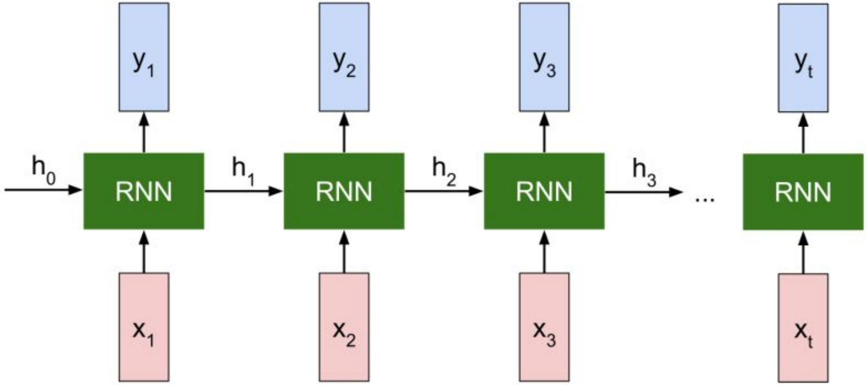
Transformers, Large Language Models (LLMs), and Foundation Models

Ju Sun
Computer Science & Engineering

Apr 08, 2025

Quick recap

RNN: model sequences



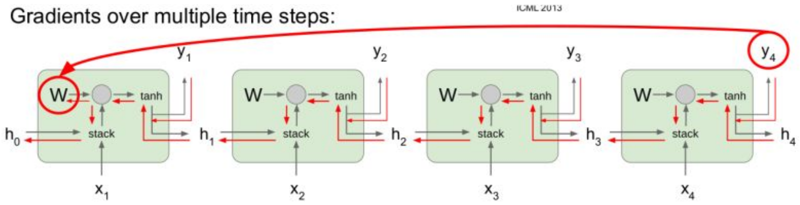
(Credit: Stanford CS231N)

$$\boldsymbol{h}_t = \tanh\left(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{\boldsymbol{x}}\boldsymbol{x}_t\right)$$

$$\boldsymbol{y}_t = \boldsymbol{V}_y\boldsymbol{h}_t$$

$\boldsymbol{W}_{\boldsymbol{h}}, \boldsymbol{W}_{\boldsymbol{x}}$ and $\boldsymbol{V}_{\boldsymbol{y}}$ are shared across the sequence

Vanishing/exploding gradient issue



(Credit: Stanford CS231N)

$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \boldsymbol{h}_t}\frac{\partial \boldsymbol{h}_t}{\partial h_{t-1}}\cdots\frac{\partial \boldsymbol{h}_1}{\boldsymbol{W}} = \frac{\partial L_t}{\partial \boldsymbol{h}_t}\left(\prod_{k=2}^{t}\frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{h}_{k-1}}\right)\frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{W}}$$

$$= \frac{\partial L_t}{\partial \boldsymbol{h}_t}\left(\prod_{k=2}^{t}\operatorname{diag}\left(\tanh'\left(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{h}_{k-1} + \boldsymbol{W}_{\boldsymbol{x}}\boldsymbol{x}_k\right)\right)\boldsymbol{W}_h\right)\frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{W}}$$

- when $\|\boldsymbol{W}_h\| > 1$, gradient **explodes** if t large
- when $\|\boldsymbol{W}_h\| < 1$, gradient **vanishes** if t large

$$\left\|\prod_{k=2}^{t}\operatorname{diag}\left(\tanh'\left(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{h}_{k-1} + \boldsymbol{W}_{\boldsymbol{x}}\boldsymbol{x}_k\right)\right)\boldsymbol{W}_h\right\|$$

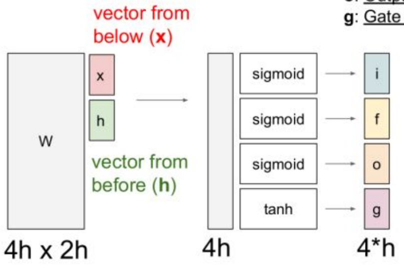
$$\leq \prod_{k=2}^{t}\left\|\operatorname{diag}\left(\tanh'\left(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{h}_{k-1} + \boldsymbol{W}_{\boldsymbol{x}}\boldsymbol{x}_k\right)\right)\right\| \|\boldsymbol{W}_h\|$$

$$\leq \prod_{k=2}^{t}\left\|\operatorname{diag}\left(\tanh'\left(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{h}_{k-1} + \boldsymbol{W}_{\boldsymbol{x}}\boldsymbol{x}_k\right)\right)\right\| \|\boldsymbol{W}_h\|^{t-1}$$

Gated RNNs

Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]

i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell
o: Output gate, How much to reveal cell
g: Gate gate (?), How much to write to cell



$$\boxed{\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

(Credit: Stanford CS231N)

$\boldsymbol{u}$: **update gate**, control state update

$\boldsymbol{r}$: **reset gate**, control how previous state affects new content

$\boldsymbol{g}$: new content

Gated recurrent unit (GRU)

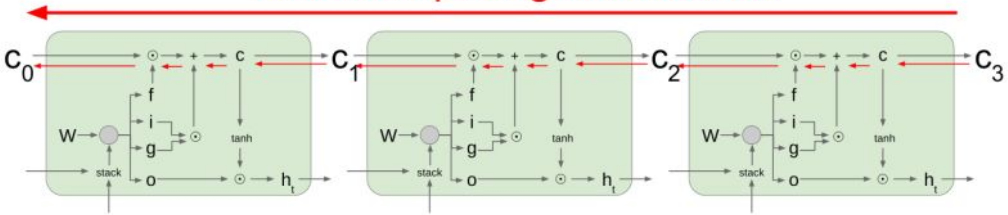
$$\begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{r} \\ \boldsymbol{g} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \end{bmatrix} \left(\boldsymbol{W} \begin{bmatrix} \boldsymbol{h}_{t-1} \\ \boldsymbol{x}_t \end{bmatrix} \right)$$

$$\boldsymbol{g} = \tanh\left(\boldsymbol{W}_h \left(\boldsymbol{r} \odot \boldsymbol{h}_{t-1}\right) + \boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{b}_g\right)$$

$$\boldsymbol{h}_t = \boldsymbol{u} \odot \boldsymbol{h}_{t-1} + (1 - \boldsymbol{u}) \odot \boldsymbol{g}$$

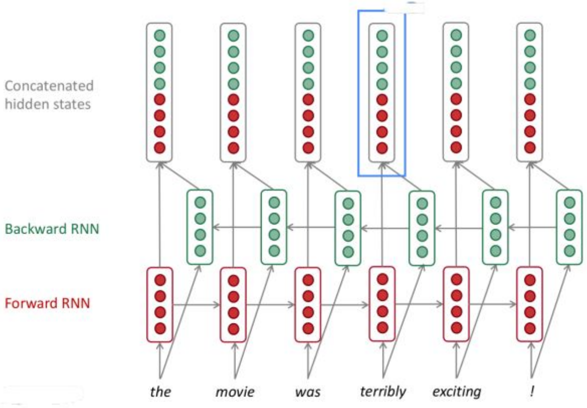
$\boldsymbol{f}$, $\boldsymbol{i}$, $\boldsymbol{o}$ are merged

Uninterrupted gradient flow!



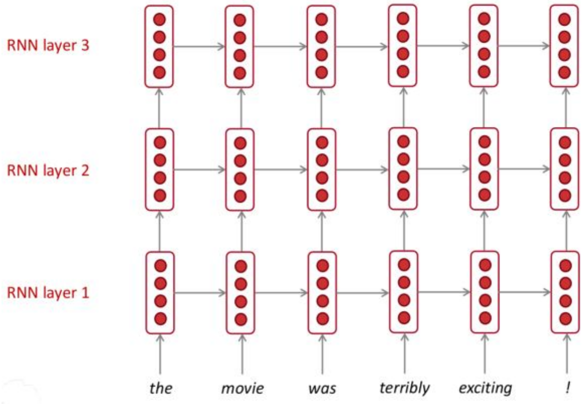
(Credit: Stanford CS231N)

Modern RNNs



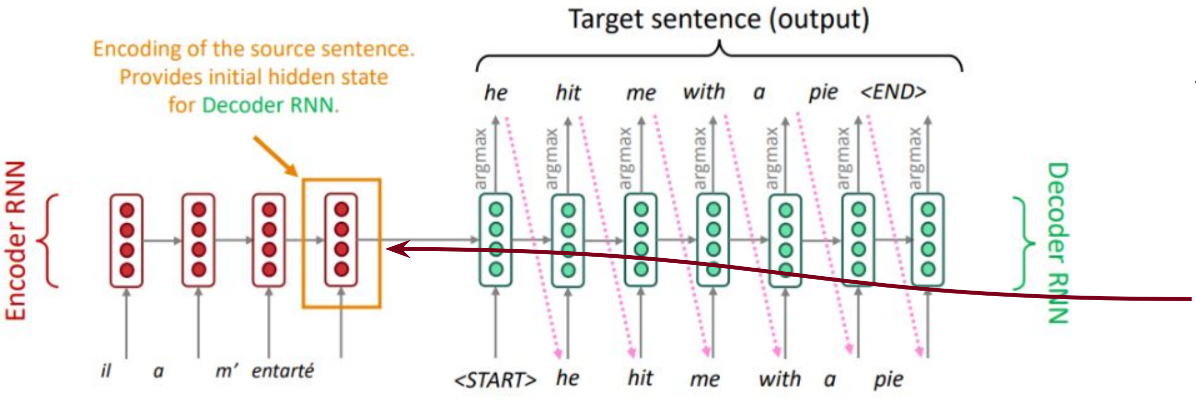
(Credit: Stanford CS224N)

Bidirectional RNN



(Credit: Stanford CS231N)

Deep RNN

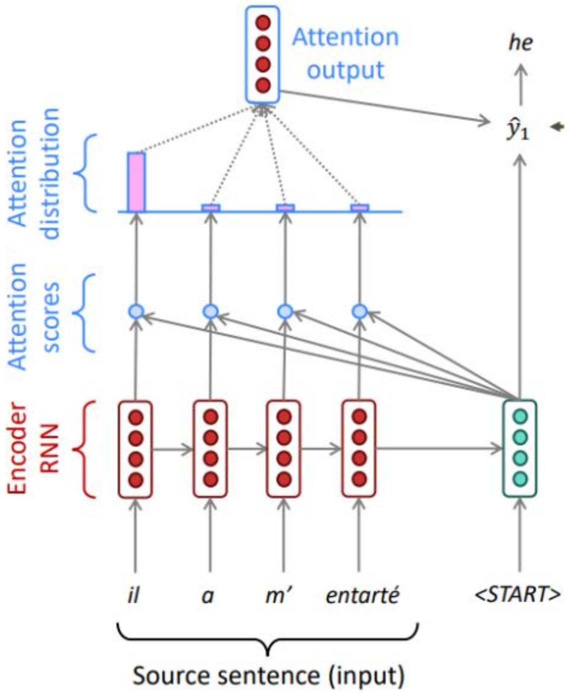


Seq2Seq model

Bottleneck problem

(Credit: Stanford CS231N)

Attention mechanism



(Credit: Stanford CS231N)

Input: source vectors $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N \in \mathbb{R}^h$, and target vector $\boldsymbol{t}$

Output: weighted summation

$$\sum_{j=1}^{N} w_j \boldsymbol{s}_j \quad \text{where } w_j = \text{similarity}(\boldsymbol{s}_j, \boldsymbol{t})$$

Many possibilities:

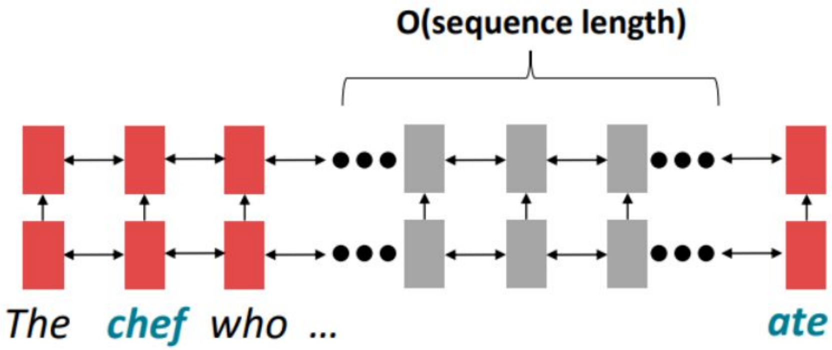
Attention scores

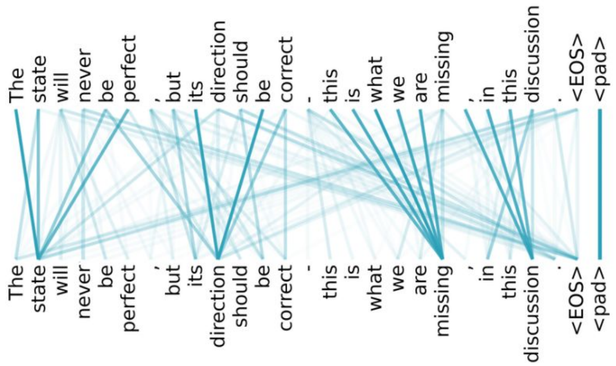
- dot-product attention: $\widehat{w_j} = \langle \boldsymbol{s}_j, \boldsymbol{t} \rangle$ (Is is better to normalize this or rescale it by the dimension factor?)
- multiplicative attention: $\widehat{w_j} = \langle \boldsymbol{s}_j, \boldsymbol{W}\boldsymbol{t} \rangle$
- "additive attention": $\widehat{w_j} = \boldsymbol{v}^\top \sigma \left(\boldsymbol{W}_1 \boldsymbol{s}_j + \boldsymbol{W}_2 \boldsymbol{t} \right)$

The actual weights are attention scores passed through **softmax**

$$w_j = \frac{\exp\left(\widehat{w_j}\right)}{\sum_k \exp\left(\widehat{w_k}\right)}$$

Self-attention





RNN

- Long interaction distance
- Resistant to parallelization

Self-attention

- O(1) interaction distance
- Highly parallelizable

Each token gets a selective summary of information from all others

Self-attention

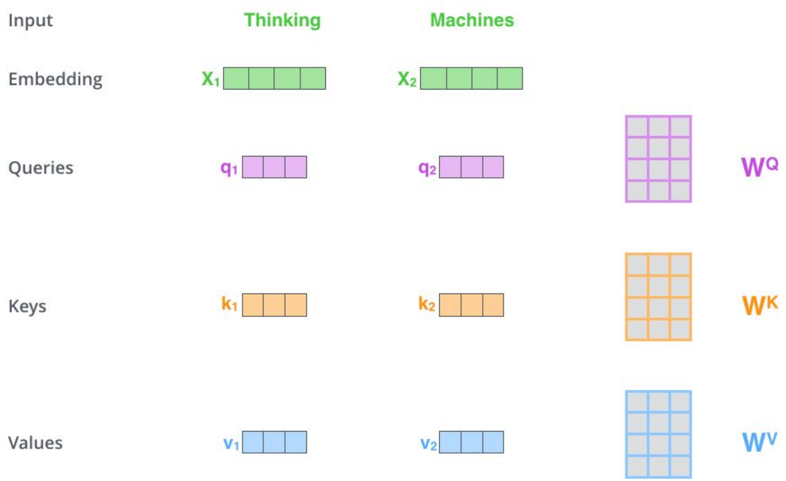


Image credit: https://jalammar.github.io/illustrated-transformer/

- Each word now encoded as (query, key, value) triple
- For an input $\boldsymbol{x}_i$, we have:

$$\boldsymbol{q}_i = (\boldsymbol{W}^Q)^\intercal \boldsymbol{x}_i, \quad \boldsymbol{k}_i = (\boldsymbol{W}^K)^\intercal \boldsymbol{x}_i, \quad \boldsymbol{v}_i = (\boldsymbol{W}^V)^\intercal \boldsymbol{x}_i$$

- Calculate attention scores between query and all keys: $e_{ij} = \langle \boldsymbol{q}_i, \boldsymbol{k}_j \rangle$
- softmax normalization $w_{ij} = \exp(e_{ij}) \,/\, \sum_k \exp(e_{ik})$
- output the weighted sum of values $\sum_j w_{ij} \boldsymbol{v}_j$

In matrix notation

- Compute queries, keys, and values

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K, \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V$$

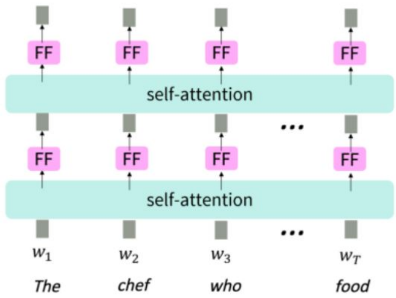
- Calculate attention scores between query and all keys: $\boldsymbol{E} = \boldsymbol{Q}\boldsymbol{K}^\intercal$
- softmax normalization $\boldsymbol{A} = \mathrm{softmax}(\boldsymbol{E})$
- output the weighted sum of values $\boldsymbol{A}\boldsymbol{V}$

$$\text{output} = \mathrm{softmax}(\boldsymbol{Q}\boldsymbol{K}^\intercal)\boldsymbol{V}$$

Question: why we need both query and key?

Equation for Feed Forward Layer

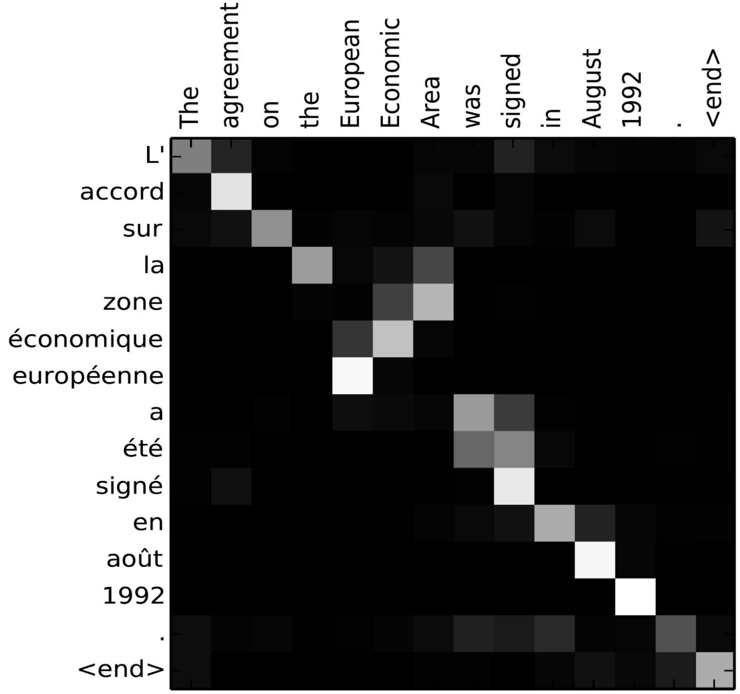
$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \mathrm{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$$



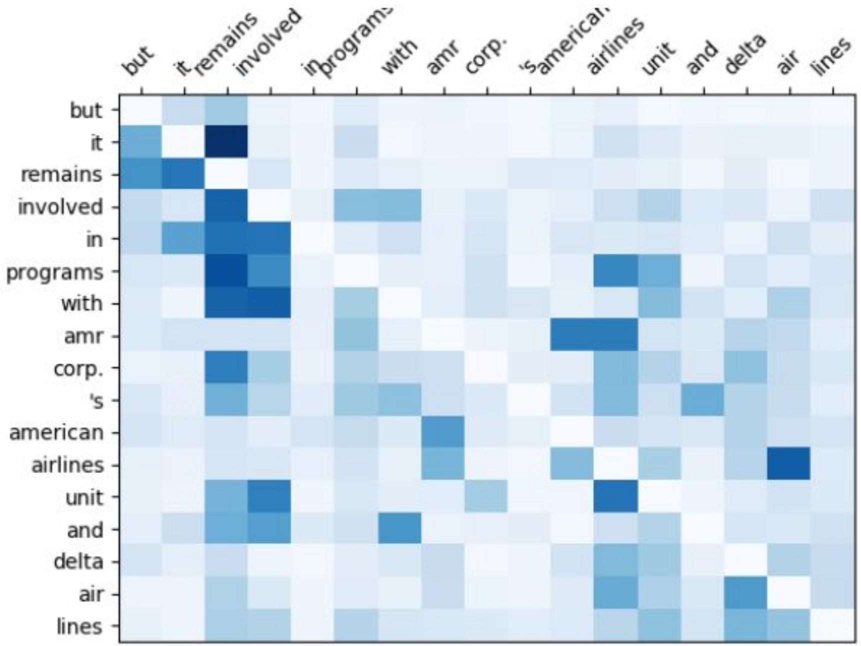
Adding in nonlinearity!

First step toward Transformers!

Attention matrices—visualizing correlations



General attention



Self-attention

Transformers

Transformers

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

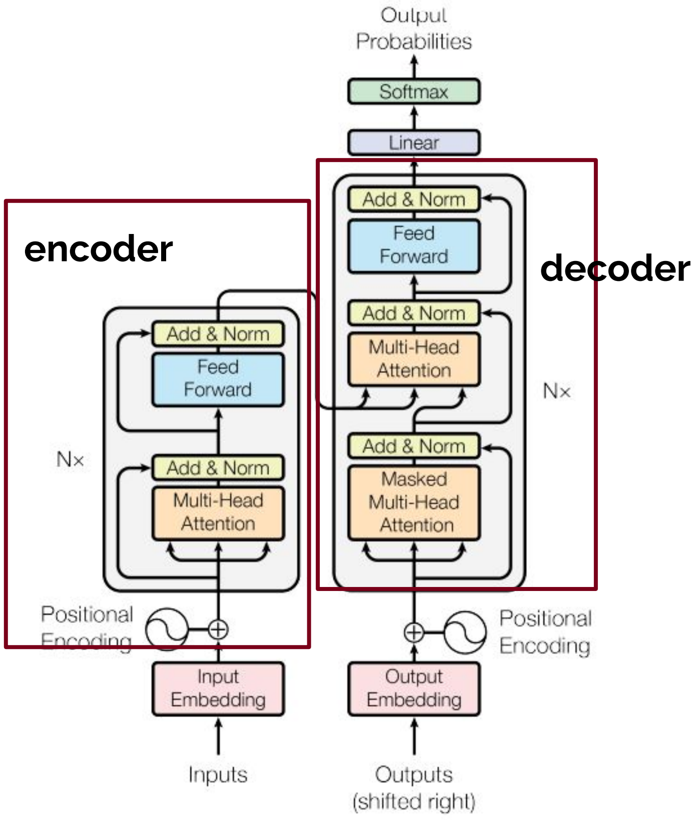


Figure 1: The Transformer - model architecture.

NIPS 2017; https://arxiv.org/abs/1706.03762

Transformers reign in NLP!

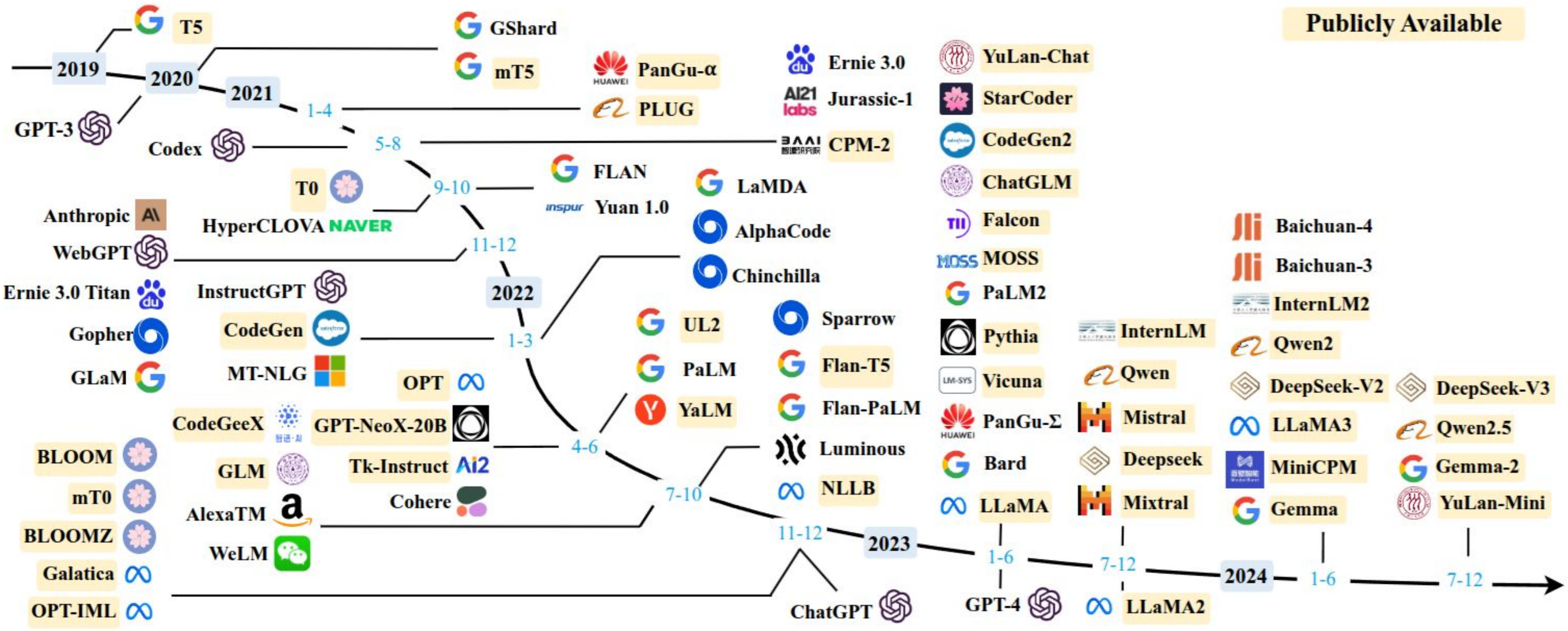


Image credit: A Survey of Large Language Models https://arxiv.org/abs/2303.18223

Transformers for everything!

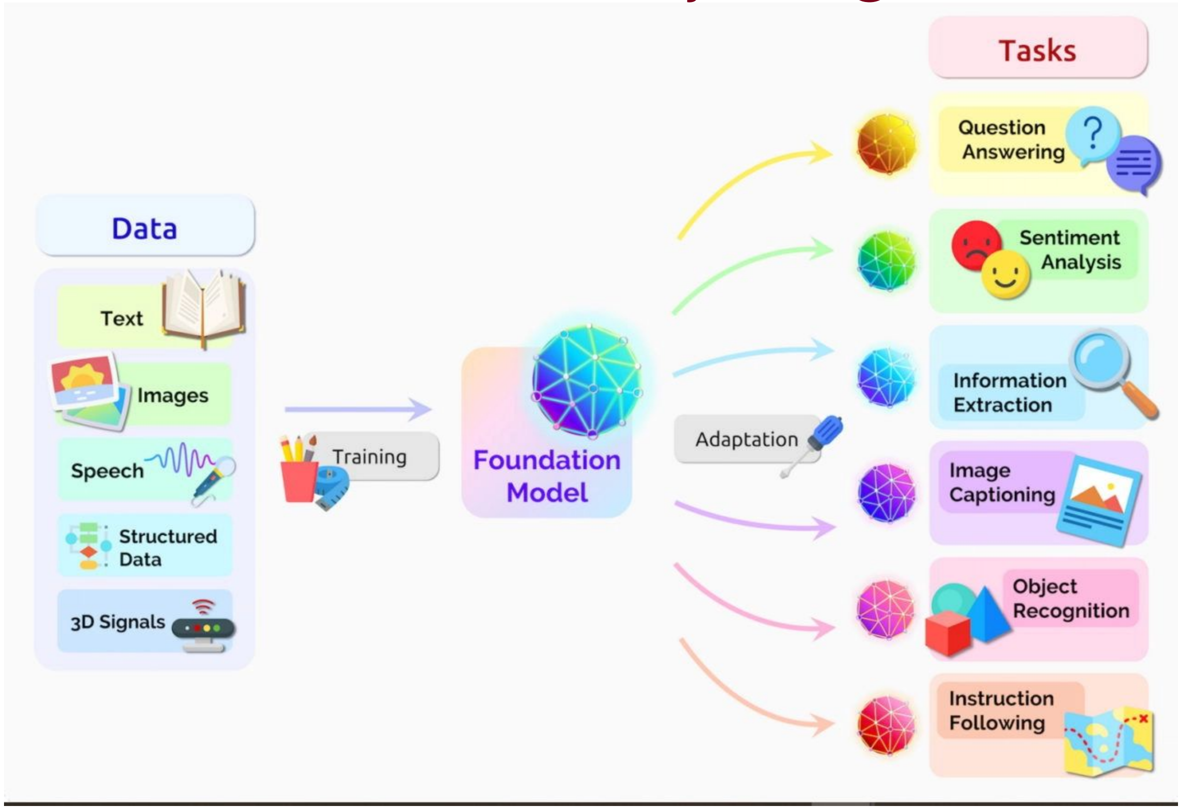


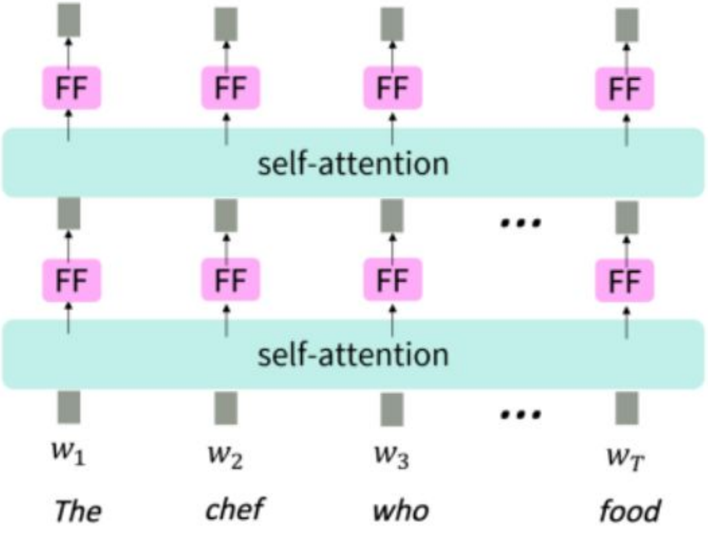
Image credit: https://blogs.nvidia.com/blog/2022/03/25/what-is-a-transformer-model/

- Transformers have been modified to deal with **almost all** kinds of structured and **unstructured** data
- Enable multimodal data integration and interaction

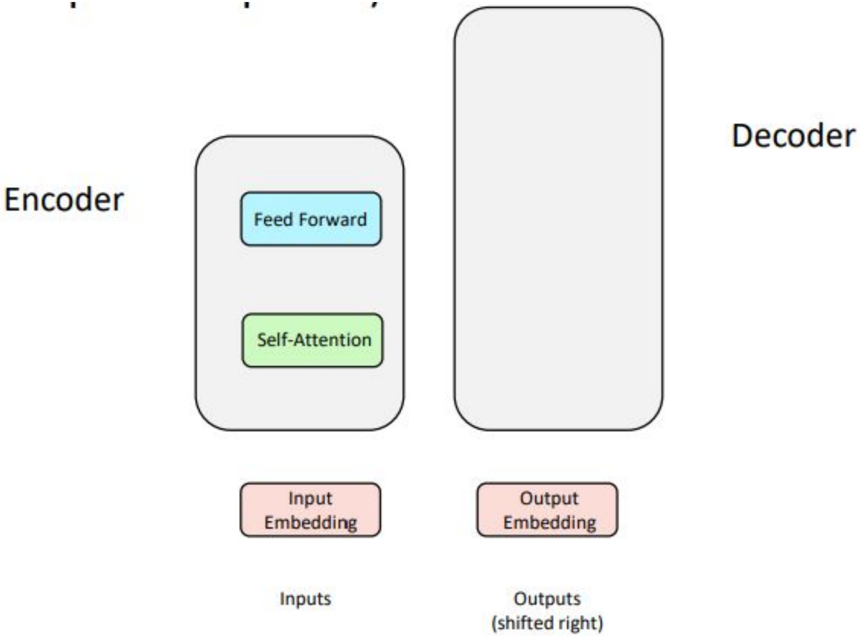
Starting from self-attention

Equation for Feed Forward Layer

$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$$



(Credit: Stanford CS231N)

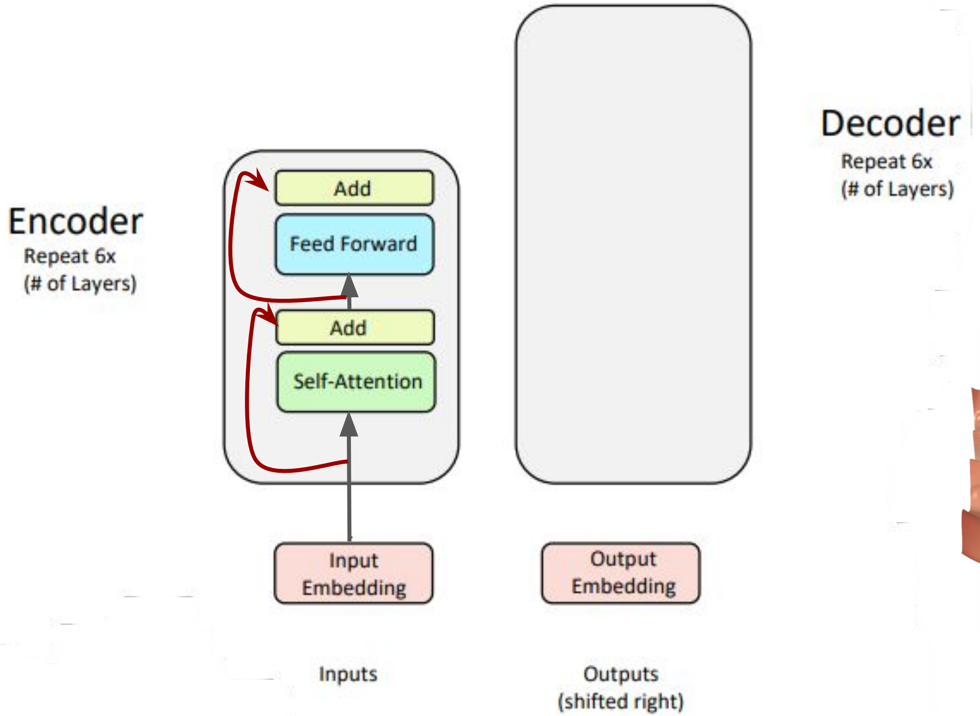


(Credit: Stanford CS231N)

Three tricks to build in depth:

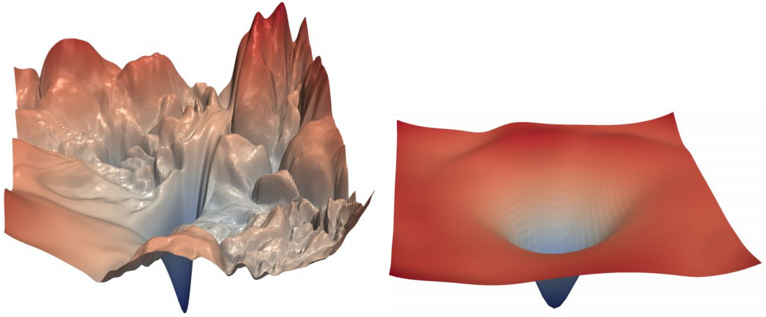
- Residual connection
- Layer normalization
- Scaled inner product attention

Trick 1: Residual connection



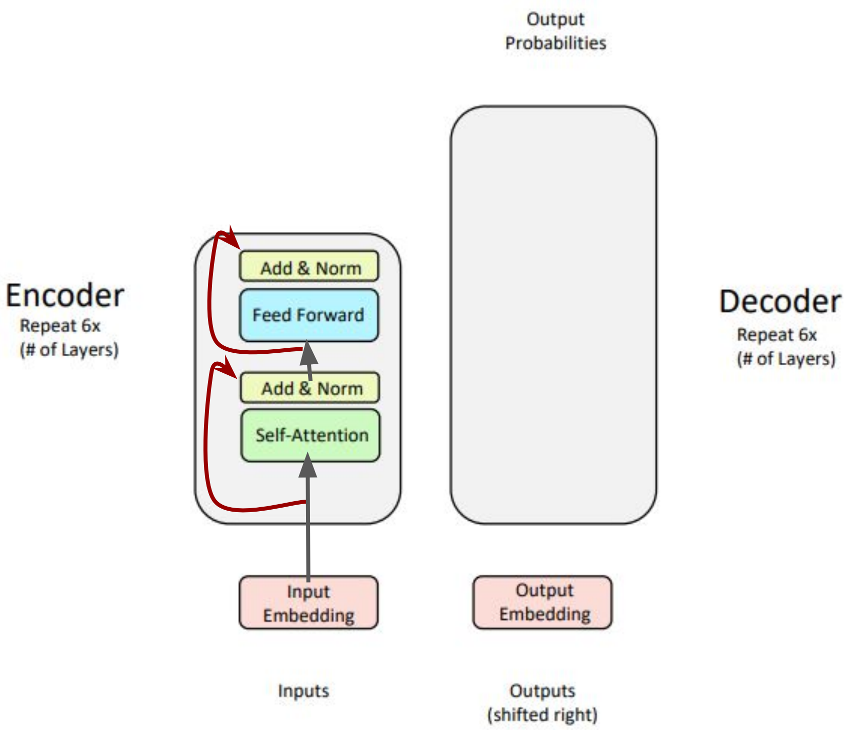
$$\boldsymbol{x}_k = F(\boldsymbol{x}_{k-1}) + \boldsymbol{x}_{k-1}$$

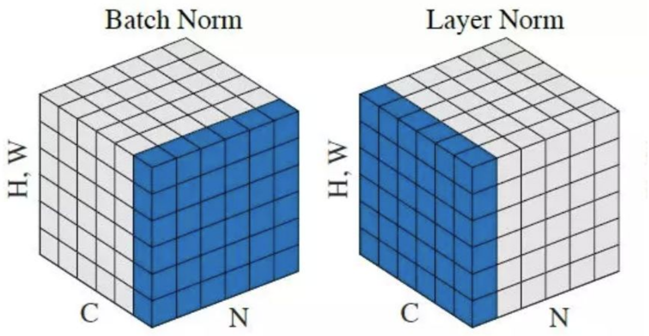
- Mitigating vanishing gradient
- Smoothing out landscape



https://arxiv.org/abs/1712.09913

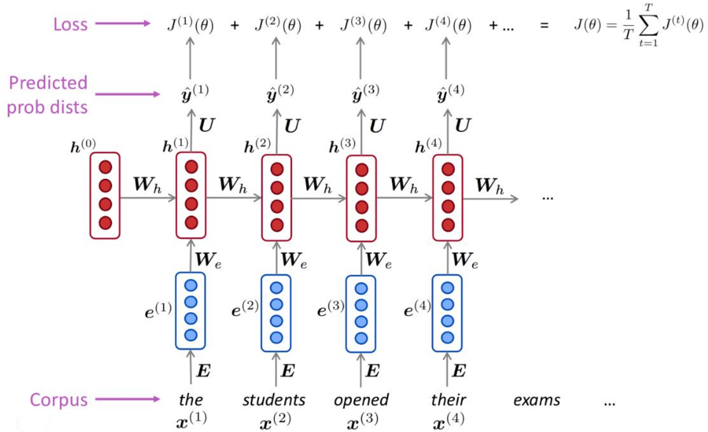
Trick 2: Layer normalization



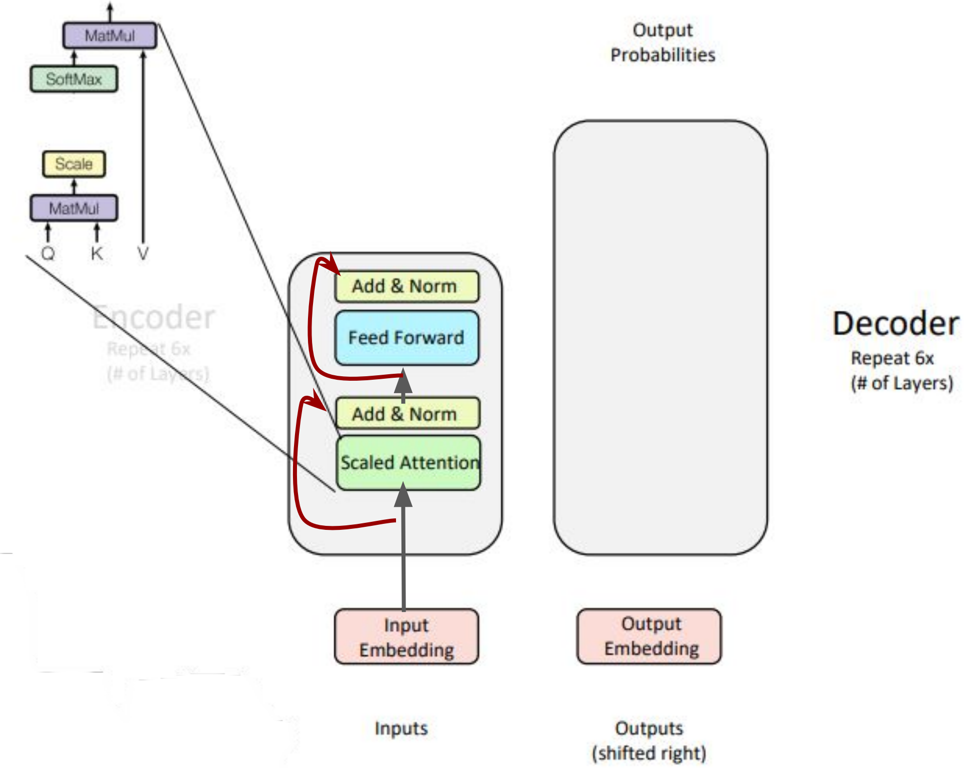


$$x^{\ell\prime} = \frac{x^\ell - \mu^\ell}{\sigma^\ell + \epsilon}$$

Why not batchnorm?



Trick 3: Scaled inner product attention



$$\text{output} = \text{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top)\boldsymbol{V}$$

- Suppose that entries of Q and K behaves like IID zero-mean, unit variance
- $\mathbb{E}\langle \boldsymbol{q}^i, \boldsymbol{k}^j \rangle = 0$ but

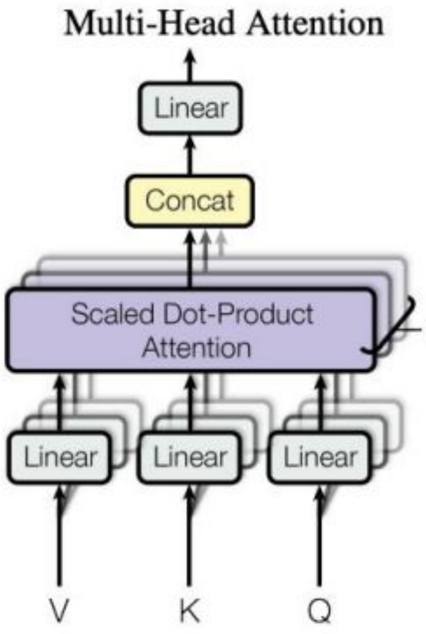
$$\text{Var}\langle \boldsymbol{q}^i, \boldsymbol{k}^j \rangle = d_k$$

This can blow up exp computation in the softmax normalization for large d_k!

Solution: normalize by standard deviation

$$\text{output} = \text{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top / \sqrt{d_k})\boldsymbol{V}$$

Multi-head attention



H blocks in parallel

[Vaswani et al. 2017]

Multiple, independent self-attention blocks in parallel

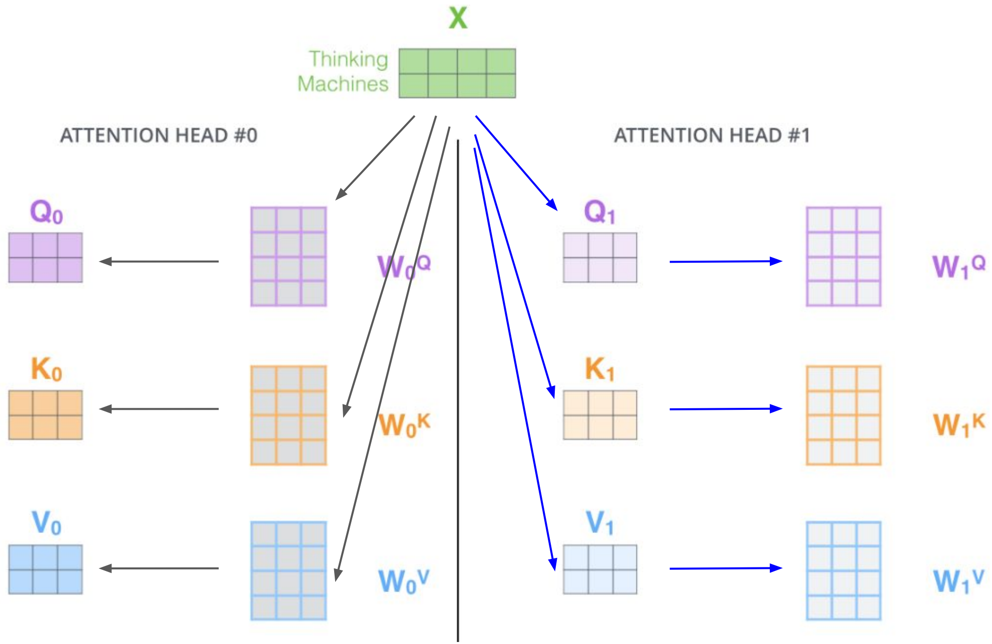


Image credit: https://jalammar.github.io/illustrated-transformer/

Intuition: allow the flexibility of capturing different kinds of "relevance"/correlations

Multi-head attention

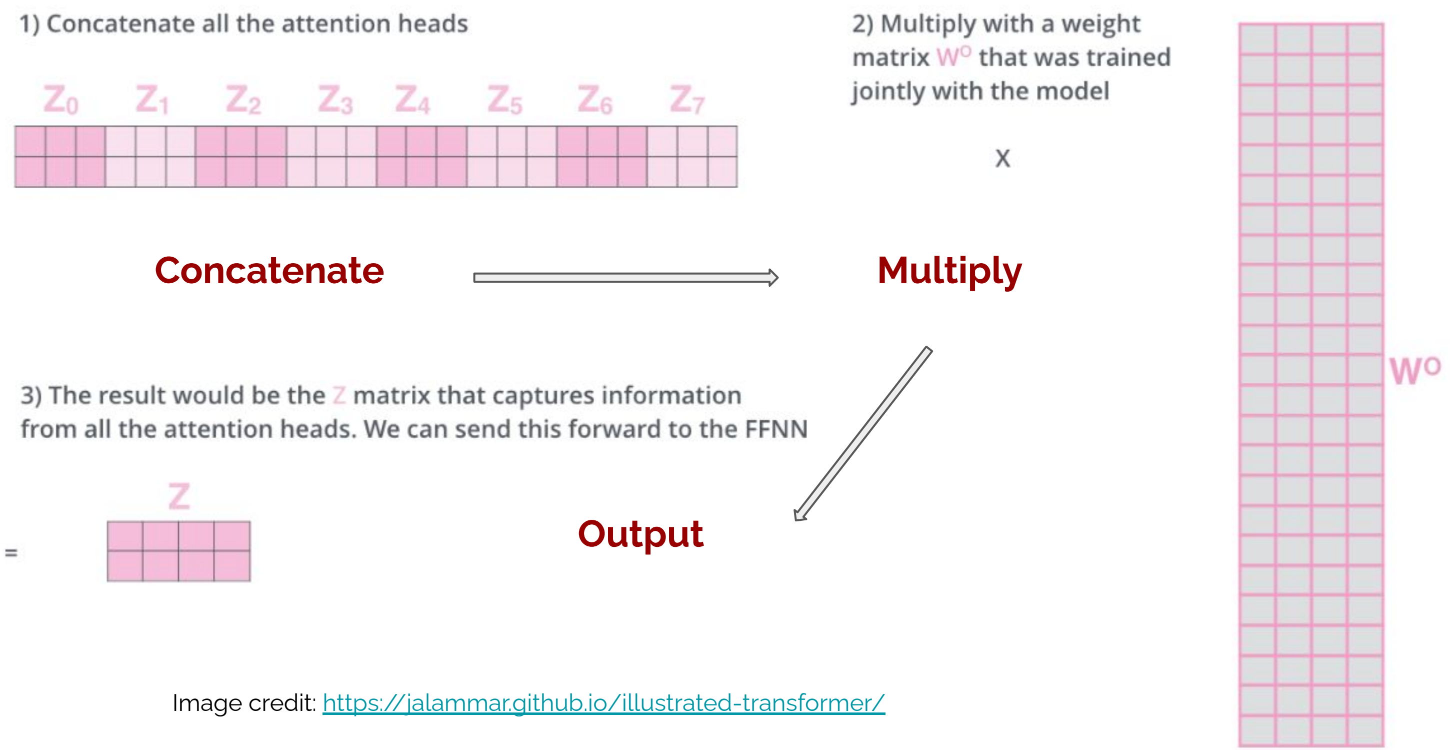


Image credit: https://jalammar.github.io/illustrated-transformer/

Multi-head attention

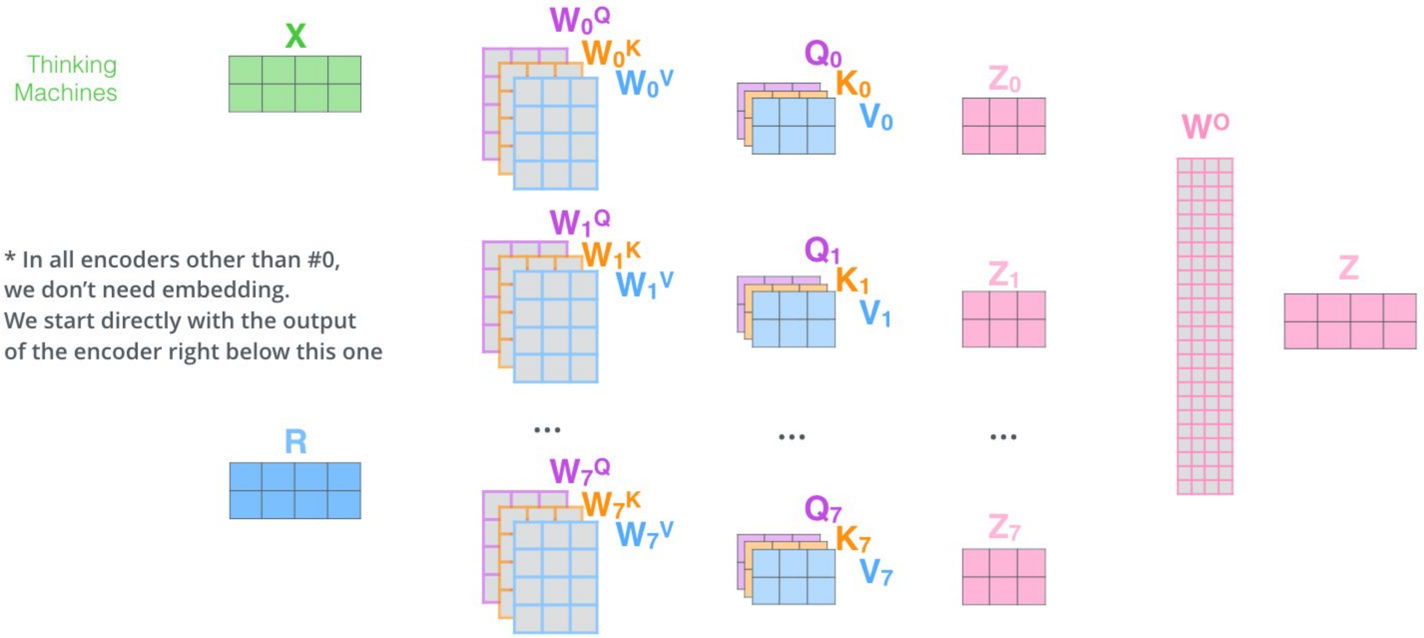
1) This is our input sentence*

2) We embed each word*

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

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

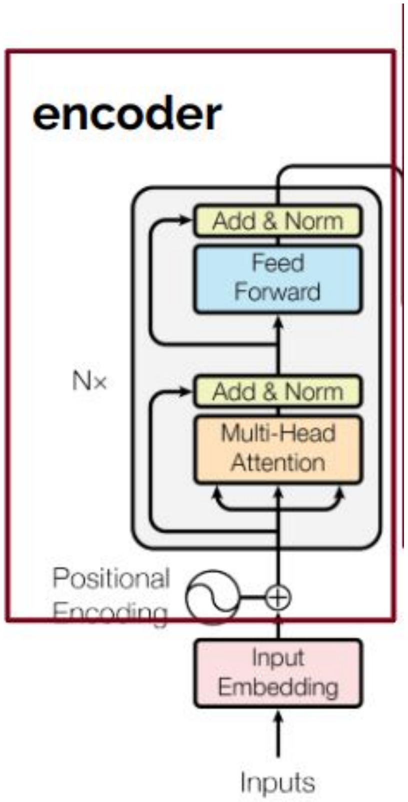
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer



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

Image credit: https://jalammar.github.io/illustrated-transformer/

Positional encoding



Does the input order matter or not?

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K, \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V$$

$$\text{output} = \text{softmax}(\boldsymbol{Q}\boldsymbol{K}^\intercal / \sqrt{d_k})\boldsymbol{V}$$

Positional encoding to break the **order invariance**

- Idea: a positional vector to (hopefully) encode the position information

  E.g., $\boldsymbol{X}_p = \boldsymbol{X} + \boldsymbol{P}, \text{ or } \boldsymbol{X}_p = [\boldsymbol{X}, \boldsymbol{P}]$

- $\boldsymbol{P}$ can be pre-defined, or made learnable

Sinusoidal positional encoding

L: sequence length
d: embedding dimension

$$\mathrm{PE}(i,\delta) = \begin{cases} \sin\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' \\ \cos\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' + 1 \end{cases}$$

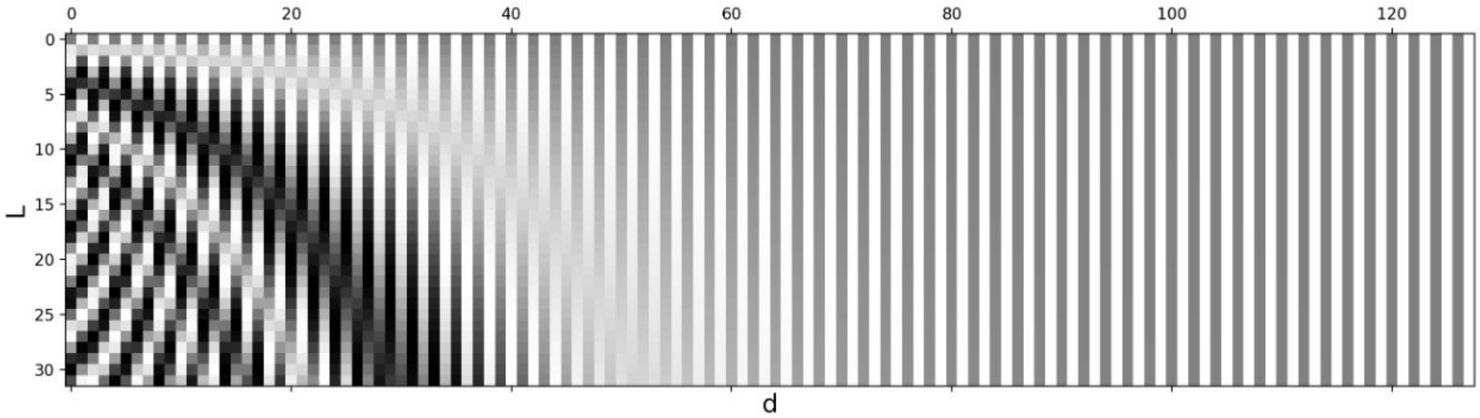


Image credit: https://lilianweng.github.io/posts/2020-04-07-the-transformer-family/

Decoder

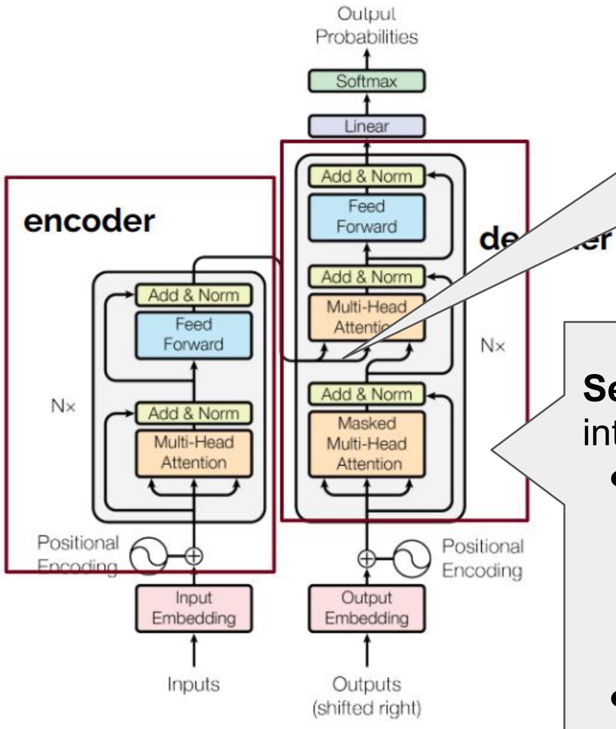
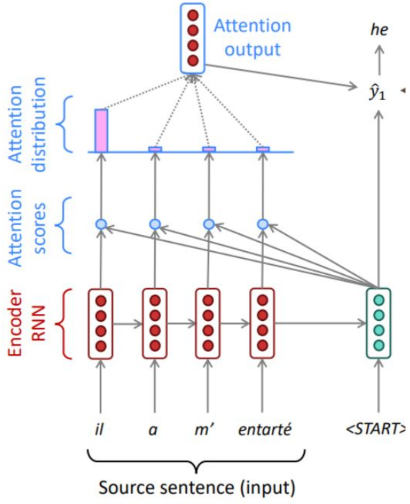


Figure 1: The Transformer - model architecture.

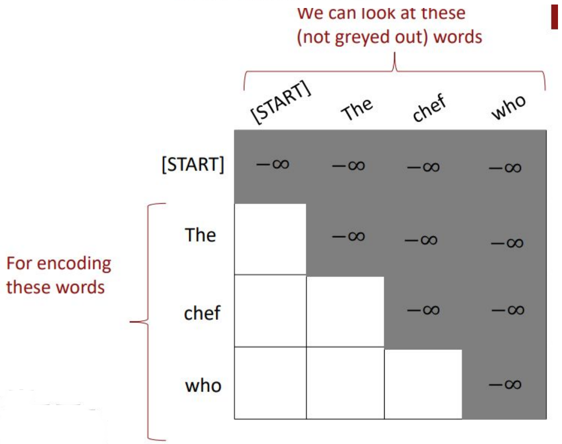
Cross-attention (to model the interaction between the encoder key-values and the current decoder query)

Self-attention (to model the interaction within itself)

- Respect the sequential nature (e.g., language modeling, assuming access to the future is cheating!)
- Masked out future tokens



(Credit: Stanford CS231N)



Attention matrix

Strong performance in machine translation

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

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	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Source: Attention Is All You Need https://arxiv.org/abs/1706.03762

Computation

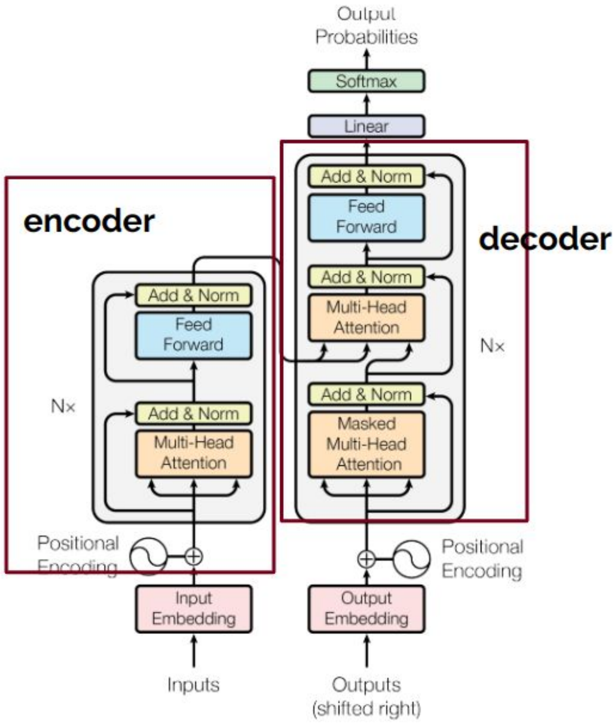


Figure 1: The Transformer - model architecture.

What's the total computation?

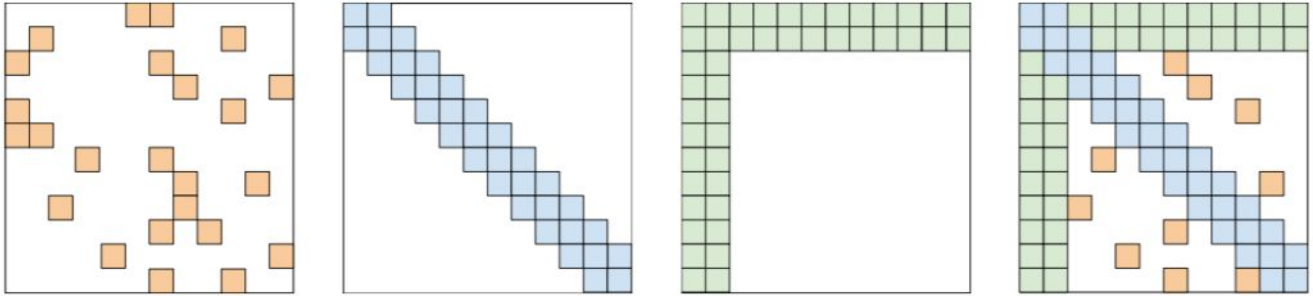
$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

$$\text{output} = \text{softmax}(QK^\top / \sqrt{d_k})V$$

$$O(T^2 d)$$

Quadratic computation vs. linear computation in RNNs (**T** is the length of each input sequence, **d** is the embedding dimension)

Computation



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Idea; building in sparsity https://arxiv.org/abs/2007.14062

Do Transformer Modifications Transfer Across Implementations and Applications?

Sharan Narang* Hyung Won Chung Yi Tay William Fedus

Thibault Fevry† Michael Matena† Karishma Malkan† Noah Fiedel

Noam Shazeer Zhenzhong Lan† Yanqi Zhou Wei Li

Nan Ding Jake Marcus Adam Roberts Colin Raffel†

But not much consistent improvement so far
https://arxiv.org/abs/2102.11972

Large language models (LLMs)

Transformers reign in NLP!

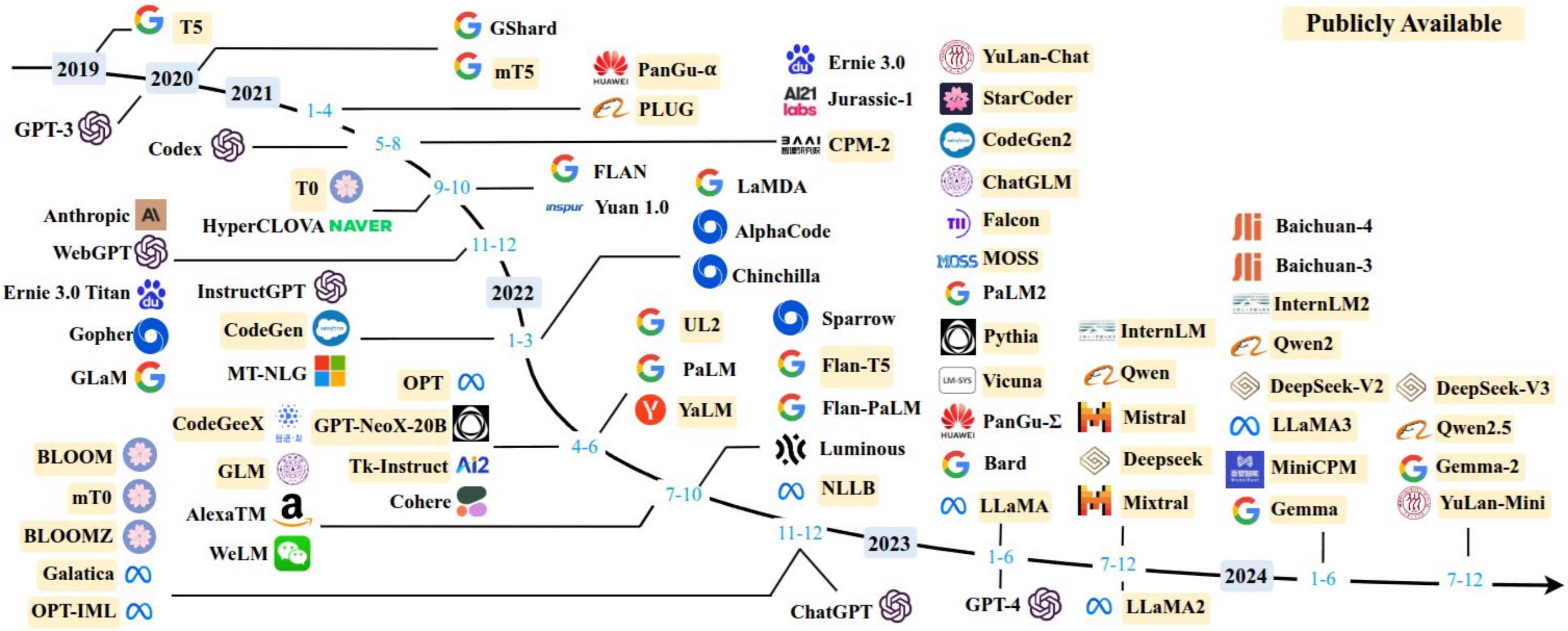


Image credit: A Survey of Large Language Models https://arxiv.org/abs/2303.18223

LLMs: large models trained on large datasets

	Model	Release Time	Size (B)
Publicly Available	T5 [73]	Oct-2019	11
	mT5 [74]	Oct-2020	13
	PanGu-α [75]	Apr-2021	13*
	CPM-2 [76]	Jun-2021	198
	T0 [28]	Oct-2021	11
	CodeGen [77]	Mar-2022	16
	GPT-NeoX-20B [78]	Apr-2022	20
	Tk-Instruct [79]	Apr-2022	11
	UL2 [80]	May-2022	20
	OPT [81]	May-2022	175
	NLLB [82]	Jul-2022	54.5
	CodeGeeX [83]	Sep-2022	13
	GLM [84]	Oct-2022	130
	Flan-T5 [64]	Oct-2022	11
	BLOOM [69]	Nov-2022	176
	mT0 [85]	Nov-2022	13
	Galactica [35]	Nov-2022	120
	BLOOMZ [85]	Nov-2022	176
	OPT-IML [86]	Dec-2022	175
	LLaMA [57]	Feb-2023	65
	Pythia [87]	Apr-2023	12
	CodeGen2 [88]	May-2023	16
	StarCoder [89]	May-2023	15.5
	LLaMA2 [90]	Jul-2023	70
Closed Source	GPT-3 [55]	May-2020	175
	GShard [91]	Jun-2020	600
	Codex [92]	Jul-2021	12
	ERNIE 3.0 [93]	Jul-2021	10
	Jurassic-1 [94]	Aug-2021	178
	HyperCLOVA [95]	Sep-2021	82
	FLAN [62]	Sep-2021	137
	Yuan 1.0 [96]	Oct-2021	245
	Anthropic [97]	Dec-2021	52
	WebGPT [72]	Dec-2021	175
	Gopher [59]	Dec-2021	280
	ERNIE 3.0 Titan [98]	Dec-2021	260
	GLaM [99]	Dec-2021	1200
	LaMDA [63]	Jan-2022	137
	MT-NLG [100]	Jan-2022	530
	AlphaCode [101]	Feb-2022	41
	InstructGPT [61]	Mar-2022	175
	Chinchilla [34]	Mar-2022	70
	PaLM [56]	Apr-2022	540
	AlexaTM [102]	Aug-2022	20
	Sparrow [103]	Sep-2022	70
	WeLM [104]	Sep-2022	10
	U-PaLM [105]	Oct-2022	540
	Flan-PaLM [64]	Oct-2022	540
	Flan-U-PaLM [64]	Oct-2022	540
	GPT-4 [46]	Mar-2023	-
	PanGu-Σ [106]	Mar-2023	1085
	PaLM2 [107]	May-2023	16

Image credit: A Survey of Large Language Models https://arxiv.org/abs/2303.18223

LLMs: large models trained on large datasets

TABLE 2: Statistics of commonly-used data sources.

Corpora	Size	Source	Latest Update Time
BookCorpus [138]	5GB	Books	Dec-2015
Gutenberg [139]	-	Books	Dec-2021
C4 [73]	800GB	CommonCrawl	Apr-2019
CC-Stories-R [140]	31GB	CommonCrawl	Sep-2019
CC-NEWS [27]	78GB	CommonCrawl	Feb-2019
REALNEWs [141]	120GB	CommonCrawl	Apr-2019
OpenWebText [142]	38GB	Reddit links	Mar-2023
Pushift.io [143]	2TB	Reddit links	Mar-2023
Wikipedia [144]	21GB	Wikipedia	Mar-2023
BigQuery [145]	-	Codes	Mar-2023
the Pile [146]	800GB	Other	Dec-2020
ROOTS [147]	1.6TB	Other	Jun-2022

Image credit: A Survey of Large Language Models https://arxiv.org/abs/2303.18223

Two crucial technical steps toward LLMs

- **Pretraining**
- **Finetuning (Adaptation)**

Recall transfer learning?

Pretraining: data collection

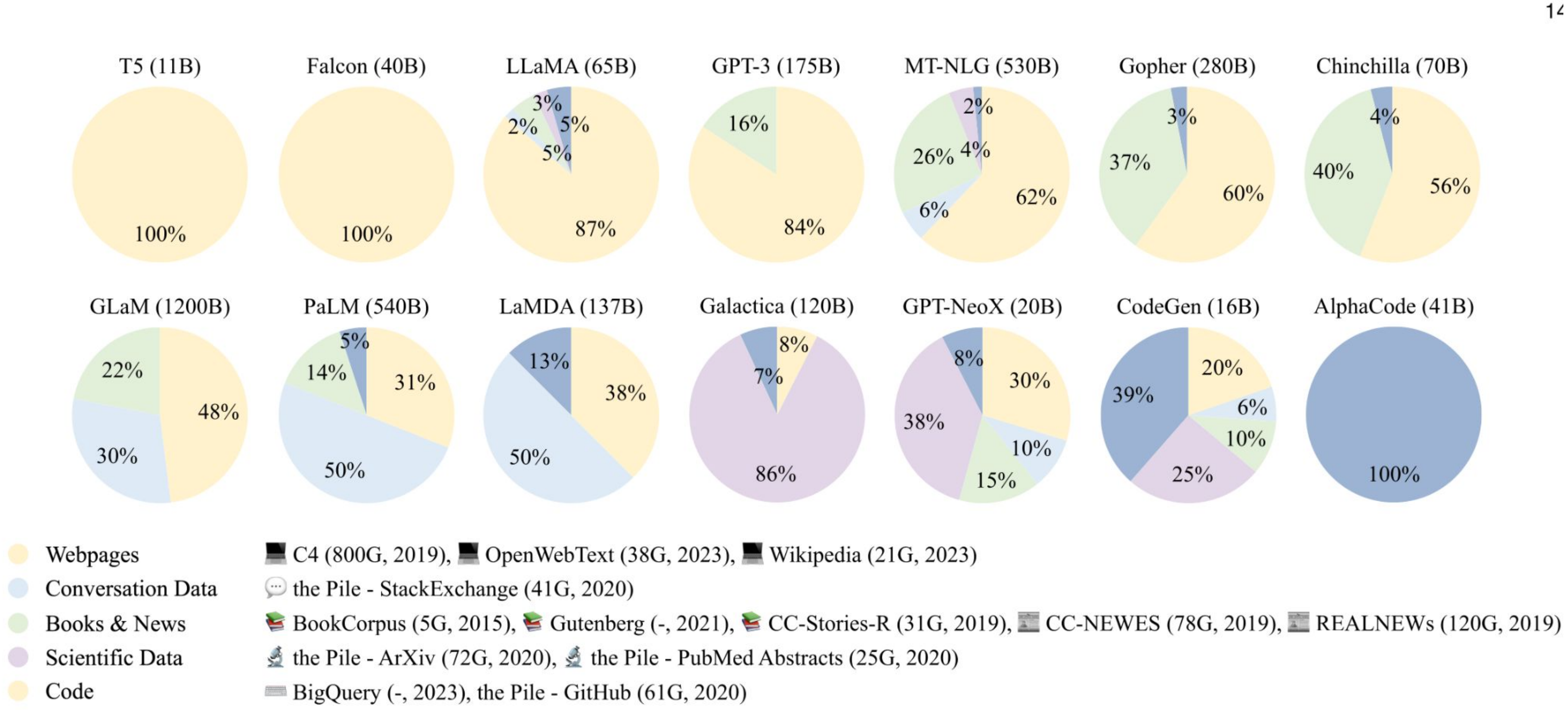


Image credit: A Survey of Large Language Models https://arxiv.org/abs/2303.18223

Pretraining: data collection

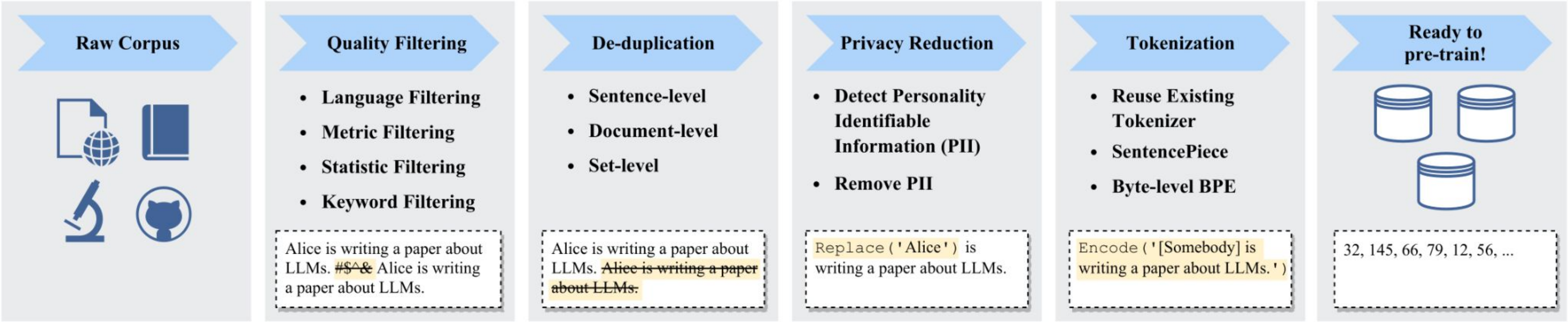
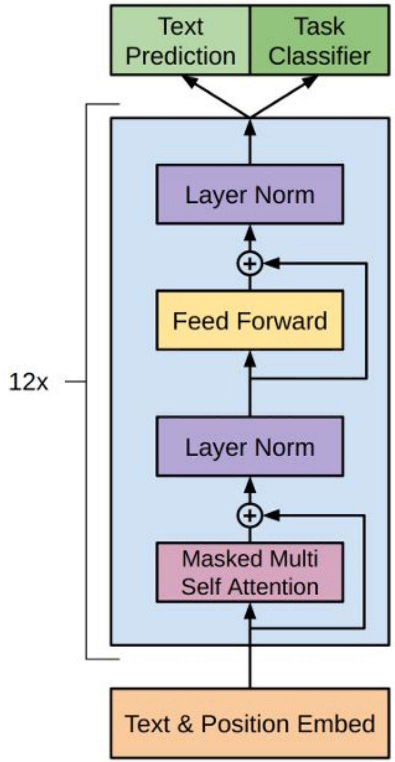


Fig. 6: An illustration of a typical data preprocessing pipeline for pre-training large language models.

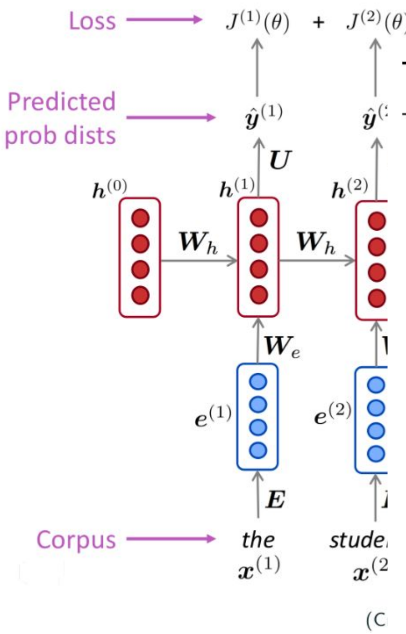
Image credit: A Survey of Large Language Models https://arxiv.org/abs/2303.18223

Pretraining: architecture & task



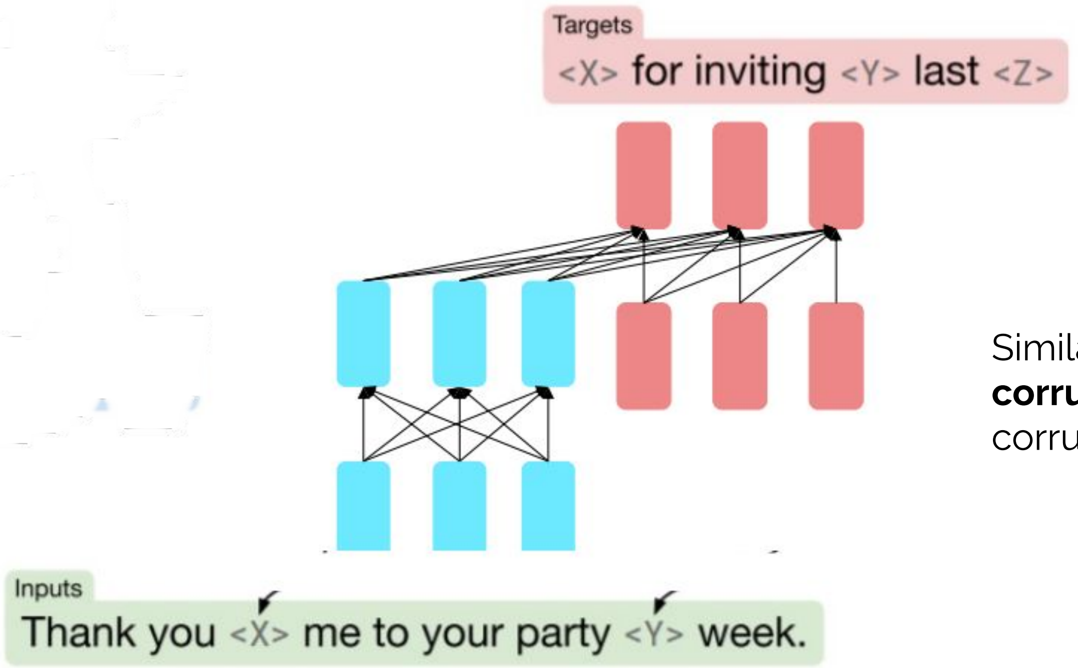
Most popular: (transformer-based) **decoder-only** architectures pretrained on **language modeling**, i.e. model

$$\mathbb{P}\left[\boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)}\right]$$



Model	Category	Size
GPT3 [55]	Causal decoder	175B
PanGU- α [75]	Causal decoder	207B
OPT [81]	Causal decoder	175B
PaLM [56]	Causal decoder	540B
BLOOM [69]	Causal decoder	176B
MT-NLG [100]	Causal decoder	530B
Gopher [59]	Causal decoder	280B
Chinchilla [34]	Causal decoder	70B
Galactica [35]	Causal decoder	120B
LaMDA [63]	Causal decoder	137B
Jurassic-1 [94]	Causal decoder	178B
LLaMA [57]	Causal decoder	65B
LLaMA 2 [90]	Causal decoder	70B
Falcon [127]	Causal decoder	40B
GLM-130B [84]	Prefix decoder	130B
T5 [73]	Encoder-decoder	11B

Pretraining: architecture & task — alternative



Similar to pretraining encoder, **corruption removal**! (called span corruption)

Pretraining: architecture details

Configuration	Method	Equation
Normalization position	Post Norm [22]	$\mathrm{Norm}(\mathbf{x}+\mathrm{Sublayer}(\mathbf{x}))$
	Pre Norm [26]	$\mathbf{x} + \mathrm{Sublayer}(\mathrm{Norm}(\mathbf{x}))$
	Sandwich Norm [201]	$\mathbf{x} + \mathrm{Norm}(\mathrm{Sublayer}(\mathrm{Norm}(\mathbf{x})))$
Normalization method	LayerNorm [202]	$\frac{\mathbf{x}-\mu}{\sqrt{\sigma}} \cdot \gamma + \beta, \quad \mu = \frac{1}{d}\sum_{i=1}^{d} x_i, \quad \sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu))^2}$
	RMSNorm [203]	$\frac{\mathbf{x}}{\mathrm{RMS}(\mathbf{x})} \cdot \gamma, \quad \mathrm{RMS}(\mathbf{x}) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}$
	DeepNorm [204]	$\mathrm{LayerNorm}(\alpha \cdot \mathbf{x} + \mathrm{Sublayer}(\mathbf{x}))$
Activation function	ReLU [205]	$\mathrm{ReLU}(\mathbf{x}) = \max(\mathbf{x}, \mathbf{0})$
	GeLU [206]	$\mathrm{GeLU}(\mathbf{x}) = 0.5\mathbf{x} \otimes [1 + \mathrm{erf}(\mathbf{x}/\sqrt{2})], \quad \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2} dt$
	Swish [207]	$\mathrm{Swish}(\mathbf{x}) = \mathbf{x} \otimes \mathrm{sigmoid}(\mathbf{x})$
	SwiGLU [208]	$\mathrm{SwiGLU}(\mathbf{x}_1, \mathbf{x}_2) = \mathrm{Swish}(\mathbf{x_1}) \otimes \mathbf{x_2}$
	GeGLU [208]	$\mathrm{GeGLU}(\mathbf{x}_1, \mathbf{x}_2) = \mathrm{GeLU}(\mathbf{x_1}) \otimes \mathbf{x_2}$
Position embedding	Absolute [22]	$\mathbf{x}_i = \mathbf{x}_i + \mathbf{p}_i$
	Relative [73]	$A_{ij} = \mathbf{W}_q \mathbf{x}_i \mathbf{x}_j^T \mathbf{W}_k^T + r_{i-j}$
	RoPE [209]	$A_{ij} = \mathbf{W}_q \mathbf{x}_i \mathbf{R}_{\theta,i-j} \mathbf{x}_j^T \mathbf{W}_k^T$
	Alibi [210]	$A_{ij} = \mathbf{W}_q \mathbf{x}_i \mathbf{R}_{\theta,i-j} \mathbf{x}_j^T \mathbf{W}_k^T A_{ij} = \mathbf{W}_q \mathbf{x}_i \mathbf{x}_j^T \mathbf{W}_k^T - m(i-j)$

Image credit: A Survey of Large Language Models https://arxiv.org/abs/2303.18223

Pretraining: optimization details

TABLE 5: Detailed optimization settings of several existing LLMs.

Model	Batch Size (#tokens)	Learning Rate	Warmup	Decay Method	Optimizer	Precision Type	Weight Decay	Grad Clip	Dropout
GPT3 (175B)	32K→3.2M	6×10^{-5}	yes	cosine decay to 10%	Adam	FP16	0.1	1.0	-
PanGu-α (200B)	-	2×10^{-5}	-	-	Adam	-	0.1	-	-
OPT (175B)	2M	1.2×10^{-4}	yes	manual decay	AdamW	FP16	0.1	-	0.1
PaLM (540B)	1M→4M	1×10^{-2}	no	inverse square root	Adafactor	BF16	lr^2	1.0	0.1
BLOOM (176B)	4M	6×10^{-5}	yes	cosine decay to 10%	Adam	BF16	0.1	1.0	0.0
MT-NLG (530B)	64 K→3.75M	5×10^{-5}	yes	cosine decay to 10%	Adam	BF16	0.1	1.0	-
Gopher (280B)	3M→6M	4×10^{-5}	yes	cosine decay to 10%	Adam	BF16	-	1.0	-
Chinchilla (70B)	1.5M→3M	1×10^{-4}	yes	cosine decay to 10%	AdamW	BF16	-	-	-
Galactica (120B)	2M	7×10^{-6}	yes	linear decay to 10%	AdamW	-	0.1	1.0	0.1
LaMDA (137B)	256K	-	-	-	-	BF16	-	-	-
Jurassic-1 (178B)	32 K→3.2M	6×10^{-5}	yes	-	-	-	-	-	-
LLaMA (65B)	4M	1.5×10^{-4}	yes	cosine decay to 10%	AdamW	-	0.1	1.0	-
LLaMA 2 (70B)	4M	1.5×10^{-4}	yes	cosine decay to 10%	AdamW	-	0.1	1.0	-
Falcon (40B)	2M	1.85×10^{-4}	yes	cosine decay to 10%	AdamW	BF16	0.1	-	-
GLM (130B)	0.4M→8.25M	8×10^{-5}	yes	cosine decay to 10%	AdamW	FP16	0.1	1.0	0.1
T5 (11B)	64K	1×10^{-2}	no	inverse square root	AdaFactor	-	-	-	0.1
ERNIE 3.0 Titan (260B)	-	1×10^{-4}	-	-	Adam	FP16	0.1	1.0	-
PanGu-Σ (1.085T)	0.5M	2×10^{-5}	yes	-	Adam	FP16	-	-	-

Image credit: A Survey of Large Language Models https://arxiv.org/abs/2303.18223

Supervised adaptation—instruction tuning

TABLE 6: A detailed list of available collections for instruction tuning.

Categories	Collections	Time	#Examples
Task	Nat. Inst. [264]	Apr-2021	193K
	FLAN [62]	Sep-2021	4.4M
	P3 [265]	Oct-2021	12.1M
	Super Nat. Inst. [79]	Apr-2022	5M
	MVPCorpus [266]	Jun-2022	41M
	xP3 [85]	Nov-2022	81M
	OIG[22]	Mar-2023	43M
Chat	HH-RLHF [267]	Apr-2022	160K
	HC3 [268]	Jan-2023	87K
	ShareGPT[23]	Mar-2023	90K
	Dolly[24]	Apr-2023	15K
	OpenAssistant [269]	Apr-2023	161K
Synthetic	Self-Instruct [129]	Dec-2022	82K
	Alpaca [123]	Mar-2023	52K
	Guanaco[25]	Mar-2023	535K
	Baize [270]	Apr-2023	158K
	BELLE [271]	Apr-2023	1.5M

Image credit: A Survey of Large Language Models
https://arxiv.org/abs/2303.18223

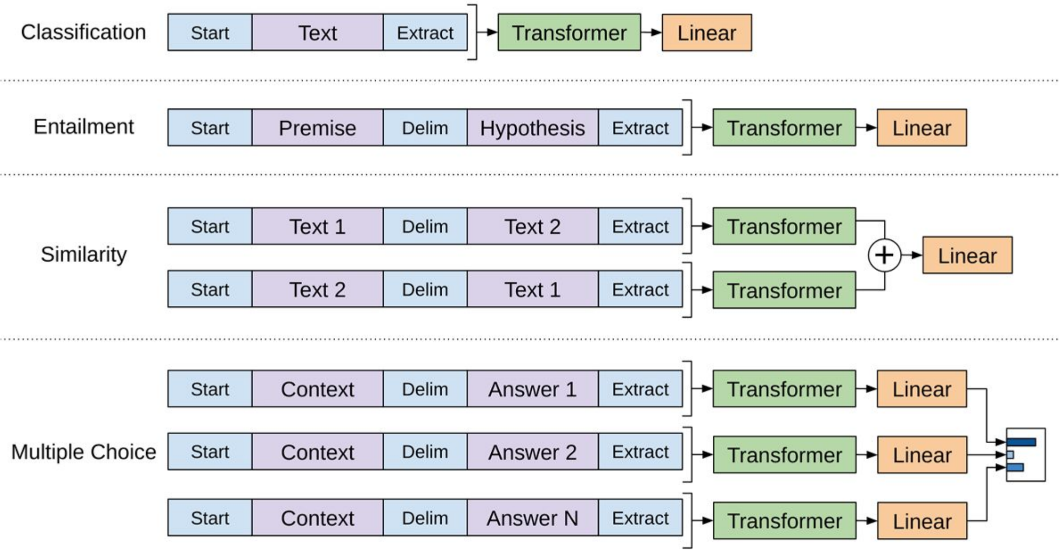


Image credit: Improving Language Understanding by Generative Pre-Training
https://gwern.net/doc/www/s3-us-west-2.amazonaws.com/d73fdc5ffa8627bce4
.pdf

Constructing the instruction sets

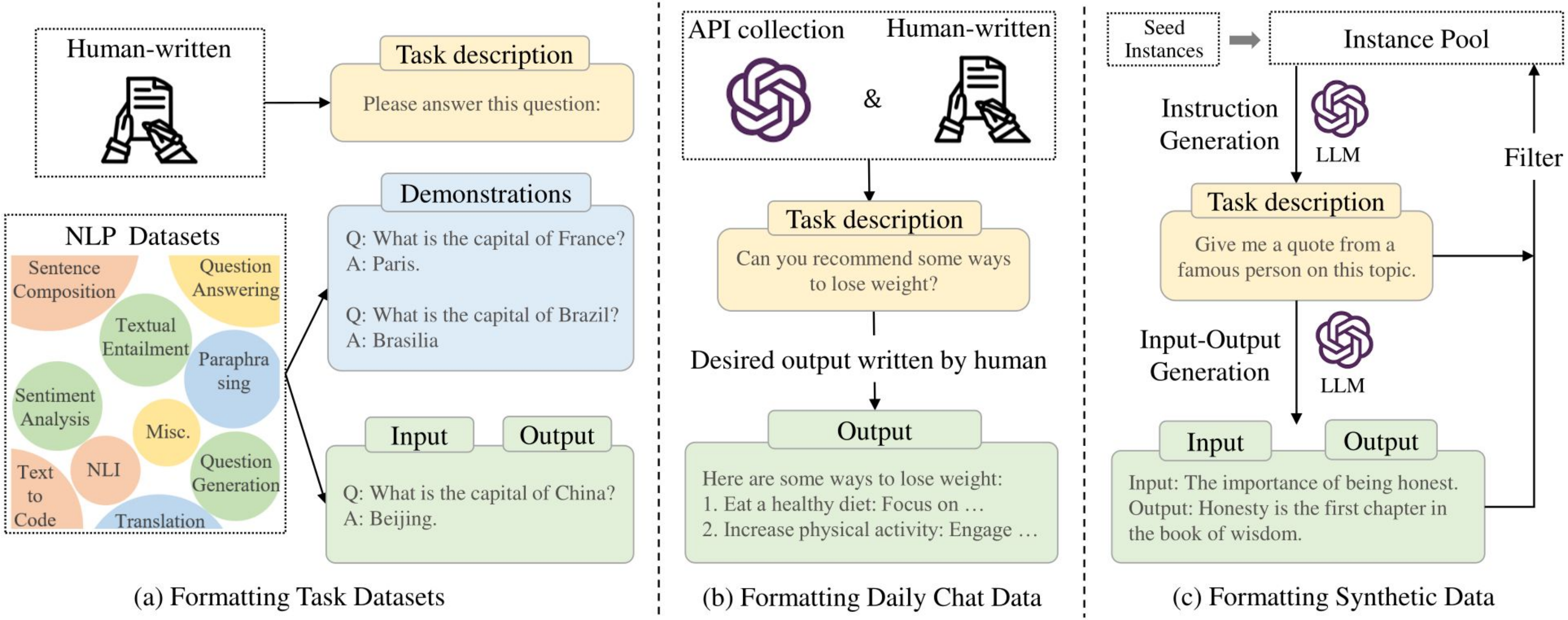
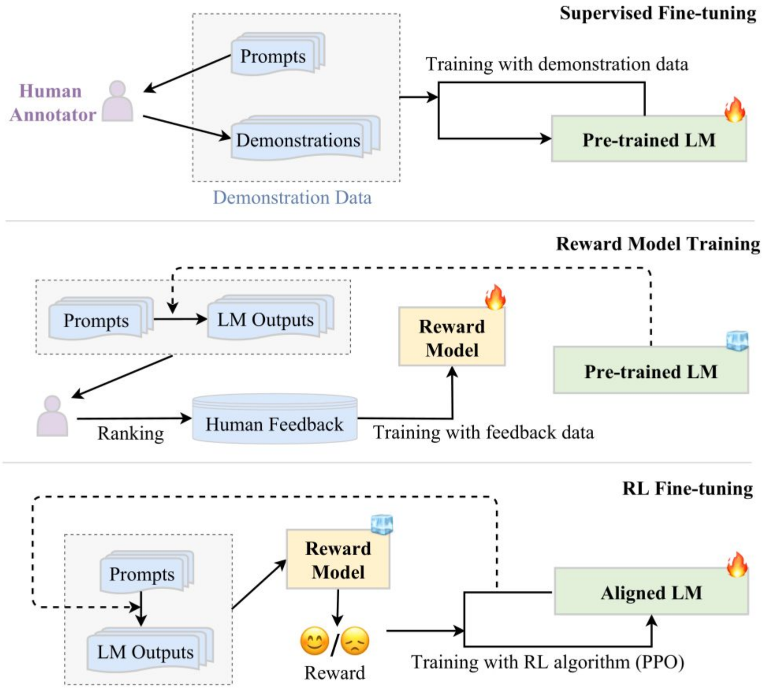


Fig. 9: An illustration of instance formatting and three different methods for constructing the instruction-formatted instances.

Image credit: A Survey of Large Language Models https://arxiv.org/abs/2303.18223

Supervised adaptation—alignment tuning



Make sure the output is aligned with human values and not harmful

Reinforcement learning with human feedback (RLHF)

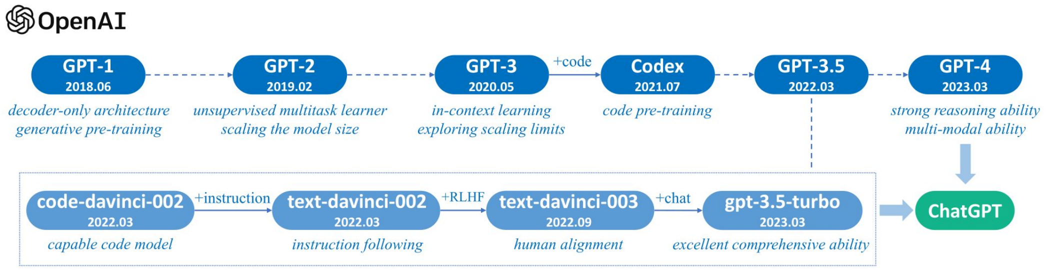


Fig. 3: A brief illustration for the technical evolution of GPT-series models. We plot this figure mainly based on the papers, blog articles and official APIs from OpenAI. Here, *solid lines* denote that there exists an explicit evidence (*e.g.*, the official statement that a new model is developed based on a base model) on the evolution path between two models, while *dashed lines* denote a relatively weaker evolution relation.

Image credit: A Survey of Large Language Models https://arxiv.org/abs/2303.18223

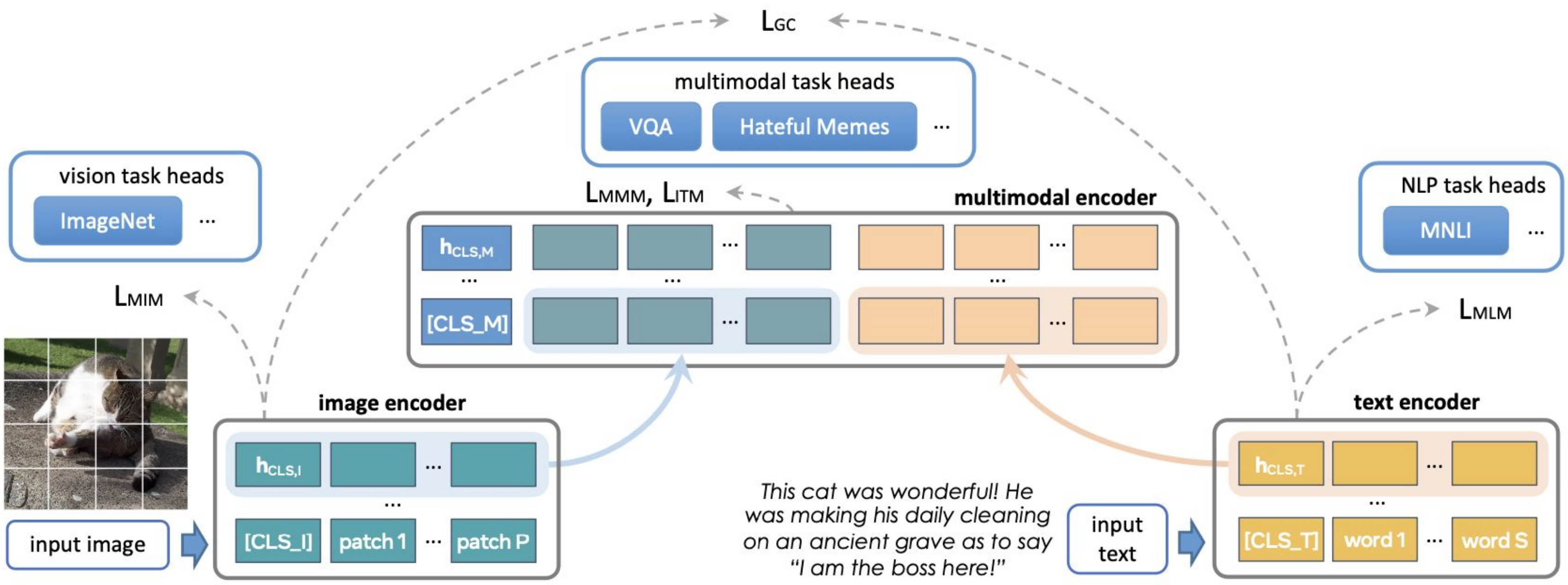
Transformers for other domains

Vision Transformers

Transformers | Davide Coccomini | 2021

https://en.wikipedia.org/wiki/Vision_transformer

Multimodal foundation models



https://pytorch.org/blog/scaling-multimodal-foundation-models-in-torchmultimodal-with-pytorch-distributed/