

Recent neural architectures for language

AI602: Recent Advances in Deep Learning

Lecture 3

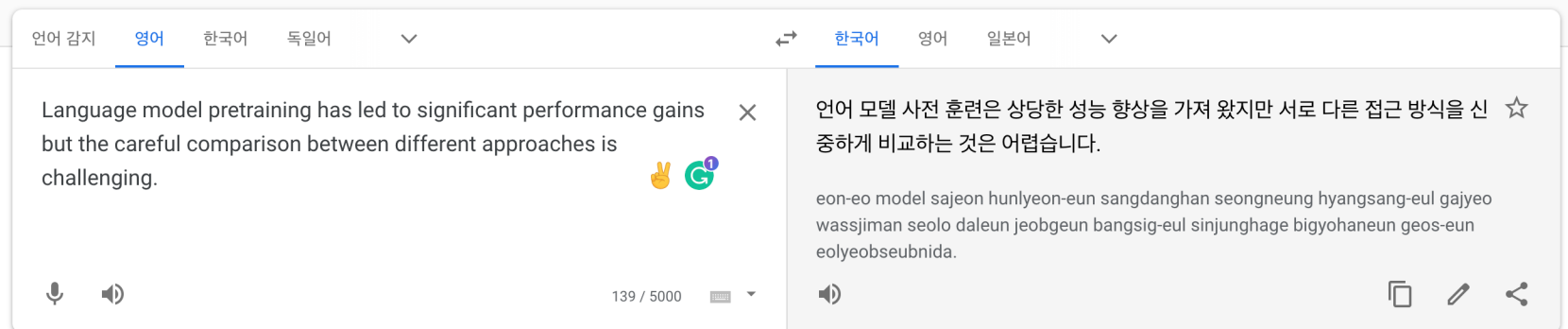
KAIST AI

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

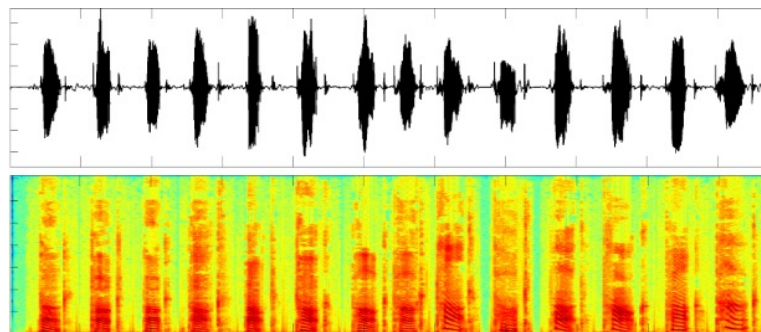


The screenshot shows the Google Translate web interface. The source language is English and the target language is Korean. The input text is: "Language model pretraining has led to significant performance gains but the careful comparison between different approaches is challenging." The output text is: "언어 모델 사전 훈련은 상당한 성능 향상을 가져 왔지만 서로 다른 접근 방식을 신중하게 비교하는 것은 어렵습니다." Below the output, the Korean text is repeated in a smaller font: "eon-eo model sajeon hunlyeon-eun sangdanghan seongneung hyangsang-eul gajyeo wassjiman seolo daleun jeobgeun bangsig-eul sinjunhage bigyohaneun geos-eun eolyeobseubnida." The interface includes a microphone icon, a speaker icon, a character count (139 / 5000), and various utility icons like copy, edit, and share.

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



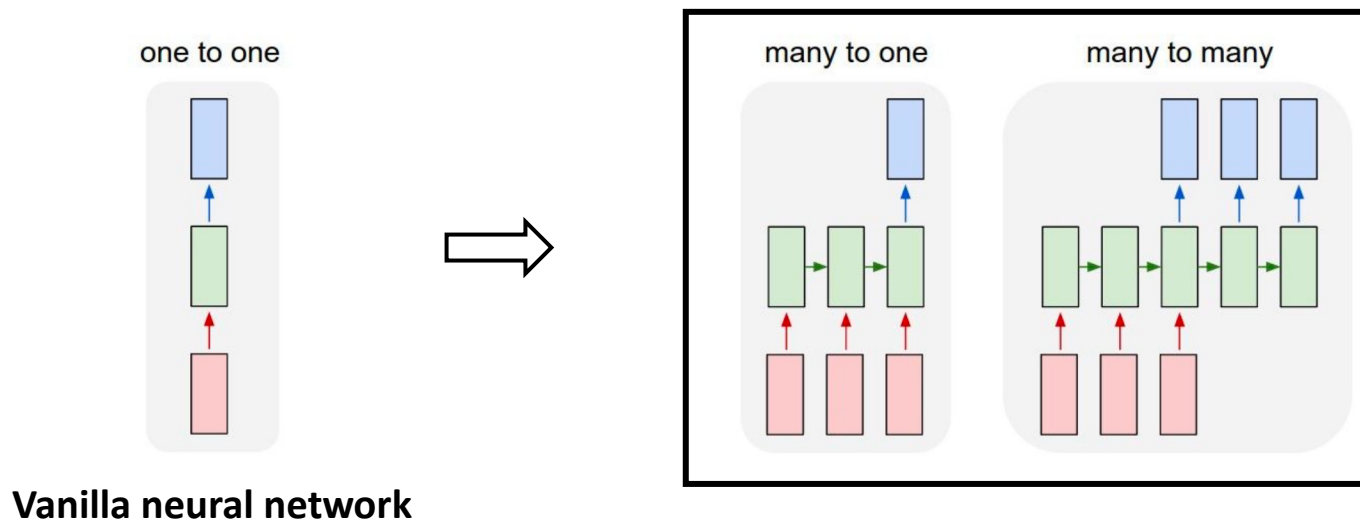
Motivation: Temporal Data in Real World

- 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



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

- Techniques for improving efficiency
- Handling long inputs with Transformers

Part 4. Summary

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

- Techniques for improving efficiency
- Handling long inputs with Transformers

Part 4. Summary

Vanilla RNN

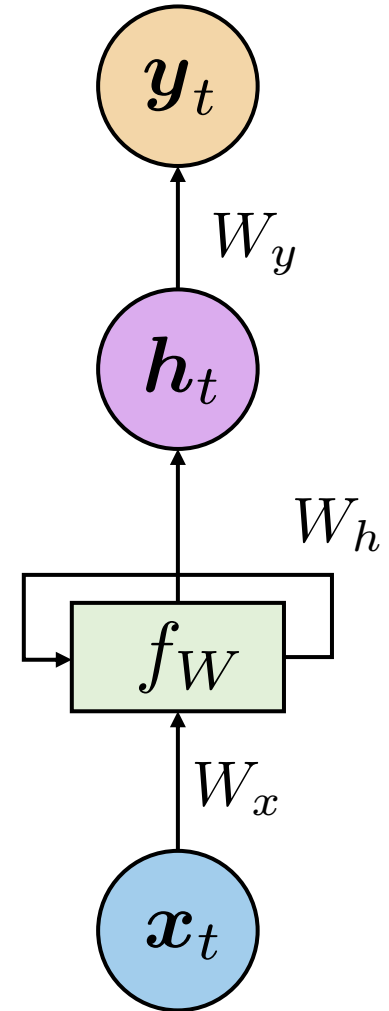
- Vanilla RNN (or sometimes called **Elman RNN**)
 - The state consists of a single “hidden” vector \mathbf{h}_t

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



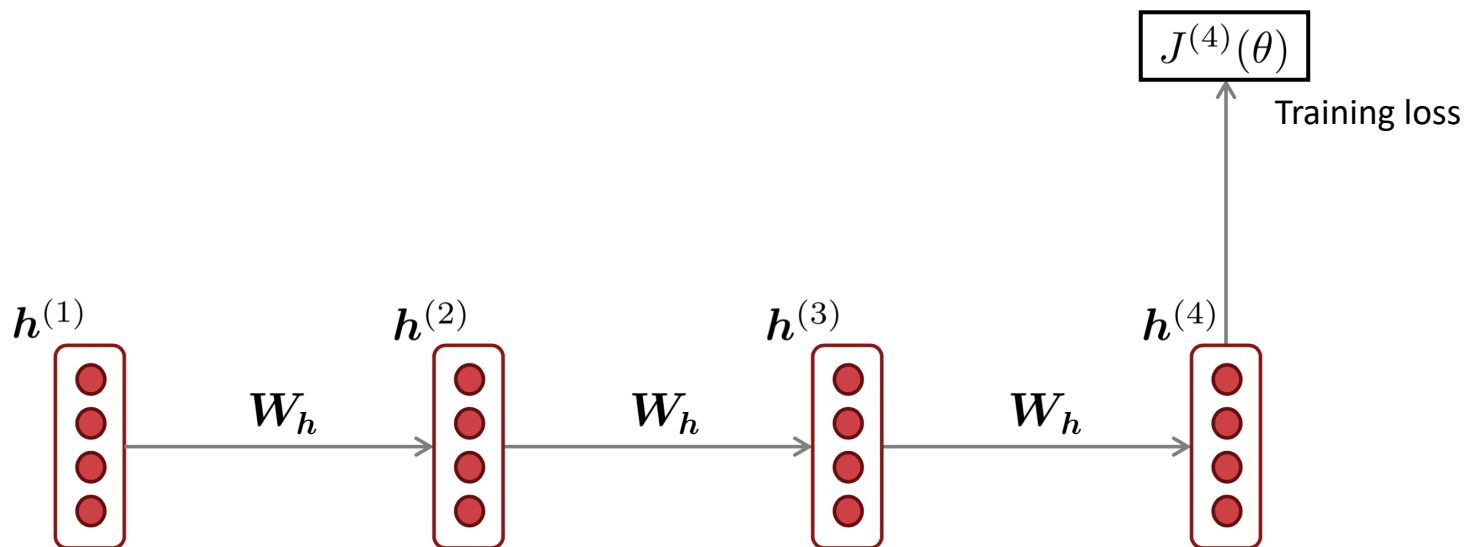
$$\mathbf{h}_t = \tanh(W_h \mathbf{h}_{t-1} + W_x \mathbf{x}_t)$$

$$\mathbf{y}_t = W_y \mathbf{h}_t$$



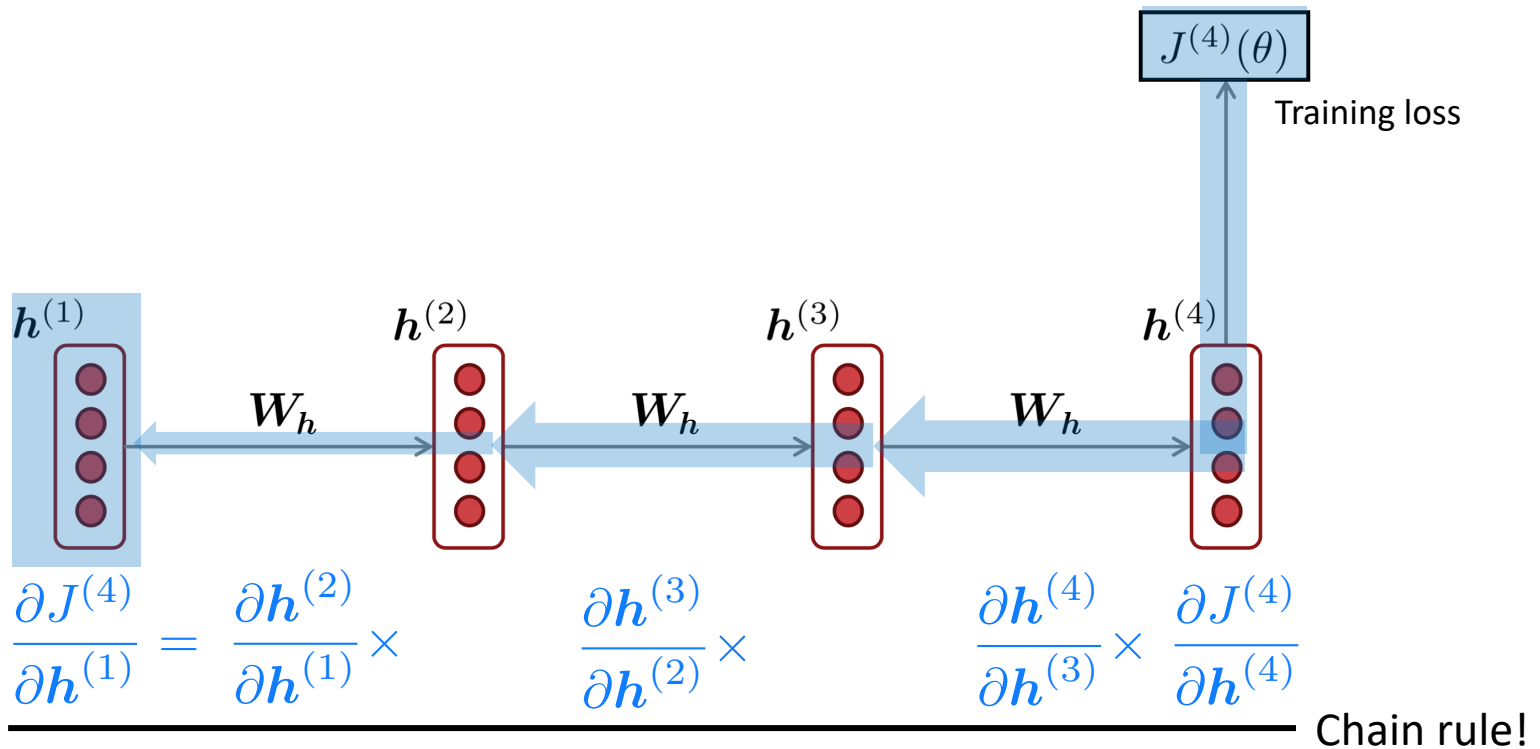
Why Do We Need to Develop RNN Architectures?

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



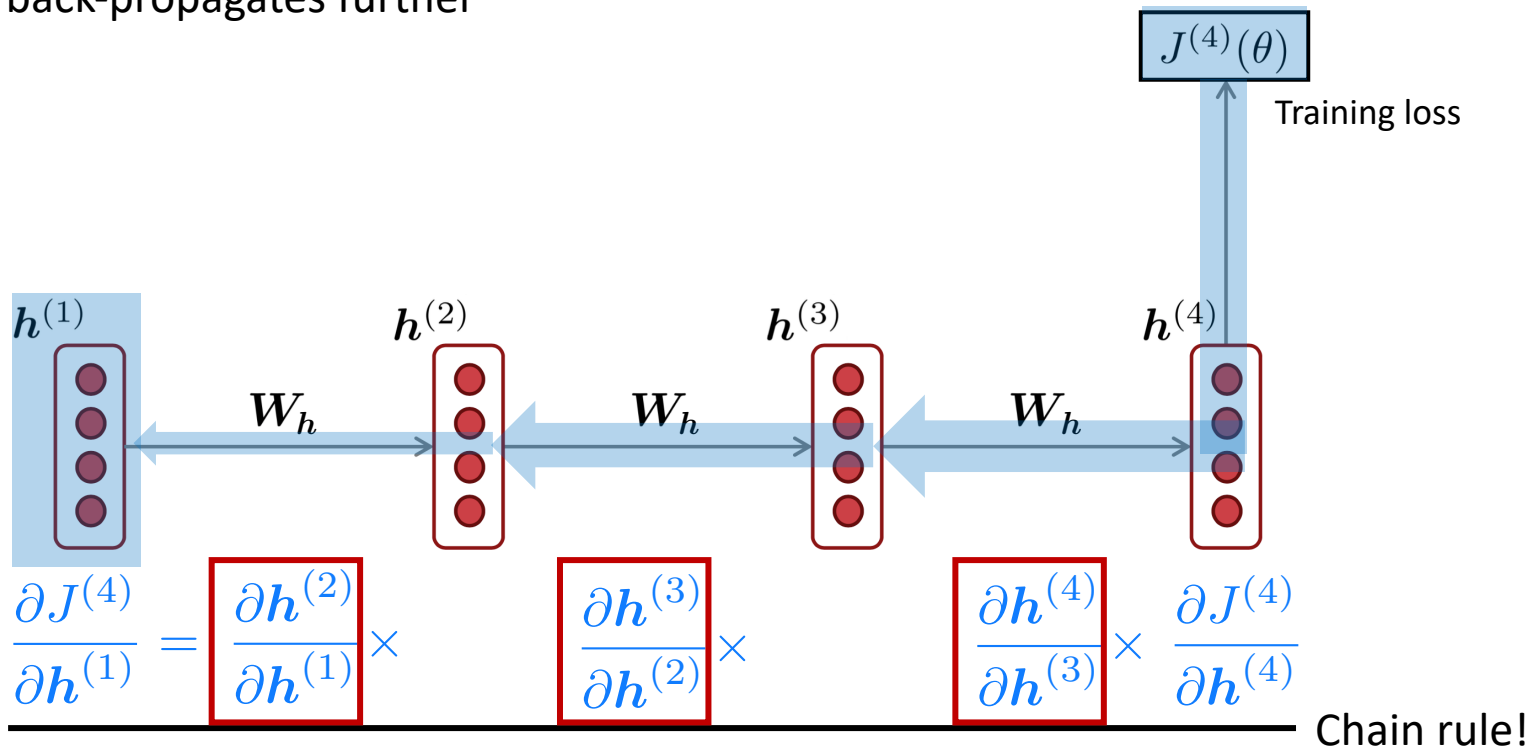
Why Do We Need to Develop RNN Architectures?

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



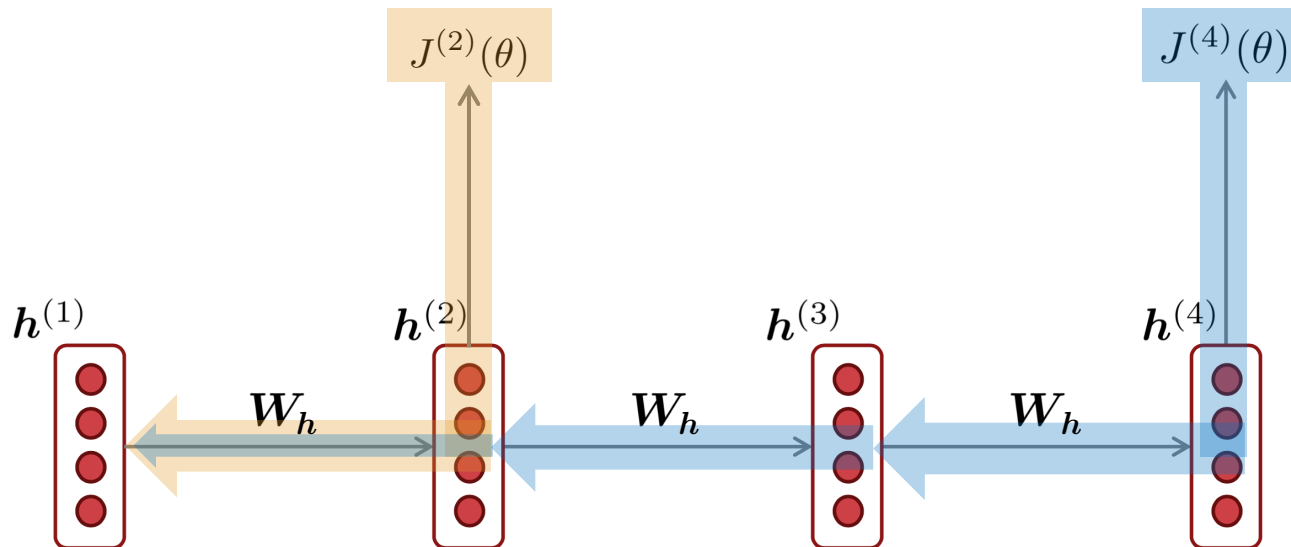
Why Do We Need to Develop RNN Architectures?

- E.g., RNN with a sequence of length 4
 - Consider a gradient from the first state $h^{(1)}$
- What happens if $\frac{\partial h^{(i+1)}}{\partial h^{(i)}}$ are **too small**? \Rightarrow **Vanishing gradient problem**
 - When these are small, the gradient signal **gets smaller and smaller** as it back-propagates further



Why Do We Need to Develop RNN Architectures?

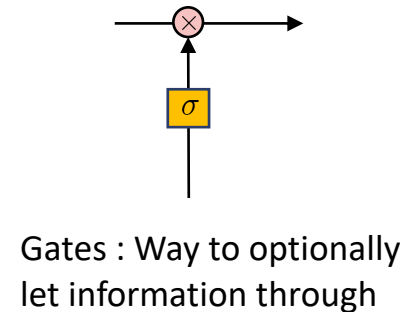
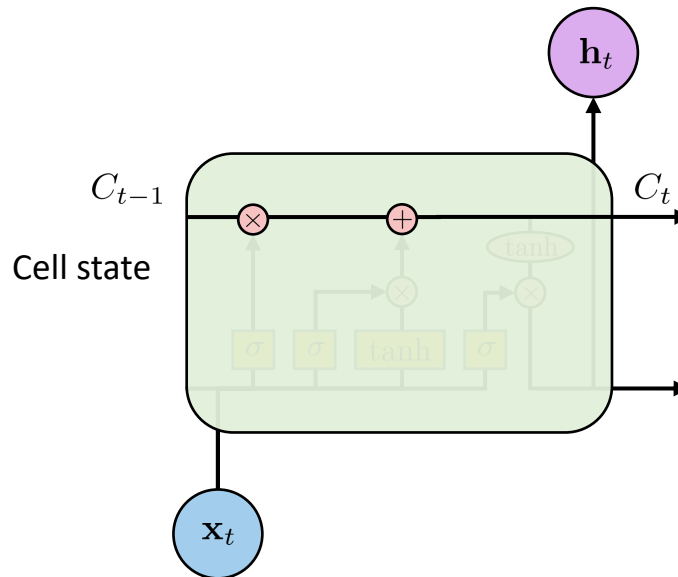
- E.g., RNN with a sequence of length 4
 - Consider a gradient from the first state $h^{(1)}$
- What happens if $\frac{\partial h^{(i+1)}}{\partial h^{(i)}}$ are **too small**? \Rightarrow **Vanishing gradient problem**
 - When these are small, the gradient signal **gets smaller and smaller** as it back-propagates further
 - So, model weights 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 $h^{(1)}$
- What happens if $\frac{\partial h^{(i+1)}}{\partial h^{(i)}}$ are **too small**? \Rightarrow **Vanishing gradient problem**
 - When these are small, the gradient signal **gets smaller and smaller** as it back-propagates further
 - So, model weights are updated only with respect to **near effects**, **not long-term effects**.
- What happens if $\frac{\partial h^{(i+1)}}{\partial h^{(i)}}$ are **too large**? \Rightarrow **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**

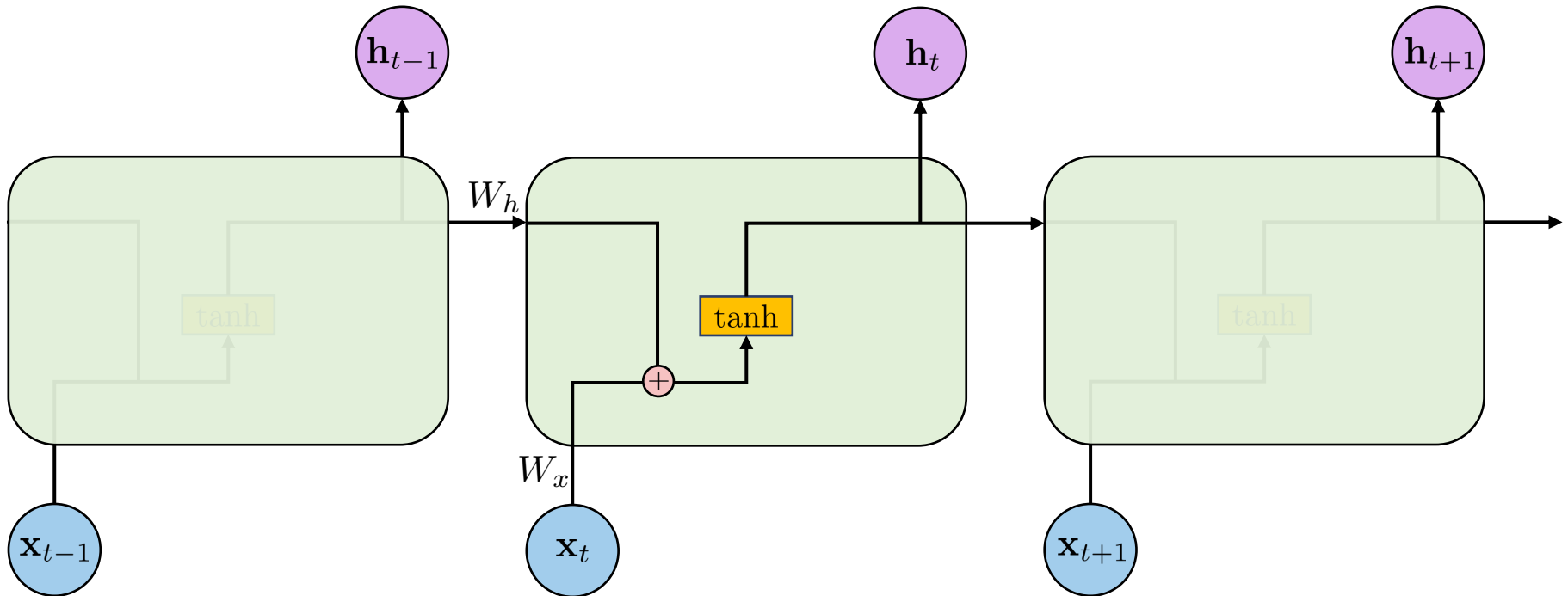
- **Long Short-Term Memory (LSTM)** [Hochreiter and Schmidhuber, 1997]
 - A special type of RNN unit, i.e., LSTM networks = RNN composed of LSTM units
 - Explicitly designed RNN to
 - Capture **long-term dependency** \Rightarrow more robust to vanishing gradient problem
- Core idea behind LSTM
 - With **cell state (memory)**, it controls **how much to remove or add information**
 - Only linear interactions from the output of each “gates” (**prevent vanishing gradient**)



RNN Architectures: Vanilla RNN

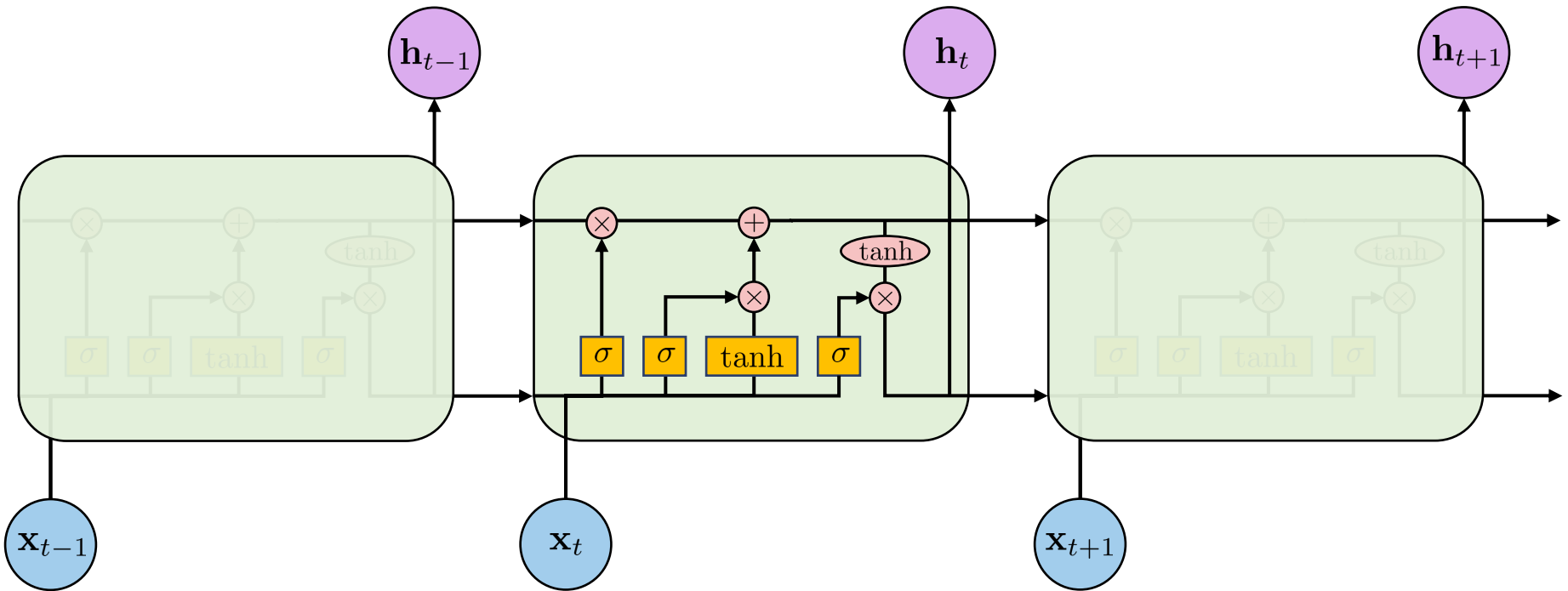
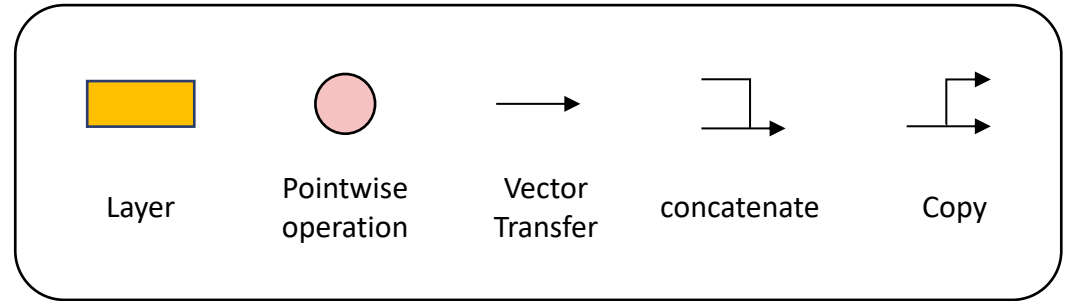
- Repeating modules in **Vanilla RNN** contains a **single layer**

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



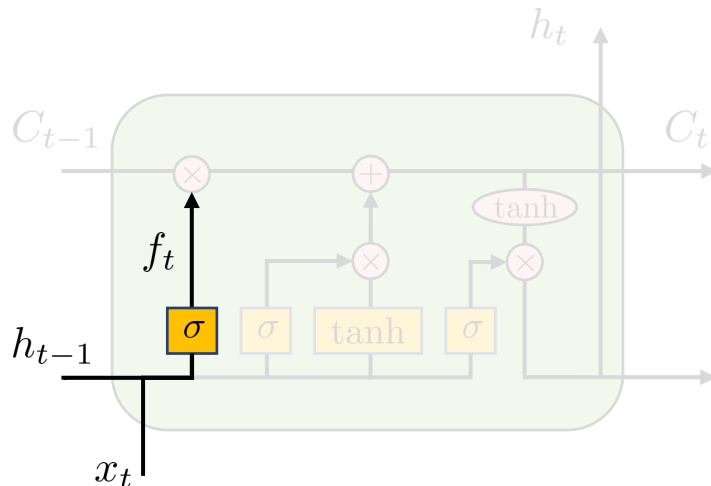
RNN Architectures: LSTM

- Repeating modules in LSTM



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

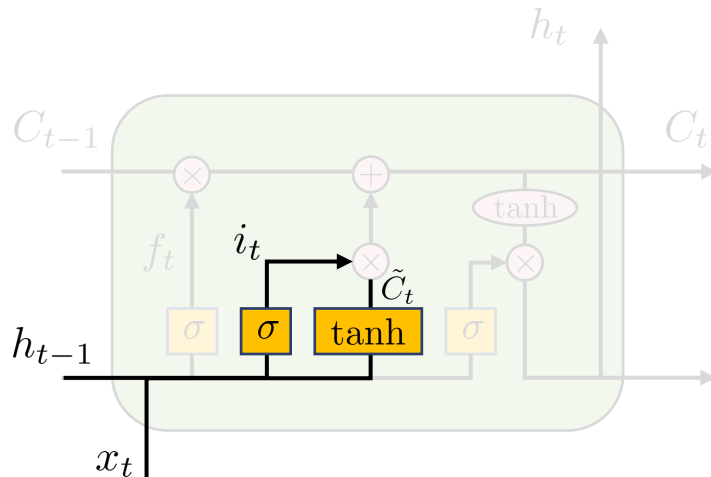
- A sigmoid layer called “**Forget gate**” f_t
- Looks at h_{t-1}, x_t and outputs a number between 0 and 1 for each cell state C_{t-1}
 - If 1: completely keep, if 0: completely remove
- E.g., language model trying to **predict the next word** based on all previous ones
 - The cell state might include 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



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

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

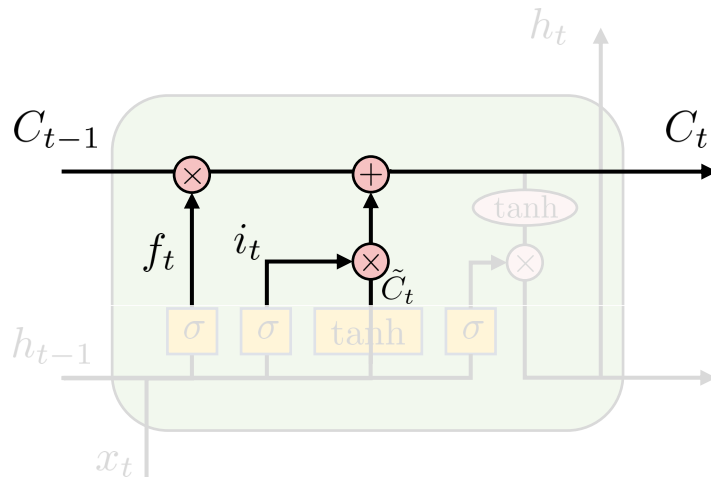


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

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)



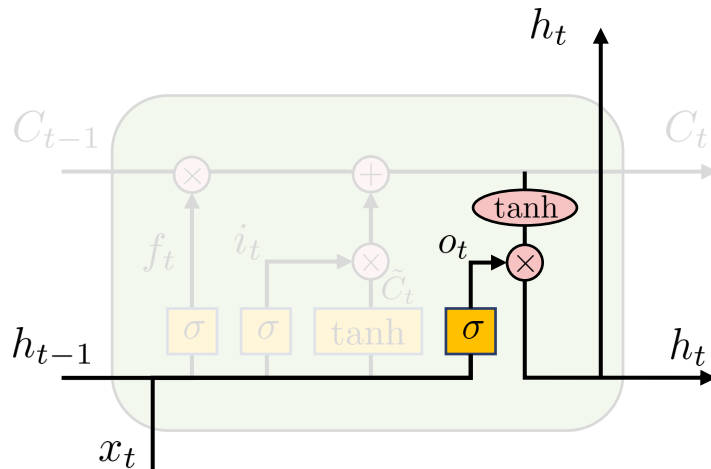
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

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

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

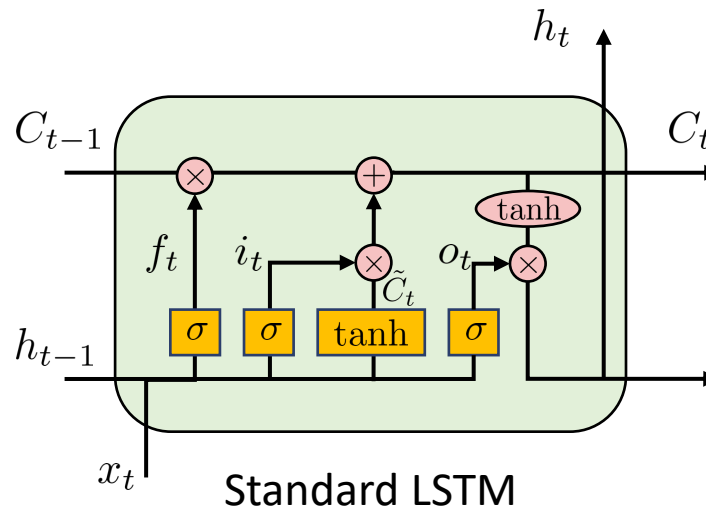
Forget gate: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$ Input gate: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

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

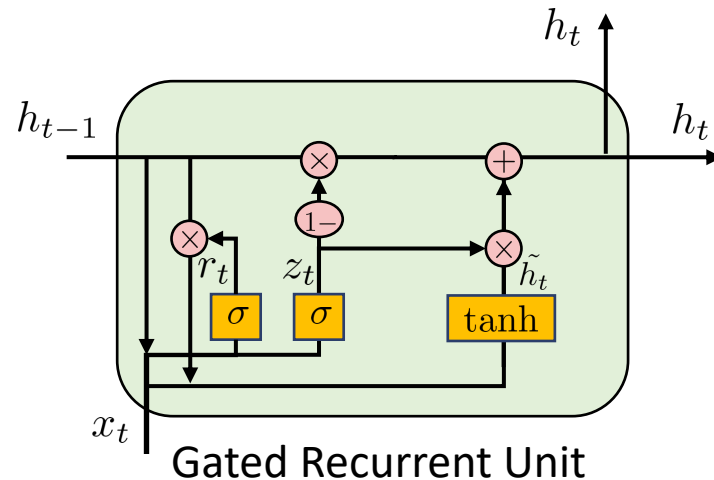
Output gate: $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$

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



- **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

$$\begin{aligned} \text{Reset gate: } r_t &= \sigma(W_r \cdot [h_{t-1}, x_t]) & \text{New content: } \tilde{h}_t &= \tanh(W \cdot [r_t * h_{t-1}, x_t]) \\ \text{Update gate: } z_t &= \sigma(W_z \cdot [h_{t-1}, x_t]) & \text{Hidden state: } h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \end{aligned}$$



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

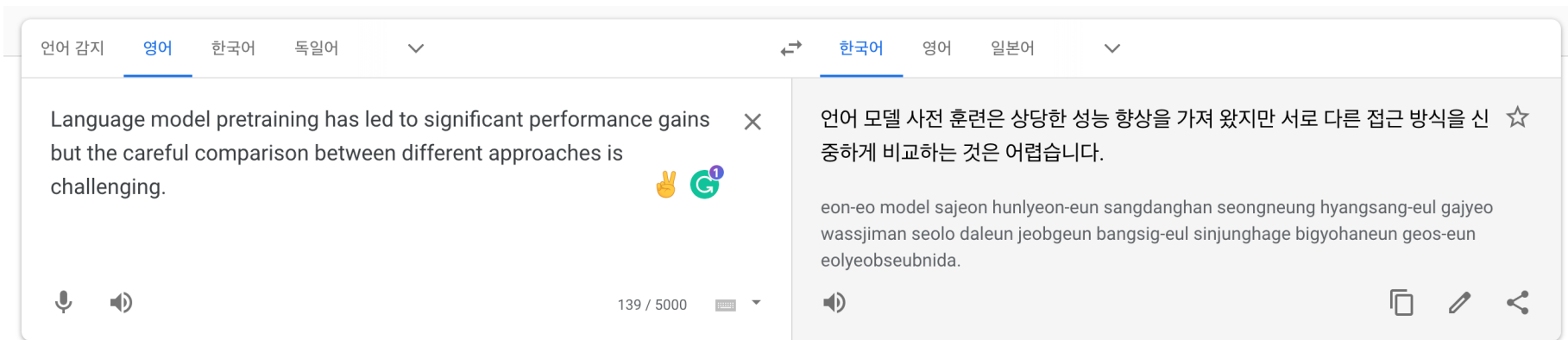
Part 3. Advanced Topics

- Techniques for improving efficiency
- Handling long inputs with Transformers

Part 4. Summary

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

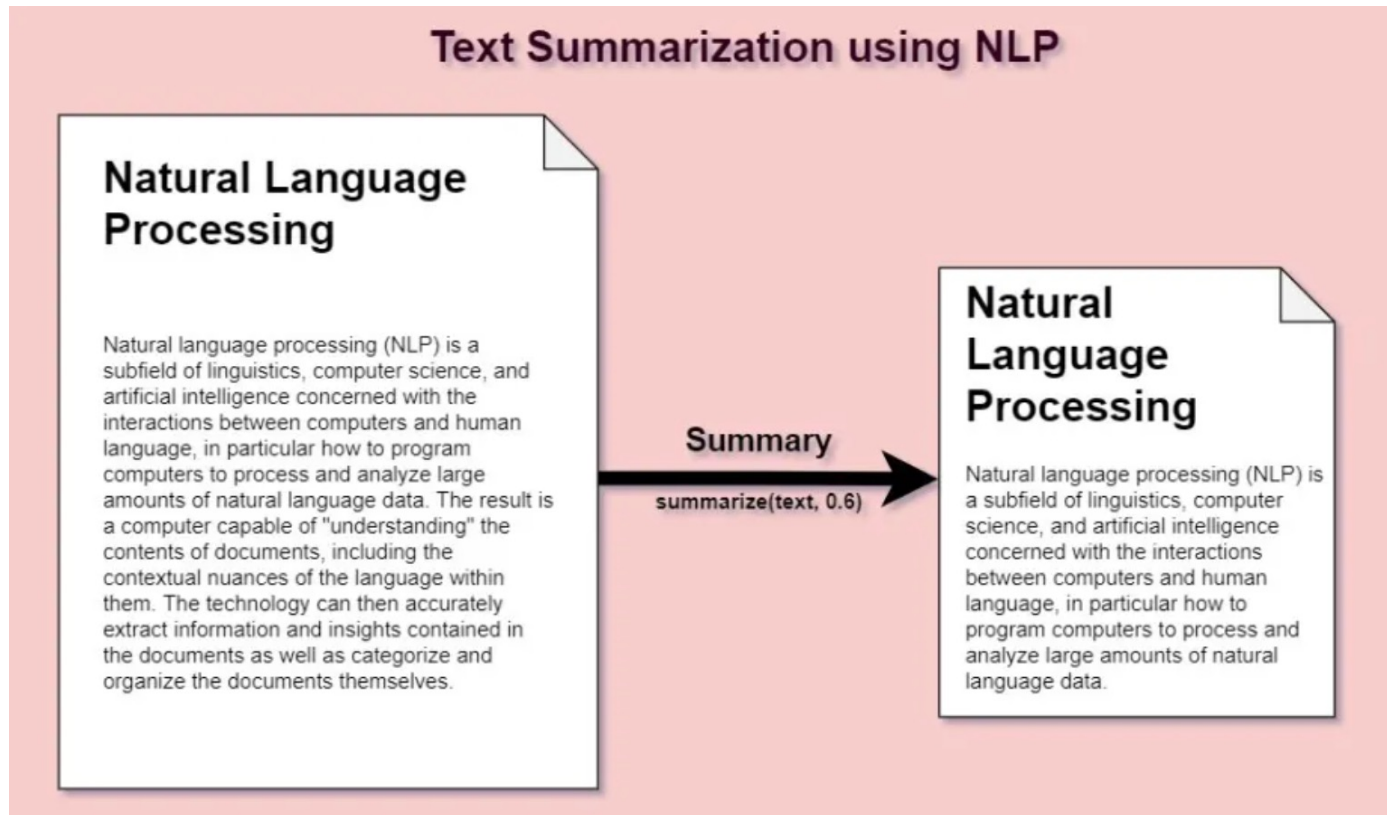
- Many natural language processing (NLP) tasks are **Sequence-to-sequence**
 - Given an input sequence, turn it into an output sequence
 - Example: **Translation**



The screenshot shows a Google Translate interface. The top bar has language selection options: '언어 감지' (Language detection), '영어' (English), '한국어' (Korean), and '독일어' (German). The main content area is split into two panels. The left panel shows the input text: 'Language model pretraining has led to significant performance gains but the careful comparison between different approaches is challenging.' Below the text are icons for voice input, voice output, and a character count '139 / 5000'. The right panel shows the translated text in Korean: '언어 모델 사전 훈련은 상당한 성능 향상을 가져 왔지만 서로 다른 접근 방식을 신중하게 비교하는 것은 어렵습니다.' Below the Korean text is a phonetic transcription: 'eon-eo model sajeon hunlyeon-eun sangdanghan seongneung hyangsang-eul gajyeo wassjiman seolo daleun jeobgeun bangsig-eul sinjunhage bigyohaneun geos-eun eolyeobseubnida.' There are also icons for voice output, copy, edit, and share.

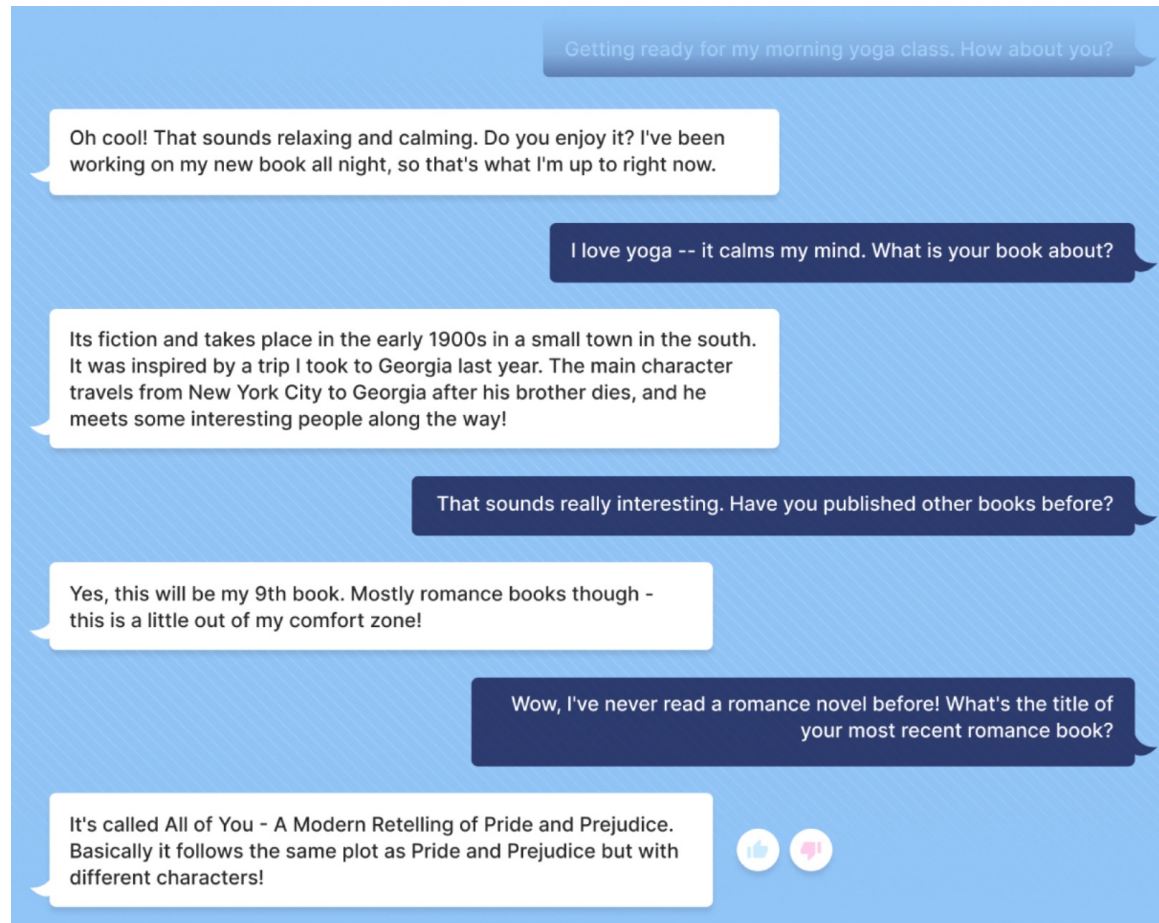
Motivation: Natural Language Processing and Sequence-to-sequence Modeling

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



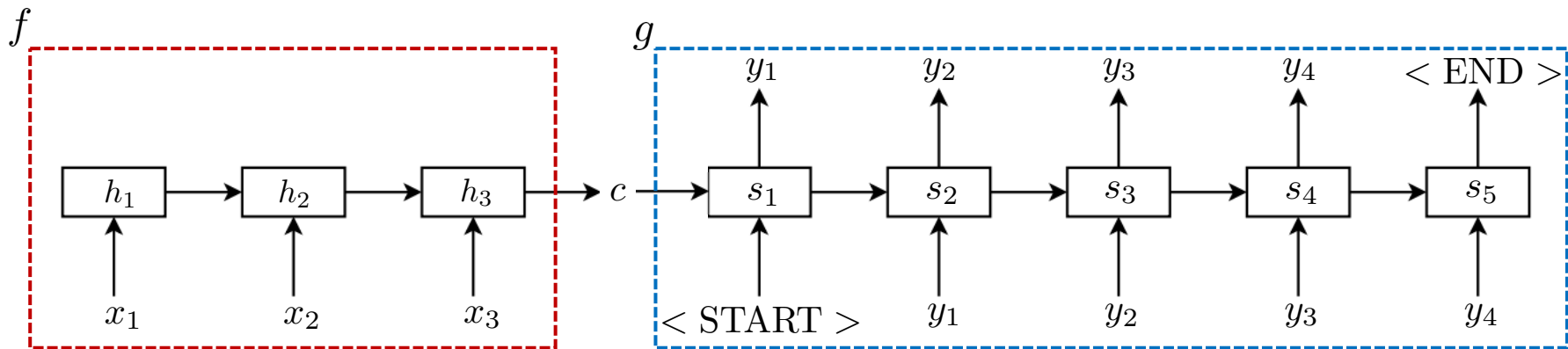
Motivation: Natural Language Processing and Sequence-to-sequence Modeling

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



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 \rightarrow vector \rightarrow 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

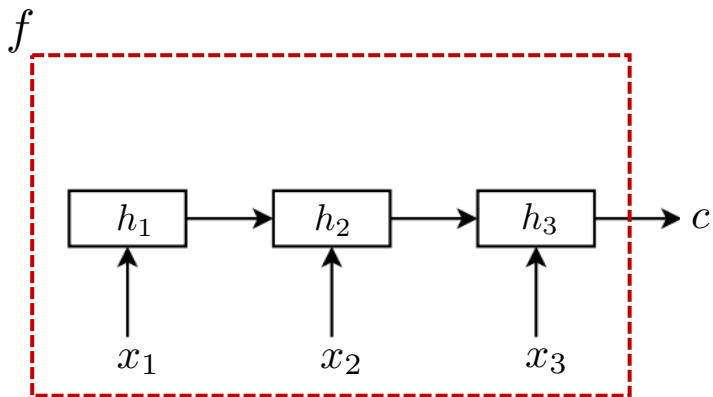


Input sequence $x = (x_1, x_2, x_3)$ and output sequence $y = (y_1, y_2, y_3, y_4)$

Sequence-to-sequence Model

- **Encoder**

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



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

Sequence-to-sequence Model

• Decoder

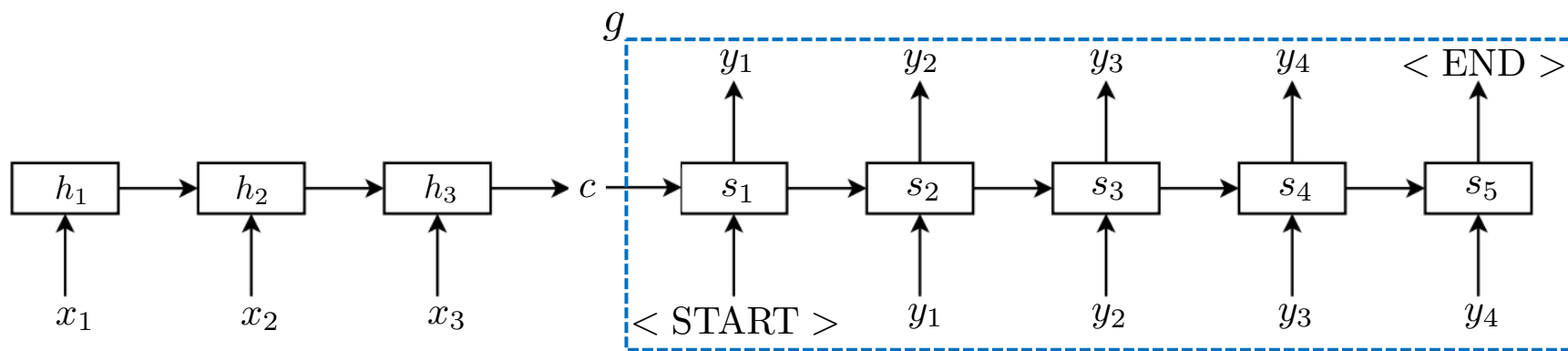
- **Predict the next word** $y_{t'}$ **given the context vector** c and the **previously predicted words** $\{y_1, \dots, y_{t'-1}\}$
- Defines a probability over the translation \mathbf{y} by **decomposing the joint probability** into the ordered conditionals where $\mathbf{y} = (y_1, \dots, y_T)$.

$$p(\mathbf{y}) = \prod_{t=1}^T p(y_t | \{y_1, \dots, y_{t-1}\}, c),$$

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

$$p(y_t | \{y_1, \dots, y_{t-1}\}, c) = g(y_{t-1}, \underline{s_t}, c),$$

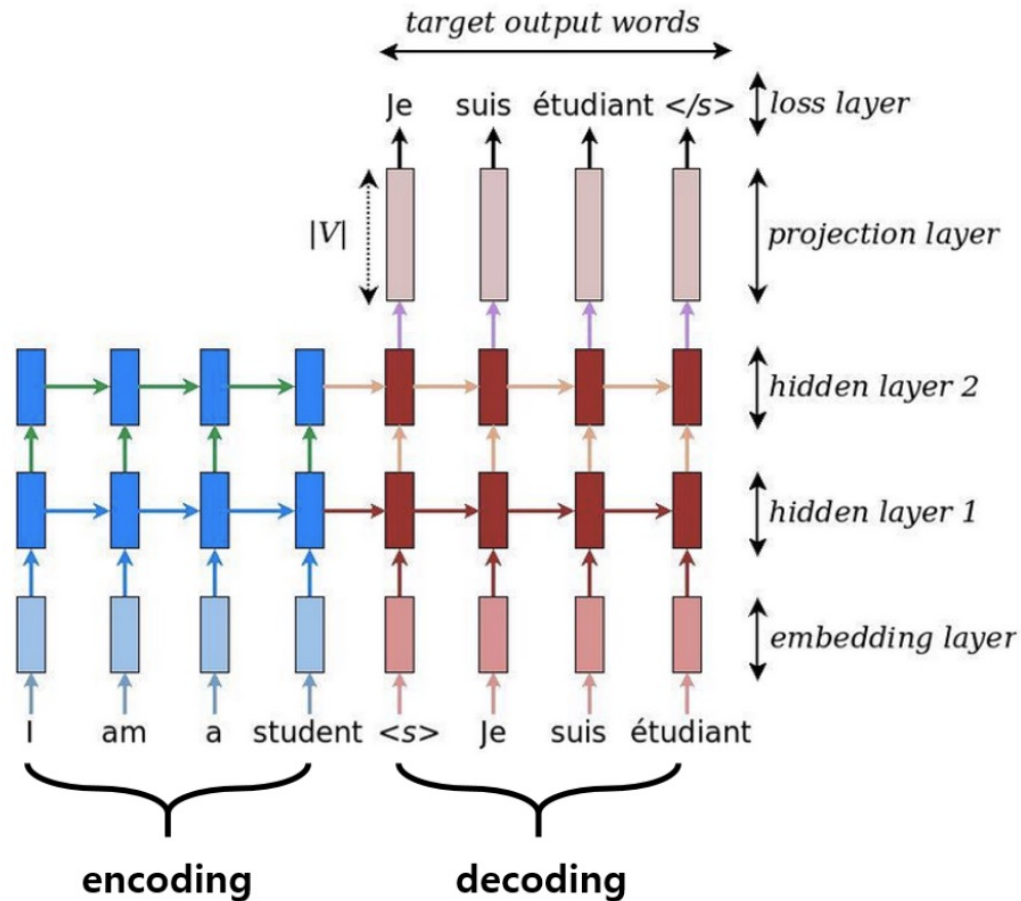
hidden state of the RNN



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

Sequence-to-sequence Model

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



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

- Techniques for improving efficiency
- Handling long inputs with Transformers

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

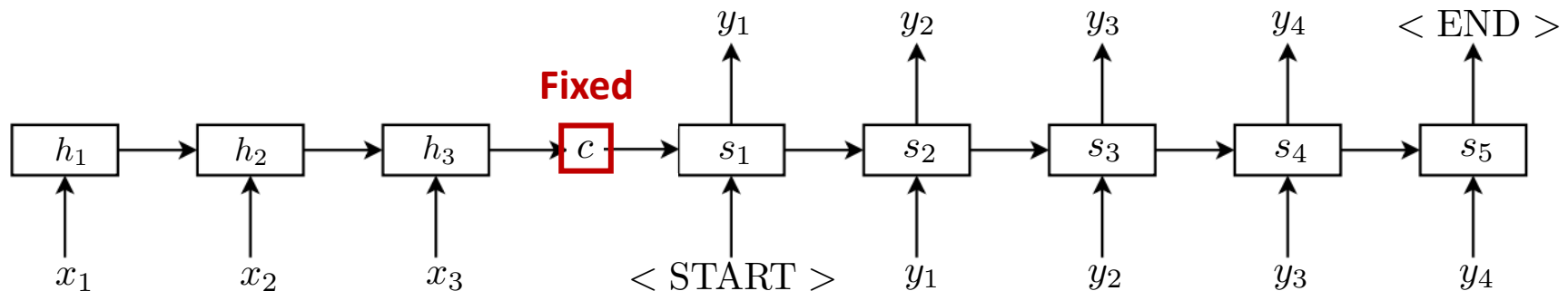
$$p(y_t | \{y_1, \dots, y_{t-1}\}, c) = g(y_{t-1}, s_t, \underline{c}),$$

- 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”

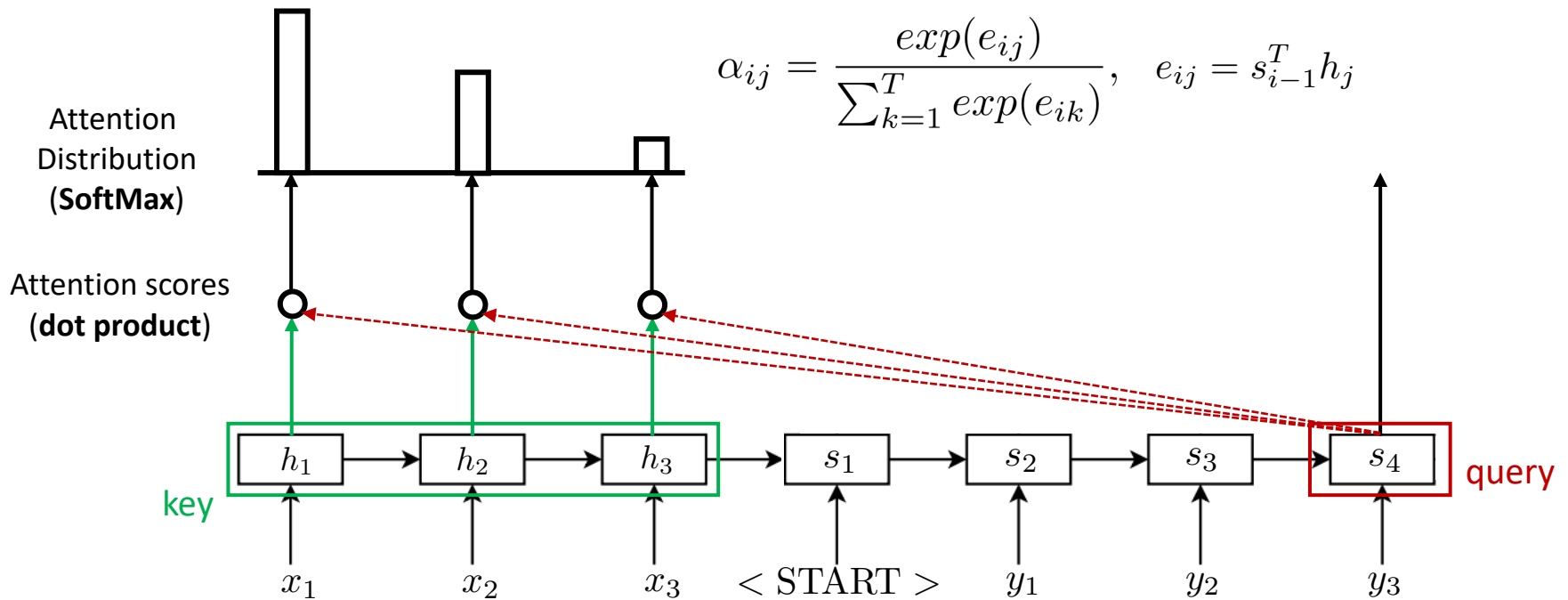
- Hence, **difficult to cope with long sentences...**



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

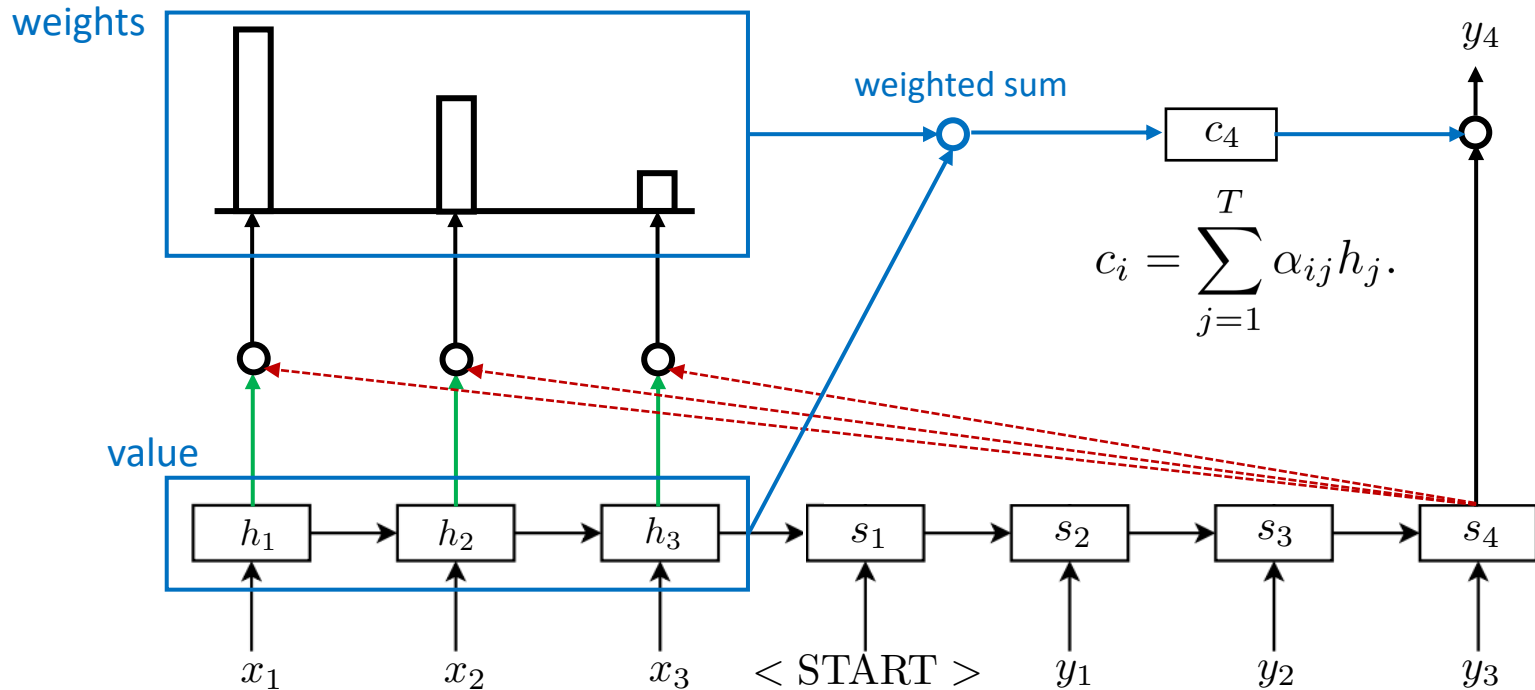
Attention-based Sequence-to-sequence Model

- 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.



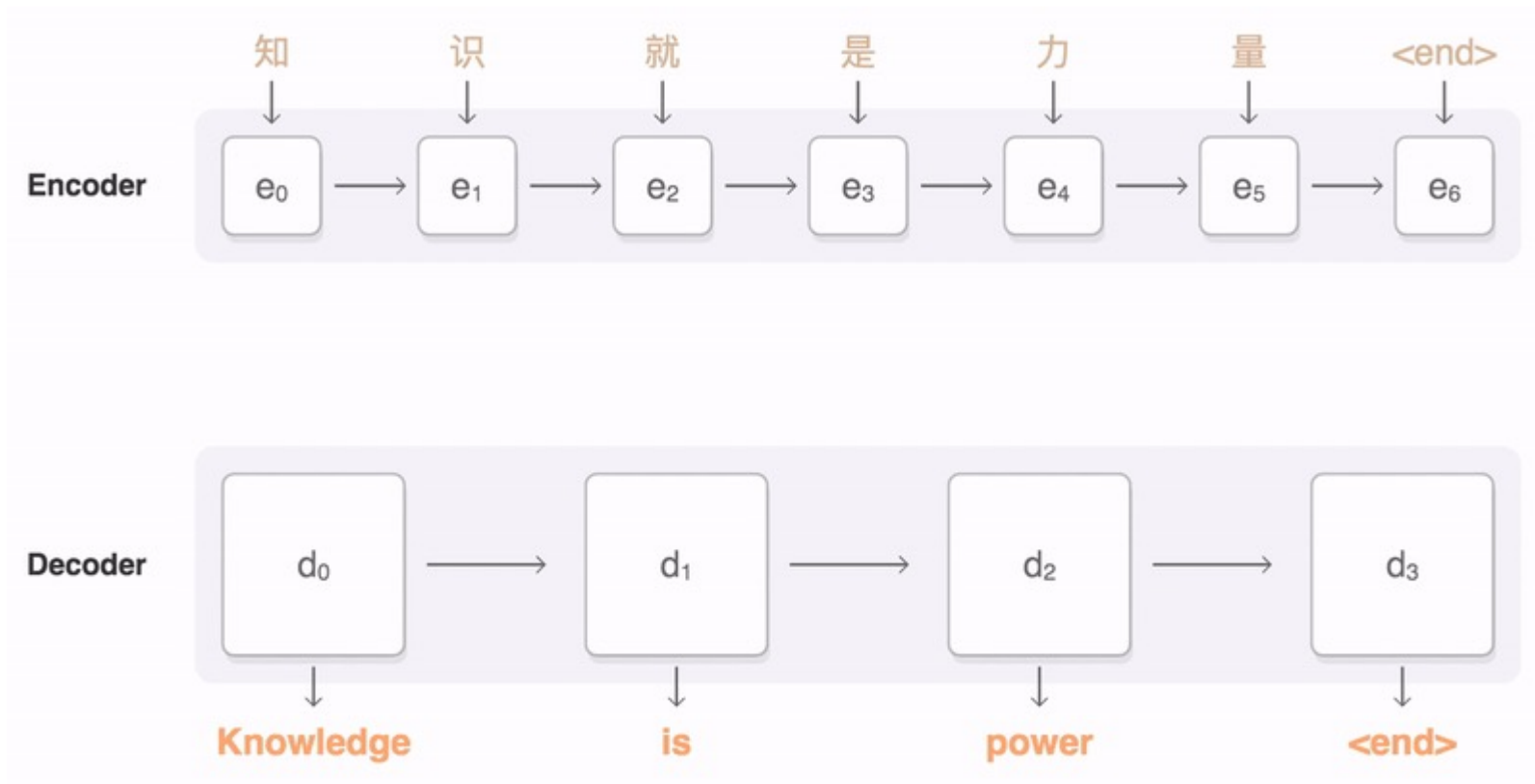
Attention-based Sequence-to-sequence Model

- 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



Attention-based Sequence-to-sequence Model

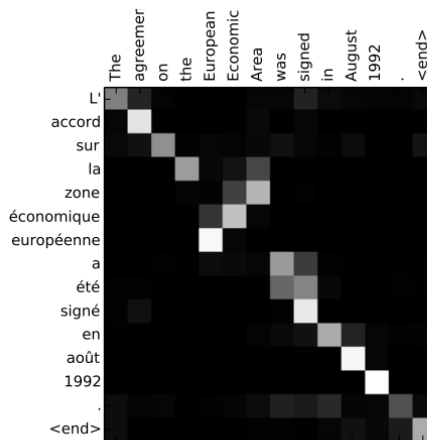
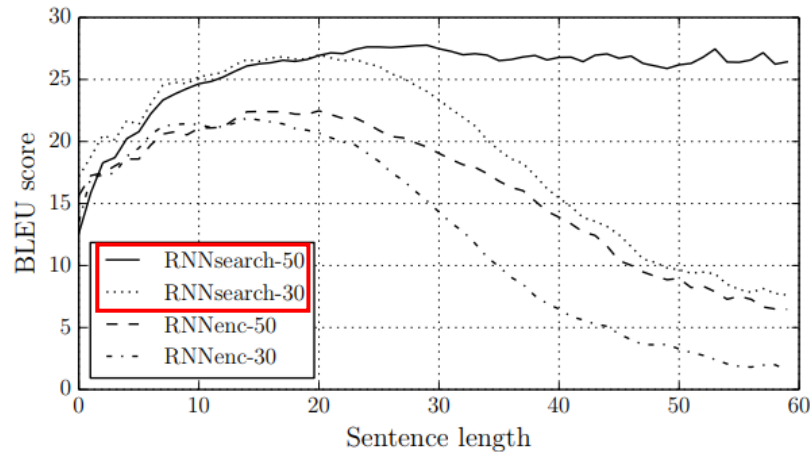
- **Graphical illustration** of seq2seq with **attention**
 - E.g., Chinese to English



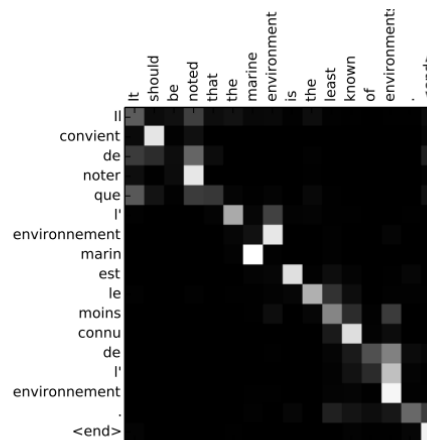
Attention-based Sequence-to-sequence Model

• Results

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



(a)

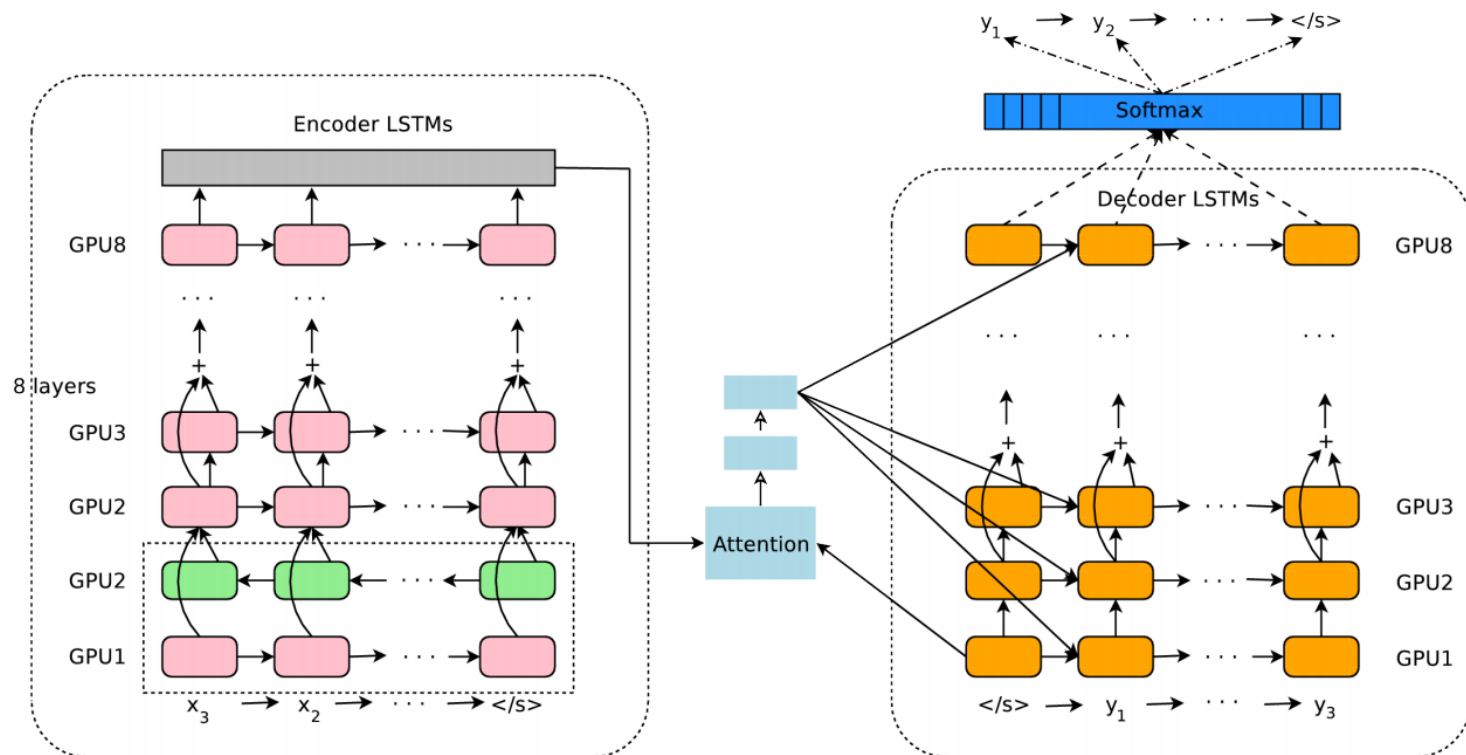


(b)

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	

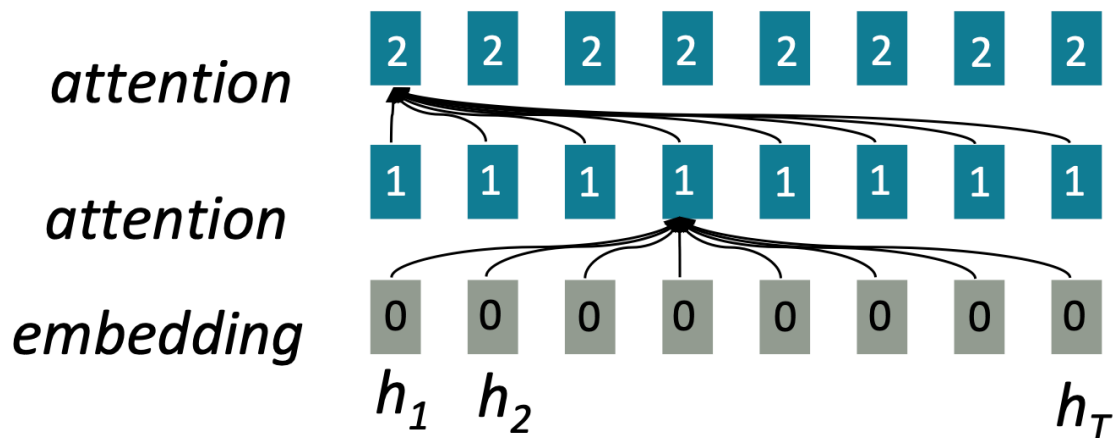
Table 10: Mean of side-by-side scores on production data

	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%

GNMT with different configurations

Limitations with Recurrent Models

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

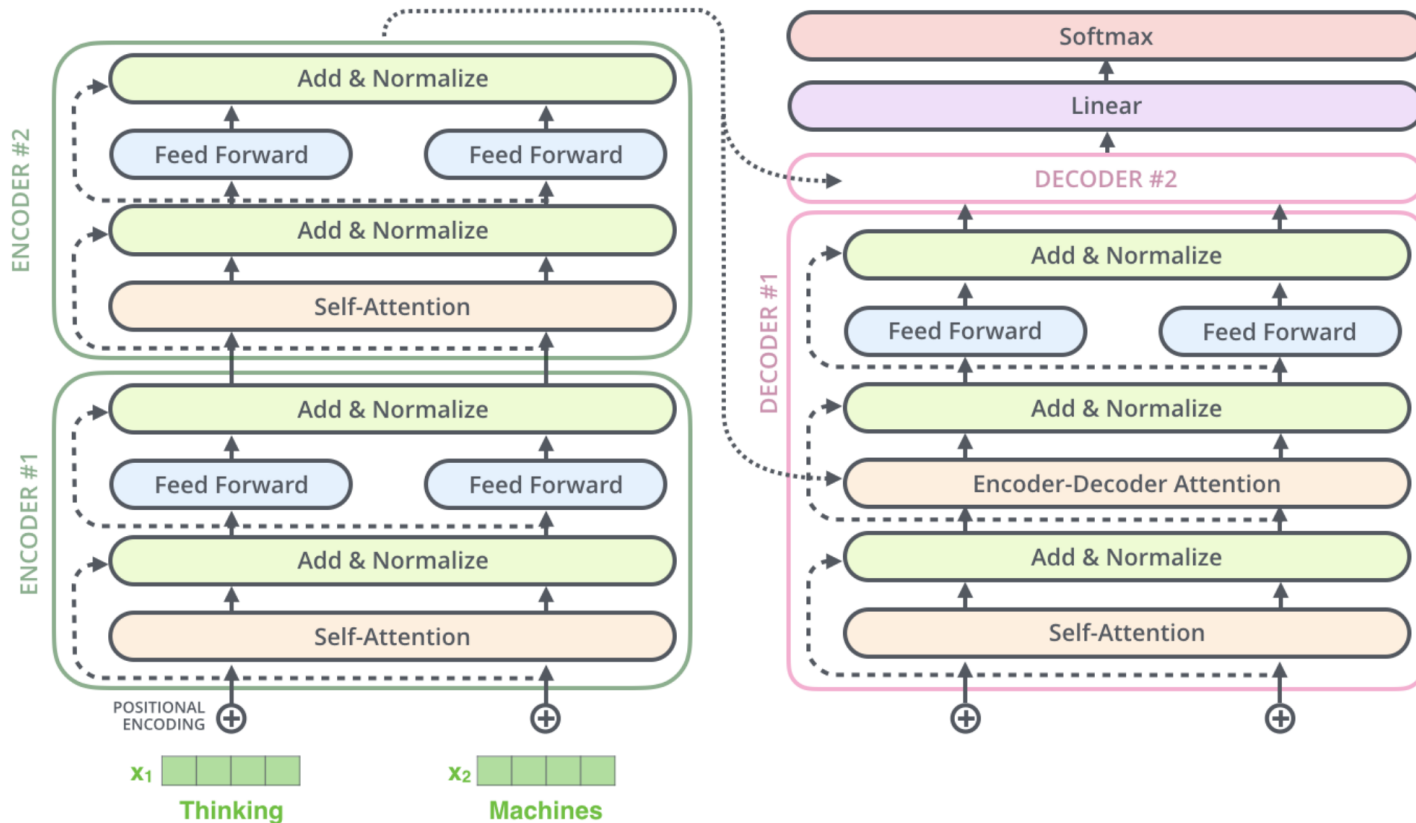
Part 3. Advanced Topics

- Techniques for improving efficiency
- Handling long inputs with Transformers

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

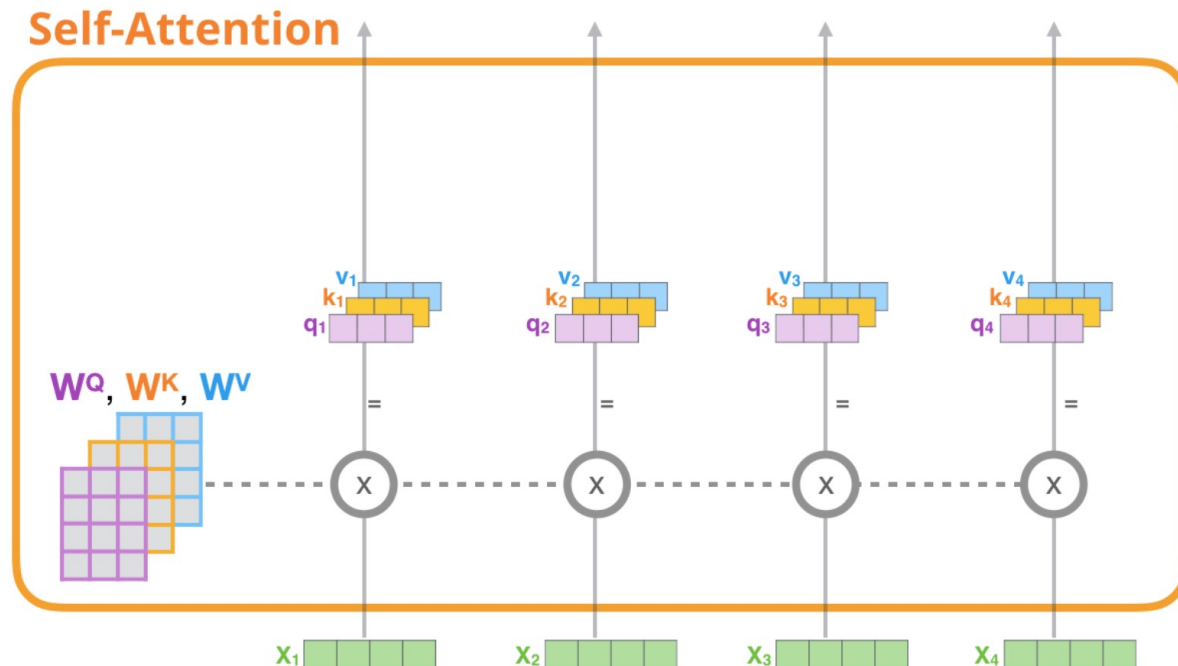


Transformer (Self-attention)

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

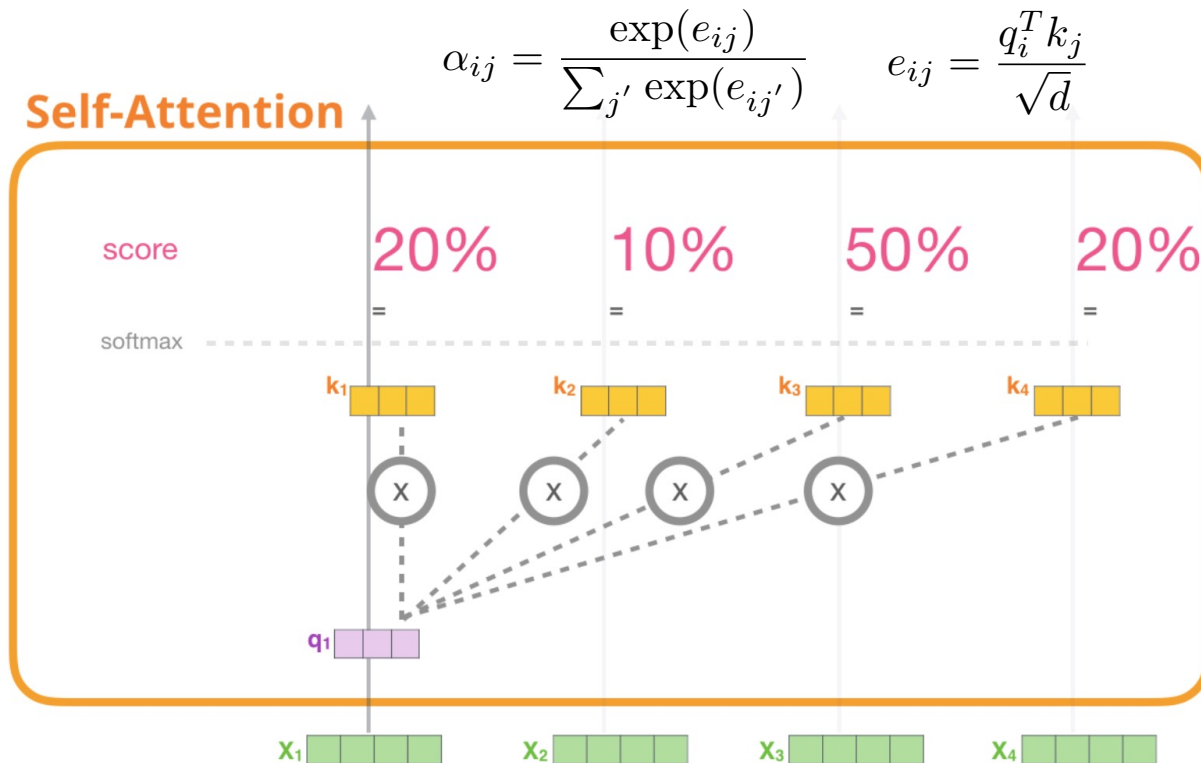
$$q_i = W^Q x_i, k_i = W^K x_i, v_i = W^V x_i$$



Transformer (Self-attention)

• Self-attention

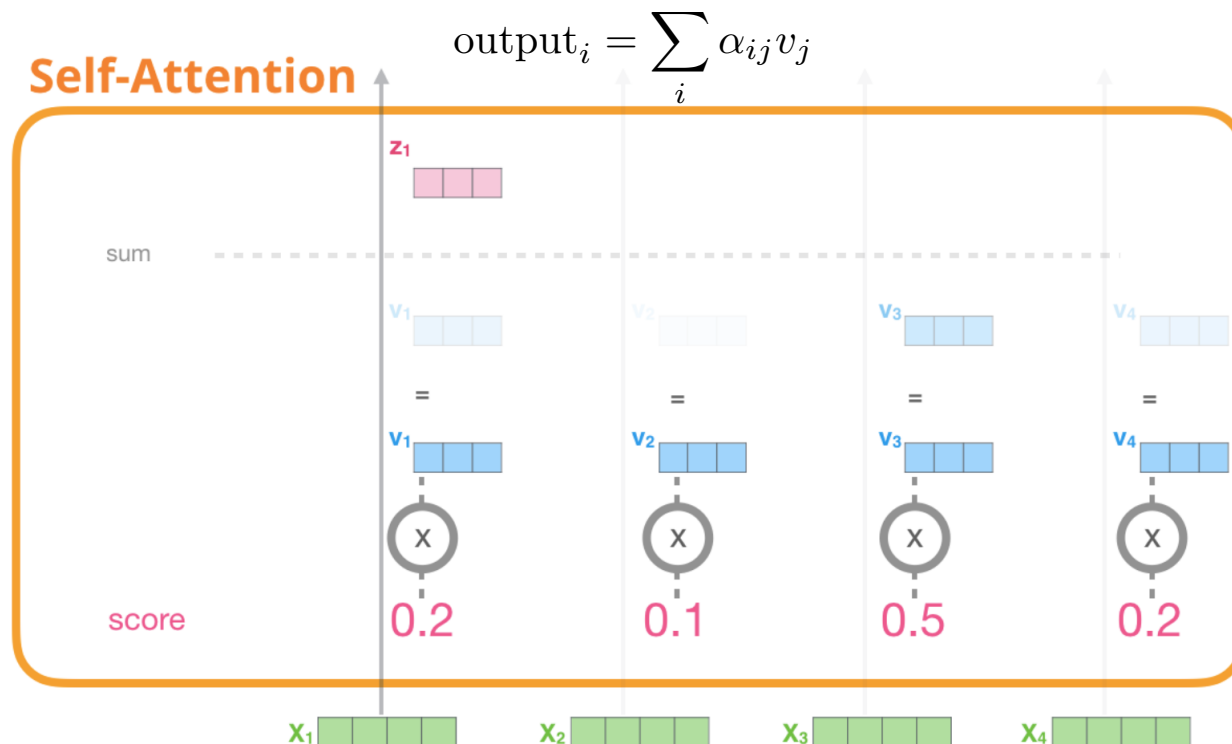
- **Recall:** Attention operates on **query**, **key**, and **value**
 - **Query** is decoder's hidden state, **key** and **value** are encoder's hidden states in seq2seq
- In self-attention, the query, key, and value are drawn from the **same source**
 1. For each input x_i , create query, key, and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



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**



Transformer (Self-attention)

- **Self-attention**

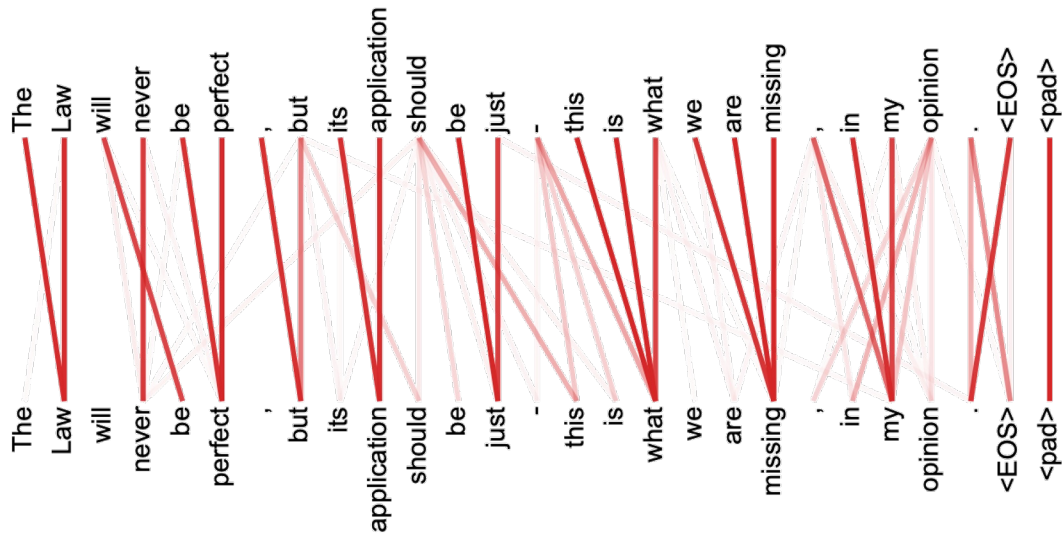
- **Recall:** Attention operates on **query**, **key**, and **value**
 - **Query** is decoder's hidden state, **key** and **value** are encoder's hidden states in seq2seq
- In self-attention, the query, key, and value are drawn from the **same source**
 1. For each input x_i , create query, key, and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

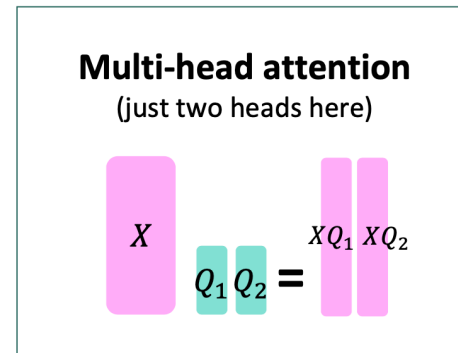
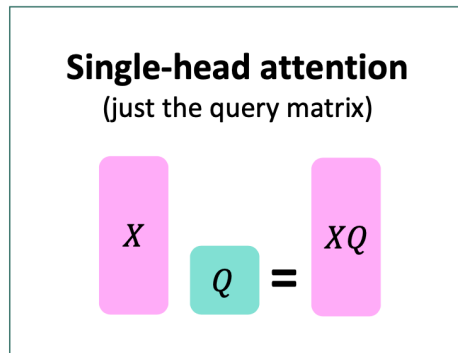
- **Recall:** Attention operates on **query**, **key**, and **value**
 - **Query** is decoder's hidden state, **key** and **value** are encoder's hidden states in seq2seq
- In self-attention, the query, key, and value are drawn from the **same source**
 1. For each input x_i , create query, key, and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



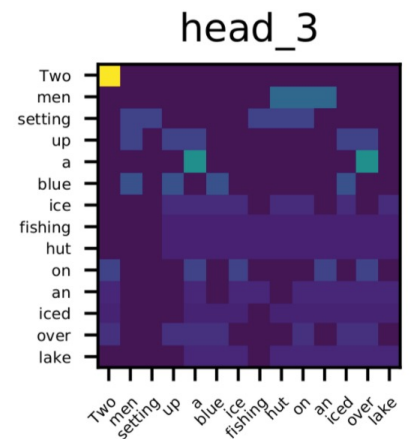
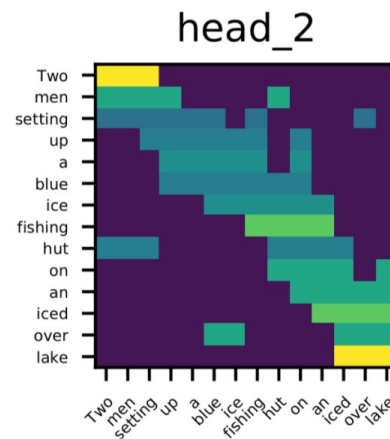
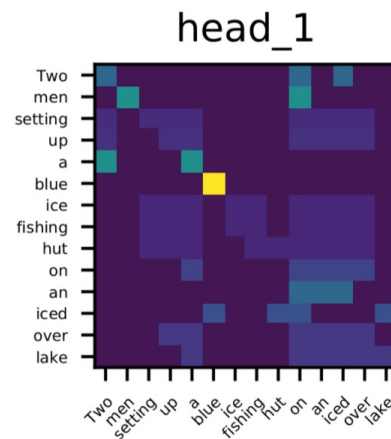
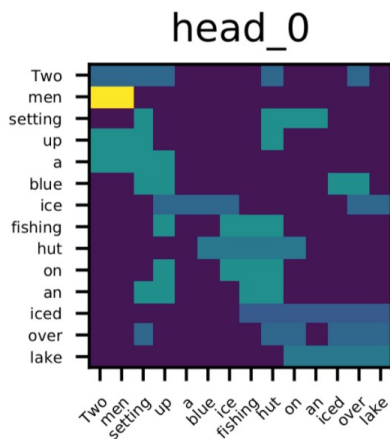
Transformer (Self-attention)

- **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**



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

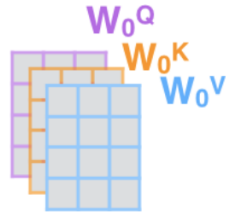
1) This is our input sentence*

Thinking
Machines

2) We embed each word*



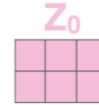
3) Split into 8 heads. We multiply X or R with weight matrices



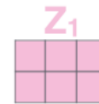
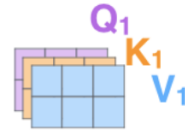
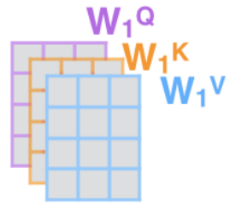
4) Calculate attention using the resulting $Q/K/V$ matrices



5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer



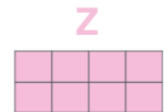
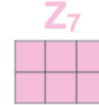
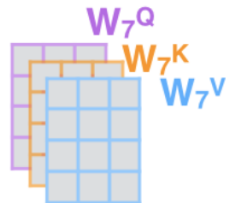
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



...

...

...



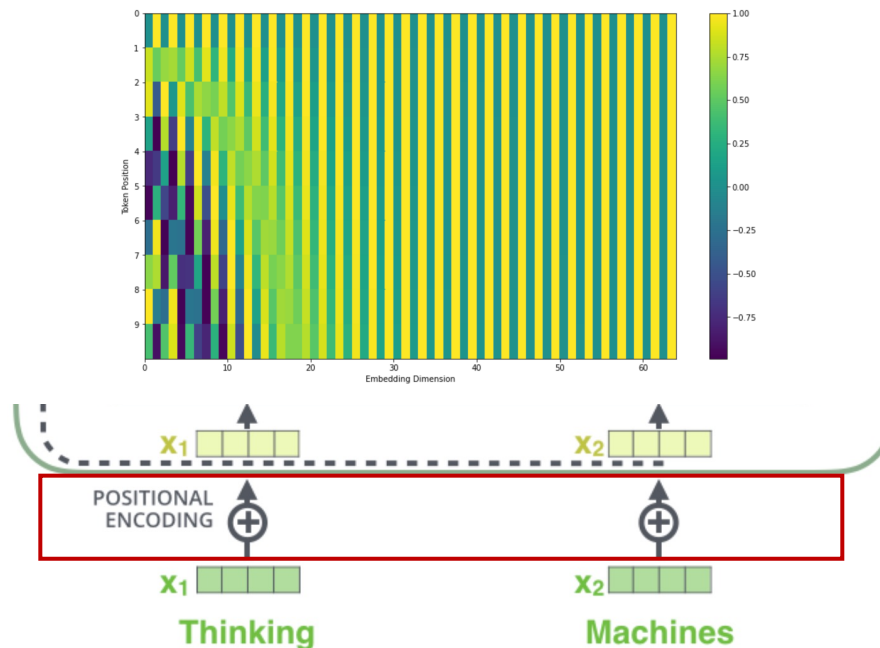
Transformer (Self-attention)

- **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

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

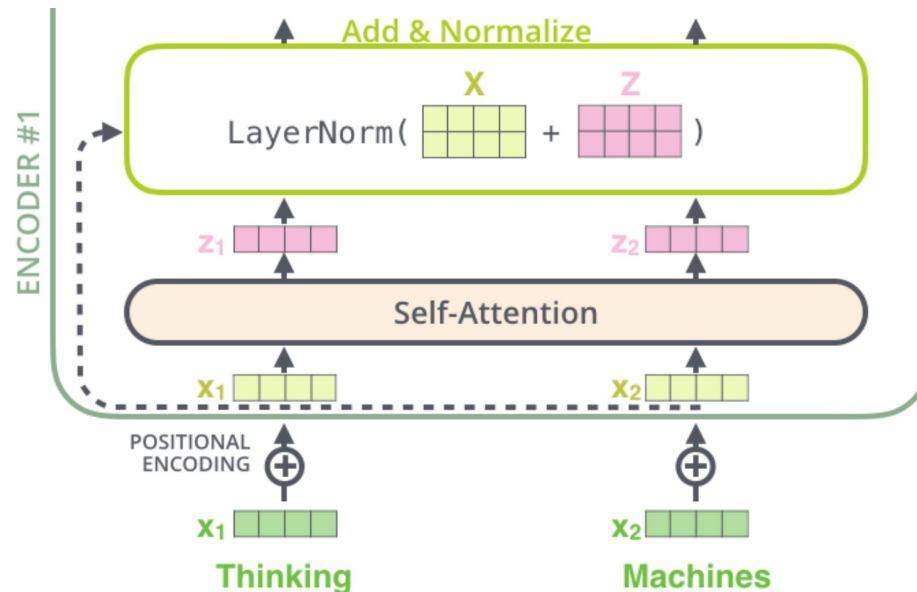
- PE_{pos+k} can be derived as a linear function of PE_{pos} → **easier to learn a relative position**
- Compare to learning encoding, it's better for **extrapolation** (not encountered in training)



Transformer (Self-attention)

- **Encoder**

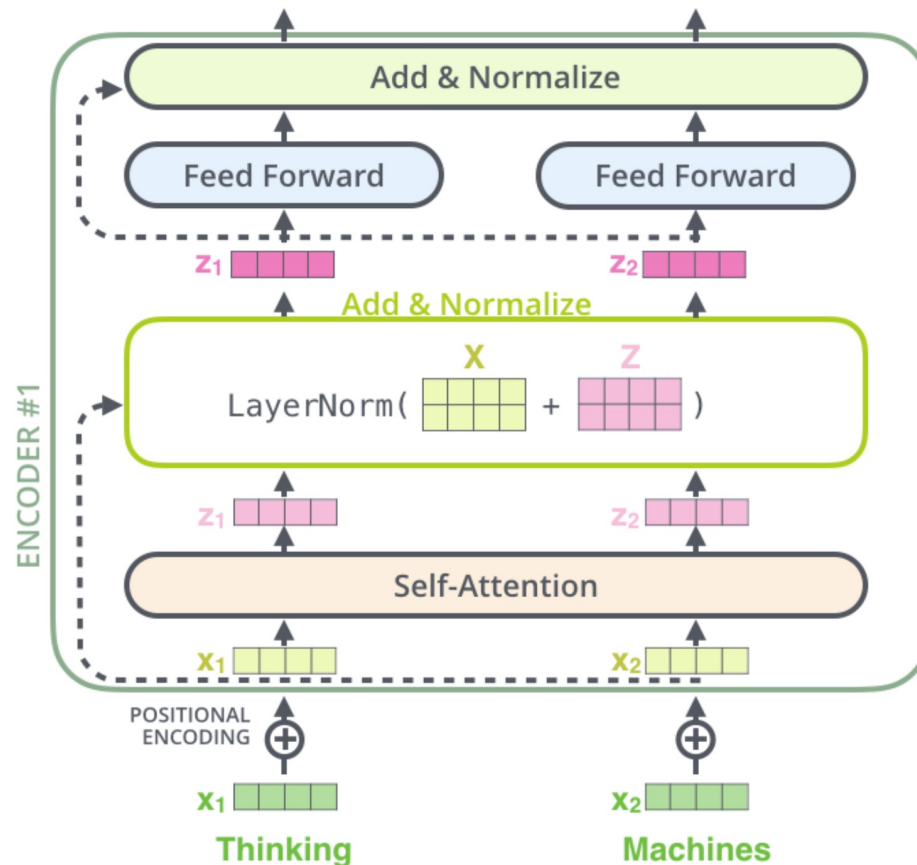
- Self-attention is **invariant to order** of input sequence → **positional encoding**
- **Residual connections** (dotted) and **layer normalization** are used to help training



Transformer (Self-attention)

- **Encoder**

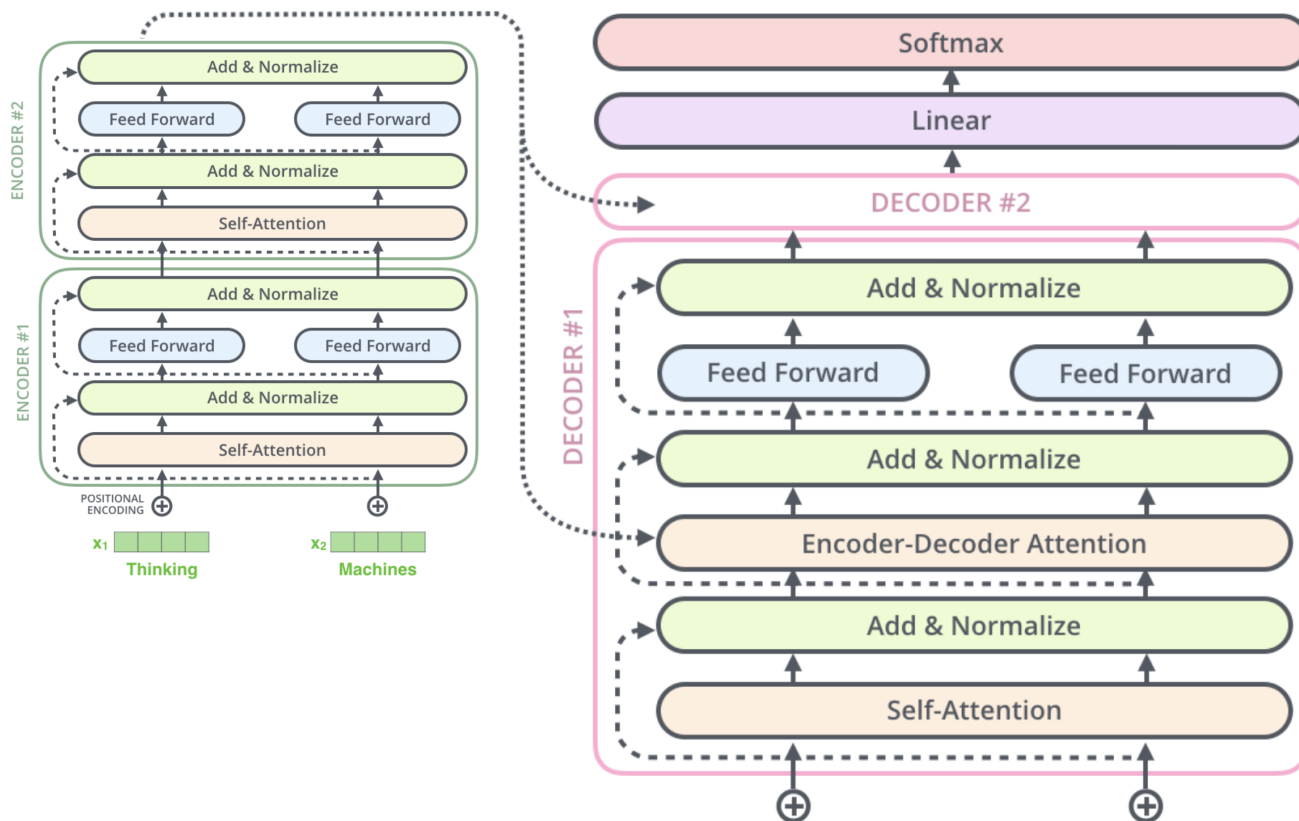
- Self-attention is **invariant to order** of input sequence → **positional encoding**
- **Residual connections** (dotted) and **layer normalization** are used to help training
- Non-linearity is imposed by adding position-wise **feed-forward networks**



Transformer (Self-attention)

- **Decoder**

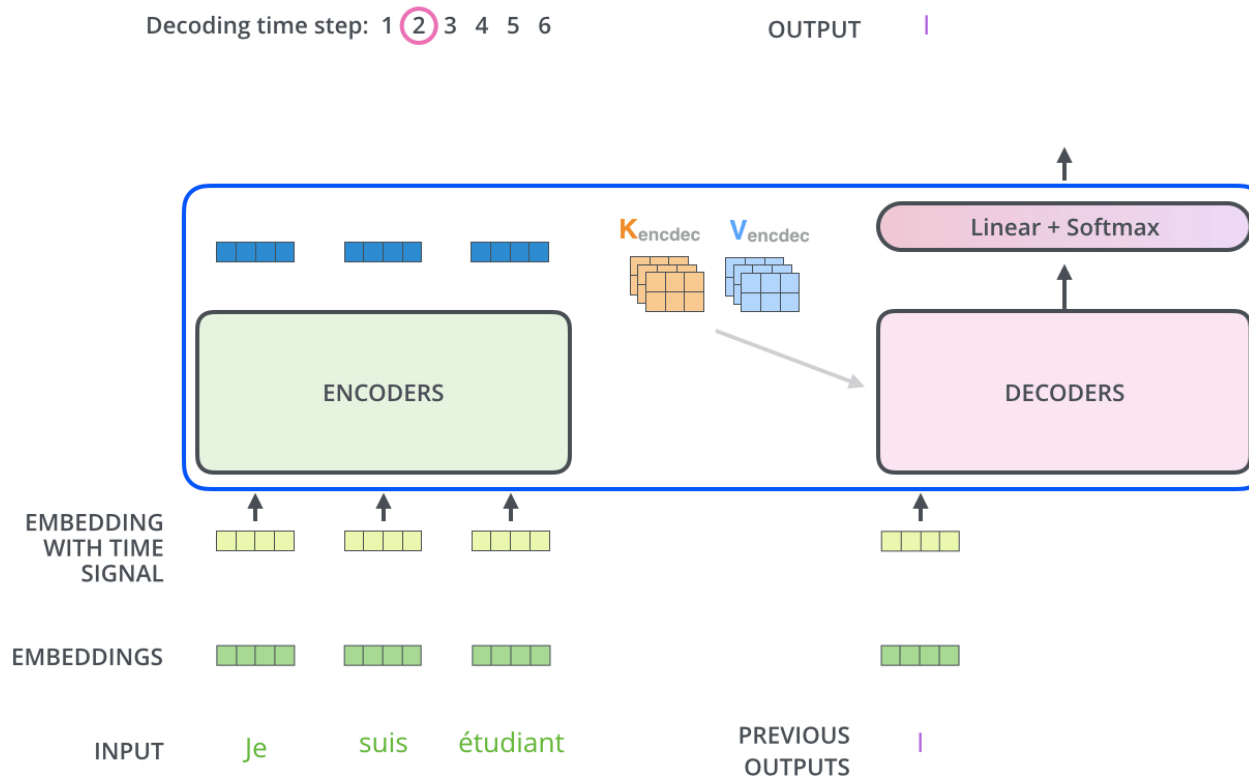
- Most parts are same with encoder except **encoder-decoder(cross) attention**
- This cross attention is previously used in seq2seq model
 - Queries are drawn from the **decoder**
 - Keys and values are drawn from the **encoder** (like context vector)



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**

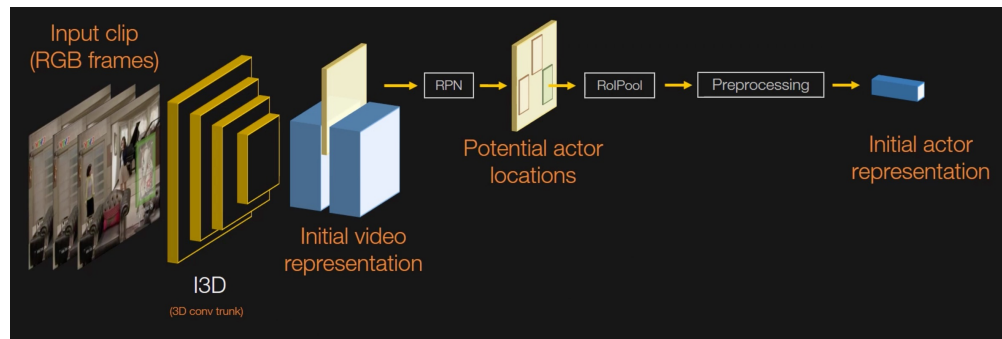
Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
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	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.0	$2.3 \cdot 10^{19}$	

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

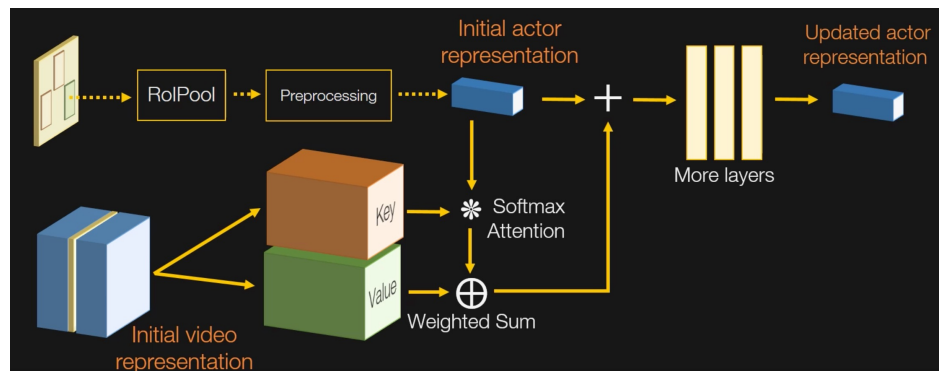
System	En→De	
	news2017	news2018
baseline	30.90	45.40
+ langid filtering	30.78	46.43
+ ffn 8192	31.15	46.28
+ BT	33.62	46.66
+ fine tuning	-	47.61
+ ensemble	-	49.27
+ reranking	-	50.63
WMT'18 submission	-	46.10
WMT'19 submission		42.7

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**

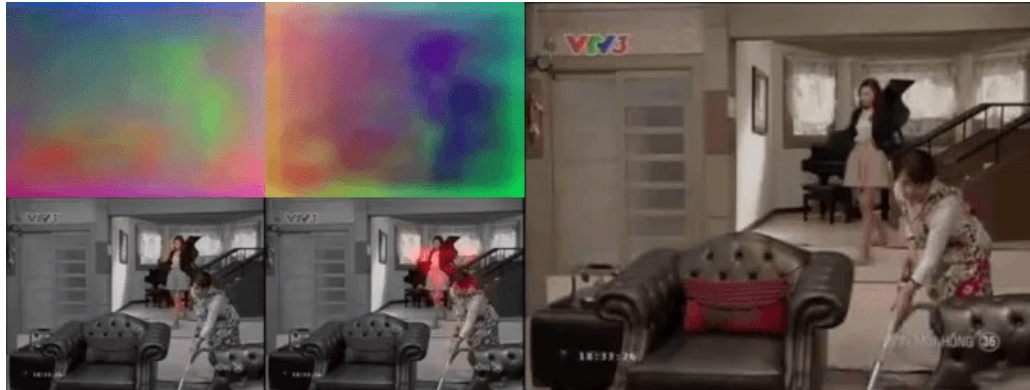


- **Self-attention is an effective way to resolve this issue**



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

- Techniques for improving efficiency
- Handling long inputs with Transformers

Part 4. Summary

- **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***

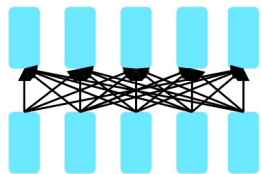
Pre-training and Fine-tuning Paradigm with Transformers

• 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

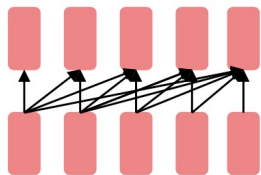
• Pre-training for two types of architectures

- 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)

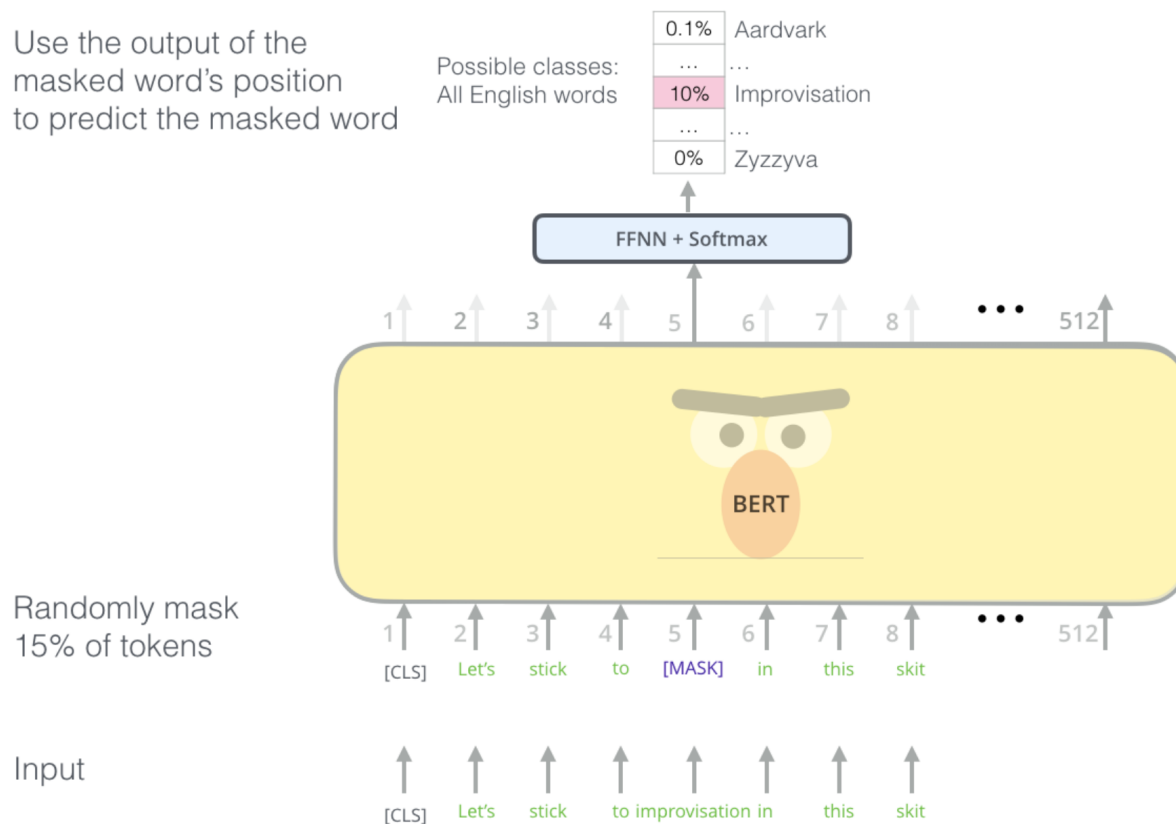


Decoders

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

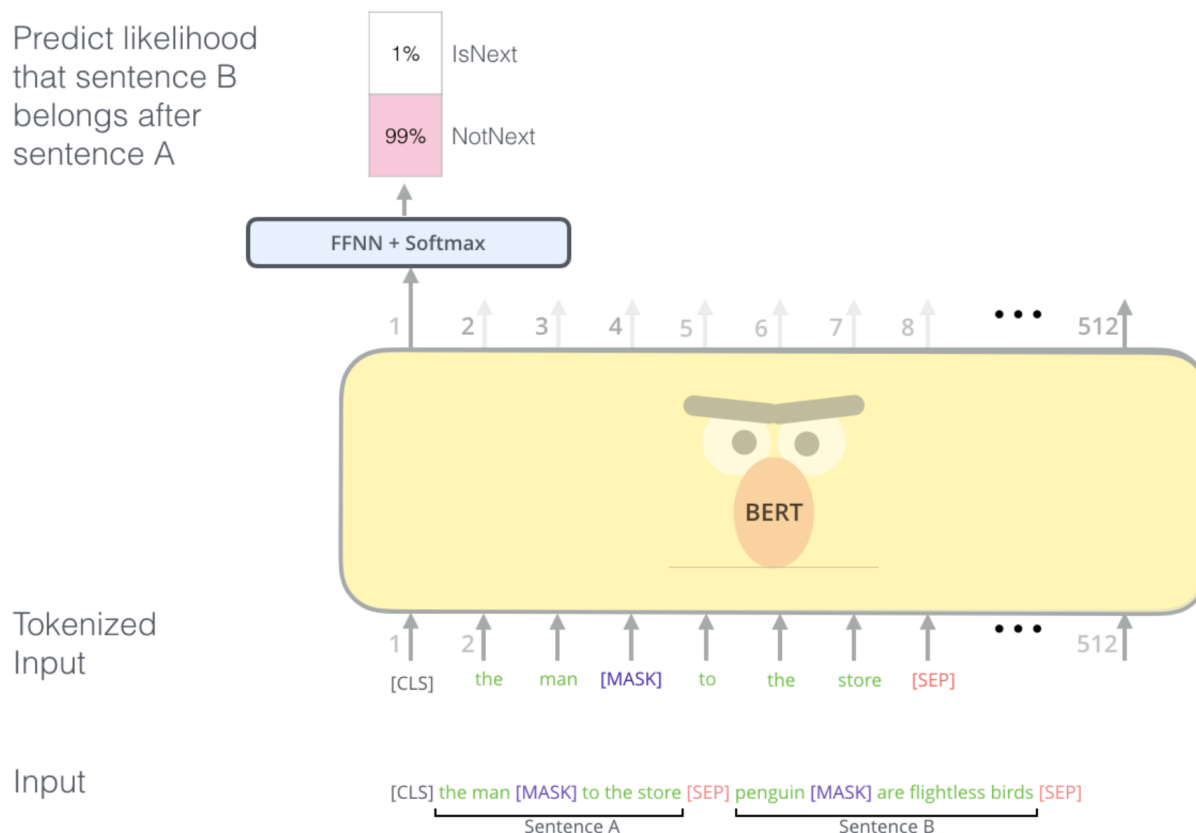
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

- **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



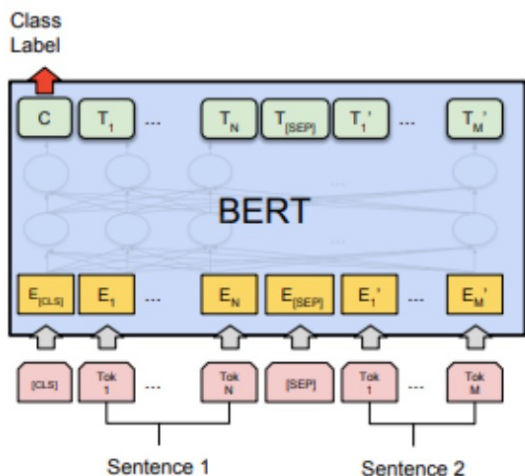
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

- **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**

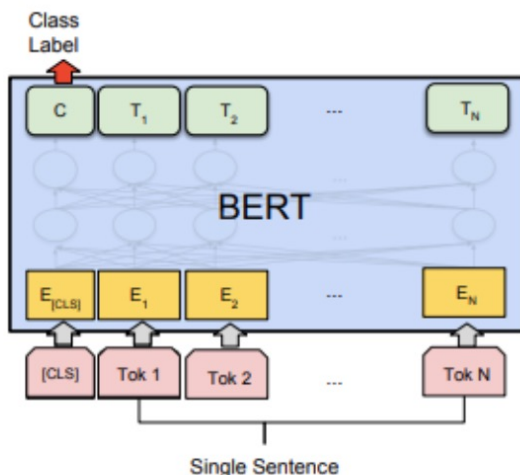


BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

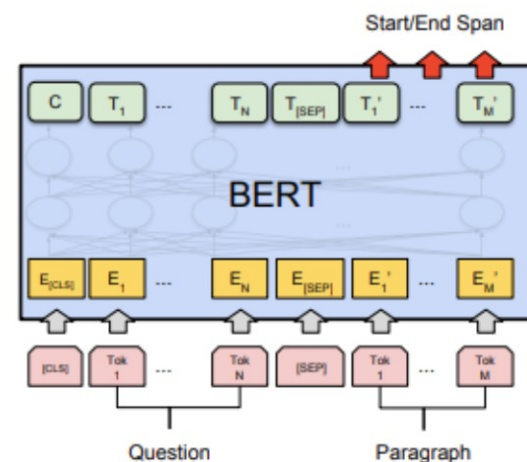
- **BERT**: Bidirectional Encoder Representations from Transformers [Devlin et al., 2018]
 - Even **without** task-specific complex architectures, BERT achieves **SOTA** for **11 NLP tasks**, including classification, question answering, tagging, etc.
 - By simply fine-tuning a whole network with **additional 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 _{BASE}	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{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 _{BASE}	96.4	92.4
BERT _{LARGE}	96.6	92.8

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
BERT _{BASE}	81.6	-
BERT _{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

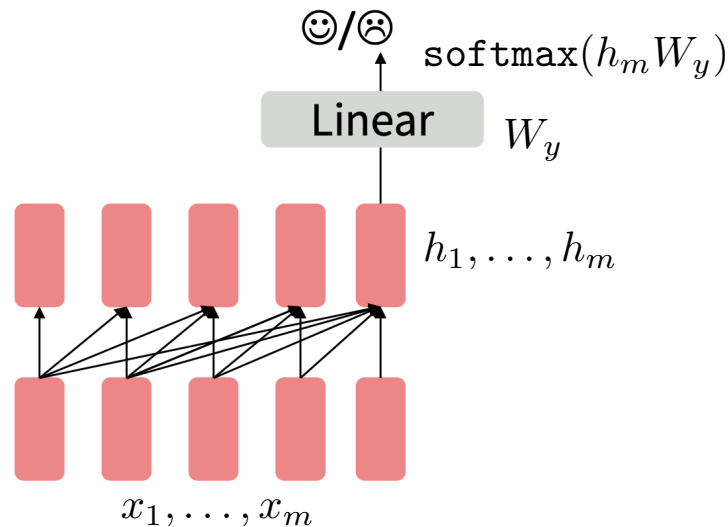
Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	94.6/89.4	90.2	96.4
BERT _{LARGE}			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

$$\arg \max_{\theta} \log p(\mathbf{x}) = \sum_n p_{\theta}(x_n | x_1, \dots, x_{n-1})$$

- **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]

$$\arg \max_{\theta} \log p(\mathbf{x}) = \sum_n p_{\theta}(x_n | x_1, \dots, x_{n-1})$$

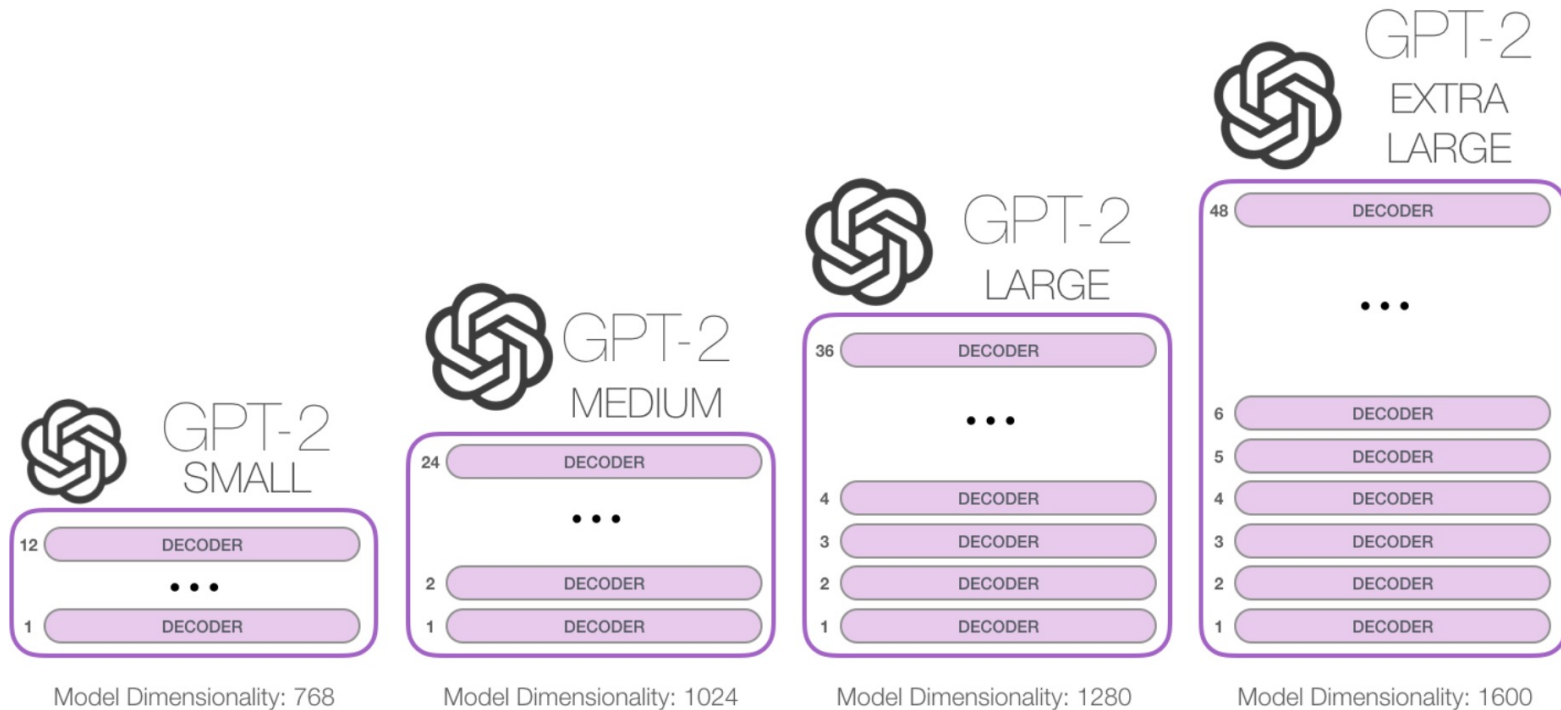
- **Pre-training** by language modeling over 7000 unique books (**unlabeled data**)
 - Contains long spans of contiguous text, for learning long-distance dependencies
- **Fine-tuning** by training a classifier with target task-specific **labeled data**
 - Classifier is added on the final transformer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

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	<u>89.3</u>	-	-	-
CAFE [58] (5x)	80.2	79.0	<u>89.3</u>	-	-	-
Stochastic Answer Network [35] (3x)	<u>80.6</u>	<u>80.1</u>	-	-	-	-
CAFE [58]	78.7	77.9	88.5	<u>83.3</u>		
GenSen [64]	71.4	71.3	-	-	<u>82.3</u>	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

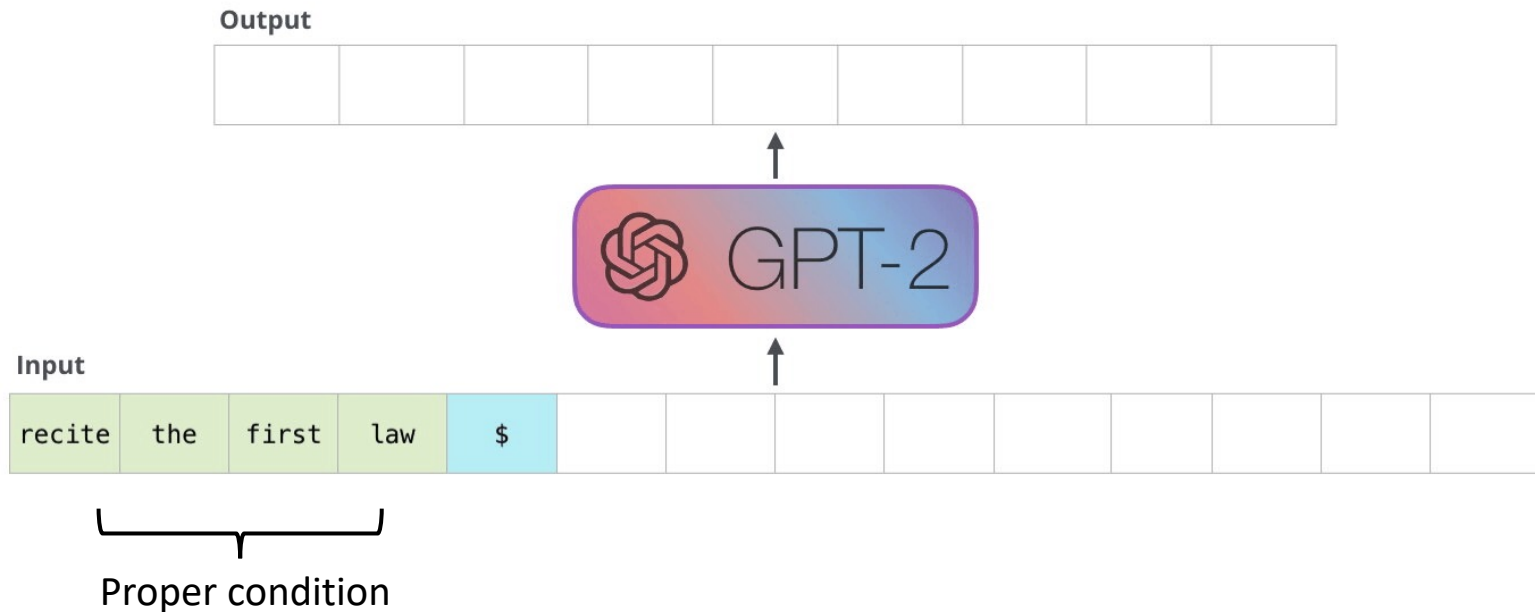
GPT-2: Language Models are Unsupervised Multitask Learners

- **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?

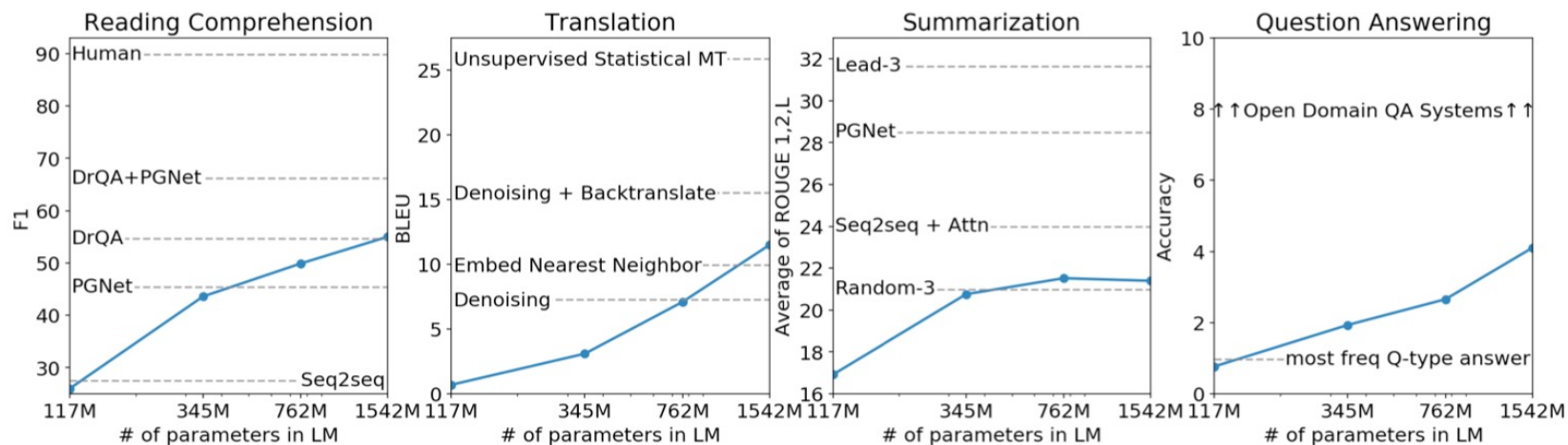


Figure 1. Zero-shot task performance of WebText LMs as a function of model size on many NLP tasks. Reading Comprehension results are on CoQA (Reddy et al., 2018), translation on WMT-14 Fr-En (Artetxe et al., 2017), summarization on CNN and Daily Mail (See et al., 2017), and Question Answering on Natural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

```

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

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

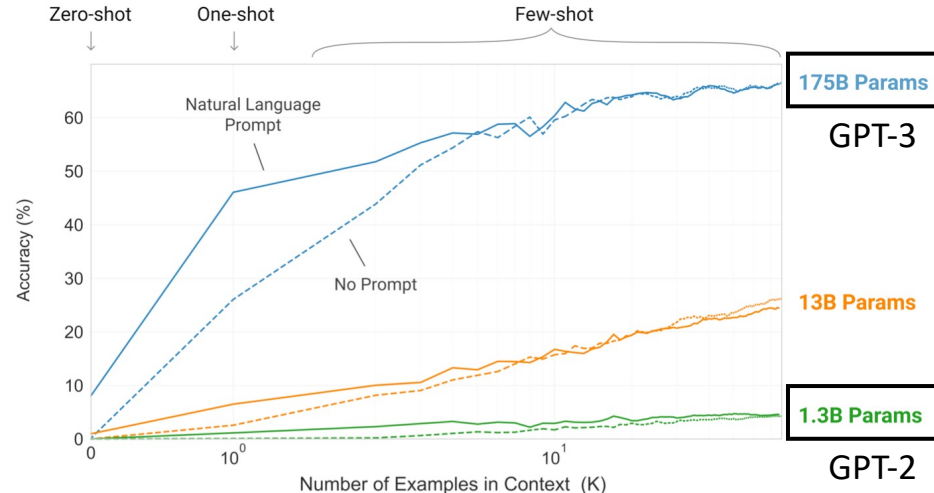
```

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

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

```

1 Translate English to French: ← task description
2 sea otter => loutre de mer ← examples
3 peppermint => menthe poivrée ←
4 plush girafe => girafe peluche ←
5 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!

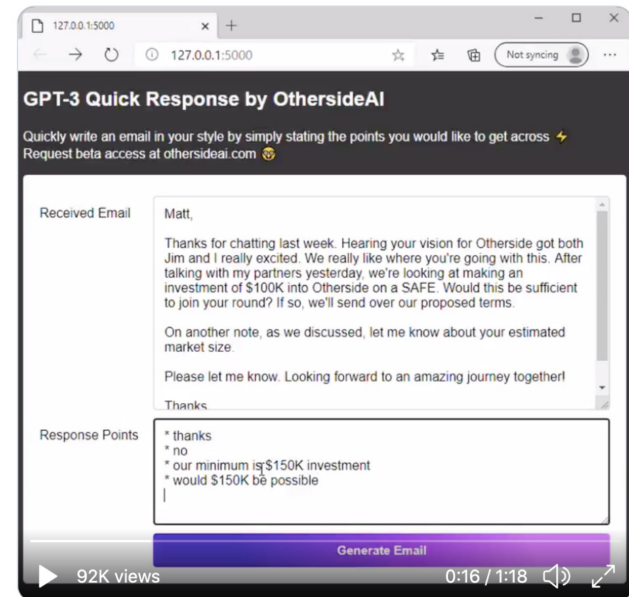
a button that looks like a watermelon

Generate

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



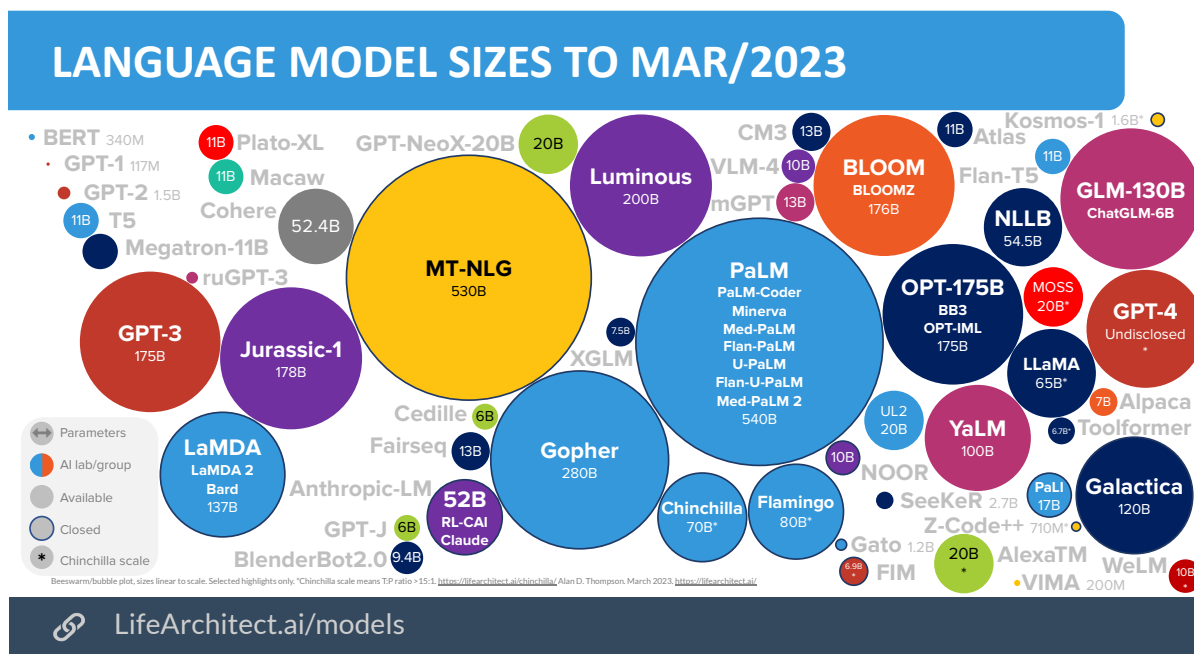
Simple code generation



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 **$O(\text{sequence length}^2)$** computation and memory
 - Poses challenges for both **training** and **deployment**



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 **$O(\text{sequence length}^2)$** 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
 - Most modern LLMs can handle **limited number of input tokens** ('context limit')



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

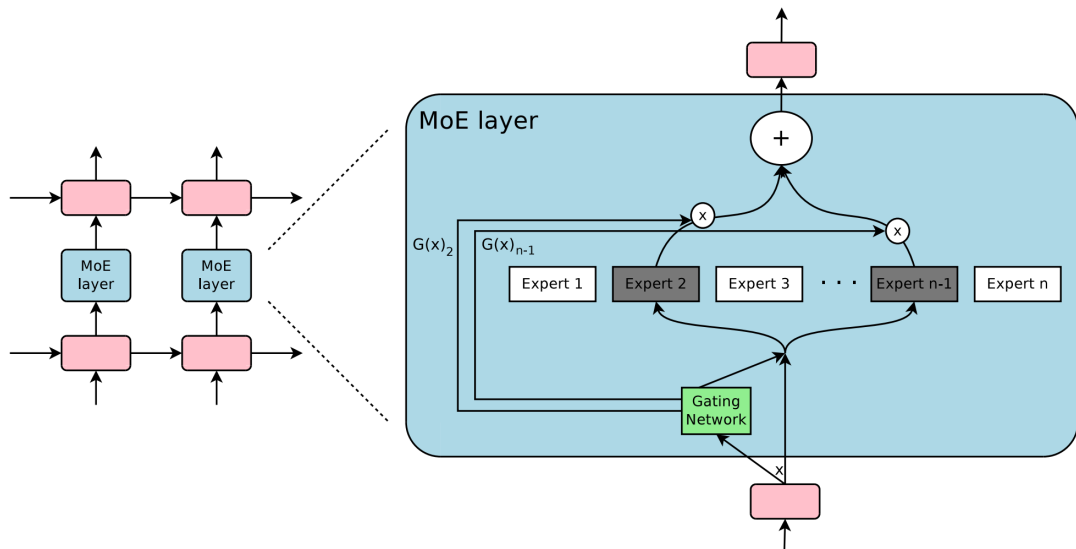
- Techniques for improving efficiency
- Handling long inputs with Transformers

Part 4. Summary

- **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 (Generalist Language Model)** [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



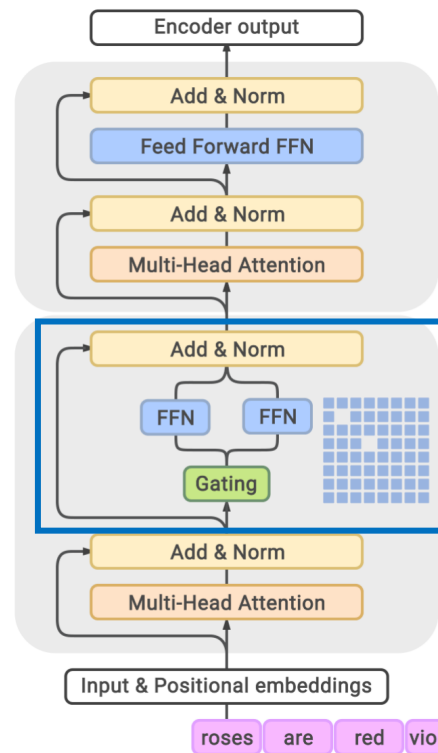
MoE layer [Shazeer et al., 2017] applied on LSTM

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

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

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

- **GLaM (Generalist Language Model)** [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 $O(\text{number of experts}^2)$ 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 (Generalist Language Model)** [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

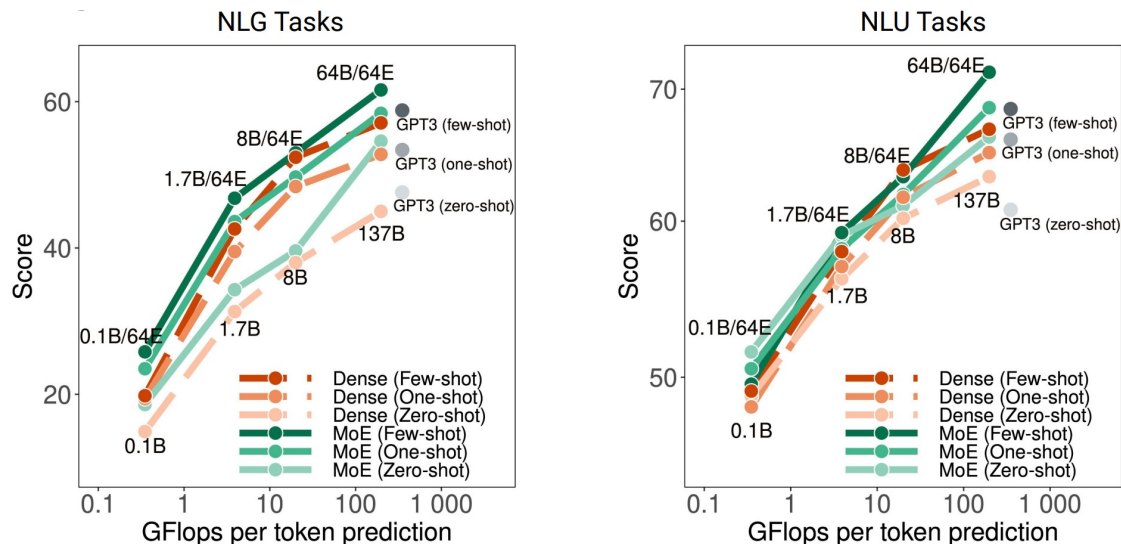
Number of total parameters and activated parameters for each GLaM model (base dense size / number of experts)

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

- **GLaM (Generalist Language Model)** [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%

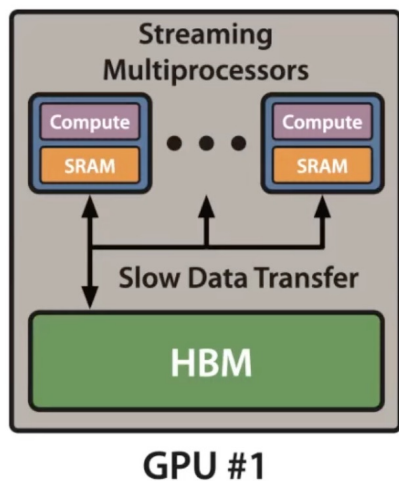
- Models trained with mixture-of-experts **successfully scale**
 - Achieves higher performance using similar FLOPs per token prediction



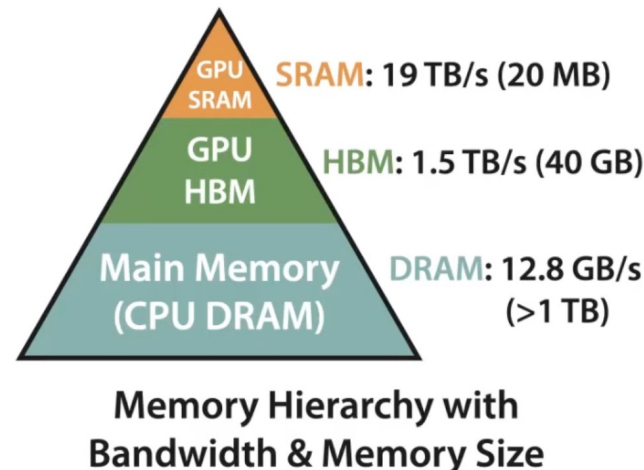
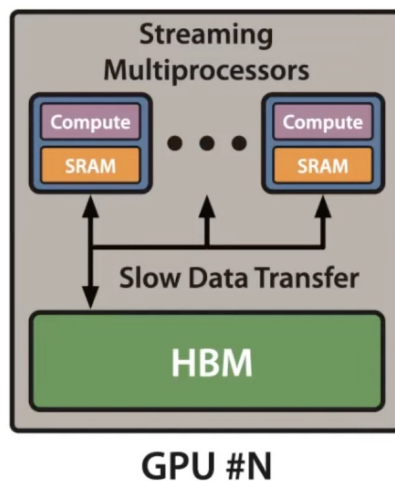
FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

- **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)**

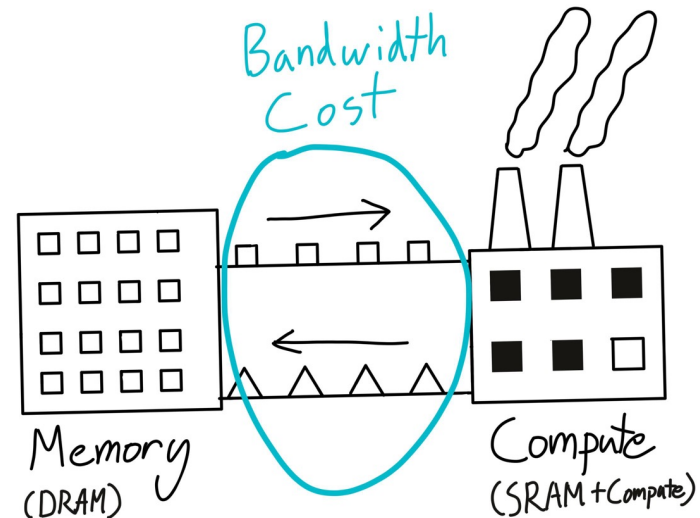
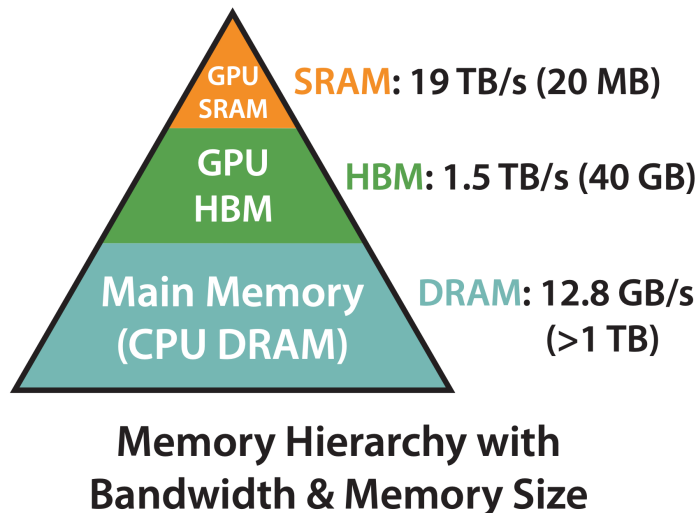


...



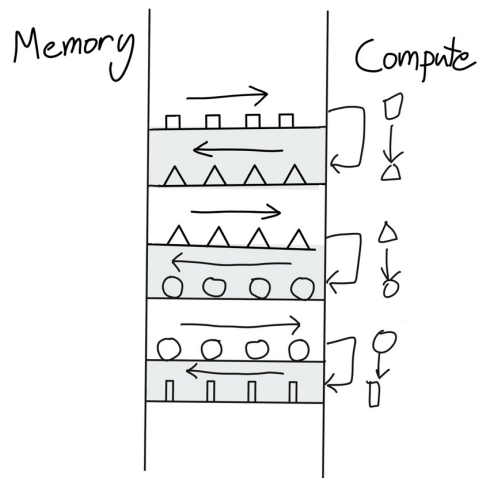
FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

- **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

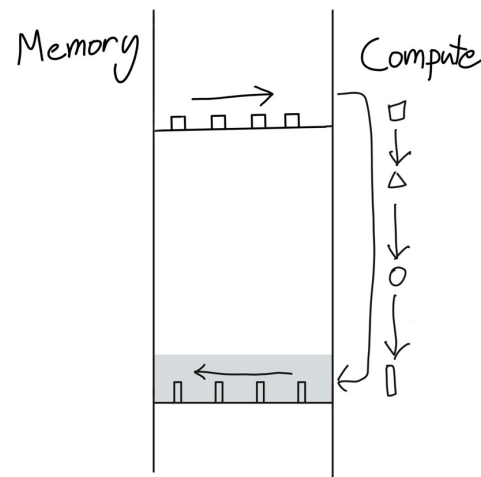


FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

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



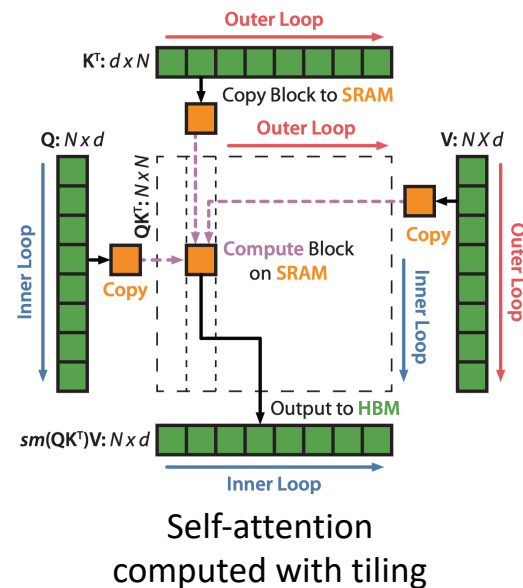
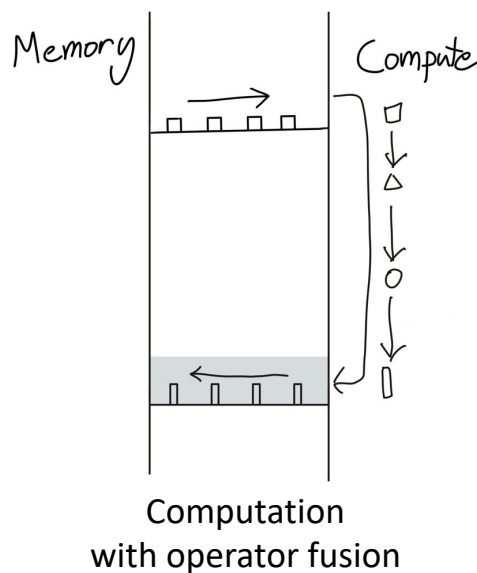
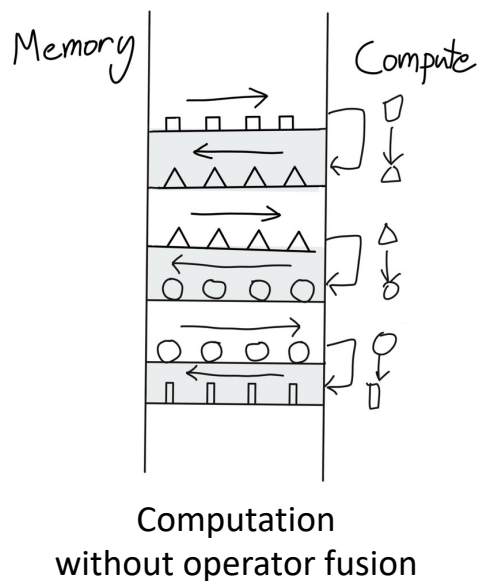
Computation
without operator fusion



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



- **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

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

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

- **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

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

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

- **Challenge 2:** Backward pass requires intermediate values

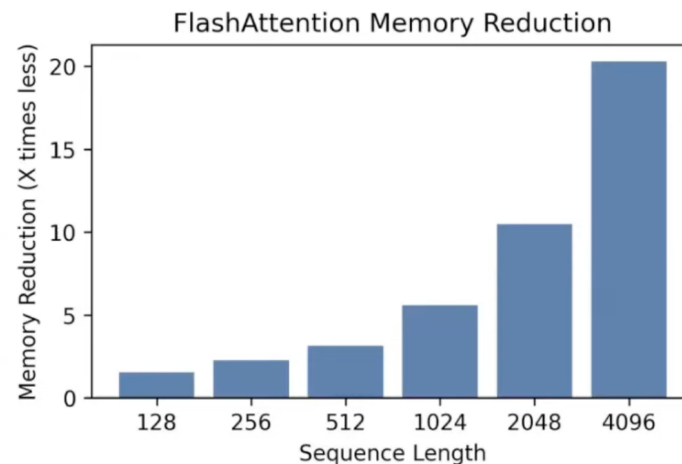
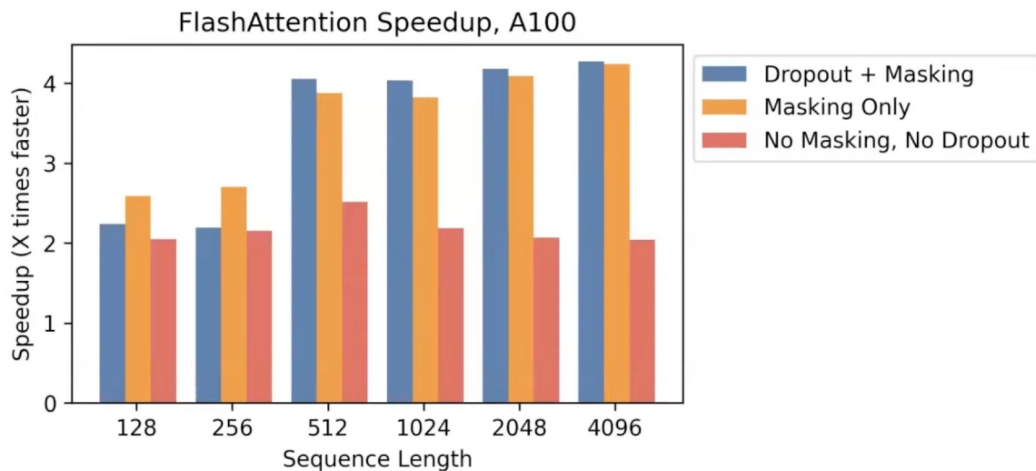
- The attention matrix have to be saved, requiring **extensive HBM access**
- Solution: **Recompute the attention matrix** during the backward pass
- This approach is **faster despite requiring more FLOPs**, thanks to **reduced HBM access**

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

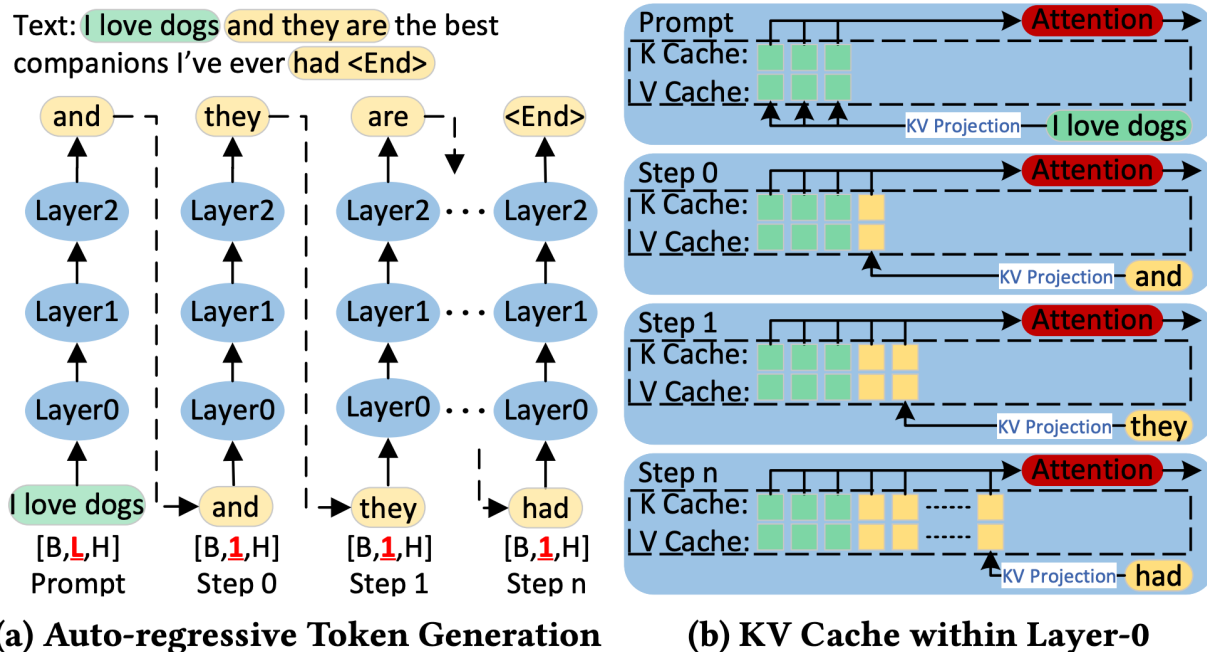
BERT Implementation	Training time (minutes)
Huggingface [91]	55.6 ± 3.9
Nvidia MLPerf 1.1 [63]	20.0 ± 1.5
FLASHATTENTION (ours)	17.4 ± 1.4

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×)

PagedAttention: Efficient Memory Management for LLM Serving

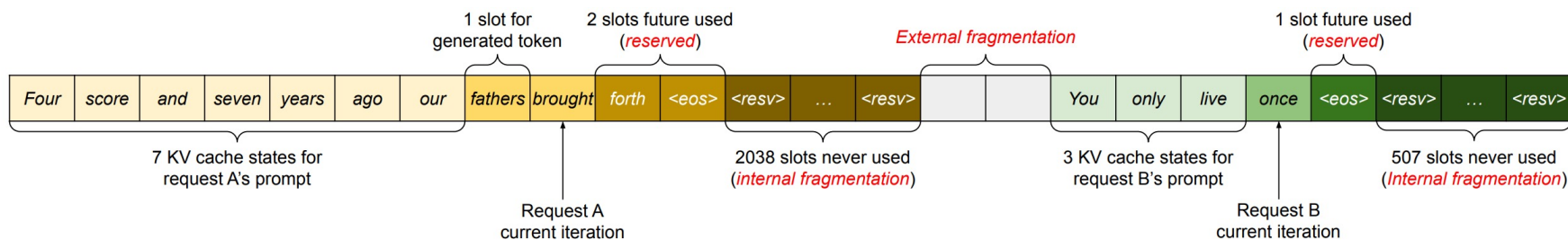
- **PagedAttention (vLLM)** [Kwon et al., 2023]

- **Motivation:** **Inefficient cache management** in existing LLM deployment systems
- **KV caching**
 - Auto-regressive language models generate 1 token at each forward step
 - Naïve forwarding **recomputes the hidden states** for previous tokens
 - Identical computation is done multiple times, **wasting compute**
 - Practical implementations **cache the previous (key, value) pairs** to reduce computation



PagedAttention: Efficient Memory Management for LLM Serving

- **PagedAttention (vLLM)** [Kwon et al., 2023]
 - 3 sources of **memory waste** in typical KV cache management systems
 - **Reserved memory**
 - Pre-allocation of memory for future generation
 - Memory is eventually used, but occupied **even when it is not in use**
 - E.g. Early generation steps, where only a few KV cache is required
 - Problematic for LLM serving systems, which handle multiple requests simultaneously



PagedAttention: Efficient Memory Management for LLM Serving

- **PagedAttention (vLLM)** [Kwon et al., 2023]

- 3 sources of **memory waste** in typical KV cache management systems

- **Reserved memory**

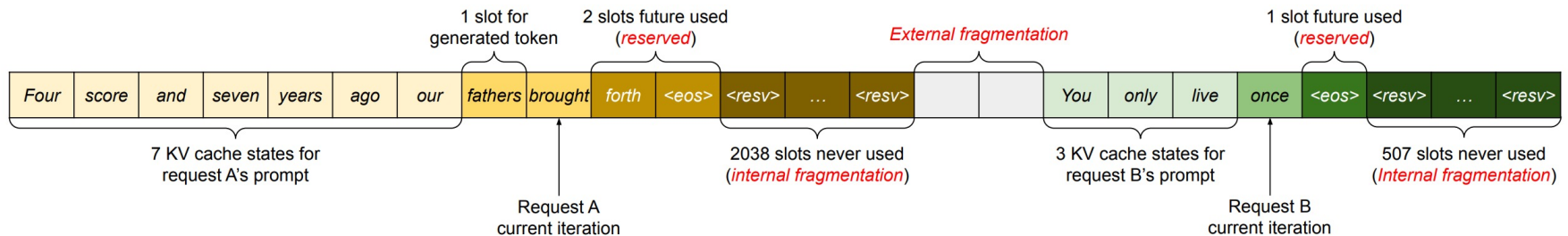
- Pre-allocation of memory for future generation
- Memory is eventually used, but occupied **even when it is not in use**
 - E.g. Early generation steps, where only a few KV cache is required
 - Problematic for LLM serving systems, which handle multiple requests simultaneously

- **Internal fragmentation**

- Caused by **over-estimating** potential maximum sequence lengths
- Memory is never used
- Amount of internal fragmentation is **unknown until generation is complete**

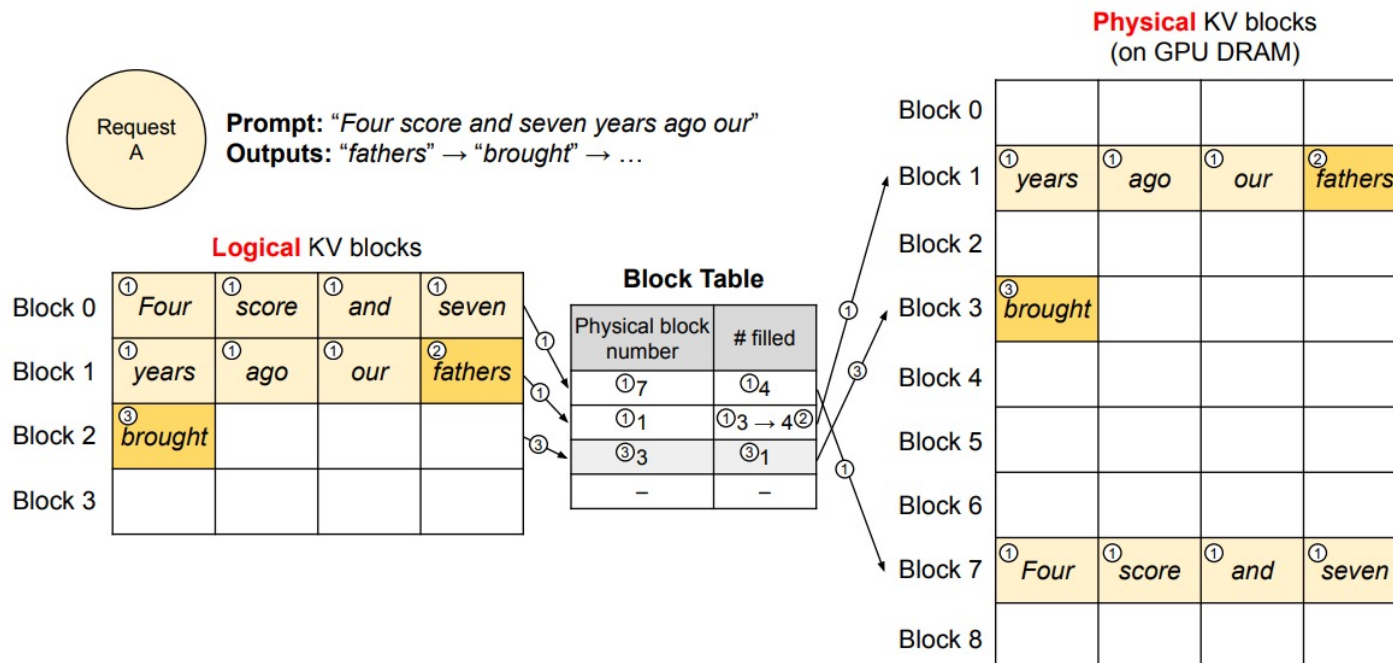
- **External fragmentation**

- Memory waste caused by the memory allocator



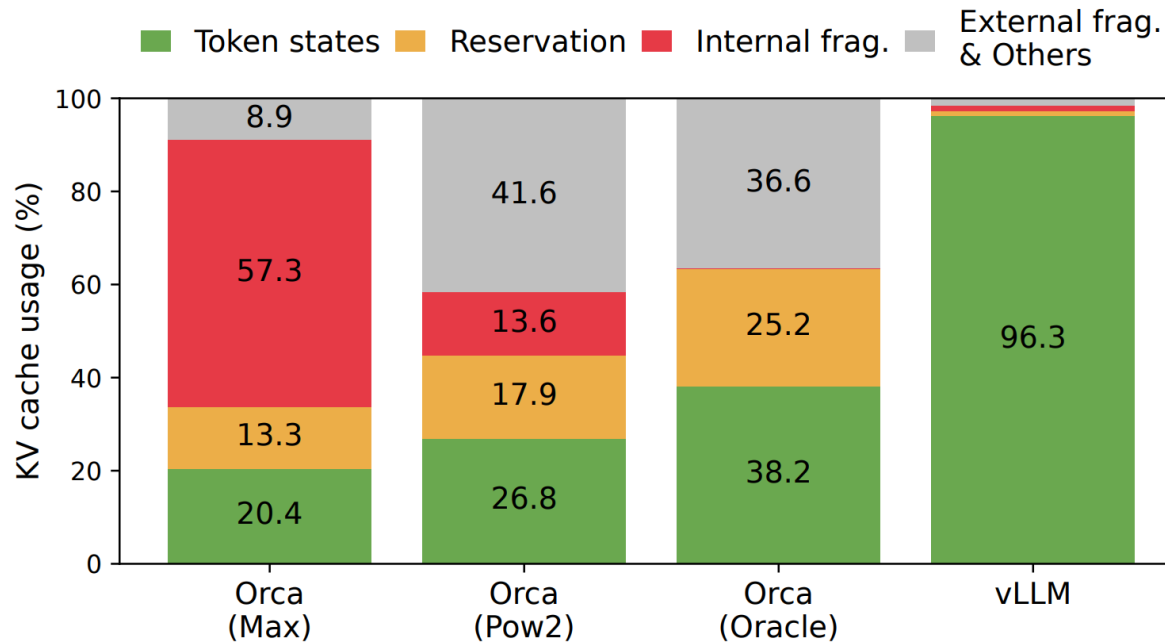
PagedAttention: Efficient Memory Management for LLM Serving

- **PagedAttention (vLLM)** [Kwon et al., 2023]
 - **Idea:** Manage KV cache in **non-contiguous memory**
 - Inspired by **virtual memory** and **paging** in OS
 - Manage KV cache in fixed-length blocks (in the figure, block size is 4)
 - **Logically**, blocks are allocated in **contiguous memory**
 - **Physically**, blocks are allocated in **non-contiguous memory**
 - Translation from logical index to physical index is performed using a **block table**



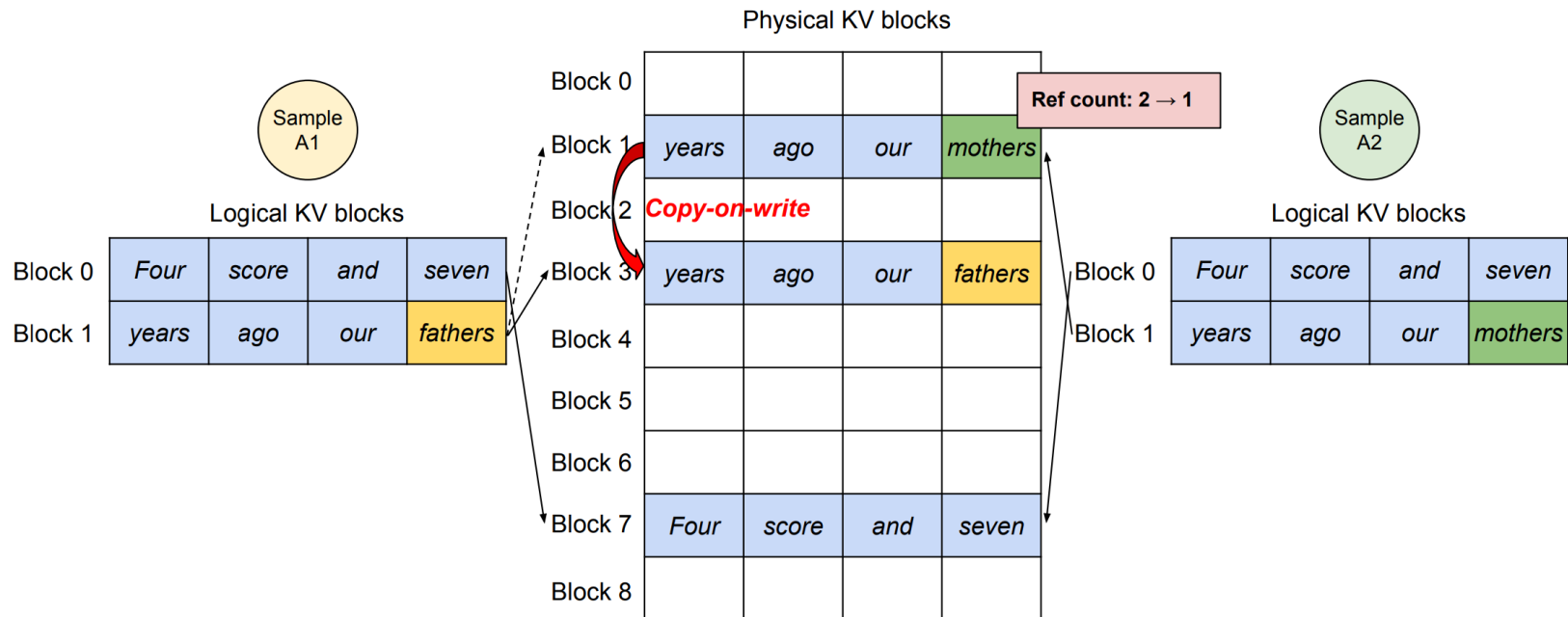
PagedAttention: Efficient Memory Management for LLM Serving

- **PagedAttention (vLLM)** [Kwon et al., 2023]
 - **Idea:** Manage KV cache in **non-contiguous memory**
 - The proposed design has several benefits
 - (1) **Near-zero memory waste** in KV cache memory



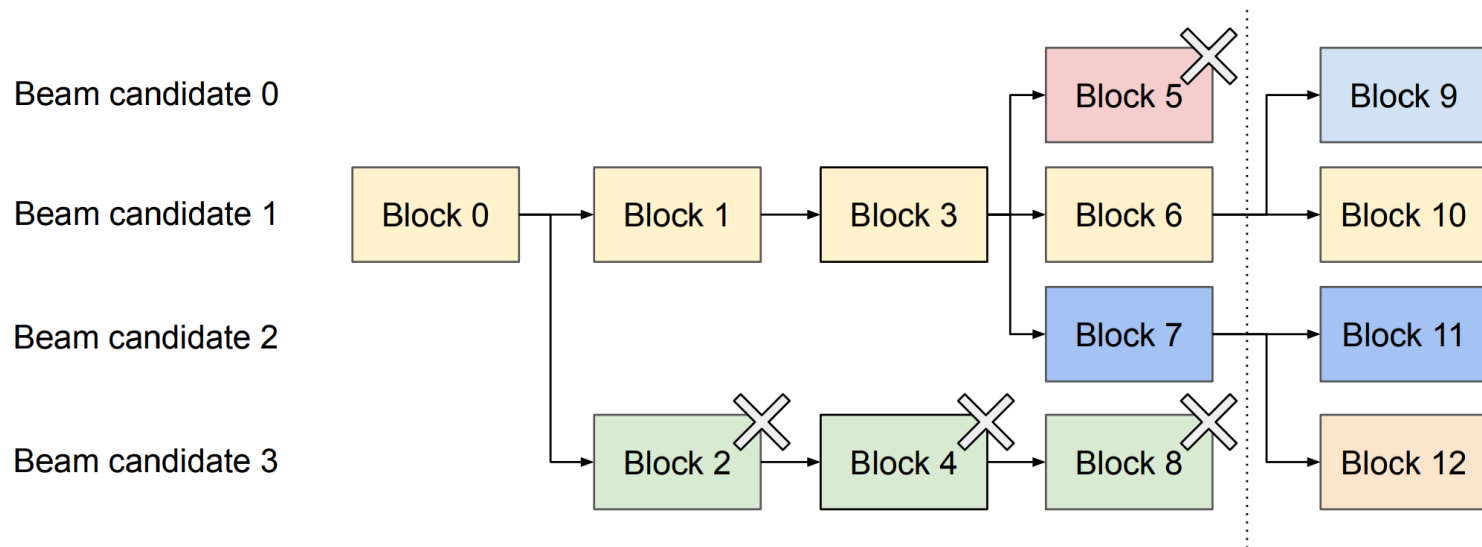
PagedAttention: Efficient Memory Management for LLM Serving

- **PagedAttention (vLLM)** [Kwon et al., 2023]
 - **Idea:** Manage KV cache in **non-contiguous memory**
 - The proposed design has several benefits
 - (1) **Near-zero memory waste** in KV cache memory
 - (2) **Flexible sharing** of KV cache enables further memory saving
 - **Parallel sampling** (multiple outputs are generated from a single input)



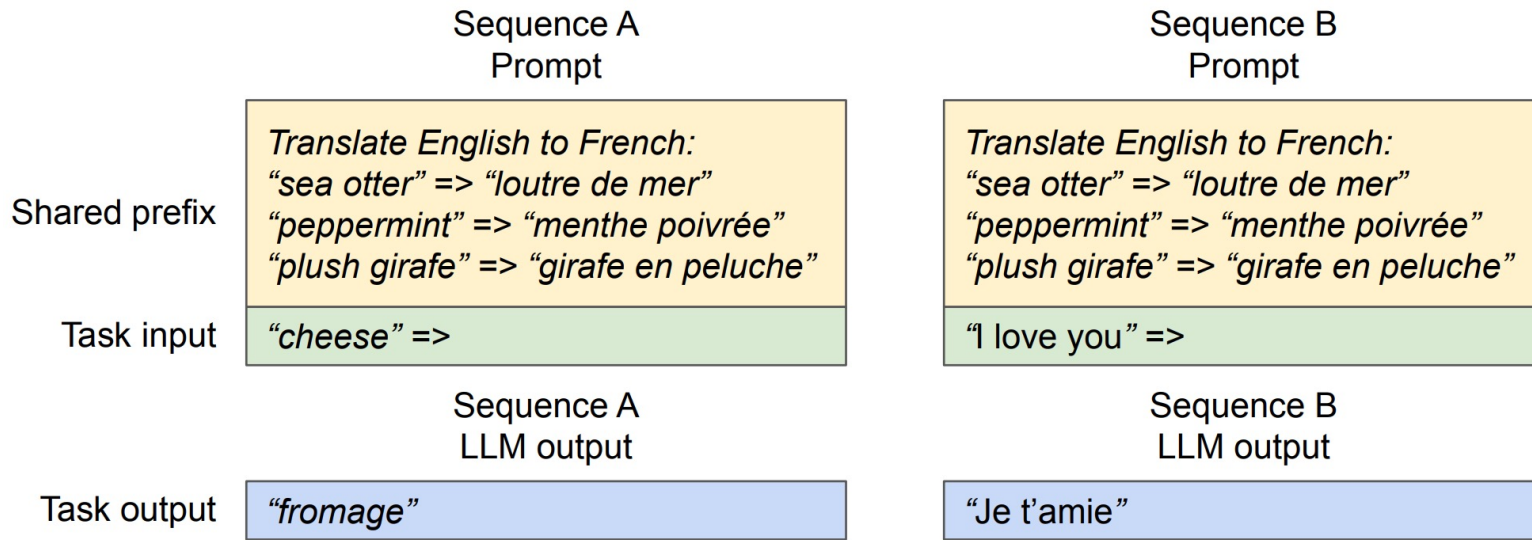
PagedAttention: Efficient Memory Management for LLM Serving

- **PagedAttention (vLLM)** [Kwon et al., 2023]
 - **Idea:** Manage KV cache in **non-contiguous memory**
 - The proposed design has several benefits
 - (1) **Near-zero memory waste** in KV cache memory
 - (2) **Flexible sharing** of KV cache enables further memory saving
 - **Parallel sampling** (multiple outputs are generated from a single input)
 - **Beam search**



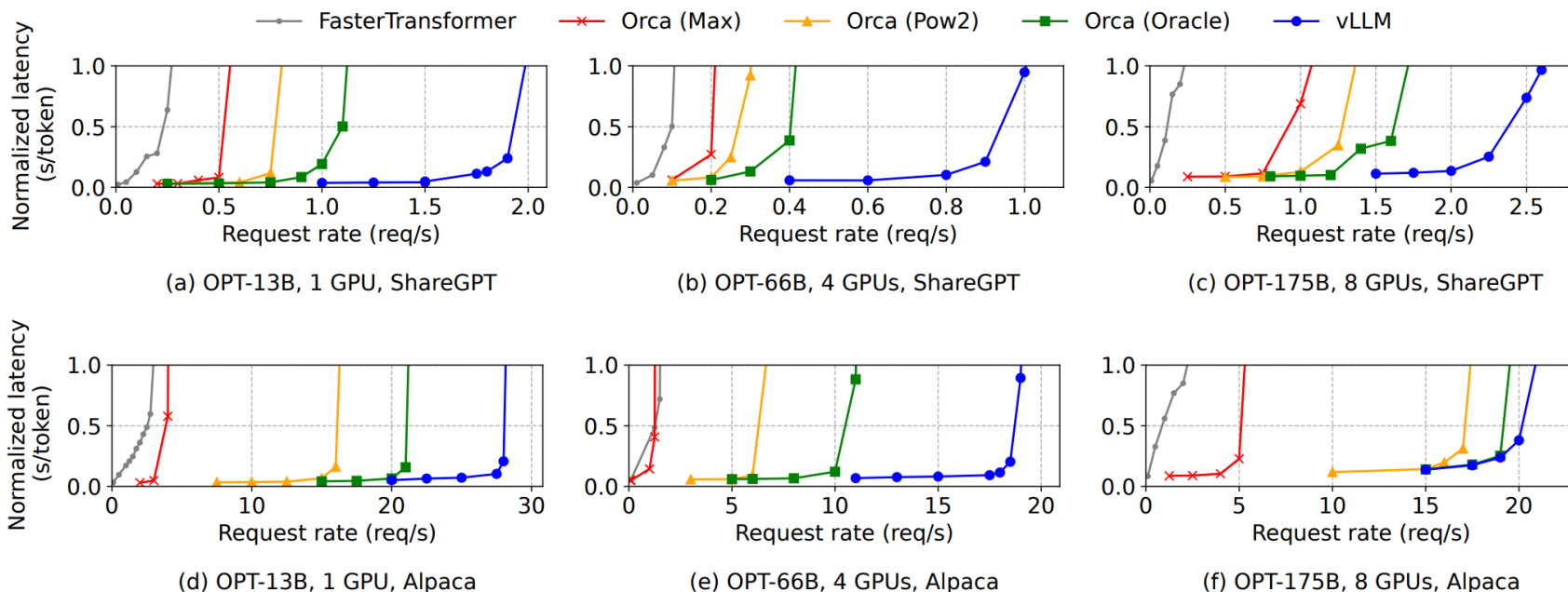
PagedAttention: Efficient Memory Management for LLM Serving

- **PagedAttention (vLLM)** [Kwon et al., 2023]
 - **Idea:** Manage KV cache in **non-contiguous memory**
 - The proposed design has several benefits
 - (1) **Near-zero memory waste** in KV cache memory
 - (2) **Flexible sharing** of KV cache enables further memory saving
 - **Parallel sampling** (multiple outputs are generated from a single input)
 - **Beam search**
 - **Shared prefix** (i.e. system prompt)



PagedAttention: Efficient Memory Management for LLM Serving

- **PagedAttention (vLLM)** [Kwon et al., 2023]
 - PagedAttention **improves throughput** by effectively **minimizing memory waste**
 - Throughput is **improved by 2-4x** compared to the previous state-of-the-art systems



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

- Techniques for improving efficiency
- Handling long inputs with Transformers

Part 4. Summary

Handling long context with Transformers

- Common limitations of current LLMs is the **limited context length**
 - E.g. Llama-2 can take maximum 4K tokens as inputs
 - Limits the ability to handle **very long sequences**
 - Book comprehension
 - Long multi-turn chat understanding
 - Repository-level code understanding
 - ...

LATEST MODEL	DESCRIPTION	MAX TOKENS	TRAINING DATA
gpt-4	More capable than any GPT-3.5 model, able to do more complex tasks, and optimized for chat. Will be updated with our latest model iteration 2 weeks after it is released.	8,192 tokens	Up to Sep 2021
gpt-4-0613	Snapshot of gpt-4 from June 13th 2023 with function calling data. Unlike gpt-4, this model will not receive updates, and will be deprecated 3 months after a new version is released.	8,192 tokens	Up to Sep 2021
gpt-4-32k	Same capabilities as the standard gpt-4 mode but with 4x the context length. Will be updated with our latest model iteration.	32,768 tokens	Up to Sep 2021
gpt-4-32k-0613	Snapshot of gpt-4-32 from June 13th 2023. Unlike gpt-4-32k, this model will not receive updates, and will be deprecated 3 months after a new version is released.	32,768 tokens	Up to Sep 2021

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

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

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 **$O(\text{network depth} \times \text{segment length})$**
- With sophisticated implementation, computation becomes **$O(\text{input length})$**

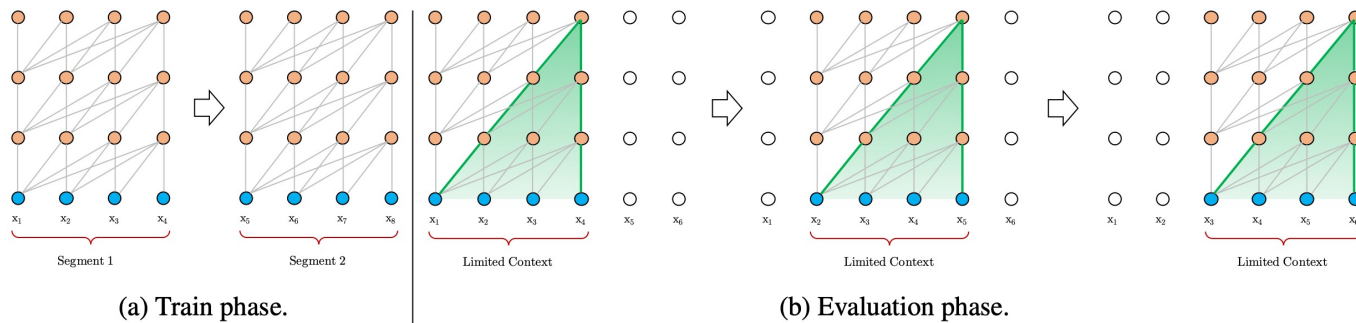


Figure 1: Illustration of the vanilla model with a segment length 4.

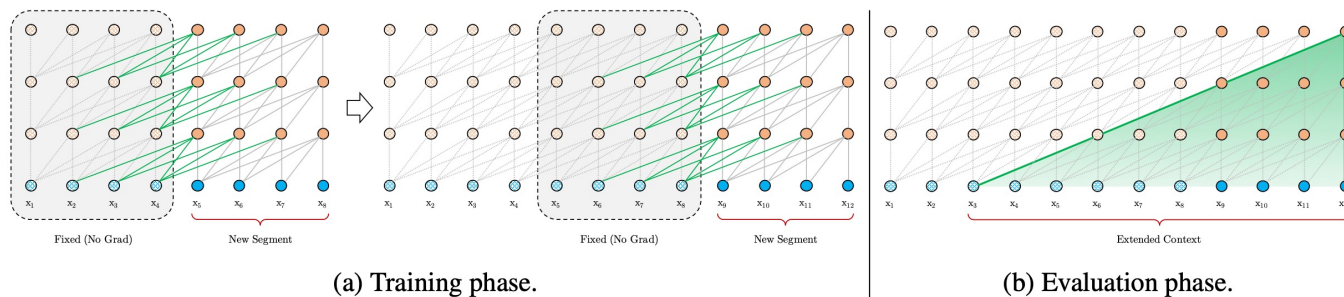
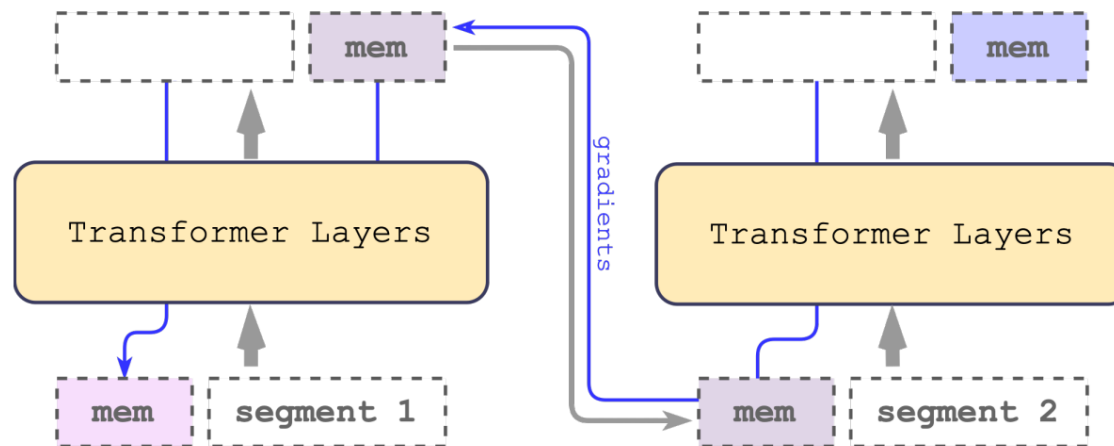


Figure 2: Illustration of the Transformer-XL model with a segment length 4.

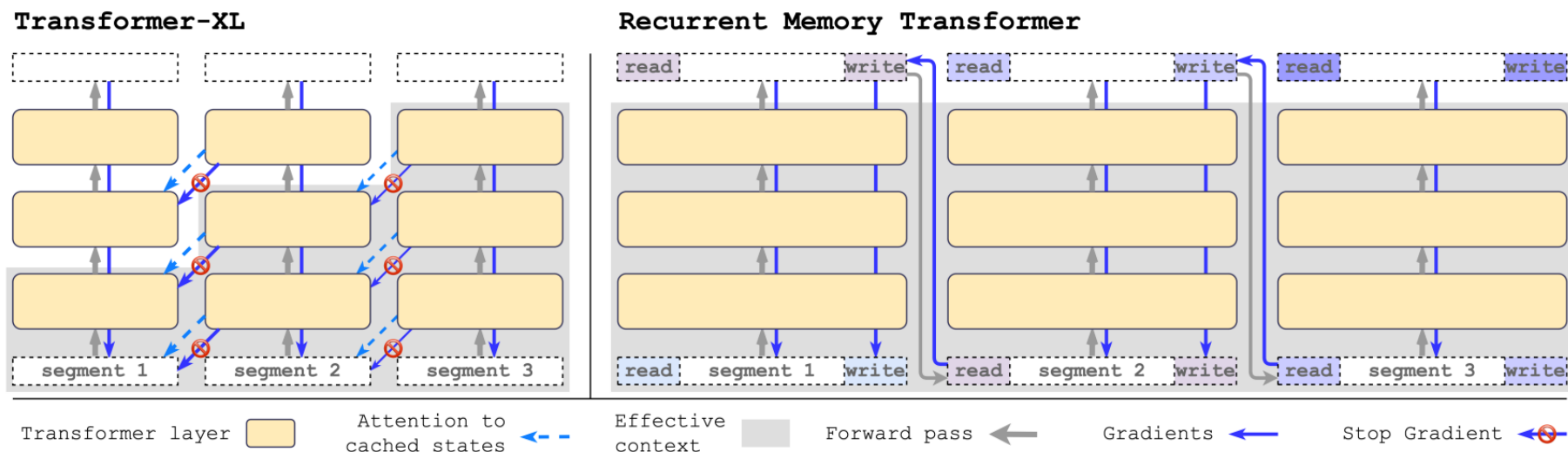
Recurrence-based Methods: Recurrent Memory Transformer

- **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

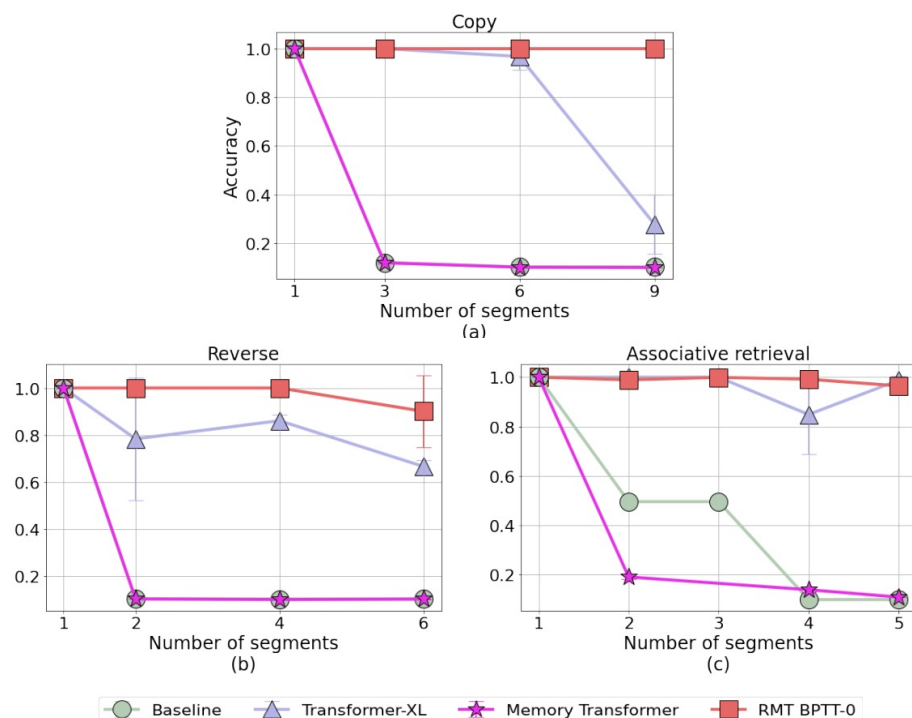
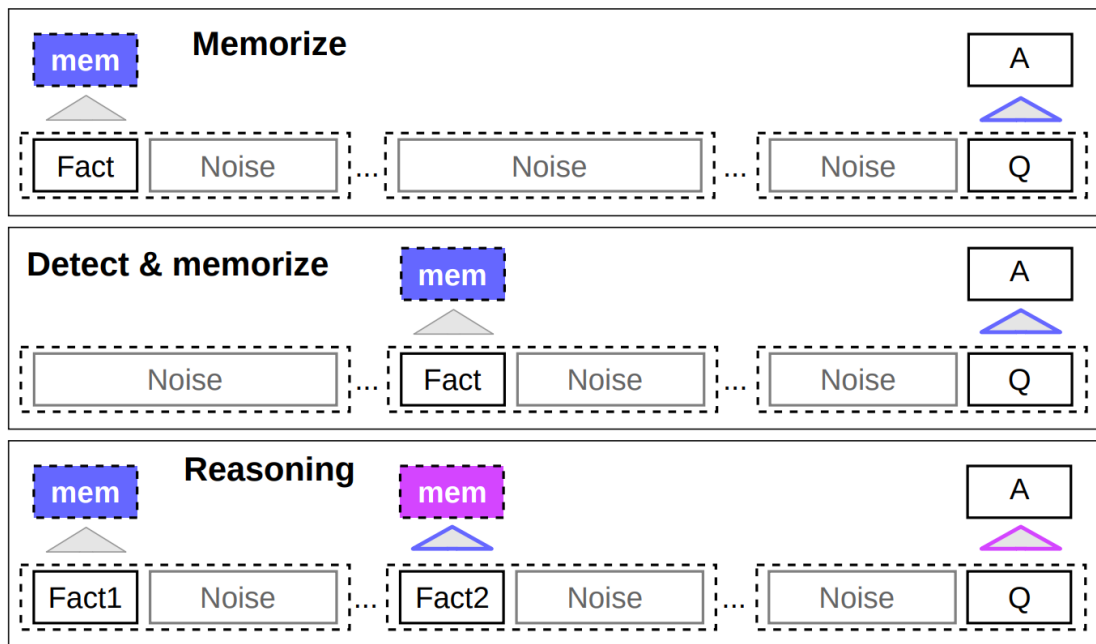


Table 2: **Language modeling on WikiText-103.** Average perplexity for the best performed variations of RMT models reported (see full results in Appendix A.5). Underlined values show Tr-XL and RMT models with close results. RMT models with smaller memory sizes achieve similar scores to Tr-XL models with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

MODEL	MEMORY	SEGMENT LEN	PPL \pm STD
TR-XL (PAPER)	150	150	24.0
BASELINE	0	150	29.95 \pm 0.15
MEMTR	10	150	29.63 \pm 0.06
TR-XL (OURS)	150	150	24.12 \pm 0.05
TR-XL	25	150	25.57 \pm 0.02
TR-XL	75	150	<u>24.68</u> \pm 0.01
RMT BPTT-3	10	150	25.04 \pm 0.07
RMT BPTT-2	25	150	<u>24.85</u> \pm 0.31
TR-XL + RMT	75+5	150	24.47 \pm 0.05
TR-XL + RMT	150+10	150	23.99 \pm 0.09
BASELINE	0	50	39.05 \pm 0.01
TR-XL	100	50	25.66 \pm 0.01
TR-XL	50	50	<u>26.54</u> \pm 0.01
TR-XL	25	50	27.57 \pm 0.09
TR-XL	10	50	28.98 \pm 0.11
RMT BPTT-1	1	50	<u>28.71</u> \pm 0.03
RMT BPTT-3	10	50	<u>26.37</u> \pm 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)

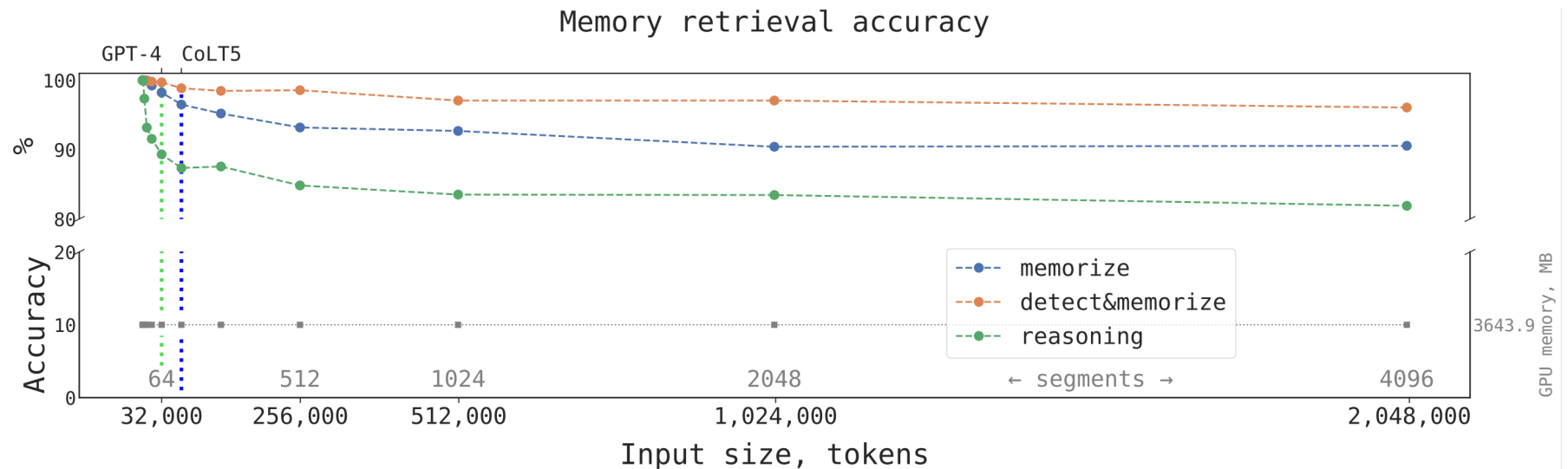


Fact: Daniel went back to the hallway.
Question: Where is Daniel?
Answer: hallway

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)

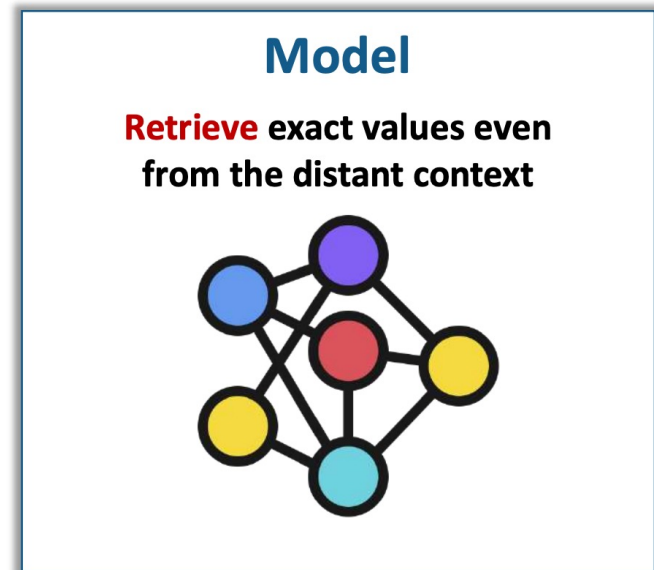
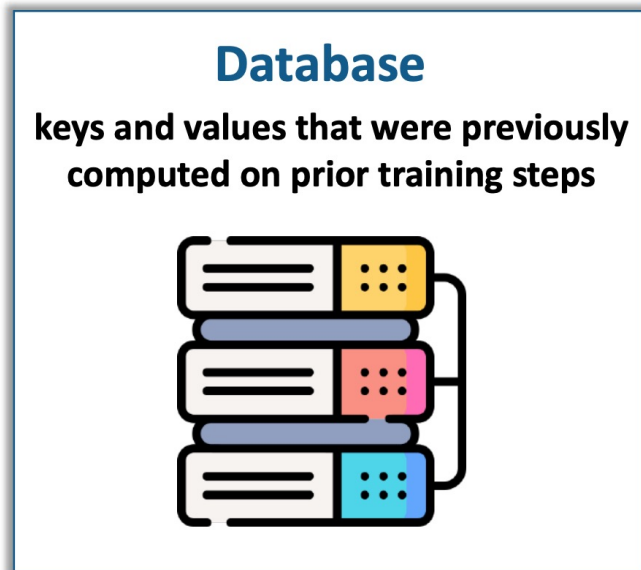


Handling long context with Transformers

- 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

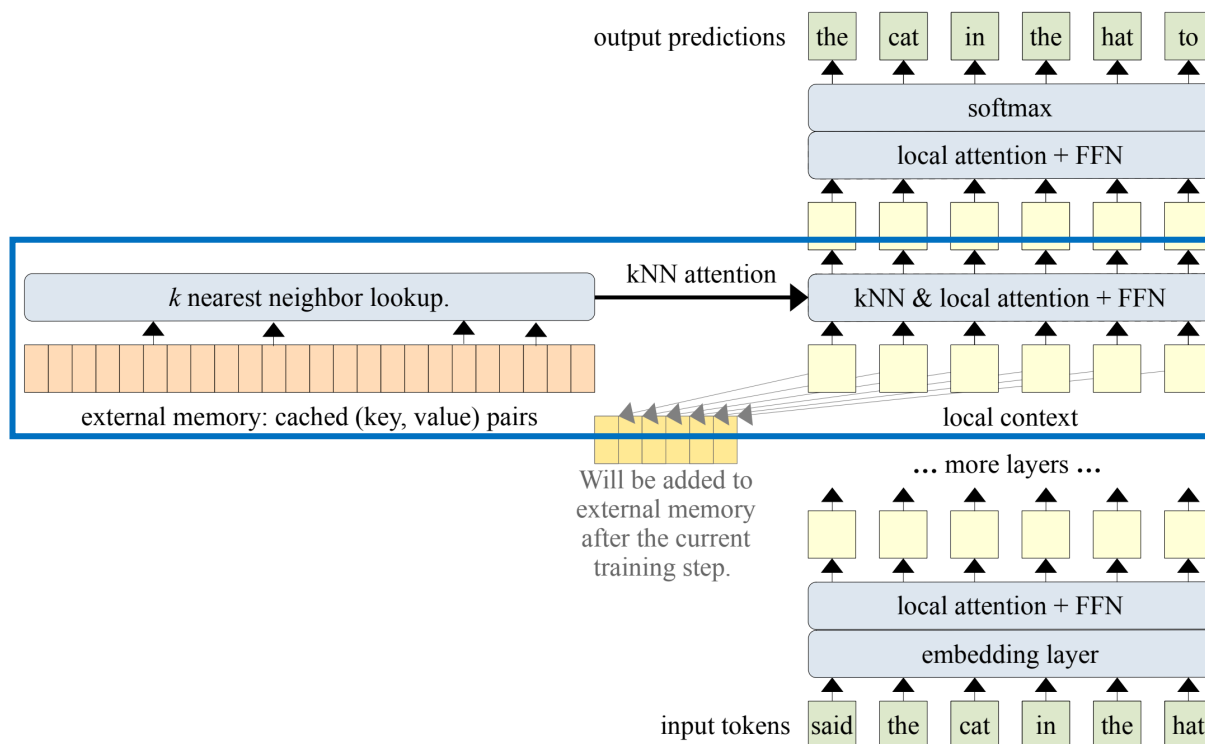
- **Retrieval-based Methods**

- 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



Retrieval-based Methods: Memorizing Transformers

- **Memorizing Transformers** [Wu et al., 2022]
 - **Idea:** Introduce a **kNN-augmented attention layer**
 - All previous (key, value) pairs are stored in an **external memory**
 - For each token, compute the hidden states in **2 different ways**
 - (1) Attending to the **local context**, as in usual self-attention
 - (2) Attending to the **top-k tokens** in the **external memory**
 - Use current token's query, and external memory's keys/values



- **Memorizing Transformers** [Wu et al., 2022]
 - **Idea:** Introduce a **kNN-augmented attention layer**
 - For each token, compute the hidden states in **2 different ways**
 - Then, combine the 2 results using a **learned gate**
 - **b_g** : Bias (learned per-head scalar parameter)
 - **V_m** : Result of attending to external memory
 - **V_c** : Result of attending to local context
 - **V_a** : Final output of the kNN-augmented attention layer

$$g = \sigma(b_g)$$
$$V_a = V_m \odot g + V_c \odot (1 - g)$$

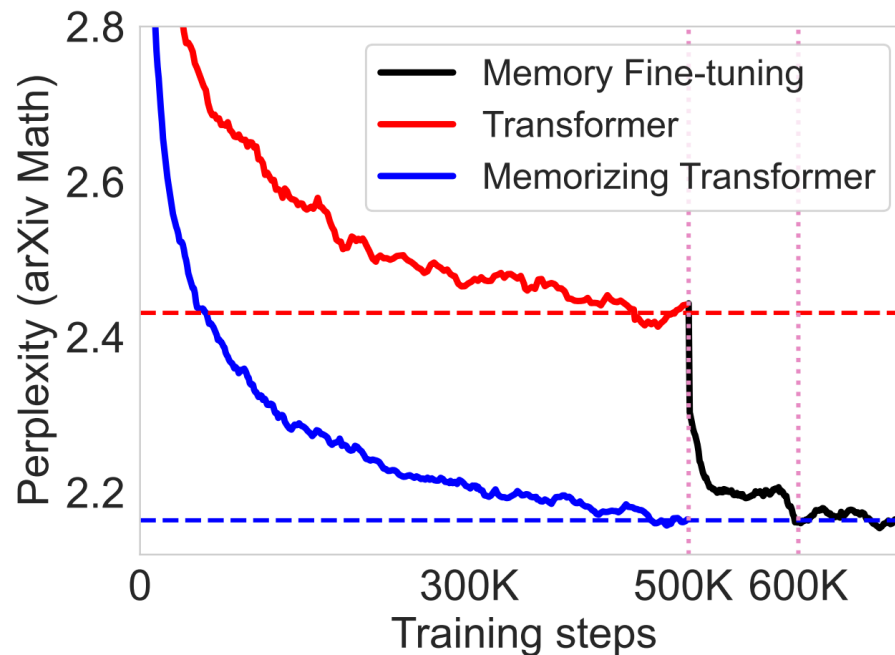
Retrieval-based Methods: Memorizing Transformers

- **Memorizing Transformers** [Wu et al., 2022]
 - **Idea:** Introduce a **kNN-augmented attention layer**
 - Experiments
 - Shows **high performance** on 5 language modeling benchmarks

Context	Memory	XL cache	arXiv	PG19	C4(4K+)	GitHub	Isabelle
512	None	None	3.29	13.71	17.20	3.05	3.09
2048	None	None	2.69	12.37	14.81	2.22	2.39
512	None	512	2.67	12.34	15.38	2.26	2.46
2048	None	2048	2.42	11.88	14.03	2.10	2.16
512	1536	None	2.61	12.50	14.97	2.20	2.33
512	8192	None	2.49	12.29	14.42	2.09	2.19
512	8192	512	2.37	11.93	14.04	2.03	2.08
512	65K	512	2.31	11.62	14.04	1.87	2.06
2048	8192	2048	2.33	11.84	13.80	1.98	2.06
2048	65K	2048	2.26	11.37	13.64	1.80	1.99

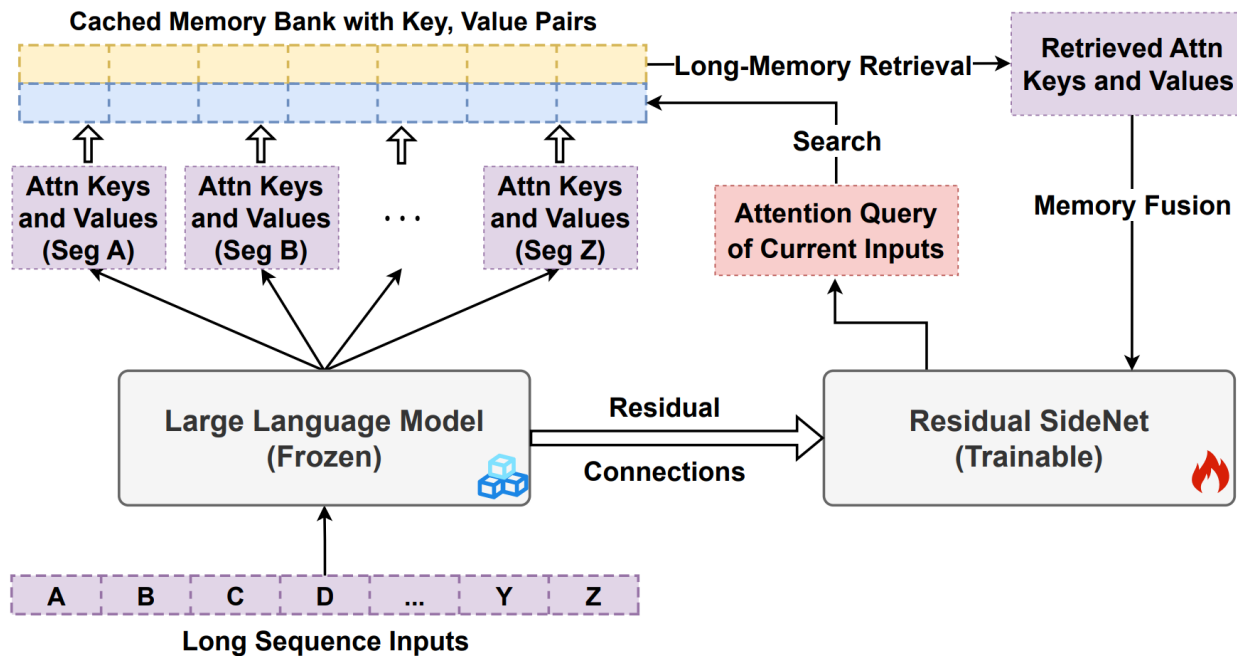
Retrieval-based Methods: Memorizing Transformers

- **Memorizing Transformers** [Wu et al., 2022]
 - **Idea:** Introduce a **kNN-augmented attention layer**
 - Experiments
 - Shows **high performance** on 5 language modeling benchmarks
 - **Regular Transformers** can be further fine-tuned to utilize memory



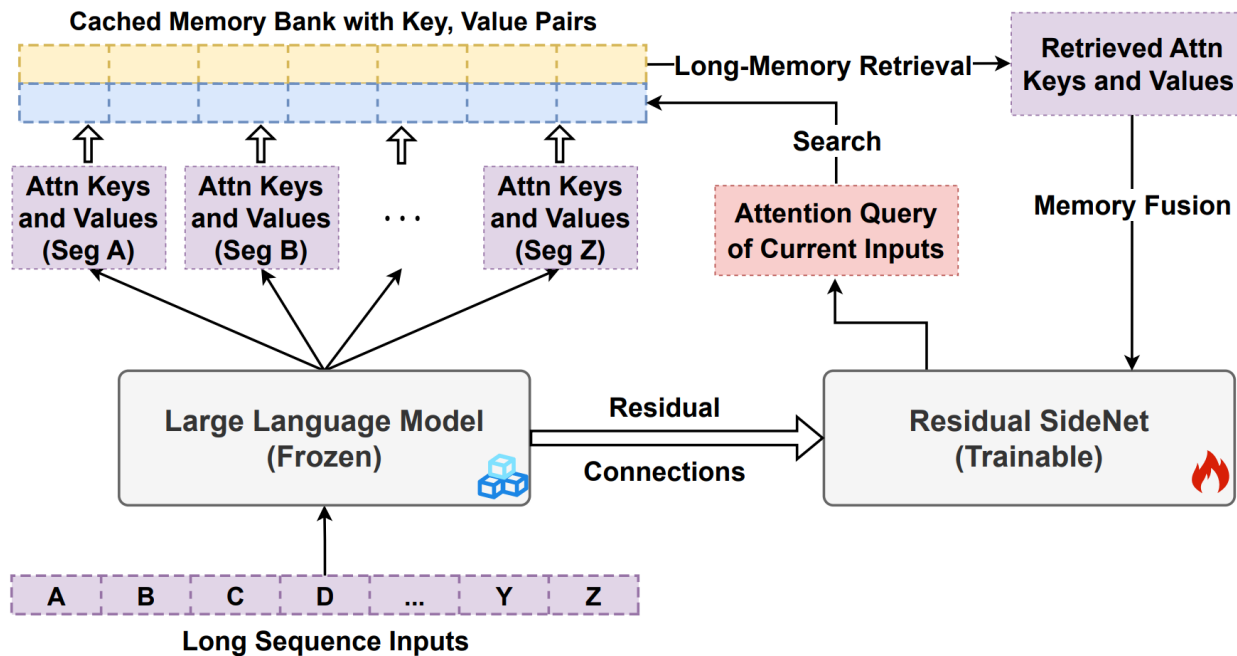
Retrieval-based Methods: LongMem

- **LongMem** [Wang et al., 2023]
 - **Motivation:** Memory transformer suffers from **memory staleness** during training
 - Cached database suffers from **distribution shift** due to parameter updates
 - This problem comes from **coupled memory design**
 - A **single model** is used for encoding and fusing memory
 - LongMem proposes a **decoupled memory design**
 - Encoding and memory fusion takes place in a separate network



Retrieval-based Methods: LongMem

- **LongMem** [Wang et al., 2023]
 - Proposed **decoupled memory design**
 - Use a **frozen LLM** for encoding
 - Train another network (**SideNet**) to perform memory fusion and language generation
 - The benefits of such design includes:
 - Utilization of **strong encoding capability** of a frozen LLM
 - Effectively bypassing the **memory staleness** challenge



Retrieval-based Methods: LongMem

- **LongMem** [Wang et al., 2023]
 - LongMem shows improved **language modeling** performance
 - Note: MemTRM refers to Memory Transformers

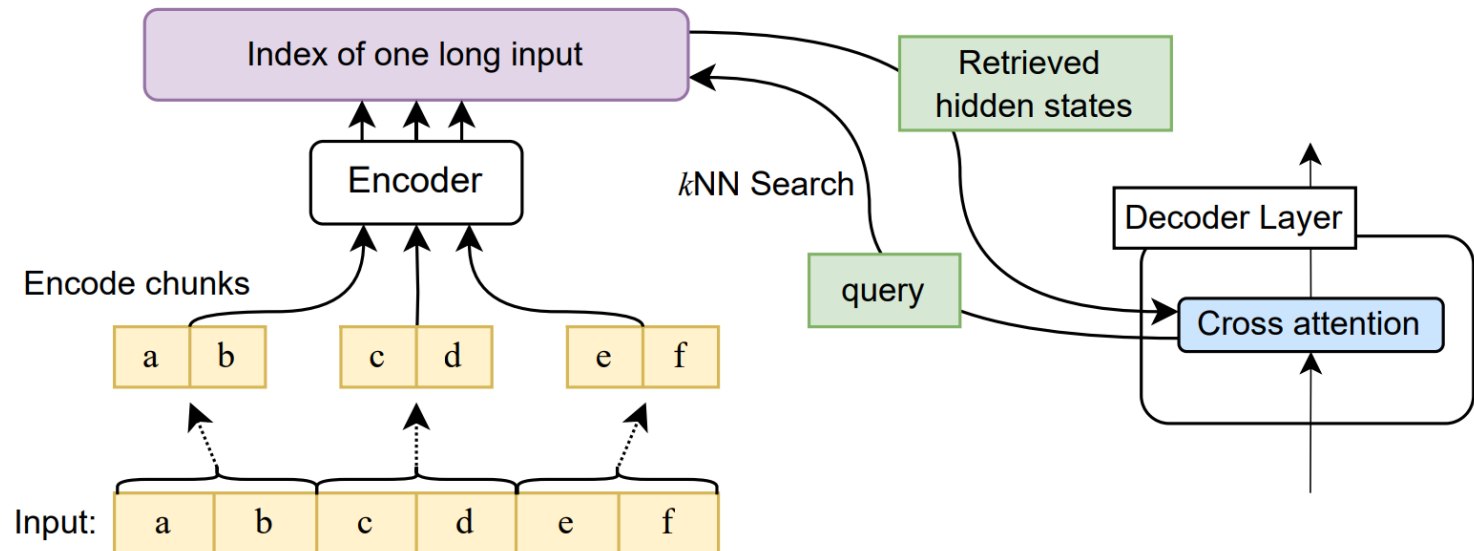
Model	In-Context Len.	In-Memory Len.	PG-22					ArXiv
			5K-10K	10K-100K	100K-500K	500K-1M	>1M	
GPT-2*	1k	N/A	22.78	24.39	24.12	24.97	18.07	11.05
MemTRM	1k	65K	21.77	23.56	23.23	24.16	17.39	10.81
LONGMEM	1k	65k	21.29	23.01	22.55	23.35	16.71	10.05

- LongMem also shows improved performance on **ChapterBreak benchmark**
 - Identifying true chapter beginnings among false ones

Model	#Params	In-Context Len.	In-Memory Len.	ChapterBreak _{ao3}		
				ctx-4k	ctx-6k	ctx-8k
GPT-2-XL [†] [RWC ⁺ 19]	1.5B	1K	N/A	24%	24%	24%
GPT-3 [†] [BMR ⁺ 20]	175B	2K	N/A	28%	28%	28%
LocalTRM [†] [RSVG21]	516M	8K	N/A	24%	24%	24%
RoutTRM [†] [RSVG21]	490M	8K	N/A	25%	24%	24%
Bigbird [†] [ZGD ⁺ 20]	128M	4K	N/A	26%	26%	26%
GPT-2*	407M	1K	N/A	18.4%	18.4%	18.4%
MemTRM	407M	1K	∞	28.3%	28.7%	28.7%
LONGMEM	558M	1K	∞	37.7%	39.4%	40.5%

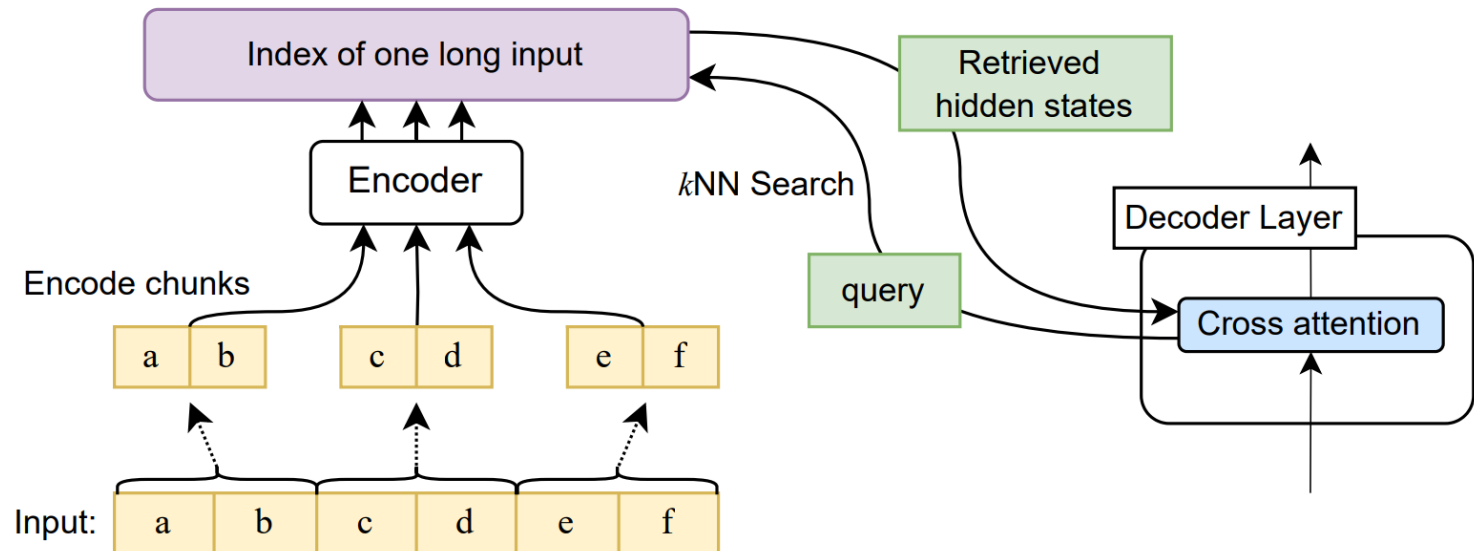
Retrieval-based Methods: Unlimiformer

- **Unlimiformer** [Bertsch et al., 2023]
 - **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., 2023]
 - 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**



- **Unlimiformer** [Bertsch et al., 2023]
 - **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

$$\begin{aligned} QK^T &= (\mathbf{h}_d W_q) (\mathbf{h}_e W_k)^T \\ &= (\mathbf{h}_d W_q) W_k^T \mathbf{h}_e^T \\ &= (\mathbf{h}_d W_q W_k^T) \mathbf{h}_e^T \end{aligned}$$

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., 2023]
 - Unlimiformer extends the context limit **without fine-tuning**
 - Evaluated on summarization benchmarks

Base model	Training method	ROUGE 1 / 2 / L / BERTScore							
		GovReport		SummScreen					
BART _{base}	Standard finetuning	48.7	19.2	22.8	64.3	29.7	6.2	17.7	56.3
BART _{base}	+test SLED (Ivgi et al., 2022)	45.8	16.1	20.2	62.7	27.5	5.5	16.7	55.9
BART _{base}	+test Unlimiformer	49.7	19.6	22.0	64.8	30.9	6.5	18.2	57.5
BART _{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

BART _{base}	Train chunked	46.2	17.8	21.7	63.3	28.1	5.6	17.0	55.6
BART _{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

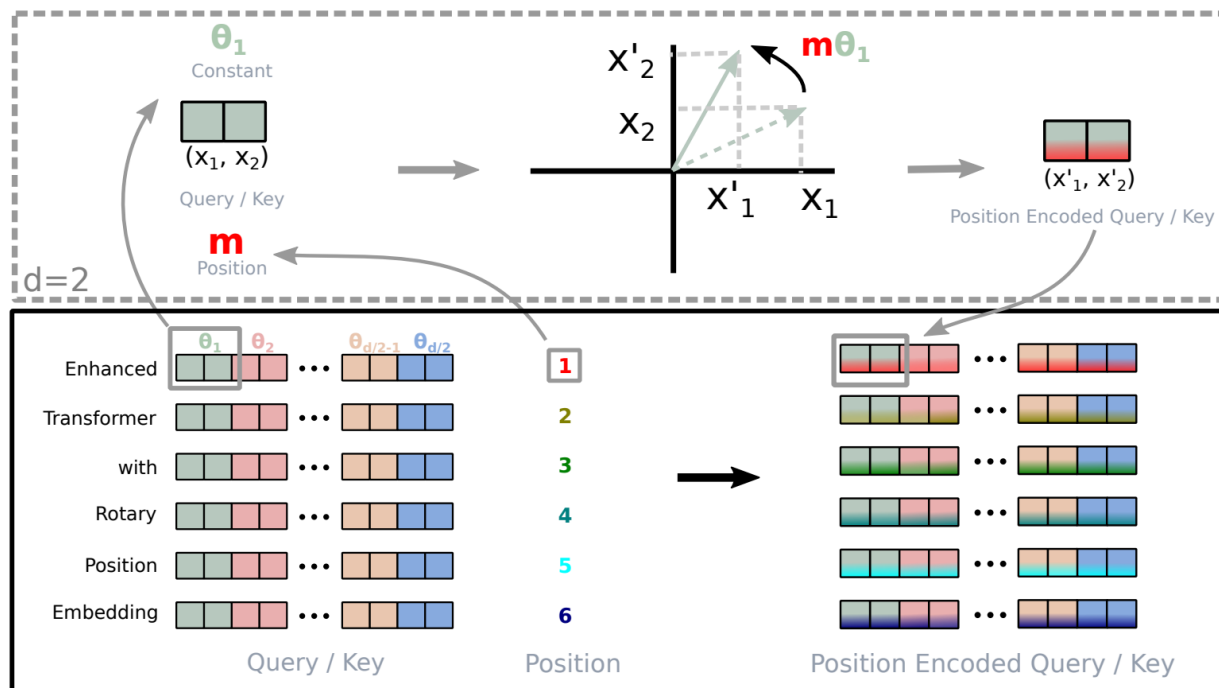
- **Unlimiformer** [Bertsch et al., 2023]
 - Unlimiformer can handle extremely long (book-level) context
 - Evaluated on BookSum (average input length ~143k tokens)

Base model	Training method	ROUGE 1 / 2 / L	EntMent
BART _{base}	Hierarchical (Kryściński et al., 2021)	30.0 / 6.0 / 11.0	-
BART _{base}	Standard finetuning	36.4 / 7.6 / 15.3	10.0
BART _{base}	+test Unlimiformer	35.5 / 7.7 / 15.4	21.9
BART _{base}	+early stop w/ Unlimiformer	35.5 / 7.7 / 15.4	21.9
BART _{base}	Memorizing Transformers	35.6 / 6.4 / 14.6	10.1
BART _{base}	Unlimiformer (random-encoded training)	37.3 / 6.7 / 15.2	20.8
BART _{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

- **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



- **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)
 - **x_m:** hidden states
 - **d:** hidden state dimension

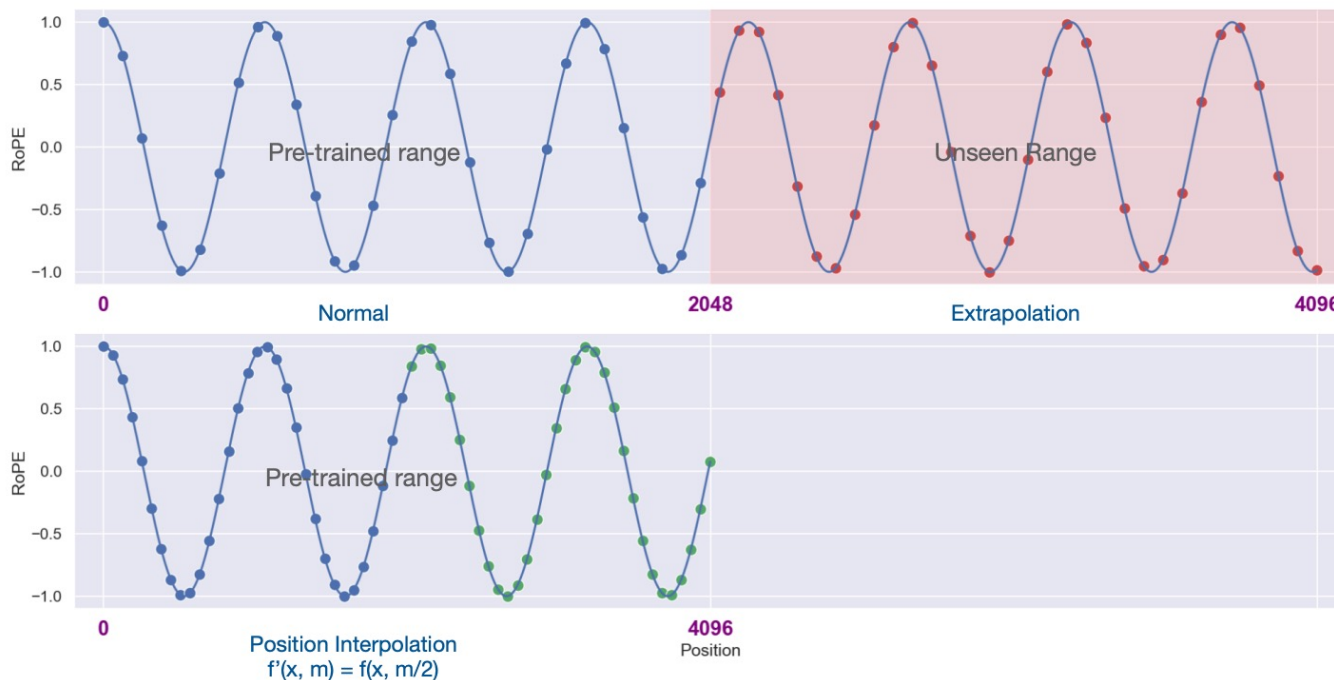
$$f_{\mathbf{W}}(\mathbf{x}_m, m, \theta_i) = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0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

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

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

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



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

Size	Model		Evaluation Context Window 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

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

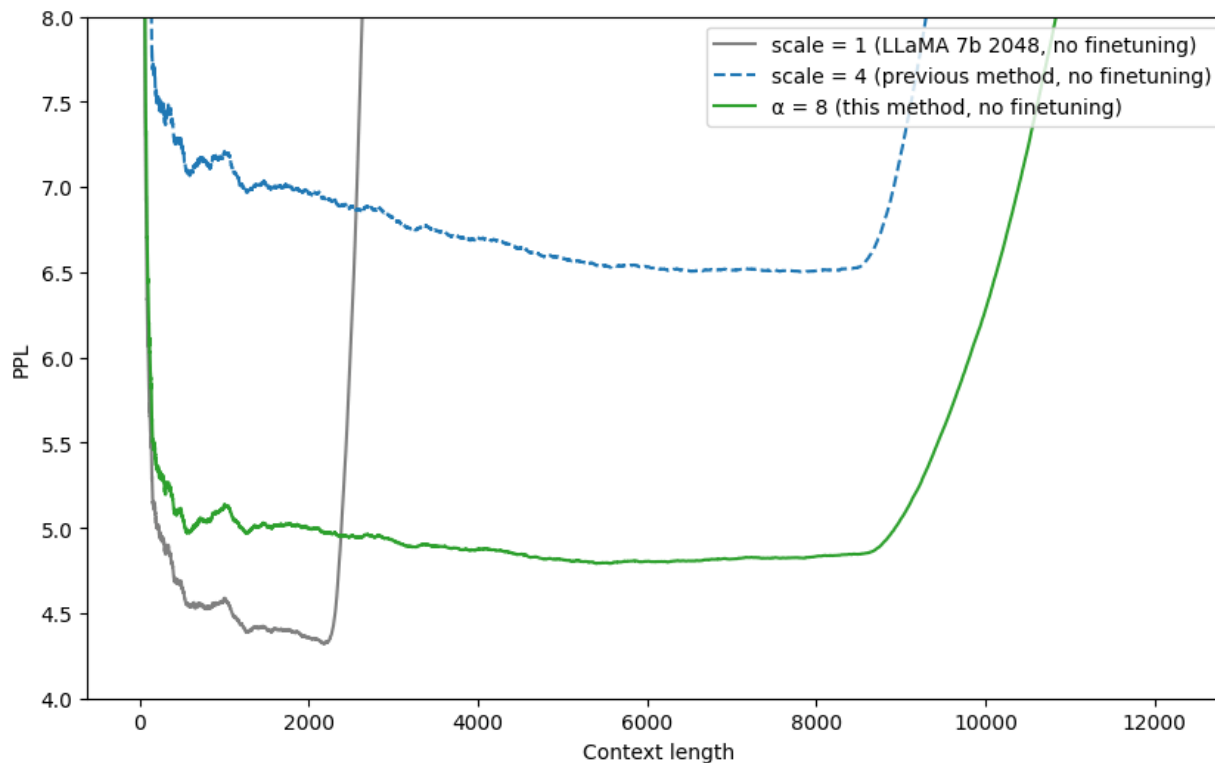


$$\theta'_i = \left(\underline{10000 \times s^{\frac{d}{d-2}}} \right)^{-2(i-1)/d}, i \in [1, 2, \dots, d/2]$$

base is scaled according to
the scaling factor s

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

- **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

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

- **NTK-by-parts Scaling** [bloc97, 2023]
 - Investigates deeper into the **different characteristics of each dimension**
 - Dimensions with $\lambda > L$ (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 $\lambda \ll L$
 - The embedding **rotates multiple times** within the context limit
 - Position embeddings are **not unique**
 - Intuition: Only **relative positional information** can be modeled

- **NTK-by-parts Scaling** [bloc97, 2023]

- **Idea:** Interpolate **only the dimensions with long wavelengths**

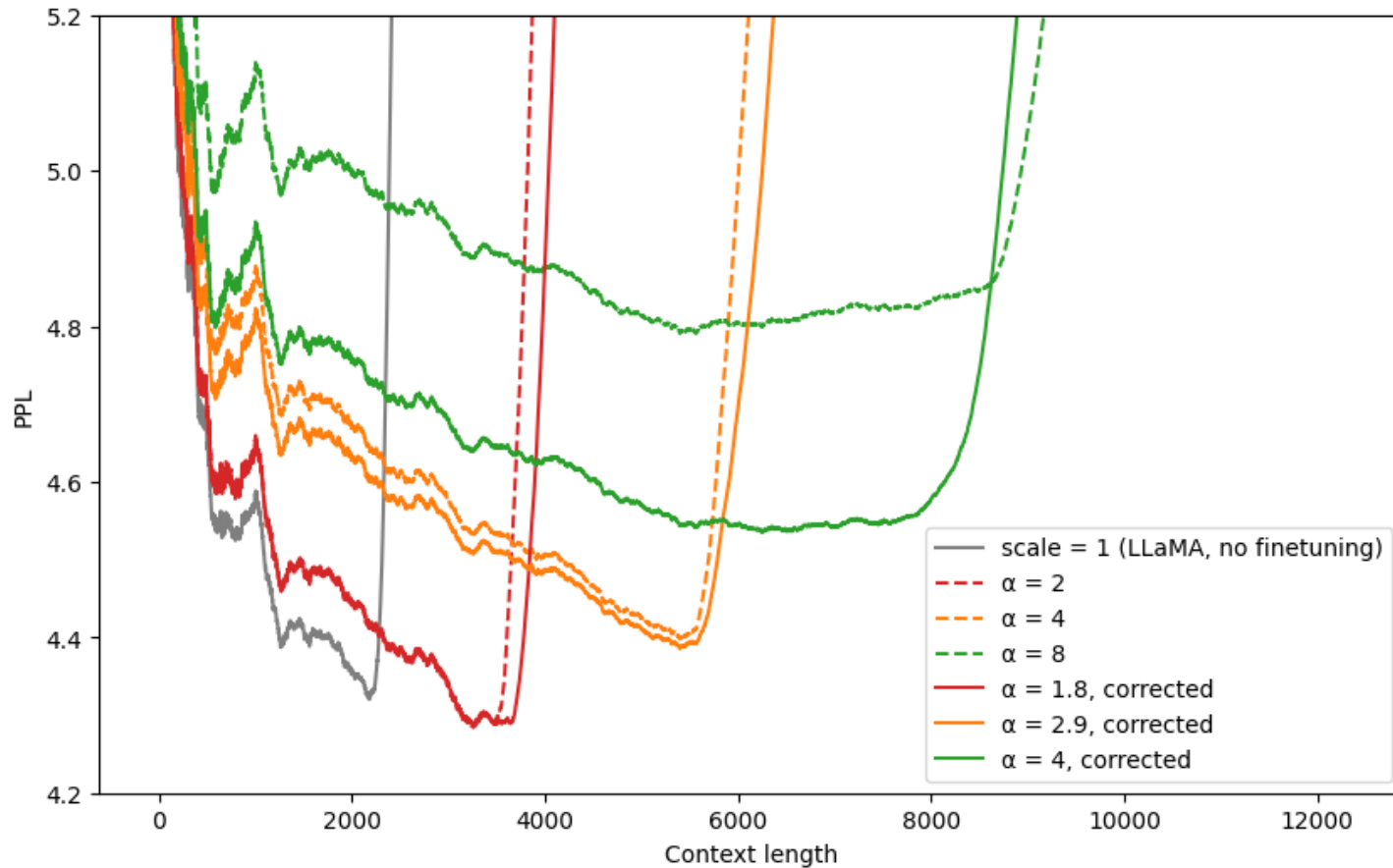
- L : Original context length, L' : Extended context length
- α, β are hyperparameters

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

- Dimensions with $\lambda > L$
 - These dimensions model the **global** position information
 - Use **linear interpolation**
- Dimensions with $\lambda \ll L$
 - 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

- **YaRN (Yet another RoPE ExtensionN method)** [Peng et al., 2023]
 - **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'

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

- 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 (Yet another RoPE Extension method)** [Peng et al., 2023]
 - 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 ($\theta = 20k$)	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 (Yet another RoPE Extension method)** [Peng et al., 2023]
 - 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

- Techniques for improving efficiency
- Handling long inputs with Transformers

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**

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-learning>
- [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” arxiv 2023

Link: <https://arxiv.org/abs/2309.00071>