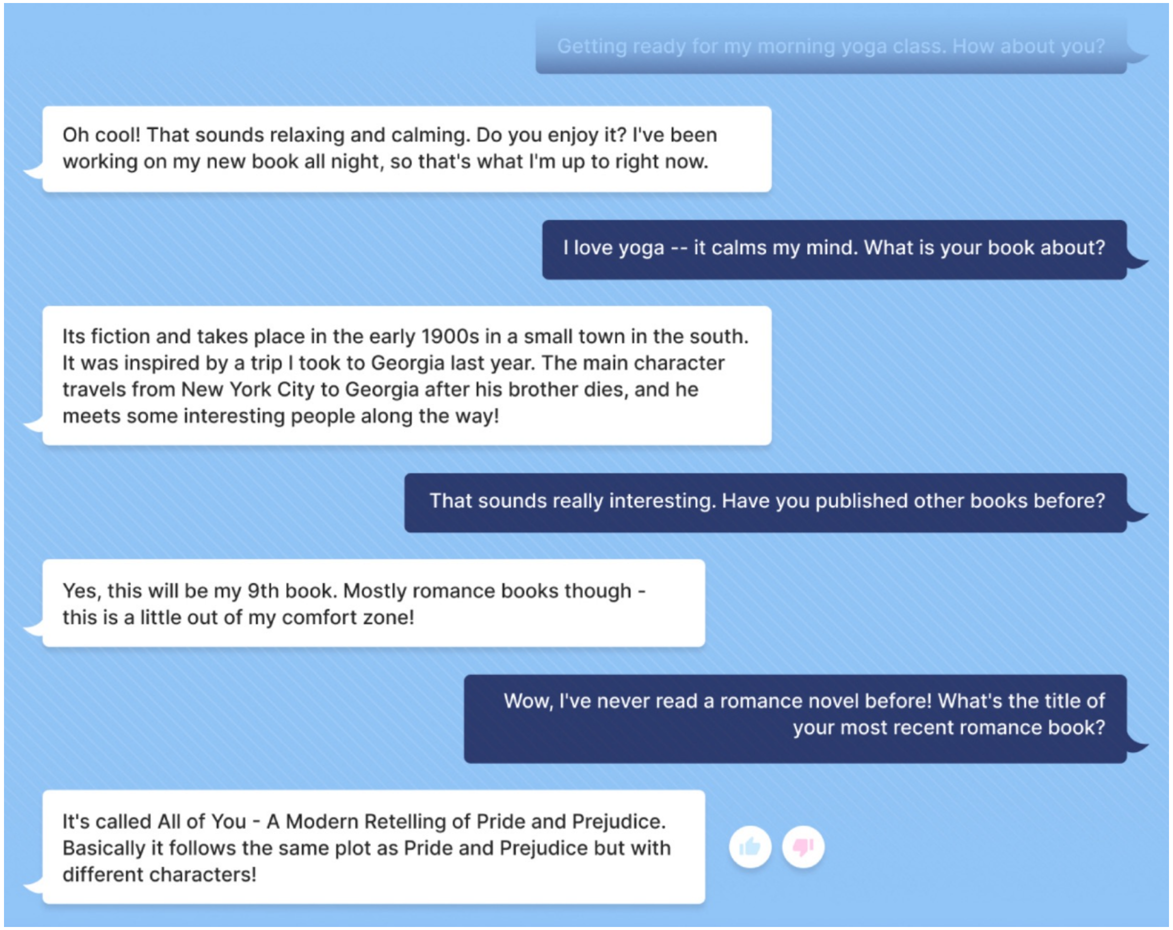
Natural language processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data. The result is a computer capable of "understanding" the contents of documents, including the contextual nuances of the language within them. The technology can then accurately extract information and insights contained in the documents as well as categorize and organize the documents themselves.

Natural Language Processing

Natural language processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data.

Motivation: Natural Language Processing and Sequence-to-sequence Modeling

- Many natural language processing (NLP) tasks are **Sequence-to-sequence**
  - Given an input sequence, turn it into an output sequence
  - Example: **ChatBot**



Getting ready for my morning yoga class. How about you?

Oh cool! That sounds relaxing and calming. Do you enjoy it? I've been working on my new book all night, so that's what I'm up to right now.

I love yoga -- it calms my mind. What is your book about?

Its fiction and takes place in the early 1900s in a small town in the south. It was inspired by a trip I took to Georgia last year. The main character travels from New York City to Georgia after his brother dies, and he meets some interesting people along the way!

That sounds really interesting. Have you published other books before?

Yes, this will be my 9th book. Mostly romance books though - this is a little out of my comfort zone!

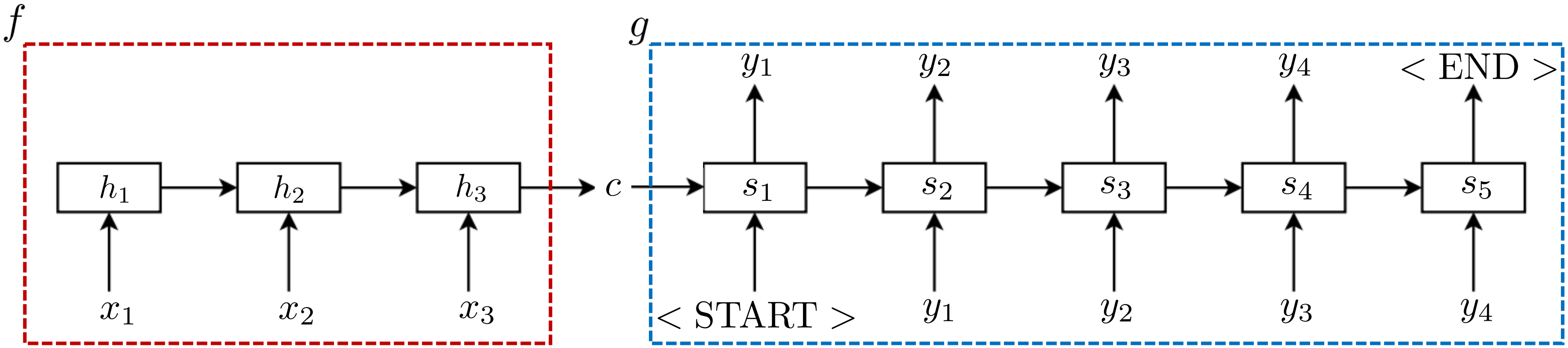
Wow, I've never read a romance novel before! What's the title of your most recent romance book?

It's called All of You - A Modern Retelling of Pride and Prejudice. Basically it follows the same plot as Pride and Prejudice but with different characters!

Sequence-to-sequence Model

- Many natural language processing (NLP) tasks are **Sequence-to-sequence**
  - Given an input sequence, turn it into an output sequence

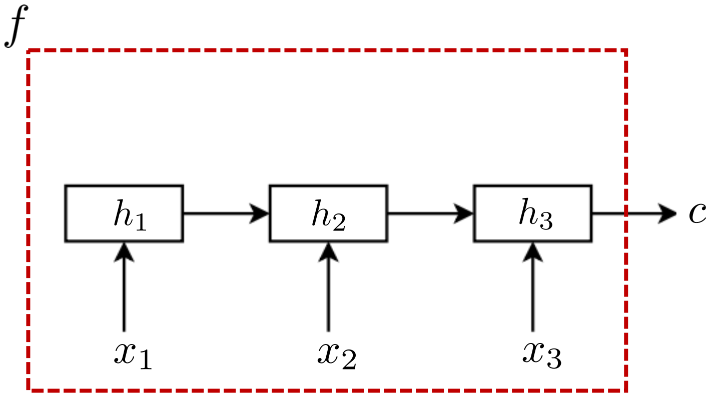
- The core idea of **Sequence-to-sequence** model [Sutskever et al., 2014]
  - **Encoder-Decoder** architecture (input → vector → output)
  - Use one network (**Encoder**) to **read input sequence** at a time for **encoding it into** a fixed-length vector representation **(context)**
  - Use another network (**Decoder**) to extract **output sequence** from context vector



Input sequence $\boldsymbol{x} = (x_1, x_2, x_3)$ and output sequence $\boldsymbol{y} = (y_1, y_2, y_3, y_4)$

Sequence-to-sequence Model

- **Encoder**
  - **Reads the input** sentence $\mathbf{x} = (x_1, \ldots, x_T)$ and **output context** vector c
  - Use RNNs such that $h_t = f(x_t, h_{t-1})$ and $c = q(\{h_1, \ldots, h_T\})$, where f and q are some non-linear functions
  - E.g., LSTMs as f and $q(\{h_1, \ldots, h_T\}) = h_T$ (in the original seq2seq model)



Input sequence $\boldsymbol{x} = (x_1, x_2, x_3)$ and output sequence $\boldsymbol{y} = (y_1, y_2, y_3, y_4)$

Sequence-to-sequence Model

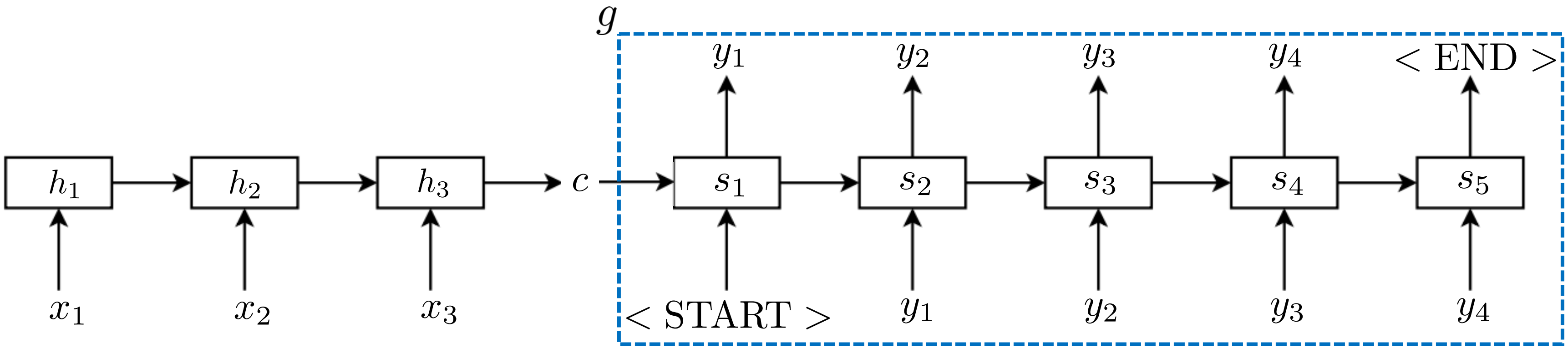
- **Decoder**
  - **Predict the next word** $y_{t'}$ **given the context** vector c and the **previously predicted** words $\{y_1, \dots, y_{t'-1}\}$
  - Defines a probability over the translation $\mathbf{y}$ by **decomposing the joint probability** into the ordered conditionals where $\mathbf{y} = (y_1, \dots, y_T)$.

$$p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t | \{y_1, \dots, y_{t'-1}\}, c),$$

  - The conditional probability is modeled with **another RNN** g as

$$p(y_t | \{y_1, \dots, y_{t'-1}\}, c) = g(y_{t-1}, \underline{s_t}, c),$$

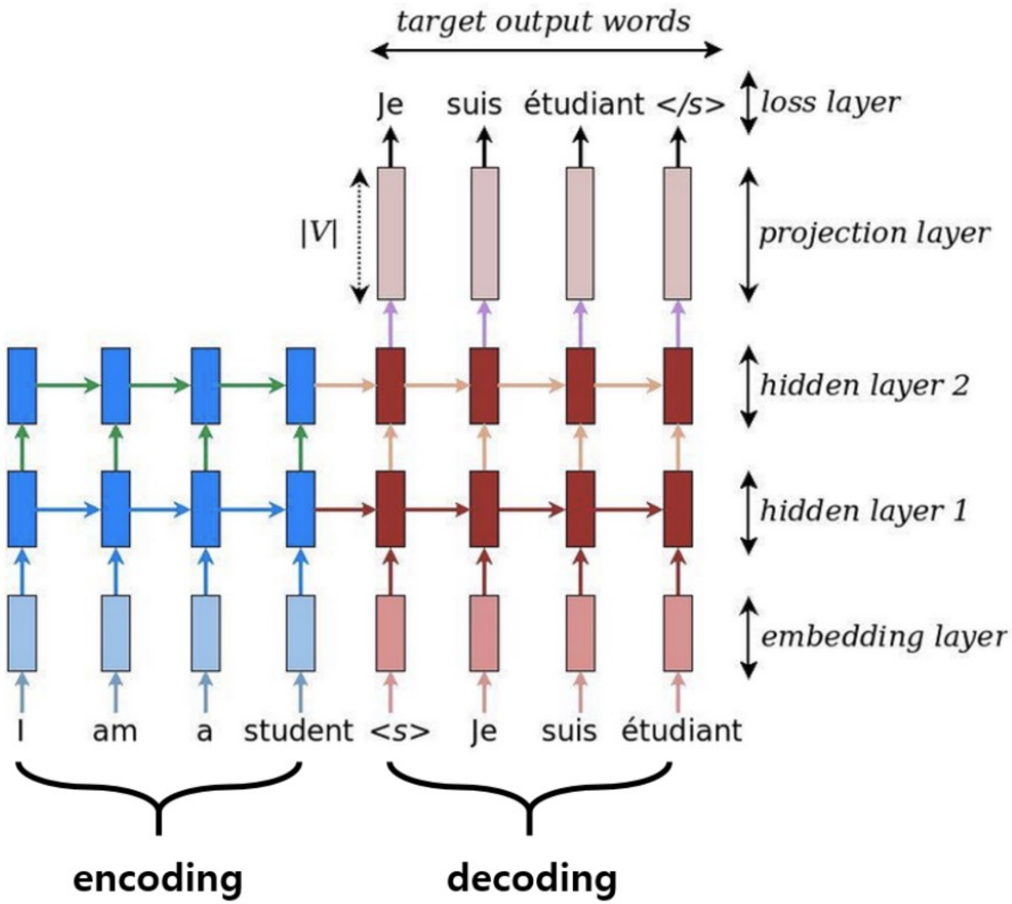
hidden state of the RNN



Input sequence $\boldsymbol{x} = (x_1, x_2, x_3)$ and output sequence $\boldsymbol{y} = (y_1, y_2, y_3, y_4)$

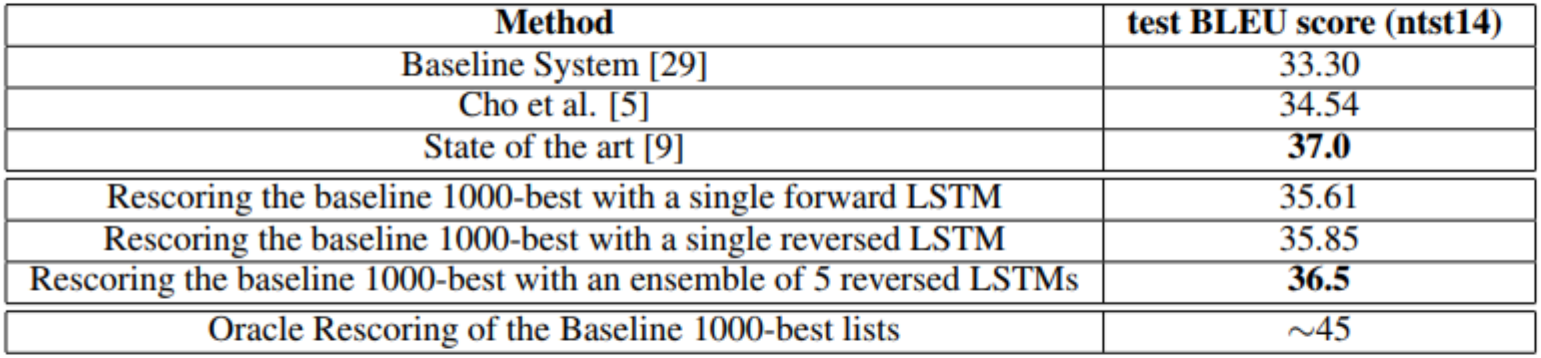
Sequence-to-sequence Model

- Example of the seq2seq model
  - For English → French task
  - With 2-layer LSTM for encoder and encoder



Sequence-to-sequence Model

- Results on WMT’14 English to French dataset [Sutskever et al., 2014]
  - Measure : BLEU (Bilingual Evaluation Understudy) score
    - Widely used quantitative measure for MT task
  - On **par with the state-of-the-art** system (without using neural network)
  - Achieved **better results than the previous baselines**



Method	**test BLEU score (ntst14)**
Baseline System [29]	33.30
Cho et al. [5]	34.54
State of the art [9]	**37.0**
Rescoring the baseline 1000-best with a single forward LSTM	35.61
Rescoring the baseline 1000-best with a single reversed LSTM	35.85
Rescoring the baseline 1000-best with an ensemble of 5 reversed LSTMs	**36.5**
Oracle Rescoring of the Baseline 1000-best lists	~45

- Seq2seq with RNNs is **simple but very powerful** in MT task

Overview

Part 1. Basics

- RNN to LSTM
- Sequence-to-sequence Model
- Attention-based NLP Model

Part 2. Transformers and Large Language Models

- Transformer (self-attention)
- Pre-training of Transformers and Language Models

Part 3. Advanced Topics

- Handling long inputs with Transformers
- Techniques for improving efficiency
- State-Space Models

Part 4. Summary

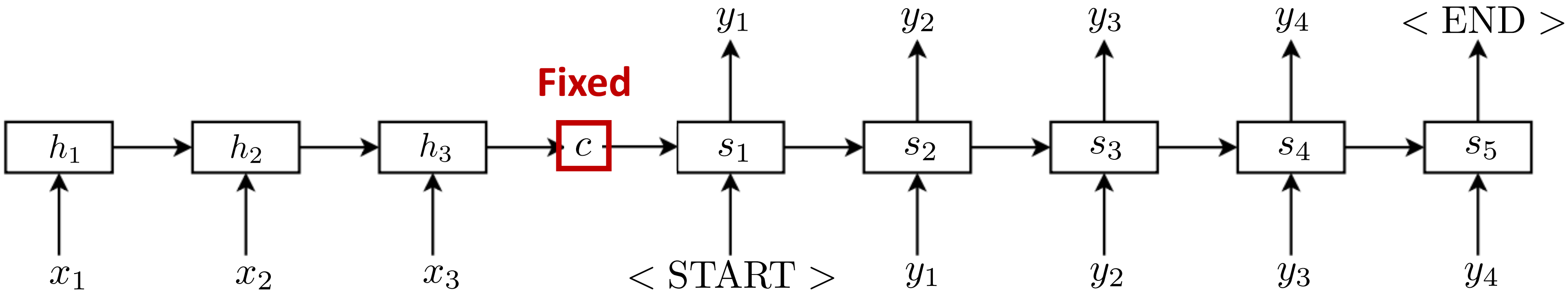
Attention-based Sequence-to-sequence Model

- Problem of original seq2seq (or encoder-decoder) model
  - Need to **compress** all the necessary information of a source sentence into a **fixed context vector**
    - All decoding steps use an identical context along with previous outputs

$$p(y_t|\{y_1,\ldots,y_{t'-1}\},c) = g(y_{t-1},s_t,\underline{c}),$$

  - But, each step of decoding **requires different part** of the source sequence
    - E.g., Step1: “**I** love you” ⟶ “**나는** 너를 사랑해”

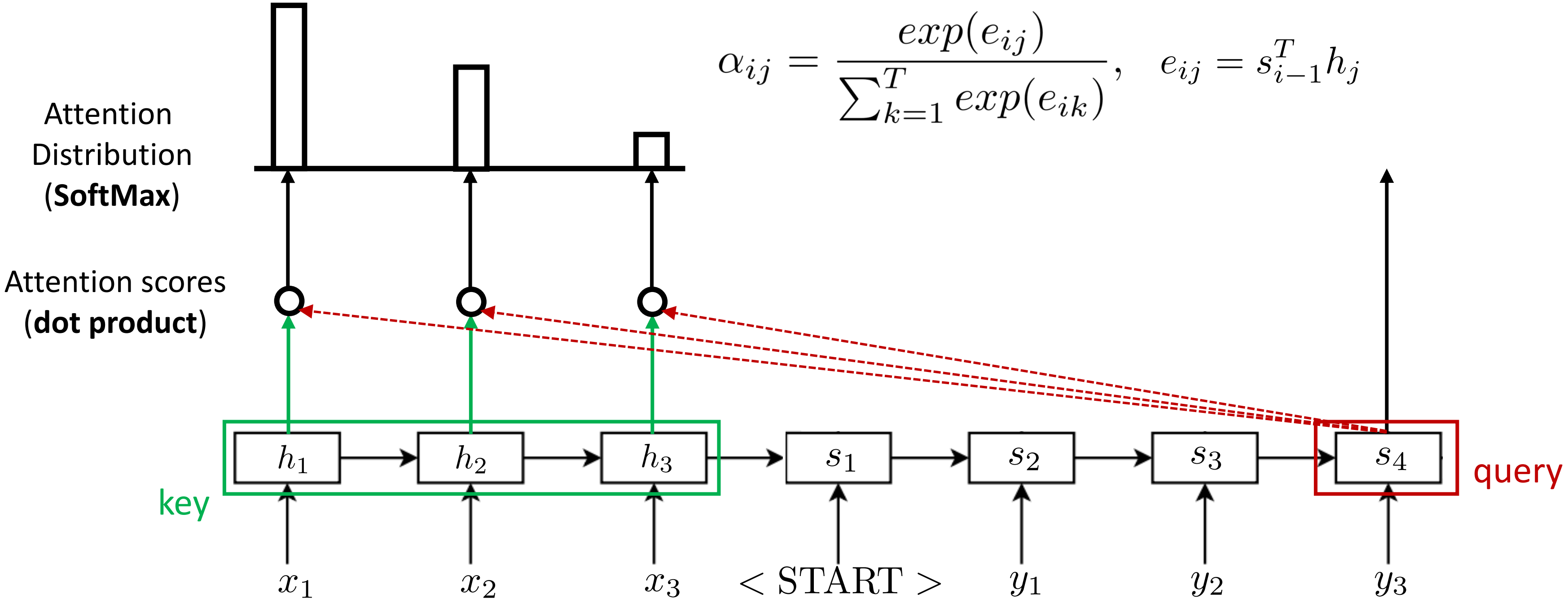
      Step2: “I **love** you” ⟶ “나는 너를 **사랑해**”
    - Hence, difficult to cope with long sentences…



Input sequence $\boldsymbol{x} = (x_1, x_2, x_3)$ and output sequence $\boldsymbol{y} = (y_1, y_2, y_3, y_4)$

Attention-based Sequence-to-sequence Model

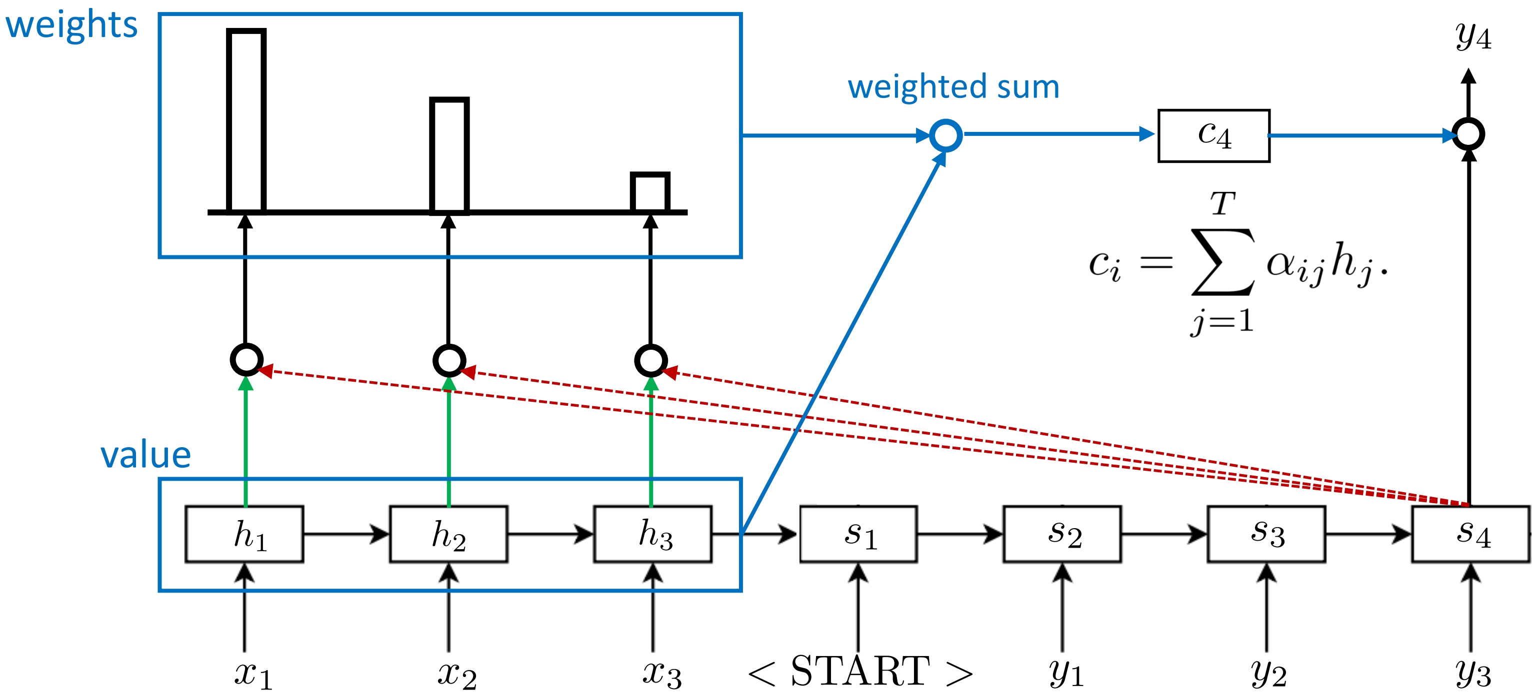
- Extension of seq2seq model with **attention** mechanism [Bahdanau et al., 2015]
  - **Core idea**: on each step of the decoder, **focus on a particular part** of the source sequence using a **direct connection (attention)** to the encoder states
  - Dependent on the query with key, **attention** is a technique to compute a weighted sum of the values
    - Query: decoder's hidden state, key and value: encoder's hidden states
    - α_{ij} is a **relative importance** which means how well the inputs around position i and the output position j match.



$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T} exp(e_{ik})}, \quad e_{ij} = s_{i-1}^{T} h_j$$

Attention-based Sequence-to-sequence Model

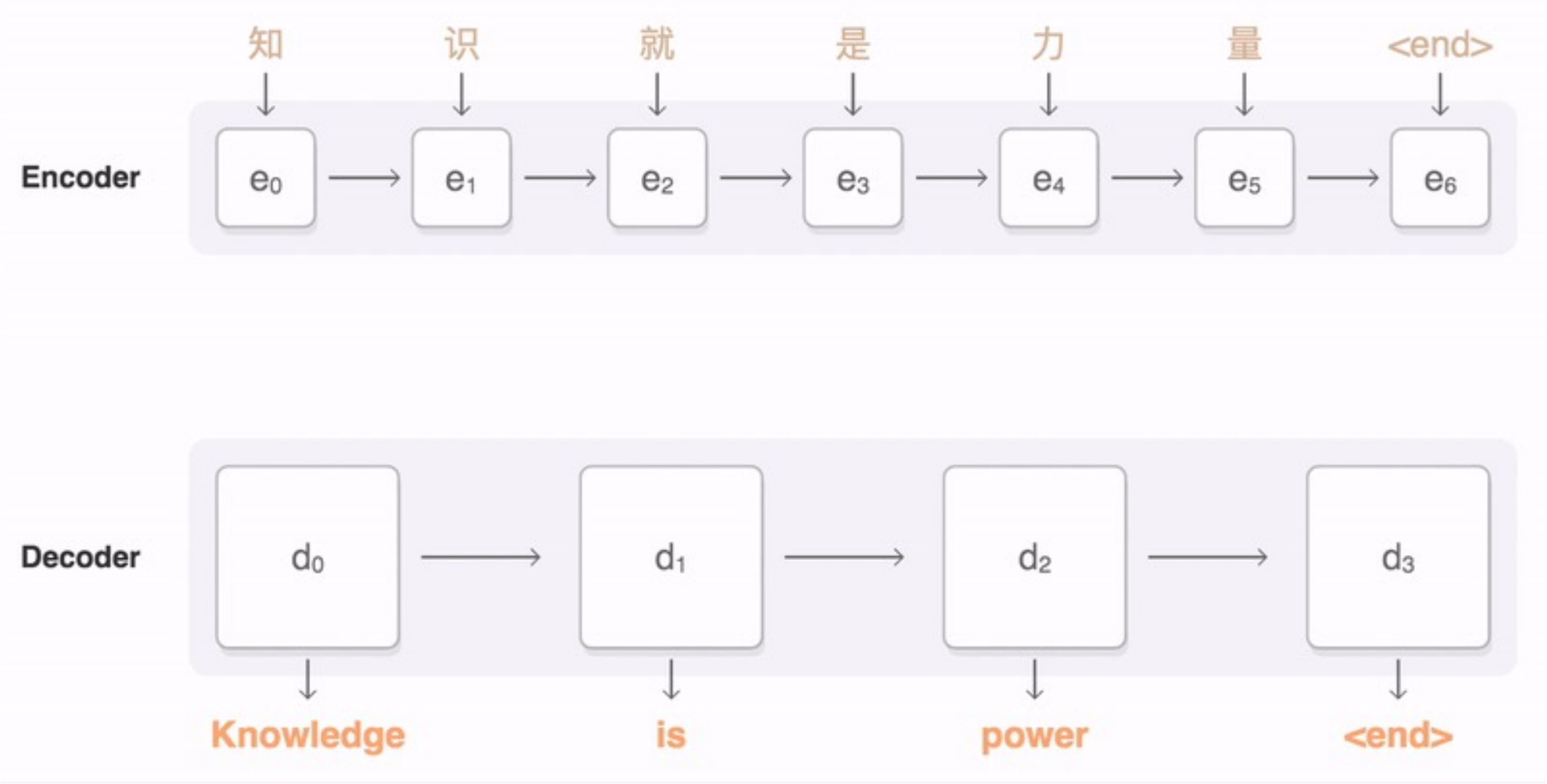
- Extension of seq2seq model with **attention** mechanism [Bahdanau et al., 2015]
  - **Core idea**: on each step of the decoder, **focus on a particular part** of the source sequence using a **direct connection (attention)** to the encoder states
  - Dependent on the query with key, **attention** is a technique to compute a weighted sum of the values
    - Query: decoder's hidden state, key and value: encoder's hidden states
    - The context vector c_i is computed as **weighted sum** of h_i



$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j.$$

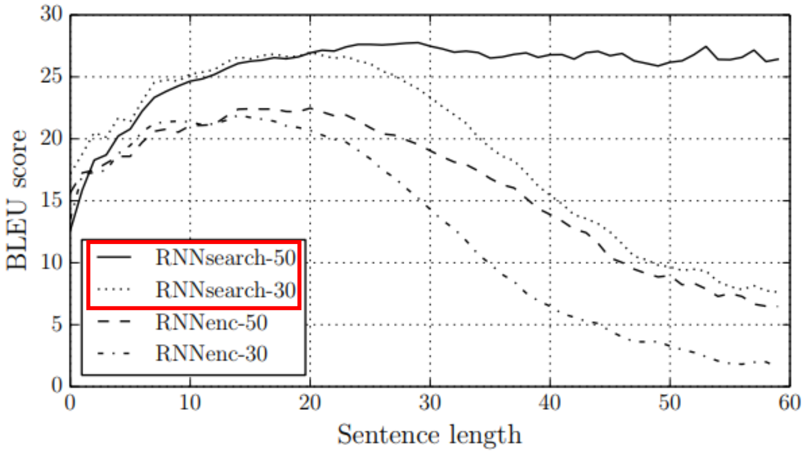
Attention-based Sequence-to-sequence Model

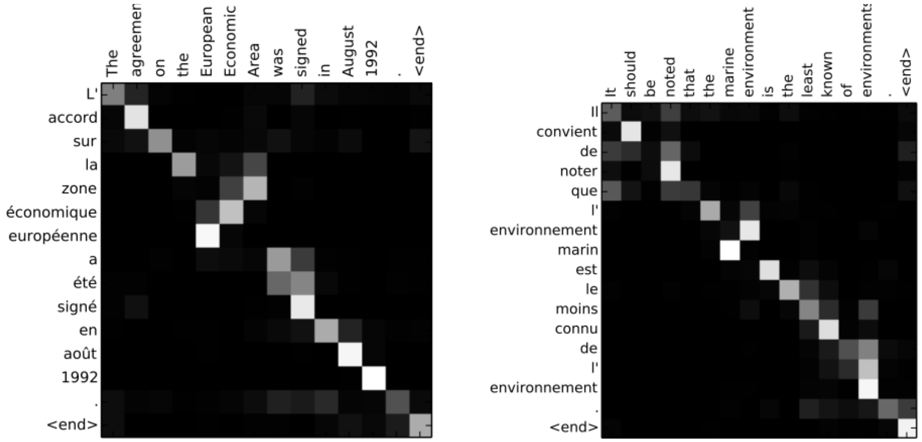
- **Graphical illustration** of seq2seq with **attention**
    - E.g., Chinese to English



Attention-based Sequence-to-sequence Model

- **Results**
  - RNNsearch (with attention) is better than RNNenc (vanilla seq2seq)
  - RNNsearch-50: model trained with sentences of length up to 50 words





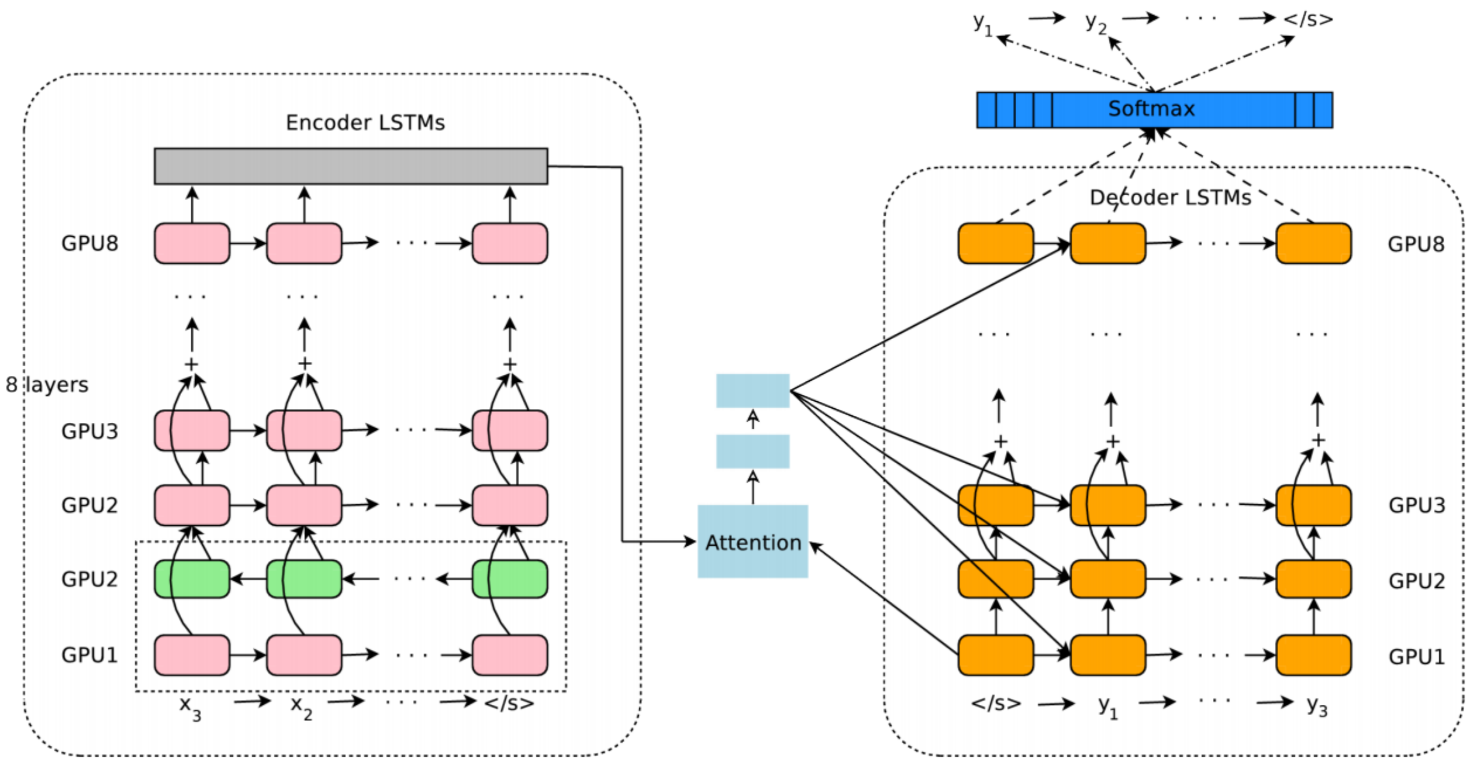
(a)

(b)

Sample alignment results (attention map)

Attention-based Sequence-to-sequence Model: Google's NMT

- **Google's NMT** [Wu et al., 2016]
  - Improves over previous NMT systems on **accuracy** and **speed**
  - **8-layer LSTMS** for encoder/decoder with **attention**
    - Achieve **model parallelism** by assigning each LSTM layer into different GPUs
    - Add **residual connections in standard LSTM**
    - … and lots of domain-specific details to apply it to production model



Attention-based Sequence-to-sequence Model: Google's NMT

- **Google's NMT** [Wu et al., 2016]
  - Improves over previous NMT systems on **accuracy** and **speed**
  - **8-layer LSTMS** for encoder/decoder with **attention**
  - State-of-the-art results on various MT datasets and comparable with Human expert

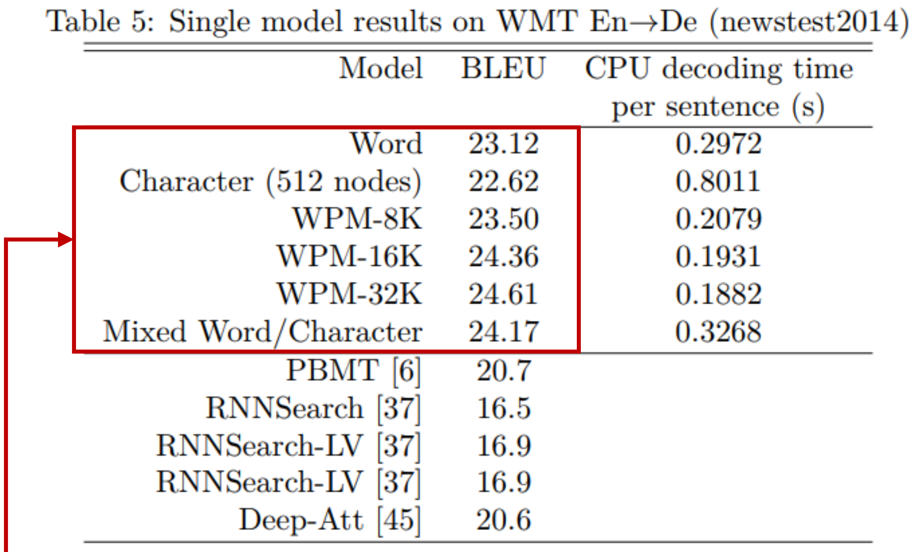


Table 5: Single model results on WMT En→De (newstest2014)

Model	BLEU	CPU decoding time per sentence (s)
Word	23.12	0.2972
Character (512 nodes)	22.62	0.8011
WPM-8K	23.50	0.2079
WPM-16K	24.36	0.1931
WPM-32K	24.61	0.1882
Mixed Word/Character	24.17	0.3268
PBMT [6]	20.7	
RNNSearch [37]	16.5	
RNNSearch-LV [37]	16.9	
RNNSearch-LV [37]	16.9	
Deep-Att [45]	20.6	

GNMT with different configurations

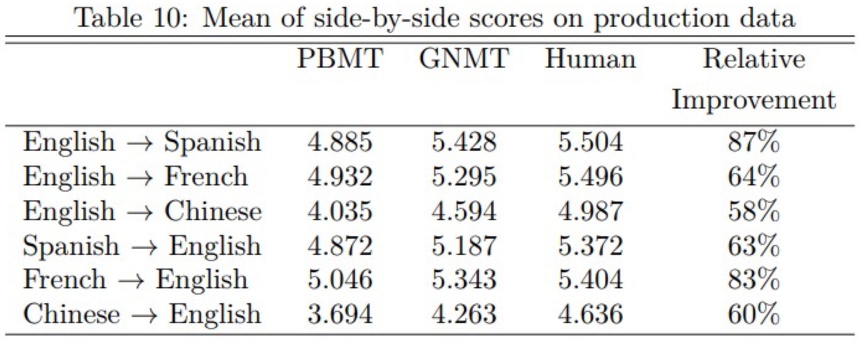
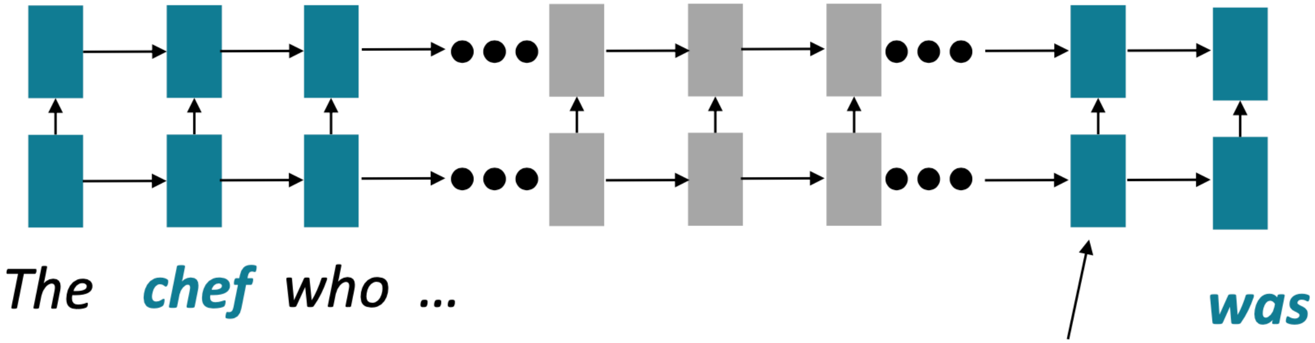


Table 10: Mean of side-by-side scores on production data

	PBMT	GNMT	Human	Relative Improvement
English → Spanish	4.885	5.428	5.504	87%
English → French	4.932	5.295	5.496	64%
English → Chinese	4.035	4.594	4.987	58%
Spanish → English	4.872	5.187	5.372	63%
French → English	5.046	5.343	5.404	83%
Chinese → English	3.694	4.263	4.636	60%

Limitations with Recurrent Models

- Although RNNs show remarkable successes, there are **fundamental issues**:
  1. **O(sequence length)** steps for distant word pairs to interact means
     - Hard to learn long-distance dependencies because of gradient problems
  2. Forward/backward passes have **O(sequence length)** unparallelizable operations
     - Future RNN hidden states can't be computed before past states have been computed
     - This aspect inhibits training on the very large datasets

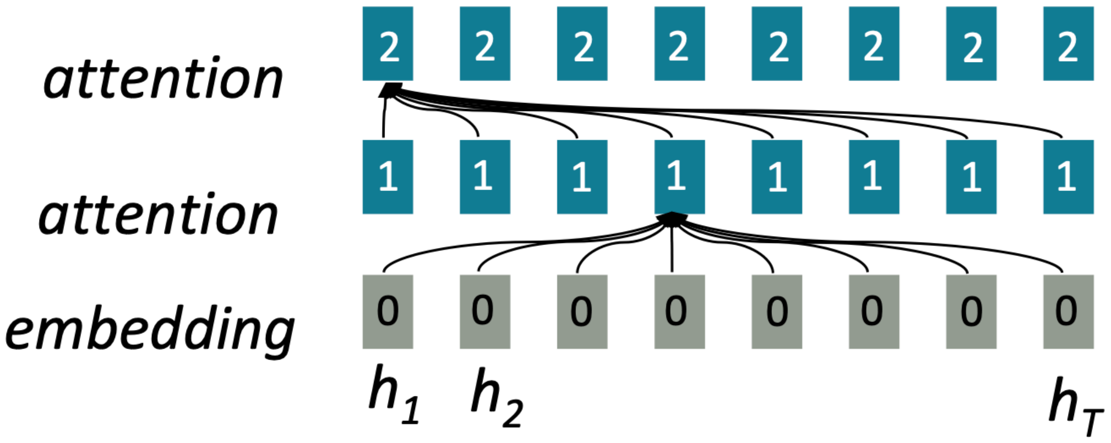


Info of chef has gone through **O(sequence length)** many layers

Limitations with Recurrent Models

- Although RNNs show remarkable successes, there are **fundamental issues**:
  1. **O(sequence length)** steps for distant word pairs to interact means
  2. Forward/backward passes have **O(sequence length)** unparallelizable operations

- In contrast, **attention has some advantages** in these aspects:
  1. Maximum interaction distance: **O(1)**
     - Since all words interact at each layer
  2. Number of unparallelizable operations does **not increase with respect to length**



All words can attend to all words in previous layer

Limitations with Recurrent Models

- Although RNNs show remarkable successes, there are **fundamental issues**:
  1. **O(sequence length)** steps for distant word pairs to interact means
  2. Forward/backward passes have **O(sequence length)** unparallelizable operations

- In contrast, **attention has some advantages** in these aspects:
  1. Maximum interaction distance: **O(1)**
     - Since all words interact at each layer
  2. Number of unparallelizable operations does **not increase with respect to length**

Q. Then, can we design an architecture **only using attention** modules?

- Remark. We saw attention from the **decoder to the encoder**;
  but here, we'll think about attention **within a single sentence**.

Overview

Part 1. Basics

- RNN to LSTM
- Sequence-to-sequence Model
- Attention-based NLP Model

Part 2. Transformers and Large Language Models

- Transformer (self-attention)
- Pre-training of Transformers and Language Models

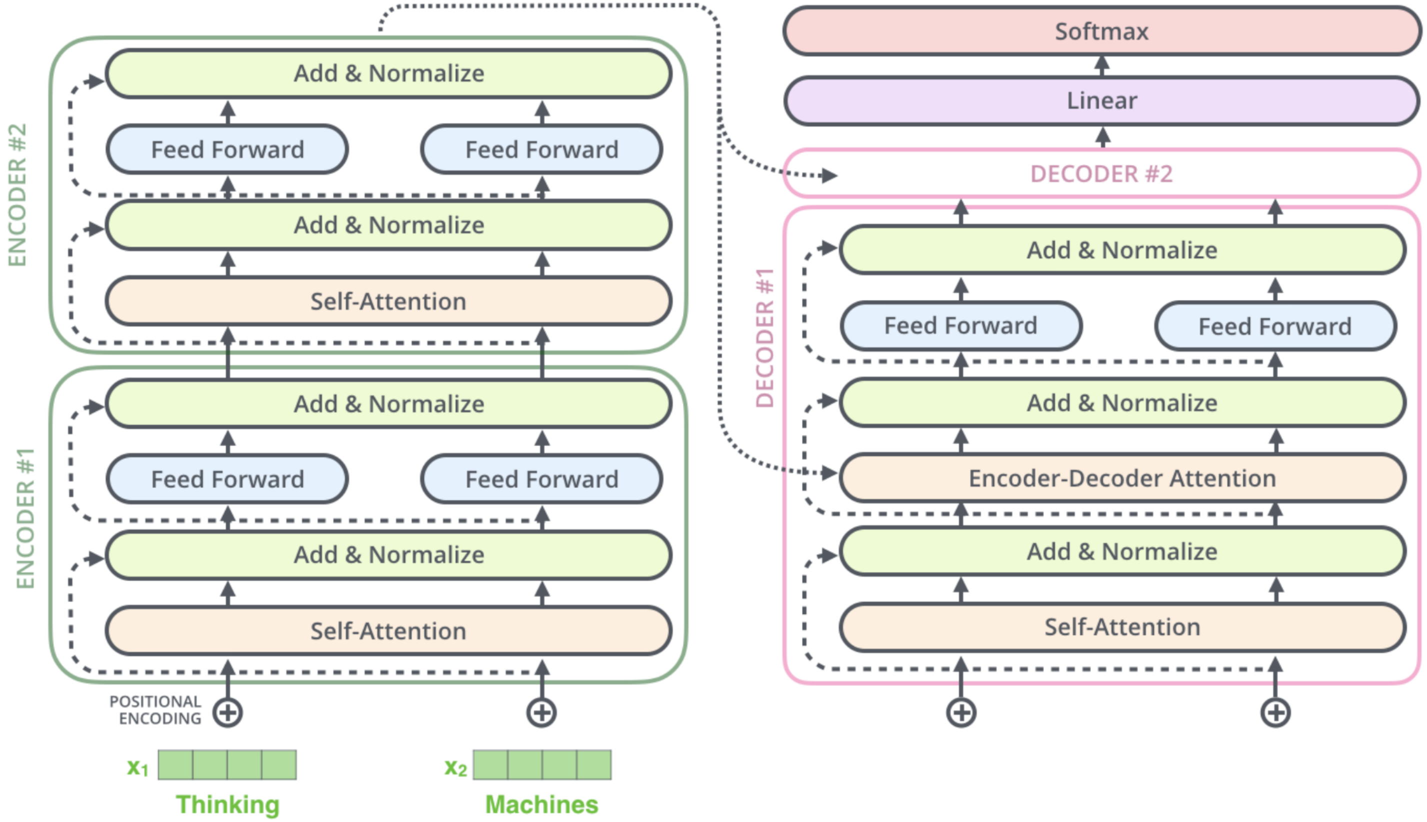
Part 3. Advanced Topics

- Handling long inputs with Transformers
- Techniques for improving efficiency
- State-Space Models

Part 4. Summary

Transformer (Self-attention)

- Transformer [Vaswani et al., 2017] has an **encoder-decoder** structure and they are composed of multiple block with **multi-head (self) attention** module

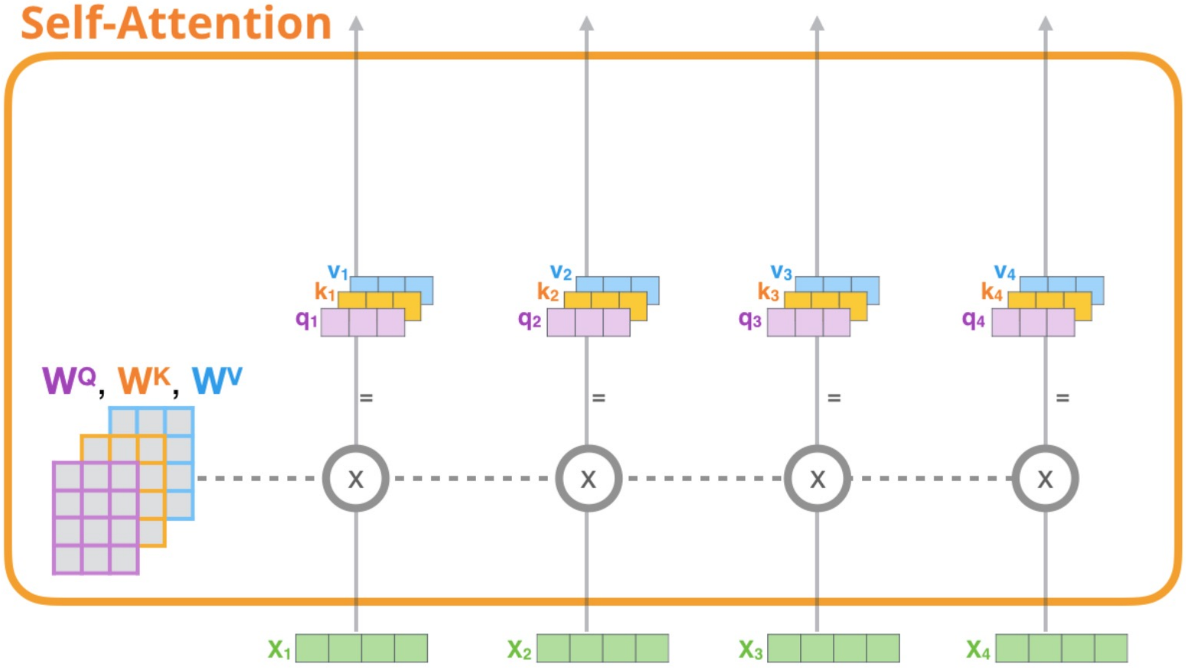


Transformer (Self-attention)

- **Self-attention**
  - **Recall**: Attention operates on query, key, and value
    - Query is decoder's hidden state, key and value are encoder's hidden states in seq2seq
  - In self-attention, the query, key, and value are drawn from the same **same source**

1. For each input x_i , create query, key, and value vectors q_i, k_i, v_i by multiplying **learnable** weight matrices

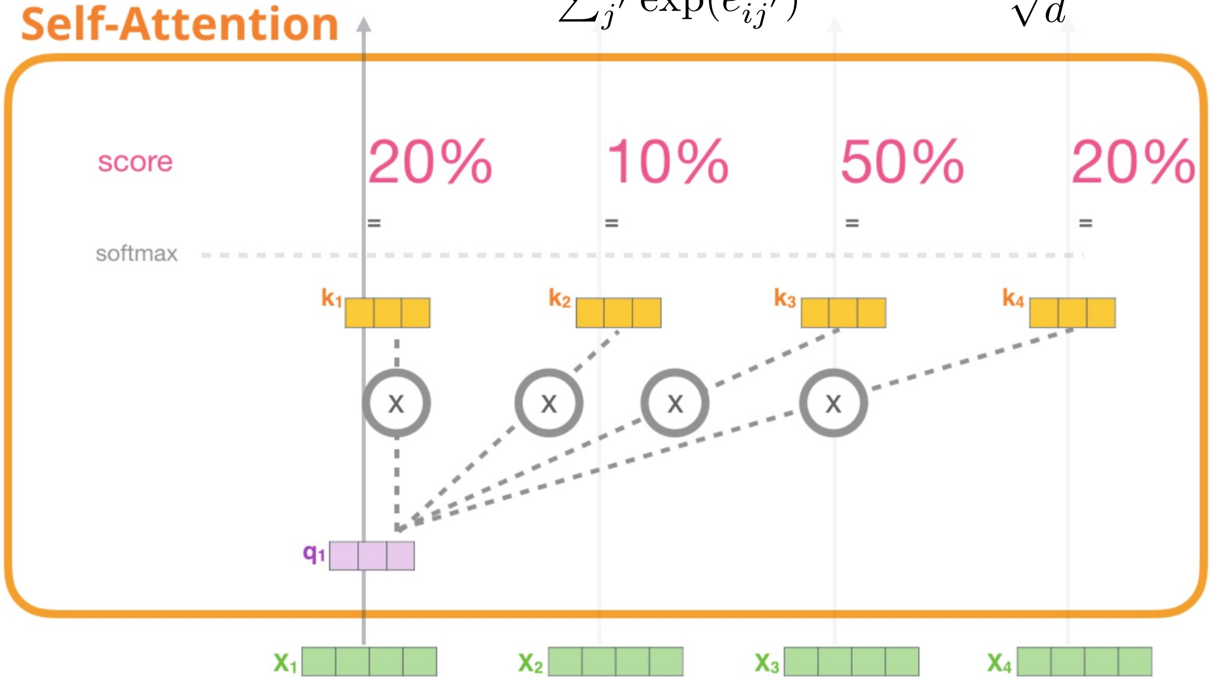
$$q_i = W^Q x_i, k_i = W^k x_i, v_i = W^V x_i$$



Transformer (Self-attention)

- **Self-attention**
  - **Recall**: Attention operates on query, key, and value
    - Query is decoder's hidden state, key and value are encoder's hidden states in seq2seq
  - In self-attention, the query, key, and value are drawn from the same **same source**
    1. For each input x_i, create query, key, and value vectors q_i, k_i, v_i
    2. Multiply (**dot product**) the current query vector, by all the key vectors, to get a **score** α_{ij} of **how well they match**

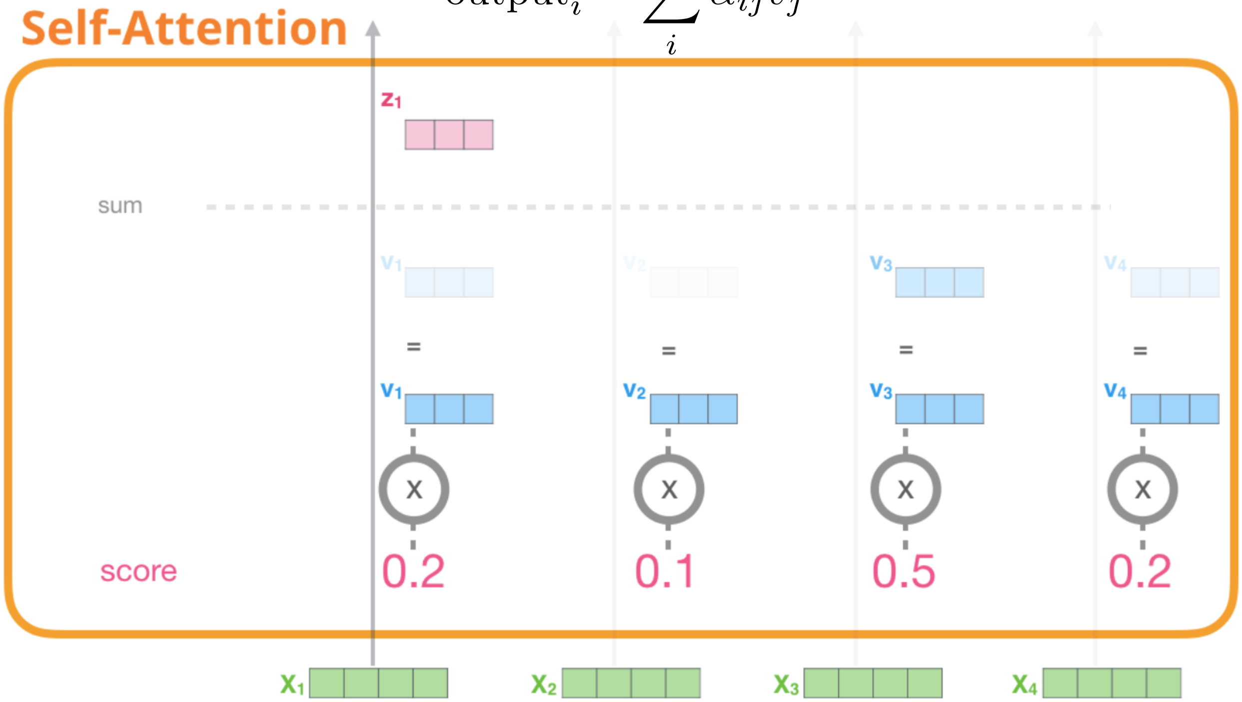
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})} \qquad e_{ij} = \frac{q_i^T k_j}{\sqrt{d}}$$



Transformer (Self-attention)

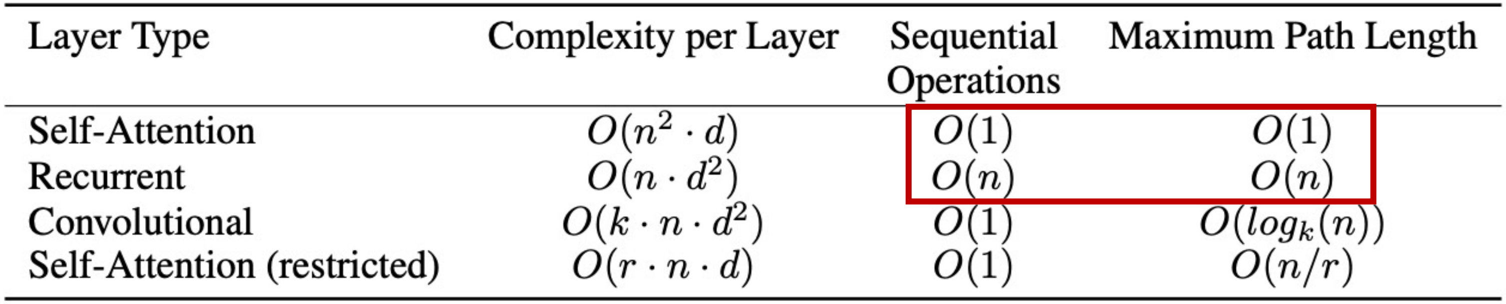
- **Self-attention**
  - **Recall**: Attention operates on query, key, and value
    - Query is decoder's hidden state, key and value are encoder's hidden states in seq2seq
  - In self-attention, the query, key, and value are drawn from the same source
    1. For each input x_i, create query, key, and value vectors q_i, k_i, v_i
    2. Multiply (**dot product**) the current query vector, by all the key vectors, to get a **score** α_{ij}
    3. Multiply the value vectors by the scores, then **sum up**

$$\text{output}_i = \sum_i \alpha_{ij} v_j$$



Transformer (Self-attention)

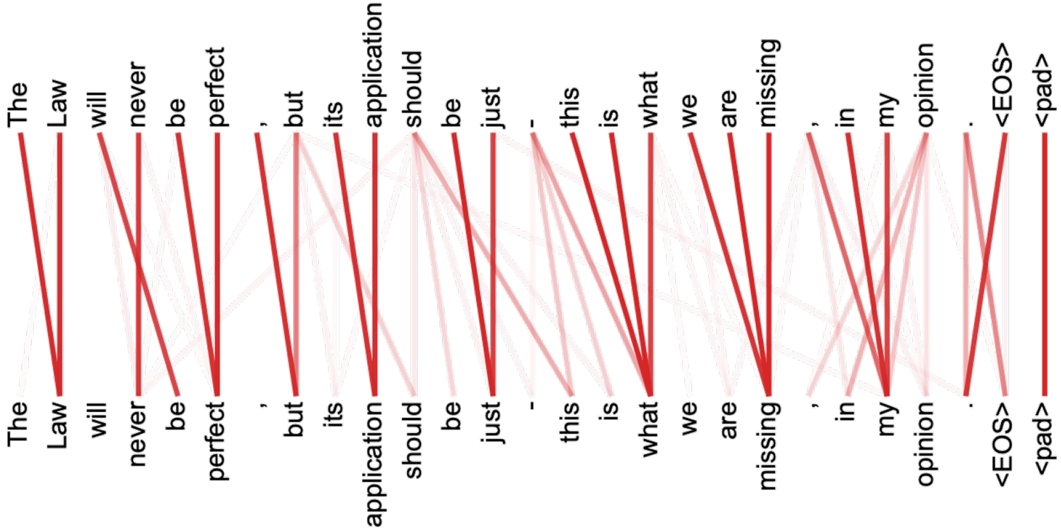
- **Self-attention**
  - **Recall**: Attention operates on query, key, and value
    - Query is decoder's hidden state, key and value are encoder's hidden states in seq2seq
  - In self-attention, the query, key, and value are drawn from the **same source**
    1. For each input x_i , create query, key, and value vectors q_i, k_i, v_i
    2. Multiply (**dot product**) the current query vector, by all the key vectors, to get a **score** α_{ij}
    3. Multiply the value vectors by the scores, then **sum up**
  - Hence, self-attention is **effective to learn the context** within given sentence
    - It's easier than recurrent layer to be parallelized and model the long-term dependency



Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

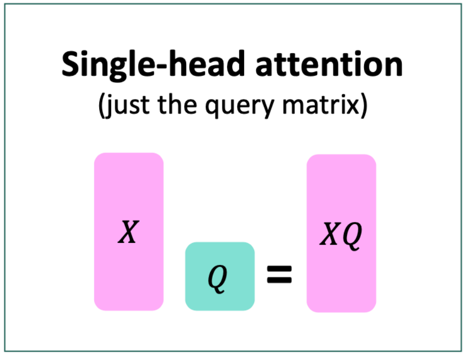
Transformer (Self-attention)

- **Self-attention**
  - **Recall**: Attention operates on query, key, and value
    - Query is decoder's hidden state, key and value are encoder's hidden states in seq2seq
  - In self-attention, the query, key, and value are drawn from the same source
    1. For each input x_i , create query, key, and value vectors q_i, k_i, v_i
    2. Multiply (**dot product**) the current query vector, by all the key vectors, to get a **score** α_{ij}
    3. Multiply the value vectors by the scores, then **sum up**
  - Hence, self-attention is **effective to learn the context** within given sentence
    - It's easier than recurrent layer to be parallelized and model the long-term dependency
    - It also provides an **interpretability** of learned representation



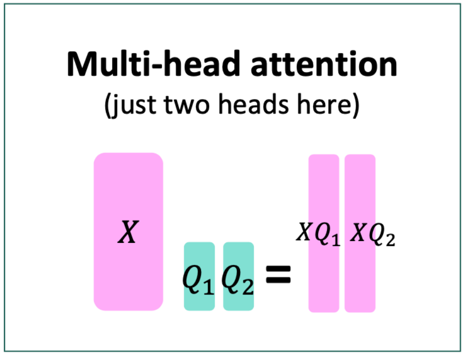
Transformer (Self-attention)

- **Multi-head attention**
  - Applying **multiple attentions at once** to look in multiple places in the sentence
    - To prevent the increase of computation, original attentions weights are **divided**



Single-head attention

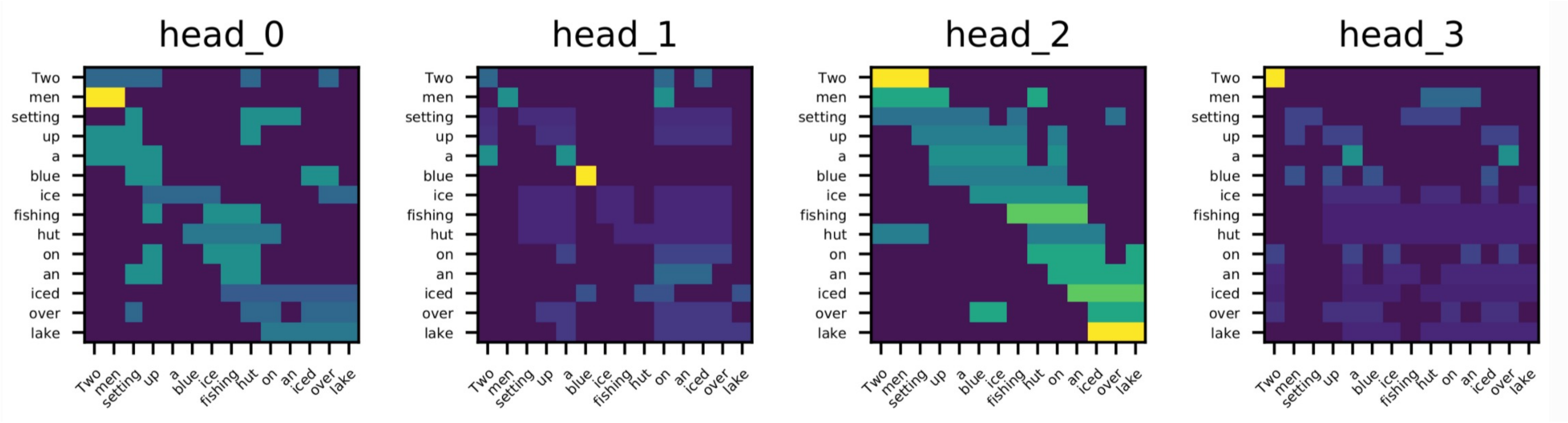
(just the query matrix)



Multi-head attention

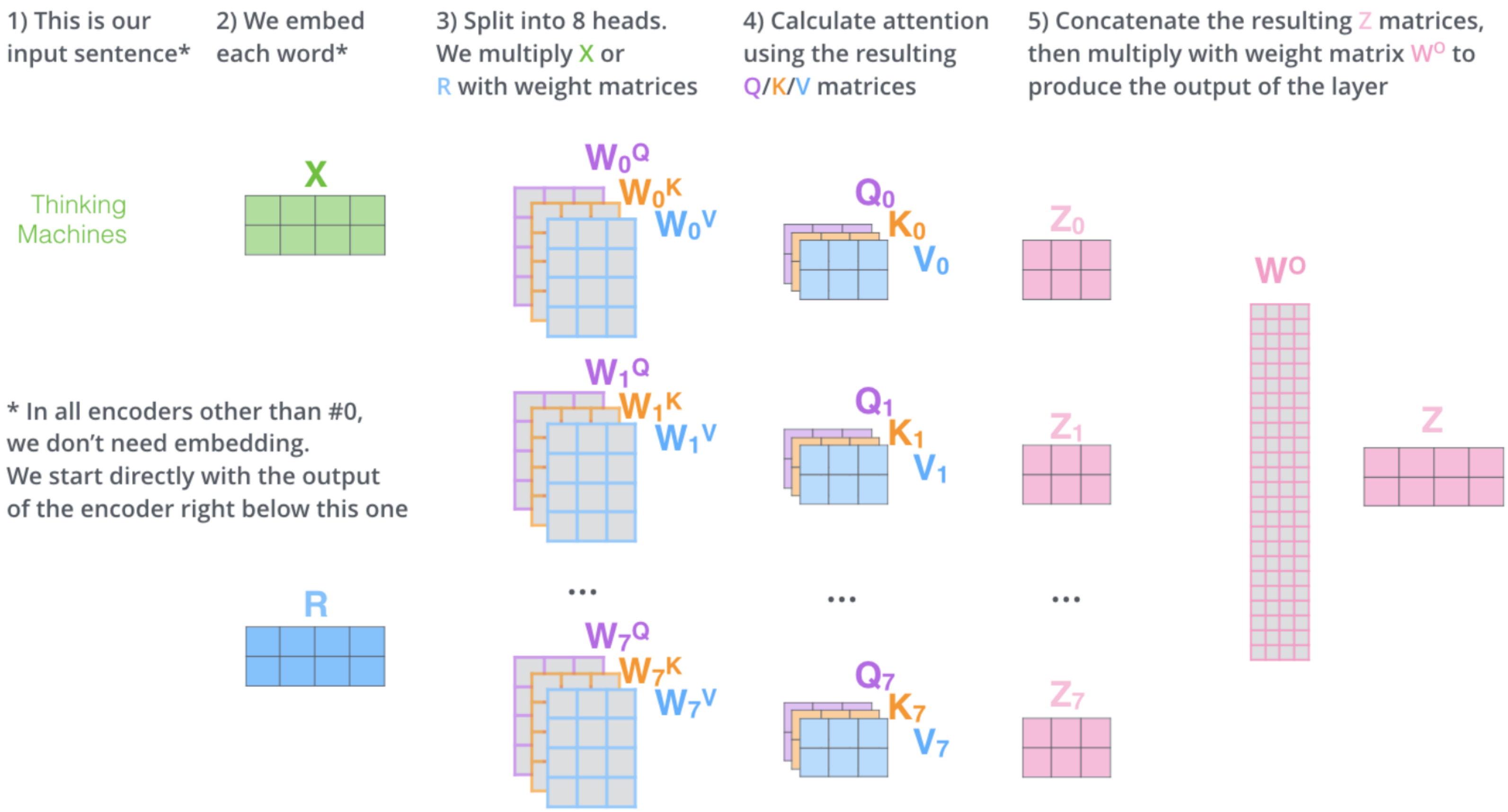
(just two heads here)

Same amount of computation as single-head self-attention



Transformer (Self-attention)

- **Multi-head attention**
  - Applying **multiple attentions at once** to look in multiple places in the sentence

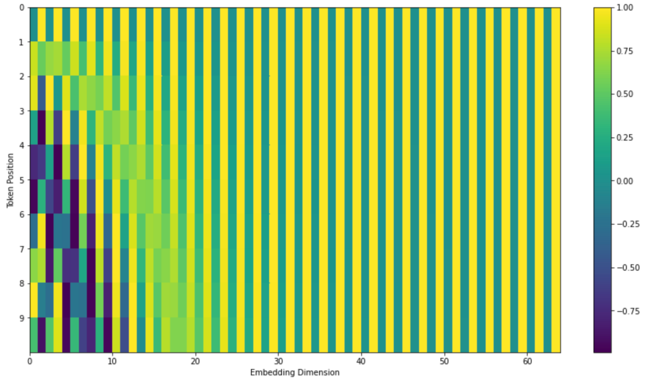


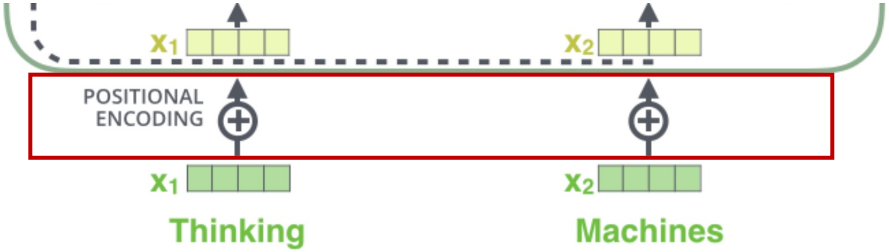
Transformer (Self-attention)

- **Encoder**
  - Self-attention is **invariant to order** of input sequence
    - To represent the order of sequence, **positional encoding** is added to input embeddings at the **bottoms of the encoder and decoder stacks**
  - Fixed sine and cosine functions are used for each position pos and dimension i

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}}) \quad PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

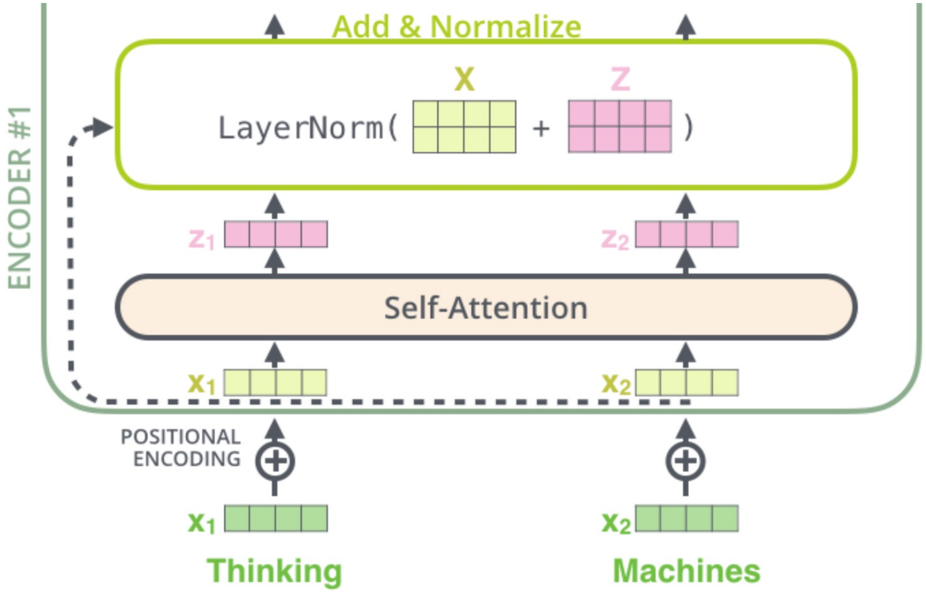
    - PE_{pos+k} can be derived as a linear function of PE_{pos} → easier to learn a relative position
    - Compare to learning encoding, it's better for **extrapolation** (not encountered in training)





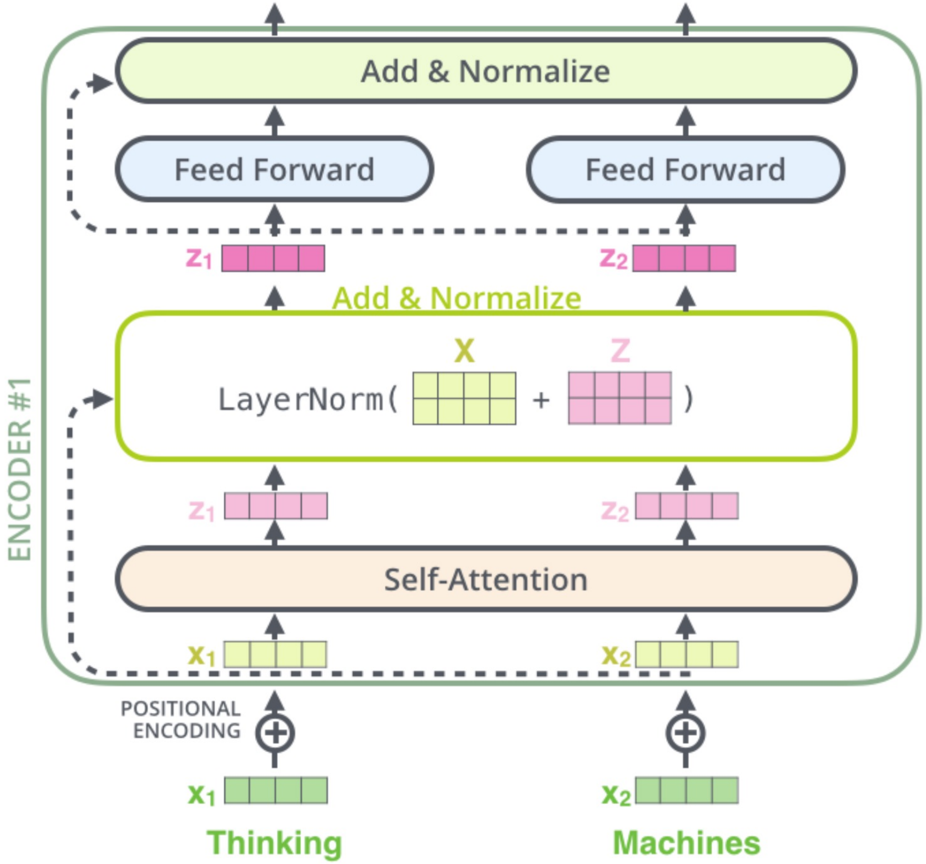
Transformer (Self-attention)

- **Encoder**
  - Self-attention is **invariant to order** of input sequence ⟶ **positional encoding**
  - **Residual connections** (dotted) and **layer normalization** are used to help training



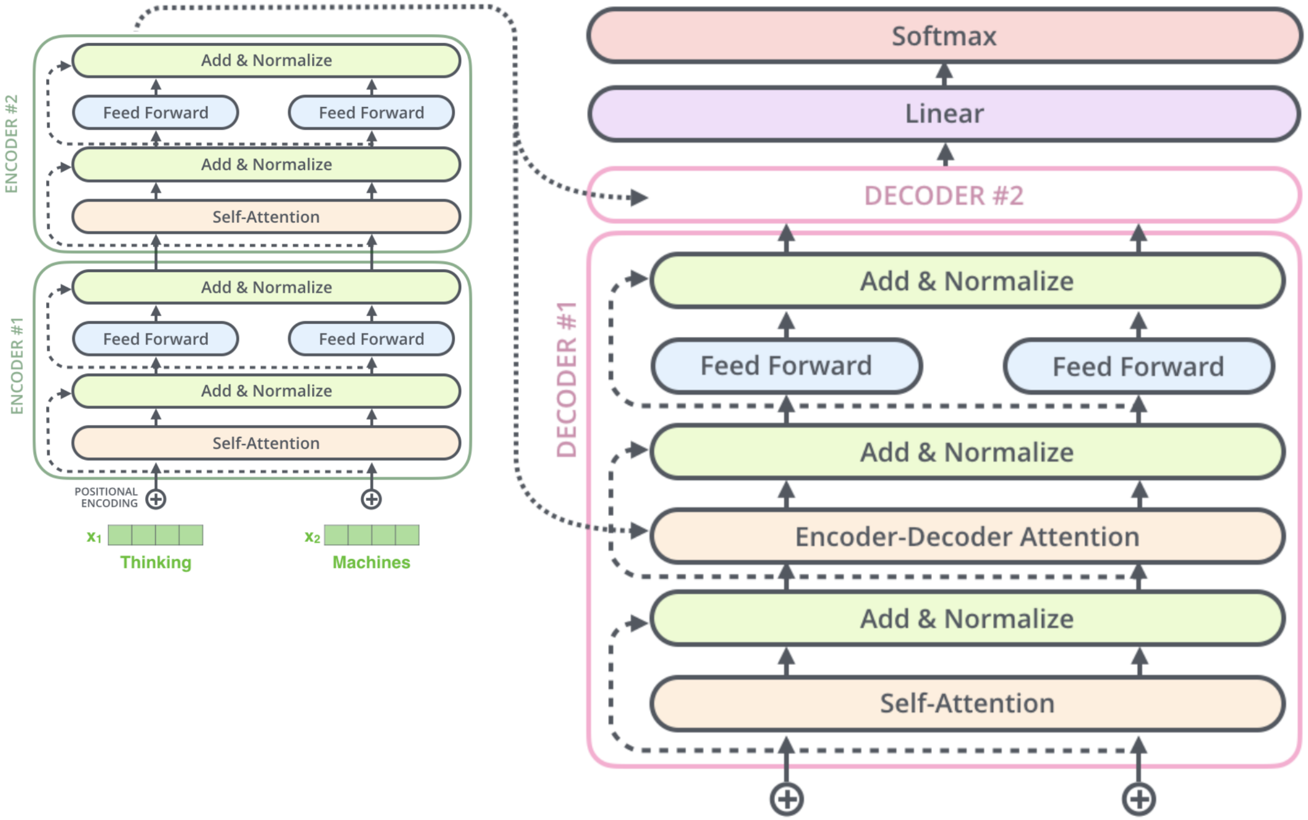
Transformer (Self-attention)

- **Encoder**
  - Self-attention is **invariant to order** of input sequence ⟶ **positional encoding**
  - **Residual connections** (dotted) and **layer normalization** are used to help training
  - Non-linearity is imposed by adding position-wise **feed-forward networks**



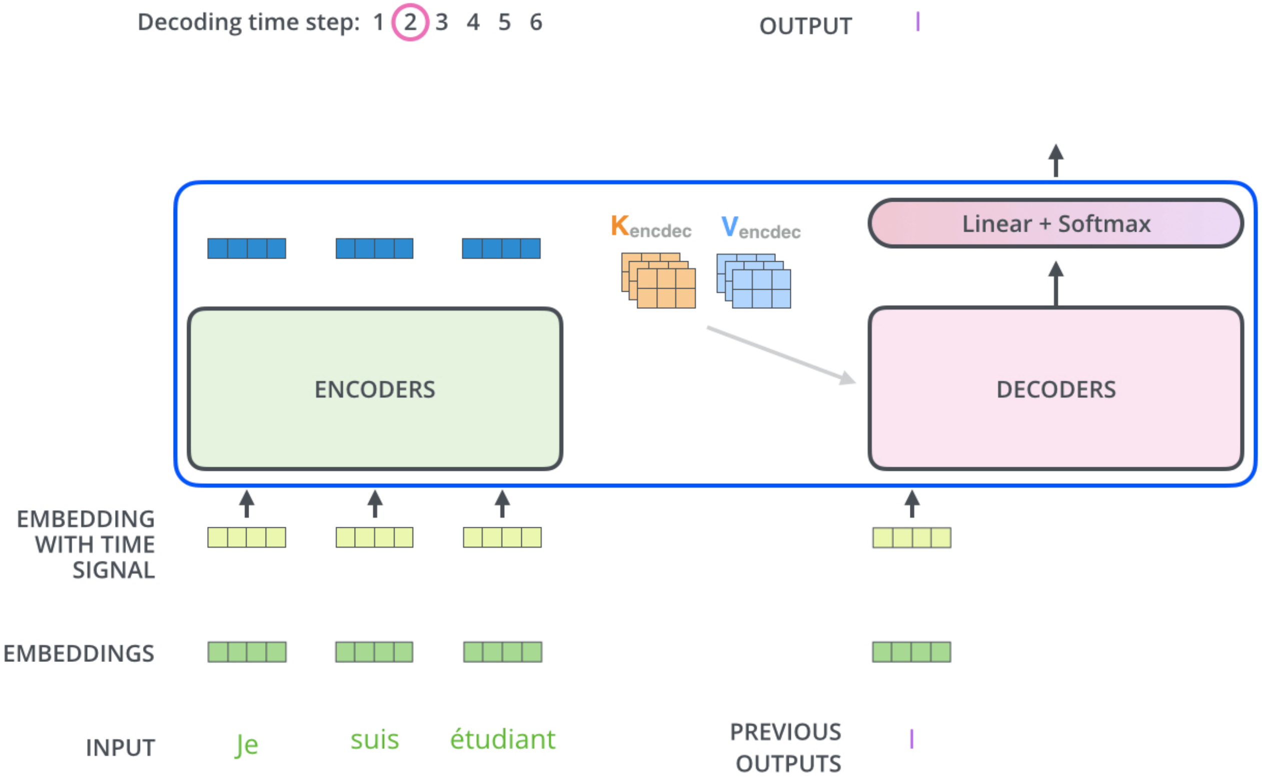
Transformer (Self-attention)

- **Decoder**
  - Most parts are same with encoder except **encoder-decoder(cross) attention**
  - This cross attention is previously used in seq2seq model
    - Queries are drawn from the **decoder**
    - Keys and values are drawn from the **encoder** (like context vector)



Transformer (Self-attention)

- **Decoder**
  - Most parts are same with encoder except **encoder-decoder(cross) attention**
  - This cross attention is previously used in seq2seq model
    - Queries are drawn from the **decoder**
    - Keys and values are drawn from the **encoder** (like context vector)



Transformer (Self-attention)

- Success of Transformer: **Machine Translation (MT)**
  - Initially, Transformer shows **better results at a fraction of the training cost**

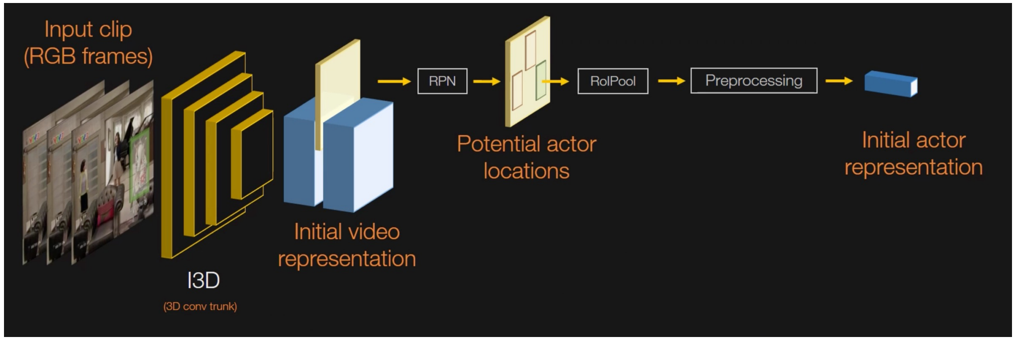
Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

  - Nowadays, Transformer is still a standard for MT with additional techniques

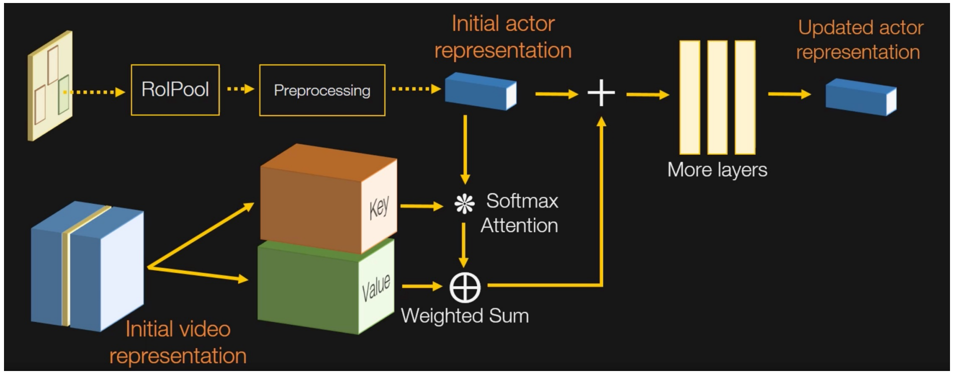
System	**En→De**	
	news2017	**news2018**
baseline	30.90	45.40
+ langid filtering	30.78	46.43
+ ffn 8192	31.15	46.28
+ BT	33.62	46.66
+ fine tuning	-	47.61
+ ensemble	-	49.27
+ reranking	-	50.63
WMT'18 submission	-	46.10
WMT'19 submission	**42.7**	

Transformer (Self-attention)

- Success of Transformer: **Video action recognition** [Girdhar et al., 2018]
  - **Goal**: localize the atomic action in space and time
  - Previous approaches just use the feature of key frame with object detection
    - But, it's hard to model the interaction between frames

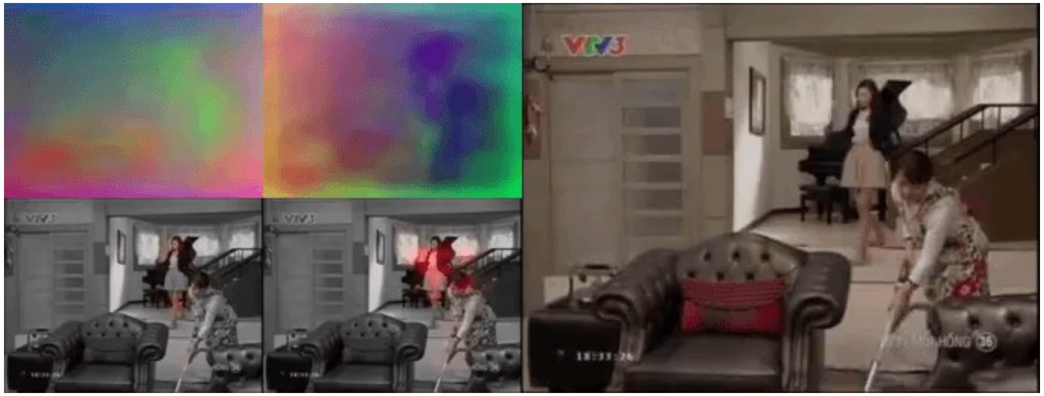


- **Self-attention is an effective way** to resolve this issue

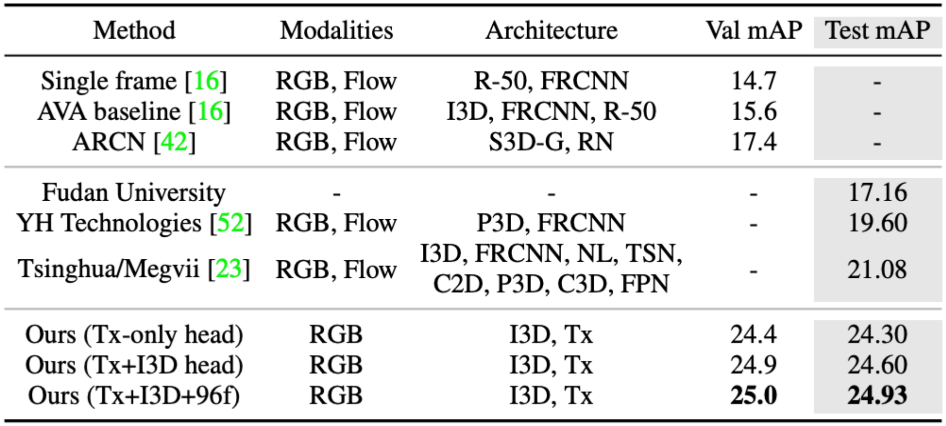


Transformer (Self-attention)

- Success of Transformer: **Video action recognition** [Girdhar et al., 2018]
  - **Qualitative results of learned attention**



  - **Winner of AVA challenge in 2019: > 3.5 %** than previous challenge winner



Method	Modalities	Architecture	Val mAP	Test mAP
Single frame [16]	RGB, Flow	R-50, FRCNN	14.7	-
AVA baseline [16]	RGB, Flow	I3D, FRCNN, R-50	15.6	-
ARCN [42]	RGB, Flow	S3D-G, RN	17.4	-
Fudan University	-	-	-	17.16
YH Technologies [52]	RGB, Flow	P3D, FRCNN	-	19.60
Tsinghua/Megvii [23]	RGB, Flow	I3D, FRCNN, NL, TSN, C2D, P3D, C3D, FPN	-	21.08
Ours (Tx-only head)	RGB	I3D, Tx	24.4	24.30
Ours (Tx+I3D head)	RGB	I3D, Tx	24.9	24.60
Ours (Tx+I3D+96f)	RGB	I3D, Tx	**25.0**	**24.93**

Transformer (Self-attention)

- Success of Transformer: **Music generation** [Huang et al., 2018]
  - **Goal**: generate music which contains structure at multiple timescales (short to long)
  - Performance RNN (LSTM): lack of long-term structure

  - Music transformer; able to continue playing with consistent style

Overview

Part 1. Basics

- RNN to LSTM
- Sequence-to-sequence Model
- Attention-based NLP Model

Part 2. Transformers and Large Language Models

- Transformer (self-attention)
- Pre-training of Transformers and Language Models

Part 3. Advanced Topics

- Handling long inputs with Transformers
- Techniques for improving efficiency
- State-Space Models

Part 4. Summary

Pre-training and Fine-tuning Paradigm with Transformers

- **Motivation**
  - Many success of computer vision comes from ImageNet-pretrained networks
    - Simple fine-tuning improves the performance than training from scratch
  - **Q**. Then, can we train a similar universal pre-trained network for NLP tasks?
    - As labeling of NLP task is more ambiguous, **unsupervised pre-training is essential**
  - **Language modeling** is simple yet effective pre-training method without label
    - i.e., predicting what will be the next word
    - With diverse examples, model can learn the useful knowledge about the world

"Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was __." ⟶ *terrible*

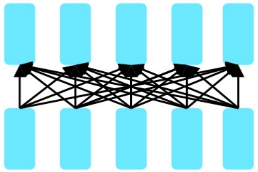
"I wat thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, __" ⟶ *34*

"I went to the ocean to see the fish, turtles, seals, and __" ⟶ *sand*

Pre-training and Fine-tuning Paradigm with Transformers

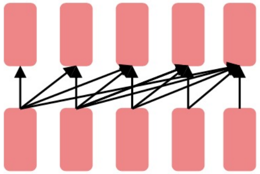
- **Motivation**
  - Many success of computer vision comes from ImageNet-pretrained networks
    - Simple fine-tuning improves the performance than training from scratch
  - **Q**. Then, can we train a similar universal pre-trained network for NLP tasks?
    - As labeling of NLP task is more ambiguous, **unsupervised pre-training is essential**
  - **Language modeling** is simple yet effective pre-training method without label
    - i.e., predicting what will be the next word
    - With diverse examples, model can learn the useful knowledge about the world

- **Pre-training for two types of architectures**
  - Architecture influences the type of pre-training, and specific use cases



Encoders

- E.g. **BERT**
- Pre-training with **masked** language modeling
- Better use for **discriminative** tasks (classification)

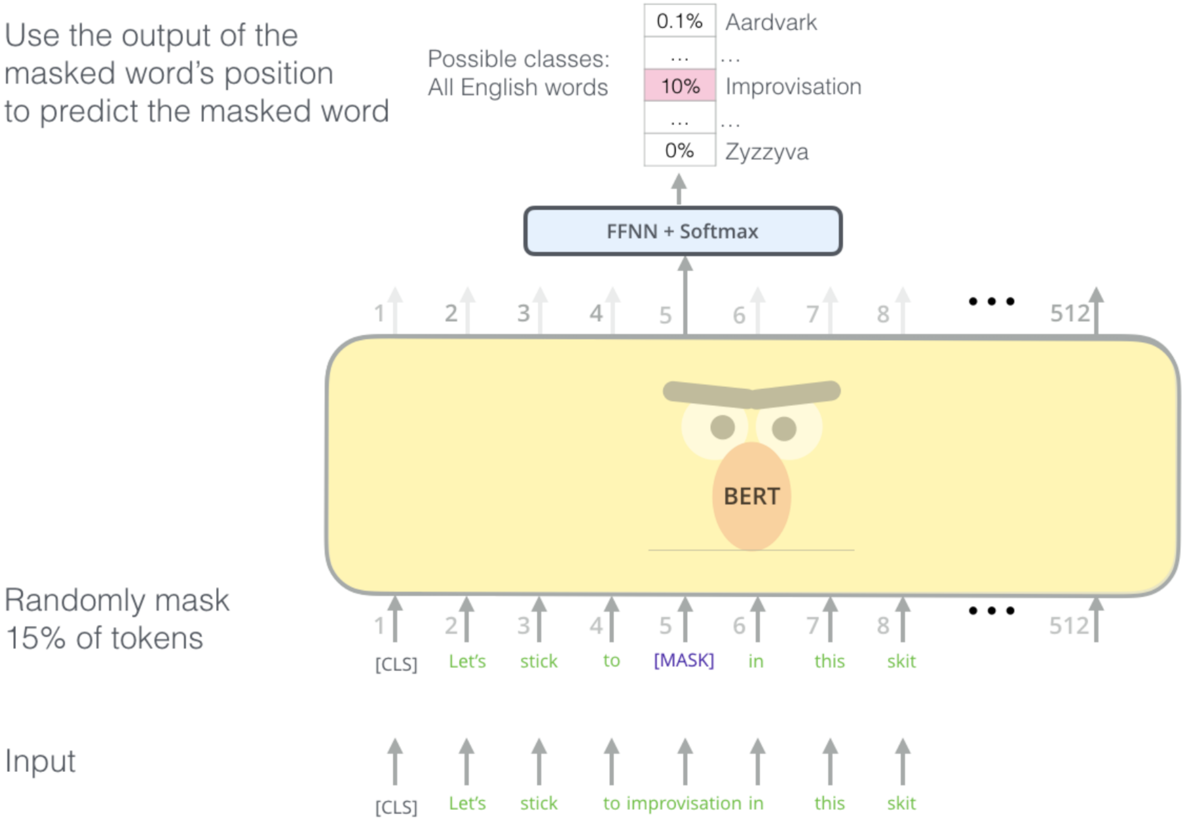


Decoders

- E.g. **GPT**
- Pre-training with **normal** language modeling
- Better use for **generation** tasks

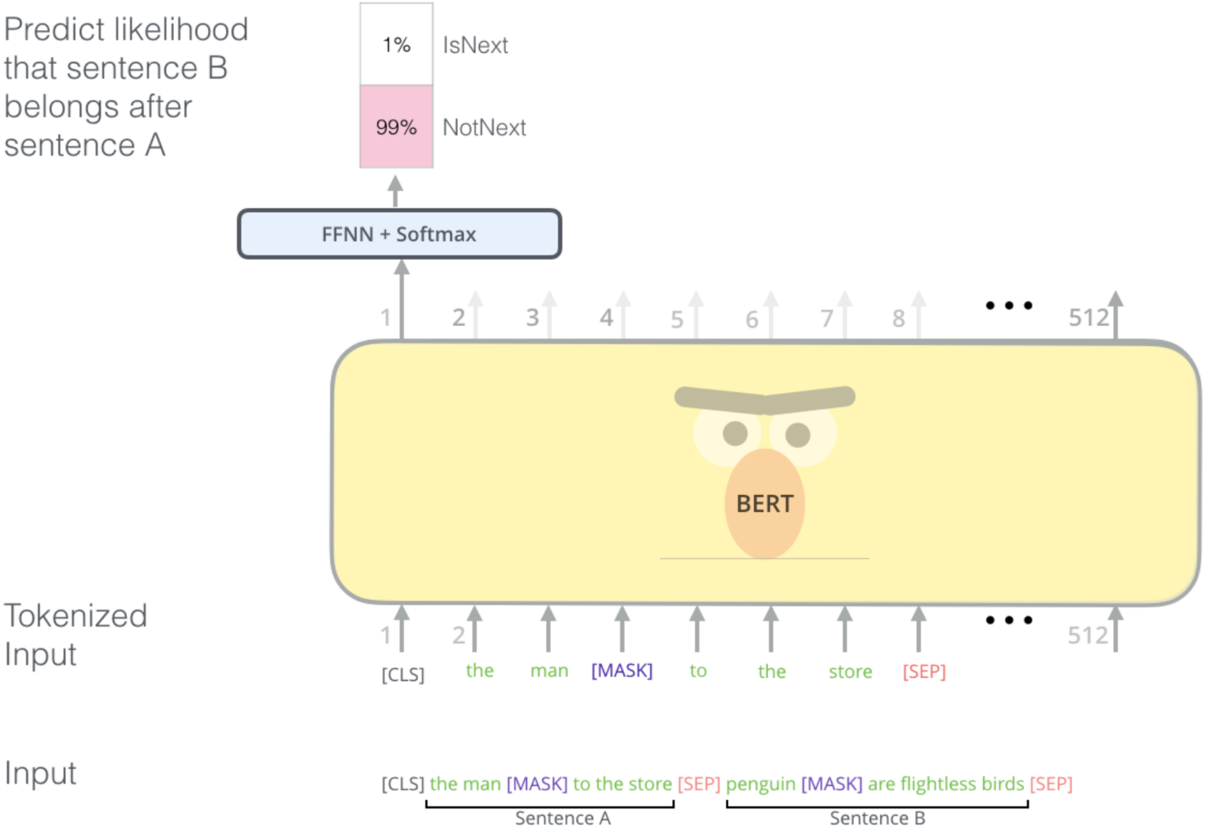
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

- **BERT**: Bidirectional Encoder Representations from Transformers [Devlin et al., 2018]
  - As **encoders get bidirectional context**, original language modeling is suboptimal
    - Not only left-to-right, but also right-to-left modeling is possible
  - Hence, **masked language modeling** is used for pre-training
    - Replace some fraction of words (15%) in the input, then predict these words



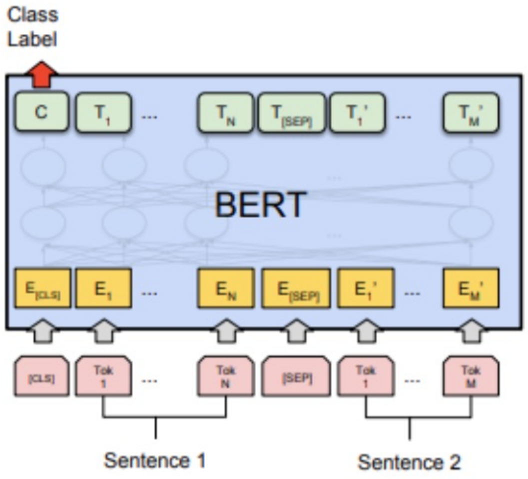
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

- **BERT**: Bidirectional Encoder Representations from Transformers [Devlin et al., 2018]
  - As **encoders get bidirectional context**, original language modeling is suboptimal
  - Hence, **masked language modeling** is used for pre-training
  - Additionally, **next sentence prediction** (NSP) task is used for pre-training
    - Decide whether two input sentences are **consecutive or not**

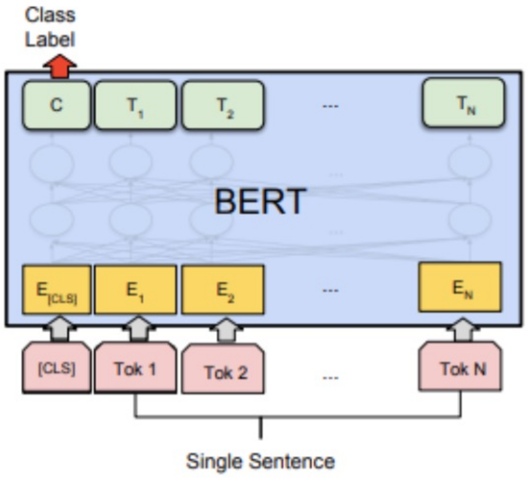


BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

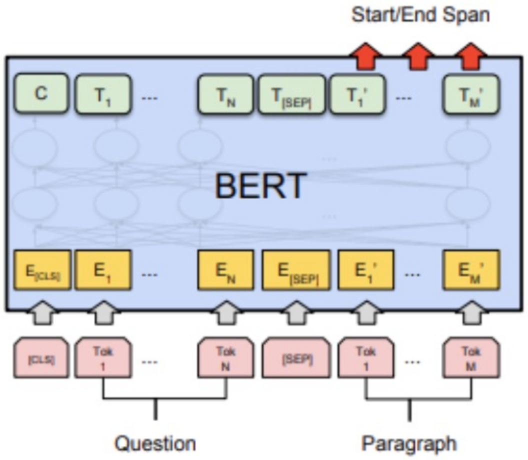
- **BERT**: Bidirectional Encoder Representations from Transformers [Devlin et al., 2018]
  - Even **without** task-specific complex architectures, BERT achieves **SOTA** for **11 NLP tasks**, including classification, question answering, tagging, etc.
    - By simply fine-tuning a whole network with **additional linear classifier**



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



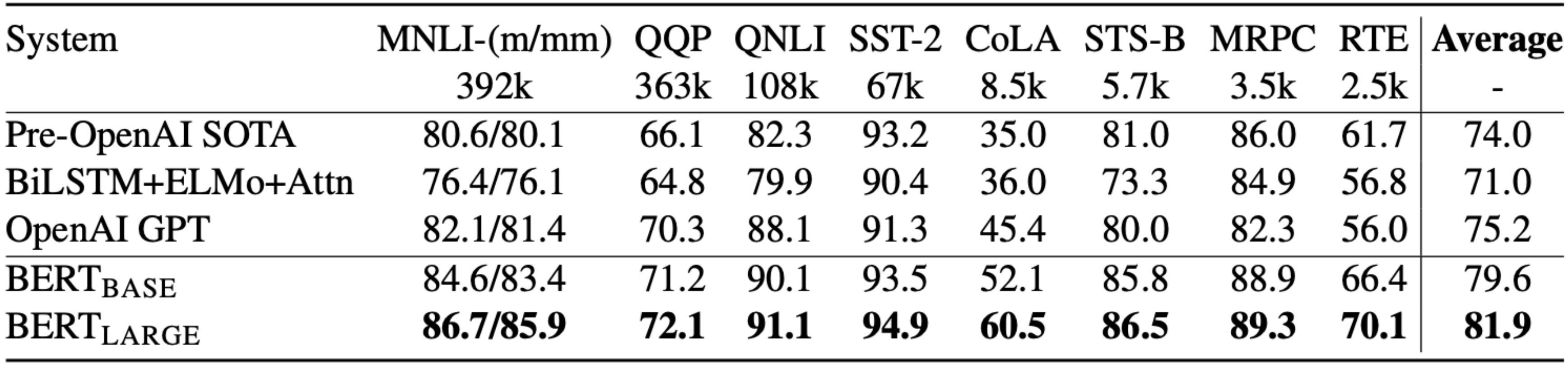
(b) Single Sentence Classification Tasks: SST-2, CoLA



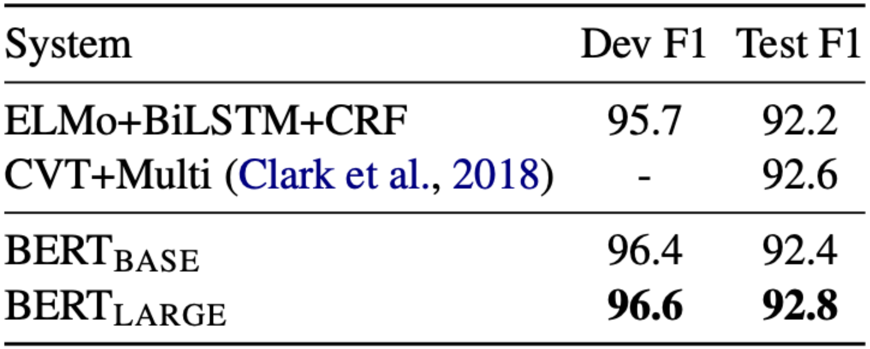
(c) Question Answering Tasks: SQuAD v1.1

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

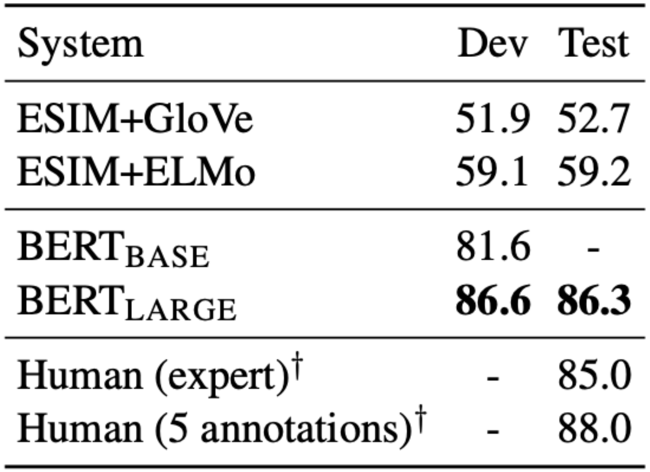
- **BERT**: Bidirectional Encoder Representations from Transformers [Devlin et al., 2018]
  - Even **without** task-specific complex architectures, BERT achieves **SOTA** for **11 NLP tasks**, including classification, question answering, tagging, etc.
    - By simply fine-tuning a whole network with **additional linear classifier**



System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
BERT$_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT$_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**



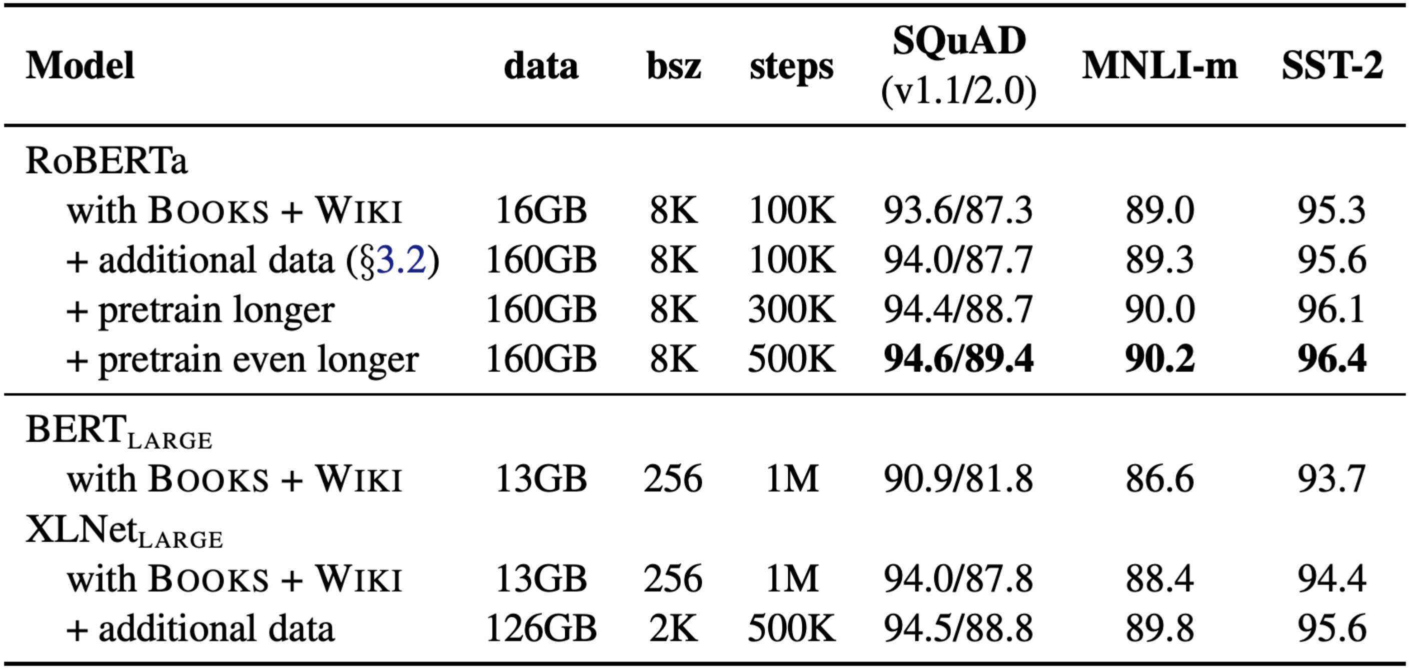
System	Dev F1	Test F1
ELMo+BiLSTM+CRF	95.7	92.2
CVT+Multi (Clark et al., 2018)	-	92.6
BERT$_{\text{BASE}}$	96.4	92.4
BERT$_{\text{LARGE}}$	**96.6**	**92.8**



System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
BERT$_{\text{BASE}}$	81.6	-
BERT$_{\text{LARGE}}$	**86.6**	**86.3**
Human (expert)†	-	85.0
Human (5 annotations)†	-	88.0

RoBERTa: A Robustly Optimized BERT Pre-training Approach

- **RoBERTa** [Liu et al., 2019]
  - Simply modifying BERT design choices and training strategies with alternatives
    - Using **dynamic masking** instead of static masking in BERT
    - **Removing NSP task** and generate training data in single document instead
    - Much **larger data** for pre-training: 16GB ⟶ 160GB, and etc…
  - But, it leads a huge improvement in many downstream tasks



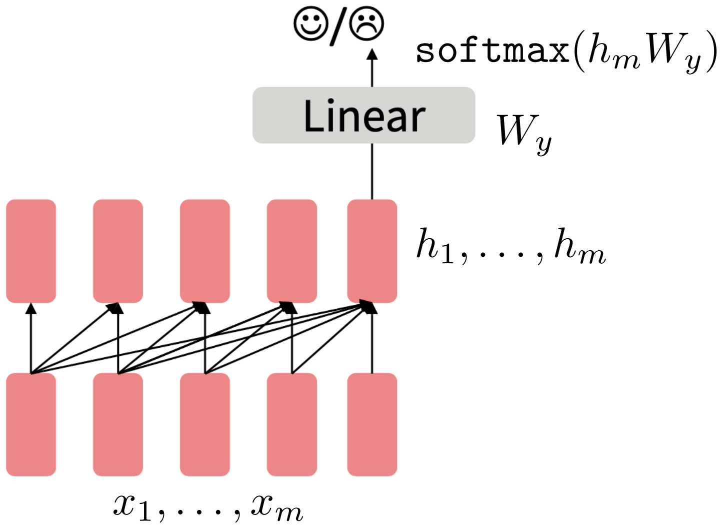
Model	**data**	**bsz**	**steps**	**SQuAD** (v1.1/2.0)	**MNLI-m**	**SST-2**
RoBERTa						
with Books + Wiki	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with Books + Wiki	13GB	256	1M	90.9/81.8	86.6	93.7
$\text{XLNet}_{\text{LARGE}}$						
with Books + Wiki	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

GPT: Generative Pre-Training with Transformer's Decoder

- **GPT** [Radford et al., 2018]

$$\arg\max_{\theta} \log p(\boldsymbol{x}) = \sum_{n} p_{\theta}(x_n | x_1, \ldots, x_{n-1})$$

  - **Pre-training** by language modeling over 7000 unique books (**unlabeled data**)
    - Contains long spans of contiguous text, for learning long-distance dependencies
  - **Fine-tuning** by training a classifier with target task-specific **labeled data**
    - Classifier is added on the final transformer block's last word's hidden state

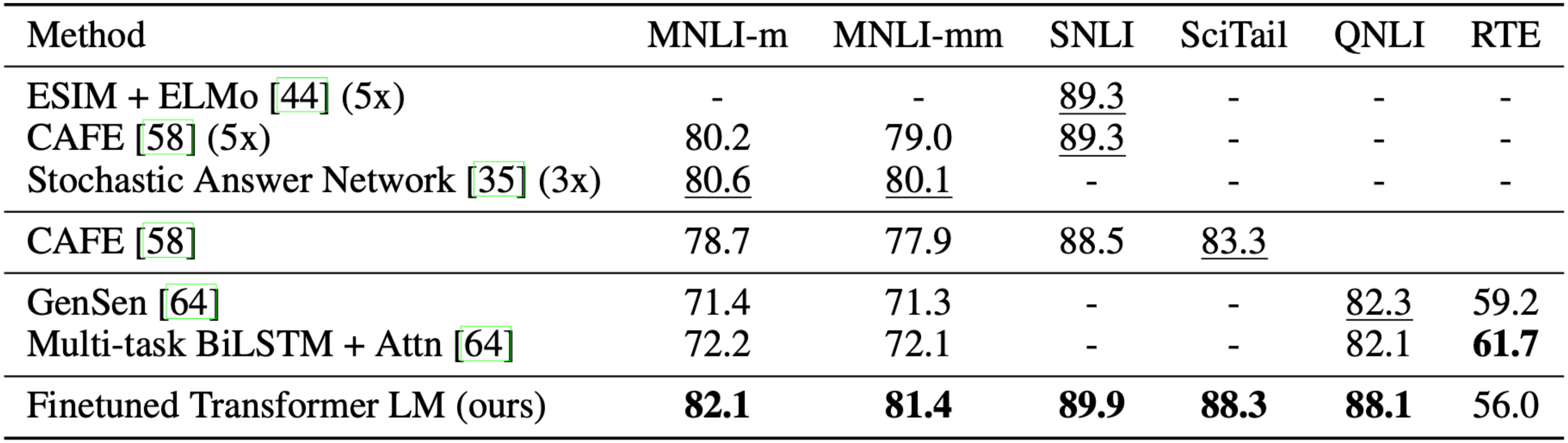


GPT: Generative Pre-Training with Transformer's Decoder

- **GPT** [Radford et al., 2018]

$$\arg\max_{\theta} \log p(\boldsymbol{x}) = \sum_{n} p_{\theta}(x_n | x_1, \ldots, x_{n-1})$$

  - **Pre-training** by language modeling over 7000 unique books (**unlabeled data**)
    - Contains long spans of contiguous text, for learning long-distance dependencies
  - **Fine-tuning** by training a classifier with target task-specific **labeled data**
    - Classifier is added on the final transformer block's last word's hidden state

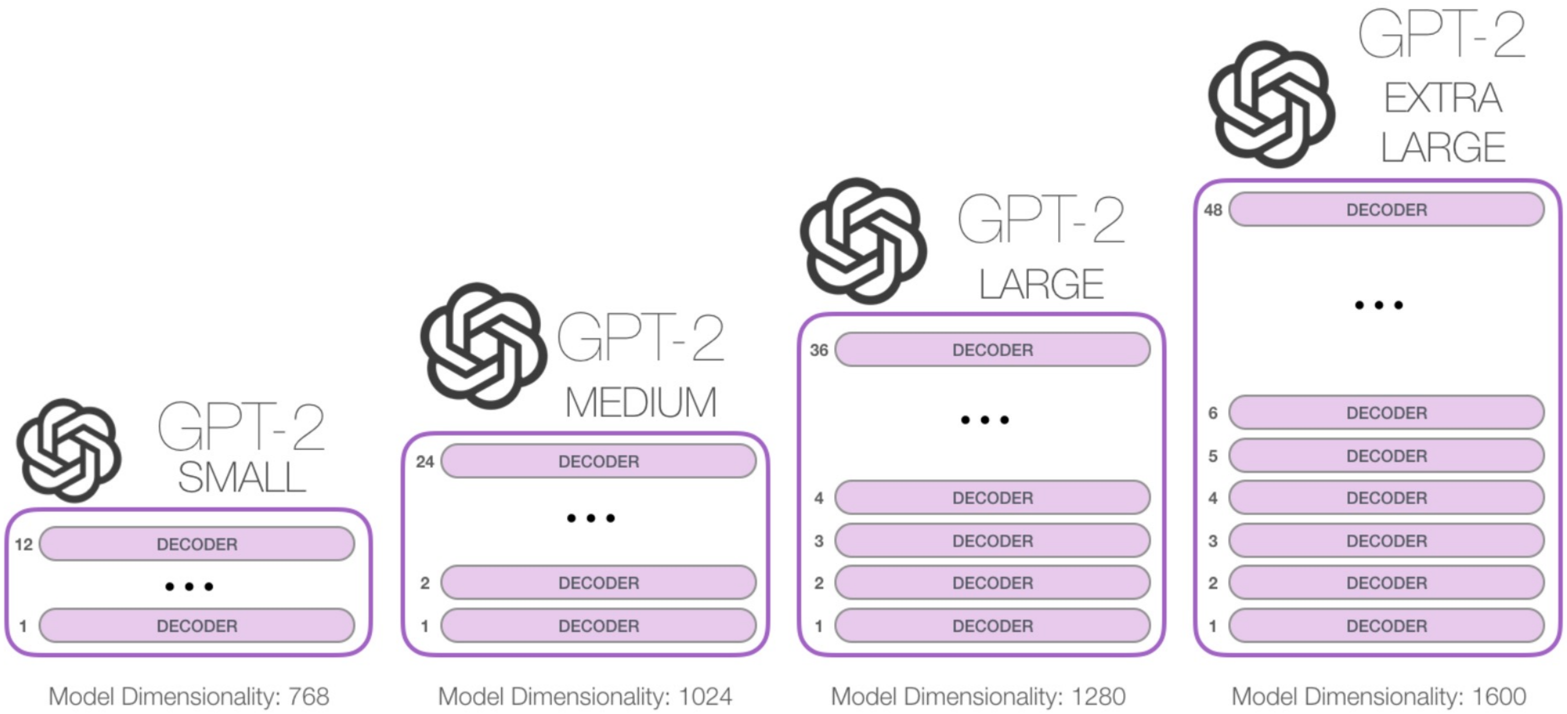


Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned Transformer LM (ours)	**82.1**	**81.4**	**89.9**	**88.3**	**88.1**	56.0

GPT's results on various *natural language inference* datasets

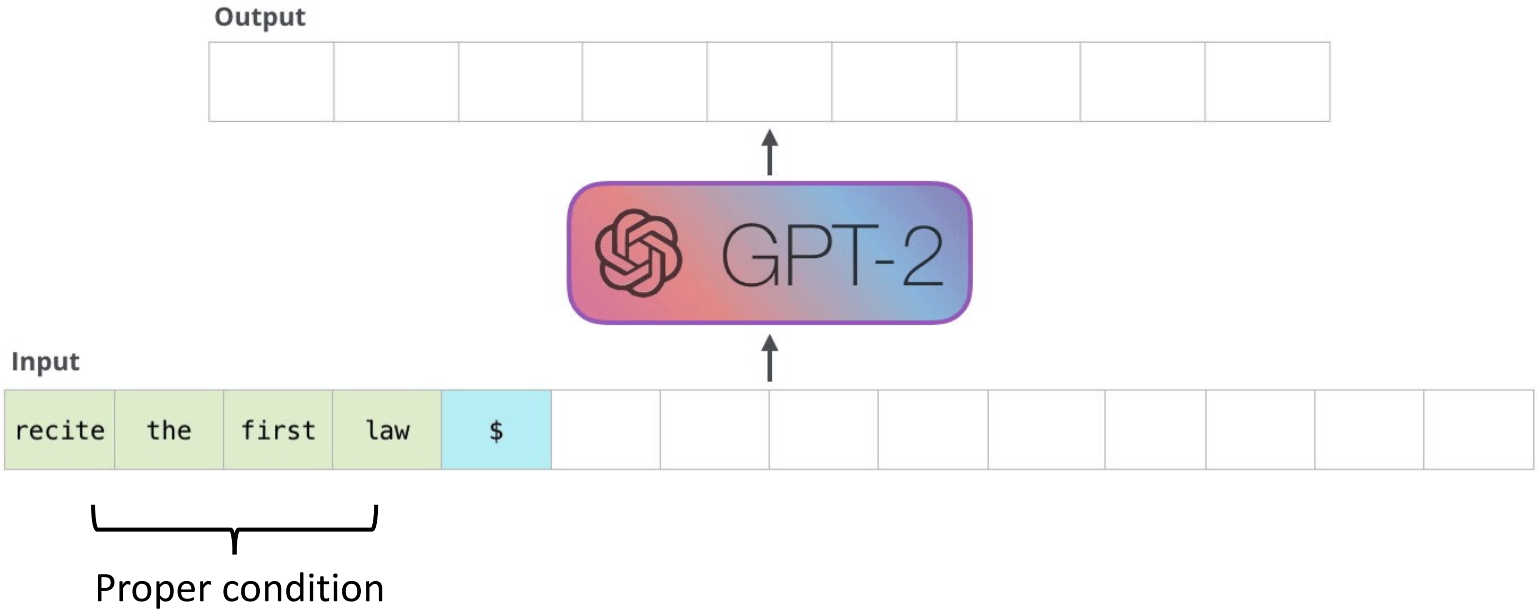
GPT-2: Language Models are Unsupervised Multitask Learners

- **GPT-2** [Radford et al., 2019]
    - **Pre-training** by language modeling as same as previous GPT-1, but **training with..**
        - Much **larger datasets**; 8 million documents from web (40 GB of text)
        - Much **larger model size**; # of parameters: 117M (GPT-1) ⟶ 1542M (extra-large GPT-2)



GPT-2: Language Models are Unsupervised Multitask Learners

- **GPT-2** [Radford et al., 2019]
  - **Pre-training** by language modeling as same as previous GPT-1, but **training with..**
    - Much **larger datasets**; 8 million documents from web (40 GB of text)
    - Much **larger model size**; # of parameters: 117M (GPT-1) ⟶ 1542M (extra-large GPT-2)
  - GPT-2 can perform down-stream tasks in a **zero-shot setting**
    - Via conditional generation without any parameter or architecture modification



GPT-2: Language Models are Unsupervised Multitask Learners

- **GPT-2** [Radford et al., 2019]
  - **Pre-training** by language modeling as same as previous GPT-1, but **training with..**
    - Much **larger datasets**; 8 million documents from web (40 GB of text)
    - Much **larger model size**; # of parameters: 117M (GPT-1) ⟶ 1542M (extra-large GPT-2)
  - GPT-2 can perform down-stream tasks in a **zero-shot setting**
    - Via conditional generation without any parameter or architecture modification
  - **Remark**. Largest model still underfits.. ⟶ larger model for better performance?

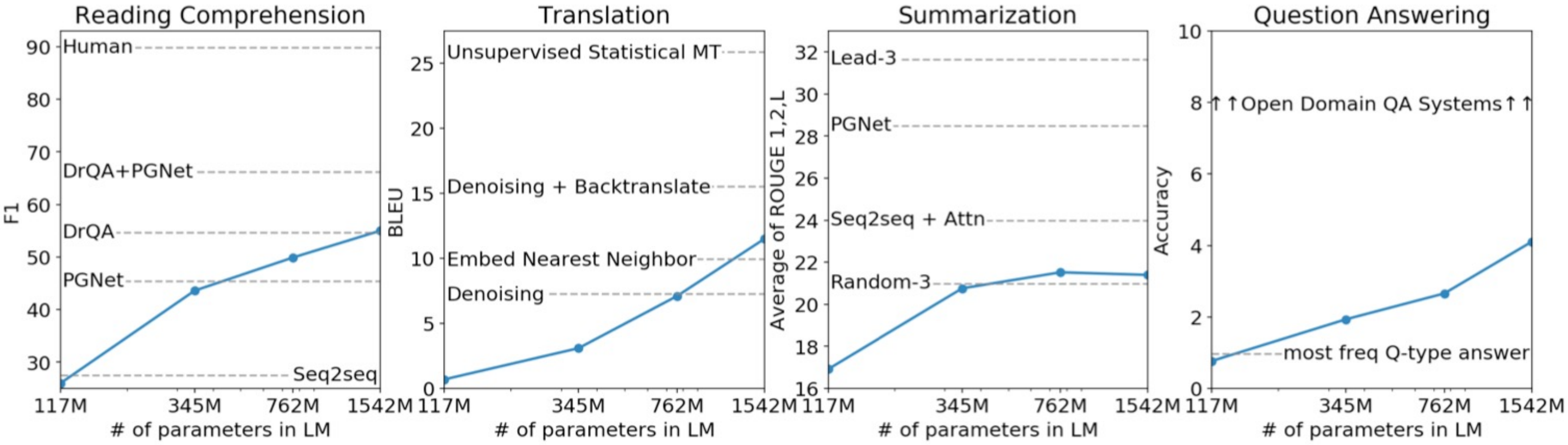
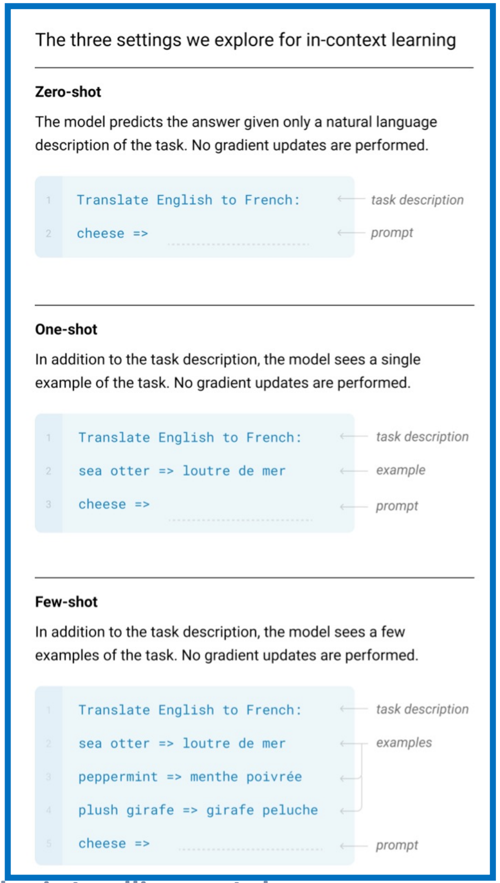


Figure 1. Zero-shot task performance of WebText LMs as a function of model size on many NLP tasks. Reading Comprehension results are on CoQA (Reddy et al., 2018), translation on WMT-14 Fr-En (Artetxe et al., 2017), summarization on CNN and Daily Mail (See et al., 2017), and Question Answering on Natural Questions (Kwiatkowski et al., 2019). Section 3 contains detailed descriptions of each result.

GPT-3: Language Models are Few-shot Learners

- **GPT-3**: Language Models are Few-shot Learners [Brown et al., 2020]
  - **First very large** language models (**LLMs**, 1B → 175B parameters)
  - With this scale-up, new capability of LMs suddenly emerges
  - E.g., it can adapt to new tasks **perform in-context learning** **without fine-tuning**
    - **In-context learning (prompting)**; adapting to task from examples with some context



The three settings we explore for in-context learning

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
Translate English to French:      ← task description
cheese =>                         ← prompt
```

One-shot

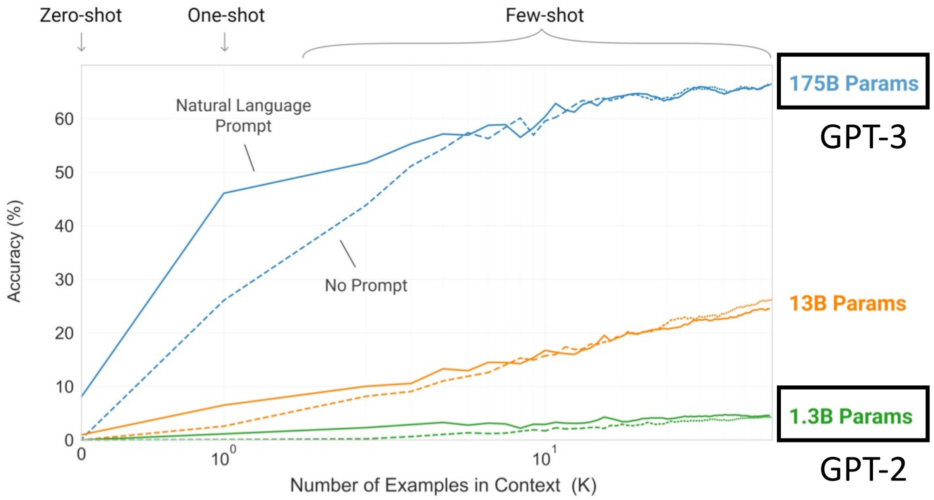
In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

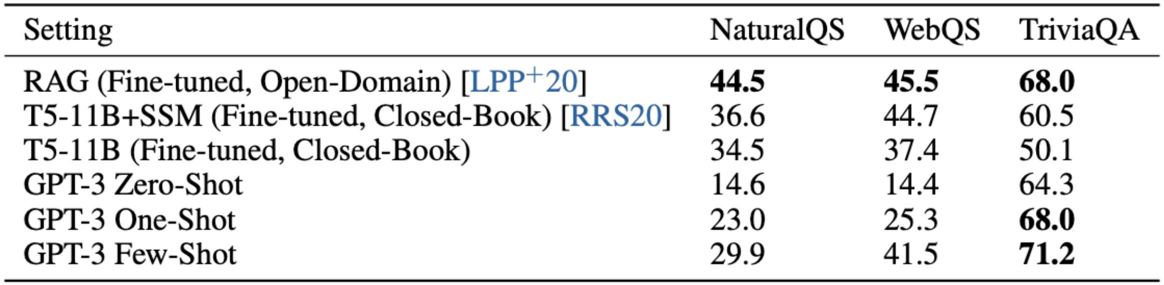
```
Translate English to French:      ← task description
sea otter => loutre de mer        ← example
cheese =>                         ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:      ← task description
sea otter => loutre de mer        ← examples
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>                         ← prompt
```

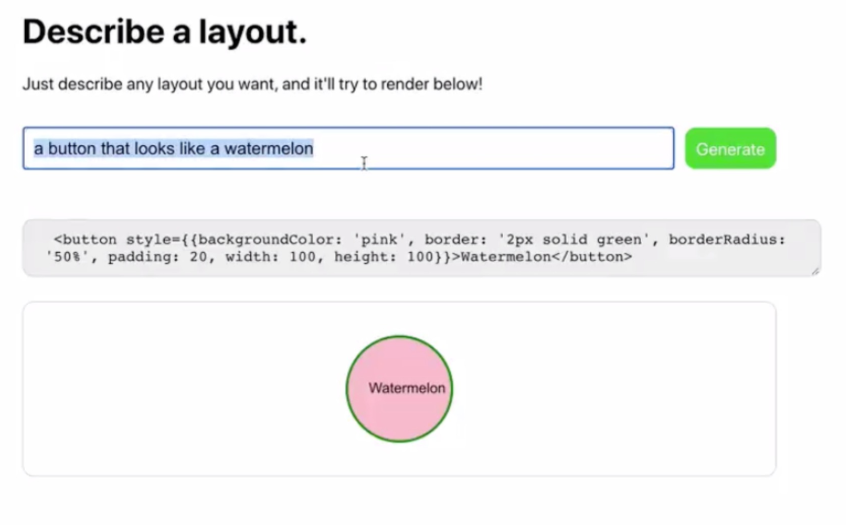




Setting	NaturalQS	WebQS	TriviaQA
RAG (Fine-tuned, Open-Domain) [LPP$^+$20]	**44.5**	**45.5**	**68.0**
T5-11B+SSM (Fine-tuned, Closed-Book) [RRS20]	36.6	44.7	60.5
T5-11B (Fine-tuned, Closed-Book)	34.5	37.4	50.1
GPT-3 Zero-Shot	14.6	14.4	64.3
GPT-3 One-Shot	23.0	25.3	**68.0**
GPT-3 Few-Shot	29.9	41.5	**71.2**

GPT-3: Language Models are Few-shot Learners

- **GPT-3**: Language Models are Few-shot Learners [Brown et al., 2020]
  - **First very large** language models (**LLMs**, 1B → 175B parameters)
  - With this scale-up, new capability of LMs suddenly emerges
  - E.g., it can adapt to new tasks **perform in-context learning** **without fine-tuning**
  - It enables us to do a lot of interesting applications!
  - E.g.,



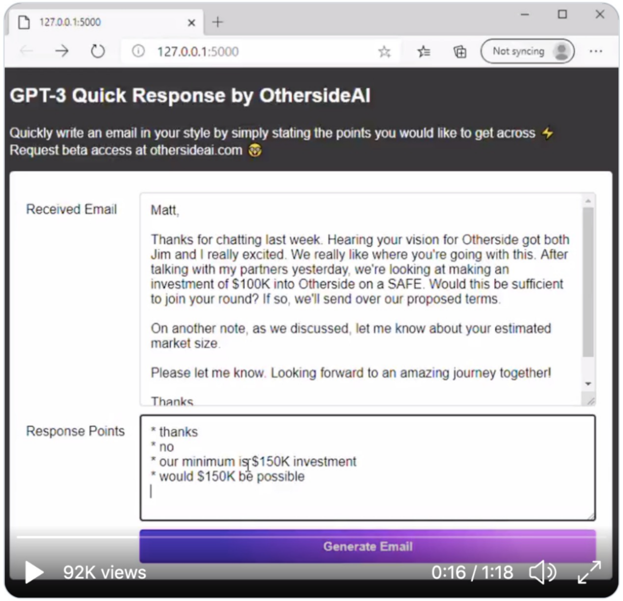
Describe a layout.

Just describe any layout you want, and it'll try to render below!

a button that looks like a watermelon

```
<button style={{backgroundColor: 'pink', border: '2px solid green', borderRadius:
'50%', padding: 20, width: 100, height: 100}}>Watermelon</button>
```

Simple code generation



GPT-3 Quick Response by OthersideAI

Quickly write an email in your style by simply stating the points you would like to get across ⚡
Request beta access at othersideai.com 😎

Received Email

Matt,

Thanks for chatting last week. Hearing your vision for Otherside got both Jim and I really excited. We really like where you're going with this. After talking with my partners yesterday, we're looking at making an investment of $100K into Otherside on a SAFE. Would this be sufficient to join your round? If so, we'll send over our proposed terms.

On another note, as we discussed, let me know about your estimated market size.

Please let me know. Looking forward to an amazing journey together!

Thanks

Response Points

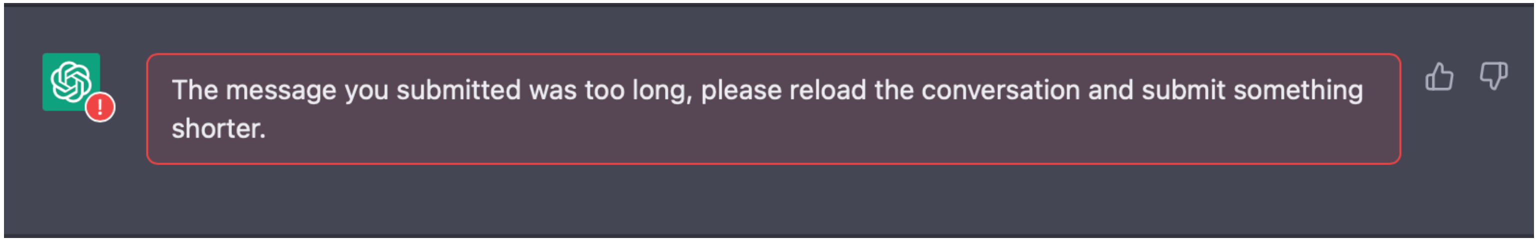
* thanks
* no
* our minimum is $150K investment
* would $150K be possible

Email response

Remaining challenges

- Despite the remarkable success of LLMs, practical challenges remain
  - **High computational demands**
    - Computational requirements increase as **LLMs grow in scale**
    - Self-attention requires **O(sequence length^2)** computation and memory
    - Poses challenges for both **training** and **deployment**

  - **Handling of long contexts**
    - Computation grows quadratically with sequence length
    - Model **does not naturally extrapolate** to long sequences unseen during training
    - Additional techniques are required to handle longer inputs with pre-trained LLMs



The message you submitted was too long, please reload the conversation and submit something shorter.

Overview

Part 1. Basics

- RNN to LSTM
- Sequence-to-sequence Model
- Attention-based NLP Model

Part 2. Transformers and Large Language Models

- Transformer (self-attention)
- Pre-training of Transformers and Language Models

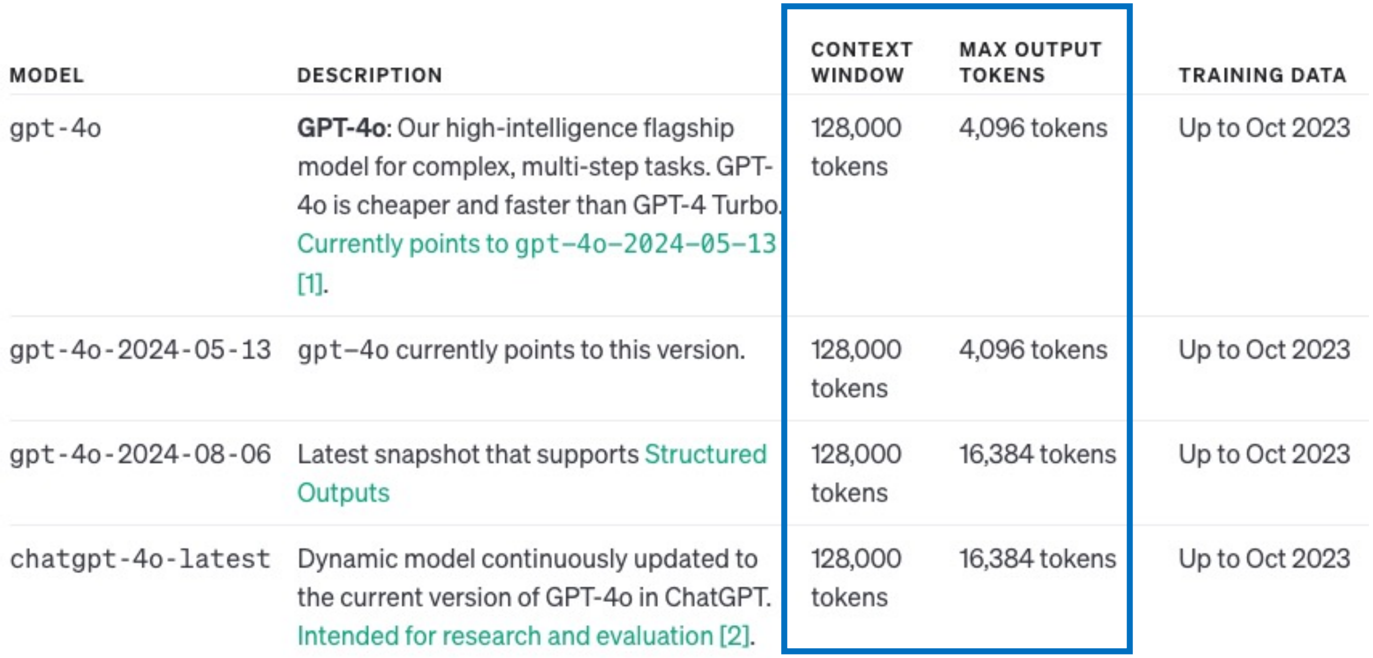
Part 3. Advanced Topics

- Handling long inputs with Transformers
- Techniques for improving efficiency
- State-Space Models

Part 4. Summary

Handling long context with Transformers

- Ability to handle long inputs is an important feature for modern LLMs
  - Motivations for handling **very long sequences**
    - Book comprehension
    - Repository-level code understanding
    - Processing long multimodal inputs (e.g. videos)
    - …



MODEL	DESCRIPTION	CONTEXT WINDOW	MAX OUTPUT TOKENS	TRAINING DATA
`gpt-4o`	**GPT-4o**: Our high-intelligence flagship model for complex, multi-step tasks. GPT-4o is cheaper and faster than GPT-4 Turbo. Currently points to `gpt-4o-2024-05-13` [1].	128,000 tokens	4,096 tokens	Up to Oct 2023
`gpt-4o-2024-05-13`	`gpt-4o` currently points to this version.	128,000 tokens	4,096 tokens	Up to Oct 2023
`gpt-4o-2024-08-06`	Latest snapshot that supports Structured Outputs	128,000 tokens	16,384 tokens	Up to Oct 2023
`chatgpt-4o-latest`	Dynamic model continuously updated to the current version of GPT-4o in ChatGPT. Intended for research and evaluation [2].	128,000 tokens	16,384 tokens	Up to Oct 2023

Handling long context with Transformers

- 3 lines of research for long-context language models:

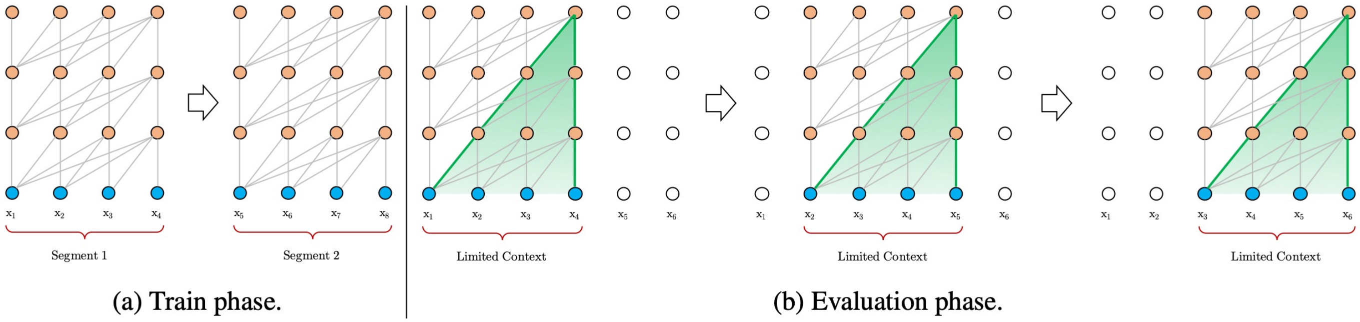
- **Recurrence-based methods**
  - Segments long inputs, and reuses the preceding segment's hidden states
  - The hidden states serve as 'memory' for the current segment

- **Retrieval-based methods**
  - Encodes prior sequences as (key, value) pairs
  - Uses a retrieval algorithm to extract previously encoded information

- **Modifying of positional encodings**
  - Applicable to language models utilizing rotary position embeddings (RoPE)
  - Interpolates the position indices, extending the context limit of existing LLMs with minimal or no additional training

Recurrence-based Methods: Transformer-XL

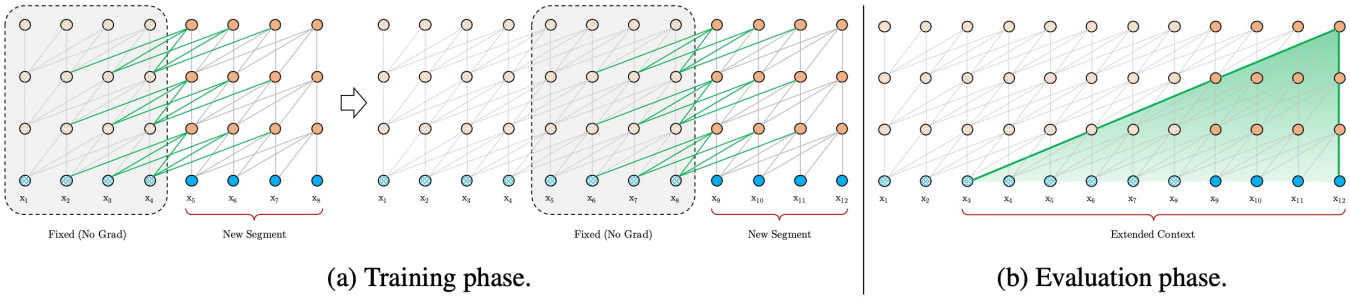
- **Transformer-XL** [Dai et al., 2019]
  - **Idea**: Split long context into segments, and attend to the previous segment
  - Largest possible dependency length becomes **O(network depth x segment length)**
  - With sophisticated implementation, computation becomes **O(input length)**



(a) Train phase.

(b) Evaluation phase.

Figure 1: Illustration of the vanilla model with a segment length 4.



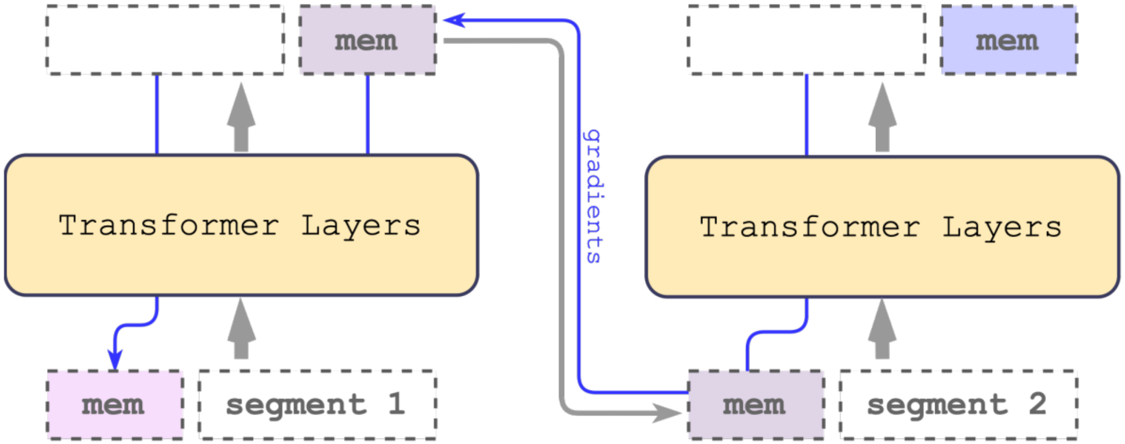
(a) Training phase.

(b) Evaluation phase.

Figure 2: Illustration of the Transformer-XL model with a segment length 4.

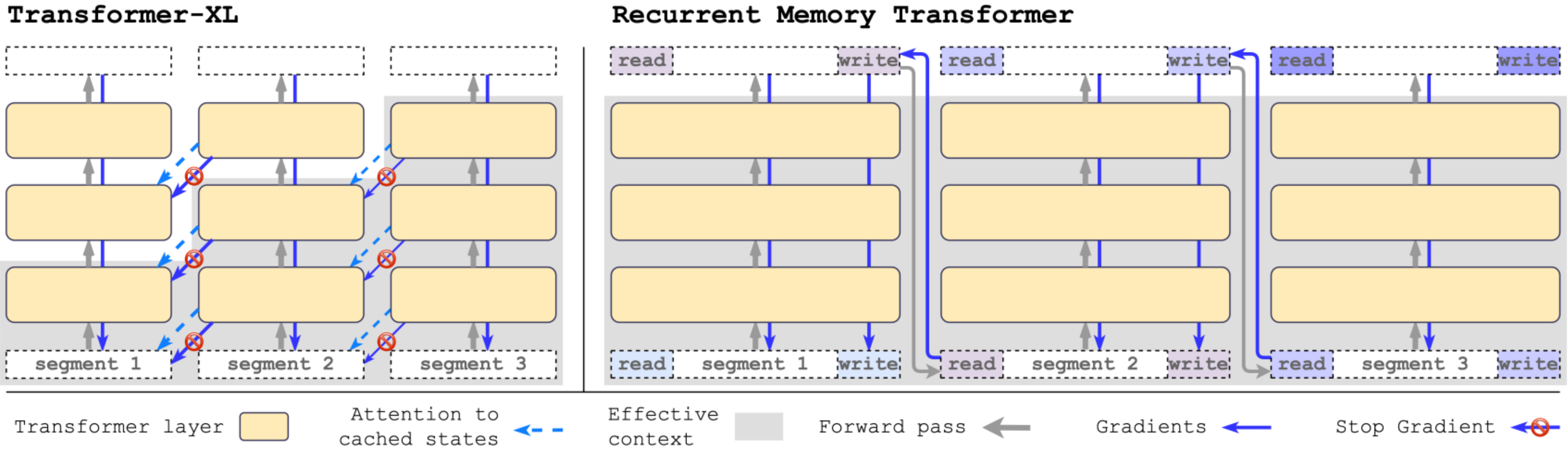
Recurrence-based Methods: Recurrent Memory Transformer

- **Recurrent Memory Transformer (RMT)** [Bulatov et al., 2022]
  - **Idea**: Transformer with token-level memory storage & segment-level recurrence
  - Recurrently compress information in **tokens**, instead of external memory
  - Naturally, computation increases **linearly** with the input length
  - Like RNNs, RMT is trained with **BPTT** (backpropagation through time)



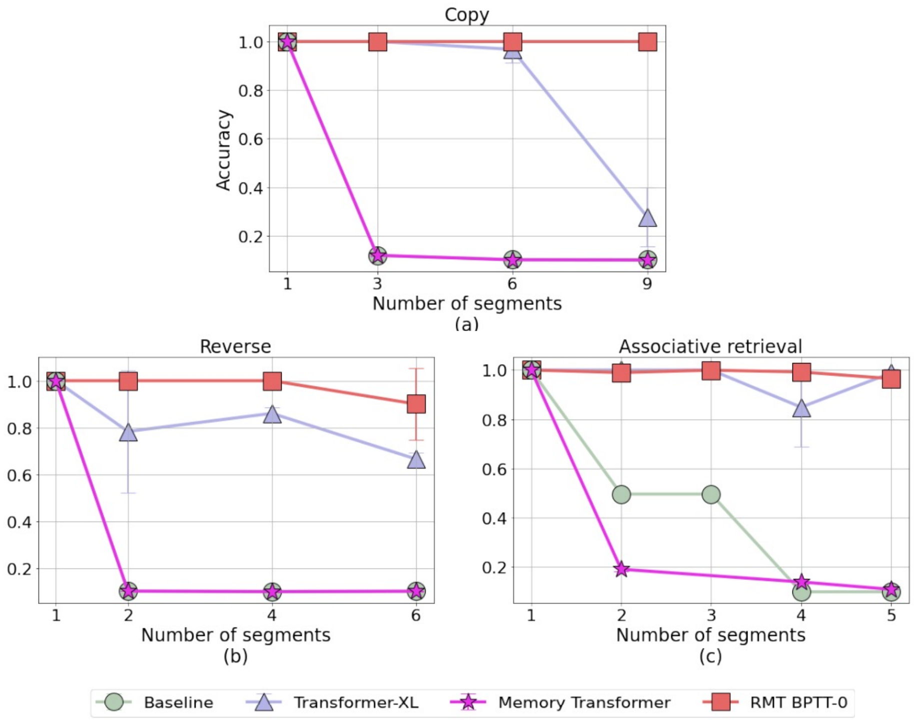
Recurrence-based Methods: Recurrent Memory Transformer

- **Recurrent Memory Transformer (RMT)** [Bulatov et al., 2022]
  - **Idea**: Transformer with token-level memory storage & segment-level recurrence
  - Comparison to Transformer-XL
    - **Unlimited** effective context length
    - **No memory overhead** for maintaining state cache



Recurrence-based Methods: Recurrent Memory Transformer

- **Recurrent Memory Transformer (RMT)** [Bulatov et al., 2022]
  - RMT outperforms Transformer-XL in algorithmic tasks and language modeling
  - Algorithmic tasks:
    - **Copy/Reverse**: Replicating & reversing the input sequence
    - **Associative retrieval**: Key-value retrieval task



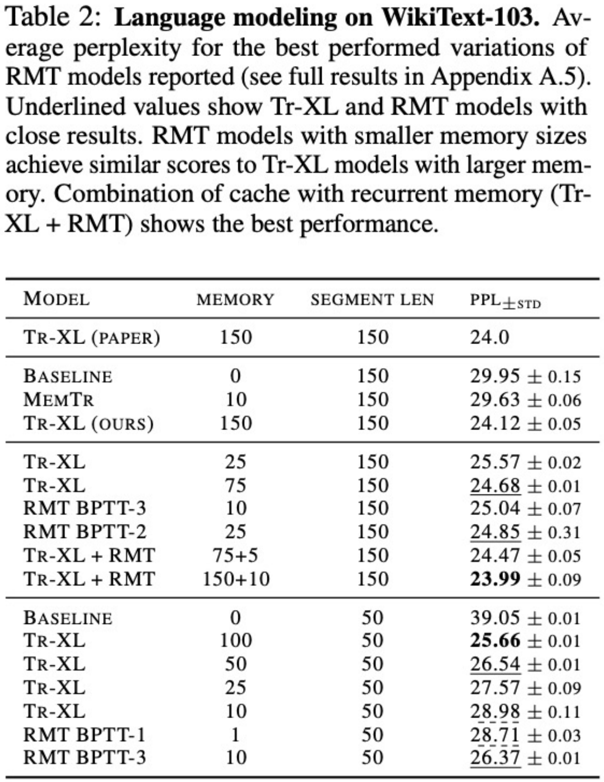
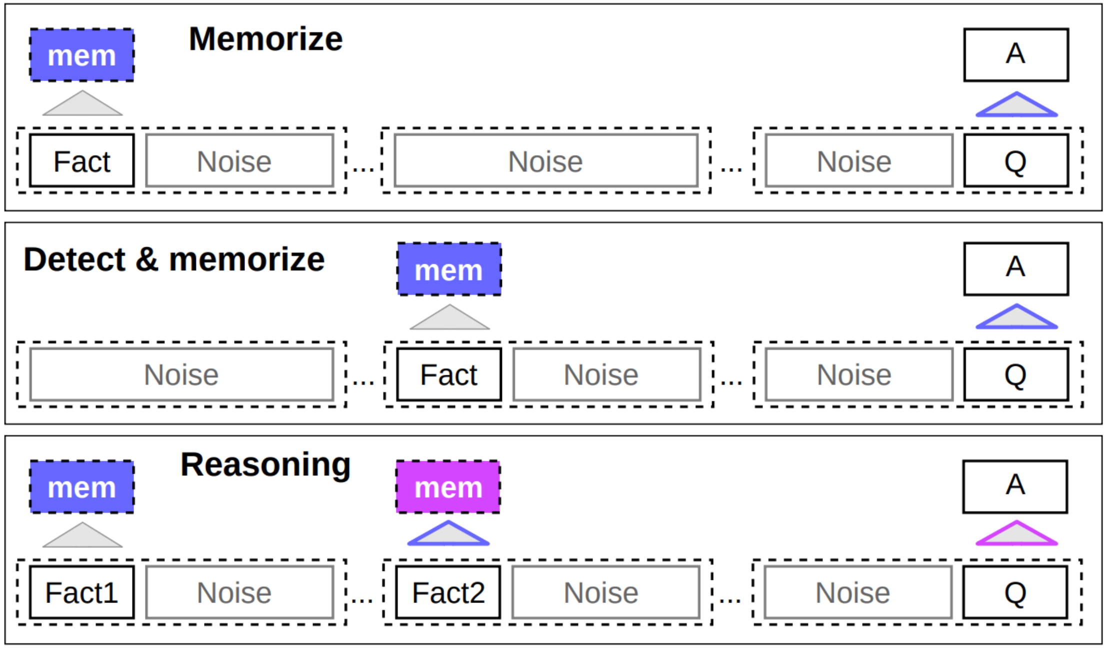


Table 2: **Language modeling on WikiText-103.** Average perplexity for the best performed variations of RMT models reported (see full results in Appendix A.5). Underlined values show Tr-XL and RMT models with close results. RMT models with smaller memory sizes achieve similar scores to Tr-XL models with larger memory. Combination of cache with recurrent memory (Tr-XL + RMT) shows the best performance.

Model	Memory	Segment len	PPL$_{\pm \text{std}}$
Tr-XL (paper)	150	150	24.0
Baseline	0	150	29.95 ± 0.15
MemTr	10	150	29.63 ± 0.06
Tr-XL (ours)	150	150	24.12 ± 0.05
Tr-XL	25	150	25.57 ± 0.02
Tr-XL	75	150	24.68 ± 0.01
RMT BPTT-3	10	150	25.04 ± 0.07
RMT BPTT-2	25	150	24.85 ± 0.31
Tr-XL + RMT	75+5	150	24.47 ± 0.05
Tr-XL + RMT	150+10	150	**23.99** ± 0.09
Baseline	0	50	39.05 ± 0.01
Tr-XL	100	50	**25.66** ± 0.01
Tr-XL	50	50	26.54 ± 0.01
Tr-XL	25	50	27.57 ± 0.09
Tr-XL	10	50	28.98 ± 0.11
RMT BPTT-1	1	50	28.71 ± 0.03
RMT BPTT-3	10	50	26.37 ± 0.01

Recurrence-based Methods: Recurrent Memory Transformer

- **Recurrent Memory Transformer (RMT)** [Bulatov et al., 2022]
  - BERT finetuned with RMT can extrapolate to long inputs over 1M tokens
  - Evaluated with synthetic, memory-intensive tasks
    - Construct long input, with a given fact hidden inside irrelevant text
    - Ask question at the end of the input (6-way classification task)

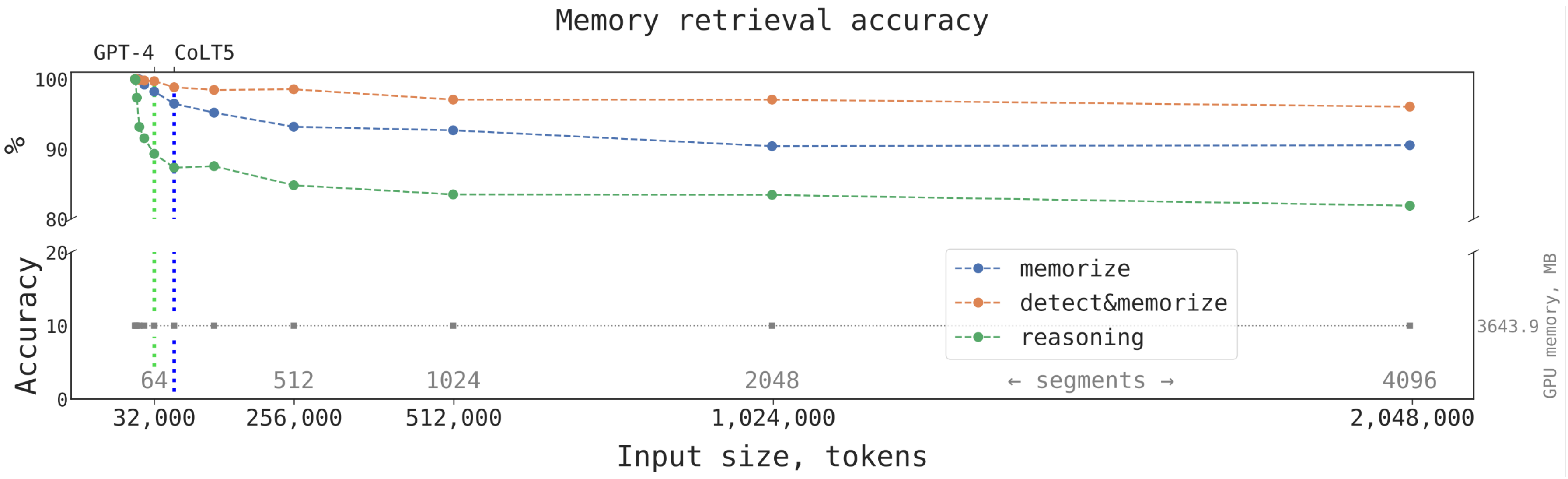


```
Fact: Daniel went back to the hallway.
Question: Where is Daniel?
Answer: hallway
```

```
Fact1: The hallway is east of the bathroom.
Fact2: The bedroom is west of the bathroom.
Question: What is the bathroom east of?
Answer: bedroom
```

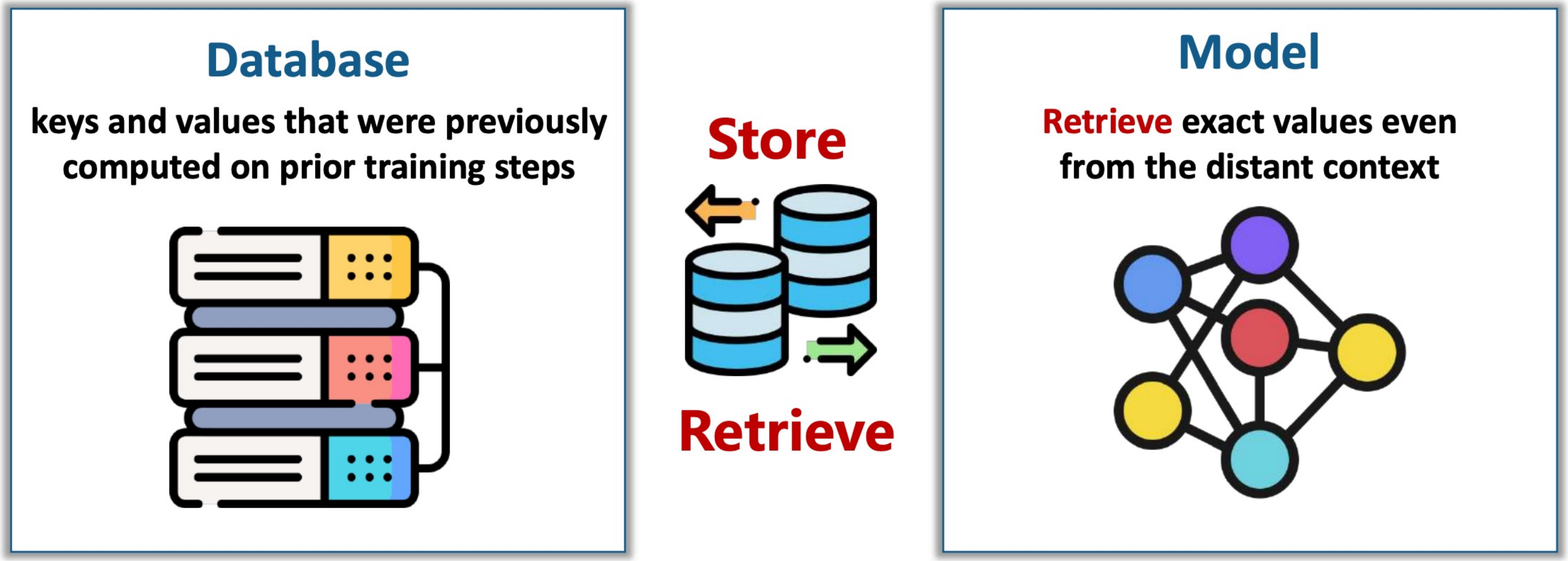
Recurrence-based Methods: Recurrent Memory Transformer

- **Recurrent Memory Transformer (RMT)** [Bulatov et al., 2022]
  - BERT finetuned with RMT can extrapolate to long inputs over 1M tokens
  - Evaluated with synthetic, memory-intensive tasks
    - Construct long input, with a given fact hidden inside irrelevant text
    - Ask question at the end of the input (6-way classification task)



Retrieval-based Methods

- **Retrieval-based Methods**
  - Instead of using the preceding segment's hidden states, store all previous states
  - When processing the current segment, **retrieve** the relevant information
  - Enables **random access** to previous inputs



Database

keys and values that were previously computed on prior training steps

Store

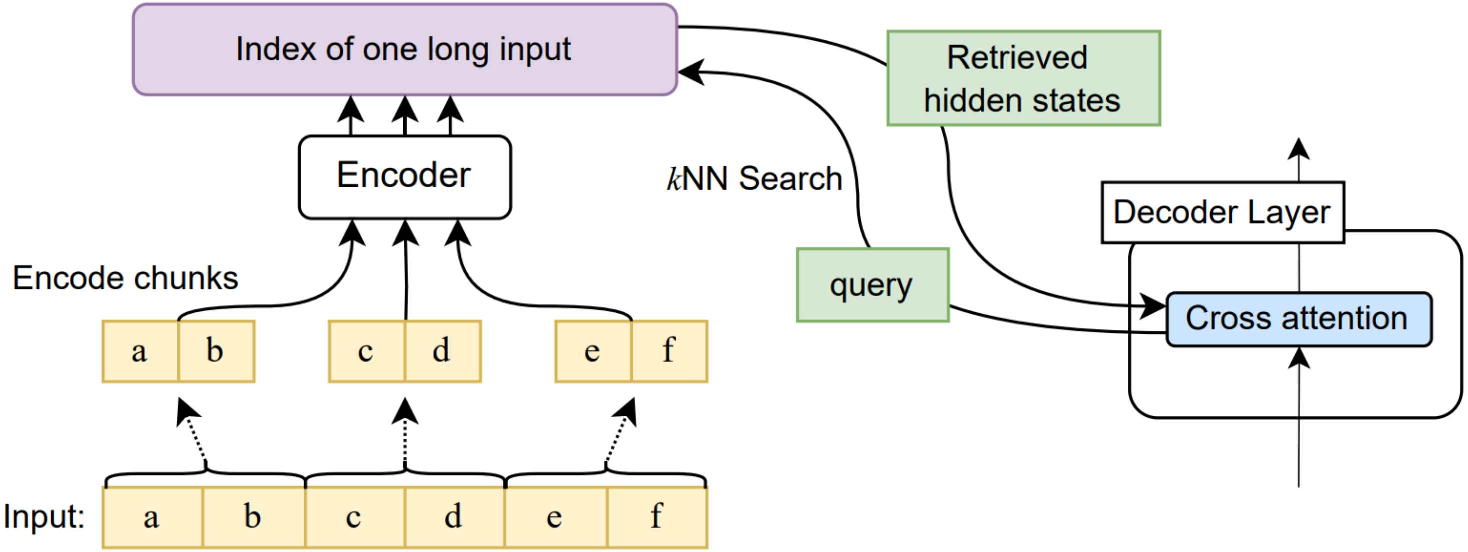
Retrieve

Model

Retrieve exact values even from the distant context

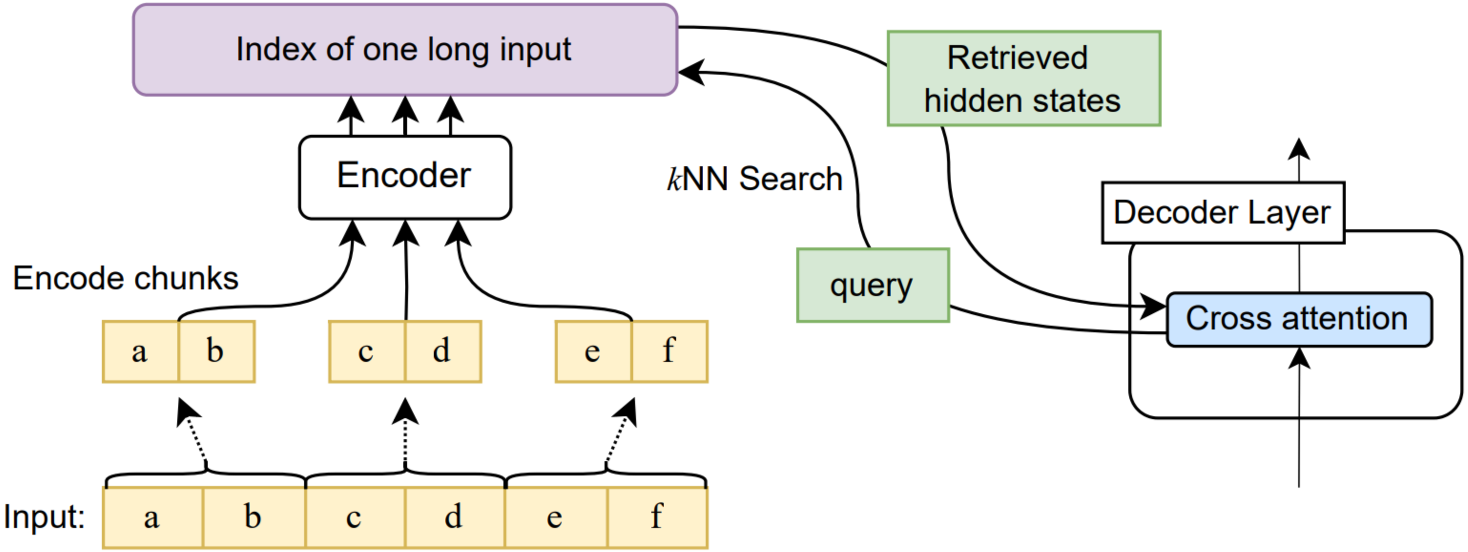
Retrieval-based Methods: Unlimiformer

- **Unlimiformer** [Bertsch et al., 2024]
  - **TL;DR**: Retrieval-based language model that does not require fine-tuning
  - Basic approach is similar to Memory Transformers
    - Separately encode multiple segments, and store the hidden states in external memory
    - Use **kNN search** to retrieve relevant tokens



Retrieval-based Methods: Unlimiformer

- **Unlimiformer** [Bertsch et al., 2024]
  - Unlimiformer applies knn retrieval on **cross-attention**
    - Retrieved tokens serve as **usual encoder outputs**
    - Memory fusion does not require gating, thus introducing **no additional parameter**
    - This allows Unlimiformer to work **without fine-tuning**



Retrieval-based Methods: Unlimiformer

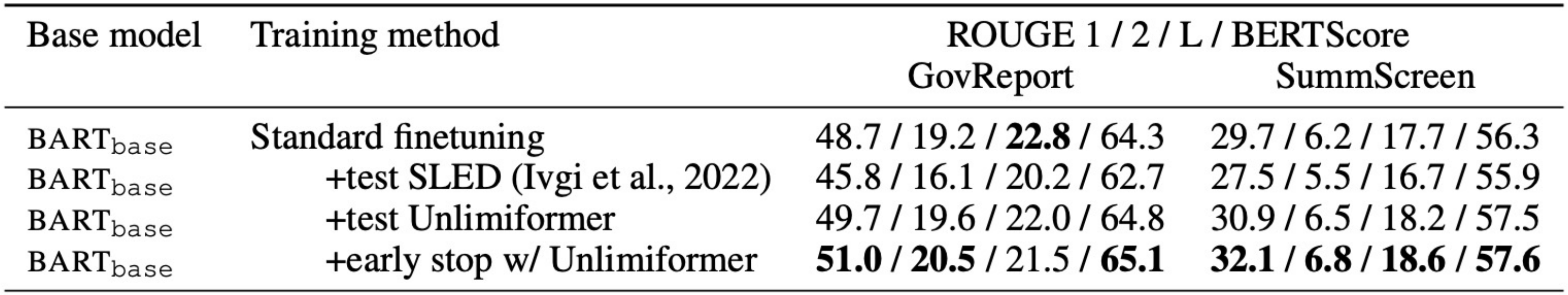
- **Unlimiformer** [Bertsch et al., 2024]
  - **Attention reformulation**
    - Originally, attention keys & values have to be stored for **every layer/attention head**
    - Memory requirement is further reduced through attention reformulation
      - Only a **single vector** has to be stored for every token
  - This allows memory retrieval to take place at **every layer**
    - Note: Memory Transformers apply retrieval on a single layer

$$
\begin{aligned}
QK^T &= (\boldsymbol{h}_d W_q)\,(\boldsymbol{h}_e W_k)^\top \\
&= (\boldsymbol{h}_d W_q)\, W_k^\top \boldsymbol{h}_e{}^\top \\
&= \left(\boldsymbol{h}_d W_q W_k^\top\right) \boldsymbol{h}_e{}^\top
\end{aligned}
$$

Q/K: Query/key vector
W_q/W_k: layer- & head-specific linear layers
h_e: encoder hidden state
h_d: decoder hidden state

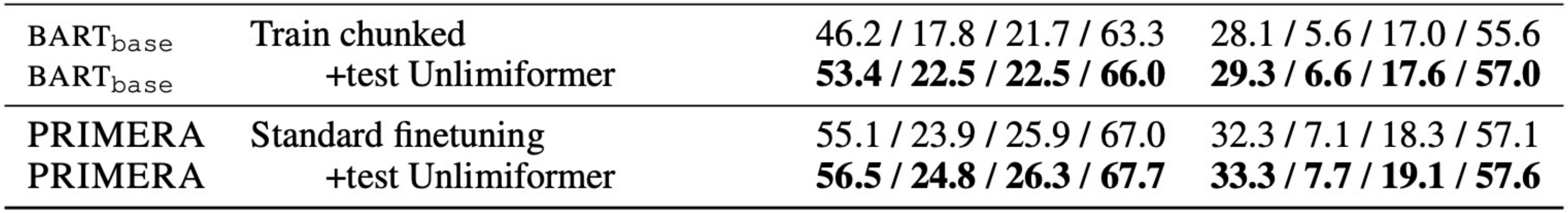
Retrieval-based Methods: Unlimiformer

- **Unlimiformer** [Bertsch et al., 2024]
  - Unlimiformer extends the context limit **without fine-tuning**
    - Evaluated on summarization benchmarks



Base model	Training method	ROUGE 1 / 2 / L / BERTScore	
		GovReport	SummScreen
$\text{BART}_{\text{base}}$	Standard finetuning	48.7 / 19.2 / **22.8** / 64.3	29.7 / 6.2 / 17.7 / 56.3
$\text{BART}_{\text{base}}$	+test SLED (Ivgi et al., 2022)	45.8 / 16.1 / 20.2 / 62.7	27.5 / 5.5 / 16.7 / 55.9
$\text{BART}_{\text{base}}$	+test Unlimiformer	49.7 / 19.6 / 22.0 / 64.8	30.9 / 6.5 / 18.2 / 57.5
$\text{BART}_{\text{base}}$	+early stop w/ Unlimiformer	**51.0** / **20.5** / 21.5 / **65.1**	**32.1** / **6.8** / **18.6** / **57.6**

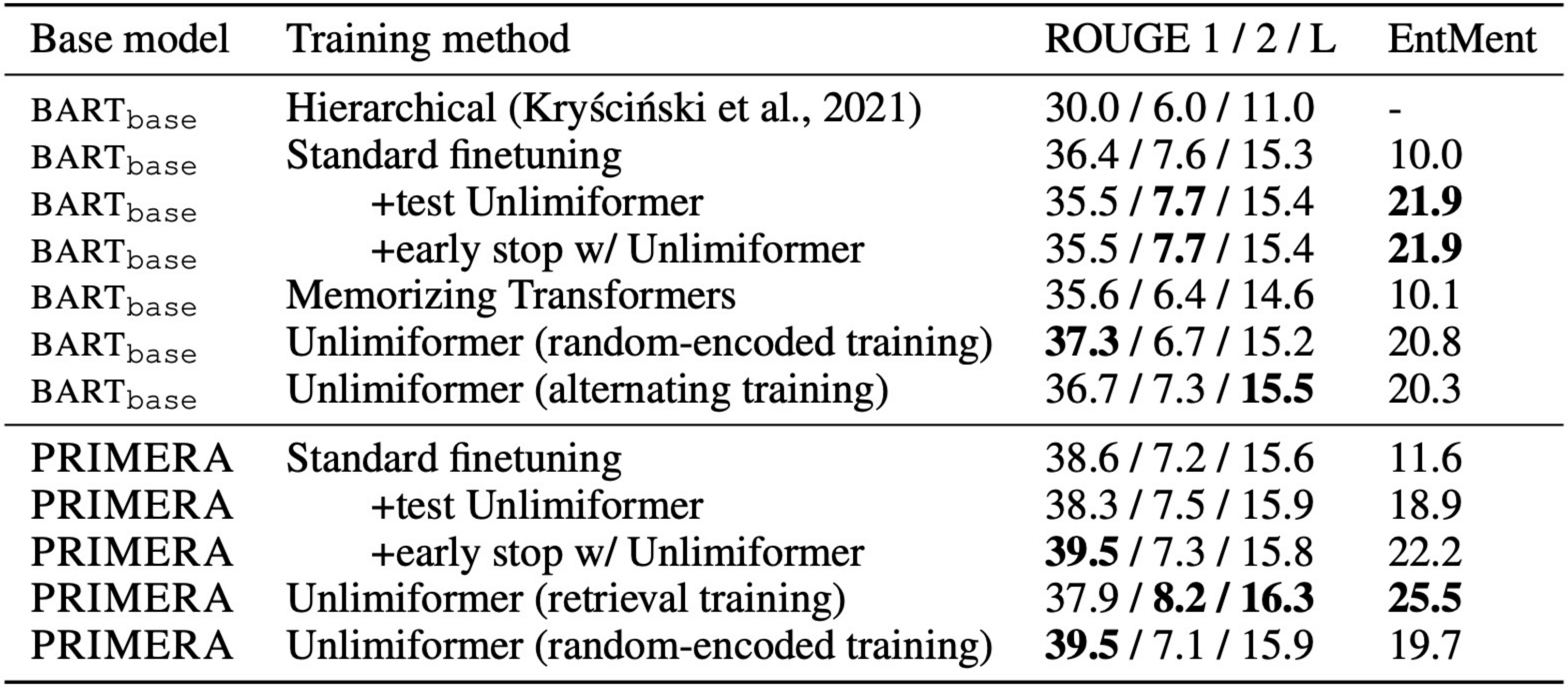
  - Performance is **further improved** with fine-tuning



$\text{BART}_{\text{base}}$	Train chunked	46.2 / 17.8 / 21.7 / 63.3	28.1 / 5.6 / 17.0 / 55.6
$\text{BART}_{\text{base}}$	+test Unlimiformer	**53.4** / **22.5** / **22.5** / **66.0**	**29.3** / **6.6** / **17.6** / **57.0**
PRIMERA	Standard finetuning	55.1 / 23.9 / 25.9 / 67.0	32.3 / 7.1 / 18.3 / 57.1
PRIMERA	+test Unlimiformer	**56.5** / **24.8** / **26.3** / **67.7**	**33.3** / **7.7** / **19.1** / **57.6**

Retrieval-based Methods: Unlimiformer

- **Unlimiformer** [Bertsch et al., 2024]
  - Unlimiformer can handle extremely long (book-level) context
    - Evaluated on BookSum (average input length ~143k tokens)



Base model	Training method	ROUGE 1 / 2 / L	EntMent
$\text{BART}_{\text{base}}$	Hierarchical (Kryściński et al., 2021)	30.0 / 6.0 / 11.0	-
$\text{BART}_{\text{base}}$	Standard finetuning	36.4 / 7.6 / 15.3	10.0
$\text{BART}_{\text{base}}$	+test Unlimiformer	35.5 / **7.7** / 15.4	**21.9**
$\text{BART}_{\text{base}}$	+early stop w/ Unlimiformer	35.5 / **7.7** / 15.4	**21.9**
$\text{BART}_{\text{base}}$	Memorizing Transformers	35.6 / 6.4 / 14.6	10.1
$\text{BART}_{\text{base}}$	Unlimiformer (random-encoded training)	**37.3** / 6.7 / 15.2	20.8
$\text{BART}_{\text{base}}$	Unlimiformer (alternating training)	36.7 / 7.3 / **15.5**	20.3
PRIMERA	Standard finetuning	38.6 / 7.2 / 15.6	11.6
PRIMERA	+test Unlimiformer	38.3 / 7.5 / 15.9	18.9
PRIMERA	+early stop w/ Unlimiformer	**39.5** / 7.3 / 15.8	22.2
PRIMERA	Unlimiformer (retrieval training)	37.9 / **8.2** / **16.3**	**25.5**
PRIMERA	Unlimiformer (random-encoded training)	**39.5** / 7.1 / 15.9	19.7

Handling long context with Transformers

- 3 lines of research for long-context language models:

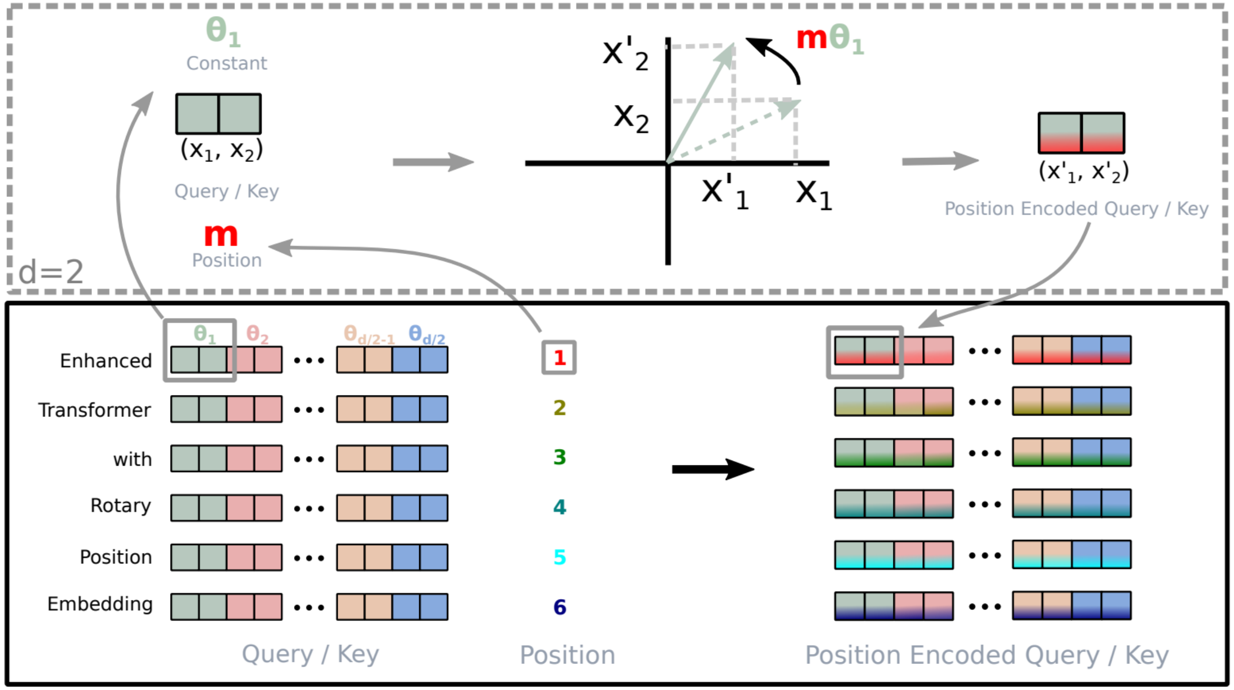
- **Recurrence-based methods**
  - Segments long inputs, and reuses the preceding segment’s hidden states
  - The hidden states serve as ‘memory’ for the current segment

- **Retrieval-based methods**
  - Encodes prior sequences as (key, value) pairs
  - Uses a retrieval algorithm to extract previously encoded information

- **RoPE scaling methods**
  - Applicable to language models utilizing rotary position embeddings (RoPE)
  - Interpolates the position indices, extending the context limit of existing LLMs with minimal or no additional training

RoPE Scaling Methods

- **Rotary Position Embeddings (RoPE)** [Su et al., 2021]
  - **Idea**: Rotate the hidden states according to the token's position
    - View a pair of hidden state values as a **complex number**
    - Rotate each pair by a different frequency θ_i
    - Captures the **relative positional information** between each token
  - One of the most **widely-used positional embeddings** for modern LLMs
    - Including Llama, GPT-NeoX, and PaLM



RoPE Scaling Methods

- **Rotary Position Embeddings (RoPE)** [Su et al., 2021]
  - **Idea**: Rotate the hidden states according to the token's position
  - Mathematical formulation of RoPE is given as follows
    - **m**: position id
    - **W**: linear projection matrix (query, key)
    - **$\mathbf{x}_m$**: hidden states
    - **d**: hidden state dimension

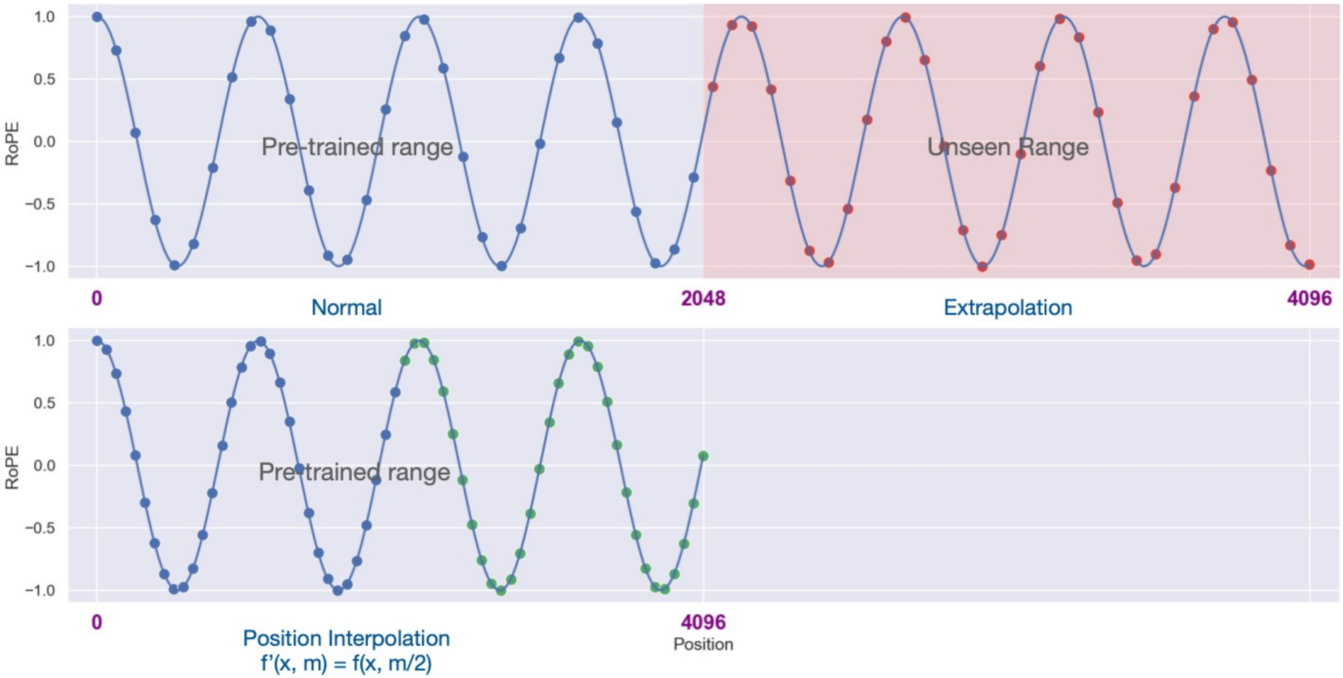
$$f_{\mathbf{W}}(\mathbf{x}_m, m, \theta_i) = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix} \mathbf{W}\mathbf{x}_m$$

$$\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, \cdots, d/2]$$

RoPE Scaling Methods: Linear Interpolation

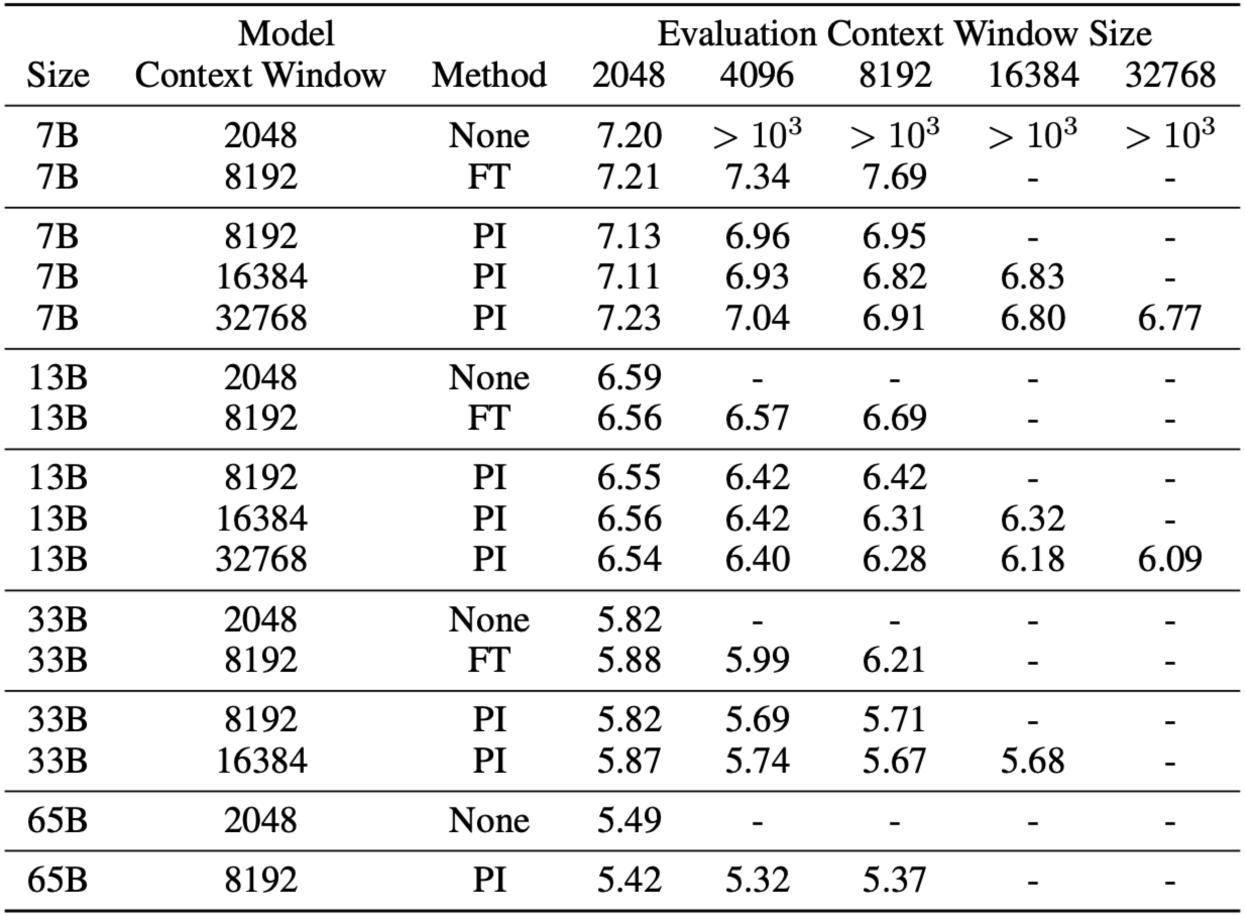
- **Positional Interpolation** (i.e. linear interpolation) [Chen et al., 2023]
  - **Motivation**: Naïvely finetuning LLMs for longer context shows limited success
    - Primarily due to **introduction of new position ids**, unseen while training
  - **Idea**: Instead of extrapolating, interpolate the position ids
    - **L**: Original context length, **L'**: Extended context length

$$f'_{\mathbf{W}}\left(\mathbf{x}_m, m, \theta_i\right) = f_{\mathbf{W}}\left(\mathbf{x}_m, \boxed{\frac{L}{L'}m}, \theta_i\right)$$



RoPE Scaling Methods: Linear Interpolation

- **Positional Interpolation** (i.e. linear interpolation) [Chen et al., 2023]
  - Interpolation effectively extends Llama context length with **minimal training**
    - Fine-tuning possible with ~1000 steps



	Model		Evaluation Context Window Size				
Size	Context Window	Method	2048	4096	8192	16384	32768
7B	2048	None	7.20	$> 10^3$	$> 10^3$	$> 10^3$	$> 10^3$
7B	8192	FT	7.21	7.34	7.69	-	-
7B	8192	PI	7.13	6.96	6.95	-	-
7B	16384	PI	7.11	6.93	6.82	6.83	-
7B	32768	PI	7.23	7.04	6.91	6.80	6.77
13B	2048	None	6.59	-	-	-	-
13B	8192	FT	6.56	6.57	6.69	-	-
13B	8192	PI	6.55	6.42	6.42	-	-
13B	16384	PI	6.56	6.42	6.31	6.32	-
13B	32768	PI	6.54	6.40	6.28	6.18	6.09
33B	2048	None	5.82	-	-	-	-
33B	8192	FT	5.88	5.99	6.21	-	-
33B	8192	PI	5.82	5.69	5.71	-	-
33B	16384	PI	5.87	5.74	5.67	5.68	-
65B	2048	None	5.49	-	-	-	-
65B	8192	PI	5.42	5.32	5.37	-	-

RoPE Scaling Methods: NTK-aware Scaling

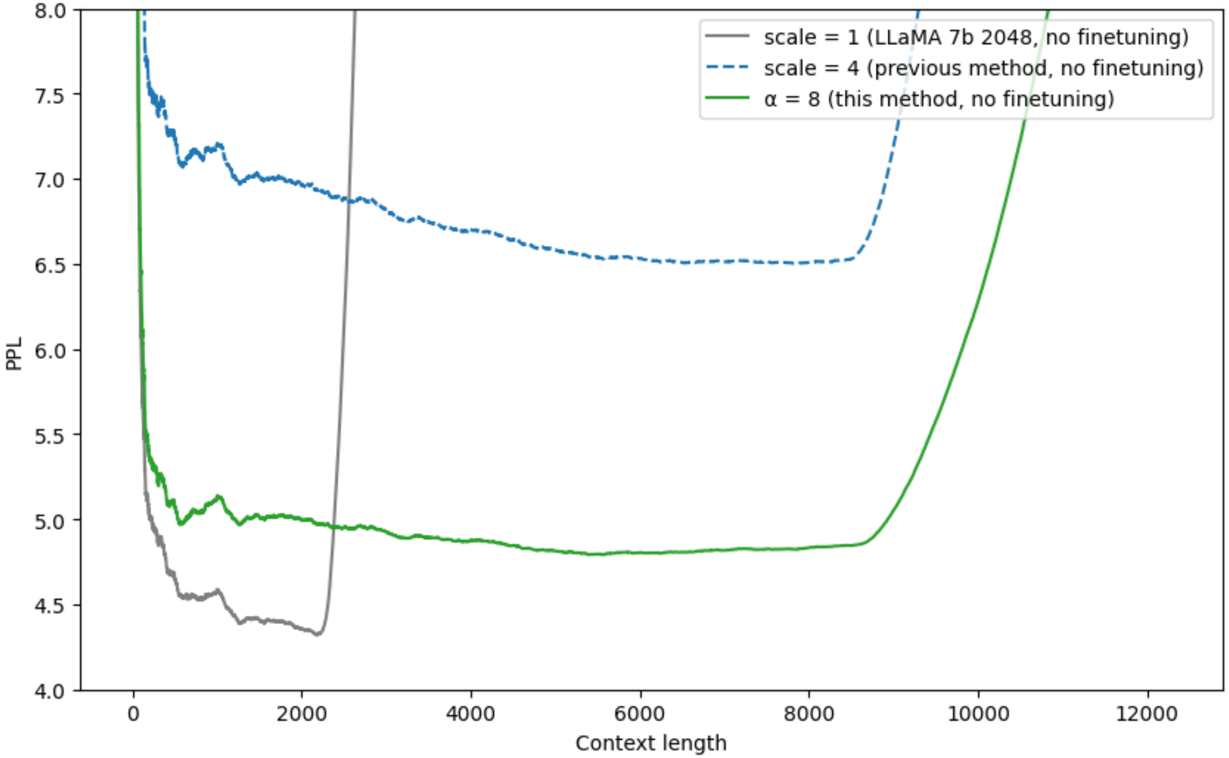
- **NTK-aware Scaling** [bloc97, 2023]
  - **Motivation**: Linear interpolation **sacrifices high-frequency information**
    - Position embeddings of **adjacent tokens** become indistinguishable
    - Such inspiration comes from the neural tangent kernel (NTK) theory
  - A simple solution is to **scale each dimension differently**
    - Position interpolation scales every dimension uniformly by a factor **s** (=L/L')
    - Scale high frequencies more, and low frequencies less
    - This work uses a simple **base scaling** to implement this

$$\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, \cdots, d/2]$$

$$\theta'_i = \underbrace{\left(10000 \times s^{\frac{d}{d-2}}\right)}_{\text{base is scaled according to the scaling factor } \mathbf{s}}{}^{-2(i-1)/d}, i \in [1, 2, \cdots, d/2]$$

RoPE Scaling Methods: NTK-aware Scaling

- **NTK-aware Scaling** [bloc97, 2023]
  - NTK-aware scaling extends the context limit without any finetuning
    - Llama-7b model maintains low perplexity for longer context
  - This scaling method was used for training **Code Llama**, which has 100k context limit



previous method: linear scaling
this method: NTK-aware scaling

RoPE Scaling Methods: NTK-by-parts Scaling

- **NTK-by-parts Scaling** [bloc97, 2023]
  - Investigates deeper into the **different characteristics of each dimension**
  - Key idea is to consider the RoPE embedding wavelength
    - The **number of tokens** needed for the RoPE embedding to perform a **full rotation** (2π).
    - Wavelength for dimension i is defined as follows

$$\lambda_i = \frac{2\pi}{\theta_i} = 2\pi \times 10000^{2(i-1)/d}$$

RoPE Scaling Methods: NTK-by-parts Scaling

- **NTK-by-parts Scaling** [bloc97, 2023]
  - Investigates deeper into the **different characteristics of each dimension**
  - Dimensions with **λ > L** (L: model's original context limit)
    - The embedding does not perform a full rotation
    - Position embeddings are **unique**
      - Intuition: The **absolute positional information** remains intact
  - Dimensions with **λ << L**
    - The embedding rotates multiple times within the context limit
    - Position embeddings are **not unique**
      - Intuition: Only **relative positional information** can be modeled

RoPE Scaling Methods: NTK-by-parts Scaling

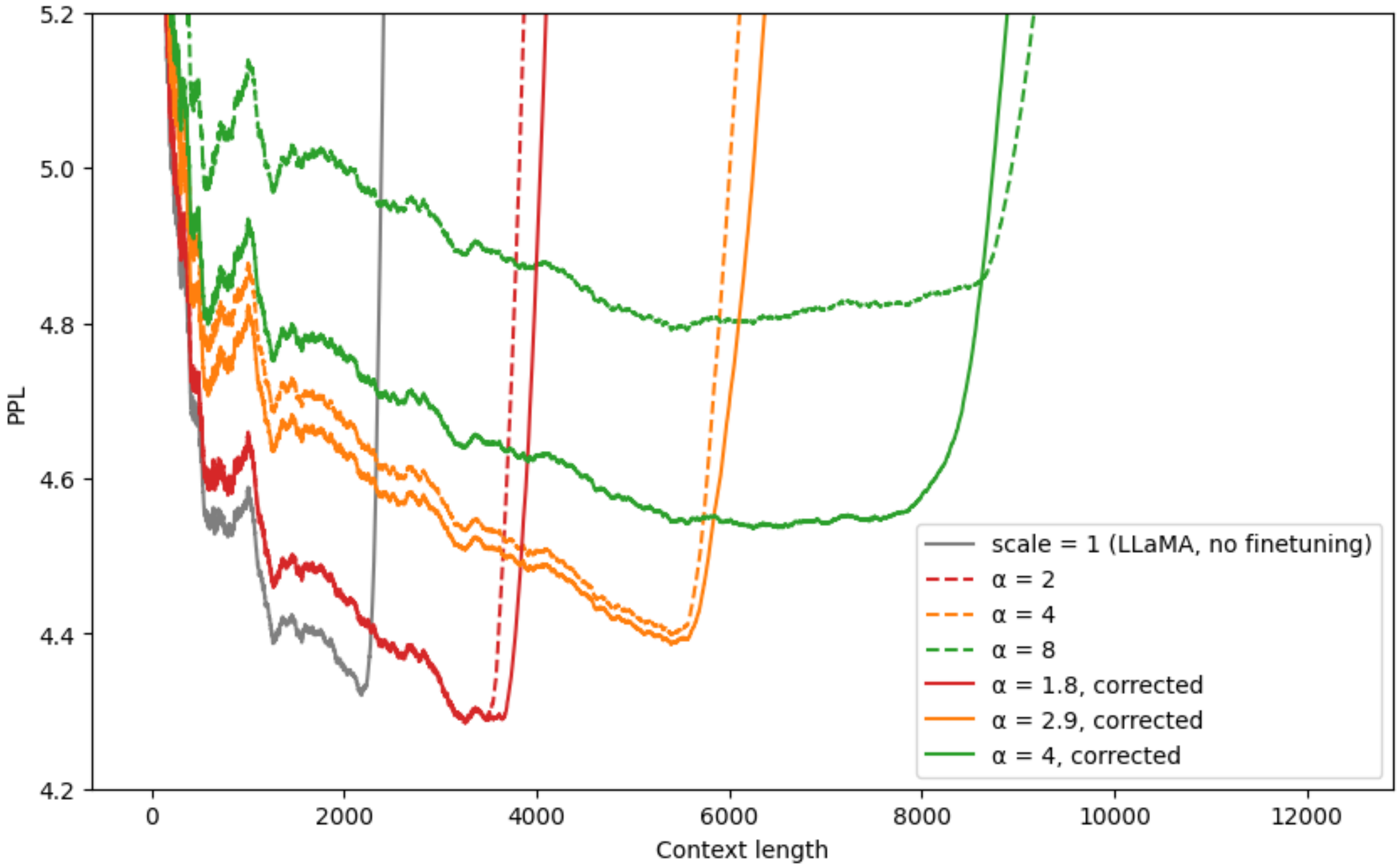
- **NTK-by-parts Scaling** [bloc97, 2023]
  - **Idea**: Interpolate only the dimensions with long wavelengths
    - **L**: Original context length, **L'**: Extended context length
    - α, β are hyperparameters

$$\theta_i' = (1-\gamma_i)\underbrace{\frac{L}{L'}}_{\text{Linear interpolation}}\theta_i + \overbrace{\gamma_i\theta_i}^{\text{No interpolation}} \qquad \gamma_i = \begin{cases} 0, & \text{if } L/\lambda_i < \alpha \\ 1, & \text{if } L/\lambda_i > \beta \\ \frac{L/\lambda_i - \alpha}{\beta - \alpha}, & \text{otherwise} \end{cases}$$

  - Dimensions with **λ > L**
    - These dimensions model the **global** position information
    - Use linear interpolation
  - Dimensions with **λ << L**
    - These dimensions model the **relative** position information
    - Use no interpolation
  - Dimensions in between
    - Use a bit of both methods, by mixing them with a **ramp function**

RoPE Scaling Methods: NTK-by-parts Scaling

- **NTK-by-parts Scaling** [bloc97, 2023]
  - The ‘by-parts’ modification further improves NTK-based scaling



'corrected’ denotes NTK-by-parts

RoPE Scaling Methods: YaRN

- **YaRN** (**Y**et **a**nother **R**oPE Extensio**N** method) [Peng et al., 2024]
  - **TL;DR**: NTK-by-parts with **temperature scaling**
    - Remark: NTK and NTK-by-parts scaling were also formally introduced in this paper
  - NTK-by-parts scaling empirically smooths the attention weights
    - **Scaling the attention weights** by a factor of t helps, making the weights ‘spikier’

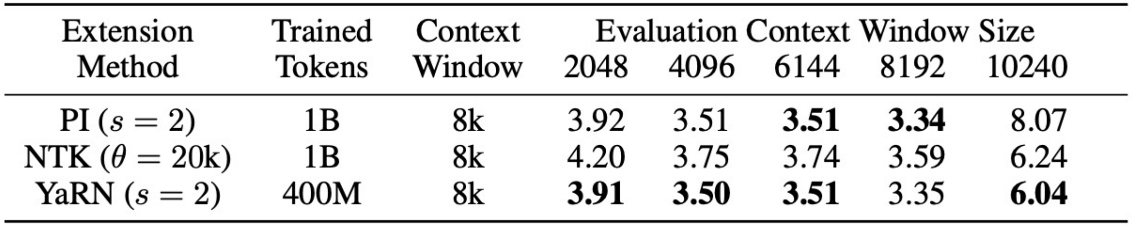
$$\mathrm{softmax}\left(\frac{\boldsymbol{q}_m^T \boldsymbol{k}_n}{t\sqrt{|D|}}\right)$$

    - Empirically, the following formula works well for Llama and Llama-2 models

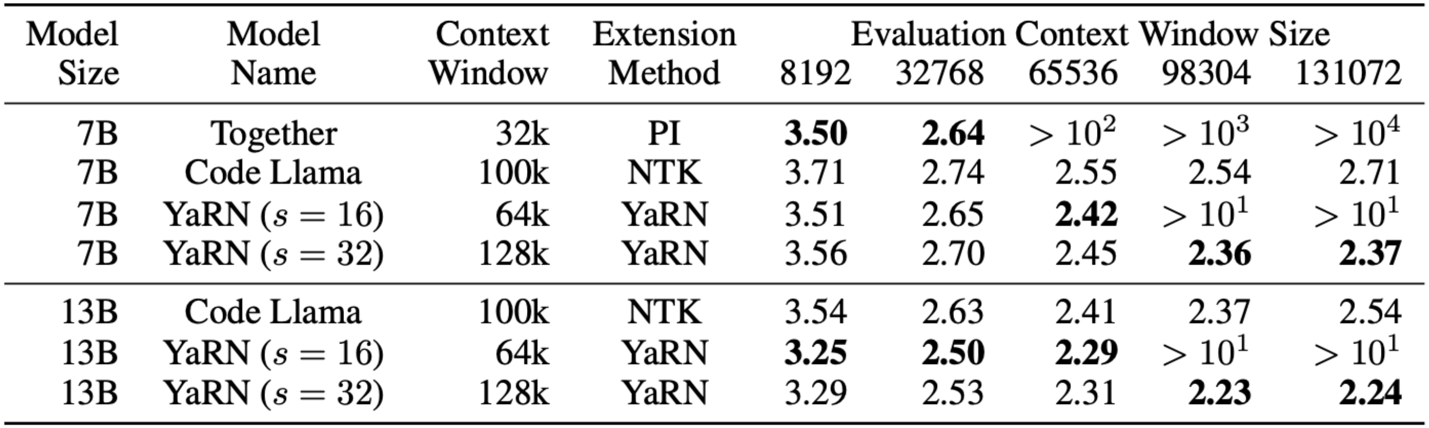
$$\sqrt{\frac{1}{t}} = 0.1\ln(s) + 1$$

RoPE Scaling Methods: YaRN

- **YaRN** (**Y**et **a**nother **R**oPE Extensio**N** method) [Peng et al., 2024]
  - Fine-tuning with YaRN is **efficient** and **effective**
    - Training data consists of 400M tokens (0.1% of original training corpus)
    - Training done with only 400 steps
  - YaRN extends the context limit of Llama-2 to **64k/128k tokens**
    - Original model has a limit of 4k tokens



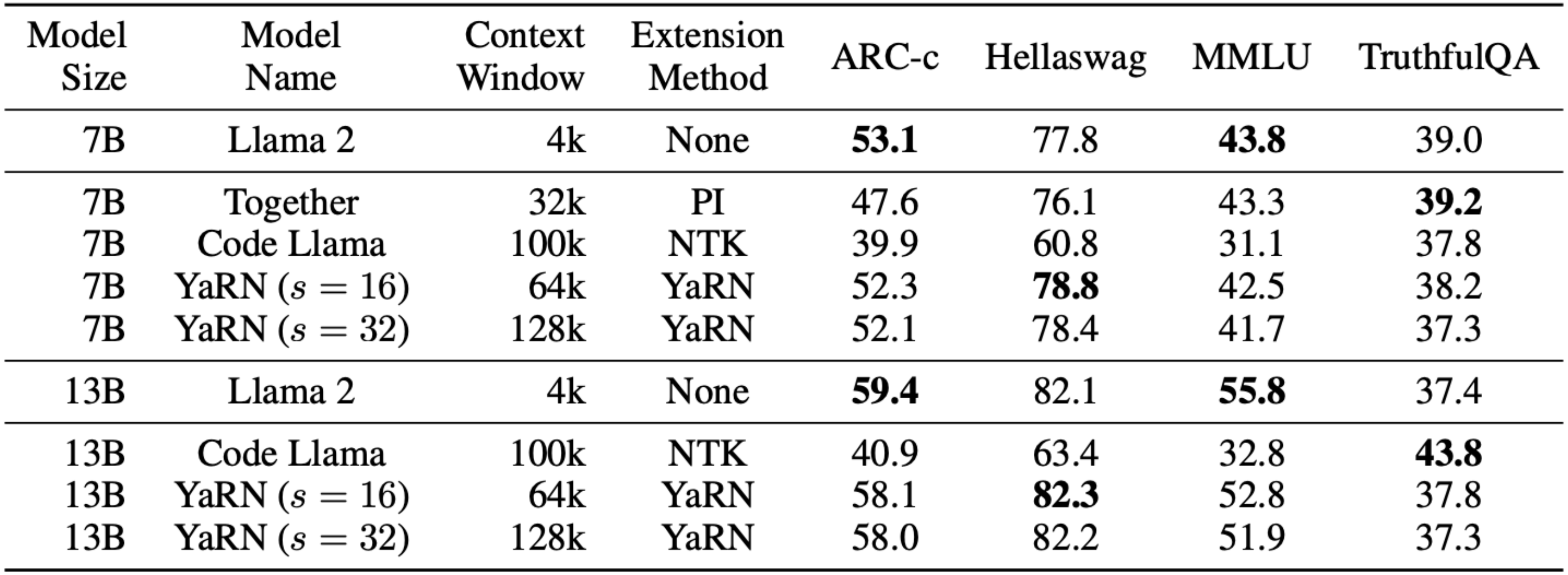
Extension Method	Trained Tokens	Context Window	Evaluation Context Window Size				
			2048	4096	6144	8192	10240
PI ($s = 2$)	1B	8k	3.92	3.51	**3.51**	**3.34**	8.07
NTK ($\theta = 20$k)	1B	8k	4.20	3.75	3.74	3.59	6.24
YaRN ($s = 2$)	400M	8k	**3.91**	**3.50**	**3.51**	3.35	**6.04**



Model Size	Model Name	Context Window	Extension Method	Evaluation Context Window Size				
				8192	32768	65536	98304	131072
7B	Together	32k	PI	**3.50**	**2.64**	$> 10^2$	$> 10^3$	$> 10^4$
7B	Code Llama	100k	NTK	3.71	2.74	2.55	2.54	2.71
7B	YaRN ($s = 16$)	64k	YaRN	3.51	2.65	**2.42**	$> 10^1$	$> 10^1$
7B	YaRN ($s = 32$)	128k	YaRN	3.56	2.70	2.45	**2.36**	**2.37**
13B	Code Llama	100k	NTK	3.54	2.63	2.41	2.37	2.54
13B	YaRN ($s = 16$)	64k	YaRN	**3.25**	**2.50**	**2.29**	$> 10^1$	$> 10^1$
13B	YaRN ($s = 32$)	128k	YaRN	3.29	2.53	2.31	**2.23**	**2.24**

RoPE Scaling Methods: YaRN

- **YaRN** (**Y**et **a**nother **R**oPE Extensio**N** method) [Peng et al., 2024]
  - YaRN-finetuned models show **minimal performance degradation** on short inputs



Model Size	Model Name	Context Window	Extension Method	ARC-c	Hellaswag	MMLU	TruthfulQA
7B	Llama 2	4k	None	**53.1**	77.8	**43.8**	39.0
7B	Together	32k	PI	47.6	76.1	43.3	**39.2**
7B	Code Llama	100k	NTK	39.9	60.8	31.1	37.8
7B	YaRN ($s = 16$)	64k	YaRN	52.3	**78.8**	42.5	38.2
7B	YaRN ($s = 32$)	128k	YaRN	52.1	78.4	41.7	37.3
13B	Llama 2	4k	None	**59.4**	82.1	**55.8**	37.4
13B	Code Llama	100k	NTK	40.9	63.4	32.8	**43.8**
13B	YaRN ($s = 16$)	64k	YaRN	58.1	**82.3**	52.8	37.8
13B	YaRN ($s = 32$)	128k	YaRN	58.0	82.2	51.9	37.3

Overview

Part 1. Basics

- RNN to LSTM
- Sequence-to-sequence Model
- Attention-based NLP Model

Part 2. Transformers and Large Language Models

- Transformer (self-attention)
- Pre-training of Transformers and Language Models

Part 3. Advanced Topics

- Handling long inputs with Transformers
- Techniques for improving efficiency
- State-Space Models

Part 4. Summary

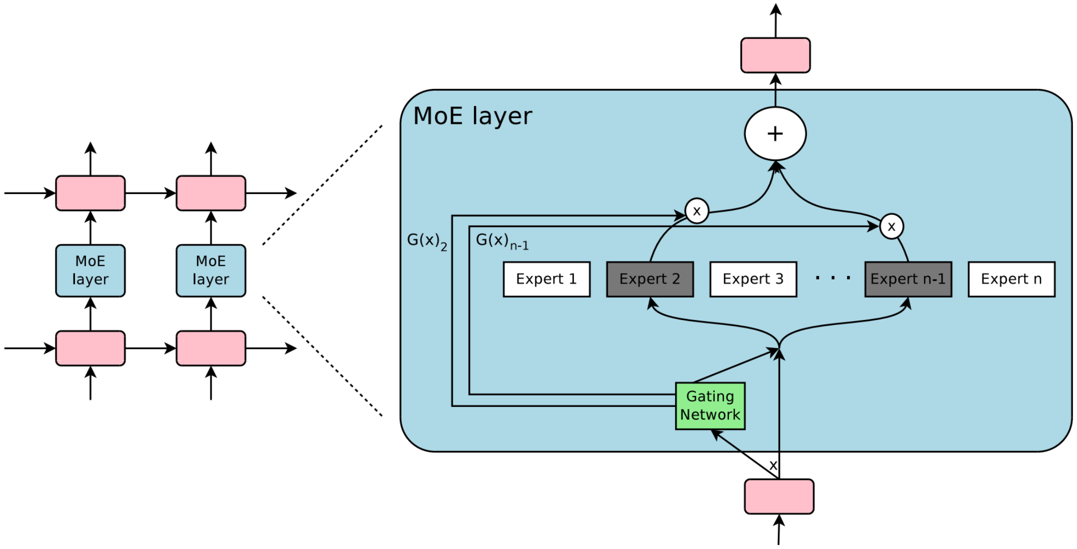
Towards efficient LLMs

- **Scaling** has been the key for the success of modern LLMs
  - However, **large computation** poses a significant bottleneck
  - Improving computational efficiency is critical for further scaling

- Possible solutions include:
  - **Architectures** that enable scaling without severely increasing computation
  - **Further computation optimization** for the current architecture

GLaM: Efficient Scaling of Language Models with Mixture-of-Experts

- **GLaM** (**G**eneralist **La**nguage **M**odel) [Du et al., 2022]
  - **Intuition**: Can we decouple the computation cost from the model size?
  - The key idea is to introduce **Mixture-of-Experts (MoE)** layers
    - Consists of **multiple experts** (simple feed-forward network) and a **gating network**
    - Gating network selects the K 'best' experts for a given input
    - Final output is the weighted sum of each expert's output



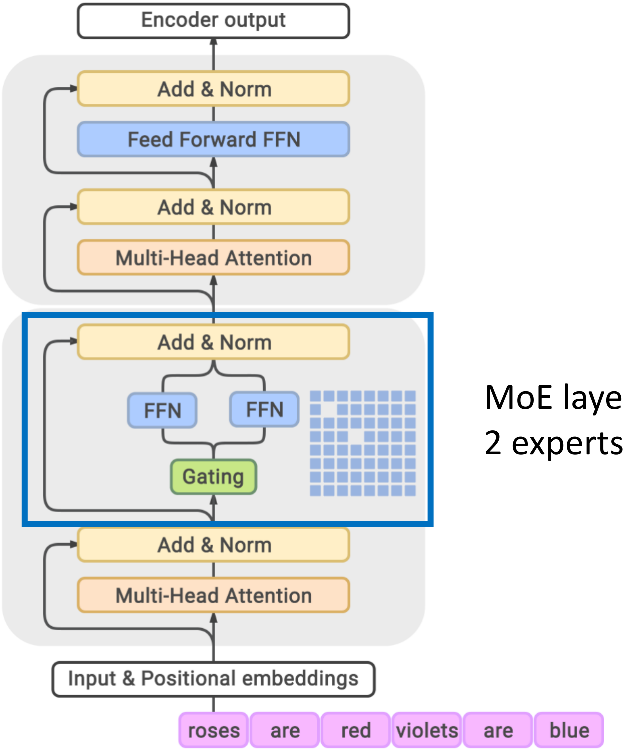
$$G(x) = \mathrm{topK}(\mathrm{Softmax}(x \cdot W_G))$$

$$y = \sum_{i=1}^{K} \underbrace{G(x)_i}_{\text{expert weight}} \cdot \underbrace{E_i(x)}_{\text{expert output}}$$

MoE layer [Shazeer et al., 2017] applied on LSTM

GLaM: Efficient Scaling of Language Models with Mixture-of-Experts

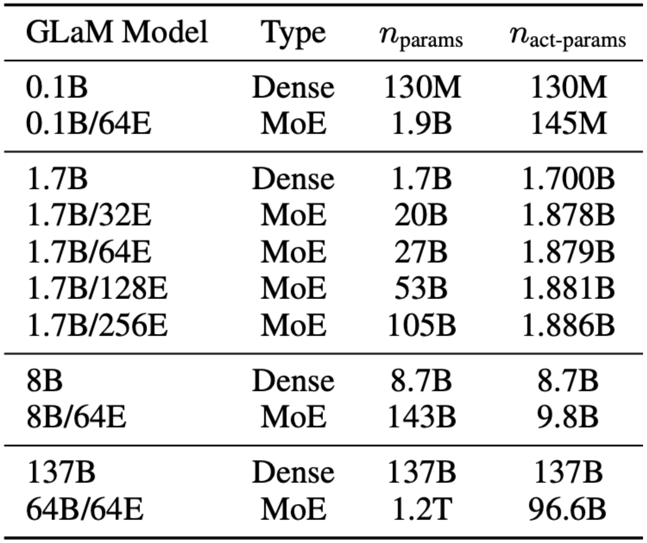
- **GLaM** (**G**eneralist **La**nguage **M**odel) [Du et al., 2022]
  - **Intuition**: Can we decouple the computation cost from the model size?
  - The key idea is to introduce **Mixture-of-Experts (MoE)** layers
    - In this work, MoE layer is applied to the FFN layer of Transformers
    - 2 experts are selected, providing O(number of experts^2) combination of FFN layers, providing more flexibility



MoE layer added to Transformers.
2 experts (FFNs) are selected out of 64.

GLaM: Efficient Scaling of Language Models with Mixture-of-Experts

- **GLaM** (**G**eneralist **La**nguage **M**odel) [Du et al., 2022]
  - Enables efficient scaling, while keeping the computation small
    - Largest GLaM model (64B/64E) has **1.2T parameters**
    - However, only **96.6B are activated** per prediction

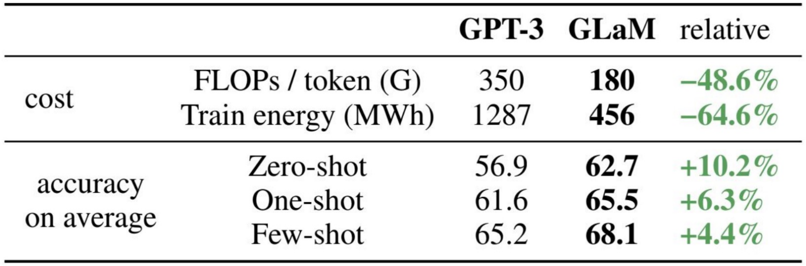


GLaM Model	Type	n_{params}	$n_{\text{act-params}}$
0.1B	Dense	130M	130M
0.1B/64E	MoE	1.9B	145M
1.7B	Dense	1.7B	1.700B
1.7B/32E	MoE	20B	1.878B
1.7B/64E	MoE	27B	1.879B
1.7B/128E	MoE	53B	1.881B
1.7B/256E	MoE	105B	1.886B
8B	Dense	8.7B	8.7B
8B/64E	MoE	143B	9.8B
137B	Dense	137B	137B
64B/64E	MoE	1.2T	96.6B

Number of total parameters and activated parameters
for each GLaM model (base dense size / number of experts)

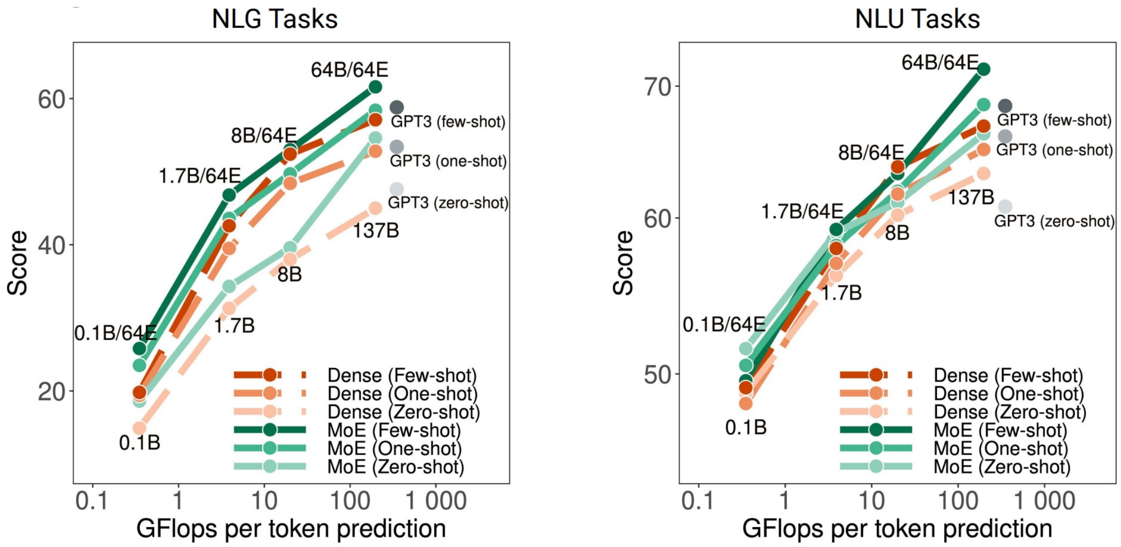
GLaM: Efficient Scaling of Language Models with Mixture-of-Experts

- **GLaM** (**G**eneralist **La**nguage **M**odel) [Du et al., 2022]
  - GLaM **outperforms GPT-3** while requiring **less computation**



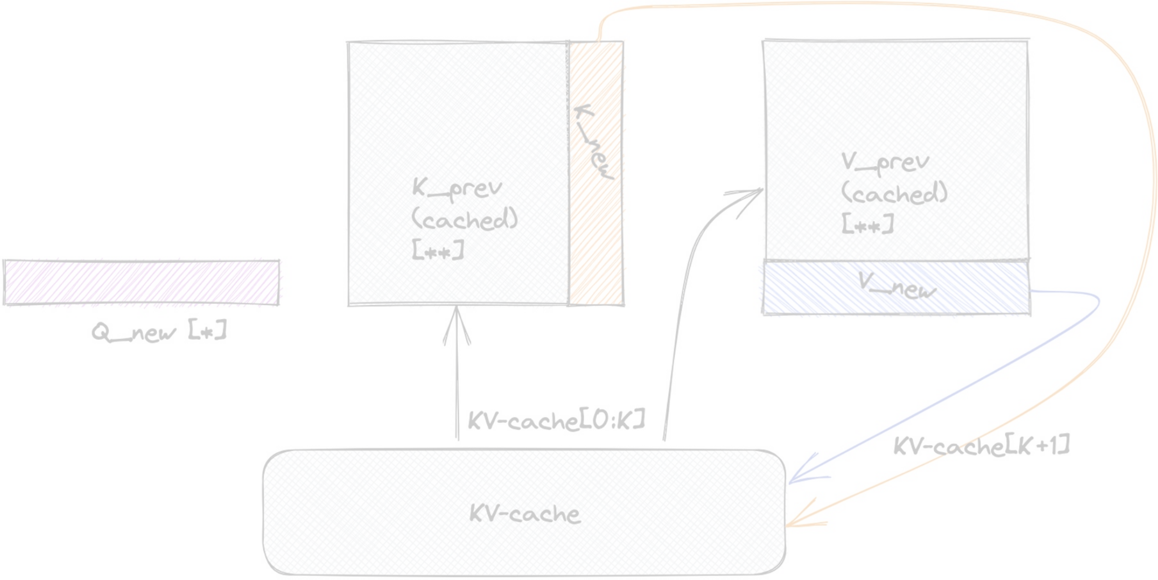
		GPT-3	GLaM	relative
cost	FLOPs / token (G)	350	**180**	−48.6%
	Train energy (MWh)	1287	**456**	−64.6%
accuracy on average	Zero-shot	56.9	**62.7**	+10.2%
	One-shot	61.6	**65.5**	+6.3%
	Few-shot	65.2	**68.1**	+4.4%

  - Models trained with mixture-of-experts **successfully scale**
    - Achieves higher performance using similar FLOPs per token prediction



Multi-head Latent Attention

- **Multi-head Latent Attention** [Liu, Aixin, et al., 2024]
  - **Motivation**: Reduce **KV cache** memory and computation costs while maintaining performance for efficient inference.
  - **KV cache:** Previously computed key and value tensors in attention mechanisms.

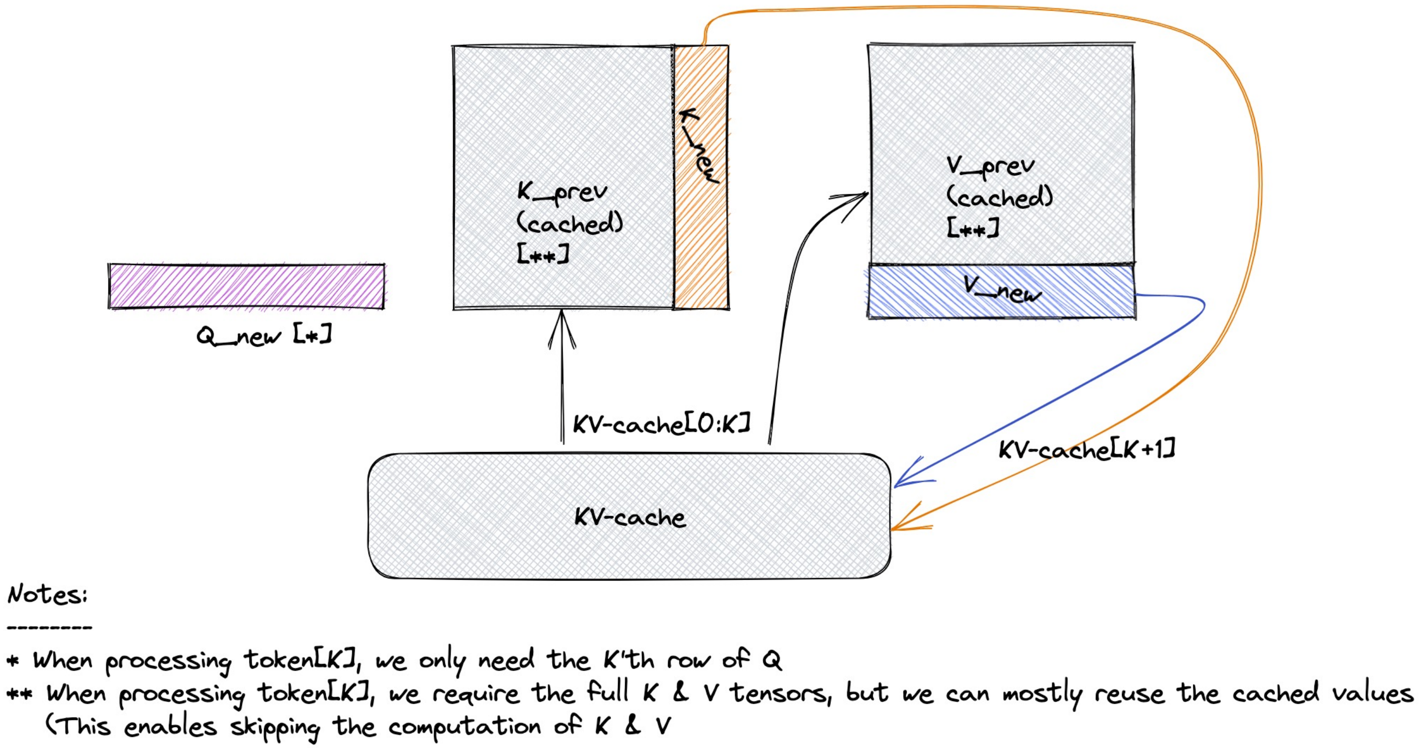


Notes:
----
* When processing token[K], we only need the K'th row of Q
** When processing token[K], we require the full K & V tensors, but we can mostly reuse the cached values
(This enables skipping the computation of K & V

Multi-head Latent Attention

- **Multi-head Latent Attention** [Liu, Aixin, et al., 2024]
  - **Motivation**: Reduce **KV cache** memory and computation costs while maintaining performance for efficient inference.

  - **Note. KV cache:** Previously computed key and value tensors in attention mechanisms.



Notes:

----

* When processing token[K], we only need the K'th row of Q

** When processing token[K], we require the full K & V tensors, but we can mostly reuse the cached values (This enables skipping the computation of K & V

Multi-head Latent Attention

- **Multi-head Latent Attention** [Liu, Aixin, et al., 2024]
  - **Idea**: Jointly compress key and value tensors into a low-rank latent representation and reconstruct them when needed.
  - **Standard Multi-Head Attention:**
    - Let d be the embedding dim., n_h be the number of heads, d_h be the dim. per head.
    - $q_t = W^Q h_t,\ k_t = W^K h_t,\ v_t = W^V h_t$, where $W^Q, W^K, W^V \in \mathbb{R}^{d_h n_h \times d}$
  - **Latent compression:**
    - $c_t^{KV} = W^{DKV} h_t$, where $W^{DKV} \in \mathbb{R}^{d_c \times d}$
    - Compress the hidden state h_t into a compressed latent vector c_t^{KV}.
  - **Key-value reconstruction:**
    - $k_t^C = W^{UK} c_t^{KV},\ v_t^C = W^{UV} c_t^{KV}$, where $W^{UK}, W^{UV} \in \mathbb{R}^{d_h n_h \times d_c}$
    - Reconstruct the key and value vector from the compressed latent vector.
  - **No need to compute Keys and Values during inference:**
    - **Attention Computation:**
      - $\mathrm{Attention}_i = \mathrm{Softmax}\left(\frac{h^T W^{QT} W^K h}{\sqrt{d}}\right) W^V h = \mathrm{Softmax}\left(\frac{h^T W^{QT} W^{UK} c^{KV}}{\sqrt{d}}\right) W^{UV} c^{KV}$
      - $\mathrm{Output} = W^O [\mathrm{Attention}_{1:n_h}]$, where [] is concatenation
    - W^{UK} can be absorbed to W^Q
    - W^{UV} can be absorbed to W^O, where W^O is the output projection matrix

Multi-head Latent Attention

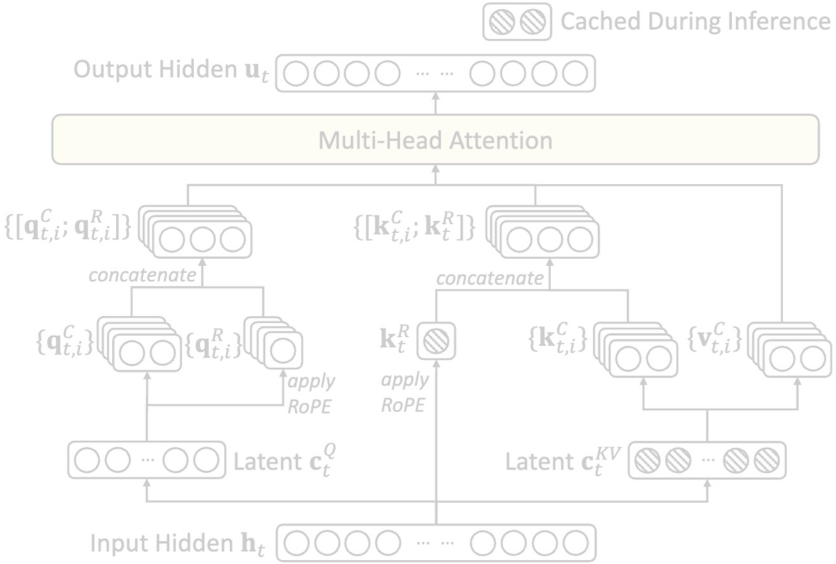
- **Multi-head Latent Attention** [Liu, Aixin, et al., 2024]
  - **Idea**: Jointly compress key and value tensors into a low-rank latent representation and reconstruct them when needed.
  - **Standard Multi-Head Attention:**
    - Let d be the embedding dim., n_h be the number of heads, d_h be the dim. per head.
    - $q_t = W^Q h_t,\ k_t = W^K h_t,\ v_t = W^V h_t$, where $W^Q, W^K, W^V \in \mathbb{R}^{d_h n_h \times d}$
  - **Latent compression:**
    - $c_t^{KV} = W^{DKV} h_t$, where $W^{DKV} \in \mathbb{R}^{d_c \times d}$
    - Compress the hidden state h_t into a compressed latent vector c_t^{KV}.
  - **Key-value reconstruction:**
    - $k_t^C = W^{UK} c_t^{KV},\ v_t^C = W^{UV} c_t^{KV}$, where $W^{UK}, W^{UV} \in \mathbb{R}^{d_h n_h \times d_c}$
    - Reconstruct the key and value vector from the compressed latent vector.
  - **No need to compute Keys and Values during inference:**
    - **Attention Computation:**
      - $\mathrm{Attention}_i = \mathrm{Softmax}\left(\frac{h^T W^{QT} W^K h}{\sqrt{d}}\right) W^V h = \mathrm{Softmax}\left(\frac{h^T W^{QT} W^{UK} c^{KV}}{\sqrt{d}}\right) W^{UV} c^{KV}$
      - $\mathrm{Output} = W^O [\mathrm{Attention}_{1:n_h}]$, where [] is concatenation
    - W^{UK} can be absorbed to W^Q
    - W^{UV} can be absorbed to W^O, where W^O is the output projection matrix

Multi-head Latent Attention

- **Multi-head Latent Attention** [Liu, Aixin, et al., 2024]
  - **Idea**: Jointly compress key and value tensors into a low-rank latent representation and reconstruct them when needed.
  - **Standard Multi-Head Attention:**
    - Let d be the embedding dim., n_h be the number of heads, d_h be the dim. per head.
    - $q_t = W^Q h_t,\ k_t = W^K h_t,\ v_t = W^V h_t$, where $W^Q, W^K, W^V \in \mathbb{R}^{d_h n_h \times d}$
  - **Latent compression:**
    - $c_t^{KV} = W^{DKV} h_t$, where $W^{DKV} \in \mathbb{R}^{d_c \times d}$
    - Compress the hidden state h_t into a compressed latent vector c_t^{KV}.
  - **Key-value reconstruction:**
    - $k_t^C = W^{UK} c_t^{KV},\ v_t^C = W^{UV} c_t^{KV}$, where $W^{UK}, W^{UV} \in \mathbb{R}^{d_h n_h \times d_c}$
    - Reconstruct the key and value vector from the compressed latent vector.
  - **No need to compute Keys and Values during inference:**
    - **Attention Computation:**
      - $\text{Attention}_i = \text{Softmax}\left(\frac{h^T W^{QT} W^K h}{\sqrt{d}}\right) W^V h = \text{Softmax}\left(\frac{h^T W^{QT} W^{UK} c^{KV}}{\sqrt{d}}\right) W^{UV} c^{KV}$
      - $\text{Output} = W^O [\text{Attention}_{1:n_h}]$, where [] is concatenation
    - W^{UK} can be absorbed to W^Q
    - W^{UV} can be absorbed to W^O, where W^O is the output projection matrix

Multi-head Latent Attention

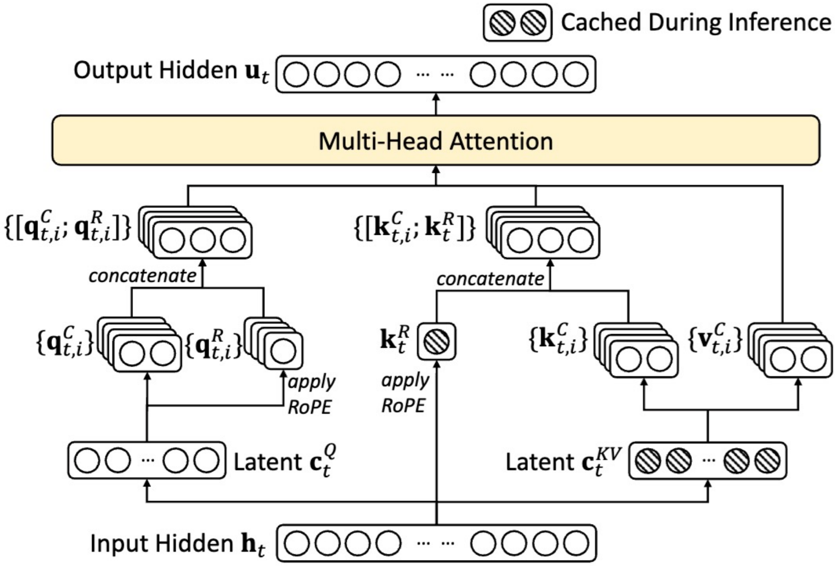
- **Multi-head Latent Attention** [Liu, Aixin, et al., 2024]
  - **However, RoPE is not compatible with matrix absorption.**
    - W^{UK} cannot be absorbed into W^{Q} anymore during the inference.
      - Since the RoPE matrix will lie between W^{UK} and W^{Q}, and RoPE weight will vary for all tokens.
      - $\text{Attention}_i = \text{Softmax}\left(\frac{RoPE\left(h^T W^Q\right)^T RoPE\left(W^{UK} c^{KV}\right)}{\sqrt{d}}\right) W^{UV} c^{KV}$
    - Thus, we must **recompute the keys for all tokens** during inference.



An illustration of MLA

Multi-head Latent Attention

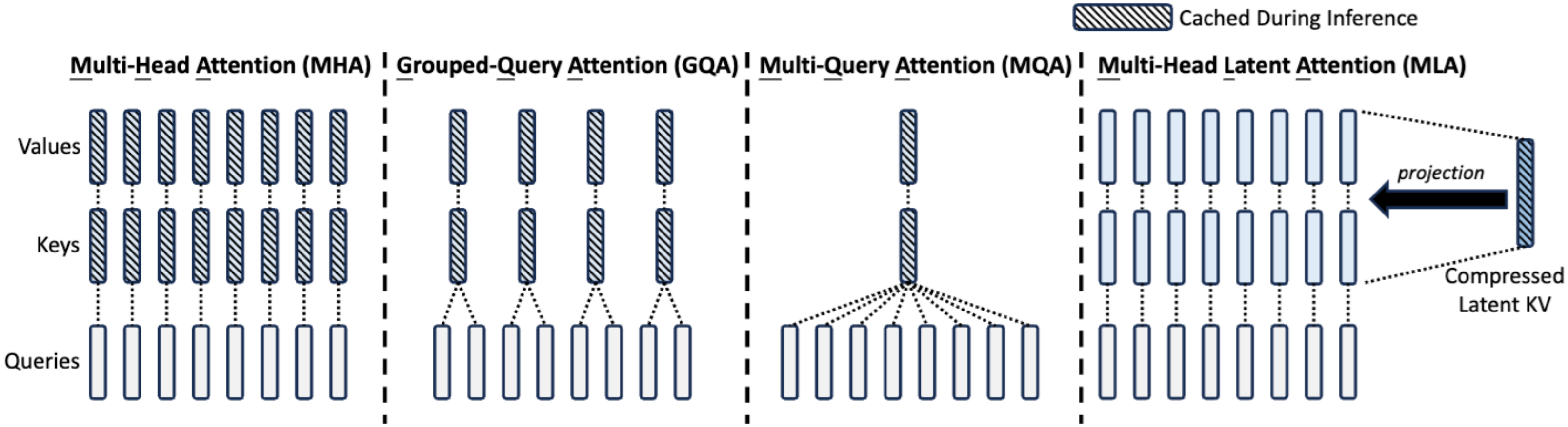
- **Multi-head Latent Attention** [Liu, Aixin, et al., 2024]
  - **Decoupled Rotary Position Embedding (RoPE)**
    - **Idea:** 1. **Partially apply the PE** for the query and keys
      2. **Share the key carrying PE** across the heads
    - Additional multi-head queries and shared key carrying RoPE:
      - $q_t^R = RoPE\left(W^{QR} c_t^Q\right),\ k_t^R = RoPE(W^{KR} h_t), W^{QR} \in \mathbb{R}^{d_h^R n_h \times d_c'}, W^{KR} \in \mathbb{R}^{d_h^R \times d_c}$
        where $d_h^R$is the per-head dim. of the decoupled q and k
    - Then, concatenate with query and keys:
      - $q_{t,i} = [q_{t,i}^C; q_{t.i}^R], k_{t,i} = \left[k_{t,i}^C; k_t^R\right]$



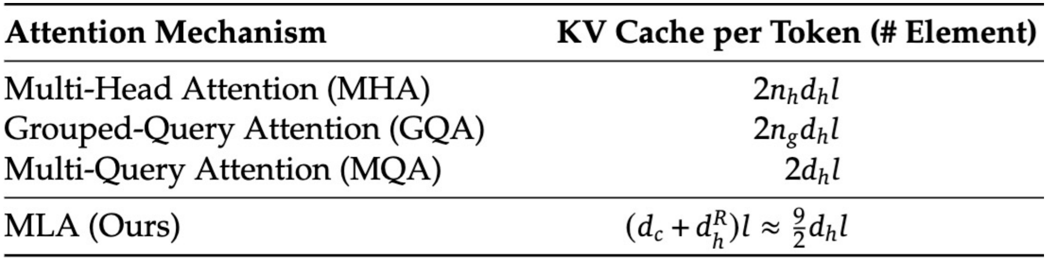
An illustration of MLA

Multi-head Latent Attention

- **Multi-head Latent Attention** [Liu, Aixin, et al., 2024]
  - **Comparing with existing methods**
    - MLA can significantly reduce the memory requirement compared to MHA, and GQA.



Simplified illustration of MHA, GQA, MQA, and MLA

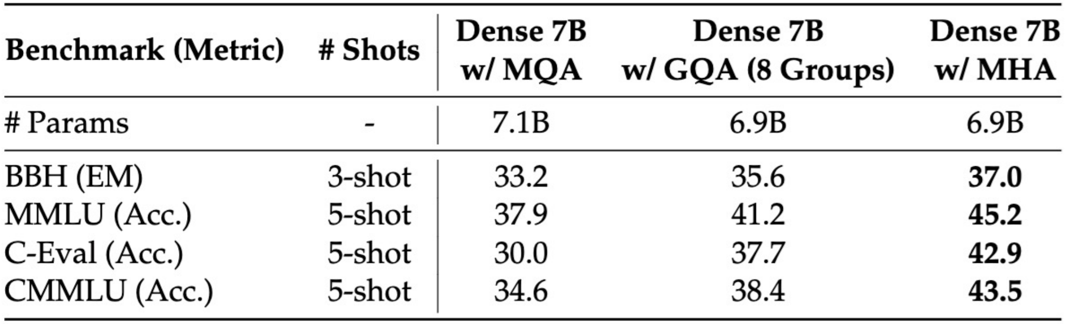


Attention Mechanism	KV Cache per Token (# Element)
Multi-Head Attention (MHA)	$2n_h d_h l$
Grouped-Query Attention (GQA)	$2n_g d_h l$
Multi-Query Attention (MQA)	$2d_h l$
MLA (Ours)	$(d_c + d_h^R)l \approx \frac{9}{2}d_h l$

KV cache per Token

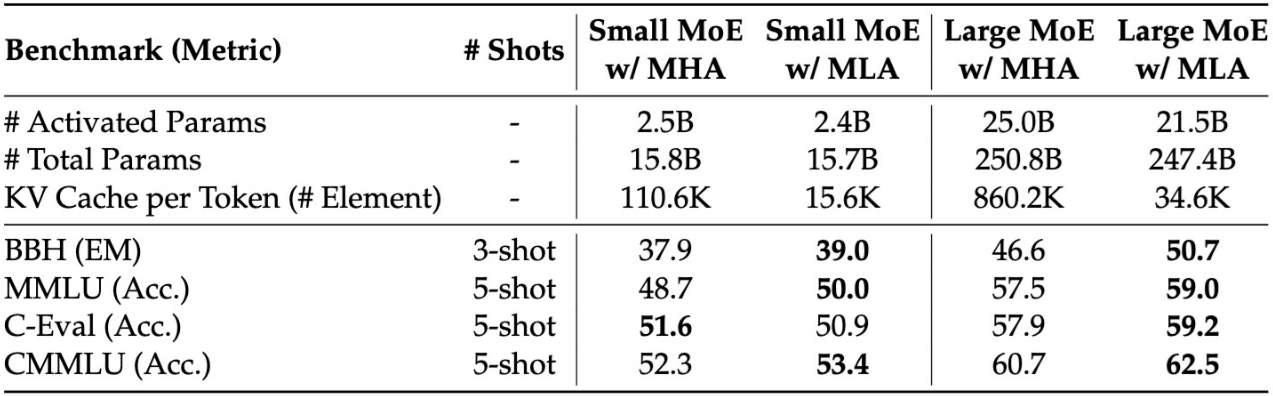
Multi-head Latent Attention

- **Multi-head Latent Attention** [Liu, Aixin, et al., 2024]
    - **Comparing with existing methods**
        - Both MQA, and GQA underperforms MHA in all benchmarks
        - However, MLA surpasses MHA on most benchmarks with different model sizes.
            - **While requiring significantly smaller amount of KV cache**



Benchmark (Metric)	**# Shots**	**Dense 7B w/ MQA**	**Dense 7B w/ GQA (8 Groups)**	**Dense 7B w/ MHA**
# Params	-	7.1B	6.9B	6.9B
BBH (EM)	3-shot	33.2	35.6	**37.0**
MMLU (Acc.)	5-shot	37.9	41.2	**45.2**
C-Eval (Acc.)	5-shot	30.0	37.7	**42.9**
CMMLU (Acc.)	5-shot	34.6	38.4	**43.5**

Comparing MQA, GQA, and MHA

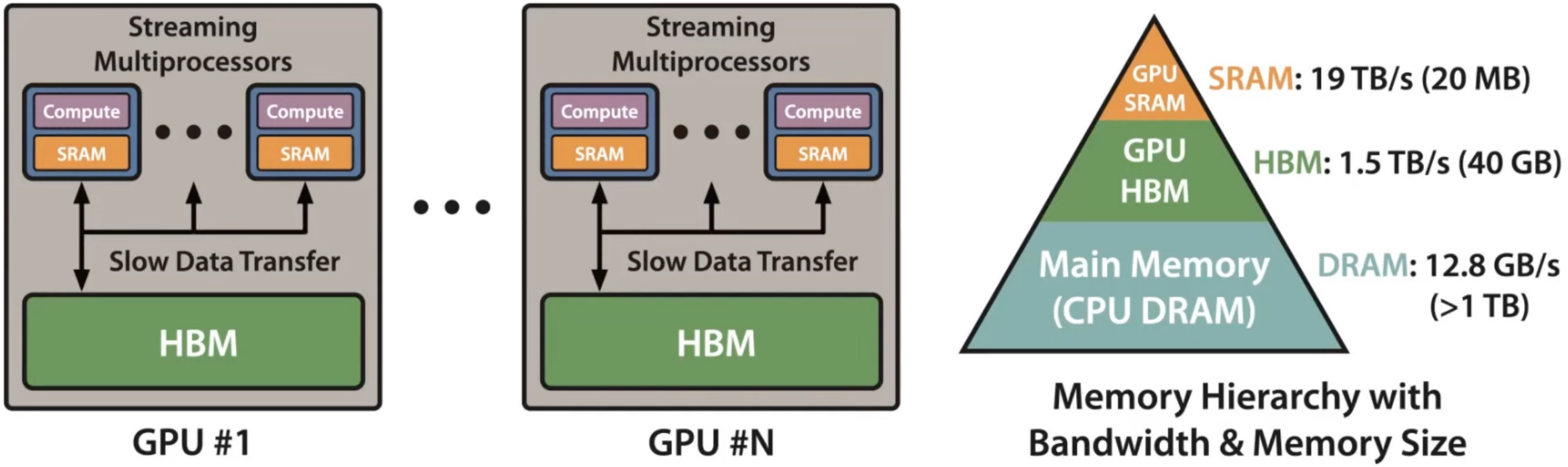


Benchmark (Metric)	**# Shots**	**Small MoE w/ MHA**	**Small MoE w/ MLA**	**Large MoE w/ MHA**	**Large MoE w/ MLA**
# Activated Params	-	2.5B	2.4B	25.0B	21.5B
# Total Params	-	15.8B	15.7B	250.8B	247.4B
KV Cache per Token (# Element)	-	110.6K	15.6K	860.2K	34.6K
BBH (EM)	3-shot	37.9	**39.0**	46.6	**50.7**
MMLU (Acc.)	5-shot	48.7	**50.0**	57.5	**59.0**
C-Eval (Acc.)	5-shot	**51.6**	50.9	57.9	**59.2**
CMMLU (Acc.)	5-shot	52.3	**53.4**	60.7	**62.5**

Comparing MHA with MLA

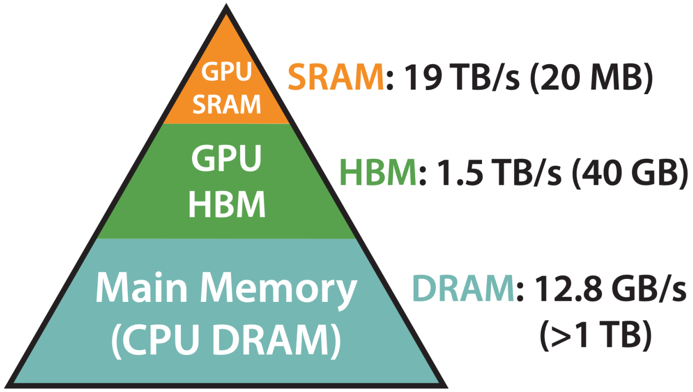
FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

- **FlashAttention** [Dao et al., 2022]
  - **Motivation**: HBM access is the bottleneck in self-attention operation
  - **Memory hierarchy**
    - GPU memory hierarchy comprises multiple forms of memory of different size and speed
    - On-chip SRAM is much smaller, but faster than high bandwidth memory (HBM)

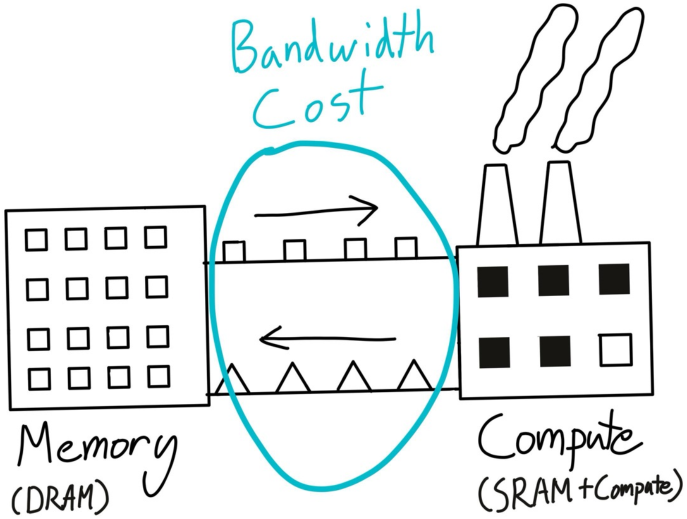


FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

- **FlashAttention** [Dao et al., 2022]
  - **Motivation**: HBM access is the bottleneck in self-attention operation
  - **Typical GPU operation**
    1. Load inputs from HBM to registers and SRAM
    2. Compute
    3. Write outputs to HBM
  - Turns out: Most operations in self-attention are **memory-bounded**
    - **More time is spent on IO to HBM**, not the compute itself

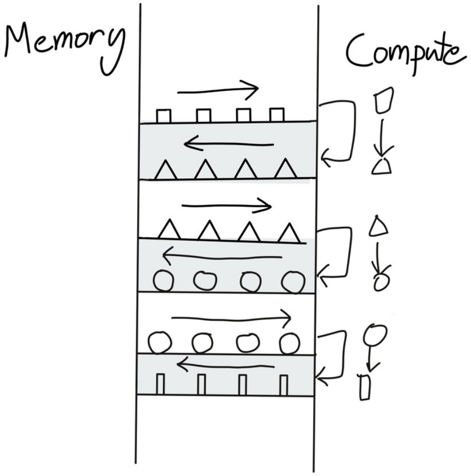


Memory Hierarchy with Bandwidth & Memory Size

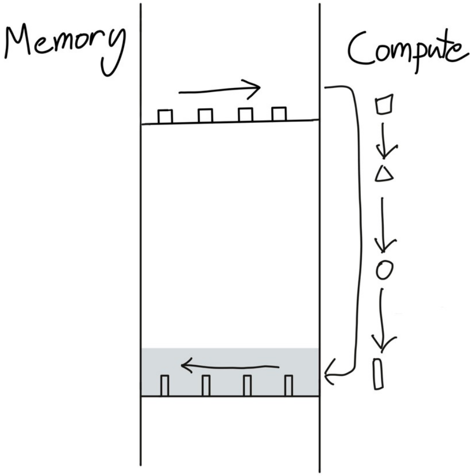


FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

- **FlashAttention** [Dao et al., 2022]
  - **Idea**: Utilize kernel fusion to reduce HBM access
    - Access HBM only once, and perform multiple operations in a row



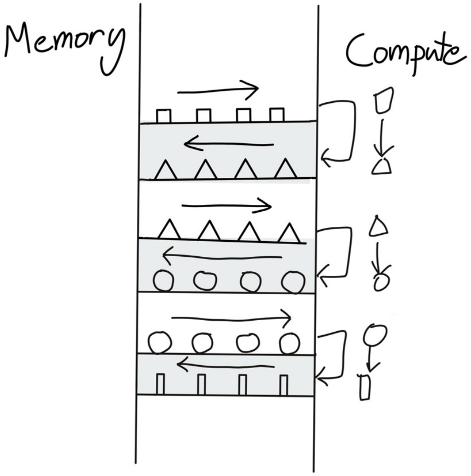
Computation
without operator fusion



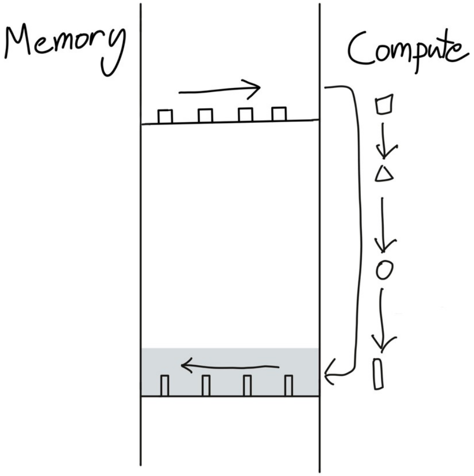
Computation
with operator fusion

FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

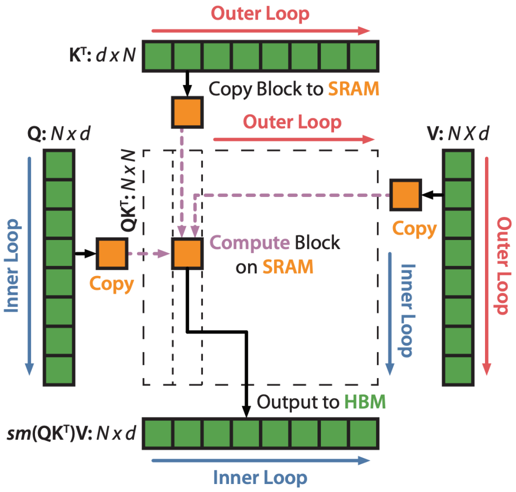
- **FlashAttention** [Dao et al., 2022]
  - **Idea**: Utilize kernel fusion to reduce HBM access
    - Access HBM only once, and perform multiple operations in a row
  - However, SRAM **memory is much smaller** than HBM
    - The full attention matrix does not fit in SRAM
    - Key intuition: compute self-attention by parts, without materializing the large attention matrix (**Tiling**)
  - This introduces a few challenges in the context of model training



Computation
without operator fusion



Computation
with operator fusion



Self-attention
computed with tiling

FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

- **FlashAttention** [Dao et al., 2022]
    - **Challenge 1**: Computing softmax without full input access
        - Solution: **Decompose large softmax** into smaller ones **by scaling**
        - With scaling, exact softmax results can be obtained after computing each block independently

$$\mathrm{softmax}([A_1, A_2]) = [\alpha\ \mathrm{softmax}(A_1),\ \ \beta\ \mathrm{softmax}(A_2)]$$

$$\mathrm{softmax}([A_1, A_2]) \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \alpha\ \mathrm{softmax}(A_1)\ V_1 + \beta\ \mathrm{softmax}(A_2)\ V_2$$

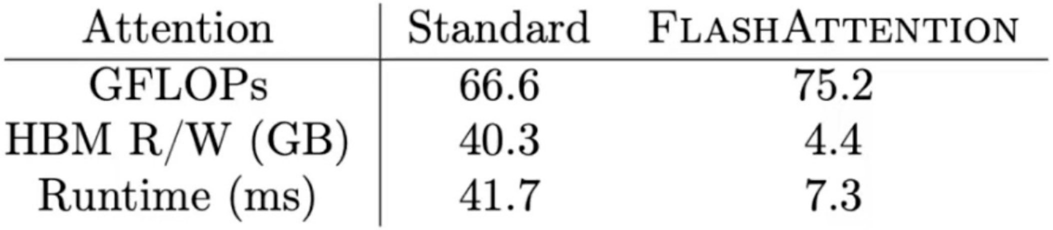
FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

- **FlashAttention** [Dao et al., 2022]
  - **Challenge 1**: Computing softmax without full input access
    - Solution: **Decompose large softmax** into smaller ones **by scaling**
    - With scaling, exact softmax results can be obtained after computing each block independently

$$\mathrm{softmax}([A_1, A_2]) = [\alpha\ \mathrm{softmax}(A_1),\ \ \beta\ \mathrm{softmax}(A_2)]$$

$$\mathrm{softmax}([A_1, A_2]) \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \alpha\ \mathrm{softmax}(A_1)\ V_1 + \beta\ \mathrm{softmax}(A_2)\ V_2$$

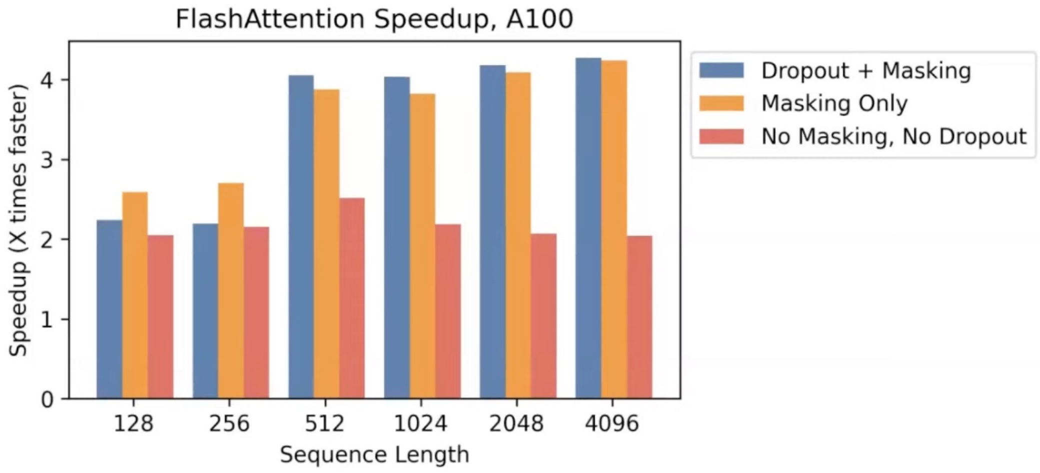
  - **Challenge 2**: Backward pass requires intermediate values
    - The attention matrix have to be saved, requiring extensive HBM access
    - Solution: **Recompute the attention matrix** during the backward pass
    - This approach is faster despite requiring more FLOPs, thanks to reduced HBM access

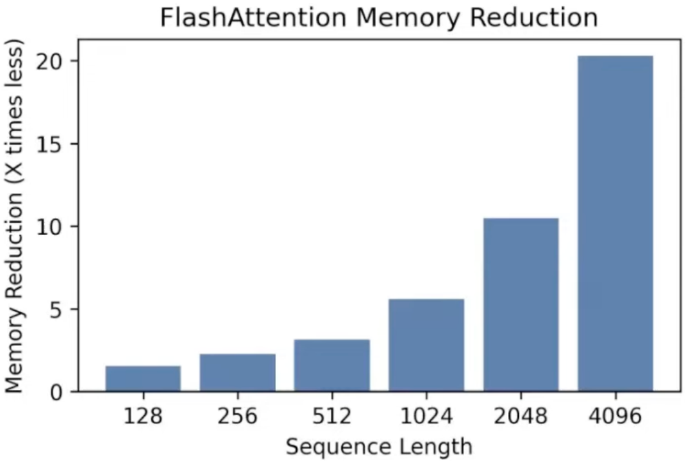


Attention	Standard	FLASHATTENTION
GFLOPs	66.6	75.2
HBM R/W (GB)	40.3	4.4
Runtime (ms)	41.7	7.3

FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

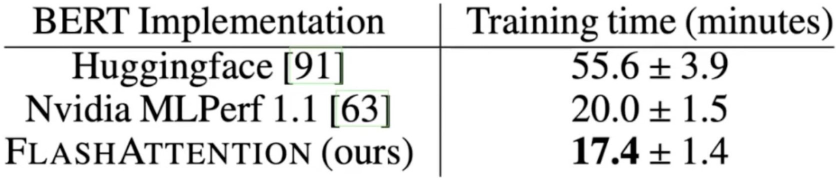
- **FlashAttention** [Dao et al., 2022]
  - FlashAttention provides actual **wall-clock speedup of 2-4x**
    - Note: Many previous approximation-based approaches focus on reducing FLOPs, and do not display wall-clock speedup
  - **Memory** requirement becomes **linear in sequence length**
    - Naïve attention requires quadratic memory
    - FlashAttention makes training & inference with longer inputs feasible



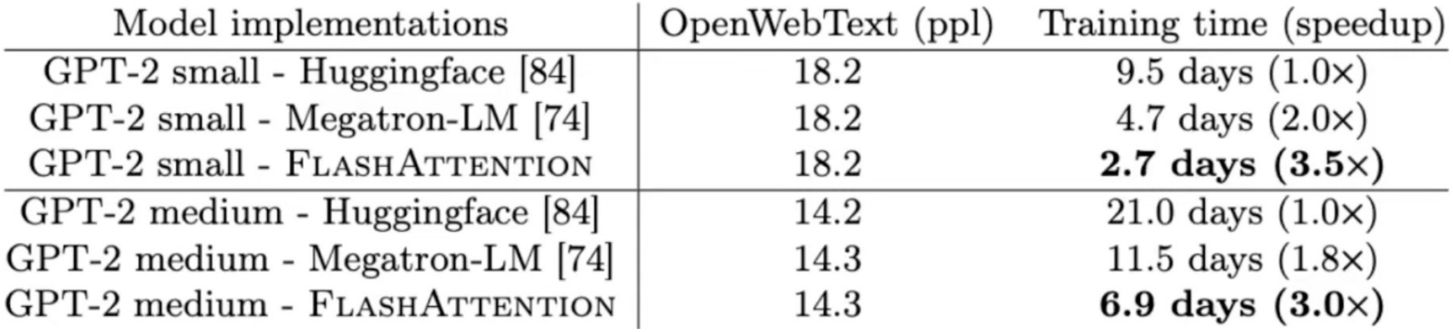


FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

- **FlashAttention** [Dao et al., 2022]
  - FlashAttention also enables **faster end-to-end training**
    - BERT training is 3.2x faster than Huggingface
    - GPT-2 training is 2.0-3.5x faster



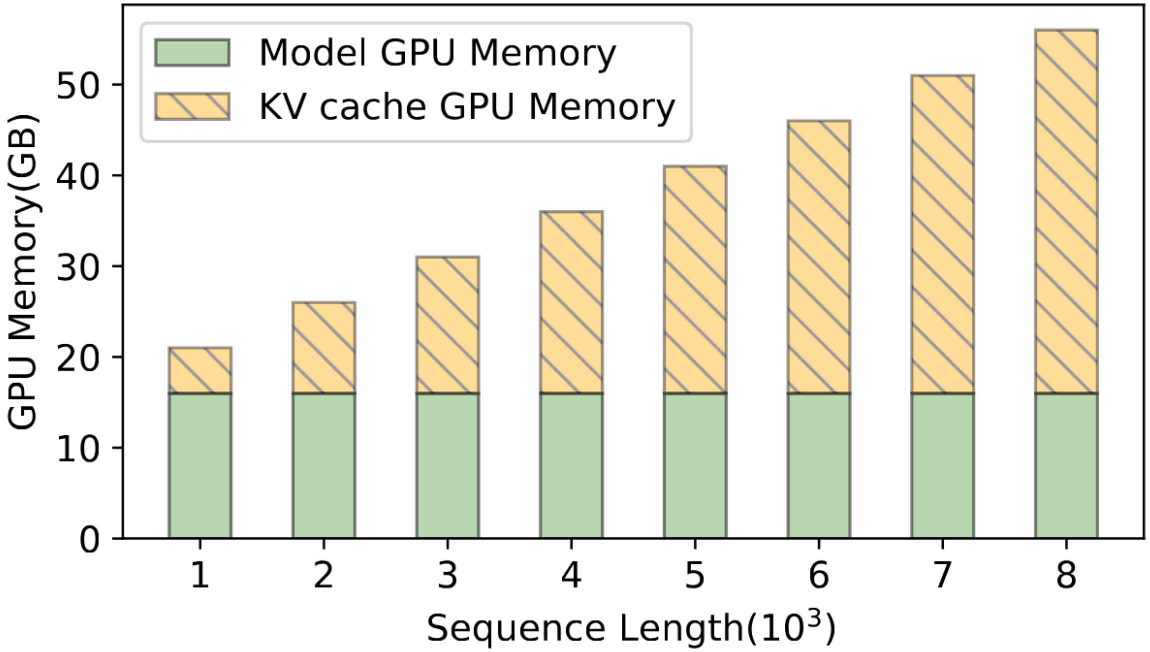
BERT Implementation	Training time (minutes)
Huggingface [91]	55.6 ± 3.9
Nvidia MLPerf 1.1 [63]	20.0 ± 1.5
FLASHATTENTION (ours)	**17.4** ± 1.4



Model implementations	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Huggingface [84]	18.2	9.5 days (1.0×)
GPT-2 small - Megatron-LM [74]	18.2	4.7 days (2.0×)
GPT-2 small - FLASHATTENTION	18.2	**2.7 days (3.5×)**
GPT-2 medium - Huggingface [84]	14.2	21.0 days (1.0×)
GPT-2 medium - Megatron-LM [74]	14.3	11.5 days (1.8×)
GPT-2 medium - FLASHATTENTION	14.3	**6.9 days (3.0×)**

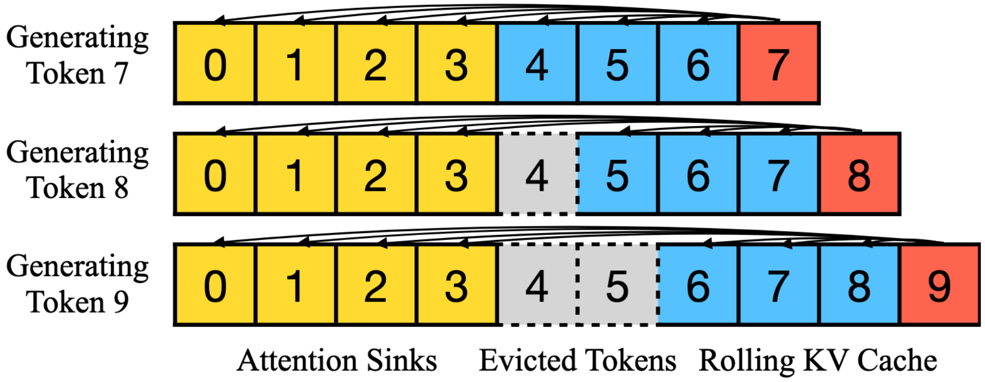
Inference-Time KV Cache Compression

- **Memory Bottleneck in Long-context Processing**
  - **Key-Value (KV) cache** grows linearly with respect to the sequence length
  - With large models and long inputs, large KV cache introduces a memory bottleneck
    - Large GPU memory is required to store the cache
  - Recent works propose to compress the KV cache at inference-time by evicting less important tokens from the cache
    - Such approach also provides decoding speedup, thanks to the smaller cache



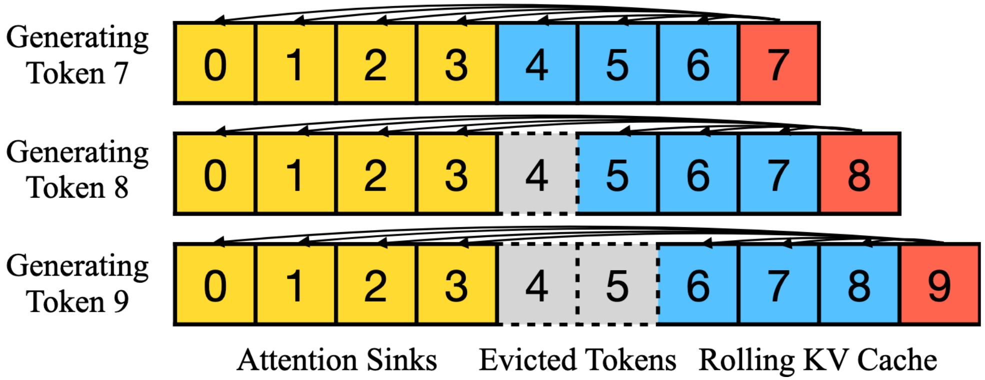
Inference-Time KV Cache Compression

- **StreamingLLM** [Xiao et al., 2024]
  - **Idea:** Only keep the initial tokens ('attention sinks') and the recent tokens
    - Initial tokens typically take up significant attention scores
    - Therefore, preserving the initial tokens is important for reducing distribution shifts introduced by the compression
  - **Position id reassignment**
    - Positional encodings are applied according to the relative position within the cache, rather than those in the original context
    - Position ids are reassigned after each eviction operation



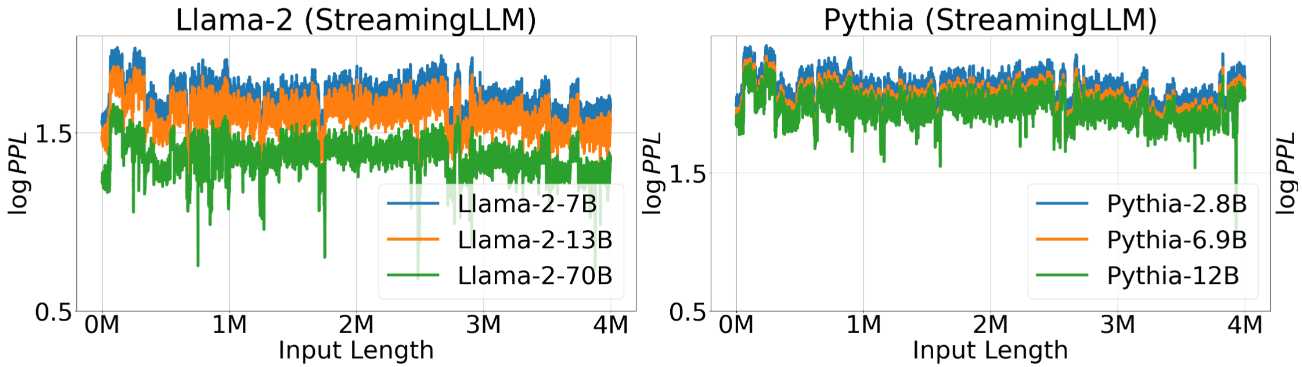
Inference-Time KV Cache Compression

- **StreamingLLM** [Xiao et al., 2024]
  - **Idea:** Only keep the initial tokens ('attention sinks') and the recent tokens
    - Initial tokens typically take up significant attention scores
    - Therefore, preserving the initial tokens is important for reducing distribution shifts introduced by the compression
  - **Position id reassignment**
    - Positional encodings are applied according to the relative position within the cache, rather than those in the original context
    - Position ids are reassigned after each eviction operation



Inference-Time KV Cache Compression

- **StreamingLLM** [Xiao et al., 2024]
  - StreamingLLM shows stable perplexity at extremely long sequences while using a fixed-sized KV cache

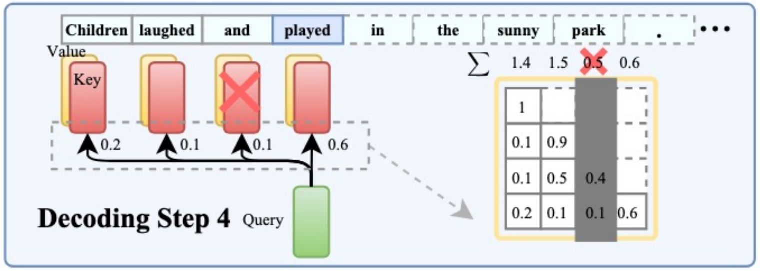


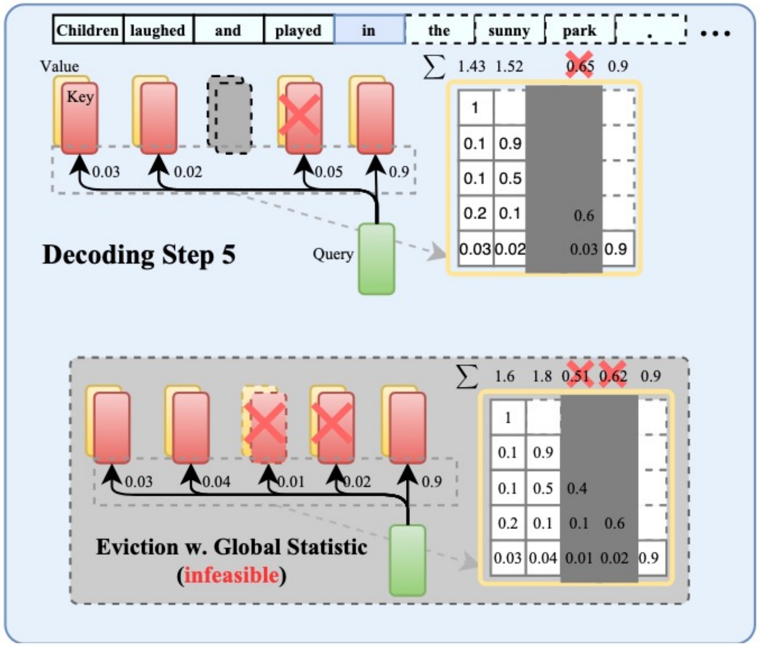
  - However, it shows limited performance at long-context benchmarks
    - Information at the middle of the prompt is lost due to token eviction
    - StreamingLLM shows comparable performance with input truncation

Llama2-7B-chat	Single-Document QA NarrativeQA	 Qasper	Multi-Document QA HotpotQA	 2WikiMQA	Summarization GovReport	 MultiNews
Truncation 1750+1750	18.7	19.2	25.4	32.8	27.3	25.8
StreamingLLM 4+3496	11.6	16.9	21.6	28.2	23.9	25.5
StreamingLLM 1750+1750	18.2	19.7	24.9	32.0	26.3	25.9

Inference-Time KV Cache Compression

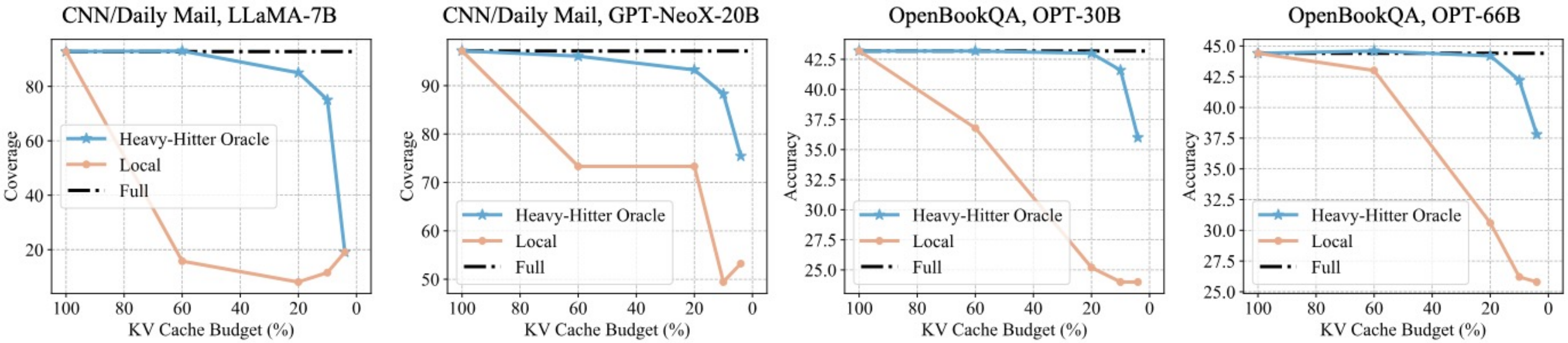
- **H2O (Heavy-Hitter Oracle)** [Zhang et al., 2023]
  - **Idea:** Keep the heavy-hitters (i.e., tokens that receive high attention scores) as well as the recent tokens
  - **Intuition**: keep the 'important' tokens in the cache to minimize information loss
  - By preserving the tokens based on the accumulated attention scores, H2O has minimal impact on attention computation despite using token eviction



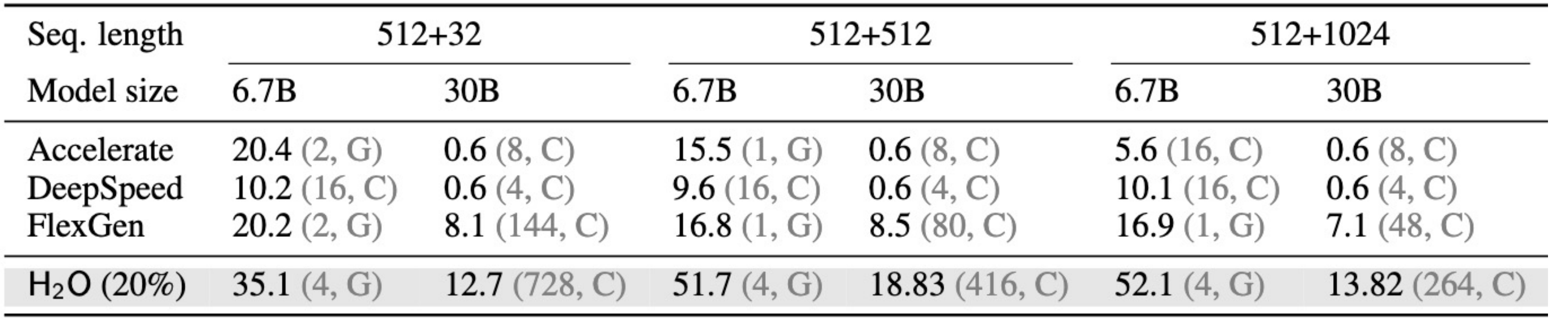


Inference-Time KV Cache Compression

- **H2O (Heavy-Hitter Oracle)** [Zhang et al., 2023]
  - H2O shows robust performance under small KV cache budgets



  - H2O also enables high-throughput inference thanks to the reduced KV cache



Seq. length	512+32		512+512		512+1024	
Model size	6.7B	30B	6.7B	30B	6.7B	30B
Accelerate	20.4 (2, G)	0.6 (8, C)	15.5 (1, G)	0.6 (8, C)	5.6 (16, C)	0.6 (8, C)
DeepSpeed	10.2 (16, C)	0.6 (4, C)	9.6 (16, C)	0.6 (4, C)	10.1 (16, C)	0.6 (4, C)
FlexGen	20.2 (2, G)	8.1 (144, C)	16.8 (1, G)	8.5 (80, C)	16.9 (1, G)	7.1 (48, C)
H_2O (20%)	35.1 (4, G)	12.7 (728, C)	51.7 (4, G)	18.83 (416, C)	52.1 (4, G)	13.82 (264, C)

Generation Throughput Comparison

Overview

Part 1. Basics

- RNN to LSTM
- Sequence-to-sequence Model
- Attention-based NLP Model

Part 2. Transformers and Large Language Models

- Transformer (self-attention)
- Pre-training of Transformers and Language Models

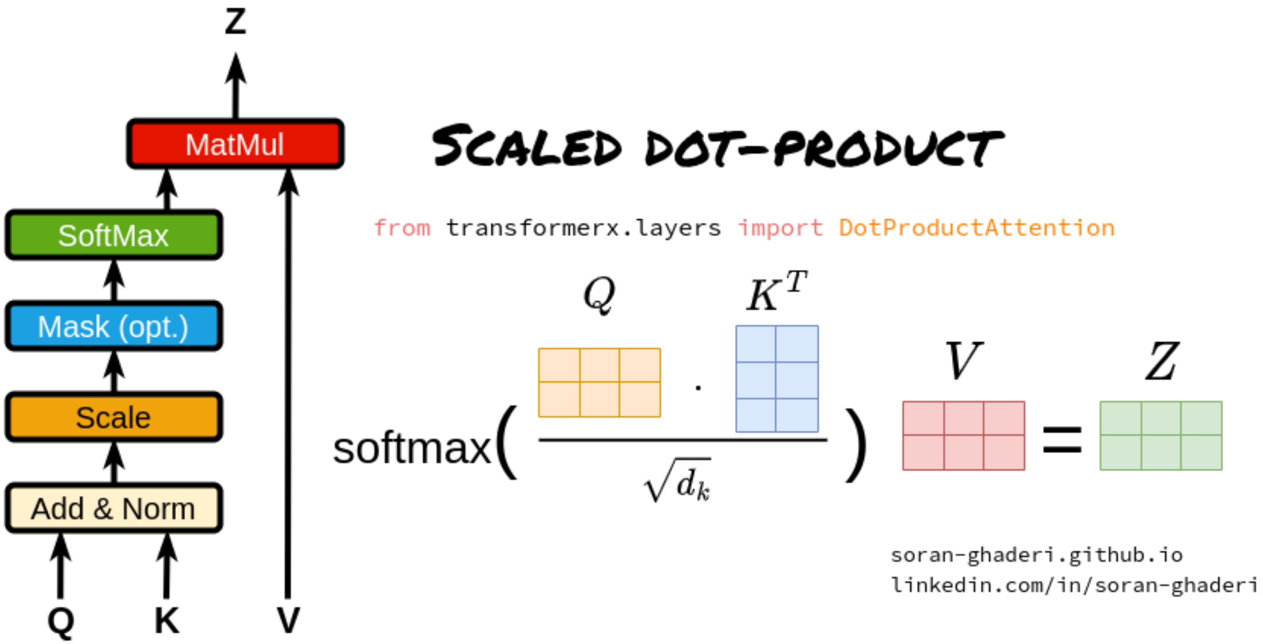
Part 3. Advanced Topics

- Handling long inputs with Transformers
- Techniques for improving efficiency
- State-Space Models

Part 4. Summary

State Space Models (SSMs) Overview

- Transformers are “Heavy” and “Slow”



```
from transformerx.layers import DotProductAttention
```

$O(N^2)$ for both inference **memory** and **time** where N is the sequence length

Need to save all the previous tokens in the memory (i.e., KV-cache)

State Space Models (SSMs) Overview

- Revisit RNNs – Efficient inference
  - Only requires saving the previous hidden state

→ Constant memory usage $O(1)$

→ Linear time inference speed $O(N)$

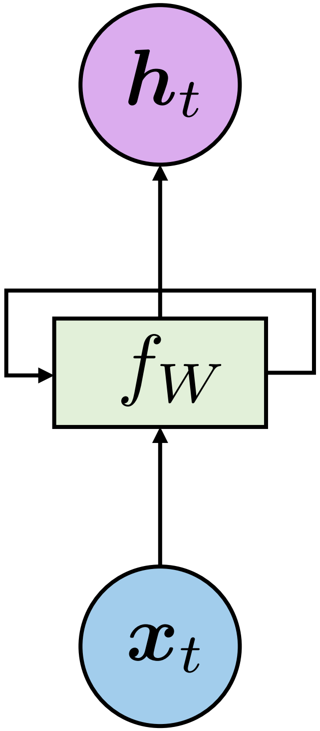
$$\boxed{\boldsymbol{h}_t} = \boxed{f_W}(\boxed{\boldsymbol{h}_{t-1}}, \boxed{\boldsymbol{x}_t})$$

New state

Old state

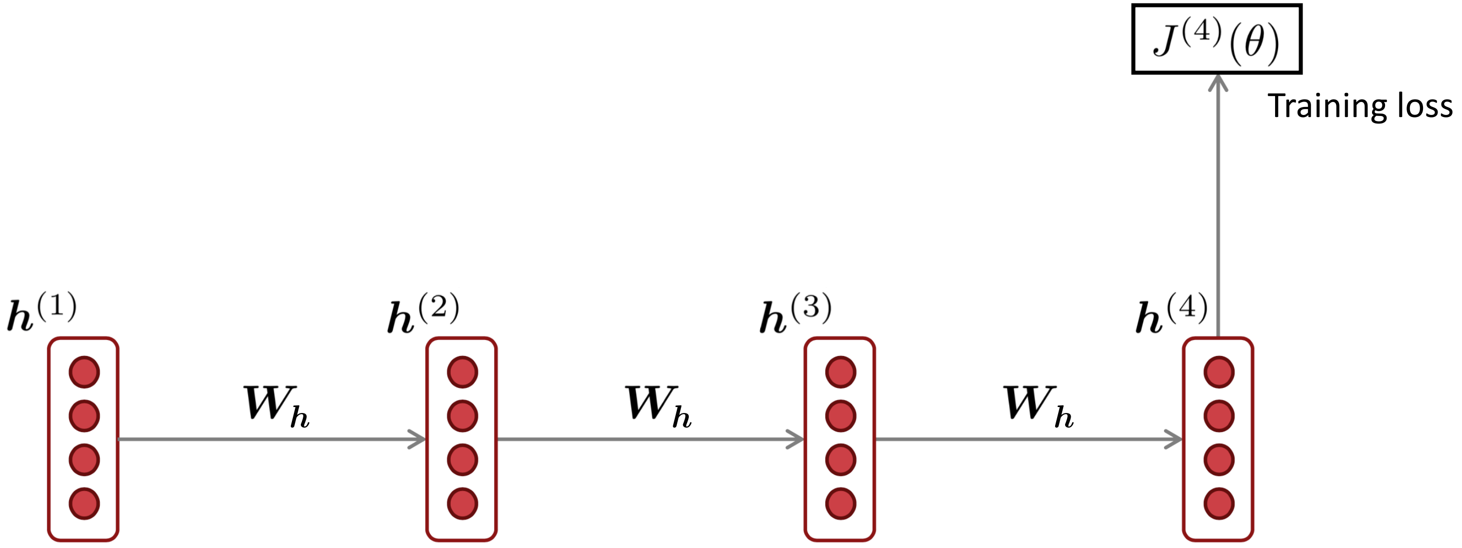
Input vector at time step t

Function parameterized by learnable W



State Space Models (SSMs) Overview

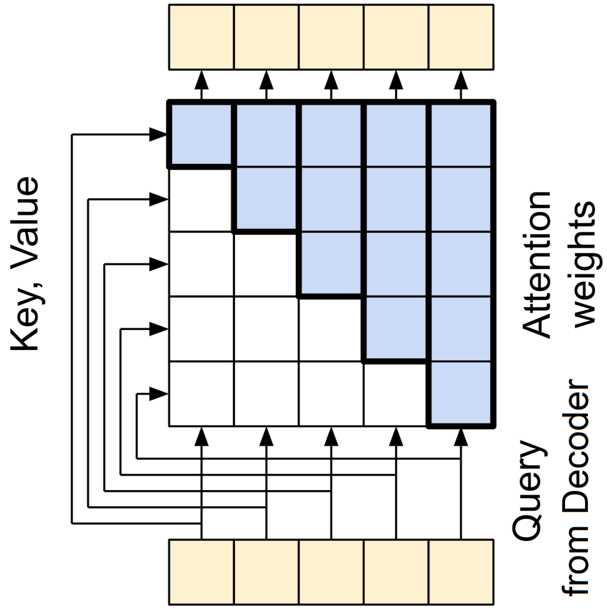
- Revisit RNNs – Efficient inference **but inefficient training**
  - Only requires saving the previous hidden state
  - But the bottleneck is the training: Not able to **parallelize**



Need to wait for the previous hidden states…

State Space Models (SSMs) Overview

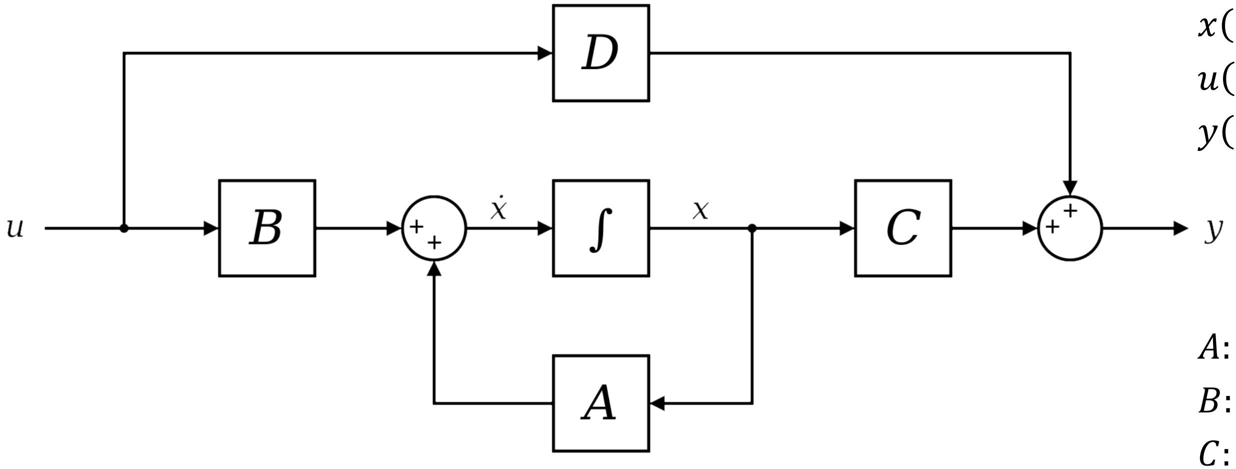
- Revisit RNNs – Efficient inference **but inefficient training**
  - Only requires saving the previous hidden state
  - But the bottleneck is the training: Not able to **parallelize**
  - **Not for Attention:** Training is super efficient as on can parallelize the forward pass.



Forward all tokens at ones,
and use causal attention mask for autoregressive learning

State Space Models (SSMs) Overview

- What is the state space model (SSM)?
    - In the context of deep learning, “Linear invariant (or stationary) systems”.



$x(t)$: state
$u(t)$: inputs
$y(t)$: outputs

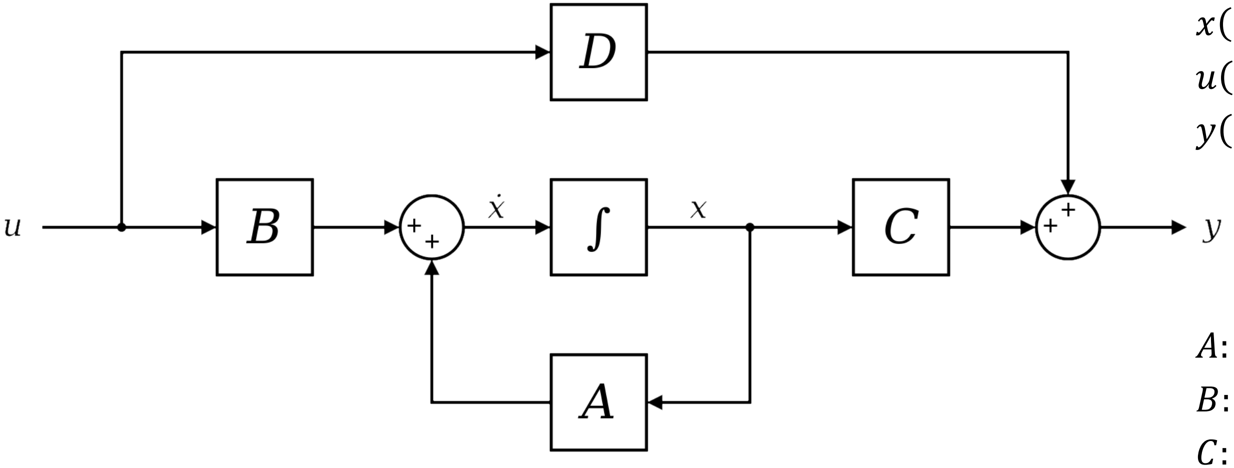
A: state matrix
B: control matrix
C: output matrix
D: command matrix

Continuous, time-invariant SSM

$$x'(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

State Space Models (SSMs) Overview

- What is the state space model (SSM)?
  - In the context of deep learning, “Linear invariant (or stationary) systems”.



$x(t)$: state
$u(t)$: inputs
$y(t)$: outputs

A: state matrix
B: control matrix
C: output matrix
D: command matrix

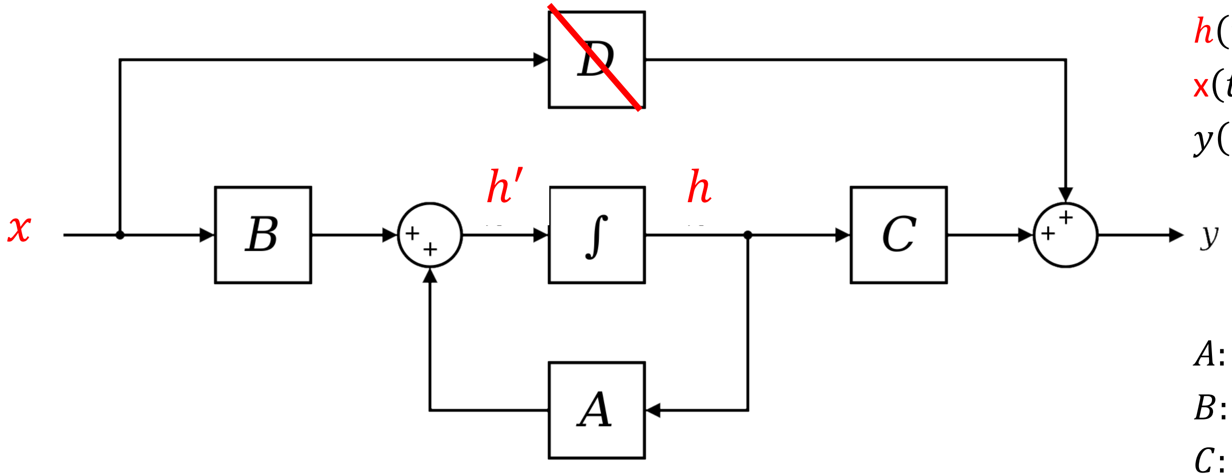
Continuous, time-invariant SSM

$$x' = Ax + Bu$$
$$y = Cx$$

We can simplify the SSM in this form of system.

State Space Models (SSMs) Overview

- What is the state space model (SSM)?
  - In the context of deep learning, “Linear invariant (or stationary) systems”.



$h(t)$: state

x(t): inputs

$y(t)$: outputs

A: state matrix

B: control matrix

C: output matrix

~~D: command matrix~~

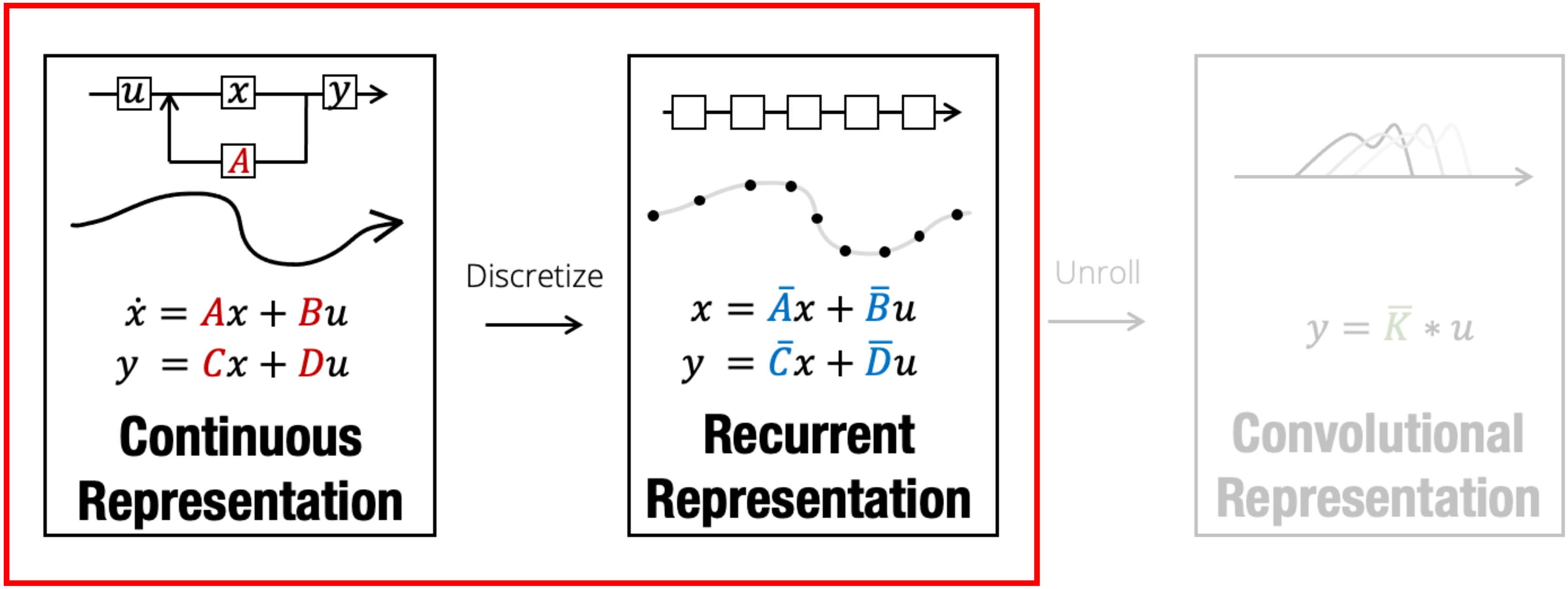
Continuous, time-invariant SSM

$$h' = Ah + Bx$$
$$y = Ch$$

We can simplify the SSM in this form of system.

State Space Models (SSMs) Overview

- Discretizing the SSMs.
  - Since continuous representation is significantly slow for both training and inference,
  - We discretize the SSM to view the system in the recurrent representation.



State Space Models (SSMs) Overview

- Discretizing the SSMs.
  - Since continuous representation is significantly slow for both training and inference,
  - We discretize the SSM to view the system in the recurrent representation.
  - Discretize matrices (we are introducing the ZOH discretization method here).

State Space Representation

$$h'(t) = Ah(t) + Bx(t)$$

$$y(t) = Ch(t)$$

Discretized State Space Model

$$h_t = \bar{A}h_{t-1} + \bar{B}x_t$$

$$y_t = \bar{C}h_t$$

ZOH discretization where Δ is a steps size

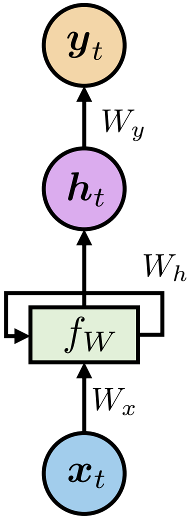
$$\bar{A} = \exp(\Delta A)$$
$$\bar{B} = (\Delta A)^{-1}(\exp \Delta A \ - I) \cdot \Delta B$$
$$\bar{C} = C$$

State Space Models (SSMs) Overview

- Discretizing the SSMs.
  - Since continuous representation is significantly slow for both training and inference,
  - We discretize the SSM to view the system in the recurrent representation.
  - Connection with recurrent neural networks (RNNs)?
    - Generally, SSMs are Linear RNNs.

Recap. Recurrent Neural Net

Discretized State Space Model



$$h_t = f_W(h_{t-1}, x_t) \xrightarrow{\text{Linear}} \begin{aligned} h_t &= \bar{A}h_{t-1} + \bar{B}x_t \\ y_t &= \bar{C}h_t \end{aligned}$$

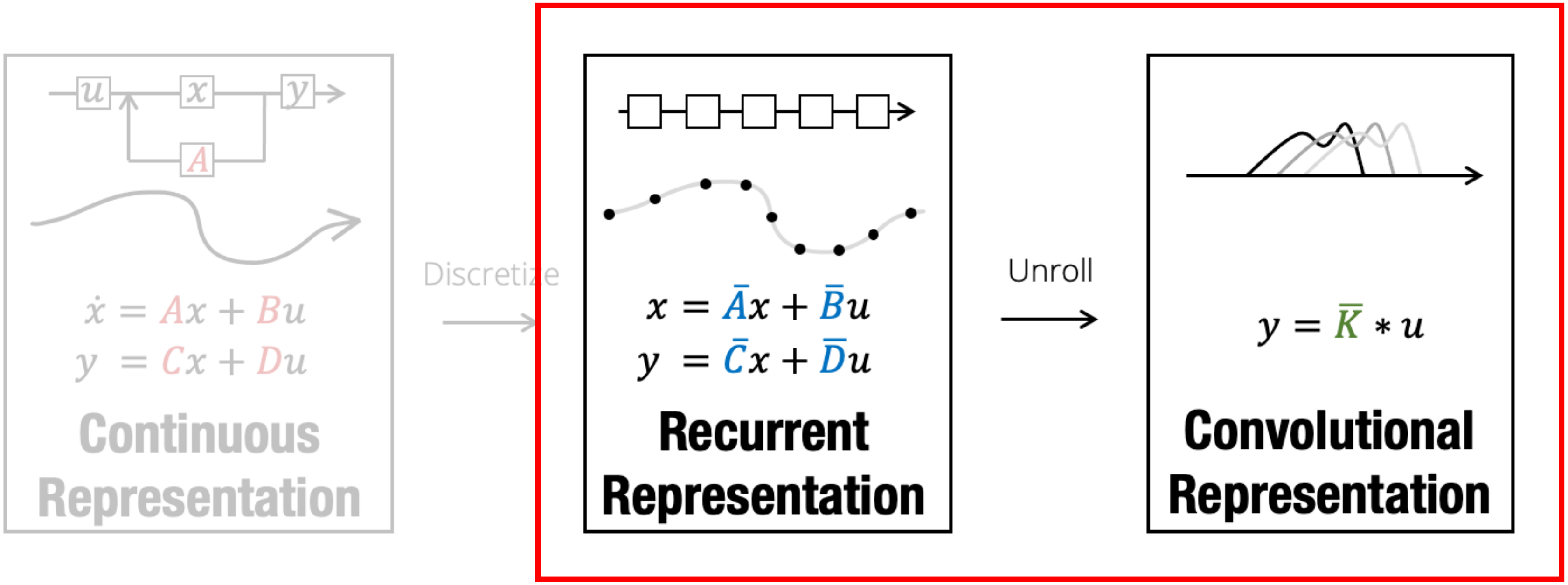
$\downarrow$ Nonlinear

$$h_t = \tanh(W_h h_{t-1} + W_x x_t)$$

$$y_t = W_y h_t$$

State Space Models (SSMs) Overview

- Parallelizing the SSM training.
  - The sequential nature of RNN is the primary cause of slow training.
  - Several approaches have been suggested to tackle this problem.
    - We will first look at the convolutional representation of SSM.



State Space Models (SSMs) Overview

- Parallelizing the SSM training.
  - The sequential nature of RNN is the primary cause of slow training.
  - **Linearity is the key**: Convolutional representation.

Discretized State Space Model (Linear RNN)

$$h_t = \bar{A}h_{t-1} + \bar{B}x_t$$

$$y_t = \bar{C}h_t$$

Assuming the initial state $\boldsymbol{h_{t-1} = 0}$, then …

$$h_0 = \bar{B}x_0 \qquad h_1 = \overline{AB}x_0 + \bar{B}x_1 \qquad h_2 = \overline{A^2B}x_0 + \overline{AB}x_1 + \bar{B}x_2$$

$$y_0 = \overline{CB}x_0 \qquad y_1 = \overline{CAB}x_0 + \overline{CB}x_1 \qquad y_2 = \overline{CA^2B}x_0 + \overline{CAB}x_1 + \overline{CB}x_2$$

State Space Models (SSMs) Overview

- Parallelizing the SSM training.
  - The sequential nature of RNN is the primary cause of slow training.
  - **Linearity is the key**: Convolutional representation.

Discretized State Space Model (Linear RNN)

$$h_t = \bar{A}h_{t-1} + \bar{B}x_t$$

$$y_t = \bar{C}h_t$$

Assuming the initial state $\boldsymbol{h_{t-1} = 0}$, then …

$$h_0 = \bar{B}x_0 \qquad h_1 = \overline{AB}x_0 + \bar{B}x_1 \qquad h_2 = \overline{A^2B}x_0 + \overline{AB}x_1 + \bar{B}x_2$$

$$y_0 = \overline{CB}x_0 \qquad y_1 = \overline{CAB}x_0 + \overline{CB}x_1 \qquad y_2 = \overline{CA^2B}x_0 + \overline{CAB}x_1 + \overline{CB}x_2$$

$$y_t = \overline{CA^tB}x_0 + \overline{CA^{t-1}B}x_1 + \cdots + \overline{CAB}x_{t-1} + \overline{CB}x_t$$

State Space Models (SSMs) Overview

- Parallelizing the SSM training.
  - The sequential nature of RNN is the primary cause of slow training.
  - **Linearity is the key**: Convolutional representation.

Discretized State Space Model (Linear RNN)

$$h_t = \bar{A}h_{t-1} + \bar{B}x_t$$

$$y_t = \bar{C}h_t$$

Assuming the initial state $h_{t-1} = 0$, then …

$$h_0 = \bar{B}x_0 \qquad h_1 = \overline{AB}x_0 + \bar{B}x_1 \qquad h_2 = \overline{A^2B}x_0 + \overline{AB}x_1 + \bar{B}x_2$$

$$y_0 = \overline{CB}x_0 \qquad y_1 = \overline{CAB}x_0 + \overline{CB}x_1 \qquad y_2 = \overline{CA^2B}x_0 + \overline{CAB}x_1 + \overline{CB}x_2$$

$$y_t = \overline{CA^tB}x_0 + \overline{CA^{t-1}B}x_1 + \cdots + \overline{CAB}x_{t-1} + \overline{CB}x_t$$

$$\rightarrow y = \bar{K} * x. \quad \text{where} \quad \bar{K} = (\overline{CB}, \overline{CAB}, \ldots, \overline{CA^TB})$$

Able to parallelize: Forward all input tokens at once and **perform convolution**

State Space Models (SSMs) Overview

- We’ve discussed SSM’s **efficiency** (Faster decoding, Parallelizing the training…).
- The next thing we will discuss is how to make SSM **effective**.
  - How to define the state matrix: Hippo[Gu et al., 2020].
  - How to make the state computation efficient: S4[Gu et al., 2022a], and S4D[Gu et al., 2022b]

State Space Models (SSMs) Overview

- How to define the state matrix.
    - A well-defined (initialized) state matrix is the key to success.
    - Hippo: High-order Polynomial Projection Operators[Gu et al., 2020]
        - Suggests a way of initializing the state matrix A so that it enjoys long-term dependencies.

$$A_{nk} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & if\ n > k \\ n+1 & if\ n = k \\ 0 & if\ n < k \end{cases}, \quad B_n = (2n+1)^{1/2}$$

State Space Models (SSMs) Overview

- How to define the state matrix.
  - A well-defined (initialized) state matrix is the key to success.
  - Hippo: High-order Polynomial Projection Operators[Gu et al., 2020]
    - Suggests a way of initializing the state matrix A so that it enjoys long-term dependencies.

$$A_{nk} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & if\ n > k \\ n+1 & if\ n = k \\ 0 & if\ n < k \end{cases}, \quad B_n = (2n+1)^{1/2}$$

- However, such a matrix is inefficient while using the convolutional kernel

$$y_t = \overline{CA^tB}x_0 + \overline{CA^{t-1}B}x_1 + \cdots + \overline{CAB}x_{t-1} + \overline{CB}x_t$$

$$y = \overline{K} * x. \ \text{ where } \ \overline{K} = (\overline{CB}, \overline{CAB}, \ldots, \overline{CA^TB})$$

State Space Models (SSMs) Overview

- How to make the state computation efficient.
  - **S4** [Gu et al., 2022a] : State matrix can be decomposed as **Normal Plus Low-rank (NPLR).**

$$A = V\Lambda V^* - PQ^T = V(\Lambda - (V^*P)(V^*Q)^*)V^*$$

$$\text{for unitary } V \in \mathbb{C}^{N\times N}\text{, diagonal } \Lambda\text{, and low}-\text{rank factorization } P, Q \in \mathbb{R}^{N\times r}.$$

  - **S4D** [Gu et al., 2022b] : State matrix can be expressed with **a diagonal matrix.**

$$(\text{S4D}-\text{Inv})\ A_n = -\frac{1}{2} + i\frac{N}{\pi}\left(\frac{N}{2n+1} - 1\right) \qquad (\text{S4D}-\text{Lin})\ A_n = -\frac{1}{2} + i\pi n$$

State Space Models (SSMs) Overview

- How to make the state computation efficient.
  - **S4** [Gu et al., 2022a] : State matrix can be decomposed as **Normal Plus Low-rank (NPLR).**

$$A = V\Lambda V^* - PQ^T = V(\Lambda - (V^*P)(V^*Q)^*)V^*$$

$$\text{for unitary } V \in \mathbb{C}^{N\times N}, \text{diagonal } \Lambda, \text{and low}-\text{rank factorization } P, Q \in \mathbb{R}^{N\times r}.$$

  - **S4D** [Gu et al., 2022b] : State matrix can be expressed with **a diagonal matrix.**

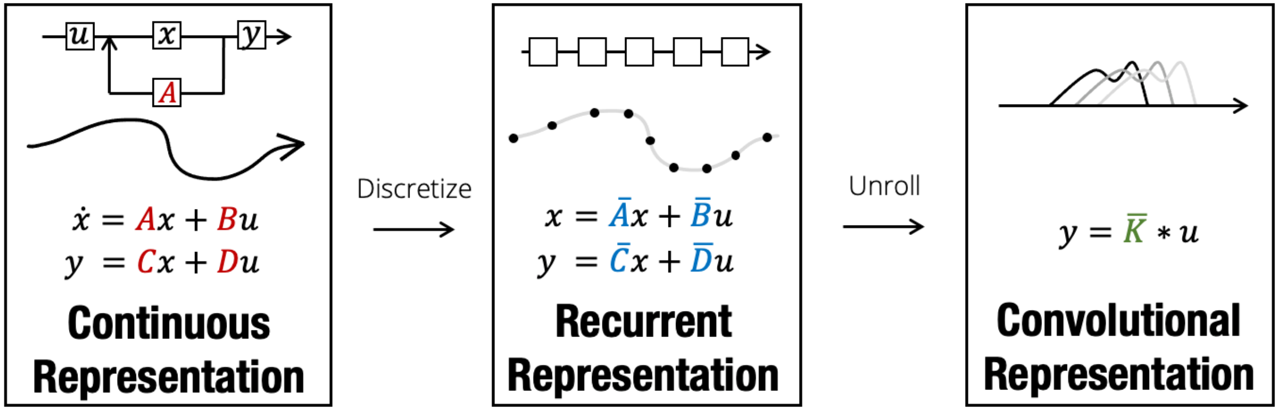
$$(\textbf{S4D}-\textbf{Inv})\ A_n = -\frac{1}{2} + i\frac{N}{\pi}\left(\frac{N}{2n+1} - 1\right) \qquad (\textbf{S4D}-\textbf{Lin})\ A_n = -\frac{1}{2} + i\pi n$$

Now the convolution kernel $\overline{K} = (\overline{CB}, \overline{CAB}, \dots, \overline{CA^TB})$ is super efficient to compute

State Space Models (SSMs) Overview

- Summary of the SSM overview.
  - Discretizing and unrolling to do parallel training.
  - Well-defining the state matrix for both efficiency and effectiveness.

Three different views of state space models

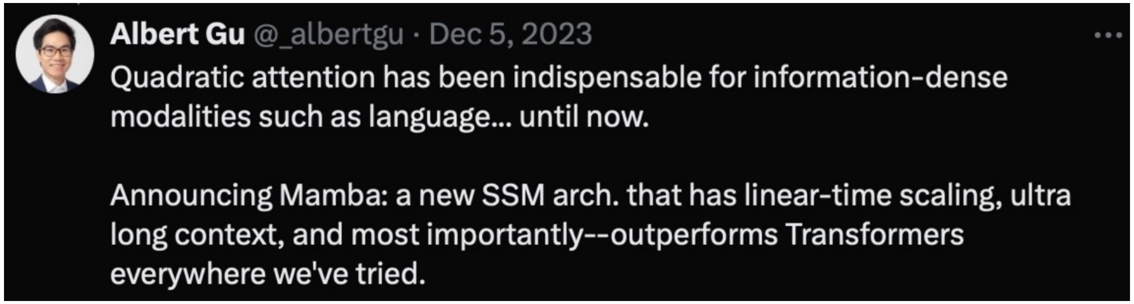


S4D[Gu et al., 2022b]: Diagonal state matrix for efficient computation

$$(\mathbf{S4D{-}Inv})\ A_n = -\frac{1}{2} + i\,\frac{N}{\pi}\left(\frac{N}{2n+1} - 1\right) \qquad (\mathbf{S4D{-}Lin})\ A_n = -\frac{1}{2} + i\pi n$$

State Space Models (SSMs) for Large Scale Language Modeling

- **Mamba**: Selective State-Space Models [Gu et al., 2024]
  - **Motivation**: SSM lacks content-based reasoning abilities
    - The state update is done independently with the input
  - Which tasks require content-based reasoning?
    - Summarization, retrieval, basically all NLP tasks



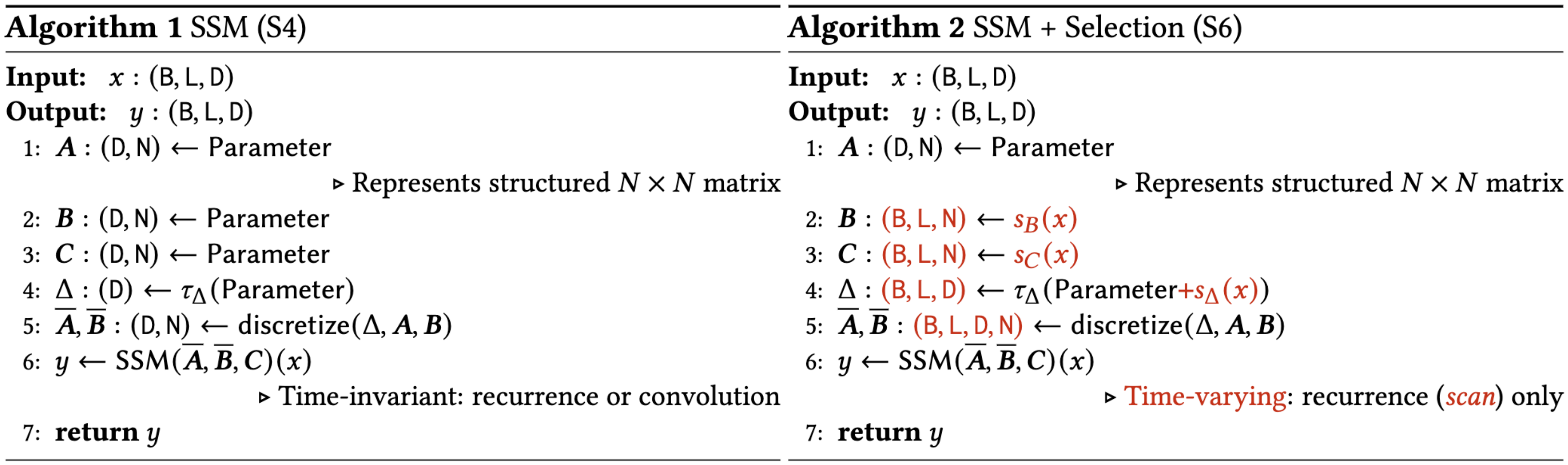
Albert Gu @_albertgu · Dec 5, 2023

Quadratic attention has been indispensable for information-dense modalities such as language... until now.

Announcing Mamba: a new SSM arch. that has linear-time scaling, ultra long context, and most importantly--outperforms Transformers everywhere we've tried.

State Space Models (SSMs) for Large Scale Language Modeling

- **Mamba**: Selective State-Space Models [Gu et al., 2024]
  - **Key idea**: Define input-dependent (content-aware) state matrices
  - Define Δ, B, C as input-dependent factors
    - Denote such SSM as a Selective SSM



Algorithm 1 SSM (S4)

```
Input: $x$ : (B, L, D)
Output: $y$ : (B, L, D)
$\boldsymbol{A}$ : (D, N) $\leftarrow$ Parameter
                    ▷ Represents structured $N \times N$ matrix
$\boldsymbol{B}$ : (D, N) $\leftarrow$ Parameter
$\boldsymbol{C}$ : (D, N) $\leftarrow$ Parameter
$\Delta$ : (D) $\leftarrow \tau_\Delta$(Parameter)
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}$ : (D, N) $\leftarrow$ discretize($\Delta, \boldsymbol{A}, \boldsymbol{B}$)
$y \leftarrow$ SSM($\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C}$)($x$)
                    ▷ Time-invariant: recurrence or convolution
return $y$
```

Algorithm 2 SSM + Selection (S6)

```
Input: $x$ : (B, L, D)
Output: $y$ : (B, L, D)
$\boldsymbol{A}$ : (D, N) $\leftarrow$ Parameter
                    ▷ Represents structured $N \times N$ matrix
$\boldsymbol{B}$ : (B, L, N) $\leftarrow s_B(x)$
$\boldsymbol{C}$ : (B, L, N) $\leftarrow s_C(x)$
$\Delta$ : (B, L, D) $\leftarrow \tau_\Delta$(Parameter+$s_\Delta(x)$)
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}$ : (B, L, D, N) $\leftarrow$ discretize($\Delta, \boldsymbol{A}, \boldsymbol{B}$)
$y \leftarrow$ SSM($\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C}$)($x$)
                    ▷ Time-varying: recurrence (scan) only
return $y$
```

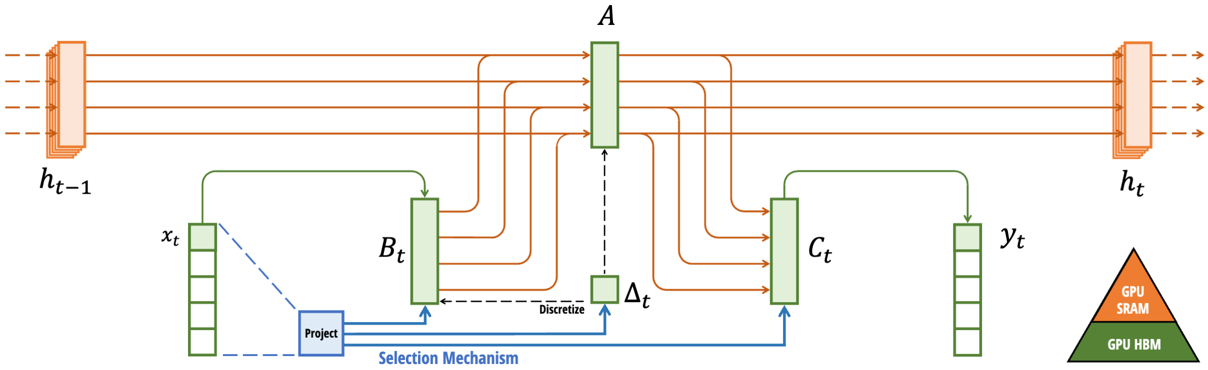
State Space Models (SSMs) for Large Scale Language Modeling

- **Mamba**: Selective State-Space Models [Gu et al., 2024]
  - Problem of selective SSM: It is not an LTI system
    - Cannot use convolution because Δ, B, C are input-dependent now

$$y_t = \overline{CA^tB}x_0 + \overline{CA^{t-1}B}x_1 + \cdots + \overline{CAB}x_{t-1} + \overline{CB}x_t$$

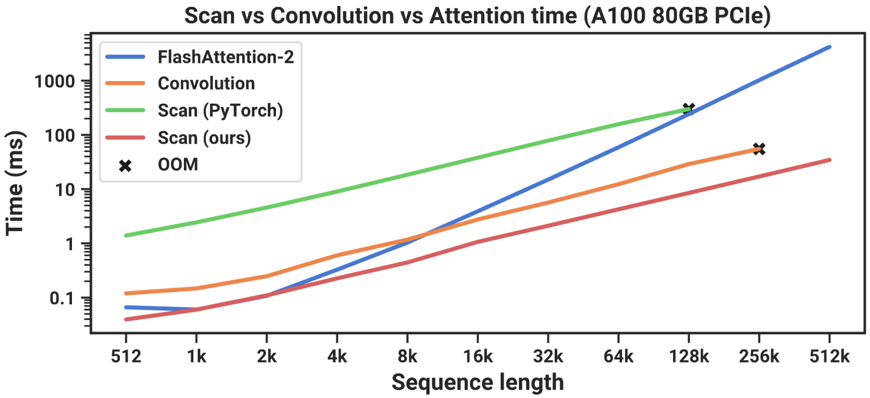
$$y = \overline{K} * x. \;\; \text{where} \;\; \overline{K} = (\overline{CB}, \overline{CAB}, \dots, \overline{CA^TB})$$

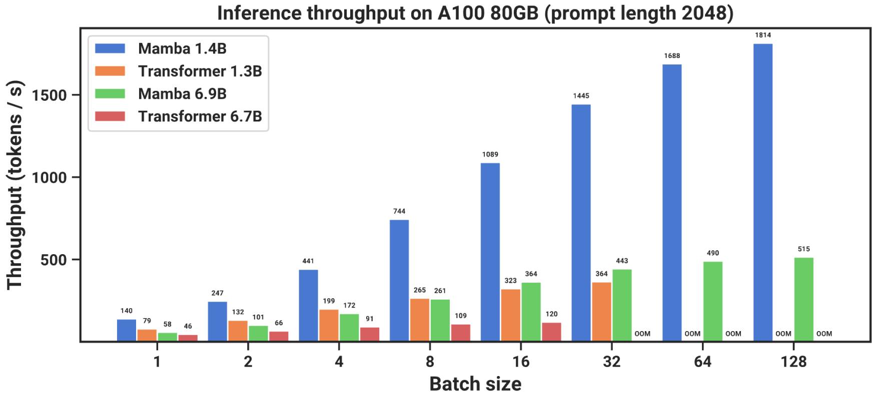
  - Another option for computing linear recurrence is associative parallel scan
    - **S5** [Smith et al., 2023] shows that linear recurrence operation is associative
    - Mamba provides efficient, hardware-aware implementation of associative scan



State Space Models (SSMs) for Large Scale Language Modeling

- **Mamba**: Selective State-Space Models [Gu et al., 2024]
  - Overall, the inference time significantly reduces
    - Faster then Flash Attention 2, especially with longer sequence lengths
    - Provides up to 5x higher throughput





State Space Models (SSMs) for Large Scale Language Modeling

- **Mamba**: Selective State-Space Models [Gu et al., 2024]
  - Mamba shows high zero-shot performance on various NLP tasks

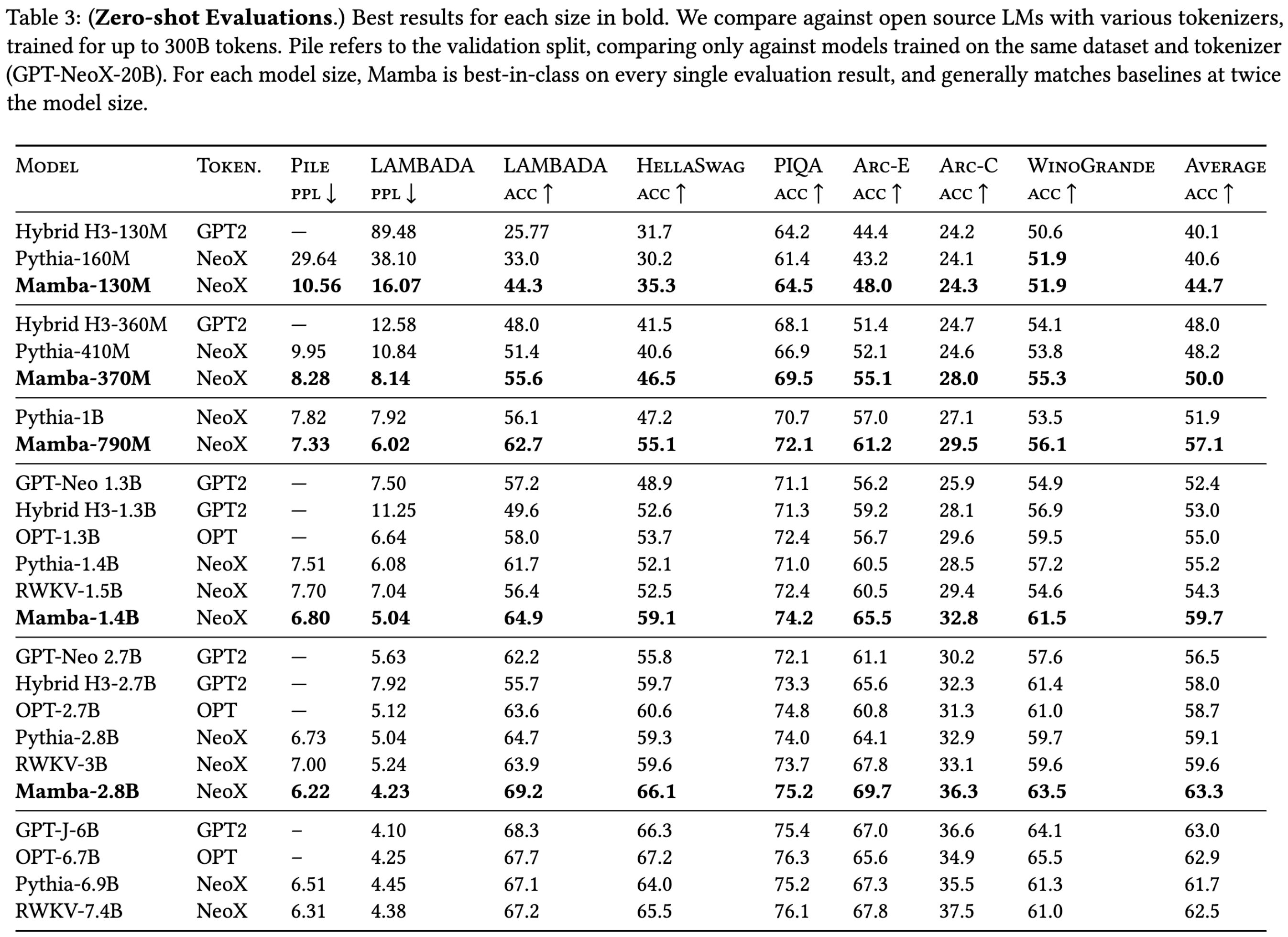
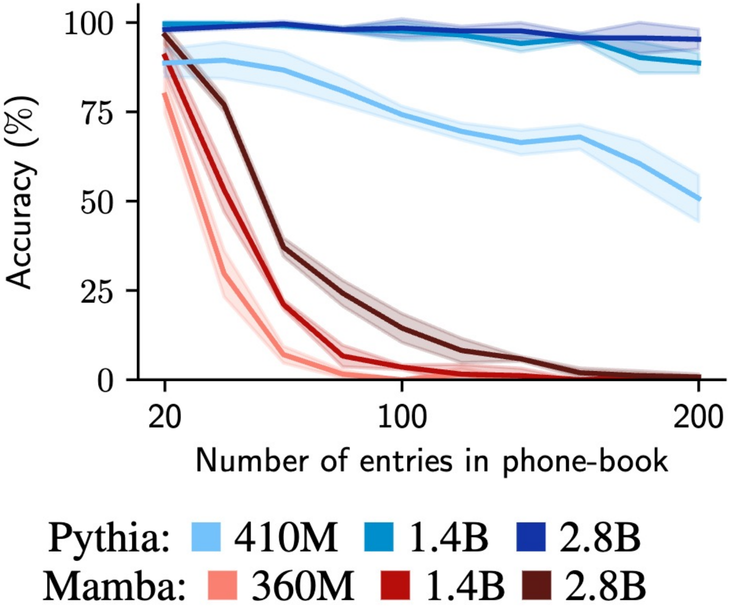


Table 3: (**Zero-shot Evaluations.**) Best results for each size in bold. We compare against open source LMs with various tokenizers, trained for up to 300B tokens. Pile refers to the validation split, comparing only against models trained on the same dataset and tokenizer (GPT-NeoX-20B). For each model size, Mamba is best-in-class on every single evaluation result, and generally matches baselines at twice the model size.

Model	Token.	Pile ppl ↓	LAMBADA ppl ↓	LAMBADA acc ↑	HellaSwag acc ↑	PIQA acc ↑	Arc-E acc ↑	Arc-C acc ↑	WinoGrande acc ↑	Average acc ↑
Hybrid H3-130M	GPT2	—	89.48	25.77	31.7	64.2	44.4	24.2	50.6	40.1
Pythia-160M	NeoX	29.64	38.10	33.0	30.2	61.4	43.2	24.1	**51.9**	40.6
Mamba-130M	NeoX	**10.56**	**16.07**	**44.3**	**35.3**	**64.5**	**48.0**	**24.3**	**51.9**	**44.7**
Hybrid H3-360M	GPT2	—	12.58	48.0	41.5	68.1	51.4	24.7	54.1	48.0
Pythia-410M	NeoX	9.95	10.84	51.4	40.6	66.9	52.1	24.6	53.8	48.2
Mamba-370M	NeoX	**8.28**	**8.14**	**55.6**	**46.5**	**69.5**	**55.1**	**28.0**	**55.3**	**50.0**
Pythia-1B	NeoX	7.82	7.92	56.1	47.2	70.7	57.0	27.1	53.5	51.9
Mamba-790M	NeoX	**7.33**	**6.02**	**62.7**	**55.1**	**72.1**	**61.2**	**29.5**	**56.1**	**57.1**
GPT-Neo 1.3B	GPT2	—	7.50	57.2	48.9	71.1	56.2	25.9	54.9	52.4
Hybrid H3-1.3B	GPT2	—	11.25	49.6	52.6	71.3	59.2	28.1	56.9	53.0
OPT-1.3B	OPT	—	6.64	58.0	53.7	72.4	56.7	29.6	59.5	55.0
Pythia-1.4B	NeoX	7.51	6.08	61.7	52.1	71.0	60.5	28.5	57.2	55.2
RWKV-1.5B	NeoX	7.70	7.04	56.4	52.5	72.4	60.5	29.4	54.6	54.3
Mamba-1.4B	NeoX	**6.80**	**5.04**	**64.9**	**59.1**	**74.2**	**65.5**	**32.8**	**61.5**	**59.7**
GPT-Neo 2.7B	GPT2	—	5.63	62.2	55.8	72.1	61.1	30.2	57.6	56.5
Hybrid H3-2.7B	GPT2	—	7.92	55.7	59.7	73.3	65.6	32.3	61.4	58.0
OPT-2.7B	OPT	—	5.12	63.6	60.6	74.8	60.8	31.3	61.0	58.7
Pythia-2.8B	NeoX	6.73	5.04	64.7	59.3	74.0	64.1	32.9	59.7	59.1
RWKV-3B	NeoX	7.00	5.24	63.9	59.6	73.7	67.8	33.1	59.6	59.6
Mamba-2.8B	NeoX	**6.22**	**4.23**	**69.2**	**66.1**	**75.2**	**69.7**	**36.3**	**63.5**	**63.3**
GPT-J-6B	GPT2	–	4.10	68.3	66.3	75.4	67.0	36.6	64.1	63.0
OPT-6.7B	OPT	–	4.25	67.7	67.2	76.3	65.6	34.9	65.5	62.9
Pythia-6.9B	NeoX	6.51	4.45	67.1	64.0	75.2	67.3	35.5	61.3	61.7
RWKV-7.4B	NeoX	6.31	4.38	67.2	65.5	76.1	67.8	37.5	61.0	62.5

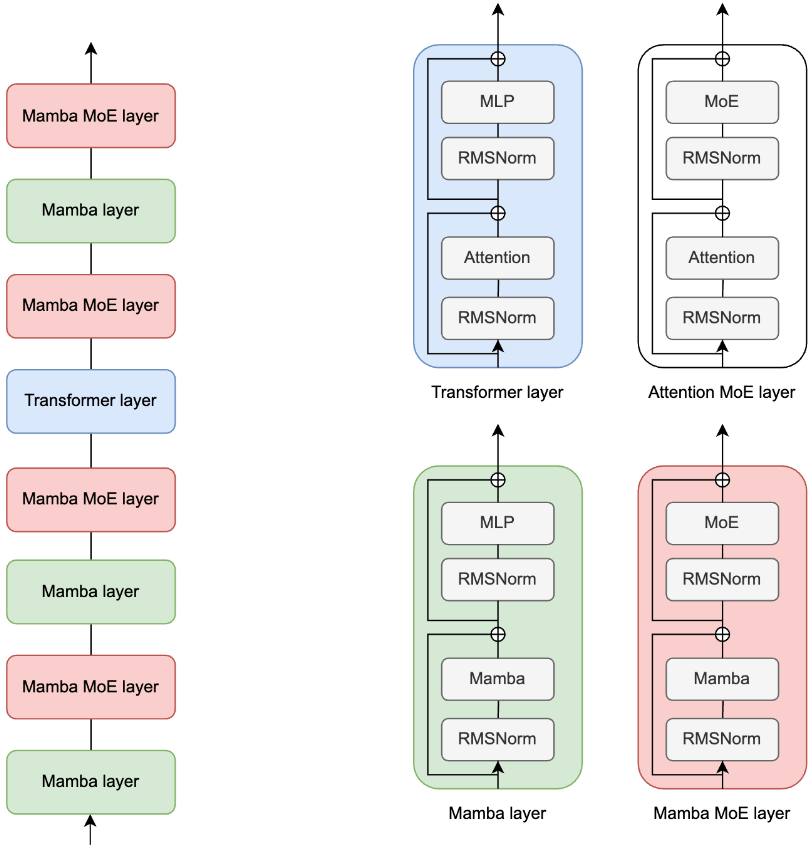
State Space Models (SSMs) for Large Scale Language Modeling

- **Hybrid architectures**
  - SSMs are known to show bad performance in certain scenarios
    - E.g., Mamba shows low accuracy in phone-book retrieval task
  - This is due to SSM's limited random-access capability
    - All information have to be compressed into a single SSM state
    - Random-access is the key advantage of attention mechanism (Transformer)



State Space Models (SSMs) for Large Scale Language Modeling

- **Hybrid architectures**
  - Recent architectures leverage both attention layers and SSM layers together
    - E.g. **Jamba** [Lieber et al., 2024] and **Griffin** [De et al., 2024]
  - Hybrid approach provides both efficiency and random-access capability



(a) Jamba block **(b)** Different types of layers

Overview

Part 1. Basics

- RNN to LSTM
- Sequence-to-sequence Model
- Attention-based NLP Model

Part 2. Transformers and Large Language Models

- Transformer (self-attention)
- Pre-training of Transformers and Language Models

Part 3. Advanced Topics

- Handling long inputs with Transformers
- Techniques for improving efficiency
- State-space models

Part 4. Summary

Summary

- For language, specified model which can capture temporal dependency is a key

- Previously, RNN architectures have developed in a way that
  - Can better model **long-term dependency** & **Robust** to vanishing gradient problems
  - Seq2seq model with **attention makes breakthroughs** in machine translation
  - It leads to the model only composed with attention ⟶ **Transformer**

- **Transformer** significantly improves the performance on **many sequential tasks**
  - With **pre-training** using large model and data, one can get **1)** standard initialization point for many NLP task (BERT) and **2)** strong language generator (GPT)

- Various techniques are emerging to address the remaining **practical challenges**
  - Improving **efficiency** in training and deployment
  - Extending the language models to **handle longer inputs**
  - **State-space models** enable efficient long context handling

References

[Hochreiter and Schmidhuber, 1997] "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.
link: http://www.bioinf.jku.at/publications/older/2604.pdf

[Graves et al., 2005] "Framewise phoneme classification with bidirectional LSTM and other neural network architectures." *Neural Networks* 18.5-6 (2005): 602-610.
Link: ftp://ftp.idsia.ch/pub/juergen/nn_2005.pdf

[Graves et al, 2013] "Speech recognition with deep recurrent neural networks." *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013.
Link: https://www.cs.toronto.edu/~graves/icassp_2013.pdf

[Cho et al., 2014] "Learning phrase representations using RNN encoder-decoder for statistical machine translation." *arXiv preprint arXiv:1406.1078* (2014).
Link: https://arxiv.org/pdf/1406.1078v3.pdf

[Sutskever et al., 2014] "Sequence to sequence learning with neural networks." *NIPS* 2014.
link : http://papers.nips.cc/paper/5346-sequence-to-sequence-learnin

[Sutskever et al., 2014] "Sequence to sequence learning with neural networks." NIPS 2014.

[Bahdanau et al., 2015] “"Neural machine translation by jointly learning to align and translate.“, ICLR 2015
Link: https://arxiv.org/pdf/1409.0473.pdf

[Jozefowicz et al., 2015] "An empirical exploration of recurrent network architectures." *ICML* 2015.
Link: http://proceedings.mlr.press/v37/jozefowicz15.pdf

[Bahdanau et al., 2015] Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." *ICLR 2015*
link : https://arxiv.org/pdf/1409.0473.pdf

References

[Kalchbrenner et al., 2016] "Grid long short-term memory." *ICLR 2016*
Link: https://arxiv.org/pdf/1507.01526.pdf

[Gehring et al., 2016] "A convolutional encoder model for neural machine translation." *arXiv preprint arXiv:1611.02344* (2016).
Link: https://arxiv.org/pdf/1611.02344.pdf

[Wu et al., 2016] "Google's neural machine translation system: Bridging the gap between human and machine translation." *arXiv preprint arXiv:1609.08144* (2016).
link: https://arxiv.org/pdf/1609.08144.pdf

[Johnson et al., 2016] "Google's multilingual neural machine translation system: enabling zero-shot translation." *arXiv preprint arXiv:1611.04558* (2016).
Link: https://arxiv.org/pdf/1611.04558.pdf

[Gehring et al., 2017] "Convolutional sequence to sequence learning." *arXiv preprint arXiv:1705.03122* (2017).
Link: https://arxiv.org/pdf/1705.03122.pdf

[Narang et al., 2017] "Exploring sparsity in recurrent neural networks.“, ICLR 2017
Link: https://arxiv.org/pdf/1704.05119.pdf

[Fei-Fei and Karpathy, 2017] “CS231n: Convolutional Neural Networks for Visual Recognition”*, 2017*. (Stanford University)
link : http://cs231n.stanford.edu/2017/

[Salehinejad et al., 2017] "Recent Advances in Recurrent Neural Networks." *arXiv preprint arXiv:1801.01078* (2017).
Link: https://arxiv.org/pdf/1801.01078.pdf

References

[Zaheer et al., 2020] “Big Bird: Transformers for Longer Sequences.” *NeurIPS 2020*
Link: https://arxiv.org/pdf/2007.14062.pdf

[Wang et al., 2020] “Linformer: Self-Attention with Linear Complexity.” *arXiv preprint arXiv:2006.04768*
Link: https://arxiv.org/pdf/2006.04768.pdf

[Choromanski et al., 2020] “Rethinking Attention with Performers.” *ICLR 2021*
link: https://arxiv.org/pdf/2009.14794.pdf

[Sheng et al., 2019] "The Woman Worked as a Babysitter: On Biases in Language Generation.” *EMNLP 2019*
Link: https://arxiv.org/pdf/1909.01326.pdf

[Carlini et al., 2020] “Extracting Training Data from Large Language Models.” *arXiv preprint arXiv:2012.07805*
Link: https://arxiv.org/pdf/2012.07805.pdf

[Vaswani et al., 2017] “Attention Is All You Need.” *NeurIPS 2017*
Link: https://arxiv.org/pdf/1706.03762.pdf

[Radford et al., 2018] “Improving Language Understanding by Generative Pre-training.” *OpenAI*
Link: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[Radford et al., 2019] “Language Models are Unsupervised Multitask Learners.” *OpenAI*
Link: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[Brown et al., 2020] “Language Models are Few-Shot Learners.” *NeurIPS 2020*
Link: https://arxiv.org/abs/2005.14165

[Devlin et al., 2018] “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” *EMNLP 2019*
Link: https://arxiv.org/abs/1810.04805

[Liu et al., 2019] “RoBERTa: A Robustly Optimized BERT Pretraining Approach.” *arXiv preprint arXiv:1907.11692*
Link: https://arxiv.org/pdf/1907.11692.pdf

References

[Shaw et al., 2018] "Self-attention with Relative Position Representations." *NAACL 2018*
Link: https://arxiv.org/abs/1803.02155

[Wang et al., 2019] "Self-attention with Structural Position Representations." *EMNLP 2019*
Link: https://arxiv.org/pdf/1909.00383.pdf

[Huang et al., 2018] "Music Transformer." *arXiv:1809.04281*
Link: https://arxiv.org/abs/1809.04281

[Girdhar et al., 2018] "Video Action Transformer Network." *CVPR 2019*
Link: https://arxiv.org/pdf/1812.02707.pdf

[Shazeer et al., 2017] "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer." ICLR 2017
Link: https://arxiv.org/abs/1701.06538

[Du et al., 2022] "GLaM: Efficient Scaling of Language Models with Mixture-of-Experts" ICML 2022
Link: https://arxiv.org/abs/2112.06905

[Dao et al., 2022] "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness" NeurIPS 2022
Link: https://arxiv.org/abs/2205.14135

[He, 2022] "Making Deep Learning Go Brrrr From First Principles"
Link: https://horace.io/brrr_intro.html

[Kwon et al., 2023] "Efficient Memory Management for Large Language Model Serving with PagedAttention" SOSP 2023
Link: https://arxiv.org/abs/2309.06180

References

[Dai et al., 2019] “Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context” ACL 2019
Link: https://arxiv.org/abs/1901.02860

[Bulatov et al., 2022] “Recurrent Memory Transformer” NeurIPS 2022
Link: https://arxiv.org/abs/2207.06881

[Bulatov et al., 2023] “Scaling Transformer to 1M tokens and beyond with RMT” arxiv 2023
Link: https://arxiv.org/abs/2304.11062

[Wu et al., 2022] “Memorizing Transformers” ICLR 2022
Link: https://arxiv.org/abs/2203.08913

[Wang et al., 2023] “Augmenting Language Models with Long-Term Memory” arxiv 2023
Link: https://arxiv.org/abs/2306.07174

[Bertsch et al., 2023] “Unlimiformer: Long-Range Transformers with Unlimited Length Input” NeurIPS 2023
Link: https://arxiv.org/abs/2305.01625

[Mohtashami et al., 2023] “Landmark Attention: Random-Access Infinite Context Length for Transformers” NeurIPS 2023
Link: https://arxiv.org/abs/2305.16300

[Su et al., 2021] “RoFormer: Enhanced Transformer with Rotary Position Embedding” arxiv 2021
Link: https://arxiv.org/abs/2104.09864

[Chen et al., 2023] “Extending Context Window of Large Language Models via Positional Interpolation” arxiv 2023
Link: https://arxiv.org/abs/2306.15595

References

[kaikoendev, 2023] “Things I’m learning while training superhot”
Link: https://kaiokendev.github.io/til#extending-context-to-8k.

[bloc97, 2023] “NTK-Aware Scaled RoPE allows LLaMA models to have extended (8k+) context size without any fine-tuning and minimal perplexity degradation.”
Link:
https://www.reddit.com/r/LocalLLaMA/comments/14lz7j5/ntkaware_%20scaled_rope_allows_llama_models_to_have

[Peng et al., 2023] “YaRN: Efficient Context Window Extension of Large Language Models” ICLR 2024
Link: https://arxiv.org/abs/2309.00071

[Gu et al., 2020] “HiPPO: Recurrent Memory with Optimal Polynomial Projections” NeurIPS 2020
Link: https://arxiv.org/abs/2008.07669

[Gu et al., 2022] “Efficiently Modeling Long Sequences with Structured State Spaces” ICLR 2022
Link: https://arxiv.org/abs/2111.00396

[Gu et al., 2022] “On the Parameterization and Initialization of Diagonal State Space Models” NeurIPS 2022
Link: https://arxiv.org/abs/2206.11893

[Gu et al., 2024] “Mamba: Linear-Time Sequence Modeling with Selective State Spaces” arxiv 2023
Link: https://arxiv.org/abs/2312.00752

[Smith et al., 2022] “Simplified State Space Layers for Sequence Modeling” arxiv 2022
Link: https://arxiv.org/abs/2208.04933

[Jelassi et al., 2024] “Repeat After Me: Transformers are Better than State Space Models at Copying” arxiv 2024
Link: https://arxiv.org/abs/2402.01032

References

[Lieber et al., 2024] “Jamba: A Hybrid Transformer-Mamba Language Model” arxiv 2024
Link: https://arxiv.org/abs/2403.19887v2

[De et al., 2024] “Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models” arxiv 2024
Link: https://arxiv.org/abs/2402.19427