

Transformers & Large Language Models

André F. T. Martins



16th Lisbon Machine Learning Summer School, LxMLS 2026

“Structure AwaRe moDeLling for Natural LanguageE”



<https://sardine-lab.github.io/>

We do research in NLP, LLMs, sparsity, long-context efficiency, ...

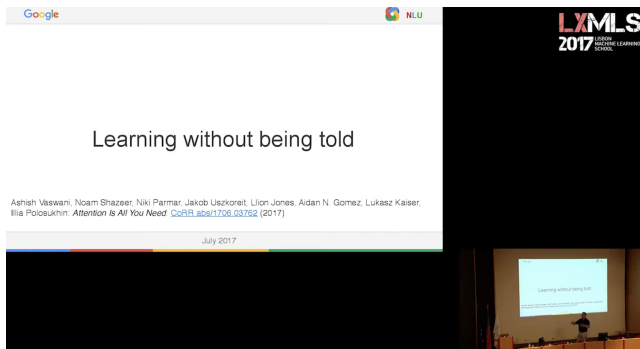
9 Years Ago: Transformers at LxMLS 2017

Published in ArXiv on June 12, 2017...

9 Years Ago: Transformers at LxMLS 2017

Published in ArXiv on June 12, 2017...

... presented at LxMLS on **July 21** by Fernando Pereira:



[slides](#) & [video](#)

Outline

1 Transformers

- The Attention Mechanism
- The Transformer Architecture
- Handling Long Contexts
- What's Next?

2 Large Language Models

- The BERT Era: Pretraining and Fine-tuning
- The GPT-3 Era: Prompting and In-Context Learning
- The ChatGPT Era: Post-training and Preference Optimization
- The DeepSeek R1 Era: Reasoning
- What Next?

3 Conclusions

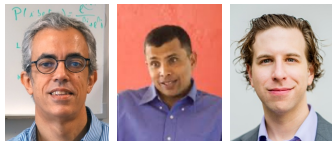
What You've Learned So Far

- ✓ Linear models
- ✓ Feedforward neural networks
- ✓ Recurrent neural networks



What You've Learned So Far

- ✓ Linear models
- ✓ Feedforward neural networks
- ✓ Recurrent neural networks

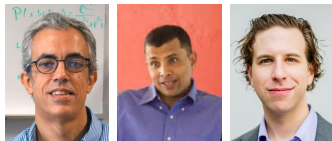


RNNs are suitable for modeling **sequences** as they carry **state** information.

- Many improved variants: GRUs, LSTMs, Mamba, DeltaNet, ...
- All these have carry **“fixed-size” memory** in their state.

What You've Learned So Far

- ✓ Linear models
- ✓ Feedforward neural networks
- ✓ Recurrent neural networks



RNNs are suitable for modeling **sequences** as they carry **state** information.

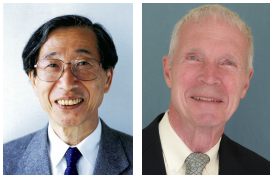
- Many improved variants: GRUs, LSTMs, Mamba, DeltaNet, ...
- All these have carry **“fixed-size” memory** in their state.

Today: models with **nonparametric associative memory**.

What is an Associative Memory?

- A memory paradigm inspired by the way the brain stores and retrieves information.
- It allows data retrieval based on **content** rather than explicit physical addressing.
- **Content-based recall**: In the brain, seeing or hearing a partial cue can trigger the recall of associated memories.

Hopfield Networks (Amari, 1972; Hopfield, 1982)



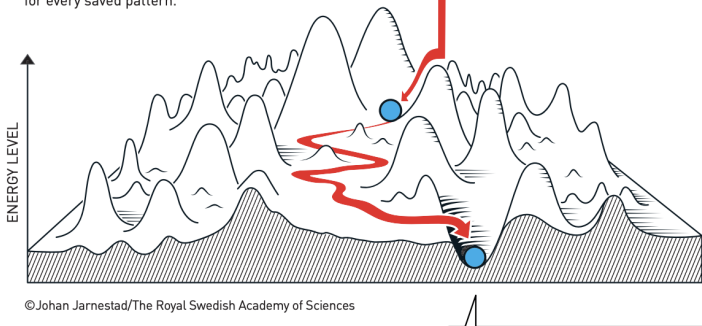
One of the earliest neural network designed to model **associative memory**.

- Named after John Hopfield, an American physicist and neuroscientist.
- Based on older ideas by Shun'ichi Amari in the 70s.
- Inspired by associative memory in the human brain.
- Stores and recalls patterns, making it useful for pattern recognition.

Hopfield Networks as Associative Memories

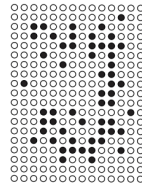
Memories are stored in a landscape

John Hopfield's associative memory stores information in a manner similar to shaping a landscape. When the network is trained, it creates a valley in a virtual energy landscape for every saved pattern.

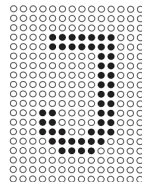


1 When the trained network is fed with a distorted or incomplete pattern, it can be likened to dropping a ball down a slope in this landscape.

INPUT PATTERN



SAVED PATTERN



2 The ball rolls until it reaches a place where it is surrounded by uphills. In the same way, the network makes its way towards lower energy and finds the closest saved pattern.

Hopfield Networks (Amari, 1972; Hopfield, 1982)

Networks of D units, fully connected.

- Memory patterns $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n] \in \{\pm 1\}^{n \times d}$

- Query $\mathbf{q} \in \{\pm 1\}^d$

- Energy function $E(\mathbf{q}) = -\frac{1}{2} \|\mathbf{X}\mathbf{q}\|^2$

- Hopfield dynamics:

$$\mathbf{q}_{t+1} = \text{sign}(\underbrace{\mathbf{X}^\top \mathbf{X}}_{:= \mathbf{W}} \mathbf{q}_t)$$

- Patterns are stored in the memory by computing the network parameters $\mathbf{W} = \mathbf{X}^\top \mathbf{X} \in \mathbb{R}^{d \times d}$

- Memory patterns become **attractors**.

Hopfield Networks

Let's try to memorize some LxMLS speakers!



Hopfield Networks

Let's try to memorize some LxMLS speakers!



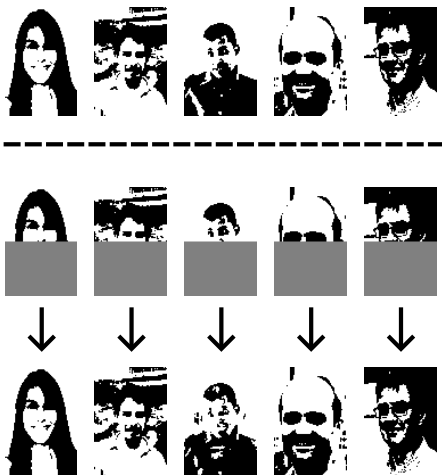
Hopfield Networks

Let's try to memorize some LxMLS speakers!



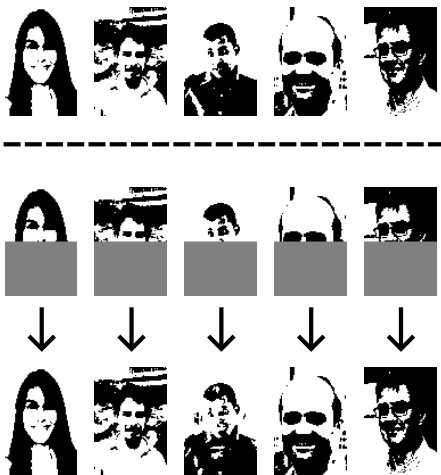
Hopfield Networks

Let's try to memorize some LxMLS speakers!



Hopfield Networks

Let's try to memorize some LxMLS speakers! **Works pretty well.**



Hopfield Networks

What if we try with **all the speakers**?



Hopfield Networks

What if we try with **all the speakers**?



Hopfield Networks

What if we try with **all the speakers**? Doesn't work due to **interference**.



Hopfield Networks

Classical Hopfield networks have a **very small** memory capacity: at most $\approx 0.14d$ patterns can be memorized.

They are a bit like **linear transformers** (more later).

We can modify the energy function to increase memory capacity, making it **nonparametric**.

“Modern” Hopfield networks do this: they are **nonparametric** like attention-based transformers.

(Ramsauer et al., “Hopfield Networks is All You Need”, ICLR 2020)

<https://ml-jku.github.io/hopfield-layers/>

Outline

1 Transformers

- The Attention Mechanism
- The Transformer Architecture
- Handling Long Contexts
- What's Next?

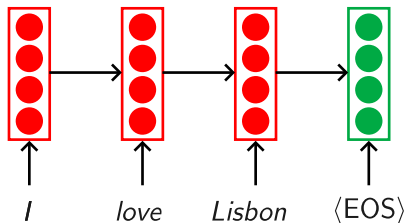
2 Large Language Models

- The BERT Era: Pretraining and Fine-tuning
- The GPT-3 Era: Prompting and In-Context Learning
- The ChatGPT Era: Post-training and Preference Optimization
- The DeepSeek R1 Era: Reasoning
- What Next?

3 Conclusions

RNNs as Seq-to-Seq Models (Sutskever et al., 2014)

An **encoder RNN** to encode the source sentence as a **vector**...

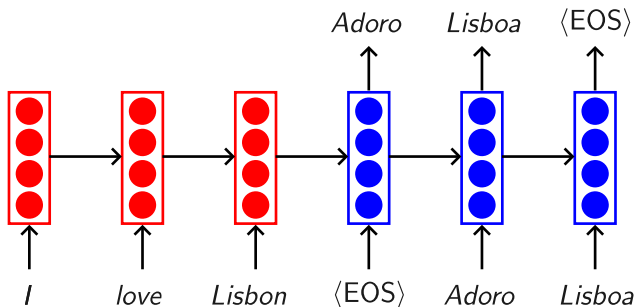


(Sutskever et al., "Sequence-to-Sequence Learning with Neural Networks", NeurIPS 2014)

RNNs as Seq-to-Seq Models (Sutskever et al., 2014)

An **encoder RNN** to encode the source sentence as a **vector**...

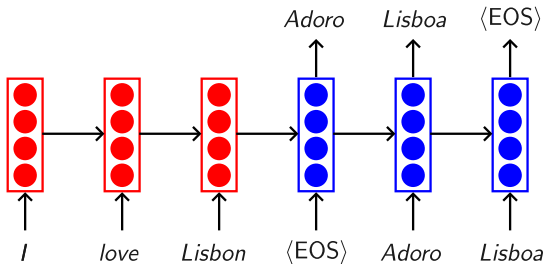
... a **decoder RNN** to decode the target sequence autoregressively.



(Sutskever et al., "Sequence-to-Sequence Learning with Neural Networks", NeurIPS 2014)

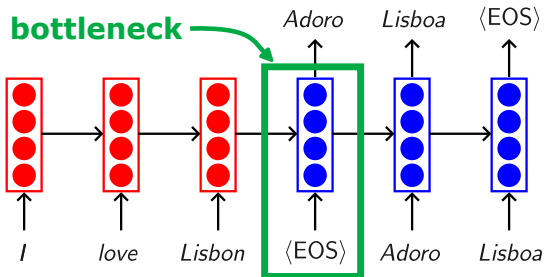
Limitations of RNNs

Sentence length is **unbounded** but vectors have **finite** capacity.



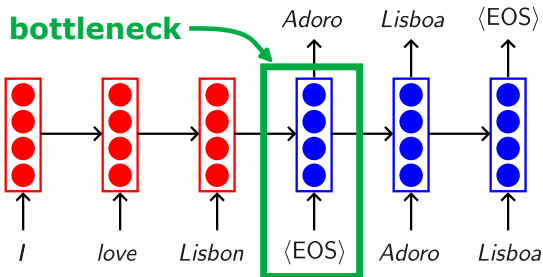
Limitations of RNNs

Sentence length is **unbounded** but vectors have **finite** capacity.



Limitations of RNNs

Sentence length is **unbounded** but vectors have **finite** capacity.



"You can't cram the meaning of a whole %&\$@# sentence into a single \$@&# vector!" (Ray Mooney)



Solution: Attention Mechanism (Bahdanau et al., 2015)

Encode sentences as matrices, not vectors!

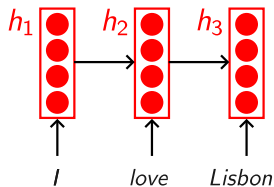
- Fixed number of rows, but number of columns depends on the number of words
- Then, before generating each word in the decoder, use an **attention mechanism** to condition on the relevant source words only.

This was proposed in a seminal paper in 2015 and opened the door for **transformer** architectures in 2017.

(Bahdanau et al., “Neural MT by Jointly Learning to Align and Translate”, ICLR 2015)

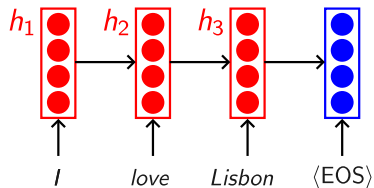
(Sukhbaatar et al., “End-to-End Memory Networks”, NeurIPS 2015)

RNNs with Attention (Bahdanau et al., 2015)



- Encoder states
 $h_1, \dots, h_n$

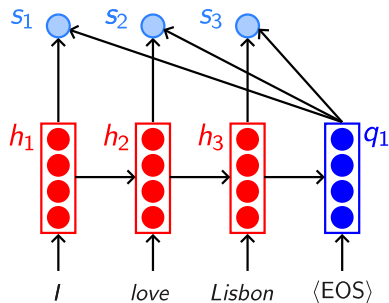
RNNs with Attention (Bahdanau et al., 2015)



■ Decoder state q_t

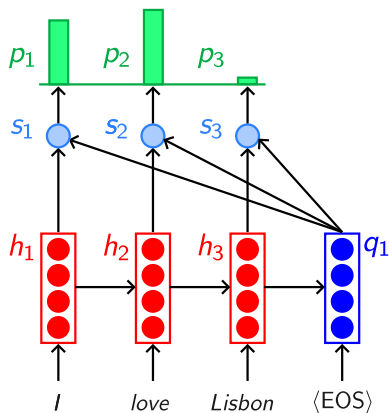
■ Encoder states
 $h_1, \dots, h_n$

RNNs with Attention (Bahdanau et al., 2015)



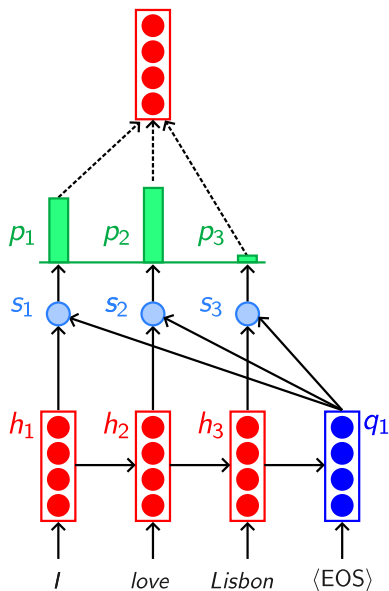
- Attention scores
 $s_1, \dots, s_n$ obtained by comparing q_t with each h_i
- Decoder state q_t
- Encoder states
 $h_1, \dots, h_n$

RNNs with Attention (Bahdanau et al., 2015)



- Attention probabilities
 $p = \text{softmax}(s)$
- Attention scores
 $s_1, \dots, s_n$ obtained by comparing q_t with each h_i
- Decoder state q_t
- Encoder states
 $h_1, \dots, h_n$

RNNs with Attention (Bahdanau et al., 2015)



- Weighted sum of encoder states:

$$\sum_{i=1}^n p_i h_i$$

- Attention probabilities

$$p = \text{softmax}(s)$$

- Attention scores

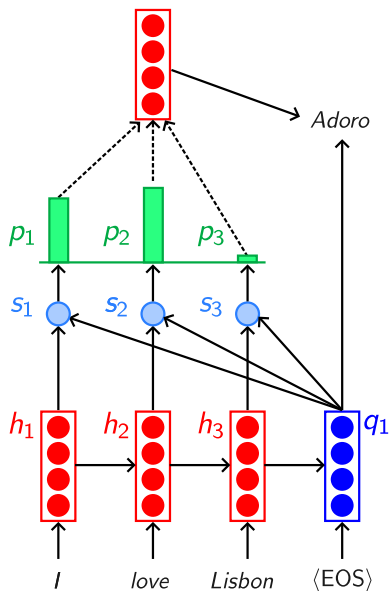
$s_1, \dots, s_n$ obtained by comparing q_t with each h_i

- Decoder state q_t

- Encoder states

$$h_1, \dots, h_n$$

RNNs with Attention (Bahdanau et al., 2015)



- Weighted sum of encoder states:
$$\sum_{i=1}^n p_i h_i$$
- Attention probabilities
 $p = \text{softmax}(s)$
- Attention scores
 $s_1, \dots, s_n$ obtained by comparing q_t with each h_i
- Decoder state q_t
- Encoder states
 $h_1, \dots, h_n$

RNNs with Attention (Bahdanau et al., 2015)

- Weighted sum of encoder states:

$$\sum_{i=1}^n p_i \mathbf{h}_i$$

- Attention probabilities

$$\mathbf{p} = \text{softmax}(\mathbf{s})$$

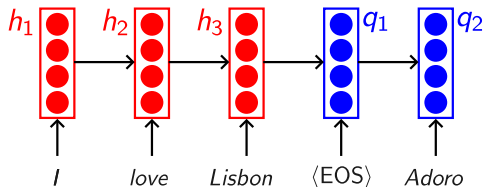
- Attention scores

$s_1, \dots, s_n$ obtained by comparing $\mathbf{q}_t$ with each $\mathbf{h}_i$

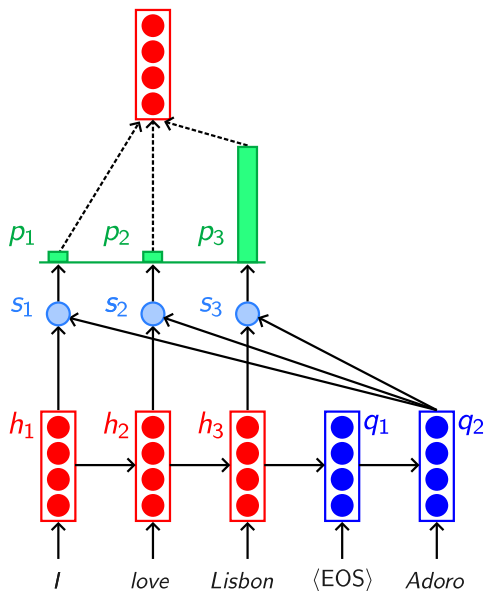
- Decoder state $\mathbf{q}_t$

- Encoder states

$\mathbf{h}_1, \dots, \mathbf{h}_n$



RNNs with Attention (Bahdanau et al., 2015)



- Weighted sum of encoder states:

$$\sum_{i=1}^n p_i \mathbf{h}_i$$

- Attention probabilities

$$\mathbf{p} = \text{softmax}(\mathbf{s})$$

- Attention scores

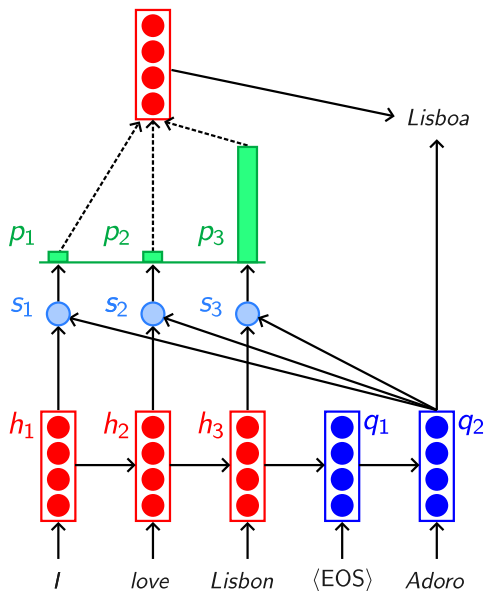
$s_1, \dots, s_n$ obtained by comparing $\mathbf{q}_t$ with each $\mathbf{h}_i$

- Decoder state $\mathbf{q}_t$

- Encoder states

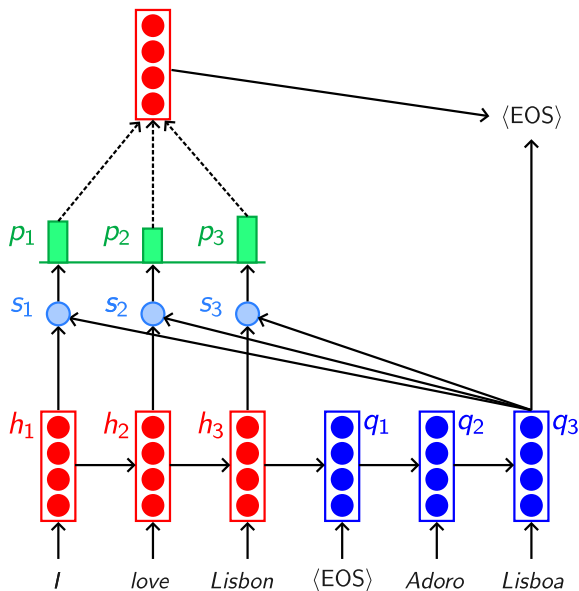
$\mathbf{h}_1, \dots, \mathbf{h}_n$

RNNs with Attention (Bahdanau et al., 2015)



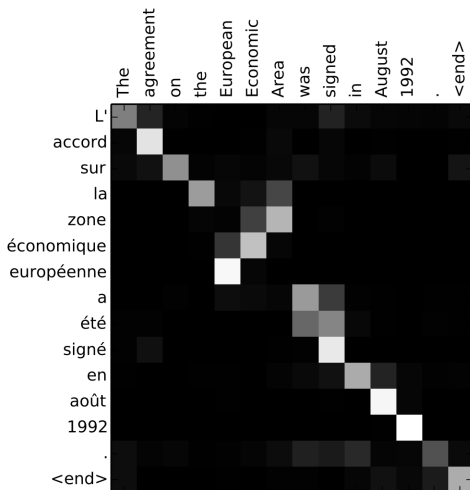
- Weighted sum of encoder states:
$$\sum_{i=1}^n p_i h_i$$
- Attention probabilities
 $p = \text{softmax}(s)$
- Attention scores
 $s_1, \dots, s_n$ obtained by comparing q_t with each h_i
- Decoder state q_t
- Encoder states
 $h_1, \dots, h_n$

RNNs with Attention (Bahdanau et al., 2015)



- Weighted sum of encoder states:
$$\sum_{i=1}^n p_i h_i$$
- Attention probabilities
 $p = \text{softmax}(s)$
- Attention scores
 $s_1, \dots, s_n$ obtained by comparing q_t with each h_i
- Decoder state q_t
- Encoder states
 $h_1, \dots, h_n$

Example: Machine Translation



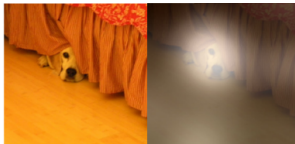
(Bahdanau et al., "Neural MT by Jointly Learning to Align and Translate", ICLR 2015)

Example: Caption Generation

Attention over images:



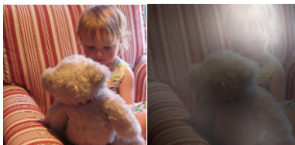
A woman is throwing a frisbee in a park.



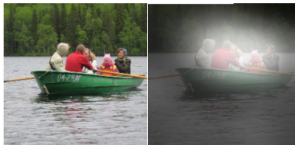
A dog is standing on a hardwood floor.



A stop sign is on a road with a mountain in the background.



A little girl sitting on a bed with a teddy bear.



A group of people sitting on a boat in the water.



A giraffe standing in a forest with trees in the background.

(Slide credit to Yoshua Bengio)

Attention Mechanism: Wrapping It Up

- 1 We have a **query vector** $\mathbf{q}$ (e.g. the decoder state)
- 2 We have **input vectors** $\mathbf{H} = [\mathbf{h}_1, \dots, \mathbf{h}_n]^\top$ (e.g. one per source word)
- 3 We compute **association scores** $s_1, \dots, s_n$ by “comparing” $\mathbf{q}$ and $\mathbf{H}$
- 4 We convert these scores to **probabilities**:

$$\mathbf{p} = \text{softmax}(\mathbf{s})$$

- 5 We use this to output a representation as a **weighted average**:

$$\mathbf{c} = \mathbf{H}^\top \mathbf{p} = \sum_{i=1}^n p_i \mathbf{h}_i$$

Attention Mechanism: Wrapping It Up

- 1 We have a **query vector** $\mathbf{q}$ (e.g. the decoder state)
- 2 We have **input vectors** $\mathbf{H} = [\mathbf{h}_1, \dots, \mathbf{h}_n]^\top$ (e.g. one per source word)
- 3 We compute **association scores** $s_1, \dots, s_n$ by “comparing” $\mathbf{q}$ and $\mathbf{H}$
- 4 We convert these scores to **probabilities**:

$$\mathbf{p} = \text{softmax}(\mathbf{s})$$

- 5 We use this to output a representation as a **weighted average**:

$$\mathbf{c} = \mathbf{H}^\top \mathbf{p} = \sum_{i=1}^n p_i \mathbf{h}_i$$

Let's take a closer look.

Association Scores

Several ways of “comparing” a query $\mathbf{q}$ and an input (“key”) vector $\mathbf{h}_i$:

- **Additive attention** (Bahdanau et al., 2015):

$$s_i = \mathbf{u}^\top \tanh(\mathbf{A}\mathbf{h}_i + \mathbf{B}\mathbf{q})$$

- **Bilinear attention** (Luong et al., 2015):

$$s_i = \mathbf{q}^\top \mathbf{U}\mathbf{h}_i$$

- **Dot product attention** (particular case of bilinear attention; queries and keys must have the same size):

$$s_i = \mathbf{q}^\top \mathbf{h}_i$$

The last two are easy to batch for multiple queries and multiple keys.

Keys and Values

The input vectors $\mathbf{H} = [\mathbf{h}_1, \dots, \mathbf{h}_n]^\top$ appear in two places:

- They are used as **keys** to “compare” them with the query vector $\mathbf{q}$ to obtain the association scores
- They are used as **values** to form the weighted average $\mathbf{c} = \mathbf{H}^\top \mathbf{p}$

Keys and Values

The input vectors $\mathbf{H} = [\mathbf{h}_1, \dots, \mathbf{h}_n]^\top$ appear in two places:

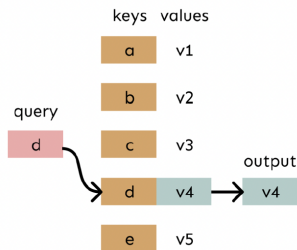
- They are used as **keys** to “compare” them with the query vector $\mathbf{q}$ to obtain the association scores
- They are used as **values** to form the weighted average $\mathbf{c} = \mathbf{H}^\top \mathbf{p}$

To be fully general, **they don't need to be the same**—we can have:

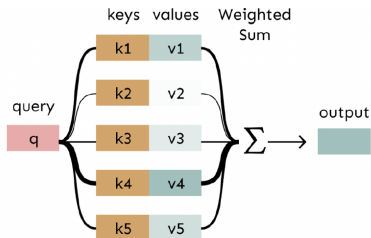
- A **key matrix** $\mathbf{K} = [\mathbf{k}_1, \dots, \mathbf{k}_n]^\top \in \mathbb{R}^{n \times d_K}$
- A **value matrix** $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_n]^\top \in \mathbb{R}^{n \times d_V}$

Attention as a Fuzzy Lookup Table

Lookup table: a table of **keys** that map to **values**. The **query** matches one of the keys, returning its value.



Attention: The **query** matches to all **keys** softly to a weight between 0 and 1. The keys' **values** are multiplied by the weights and summed.



(Credits: Danqi Chen)

Attention Mechanism: More General Version

- 1 We have a **query vector** $\mathbf{q}$ (e.g. the decoder state)
- 2 We have **key vectors** $\mathbf{K} = [\mathbf{k}_1, \dots, \mathbf{k}_n]^\top \in \mathbb{R}^{n \times d_K}$
and **value vectors** $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_n]^\top \in \mathbb{R}^{n \times d_V}$
(one of each per source word)
- 3 Compute **query-key association scores** $s_1, \dots, s_n$ comparing $\mathbf{q}$ and $\mathbf{K}$
- 4 We convert these scores to **probabilities**:

$$\mathbf{p} = \text{softmax}(\mathbf{s})$$

- 5 We output a weighted average of the **values**:

$$\mathbf{c} = \mathbf{V}^\top \mathbf{p} = \sum_{i=1}^n p_i \mathbf{v}_i \in \mathbb{R}^{d_V}$$

Next: Self-Attention

- So far we talked about **contextual attention**—the decoder attends to encoder states (this is called “input context”)
- Attention allows **focusing on an arbitrary position** in the source sentence, shortcutting the computation graph;
- But if attention gives access to any state...

... do we really need the RNN?

Next: Self-Attention

- So far we talked about **contextual attention**—the decoder attends to encoder states (this is called “input context”)
- Attention allows **focusing on an arbitrary position** in the source sentence, shortcutting the computation graph;
- But if attention gives access to any state...

... **do we really need the RNN?**

- Alternative: **self-attention**—at each position, the encoder attends to the other positions **in the encoder itself**
- Same for the decoder.

Self-Attention Layer

- 1 Query vectors $\mathbf{Q} = [\mathbf{q}_1, \dots, \mathbf{q}_n]^\top \in \mathbb{R}^{n \times d_Q}$
- 2 Key vectors $\mathbf{K} = [\mathbf{k}_1, \dots, \mathbf{k}_n]^\top \in \mathbb{R}^{n \times d_K}$
- 3 Value vectors $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_n]^\top \in \mathbb{R}^{n \times d_V}$

Self-Attention Layer

- 1 Query vectors $\mathbf{Q} = [\mathbf{q}_1, \dots, \mathbf{q}_n]^\top \in \mathbb{R}^{n \times d_Q}$
- 2 Key vectors $\mathbf{K} = [\mathbf{k}_1, \dots, \mathbf{k}_n]^\top \in \mathbb{R}^{n \times d_K}$
- 3 Value vectors $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_n]^\top \in \mathbb{R}^{n \times d_V}$
- 4 Compute query-key association scores “comparing” $\mathbf{Q}$ and $\mathbf{K}$, e.g.,

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^\top \in \mathbb{R}^{n \times n} \quad (\text{dot-product attention})$$

Self-Attention Layer

- 1 Query vectors $\mathbf{Q} = [\mathbf{q}_1, \dots, \mathbf{q}_n]^\top \in \mathbb{R}^{n \times d_Q}$
- 2 Key vectors $\mathbf{K} = [\mathbf{k}_1, \dots, \mathbf{k}_n]^\top \in \mathbb{R}^{n \times d_K}$
- 3 Value vectors $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_n]^\top \in \mathbb{R}^{n \times d_V}$
- 4 Compute query-key association scores “comparing” $\mathbf{Q}$ and $\mathbf{K}$, e.g.,

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^\top \in \mathbb{R}^{n \times n} \quad (\text{dot-product attention})$$

- 5 Convert these scores to probabilities (row-wise):

$$\mathbf{P} = \text{softmax}(\mathbf{S}) \in \mathbb{R}^{n \times n}$$

Self-Attention Layer

- 1 **Query vectors** $\mathbf{Q} = [\mathbf{q}_1, \dots, \mathbf{q}_n]^\top \in \mathbb{R}^{n \times d_Q}$
- 2 **Key vectors** $\mathbf{K} = [\mathbf{k}_1, \dots, \mathbf{k}_n]^\top \in \mathbb{R}^{n \times d_K}$
- 3 **Value vectors** $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_n]^\top \in \mathbb{R}^{n \times d_V}$
- 4 Compute **query-key** association scores “comparing” $\mathbf{Q}$ and $\mathbf{K}$, e.g.,

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^\top \in \mathbb{R}^{n \times n} \quad (\text{dot-product attention})$$

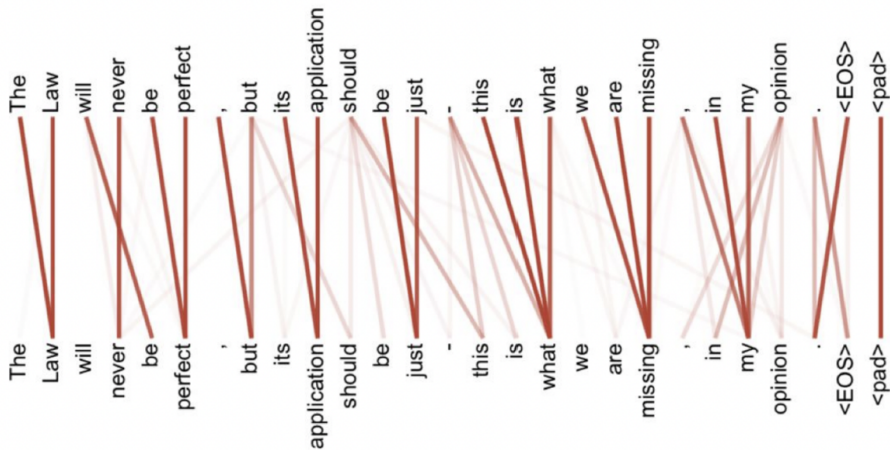
- 5 Convert these scores to **probabilities** (row-wise):

$$\mathbf{P} = \text{softmax}(\mathbf{S}) \in \mathbb{R}^{n \times n}$$

- 6 Output the weighted average of the **values**:

$$\mathbf{Z} = \mathbf{P}\mathbf{V} = \underbrace{\text{softmax}(\mathbf{Q}\mathbf{K}^\top)}_{\mathbf{P}} \mathbf{V} \in \mathbb{R}^{n \times d_V}.$$

Self-Attention



(Vaswani et al., "Attention is All You Need", NeurIPS 2017)

Outline

1 Transformers

- The Attention Mechanism
- The Transformer Architecture
- Handling Long Contexts
- What's Next?

2 Large Language Models

- The BERT Era: Pretraining and Fine-tuning
- The GPT-3 Era: Prompting and In-Context Learning
- The ChatGPT Era: Post-training and Preference Optimization
- The DeepSeek R1 Era: Reasoning
- What Next?

3 Conclusions

Transformer (Vaswani et al., 2017)

- **Key idea:** instead of RNNs, use **self-attention** in the encoder
- Each word state attends to **all the other words**
- Each self-attention is followed by a feed-forward transformation
- Do several layers of this
- Do the same for the decoder, attending only to already generated words.

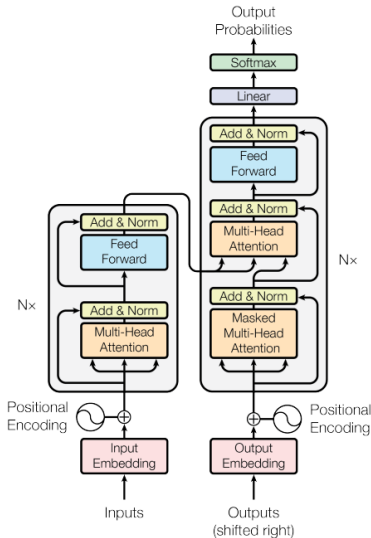
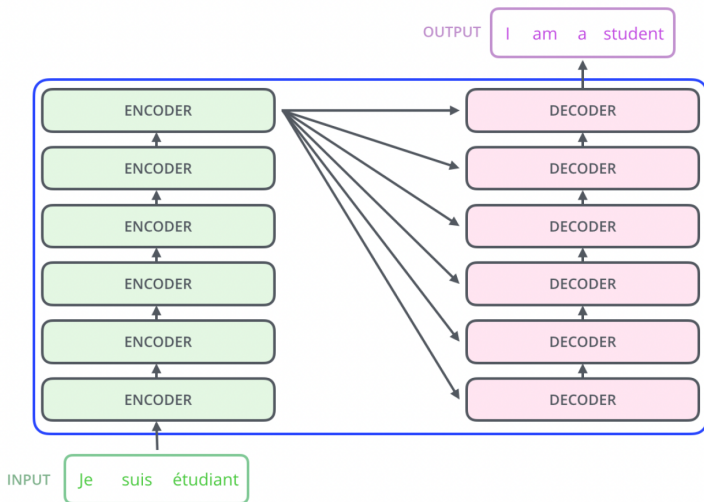


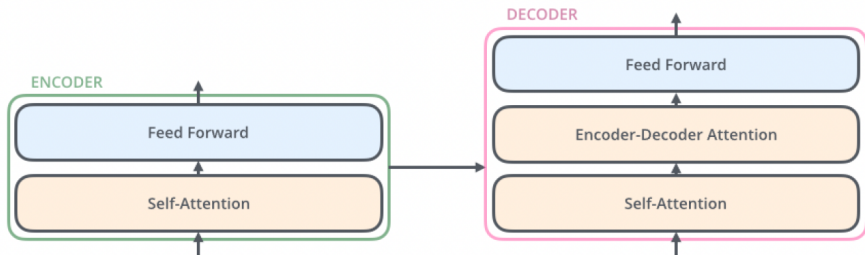
Figure 1: The Transformer - model architecture.

Transformer



Illustrated transformer: <http://jalammar.github.io/illustrated-transformer/>

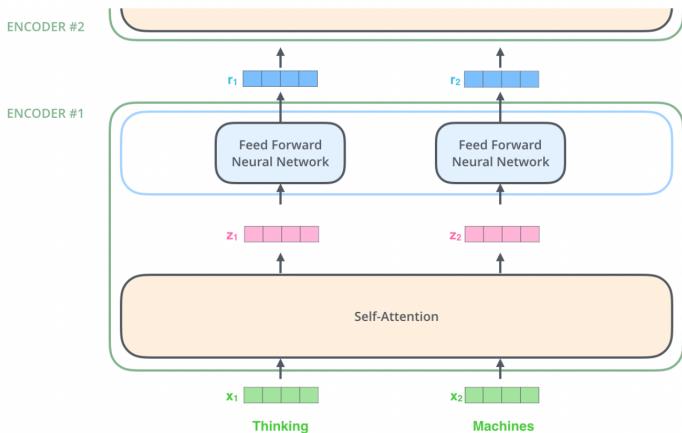
Transformer Blocks



Illustrated transformer: <http://jalammar.github.io/illustrated-transformer/>

The Encoder

Example for a sentence with 2 words:



Transformer Self-Attention: Queries, Keys, Vectors

- Obtained by projecting the embedding matrix $\mathbf{X} \in \mathbb{R}^{n \times e}$ to a lower dimension:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q$$

$$\mathbf{K} = \mathbf{X}\mathbf{W}^K$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}^V$$

- The projection matrices $\mathbf{W}^Q$, $\mathbf{W}^K$, $\mathbf{W}^V$ are model parameters.



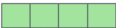
Transformer Self-Attention: Queries, Keys, Vectors

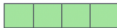
Input

Thinking

Machines

Embedding

x_1 

x_2 

Queries

q_1 

q_2 



W^Q

Keys

k_1 

k_2 

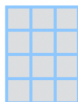


W^K

Values

v_1 

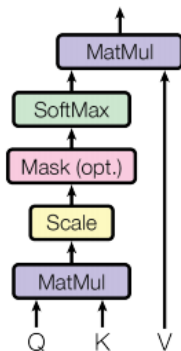
v_2 



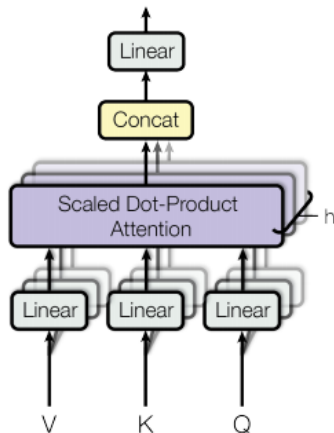
W^V

Next: Two Transformer Innovations

Scaled Dot-Product Attention



Multi-Head Attention

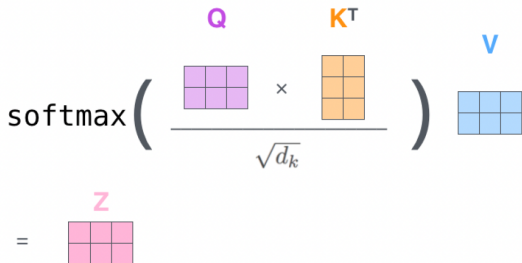


(Vaswani et al., "Attention is All You Need", NeurIPS 2017)

Scaled Dot-Product Attention

Problem: As d_K gets large, the variance of $\mathbf{q}^\top \mathbf{k}$ increases, the softmax gets very peaked, hence its gradient gets smaller.

Solution: scale by length of query/key vectors:



The diagram illustrates the Scaled Dot-Product Attention formula. It shows a purple 2x3 matrix labeled **Q** multiplied by an orange 3x2 matrix labeled **K^T**. The result of this multiplication is divided by the square root of the key dimension, $\sqrt{d_k}$. This entire expression is passed through a softmax function, represented by a large right parenthesis. To the right of the softmax is a blue 2x2 matrix labeled **V**. Below the softmax expression is an equals sign followed by a pink 2x3 matrix labeled **Z**.

$$\text{softmax}\left(\frac{\mathbf{Q} \times \mathbf{K}^\top}{\sqrt{d_k}}\right) \mathbf{V}$$
$$= \mathbf{Z}$$

Why $\sqrt{d}$?

For any pair of random variables X and Y and coefficients a and b :

$$\text{Var}(aX + bY) = a^2\text{Var}(X) + b^2\text{Var}(Y) + \underbrace{2ab\text{Cov}(X, Y)}_{0 \text{ if } X, Y \text{ are uncorrelated}}.$$

Therefore, if $(q_i k_i)_{i=1, \dots, d}$ were all uncorrelated with fixed variance σ^2 :

$$\text{Var}(\mathbf{q}^\top \mathbf{k}) = \text{Var}\left(\sum_{i=1}^d q_i k_i\right) = d\text{Var}(q_i k_i) = d\sigma^2.$$

Therefore, $\text{Var}\left(\frac{\mathbf{q}^\top \mathbf{k}}{\sqrt{d}}\right) = \sigma^2$: **variance does not increase with d .**

Multi-Head Attention

Self-attention: each word forms a **query vector** and attends to the other words' **key vectors**

Somewhat similar to a **1D convolution**, with “adaptive” weights and the window size spans the entire sentence

Problem: only one channel for words to interact with one-another

Multi-Head Attention

Self-attention: each word forms a **query vector** and attends to the other words' **key vectors**

Somewhat similar to a **1D convolution**, with “adaptive” weights and the window size spans the entire sentence

Problem: only one channel for words to interact with one-another

Solution: **multi-head attention!**

- define h attention heads, each with their own projection matrices (e.g. $h = 8$)
- apply attention in multiple channels, concatenate the outputs and pipe through linear layer via an **output projection matrix** W^O .

Multi-Head Attention

1) This is our input sentence*

Thinking
Machines

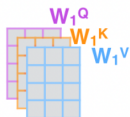
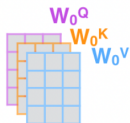
2) We embed each word*



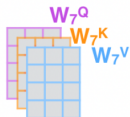
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



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



...



Illustrated transformer: <http://jalammr.github.io/illustrated-transformer/>

Other Tricks

- Self-attention blocks are repeated several times (e.g. 6 or 12)
- Positional encodings (to distinguish word positions)
- Residual connections on each attention block
- Layer normalization

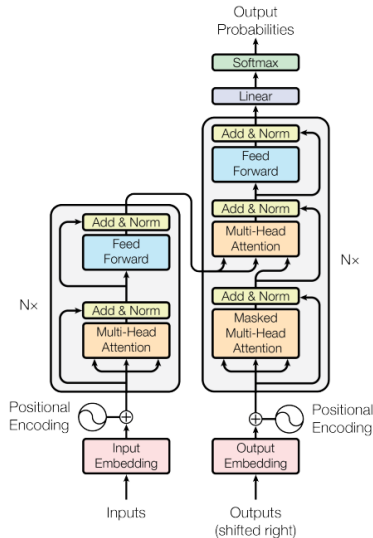
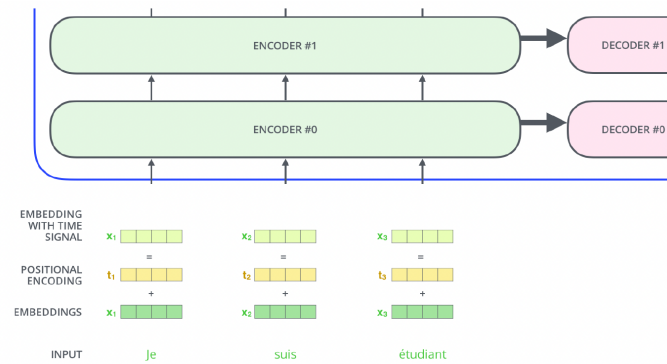


Figure 1: The Transformer - model architecture.

Positional Encodings

- As just described, the transformer encoder is **insensitive to word order!**
- Queries attend to keys regardless of their position in the sequence.
- To make it sensitive to order, we must add **positional encodings**.



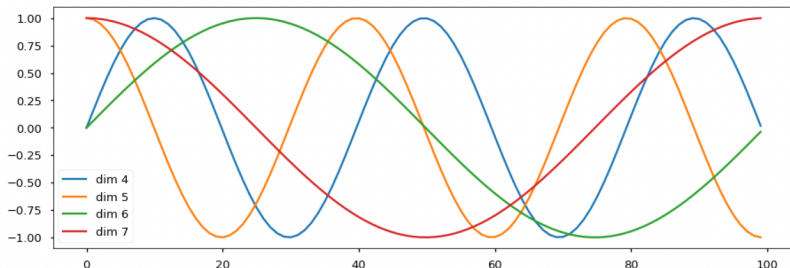
Positional Encodings

Several possible strategies:

- learn **one embedding for each position** (up to a maximum length)
- use **sinusoidal** positional encodings
- use **relative** positional encodings (instead of **absolute**).

Sinusoidal Positional Encodings

$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right) \quad PE_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$



Idea: create a unique **signature** for each position, but let it vary **smoothly** across nearby positions, using a mix of low and high frequencies.

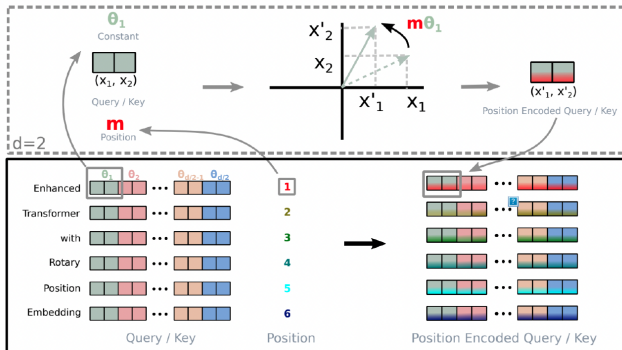
Relative Positional Encodings (Shaw et al., 2018)

Absolute positional encodings are hard to scale to arbitrary context lengths. Instead, **relative positional encodings** model the **distances** between pairs of tokens:

$$s_{ij} = \frac{\overbrace{x_i^\top W_Q}^{q_i} (\overbrace{x_j^\top W_K}^{k_j} + a_{ij})^\top}{\sqrt{d_K}}, \quad a_{ij} \text{ depends on the distance } |j - i|.$$

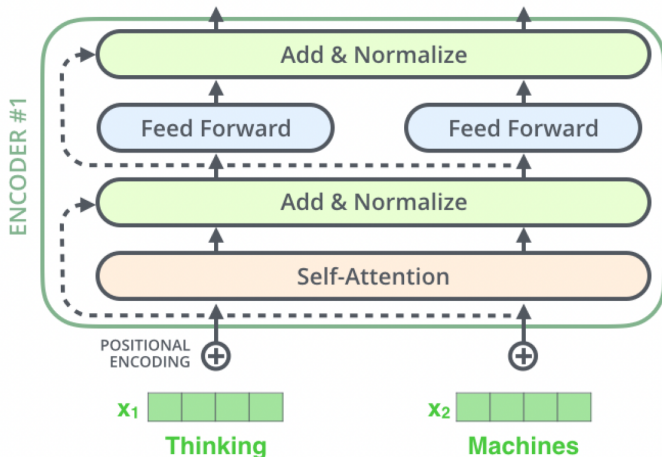
Rotary positional encodings (RoPE) combine both **absolute** and **relative** positional information (Su et al., 2024).

Rotary Positional Encodings (RoPE) (Su et al., 2024)

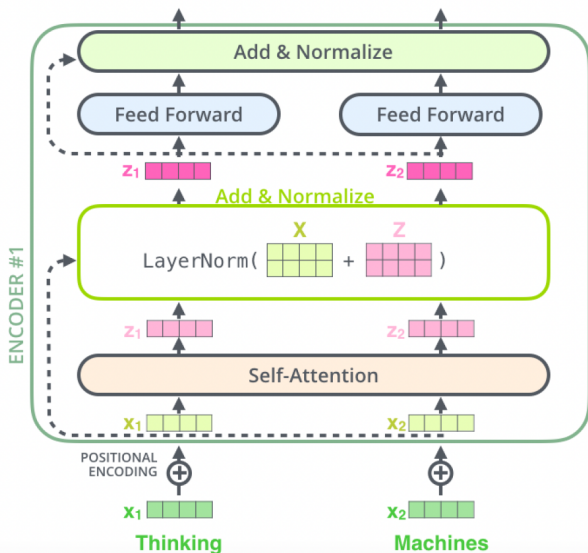


$$\begin{bmatrix} x'_{2i} \\ x'_{2i+1} \end{bmatrix} = \begin{bmatrix} \cos(m\theta_i) & -\sin(m\theta_i) \\ \sin(m\theta_i) & \cos(m\theta_i) \end{bmatrix} \begin{bmatrix} x_{2i} \\ x_{2i+1} \end{bmatrix}, \quad \begin{cases} m \text{ position} \\ i \text{ emb. index} \\ \theta_i = b^{-2i/d} \\ b = 10,000. \end{cases}$$

Residuals and Layer Normalization



Residuals and Layer Normalization



Residual Connections

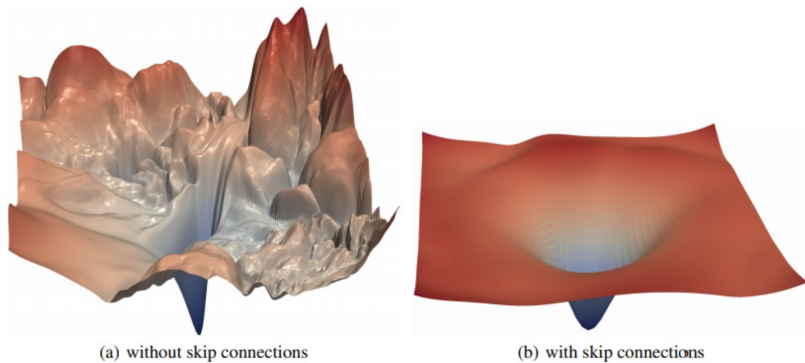


Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

(Li et al., 2018)

Originally proposed for vision in ResNets; similar ideas go back to Sepp Hochreiter's PhD thesis in 1991 (He et al., 2016; Hochreiter, 1991).

Layer Normalization (Ba et al., 2016)

Idea: remove uninformative variation in hidden layers $\mathbf{h} \in \mathbb{R}^d$ by normalizing to unit mean μ and standard deviation σ **within each layer**.

$$\text{LayerNorm}(\mathbf{h}) = \frac{\mathbf{h} - \mu \mathbf{1}}{\sqrt{\sigma^2 + \epsilon}} \odot \boldsymbol{\gamma} + \boldsymbol{\beta}$$

where

- $\mu = \frac{1}{d} \sum_{i=1}^d h_i$
- $\sigma^2 = \frac{1}{d} \sum_{i=1}^d (h_i - \mu)^2$
- ϵ is a small coefficient (e.g. 1×10^{-5})
- $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learnable weights and biases (default $\boldsymbol{\gamma} = \mathbf{1}$ and $\boldsymbol{\beta} = \mathbf{0}$).

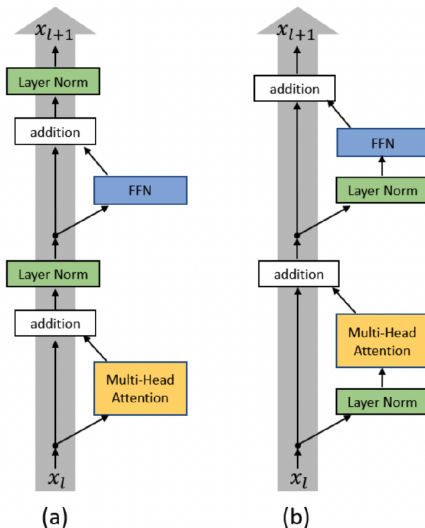
This helps accelerate convergence.

More recently: **RMSNorm** removes the mean subtraction, setting $\mu = 0$ (Zhang and Sennrich, 2019).

Residual Stream

Different implementations order these operations differently:

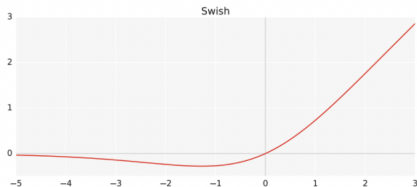
- Left: **post-normalization**
- Right: **pre-normalization**.



(Xiong et al., 2020)

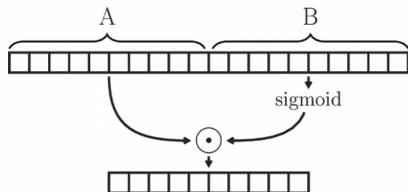
Replacing ReLU activations by SwiGLU

$$\text{SwiGLU}(x, W, V, b, c, \beta) = \text{Swish}_{\beta}(xW + b) \otimes (xV + c)$$



$$\text{Swish}(x) = x \cdot \text{sigmoid}(\beta x)$$

(Shazeer, 2020)



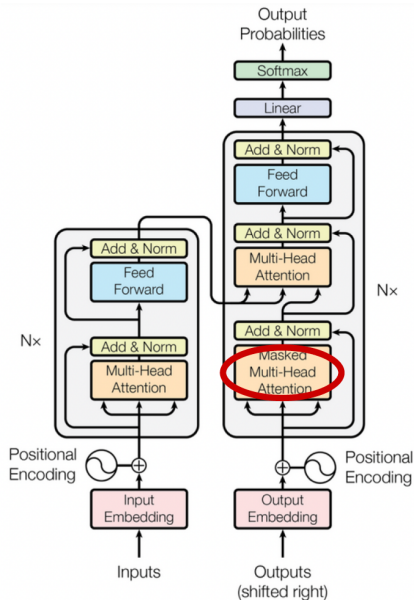
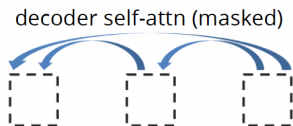
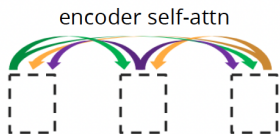
The Decoder

What about the self-attention blocks in the **decoder**?

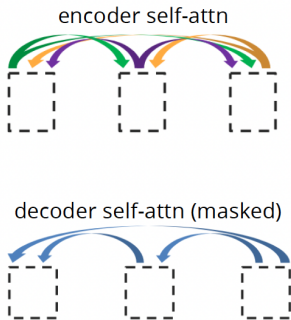
Everything is pretty much the same as in the **encoder**, with two twists:

- The decoder cannot see the future! Use “**causal**” **masking**.
- The decoder should attend to itself (**self-attention**), but also to the encoder states (**contextual attention**).

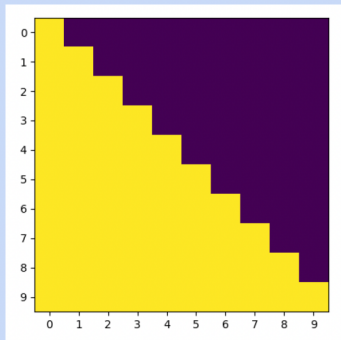
The Decoder



The Decoder



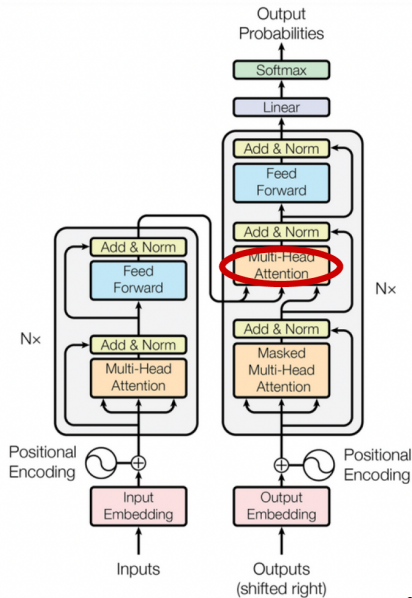
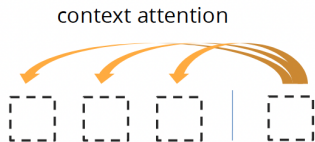
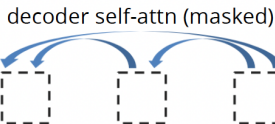
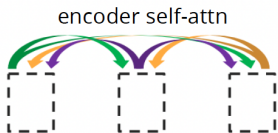
- Mask subsequent positions (before softmax)



- In PyTorch

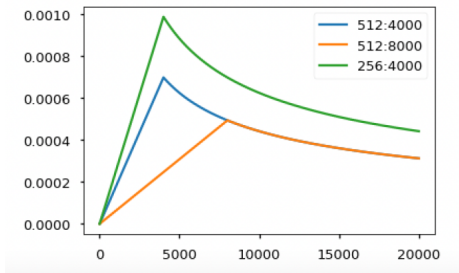
```
scores.masked_fill_(~mask, float('-inf'))
```


The Decoder



Optimizer and Scheduler

Adam optimizer with warmup steps and learning-rate decay:



More recently: Muon optimizer (Jordan et al., 2024)

Overall, transformers are **harder** to optimize than RNN seq2seq models

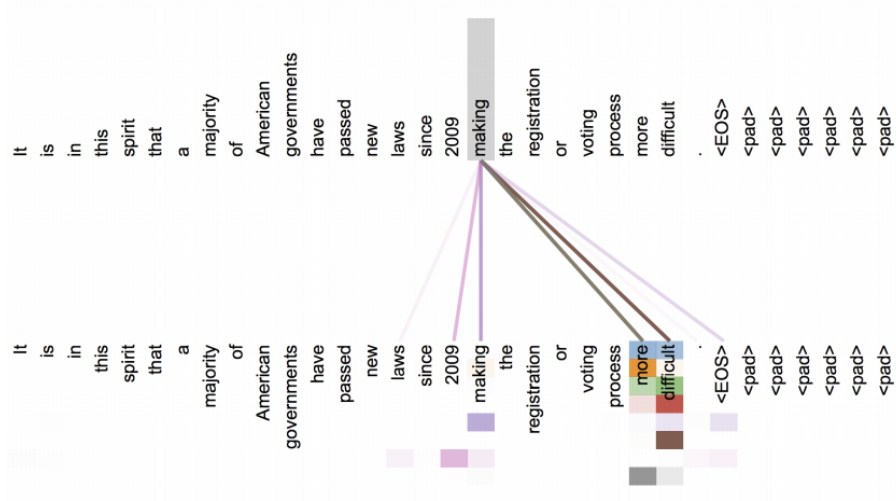
They don't work out of the box: hyperparameter tuning is very important.

Transformer Results

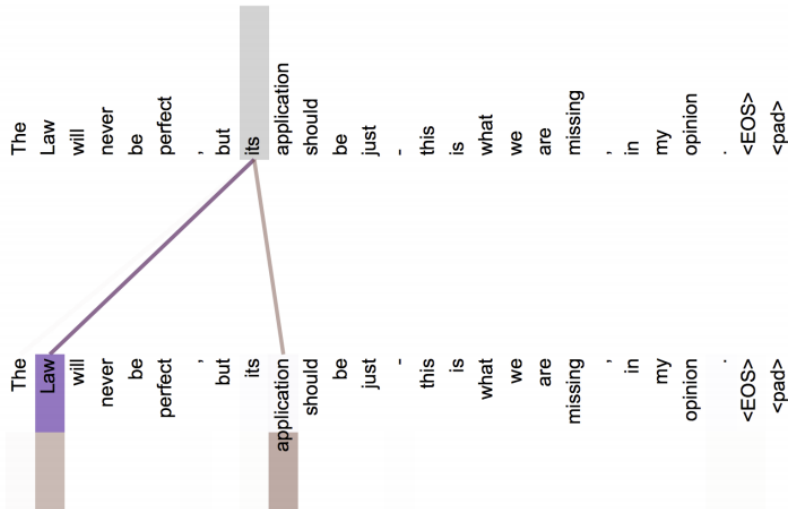
Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	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	41.8	$2.3 \cdot 10^{19}$	

(Vaswani et al., “Attention is All You Need”, NeurIPS 2017)

Attention Visualization Layer 5



Implicit Anaphora Resolution



Computational Cost

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

n = seq. length d = hidden dim k = kernel size

- **Faster to train** (due to self-attention parallelization)
- ... but **more expensive to decode**
- Scale **quadratically** with respect to sequence length (problematic for long sequences).

Outline

1 Transformers

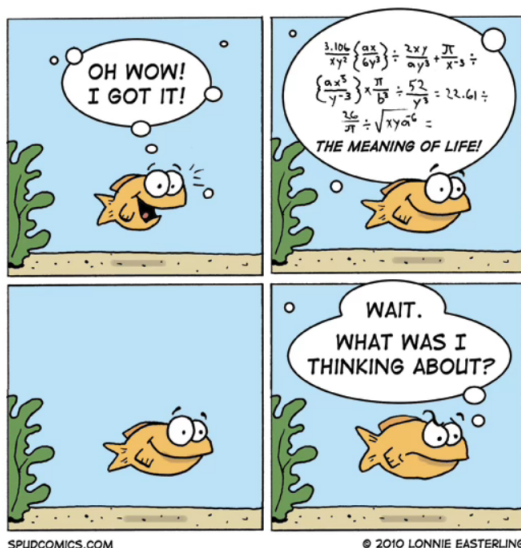
- The Attention Mechanism
- The Transformer Architecture
- Handling Long Contexts
- What's Next?

2 Large Language Models

- The BERT Era: Pretraining and Fine-tuning
- The GPT-3 Era: Prompting and In-Context Learning
- The ChatGPT Era: Post-training and Preference Optimization
- The DeepSeek R1 Era: Reasoning
- What Next?

3 Conclusions

Long-Term Memory



THE TRAGEDY OF A THREE SECOND MEMORY

Long Contexts Are Increasingly Important

Several applications require long-context windows (e.g. 32K–1M):

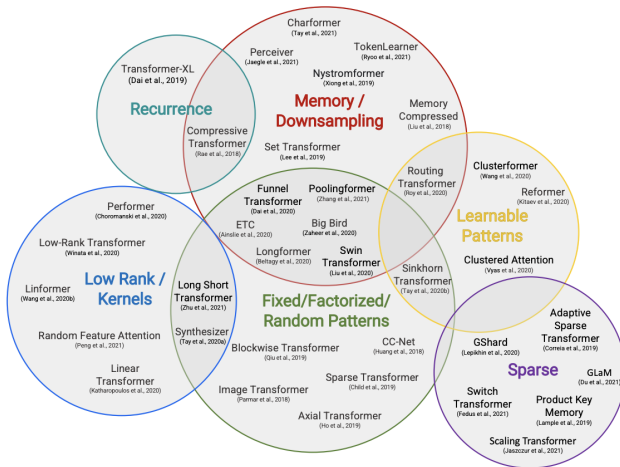
- Software engineering tasks that demand understanding the entire repository
- Solving extremely challenging math problems that require elaborate sequences of trial and error across different problem-solving strategies
- AI for scientific discovery
- Multi-turn personalized chat interactions conditioned on prolonged interaction histories
- Multimodal data (e.g. long videos)
- ...

The **quadratic** complexity of transformers is a bottleneck for these tasks...

Typical Document Lengths

Source Type	Typical Length (#toks)	Description
Common Crawl (Web)	600–1,200	Includes blog posts, news, and landing pages. Many “documents” are highly fragmented.
Wikipedia	500–1,500	While some entries are long, the median article is relatively concise.
Scientific Papers (arXiv)	5,000–10,000+	These are among the longest “natural” documents in most sets.
Books (Project Gutenberg)	50,000–100,000+	The “long-tail” of the data; rare but critical for long-range dependency.
GitHub (Code)	100–5,000	Code files vary wildly; many scripts are quite short, while libraries are long.

Efficient Transformers

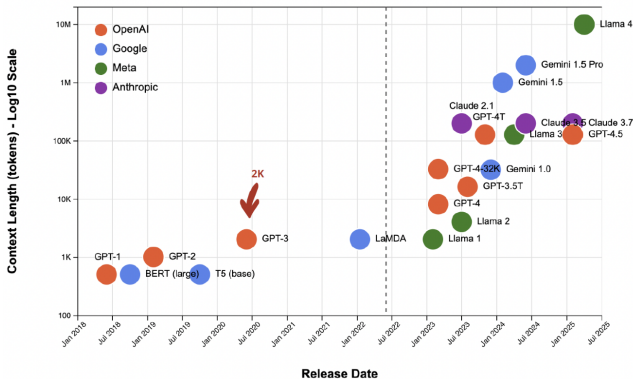


(Tay et al., “Efficient transformers: A survey”, ACM Computing Surveys, 2022)

Some Proposed Solutions

- GPU-aware kernel implementations to reduce memory from $O(n^2)$ to $O(n)$ (e.g. **FlashAttention**).
- **Sparse / sliding window attention** combined with global attention tokens (e.g. Longformer).
- **Grouped Query Attention** (GQA) or Multi-Query Attention (MQA) to reduce KV cache memory.
- **Hybrid architectures**: use a fraction of self-attention layers, and a fraction of recurrent layers like Mamba or Gated Deltanet).
- **Dynamic sparsity** (sparsemax, α -entmax) using AdaSplash kernels (Correia et al., 2019; Gonçalves et al., 2025; Gonçalves et al., 2026)
- **Hierarchical attention** (Yuan et al., 2025; Huang et al., 2026).

FlashAttention (Dao et al., 2022)

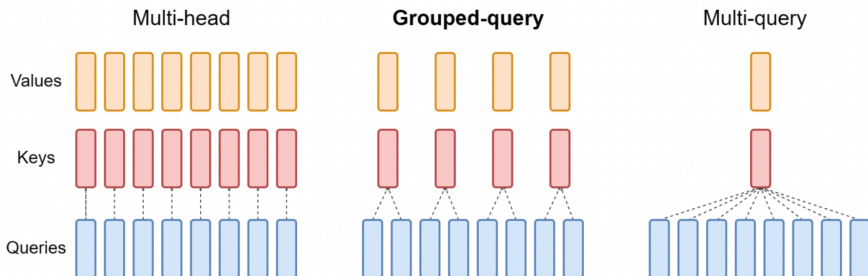


Compute **exact** attention leveraging the fast memory of the GPU.

Reduces memory from $O(n^2)$ to $O(n)$, but runtime is **still quadratic**.

The main enabler for the exponential increase in context length after 2023!

Grouped Query Attention (GQA)



(Ainslie et al., 2023)

Let queries **share** some keys/values to reduce KV cache size.

Linear Transformers

Recall the attention expression:

$$\mathbf{Z} = \text{softmax}(\mathbf{Q}\mathbf{K}^\top)\mathbf{V}$$

What happens if we **remove** the softmax transformation? By the associative property,

$$\mathbf{Z} = (\mathbf{Q}\mathbf{K}^\top)\mathbf{V} = \mathbf{Q}(\mathbf{K}^\top\mathbf{V}).$$

For long-context $n \gg d$, this avoids materializing the n -by- n matrix $\mathbf{Q}\mathbf{K}^\top$. The computational cost becomes $\mathcal{O}(d^2n)$ instead of $\mathcal{O}(dn^2)$.

■ **Linear** instead of **quadratic**.

Related: **fast weight programmers** (Schmidhuber, 1992; Schlag et al., 2021).

Linear Transformers are RNNs!

In the decoder, due to causal masking, we have

$$z_i = \underbrace{\left(\sum_{j \leq i} k_j v_j^\top \right)}_{:= \mathbf{H}_i} \mathbf{q}_i,$$

which is equivalent to the following updates (starting with $\mathbf{H}_0 = 0$):

$$\begin{aligned} \mathbf{H}_i &= \mathbf{H}_{i-1} + k_i v_i^\top \\ z_i &= \mathbf{H}_i \mathbf{q}_i. \end{aligned}$$

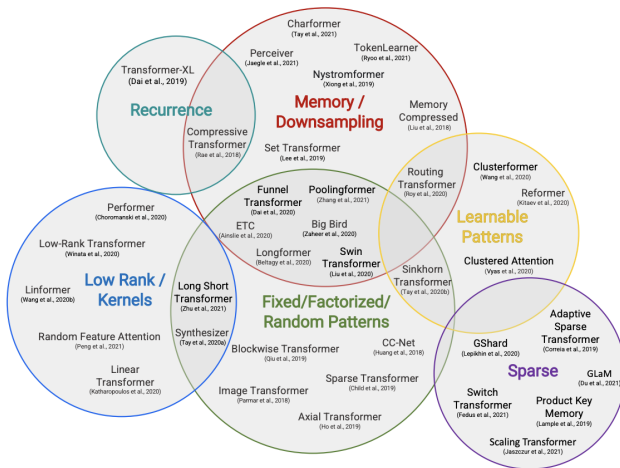
We can interpret $\mathbf{H}_1, \dots, \mathbf{H}_L$ as the states of a **linear RNN**!

Related: state space models, Mamba, RWKV, Gated DeltaNet, ...

- Many recent architectures (Qwen3.5, Kimi K3, Nemotron 3, ...) are **hybrids** interleaving full attention and linear layers.

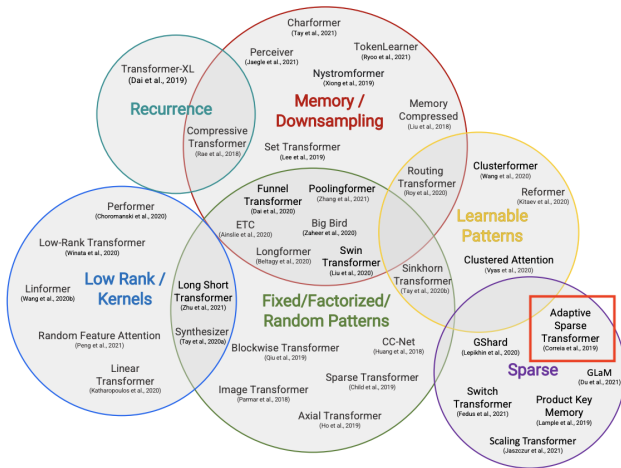
(Katharopoulos et al., “Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention”,
ICML 2020)

Efficient Transformers



(Tay et al., “Efficient transformers: A survey”, ACM Computing Surveys, 2022)

Efficient Transformers

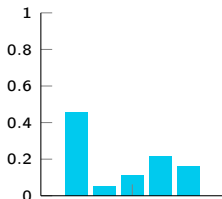


(Tay et al., “Efficient transformers: A survey”, ACM Computing Surveys, 2022)

Adaptive Sparsity through α -entmax (Peters et al., 2019)

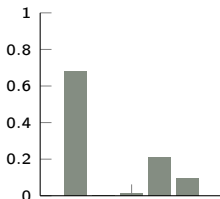
$\alpha = 1$

softmax($\mathbf{s}$)



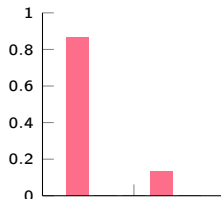
$\alpha = 1.5$

1.5-entmax($\mathbf{s}$)



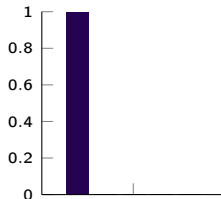
$\alpha = 2$

sparsemax($\mathbf{s}$)



$\alpha = \infty$

argmax($\mathbf{s}$)



(Same $\mathbf{s} = [1.0716, -1.1221, -0.3288, 0.3368, 0.0425]$)

$$\alpha\text{-entmax}(\mathbf{s}) = [(\alpha - 1)\mathbf{s} - \tau]_+^{1/(\alpha-1)}$$

Adaptively Sparse Transformers

Replaces **softmax** attention by α -entmax.

$\alpha > 1$ can return **sparse** attention probabilities $\rightarrow$ **adaptive sparse attention**.

... so the model learns to **ignore** irrelevant tokens.



Marcos Treviso will talk more about this tomorrow afternoon!

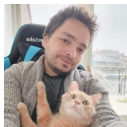
Peters, Niculae, and Martins, "Sparse Sequence-to-Sequence Models", ACL 2019

Correia, Niculae, and Martins, "Adaptively Sparse Transformers", EMNLP 2019

Efficient Implementation: AdaSplash

- The equivalent of FlashAttention to α -entmax.

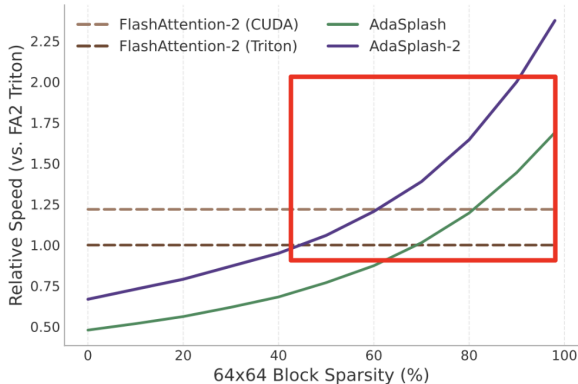
```
1 # pip install adasplash
2 from adasplash import adasplash
3
4 y = adasplash(q, k, v, alpha=1.5, niter=3, is_causal=True)
```



Gonçães, Treviso, and Martins, “AdaSplash: Adaptive Sparse Flash Attention”, ICML 2025

Gonçães, Pitorro, Niculae, Ponti, Li, Martins, and Treviso, “AdaSplash-2: Faster Differentiable Sparse Attention”, ICML 2026

Efficient Implementation: AdaSplash



AdaSplash scales better than FlashAttention with context size!

Gonçalves, Treviso, and Martins, “AdaSplash: Adaptive Sparse Flash Attention”, ICML 2025

Gonçalves, Pitorro, Niculae, Ponti, Li, Martins, and Treviso, “AdaSplash-2: Faster Differentiable Sparse Attention”, ICML 2026

... Also Better Length Generalization!

Model	S-NIAH-1 (ID)		S-NIAH-1 (OOD)			S-NIAH-2 (ID)		S-NIAH-2 (OOD)	
	1K	2K	4K	8K	16K	1K	2K	4K	8K
Softmax	100.0	99.4	94.2	11.4	0.8	100.0	100.0	4.8	0.0
SSMax	100.0	99.8	99.2	92.0	75.2	99.4	99.2	64.4	14.8
Entmax	99.8	99.8	89.0	21.6	1.2	99.6	99.4	64.8	7.2
ASEntmax	99.6	100.0	100.0	99.8	97.4	99.4	99.4	83.2	25.4

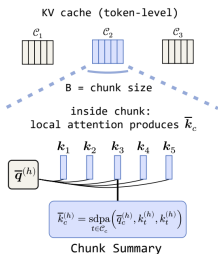
Better
long-context
generalization!



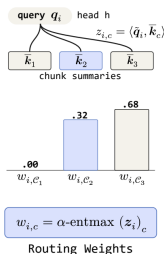
Vasylenko, Pitorro, Martins, and Treviso,
“Long-Context Generalization with Sparse Attention”, ICLR 2026

New: Hierarchical Memory through DashAttention

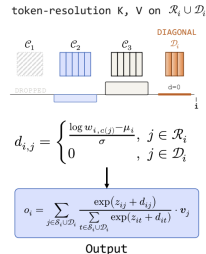
STAGE 0 LOCAL CHUNK SUMMARIZATION



STAGE 1 ENTMAX BLOCK ROUTING



STAGE 2 PRIOR-INDUCED SPARSE SOFTMAX



Huang, Gonçalves, Alvetreti, Li, Han, Ponti, Martins, and Treviso, "DashAttention: Differentiable and Adaptive Sparse Hierarchical Attention", 2026 (arxiv)

Continuous Attention: ∞ -former

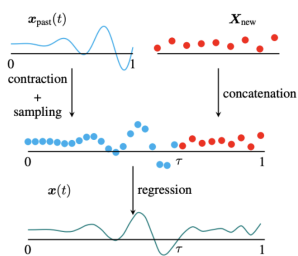
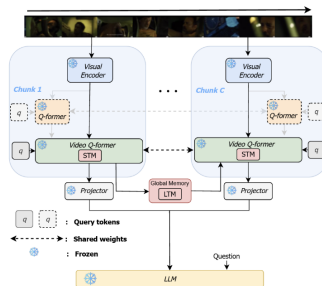


Figure 2. Proposed Memory Consolidation Mechanism.



Martins, Marinho, and Martins, “ ∞ -former: Infinite Memory Transformer”, ACL 2022

Santos, Farinhas, McNamee, and Martins, “ ∞ -Video: A Training-Free Approach to Long Video

Understanding via Continuous-Time Memory Consolidation”, ICML 2025

Outline

1 Transformers

- The Attention Mechanism
- The Transformer Architecture
- Handling Long Contexts
- What's Next?

2 Large Language Models

- The BERT Era: Pretraining and Fine-tuning
- The GPT-3 Era: Prompting and In-Context Learning
- The ChatGPT Era: Post-training and Preference Optimization
- The DeepSeek R1 Era: Reasoning
- What Next?

3 Conclusions

Are We Still Going to Use Transformers in 2027?

Is Attention All You Need?



Current Status: Yes

Time Remaining: 162d 21h 48m 53s

Proposition:

On January 1, 2027, a Transformer-like model will continue to hold the state-of-the-art position in most benchmarked tasks in natural language processing.

For the Motion

Jonathan Frankle
@jefrankle
Harvard Professor
Chief Scientist Mosaic ML



Against the Motion

Sasha Rush
@srush_nlp
Cornell Professor
Research Scientist Hugging Face 🤗



<https://www.isattentionallyouneed.com/>

What Are Transformers Are Doing?

If we understand what transformers do, perhaps we can modify them and come up with better architectures.

I started the talk mentioning **associative memories**.

We saw transformers can be seen as (nonparametric) **“modern” Hopfield networks**.

Another perspective: transformers are fast in-context learners through test-time **kernel regression**.

Attention as Kernel Regression (Wang et al., 2025; Zhang et al., 2025)

Let's assume queries = keys, and assume they are normalized ($\|k_i\| = 1$)

The “data” (memory) is the **key-value cache**

$$\mathcal{M}_{<t} := \{(k_1, v_1), \dots, (k_{t-1}, v_{t-1})\}.$$

Given a **new query** (key) $q = k_t$, we want to predict

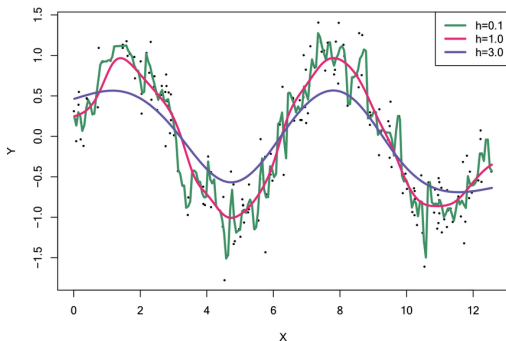
$$\hat{v}_t := \mathbb{E}[v_t | k_t, \mathcal{M}_{<t}].$$

That is, use the **KV cache** as “data” to predict the **current value** $\hat{v}_t$ from the **current key** k_t .

This is a regression problem!

Nadaraya-Watson Estimator (Nadaraya, 1964; Watson, 1964)

- An instance of **non-parametric** kernel regression
- Define a smoothing kernel function with **bandwidth** h , e.g. a Gaussian kernel $K_h(\mathbf{u}) := \mathcal{N}(\mathbf{u}; 0, h^2 \mathbf{I}) \propto \exp(-\|\mathbf{u}\|^2/h^2)$.
- Given a new input $\mathbf{q} = \mathbf{k}_t$, estimate $\hat{\mathbf{v}}_t := \frac{\sum_{i < t} K_h(\mathbf{q} - \mathbf{k}_i) \mathbf{v}_i}{\sum_{i < t} K_h(\mathbf{q} - \mathbf{k}_i)}$.



Nadaraya-Watson Estimator (Nadaraya, 1964; Watson, 1964)

With a **Gaussian kernel** $K_h(\mathbf{u}) \propto \exp(-\|\mathbf{u}\|^2/h^2)$ and normalized keys:

$$\begin{aligned}\hat{\mathbf{v}}_t &= \frac{\sum_{i < t} K_h(\mathbf{q} - \mathbf{k}_i) \mathbf{v}_i}{\sum_{i < t} K_h(\mathbf{q} - \mathbf{k}_i)} \\&= \frac{\sum_{i < t} \exp(-\|\mathbf{q} - \mathbf{k}_i\|^2/h^2) \mathbf{v}_i}{\sum_{i < t} \exp(-\|\mathbf{q} - \mathbf{k}_i\|^2/h^2)} \\&= \frac{\sum_{i < t} \exp(2\mathbf{q}^\top \mathbf{k}_i/h^2 - \cancel{2/h^2}) \mathbf{v}_i}{\sum_{i < t} \exp(2\mathbf{q}^\top \mathbf{k}_i/h^2 - \cancel{2/h^2})} \\&= \sum_{i < t} \text{softmax}(2\mathbf{K}\mathbf{q}/h^2)_i \mathbf{v}_i\end{aligned}$$

This is **softmax self-attention** with temperature $h^2/2$.

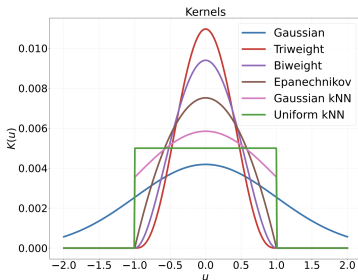
That is, **transformers are (roughly) doing test-time regression with a Gaussian kernel**.

Alternatives to Gaussian Kernel

The Gaussian kernel (= softmax attention) has full support.

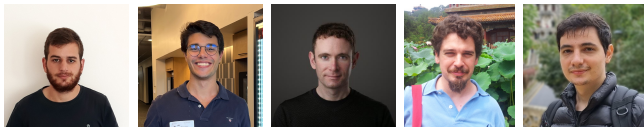
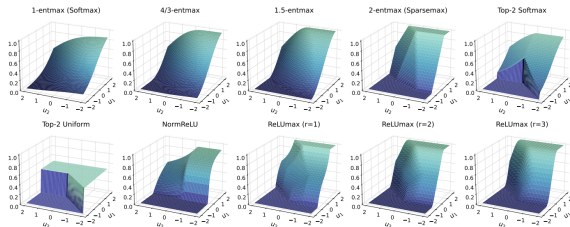
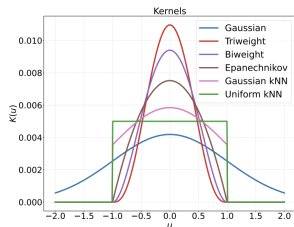
That is, it reads **all the memory** for generating each token. Do we need this?

Nonparametric statistics favor other kernels with **compact** support (e.g. Epanechnikov, biweight, triweight, ...)



If we replace the Gaussian kernel by such kernels, we obtain **dynamic sparse α -entmax attention!**

Compact Kernel Regression

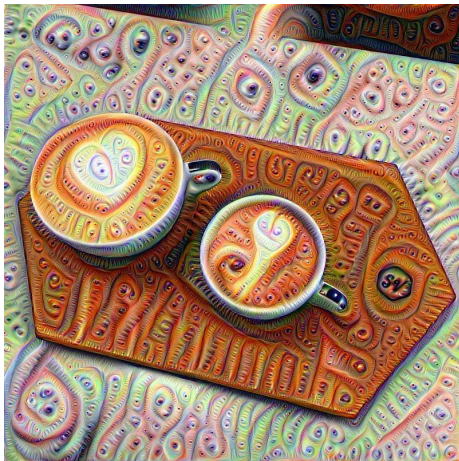


Santos, Gonçalves, McNamee, Treviso, and Martins,
“Sparse Attention as Compact Kernel Regression”, 2026 (arxiv)

Open Research Directions

- Are recurrent models coming back? Mamba, Gated DeltaNet, etc.
- Current trend is to hybridize self-attention layers with some of these alternatives... at which point will this cease to be a “transformer”?
- Two related perspectives:
 - **Compress** the KV cache in transformers to avoid quadratic growth
 - **Expand** state size in recurrent models to increase memory capacity
- It's insightful to think about the memory properties we want
- Lots of recent work building upon this perspective (memory mosaics, Titans, MIRAS, ...)
- Also hierarchy, dynamic sparsity, MoEs, ...
 - Check out our NeurIPS'24 tutorial on this topic:
<https://dynamic-sparsity.github.io/>

Coffee Break!



Outline

1 Transformers

- The Attention Mechanism
- The Transformer Architecture
- Handling Long Contexts
- What's Next?

2 Large Language Models

- The BERT Era: Pretraining and Fine-tuning
- The GPT-3 Era: Prompting and In-Context Learning
- The ChatGPT Era: Post-training and Preference Optimization
- The DeepSeek R1 Era: Reasoning
- What Next?

3 Conclusions

Outline

1 Transformers

- The Attention Mechanism
- The Transformer Architecture
- Handling Long Contexts
- What's Next?

2 Large Language Models

- The BERT Era: Pretraining and Fine-tuning
- The GPT-3 Era: Prompting and In-Context Learning
- The ChatGPT Era: Post-training and Preference Optimization
- The DeepSeek R1 Era: Reasoning
- What Next?

3 Conclusions

Pretraining through Next Word Prediction

Recall the **language modeling** (LM) task you learned in Noah's lecture:

- Model $p_{\theta}(y_t \mid y_{1:(t-1)})$, the probability distribution of words given their past contexts.
- There's lots of data for this! No **labels** are necessary, just **raw text**.
- This is called **unsupervised pretraining** or **self-supervised learning**.

Self-Supervised Learning

Pretraining on LM task is a form of self-supervised learning.

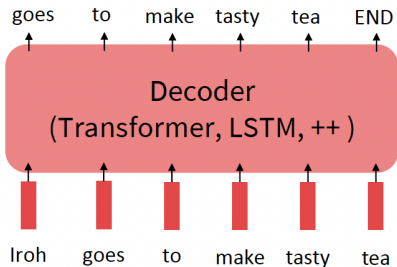
- Take raw (unlabeled) data, remove information and train a model to recover that information
- In the case of language modeling, the information removed is the next word; the model is trained to predict future words given the context
- Other strategies: mask words (later)
- This can be done with other data modalities too (e.g. speech, images), not just text
- For example, take images, obfuscate a region, and train a model to predict the missing region (image completion).

Pretraining and Fine-tuning

Pretraining can be very effective by serving as parameter initialization.

Step 1: Pretrain (e.g. on LM)

Lots of text; learn general things!

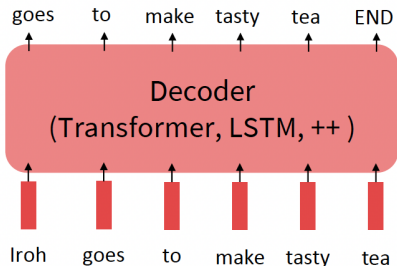


Pretraining and Fine-tuning

Pretraining can be very effective by serving as parameter initialization.

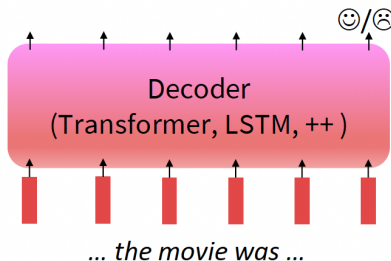
Step 1: Pretrain (e.g. on LM)

Lots of text; learn general things!

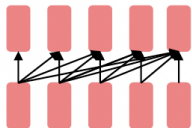


Step 2: Finetune on your task

Not many labels; adapt to the task!

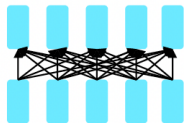


Three Architectures for Pretraining



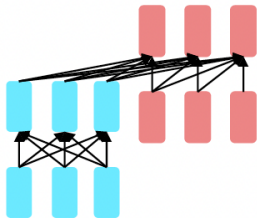
Decoder-only

- Language models with causal masking.
- Nice to generate from; but can't condition on future words.



Encoder-only

- Bidirectional context $\Rightarrow$ can condition on future!
- Wait, how do we pretrain them?



Encoder-Decoder

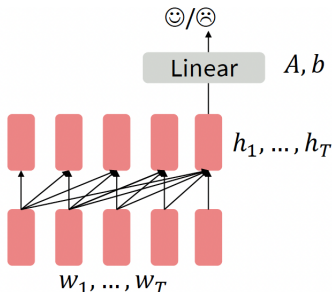
- Good parts of decoders and encoders?
- What's the best way to pretrain them?

Pretrained Decoders

- When using LM-pretrained decoders, we can ignore that they were trained to model $p_{\theta}(x_t \mid x_{1:(t-1)})$
- Fine-tuning by training a classifier on the last hidden state.

$$\begin{aligned}h_1, \dots, h_L &= \text{Decoder}(x_1, \dots, x_L) \\ y &= \text{softmax}(\mathbf{A}h_L + \mathbf{b})\end{aligned}$$

where $\mathbf{A}$ and $\mathbf{b}$ are randomly initialized and learned by the downstream task.



Pretrained Decoders

Two common choices for **fine-tuning**:

- **Freeze** the pretrained model and train only **A** and **b**
- **Finetune everything**, backpropagate gradients through the whole network.

Pretrained Decoders

Two common choices for **fine-tuning**:

- **Freeze** the pretrained model and train only **A** and **b**
- **Finetune everything**, backpropagate gradients through the whole network.

Pretrained decoders can be finetuned directly for **generation** tasks.

The family of **GPT** models developed by OpenAI is an example.

Generative Pretrained Transformer (GPT)

(Radford et al., 2018)

- 2018's GPT was a big success in pretraining a decoder!
- Transformer decoder with 12 layers.
- 768-D hidden states, 3072-D feed forward hidden layers.
- Byte pair encoding with 40,000 merges (vocabulary size)
- Trained on BooksCorpus over 7000 books.
- Contains long spans of contiguous text, for learning long distance dependencies.

Another iteration of GPT, with larger models and trained on much more data (40GB of Internet text)

Stronger in language generation tasks.

- A transformer-based language model with 1.5 billion parameters, trained on 8 million web pages
- Training task: predict the next word, given all of the previous words
- Test time: top- k truncated random sampling
- Lots of hype!

GPT-2's Famous Example (Radford et al., 2019)

Given a prompt, perform **story completion**:

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

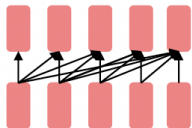
GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

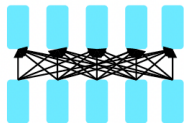
Later: GPT-3, InstructGPT, ChatGPT, ...

Three Architectures for Pretraining



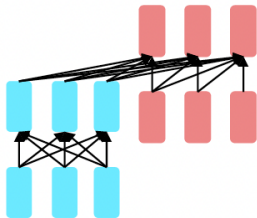
Decoder-only ✓

- Language models with causal masking.
- Nice to generate from; but can't condition on future words.



Encoder-only

- Bidirectional context $\Rightarrow$ can condition on future!
- Wait, how do we pretrain them?



Encoder-Decoder

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

Pretrained Encoders

So far, we've looked at language model pretraining. But encoders get **bidirectional context**, so we can't do language modeling!

So, what pretraining objective to use?

Pretrained Encoders

So far, we've looked at language model pretraining. But encoders get **bidirectional context**, so we can't do language modeling!

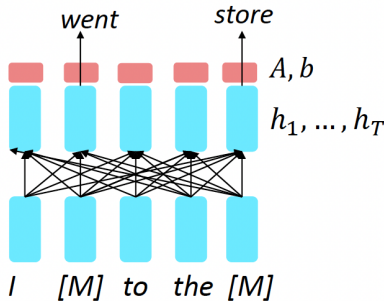
So, what pretraining objective to use? **Masked language modeling**.

Masked Language Modeling

- Idea: replace a fraction of words in the input with a special [MASK] token; predict these words.

$$\mathbf{h}_1, \dots, \mathbf{h}_L = \text{Encoder}(x_1, \dots, x_L)$$
$$\mathbf{y}_i = \text{softmax}(\mathbf{A}\mathbf{h}_i + \mathbf{b}).$$

- Loss terms only from masked word.
If $\tilde{x}$ is the masked version of x ,
we're learning $p_\theta(x|\tilde{x})$
- Analogous to a **denoising auto-encoder**.



(Devlin et al., 2018)

Example: BERT (Devlin et al., 2018)

“Bidirectional Encoder Representations from Transformers”

- Randomly mask 15% of the input words and train a **transformer** to recover them from the context
- Both left and right context, used simultaneously!
- In doing so, learn **contextualized word representations**
- Can use this as a pre-trained model and fine-tune it to any downstream task
- Extremely effective! Achieved SOTA on 11 NLP tasks (7.7% percentage point improvement on GLUE score).



Details about BERT (Devlin et al., 2018)

Two models were released:

- BERT base: 12 layers, 768 dim hidden states, 12 attention heads, 110 million params.
- BERT large: 24 layers, 1024 dim hidden states, 16 attention heads, 340 million params.

Trained on:

- BooksCorpus (800 million words)
- English Wikipedia (2.5 billion words)

Pretraining is expensive and impractical on a single GPU.

- BERT was pretrained with 64 TPU chips for a total of 4 days.

Fine-tuning is practical and common on a single GPU:

- “Pretrain once, finetune many times.”

Finetuning BERT (Devlin et al., 2018)

BERT became very popular and versatile; finetuning BERT led to new state of the art results on a broad range of NLP tasks:

- Paraphrase detection (QQP, MRPC)
- Natural language inference (QNLI, RTE)
- Sentiment analysis (SST-2)
- Grammatical correctness (CoLA)
- Semantic textual similarity (STS-B)

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	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT _{BASE}	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{LARGE}	86.7/85.9	72.1	92.7	94.9	60.5	86.5	89.3	70.1	82.1

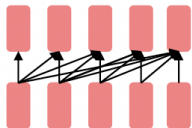
Other variants of BERT

- RoBERTA (similar to BERT, but trained on more data and removing next-sentence prediction)
- XLM-RoBERTA (multilingual version)
- Check out [EuroBERT](#) from our group (Boizard et al., 2025)

Overall, encoder-only models fell a bit out of fashion, but they are still relevant in several scenarios:

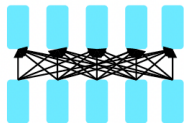
- as **embedding models** (e.g. for RAG)
- as **reward models**
- to build masked **diffusion** (non-autoregressive) language models.

Three Architectures for Pretraining



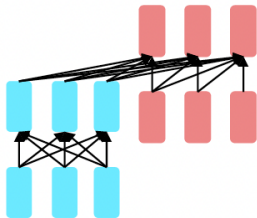
Decoder-only ✓

- Language models with causal masking.
- Nice to generate from; but can't condition on future words.



Encoder-only ✓

- Bidirectional context $\Rightarrow$ can condition on future!
- Wait, how do we pretrain them?



Encoder-Decoder

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

Pretrained Encoder-Decoder

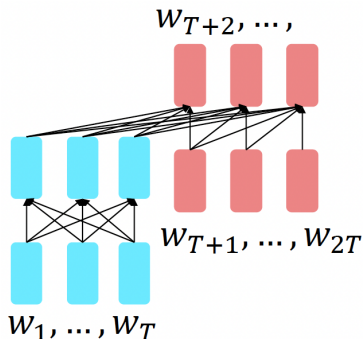
- For encoder-decoders, we can do something like LM, but where a prefix of every input is provided to the encoder and is not predicted.

$$\mathbf{h}_1, \dots, \mathbf{h}_T = \text{Encoder}(x_1, \dots, x_T)$$

$$\mathbf{h}_{T+1}, \dots, \mathbf{h}_{2T} = \text{Decoder}(x_{T+1}, \dots, x_{2T})$$

$$\mathbf{y}_i = \text{softmax}(\mathbf{A}\mathbf{h}_i + \mathbf{b}), i > T$$

- The **encoder** benefits from bidirectional context
- The **decoder** is used to train the whole model through LM.



(Raffel et al., 2020)

T5: Text-to-Text Transfer Transformer (Raffel et al., 2020)

Use **span corruption** as an auxiliary task:

- Replace different length spans from the input with unique placeholders; decode out the spans that were removed!
- This is implemented in text preprocessing: it's still an objective that looks like LM at the decoder side.

Pre-training

(always 15% span masking)

Original text

Thank you for inviting me to your party last week.

Inputs

Thank you <X> me to your party <Y> week.

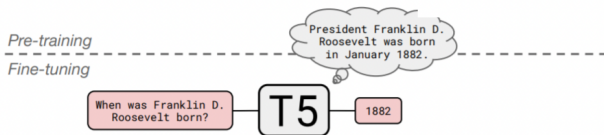
Targets

<X> for inviting <Y> last <Z>

T5: Text-to-Text Transfer Transformer (Raffel et al., 2020)

T5 can be fine-tuned to answer a wide range of questions, retrieving **factual knowledge** from its parameters!

Natural Questions (NQ), WebQuestions (WQ), TriviaQA (TQA)



	NQ	WQ	TQA		
			dev	test	
<u>Karpukhin et al. (2020)</u>	41.5	42.4	57.9	—	
T5.1.1-Base	25.7	28.2	24.2	30.6	220 million params
T5.1.1-Large	27.3	29.5	28.5	37.2	770 million params
T5.1.1-XL	29.5	32.4	36.0	45.1	3 billion params
T5.1.1-XXL	32.8	35.6	42.9	52.5	11 billion params
<u>T5.1.1-XXL + SSM</u>	35.2	42.8	51.9	61.6	

Limitations of Fine-Tuning

Pretraining and **fine-tuning** is a very successful recipe, but what if we want to perform a very large number of tasks?

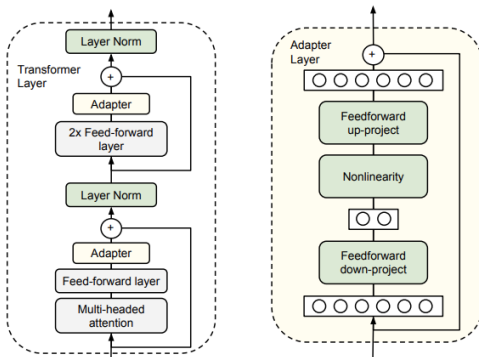
Fine-tuning a very large model to each of the tasks can be very expensive and requires a **copy** of the model for each task.

Can we make this more efficient?

Adapters (Houlsby et al., 2019)

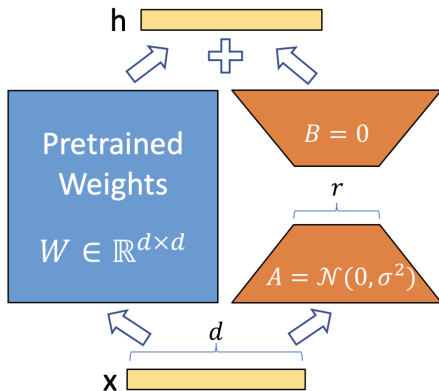
- Alternative to fine-tuning language models on a downstream task
- Instead of **fine-tuning** the full model, **a small set** of task-specific parameters (**adapter**) is included in the model and updated during fine-tuning
- The rest of the model is kept fix
- Several advantages:
 - ✓ Much fewer parameters to fine-tune
 - ✓ Can share the same big pretrained model across tasks, and fine-tune only the task-specific adapters
 - ✓ Can also be used to create multilingual models (language adapters)

Adapters (Houlsby et al., 2019)



- Adapter layers interleaved in the other transformer layers
- In fine-tuning, only these adapter layers are updated
- The big pretrained model stays untouched.

Low-Rank Adaptation (LoRA) (Hu et al., 2022)

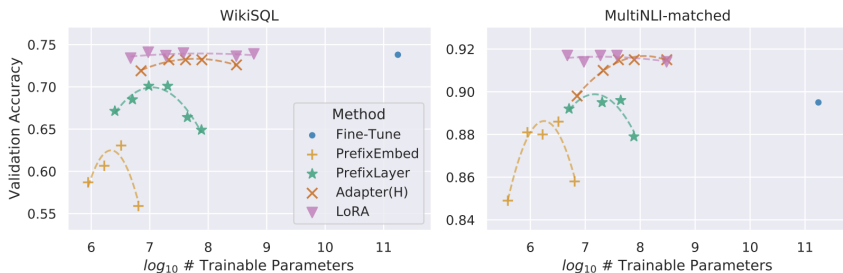


Place adapter **in parallel** rather than sequentially, and skip non-linearity.

Weights after finetuning: $W + \Delta W = W + \alpha BA$, where W is **frozen** and $A \in \mathbb{R}^{r \times d}$, $B \in \mathbb{R}^{d \times r}$ (with $r \ll d$) are **trainable LoRA parameters**.

Low-Rank Adaptation (LoRA) (Hu et al., 2022)

LoRA exhibits better scalability (and sometimes task performance) than full finetuning and other methods.



(<https://lightning.ai/pages/community/article/lora-llm/>)

Outline

1 Transformers

- The Attention Mechanism
- The Transformer Architecture
- Handling Long Contexts
- What's Next?

2 Large Language Models

- The BERT Era: Pretraining and Fine-tuning
- The GPT-3 Era: Prompting and In-Context Learning
- The ChatGPT Era: Post-training and Preference Optimization
- The DeepSeek R1 Era: Reasoning
- What Next?

3 Conclusions

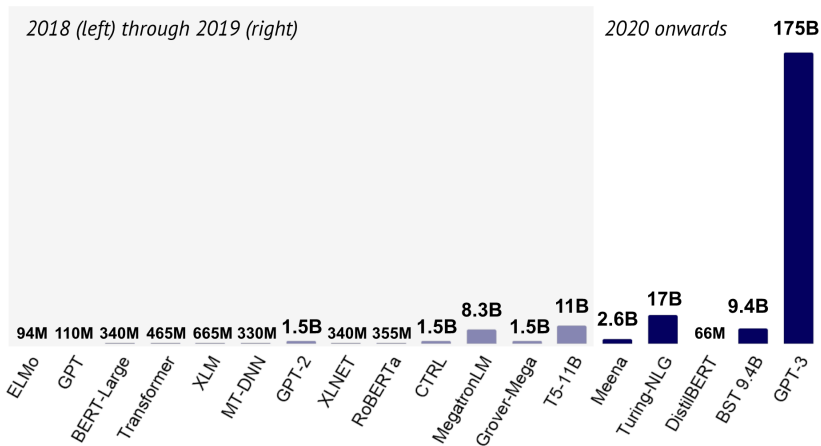
Few-Shot Learning

- What if we want to solve a completely **new** task for which not enough data exists, not even for fine-tuning?
- Can we do it **on-the-fly**?
- This is called **few-shot learning**
- Powerful models such as GPT-3 can do this via **prompting**
- **Leveraging the versatility of LMs is all we need!**

Keep scaling up... Even larger models pretrained on more data:

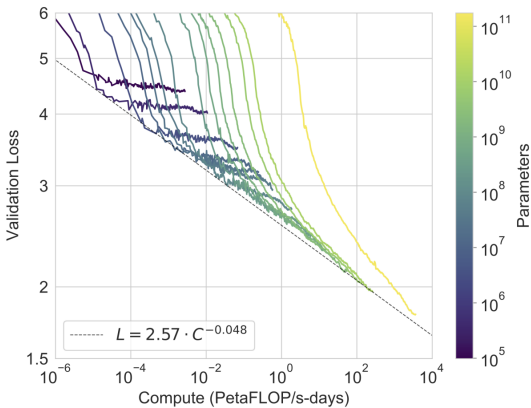
- 175 billion parameters (!!!)
- Trained on 500 billion words
- Took $\sim 3 \times 10^{23}$ FLOPS to train (on a standard GPU, it would cost \$4.6M and it would require 355 years to train such a model)
- Introduces **prompting** as an alternative to fine-tuning
- Demonstrates **few-shot** learning capabilities (learning new tasks on the fly from very few examples)

Large Pretrained Models



Scaling up LLMs is a huge engineering effort!

Scaling Laws (Brown et al., 2020; Kaplan et al., 2020)



(Brown et al., 2020)

Scaling laws help us **predicting performance** for different hyperparameter choices before running expensive experiments.

Chinchilla Laws

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
<i>Chinchilla</i>	70 Billion	1.4 Trillion

(Hoffmann et al., 2022)

Roughly, the cost of training a large transformer scales as parameters $\times$ tokens.

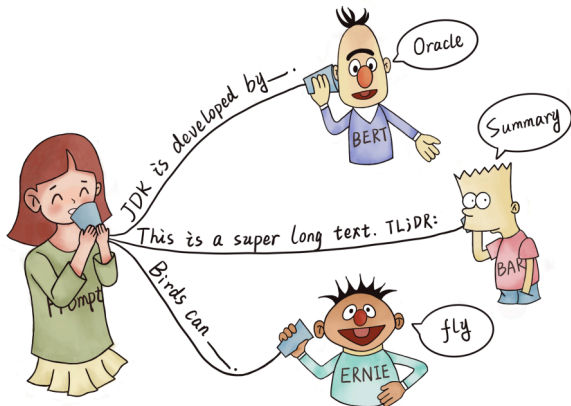
In fact, GPT3 was **very** undertrained (300B tokens only), “Chinchilla laws” corrects earlier Kaplan’s laws.

But what about **inference** costs?

More recently: LLaMA 7B / 13B on 1T tokens (2023), Qwen3 models on 36T tokens, etc. (**well beyond Chinchilla**)

Prompting and In-Context Learning

- Pretrained language models acquire a lot of **factual knowledge!**
- This suggests we can **prompt them on-the-fly** to solve new tasks.



From Liu et al. (2021)

Prompting and In-Context Learning (Brown et al., 2020)

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
```

Traditional fine-tuning (not used for GPT-3)

Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.



Example: Using New Words (Brown et al., 2020)

A "whatpu" is a small, furry animal native to Tanzania. An example of a sentence that uses the word whatpu is:

We were traveling in Africa and we saw these very cute whatpus.

To do a "farduddle" means to jump up and down really fast. An example of a sentence that uses the word farduddle is:

One day when I was playing tag with my little sister, she got really excited and she started doing these crazy farduddles.

A "yalubalu" is a type of vegetable that looks like a big pumpkin. An example of a sentence that uses the word yalubalu is:

I was on a trip to Africa and I tried this yalubalu vegetable that was grown in a garden there. It was delicious.

A "Burringo" is a car with very fast acceleration. An example of a sentence that uses the word Burringo is:

In our garage we have a Burringo that my father drives to work every day.

A "Gigamuru" is a type of Japanese musical instrument. An example of a sentence that uses the word Gigamuru is:

I have a Gigamuru that my uncle gave me as a gift. I love to play it at home.

To "screeg" something is to swing a sword at it. An example of a sentence that uses the word screeg is:

We screeghed at each other for several minutes and then we went outside and ate ice cream.

Example: Grammar Correction (Brown et al., 2020)

Poor English input: I eated the purple berries.
Good English output: I ate the purple berries.

Poor English input: Thank you for picking me as your designer. I'd appreciate it.
Good English output: Thank you for choosing me as your designer. I appreciate it.

Poor English input: The mentioned changes have done. or I did the alteration that you requested. or I changed things you wanted and did the modifications.
Good English output: The requested changes have been made. or I made the alteration that you requested. or I changed things you wanted and made the modifications.

Poor English input: I'd be more than happy to work with you in another project.
Good English output: I'd be more than happy to work with you on another project.

Poor English input: Please provide me with a short brief of the design you're looking for and that'd be nice if you could share some examples or project you did before.
Good English output: Please provide me with a brief description of the design you're looking for and that would be nice if you could share some examples or projects you have done before.

Poor English input: The patient was died.
Good English output: The patient died.

Poor English input: We think that Leslie likes ourselves.
Good English output: We think that Leslie likes us.

Poor English input: Janet broke Bill on the finger.
Good English output: Janet broke Bill's finger.

Poor English input: Mary arranged for, in St. Louis, John to rent a house cheap.
Good English output: Mary arranged for John to rent a house in St. Louis.

Poor English input: Today I have went to the store to to buys some many bottle of water.
Good English output: Today I went to the store to buy some bottles of water.

Poor English input: I have tried to hit ball with bat, but my swing is has miss.
Good English output: I tried to hit the ball with the bat, but my swing missed.

Example: Auto-Completing Code (Chen et al., 2021)

```
def incr_list(l: list):  
    """Return list with elements incremented by 1.  
    >>> incr_list([1, 2, 3])  
    [2, 3, 4]  
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])  
    [6, 4, 6, 3, 4, 4, 10, 1, 124]  
    """  
    return [i + 1 for i in l]
```

```
def solution(lst):  
    """Given a non-empty list of integers, return the sum of all of the odd elements  
    that are in even positions.  
  
    Examples  
    solution([5, 8, 7, 1]) ==>12  
    solution([3, 3, 3, 3, 3]) ==>9  
    solution([30, 13, 24, 321]) ==>0  
    """  
    return sum(lst[i] for i in range(0, len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

Language Modeling $\neq$ Assisting Users

Is **next word prediction** all we need?

PROMPT *Explain the moon landing to a 6 year old in a few sentences.*

COMPLETION GPT-3

Explain the theory of gravity to a 6 year old.

Explain the theory of relativity to a 6 year old in a few sentences.

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.

Language Modeling $\neq$ Assisting Users

Is **next word prediction** all we need?

PROMPT *Explain the moon landing to a 6 year old in a few sentences.*

COMPLETION GPT-3

Explain the theory of gravity to a 6 year old.

Explain the theory of relativity to a 6 year old in a few sentences.

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.

No, language models are not automatically aligned with user intent—we need **instruction tuning** (Ouyang et al., 2022)

Language Modeling $\neq$ Assisting Users

Is **next work prediction** all we need?

PROMPT *Explain the moon landing to a 6 year old in a few sentences.*

COMPLETION **Human**

A giant rocket ship blasted off from Earth carrying astronauts to the moon. The astronauts landed their spaceship on the moon and walked around exploring the lunar surface. Then they returned safely back to Earth, bringing home moon rocks to show everyone.

No, language models are not automatically aligned with user intent—we need **instruction tuning** (Ouyang et al., 2022)

Outline

1 Transformers

- The Attention Mechanism
- The Transformer Architecture
- Handling Long Contexts
- What's Next?

2 Large Language Models

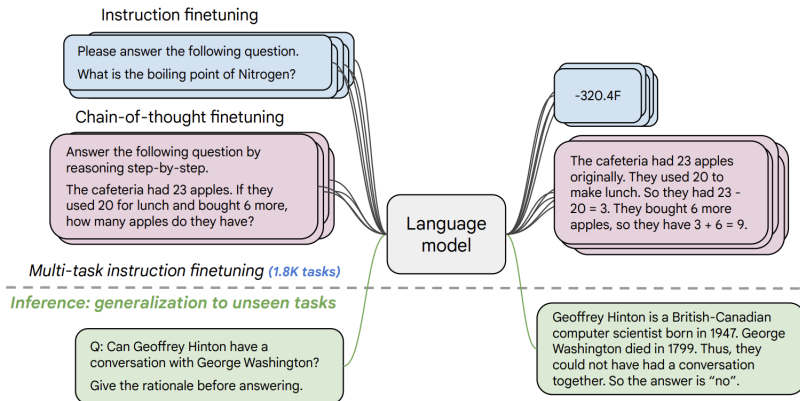
- The BERT Era: Pretraining and Fine-tuning
- The GPT-3 Era: Prompting and In-Context Learning
- The ChatGPT Era: Post-training and Preference Optimization
- The DeepSeek R1 Era: Reasoning
- What Next?

3 Conclusions

Instruction Tuning

Idea: finetune on instruction data (question-answer pairs).

Also called **supervised finetuning**.

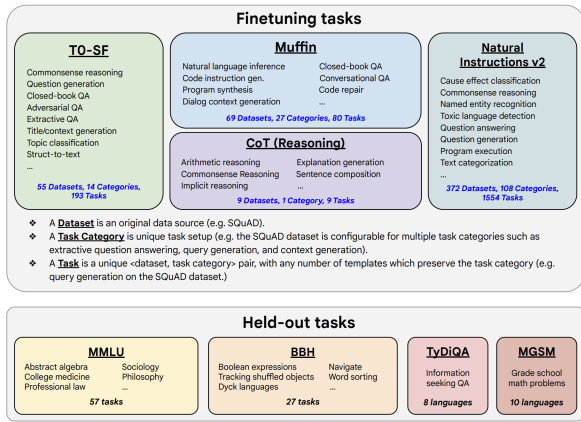


FLAN-T5 (Chung et al., 2024)

Instruction Tuning

Idea: finetune on instruction data (question-answer pairs).

Also called **supervised finetuning**.

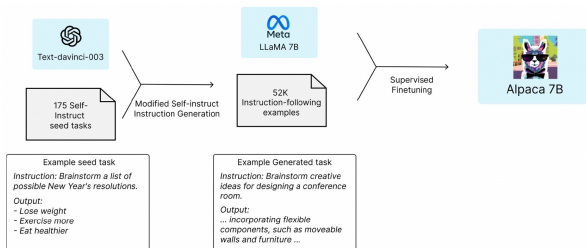


FLAN-T5 (Chung et al., 2024)

Synthetic Instruction Data

Idea: Generate **synthetic** instructions from an LLM (Wang et al., 2023).

Examples: Alpaca (Taori et al., 2023), Vicuna (Chiang et al., 2023), Tülu (Lambert et al., 2025), check out **EuroBlocks** from our group (Ramos et al., 2026), ...



Hypothesis: pretrained LLMs already have all the knowledge, so only ~1000 instructions are needed

(Zhou et al., “LIMA: Less Is More for Alignment”, NeurIPS 2023)

Limitations of Instruction Tuning

- It's **expensive** to collect ground-truth data for tasks.
- Tasks like open-ended creative generation have no **single** correct answer (e.g. “write me a story about a dog and her pet grasshopper”).
- More importantly: SFT penalizes all token-level mistakes equally, but **some errors are worse than others**.
- Even with instruction tuning, there is a **mismatch** between the objective function (minimize perplexity) and the objective of “satisfy human preferences”!

Can we explicitly attempt to satisfy **human preferences**?

How Do We Model Human Preferences?

- 1 For a given question x , sample **several** possible answers $y_1, y_2, \dots$, and ask humans to express their preferences.
- 2 Use this information to train a **reward model** $R_\phi(y; x)$.
- 3 Then let the LLM explore and tune it to maximize the reward.

The last step is what **reinforcement learning** (RL) is about! (covered in Zita Marinho's lecture this afternoon.)

(Fernandes et al., "Bridging the Gap: A Survey on Integrating (Human) Feedback for Natural Language Generation", TACL 2023)

Reward Model

Problem: Human judgments are often noisy and miscalibrated...

... instead of direct scores / rankings, ask them for **pairwise preferences**

This leads to triples (x, y^w, y^l) (question, **winning answer**, **losing answer**)

Then train the **reward model** $R_\phi(y; x)$ to satisfy those preferences.

The Bradley-Terry model does exactly this! (Bradley and Terry, 1952)

$$J_{\text{RM}}(\phi) = -\mathbb{E}_{x, y^w, y^l} \left[\log \sigma(R_\phi(y^w; x) - R_\phi(y^l; x)) \right].$$

Intuition: y^w should score higher than y^l according to R_ϕ .

Putting It All Together (Christiano et al., 2017; Stiennon et al., 2020)

Finally, we have everything we need:

- A pretrained (instruction-tuned) LLM $p^{\text{PT}}(y|x)$
- A reward model $R_\phi(y; x)$ that produces scalar rewards for LLM outputs, trained on a dataset of human comparisons
- A method for optimizing LLM parameters towards this reward.

Now to do RLHF (*Reinforcement Learning with Human Feedback*):

- Initialize a copy of the model $p_\theta^{\text{RL}}(y|x)$ with parameters θ
- Optimize the following reward with RL (PPO):

$$\mathbb{E}_{y \sim p_\theta^{\text{RL}}(y|x)} \left[R_\phi(y; x) - \beta \log \frac{p_\theta^{\text{RL}}(y|x)}{p^{\text{PT}}(y|x)} \right].$$

- The last term is a penalty to prevent us from diverging too much from the pretrained model (in expectation it's $\text{KL}(p_\theta^{\text{RL}} \| p^{\text{PT}})$).

Reinforcement Learning from Human Feedback (RLHF)

Step 1

**Collect demonstration data,
and train a supervised policy.**

A prompt is
sampled from our
prompt dataset.

A labeler
demonstrates the
desired output
behavior.

This data is used
to fine-tune GPT-3
with supervised
learning.



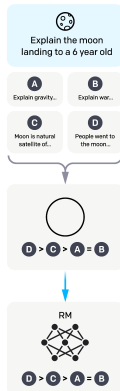
Step 2

**Collect comparison data,
and train a reward model.**

A prompt and
several model
outputs are
sampled.

A labeler ranks
the outputs from
best to worst.

This data is used
to train our
reward model.



Step 3

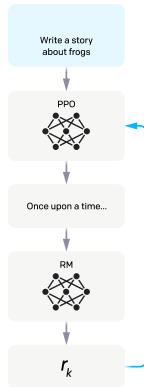
**Optimize a policy against
the reward model using
reinforcement learning.**

A new prompt
is sampled from
the dataset.

The policy
generates
an output.

The reward model
calculates a
reward for
the output.

The reward is
used to update
the policy
using PPO.



Scaled up RLHF to **tens of thousands of tasks**.

Collected data (demonstration data and comparison data) from human annotators for brainstorming, generation, and rewrite tasks.

PROMPT *Explain the moon landing to a 6 year old in a few sentences.*

COMPLETION GPT-3

Explain the theory of gravity to a 6 year old.

Explain the theory of relativity to a 6 year old in a few sentences.

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.

InstructGPT

People went to the moon, and they took pictures of what they saw, and sent them back to the earth so we could all see them.

ChatGPT

Same methods as InstructGPT (SFT + RLHF) for **dialog** agents.

Humans played both sides: user and AI assistant; several RLHF rounds.

Explain in a few sentences how ChatGPT was trained. The audience are students from LxMLS (a summer school in Lisbon). Many are Gen Z so make sure to use a ton of emojis to sound cool.

Alright, quick and vibey version 🤝👉

ChatGPT was trained in **two main phases**:

1. Pretraining 📖

It read a *massive chunk of the internet* (books, websites, code, etc.) and learned to **predict the next word** in a sentence 🧠➡️🔍. That's how it picks up language, facts, and patterns.

2. Alignment (post-training) 🎯

Humans stepped in 🧑🏫 and gave feedback on its answers (what's good 👍 vs bad 👎), often using techniques like reinforcement learning. This helps it be more **helpful, safe, and less chaotic** 😊.

So basically:

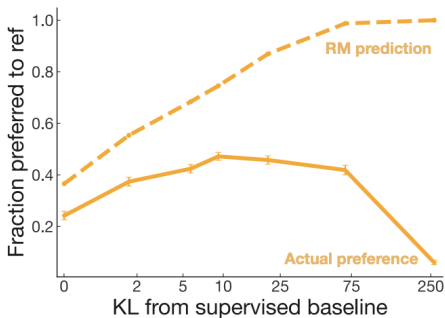
learn from tons of text 📖 + get human feedback 💬 = ChatGPT 🚀

But RLHF Has Limitations Too!

If human preferences are unreliable, *models* of preferences are even worse!

Since the reward model (RM) is imperfect—**reward hacking** is an issue.

Chatbots are rewarded to produce responses that seem authoritative and helpful, regardless of truth—as a result, they often **hallucinate**.



(Stiennon et al., 2020)

Other Variants and Extensions

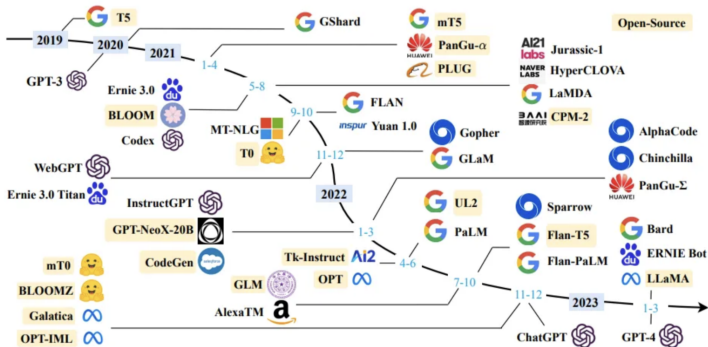
- RL with AI Feedback (“Constitutional AI”) (Bai et al., 2022)
- Direct Preference Optimization (DPO): bypasses the need to train a reward model (Rafailov et al., 2023)
- GRPO (Shao et al., 2024)
- DAPO (Yu et al., 2025)
- ...

This is a very active area of research!

Life After ChatGPT

ChatGPT was released by OpenAI in November 2022 serving a **closed** model.

In early 2023, Meta released LLaMA as a **open-weight** model, and an entire open ecosystem emerged.



Advances in 2023-26

- Some other **closed** models: GPT4-5, Gemini, Claude, Cohere, ...
- Many more **open-weight** models: LLaMA, Mistral, Gemma, Qwen, DeepSeek, Moonshot's Kimi, LongCat, Xiaomi's MiMo, ...
- A few **fully open** models: OLMo, Nemotron, Apertus, EuroLLM, ...
- Many of the recent models do **"reasoning"**.
- Many of the recent models are now **multimodal** or **omni** (e.g. language, vision, speech).
- Several have **agentic** capabilities.

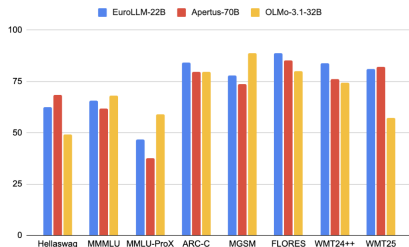
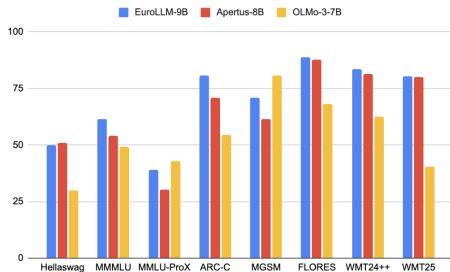
Shameless Plug: EuroLLM (Ramos et al., 2026)

- 4 sizes, **fully open**: **1.7B**, **2.6B-A600M**, **9B**, **22B** (released Dec 2025).
- Supports **all 24 EU official languages**, plus 11 additional ones.
- Compute: **1.68M** GPU-hours in MareNostrum5 (EuroHPC).



(Ramos et al., “EuroLLM-22B: Technical Report”, 2026, <https://eurollm.io>)

Multilingual Benchmarks (EuroLLM $\succ$ Apertus $\succ$ OLMo)

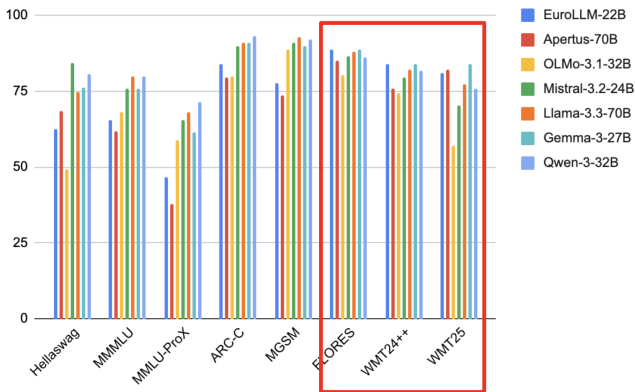


EuroLLM-9B/22B $\succ$ Apertus 8B/70B (despite fewer training tokens, 4T vs 15T).

OLMo-3 worse than both (expected since it's not a multilingual model).

(Ramos et al., "EuroLLM-22B: Technical Report", 2026, <https://eurolm.io>)

Multilingual Benchmarks (incl. open-weight models)



There is still a considerable gap, except for MT where EuroLLM-22B is state-of-the-art or close.

(Ramos et al., “EuroLLM-22B: Technical Report”, 2026, <https://eurollm.io>)

Outline

1 Transformers

- The Attention Mechanism
- The Transformer Architecture
- Handling Long Contexts
- What's Next?

2 Large Language Models

- The BERT Era: Pretraining and Fine-tuning
- The GPT-3 Era: Prompting and In-Context Learning
- The ChatGPT Era: Post-training and Preference Optimization
- The DeepSeek R1 Era: Reasoning
- What Next?

3 Conclusions

Chain-of-Thought Prompting

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ❌

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23 - 20 = 3$. They bought 6 more apples, so they have $3 + 6 = 9$. The answer is 9. ✅

(Wei et al., 2022)

Zero-Shot Chain-of-Thought Prompting

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: **Let's think step by step.** *There are 16 balls in total. Half of the balls are golf balls. That means there are 8 golf balls. Half of the golf balls are blue. That means there are 4 blue golf balls.* ✓

(Kojima et al., 2022)

Starting the answer like this triggers the model to switch to “reasoning” mode.

Many Possible Prompts to Choose From...

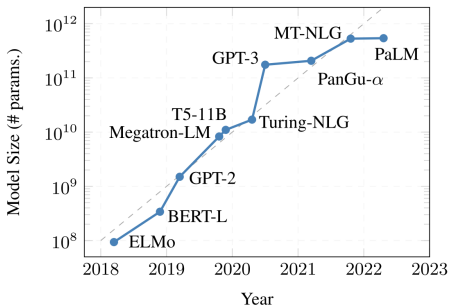
No.	Category	Template	Accuracy
1	instructive	Let's think step by step.	78.7
2		First, (*1)	77.3
3		Let's think about this logically.	74.5
4		Let's solve this problem by splitting it into steps. (*2)	72.2
5		Let's be realistic and think step by step.	70.8
6		Let's think like a detective step by step.	70.3
7		Let's think	57.5
8		Before we dive into the answer,	55.7
9		The answer is after the proof.	45.7
10	misleading	Don't think. Just feel.	18.8
11		Let's think step by step but reach an incorrect answer.	18.7
12		Let's count the number of "a" in the question.	16.7
13		By using the fact that the earth is round,	9.3
14	irrelevant	By the way, I found a good restaurant nearby.	17.5
15		Abrakadabra!	15.5
16		It's a beautiful day.	13.1
-		(Zero-shot)	17.7

(Kojima et al., 2022)

How to Solve Increasingly Complex Problems?

Scaling up model size makes new skills emerge, but **training-time** scaling is just too expensive...

An alternative is to scale at **test-time**!



(Treviso et al., 2023)

Instead of larger models, make models produce **more tokens** before providing the final answer.

This trend started with OpenAI o1, o3, ..., then followed up by **DeepSeek R1** and many others.

Training versus Test-Time Scaling

Simple example: What can we do if we have an LLM and a reward model?

Training versus Test-Time Scaling

Simple example: What can we do if we have an LLM and a reward model?

- We can do RLHF (training time) ✓

Training versus Test-Time Scaling

Simple example: What can we do if we have an LLM and a reward model?

- We can do RLHF (training time) ✓
- We can use the reward model to generate preference pairs and do DPO (also training time) ✓

Training versus Test-Time Scaling

Simple example: What can we do if we have an LLM and a reward model?

- We can do RLHF (training time) ✓
- We can use the reward model to generate preference pairs and do DPO (also training time) ✓
- We can generate many samples and use the reward model to pick the best (at test time!)

Training versus Test-Time Scaling

Simple example: What can we do if we have an LLM and a reward model?

- We can do RLHF (training time) ✓
- We can use the reward model to generate preference pairs and do DPO (also training time) ✓
- We can generate many samples and use the reward model to pick the best (at test time!)

The latter option is called reranking or best-of- N .

... it trades off training-time by test-time compute.

Example of Test-Time Scaling: Best-of- N

Very simple idea:

- 1 Sample N answers from the LLM (may need to increase temperature)
- 2 Use the reward model to pick the “best” answer.

... it works very well in practice!

Example of Test-Time Scaling: Best-of- N

Very simple idea:

- 1 Sample N answers from the **LLM** (may need to increase temperature)
- 2 Use the **reward model** to pick the “best” answer.

... it works very well in practice!

- Widely used even before LLMs (e.g. in machine translation)
(Fernandes et al., 2022; Farinhas et al., 2023)
- More sophisticated variants use **pairwise comparisons**, e.g. MBR decoding (Eikema and Aziz, 2020)

Example of Test-Time Scaling: Best-of- N

Very simple idea:

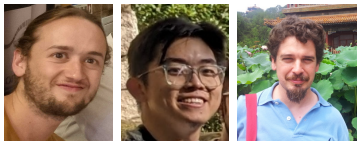
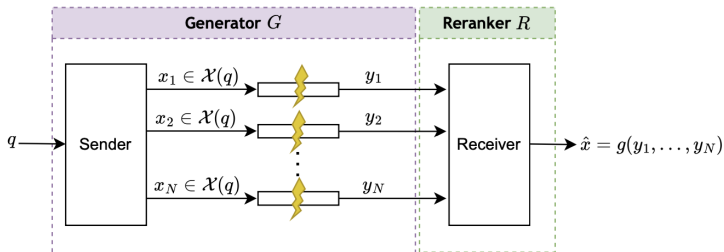
- 1 Sample N answers from the **LLM** (may need to increase temperature)
- 2 Use the **reward model** to pick the “best” answer.

... it works very well in practice!

- Widely used even before LLMs (e.g. in machine translation)
(Fernandes et al., 2022; Farinhas et al., 2023)
- More sophisticated variants use **pairwise comparisons**, e.g. MBR decoding (Eikema and Aziz, 2020)

Intuition: from a communication-theoretic perspective, this is a way to add **redundancy** to decrease decoding error (Farinhas et al., 2024).

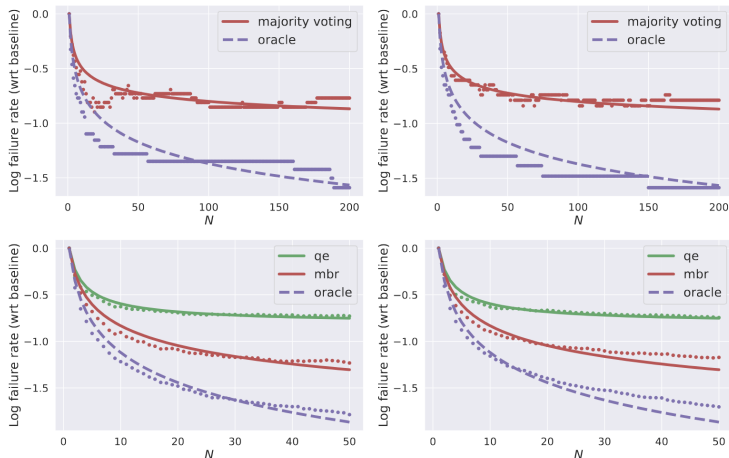
Communication-Theoretic Perspective of Best-of- N



Farinhas, Li, and Martins, “Reranking Laws for Language Generation”, NeurIPS 2024

A Communication-Theoretic Perspective of Best-of- N

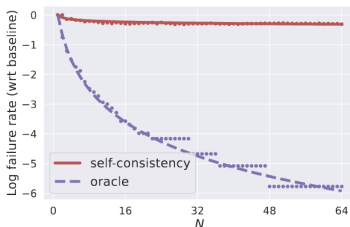
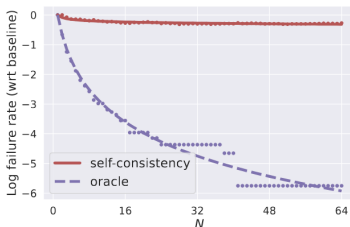
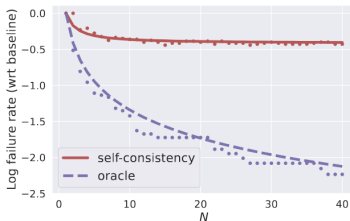
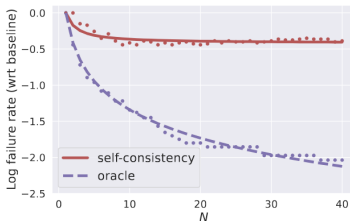
Code generation and machine translation:



Farinhas, Li, and Martins, "Reranking Laws for Language Generation", NeurIPS 2024

A Communication-Theoretic Perspective of Best-of- N

Math and commonsense reasoning:



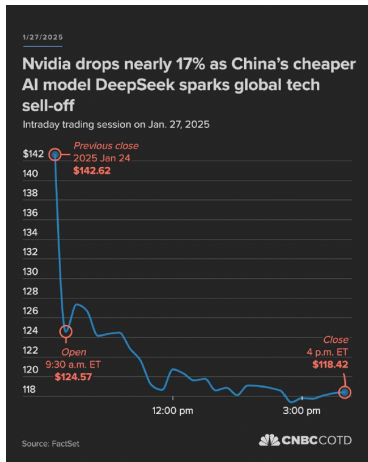
Farinhas, Li, and Martins, "Reranking Laws for Language Generation", NeurIPS 2024

Limitations of Best-of- N and Self-Consistency

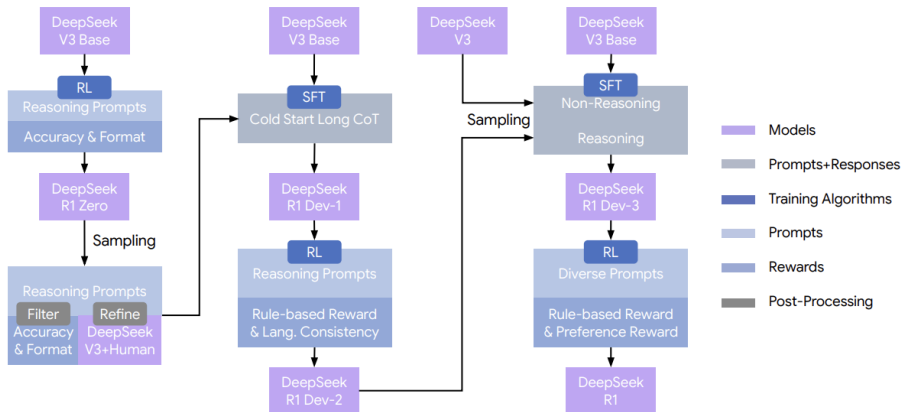
- These methods boost performance for relatively simple problems but they are less effective for problems that are too complex.
- This is because they are limited to the N hypotheses they generate initially.
- For complex problems, N would need to be too large.
- More sophisticated techniques use some sort of tree exploration (e.g. Monte Carlo tree search) or sequential “reasoning” (by generating “thinking” tokens before generating the final answer).

DeepSeek R1 (Guo et al., 2025)

- Released in early 2025 as an **open-weight model**
- A few innovations: a new RL algorithm (GRPO), verifiable rewards, RL without SFT (R1 Zero), ...
- At the time all reasoning models were closed
- Caused some disruption in the stock market!



DeepSeek R1 Pipeline



(Guo et al., 2025)

First Stage: DeepSeek R1 Zero

- Format: `<think> ... </think>` followed by `<answer> ... </answer>`
- **Binary (verifiable) rewards (RLVR)**: for math, compare final answer against ground truth; for code, run test cases.
- Include also a small **format reward** to encourage using the `<think>` and `<answer>` tags.
- **Outcome-based rewards** (i.e. based on the final answer) as opposed to process reward models.
- Doing RL before SFT encourages the model to explore more and **discover novel reasoning strategies**.
- Take this with a grain of salt: it's likely that the model was exposed to math proofs and code during pretraining!

DeepSeek R1 Aha Moment

Table 2 | An interesting “aha moment” of an intermediate version of DeepSeek-R1-Zero. The model learns to rethink using an anthropomorphic tone. This is also an aha moment for us, allowing us to witness the power and beauty of reinforcement learning.

Question: If $a > 1$, then the sum of the real solutions of $\sqrt{a - \sqrt{a + x}} = x$ is equal to

Response: <think>

To solve the equation $\sqrt{a - \sqrt{a + x}} = x$, let's start by squaring both ...

$$\left(\sqrt{a - \sqrt{a + x}}\right)^2 = x^2 \implies a - \sqrt{a + x} = x^2.$$

Rearrange to isolate the inner square root term:

$$(a - x^2)^2 = a + x \implies a^2 - 2ax^2 + (x^2)^2 = a + x \implies x^4 - 2ax^2 - x + (a^2 - a) = 0$$

...

Wait, wait. Wait. That's an aha moment I can flag here.

Let's reevaluate this step-by-step to identify if the correct sum can be ...

We started with the equation:

$$\sqrt{a - \sqrt{a + x}} = x$$

First, let's square both sides:

$$a - \sqrt{a + x} = x^2 \implies \sqrt{a + x} = a - x^2$$

Next, I could square both sides again, treating the equation: ...

...

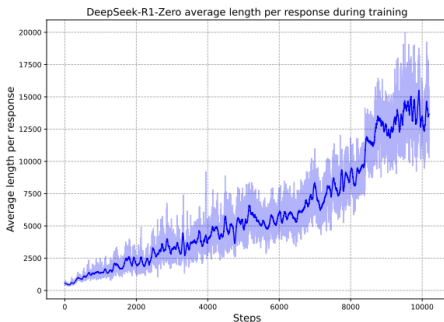
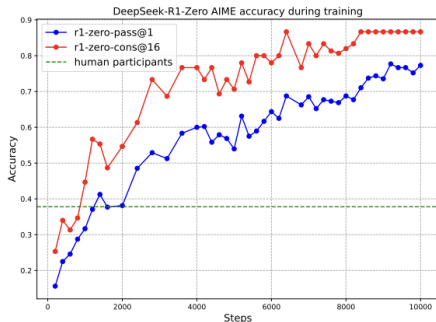
(Guo et al., 2025)

Some Emergent Reasoning Capabilities

- **Self-verification:** the model checks its own work after deriving an answer. Emerges because self-checked outputs are more often correct.
- **Reflection and backtracking** (“Wait, let me reconsider”): the model recognizes wrong paths and restarts with a different strategy.
- **Extended deliberation:** the response length grows from hundreds to thousands of tokens. The model is essentially learning that thinking longer leads to better answers, a learned form of test-time compute scaling.

On the negative side, reasoning traces are often poorly understandable and switch between languages.

Accuracy and Reasoning Length During Training



(Guo et al., 2025)

Reasoning increases **context size** significantly → cf. first part of the lecture on **long context adaptation** and **efficient transformers**.

Reinforcement Learning with Verifiable Rewards

The rewards in DeepSeek-R1 are **verifiable**.

No reward model needed! A simple regex can check the answer.

This allows simplifying many things.

DeepSeek-R1 also replaced PPO by a better RL algorithm:

- “Group Relative Policy Optimization (GRPO)” (Shao et al., 2024)

More about this in Zita’s lecture this afternoon!

Outline

1 Transformers

- The Attention Mechanism
- The Transformer Architecture
- Handling Long Contexts
- What's Next?

2 Large Language Models

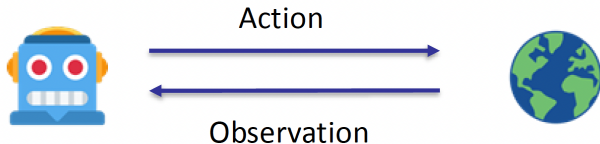
- The BERT Era: Pretraining and Fine-tuning
- The GPT-3 Era: Prompting and In-Context Learning
- The ChatGPT Era: Post-training and Preference Optimization
- The DeepSeek R1 Era: Reasoning
- What Next?

3 Conclusions

From Tokens to Action

LLMs generate textual output for a given input, but they don't intervene.

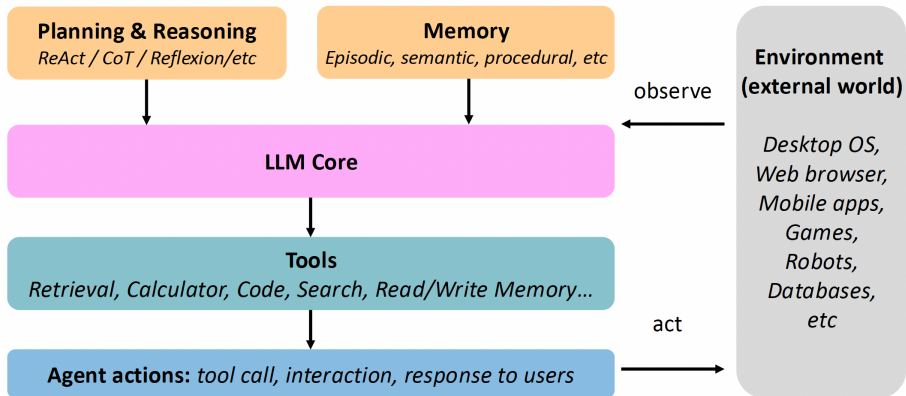
Agents perform **actions** based on observations from the environment/world.



Two perspectives on LLMs and agents:

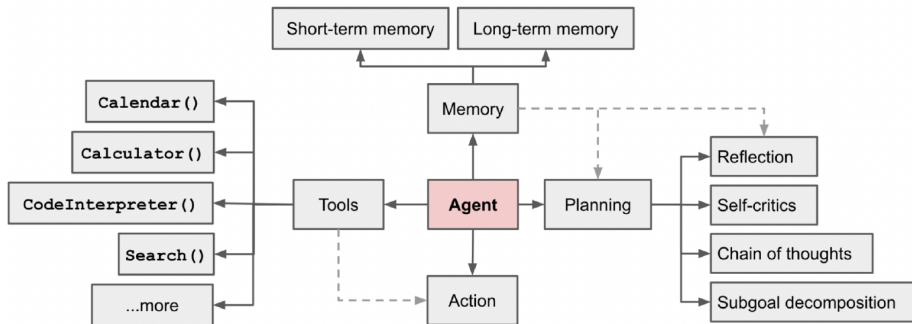
- Agents can use LLMs to help reasoning and planning.
- LLMs can be enhanced to be given “agentic” capabilities.

Key Components



(Credits: Diyi Yang's NLP class)

Key Components



(<https://lilianweng.github.io/posts/2023-06-23-agent/>)

Agents can combine LLM knowledge, reasoning, and planning capabilities with external tools.

Toolformer (Schick et al., 2023)

- Trains LLMs to decide which APIs to call, when to call them, what arguments to pass, and how to best incorporate the results.
- Use self-supervised learning with only a handful of demonstrations for each API.
- Tools: calculator, Q&A system, search engine, translation system, calendar.

The New England Journal of Medicine is a registered trademark of [QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society] the MMS.

Out of 1400 participants, 400 (or [Calculator(400 / 1400) → 0.29] 29%) passed the test.

The name derives from "la tortuga", the Spanish word for [MT("tortuga") → turtle] turtle.

The Brown Act is California's law [WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.] that requires legislative bodies, like city councils, to hold their meetings open to the public.

(Schick et al., 2023)

Toolformer (Schick et al., 2023)

- Trains LLMs to decide which APIs to call, when to call them, what arguments to pass, and how to best incorporate the results.
- Use self-supervised learning with only a handful of demonstrations for each API.
- Tools: calculator, Q&A system, search engine, translation system, calendar.

Your task is to add calls to a Question Answering API to a piece of text. The questions should help you get information required to complete the text. You can call the API by writing "[QA(question)]" where "question" is the question you want to ask. Here are some examples of API calls:

Input: Joe Biden was born in Scranton, Pennsylvania.

Output: Joe Biden was born in [QA("Where was Joe Biden born?")] Scranton, [QA("In which state is Scranton?")] Pennsylvania.

Input: Coca-Cola, or Coke, is a carbonated soft drink manufactured by the Coca-Cola Company.

Output: Coca-Cola, or [QA("What other name is Coca-Cola known by?")] Coke, is a carbonated soft drink manufactured by [QA("Who manufactures Coca-Cola?")] the Coca-Cola Company.

(Schick et al., 2023)

Why Is Reasoning Helpful for Agents?

Obs t

You are cooking a dish and seeing salt is out...

Reasoning

“The dish should be savory, and since salt is out, I should find the soy sauce instead. It is in the cabinet to my right...”

Act t

Turn right

Obs $t+1$

You see a cabinet and a table...

Act $t+1$

Open cabinet

(Credits: Diyi Yang's NLP class)

Orchestrator

In some scenarios it is useful to have an **orchestrator** to divide the task into subgoals and coordinate with multiple systems.



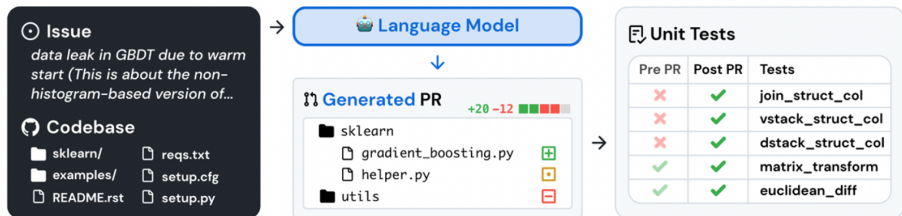
<https://www.microsoft.com/en-us/research/publication/magnetic-one-a-generalist-multi-agent-system-for-solving-complex-tasks/>

Agents Can Be Used In Many Domains and Environments

- Coding agents
- Gaming agents
- Mobile agents
- Web/app agents
- Computer agents
- Robotics

Example: Coding Agents

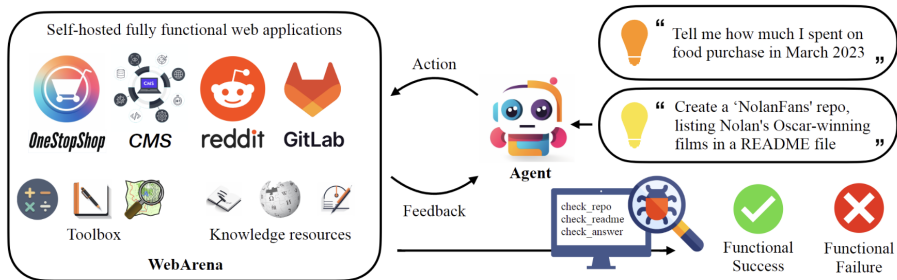
- Environment: project code repos, filesystems, IDEs, ...
- Observation space: code files, exe outputs, docs, errors, commit history, ...
- Action space: code edits, file search/view, test updates, ...



SWE-bench (Jimenez et al., 2023)

Example: Web/App Agents

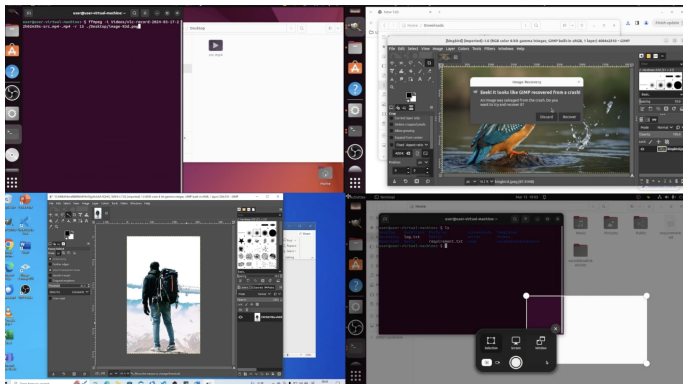
- Environment: web browsers/apps
- Observation space: screenshots, HTML, historical actions, ...
- Action space: browser/app controls (click, type, scroll, drag, hover, ...)



WebArena (Zhou et al., 2023b), <https://webarena.dev/og/>

Example: Computer Use Agents

- Environment: desktop operating systems
- Observation space: screenshots, accessibility trees, historical action, ...
- Action space: keyboard/mouse controls (click, type, drag, shortcuts)



OSWorld (Xie et al., 2024), <https://os-world.github.io/>

Many Topics Not Covered in This Lecture

- Retrieval-augmented generation (RAG)
- Tools and agents only covered very superficially
- Deep research, AI for science, ...
- Vision-language-action models, world models, ...
- Pretraining and midtraining optimization
- Evaluation
- Reinforcement learning → Zita's lecture this afternoon!
- Multimodal LLMs → Desmond's lecture on Saturday!
- ...

Outline

1 Transformers

- The Attention Mechanism
- The Transformer Architecture
- Handling Long Contexts
- What's Next?

2 Large Language Models

- The BERT Era: Pretraining and Fine-tuning
- The GPT-3 Era: Prompting and In-Context Learning
- The ChatGPT Era: Post-training and Preference Optimization
- The DeepSeek R1 Era: Reasoning
- What Next?

3 Conclusions

Conclusions

- LLMs went through several stages, from pretraining and finetuning to prompting, reasoning, acting, ...
- Increasing the model size and training compute is not the only way to drive progress.
- Test-time scaling can be effective—models can “reason” at inference time and explore multiple paths to reach a solution.
- Challenge: we need to accommodate very long context lengths.
- Compressing the KV cache, using sparse attention, or hybrid architectures are potential solutions.
- We can also combine retrieval with generation through RAG or use external tools.
- Very active research on models with agent capabilities (code agents, Web agents, embodied agents, ...)

Thank you!

Questions?



References I

- Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, 2023.
- S-I Amari. Learning patterns and pattern sequences by self-organizing nets of threshold elements. *IEEE Transactions on computers*, 100(11): 1197–1206, 1972.
- Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.
- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations*, 2015.
- Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.
- Nicolas Boizard, Hippolyte Gisserot-Boukhlef, Duarte M Alves, André FT Martins, Ayoub Hammal, Caio Corro, Céline Hudelot, Emmanuel Malherbe, Etienne Malaboeuf, Fanny Jourdan, et al. Eurobert: Scaling multilingual encoders for european languages. In *Second Conference on Language Modeling*, 2025.
- Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39 (3/4):324–345, 1952.
- Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- Wei-Lin Chiang, Zhuohan Li, Ziqing Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. See <https://vicuna.lmsys.org> (accessed 14 April 2023), 2(3):6, 2023.
- Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

References II

- Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024. URL <http://jmlr.org/papers/v25/23-0870.html>.
- Gonalo M Correia, Vlad Niculae, and Andr  FT Martins. Adaptively sparse transformers. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2174–2184, 2019.
- Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher R . Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- Bryan Eikema and Wilker Aziz. Is map decoding all you need? the inadequacy of the mode in neural machine translation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 4506–4520, 2020.
- Ant nio Farinhas, Jos  de Souza, and Andr  FT Martins. An empirical study of translation hypothesis ensembling with large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 11956–11970, 2023.
- Ant nio Farinhas, Haau-Sing Li, and Andr  F Martins. Reranking laws for language generation: A communication-theoretic perspective. *Advances in Neural Information Processing Systems*, 37:111074–111105, 2024.
- Patrick Fernandes, Ant nio Farinhas, Ricardo Rei, Jos  GC de Souza, Perez Ogayo, Graham Neubig, and Andr  FT Martins. Quality-aware decoding for neural machine translation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1396–1412, 2022.
- Patrick Fernandes, Aman Madaan, Emmy Liu, Ant nio Farinhas, Pedro Henrique Martins, Amanda Bertsch, Jos  GC de Souza, Shuyan Zhou, Tongshuang Wu, Graham Neubig, et al. Bridging the gap: A survey on integrating (human) feedback for natural language generation. *Transactions of the Association for Computational Linguistics*, 11:1643–1668, 2023.
- Nuno Gonalves, Marcos V Treviso, and Andre Martins. Adasplash: Adaptive sparse flash attention. In *Forty-second International Conference on Machine Learning*, 2025. URL <https://openreview.net/forum?id=0WIPDWhUc0>.

References III

- Nuno Gonçalves, Hugo Pitorro, Vlad Niculae, Edoardo Ponti, Lei Li, Andre Martins, and Marcos Treviso. Adasplash-2: Faster differentiable sparse attention. In *Forty-third International Conference on Machine Learning*, 2026. URL <https://arxiv.org/abs/2604.15180>.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1):31, 1991.
- Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. In *NeurIPS*, 2022.
- John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.
- Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL <https://openreview.net/forum?id=nZeVKeeFYf9>.
- Yuxiang Huang, Nuno MT Gonçalves, Federico Alvetreti, Lei Li, Xu Han, Edoardo M Ponti, André FT Martins, and Marcos V Treviso. Dashattention: Differentiable and adaptive sparse hierarchical attention. *arXiv preprint arXiv:2605.18753*, 2026.
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.
- Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL <https://kellerjordan.github.io/posts/muon>, 6(3):4, 2024.
- Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

References IV

- Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.
- Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxu Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*, 2025.
- Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in Neural Information Processing Systems*, 31, 2018.
- Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586*, 2021.
- Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.
- André F. T. Martins and Ramón Astudillo. From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. In *Proc. of the International Conference on Machine Learning*, 2016.
- Pedro Henrique Martins, Zita Marinho, and André FT Martins. ∞ -former: Infinite memory transformer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5468–5485, 2022.
- Elizbar A Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142, 1964.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.
- Ben Peters, Vlad Niculae, and André FT Martins. Sparse sequence-to-sequence models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1504–1519, 2019.
- Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

References V

- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67, 2020.
- Miguel Moura Ramos, Duarte M Alves, Hippolyte Gisserot-Boukhlef, João Alves, Pedro Henrique Martins, Patrick Fernandes, José Pombal, Nuno M Guerreiro, Ricardo Rei, Nicolas Boizard, et al. Eurollm-22b: Technical report. *arXiv preprint arXiv:2602.05879*, 2026.
- Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Thomas Adler, Lukas Gruber, Markus Holzleitner, Milena Pavlović, Geir Kjetil Sandve, et al. Hopfield networks is all you need. *arXiv preprint arXiv:2008.02217*, 2020.
- Saul Santos, António Farinhas, Daniel C McNamee, and Andre Martins. ∞ -video: A training-free approach to long video understanding via continuous-time memory consolidation. In *Forty-second International Conference on Machine Learning*, 2025.
- Saul Santos, Nuno Gonçalves, Daniel C McNamee, Marcos Treviso, and André FT Martins. Sparse attention as compact kernel regression. *arXiv preprint arXiv:2601.22766*, 2026.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.
- Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International conference on machine learning*, pages 9355–9366. PMLR, 2021.
- Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1): 131–139, 1992.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, 2018.
- Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.
- Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in neural information processing systems*, 33:3008–3021, 2020.
- Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

References VI

- Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. *Advances in neural information processing systems*, 28, 2015.
- Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.
- Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models*. <https://crfm.stanford.edu/2023/03/13/alpaca.html>, 3(6):7, 2023.
- Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28, 2022.
- Marcos Treviso, Ji-Ung Lee, Tianchu Ji, Betty Van Aken, Qingqing Cao, Manuel R Ciosici, Michael Hassid, Kenneth Heafield, Sara Hooker, Colin Raffel, et al. Efficient methods for natural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 11:826–860, 2023.
- Constantino Tsallis. Possible generalization of boltzmann-gibbs statistics. *Journal of Statistical Physics*, 52:479–487, 1988.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.
- Pavlo Vasylenko, Hugo Pitorro, André FT Martins, and Marcos Treviso. Long-context generalization with sparse attention. In *International Conference on Learning Representations*, 2025.
- Ke Alexander Wang, Jiaxin Shi, and Emily B Fox. Test-time regression: a unifying framework for designing sequence models with associative memory. *arXiv preprint arXiv:2501.12352*, 2025.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023.
- Geoffrey S Watson. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 359–372, 1964.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

References VII

- Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International conference on machine learning*, pages 10524–10533. PMLR, 2020.
- Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.
- Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23078–23097, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1126. URL <https://aclanthology.org/2025.acl-long.1126/>.
- Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.
- Jianyu Zhang, Niklas Nolte, Ranajoy Sadhukhan, Beidi Chen, and Leon Bottou. Memory mosaics. In *The Thirteenth International Conference on Learning Representations*, 2025.
- Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36:55006–55021, 2023a.
- Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023b.