

Transformer

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

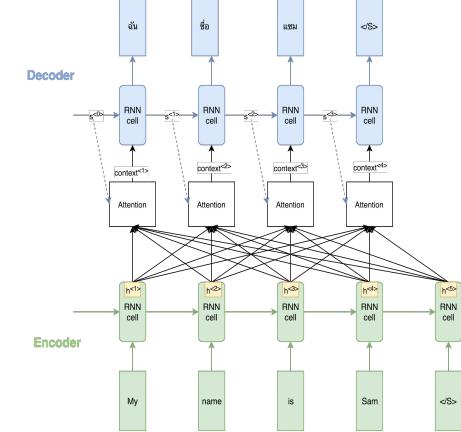
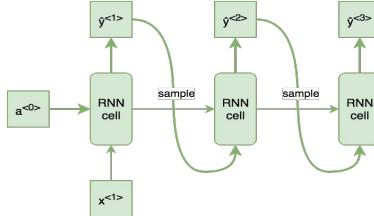
Llion Jones*
Google Research
llion@google.com

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

Lukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

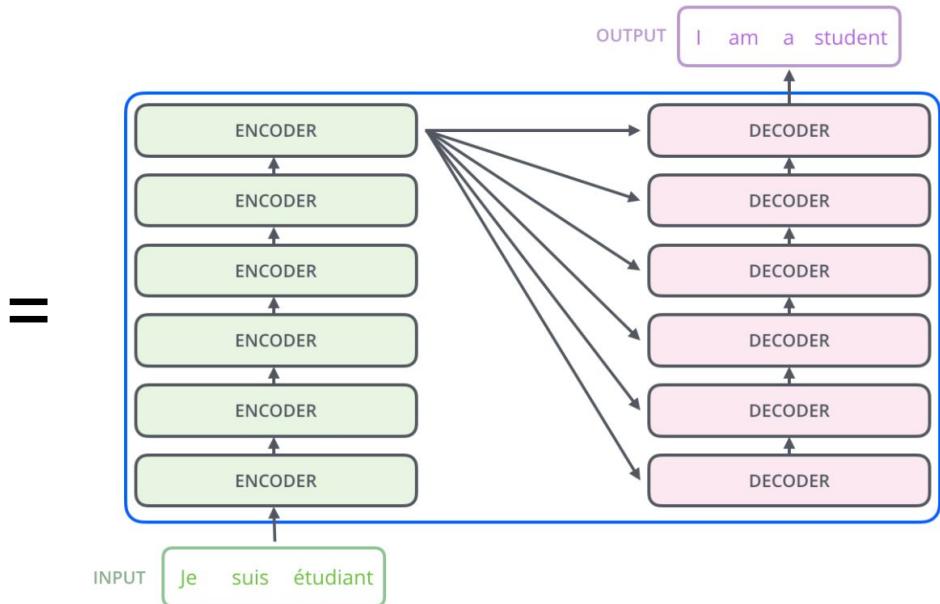
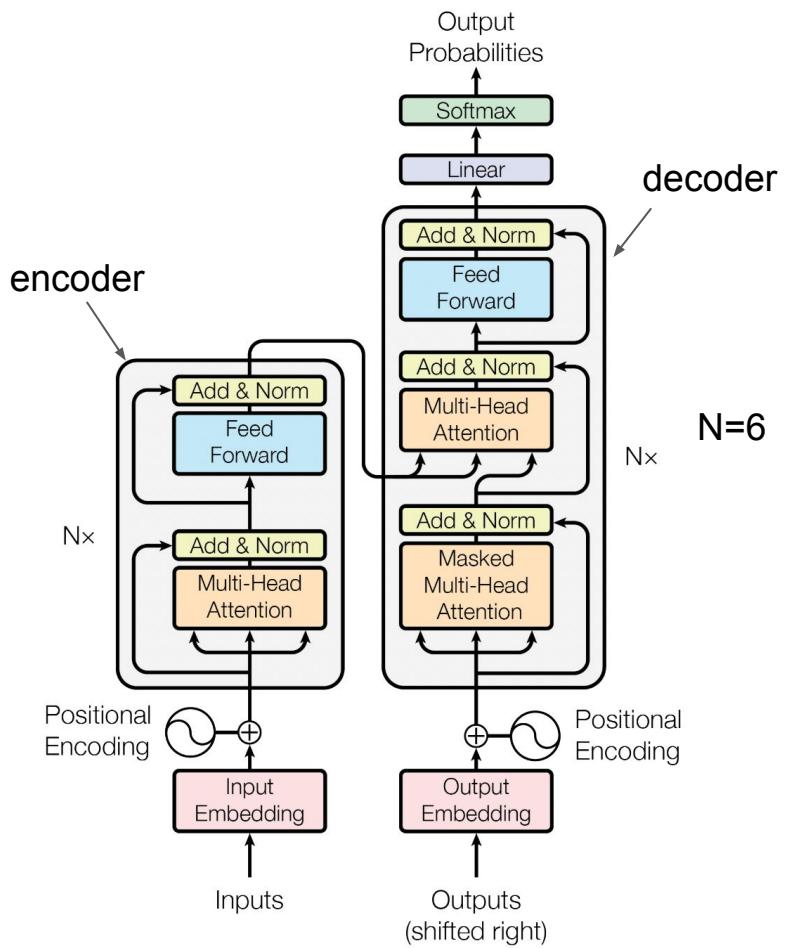
The Transformer



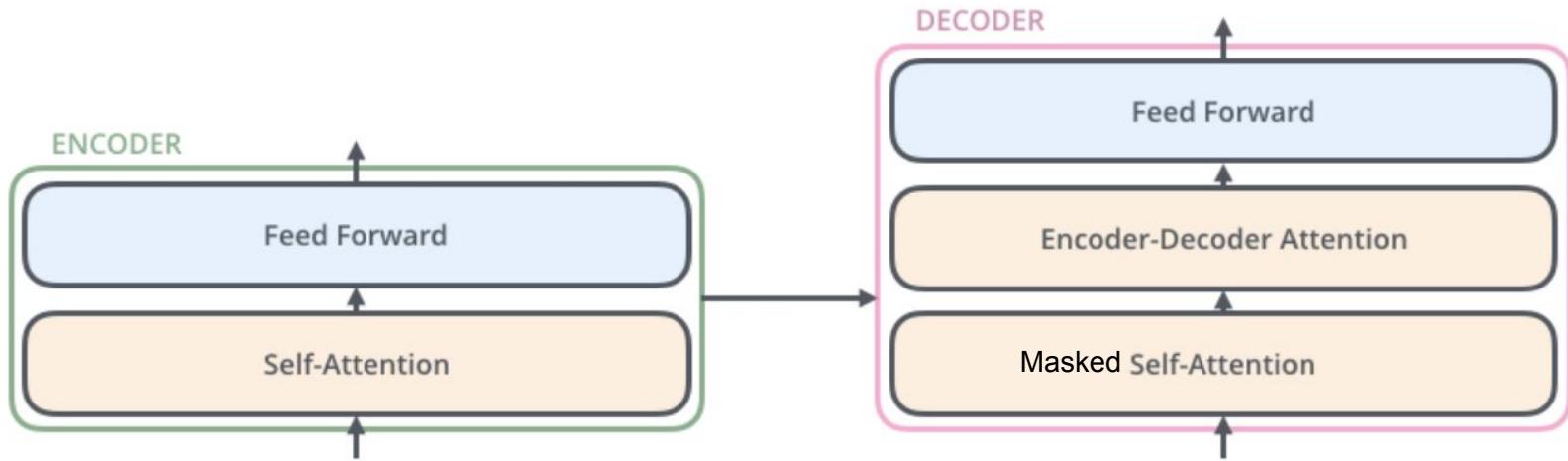
Only rely on the attention mechanism - **No RNN!**

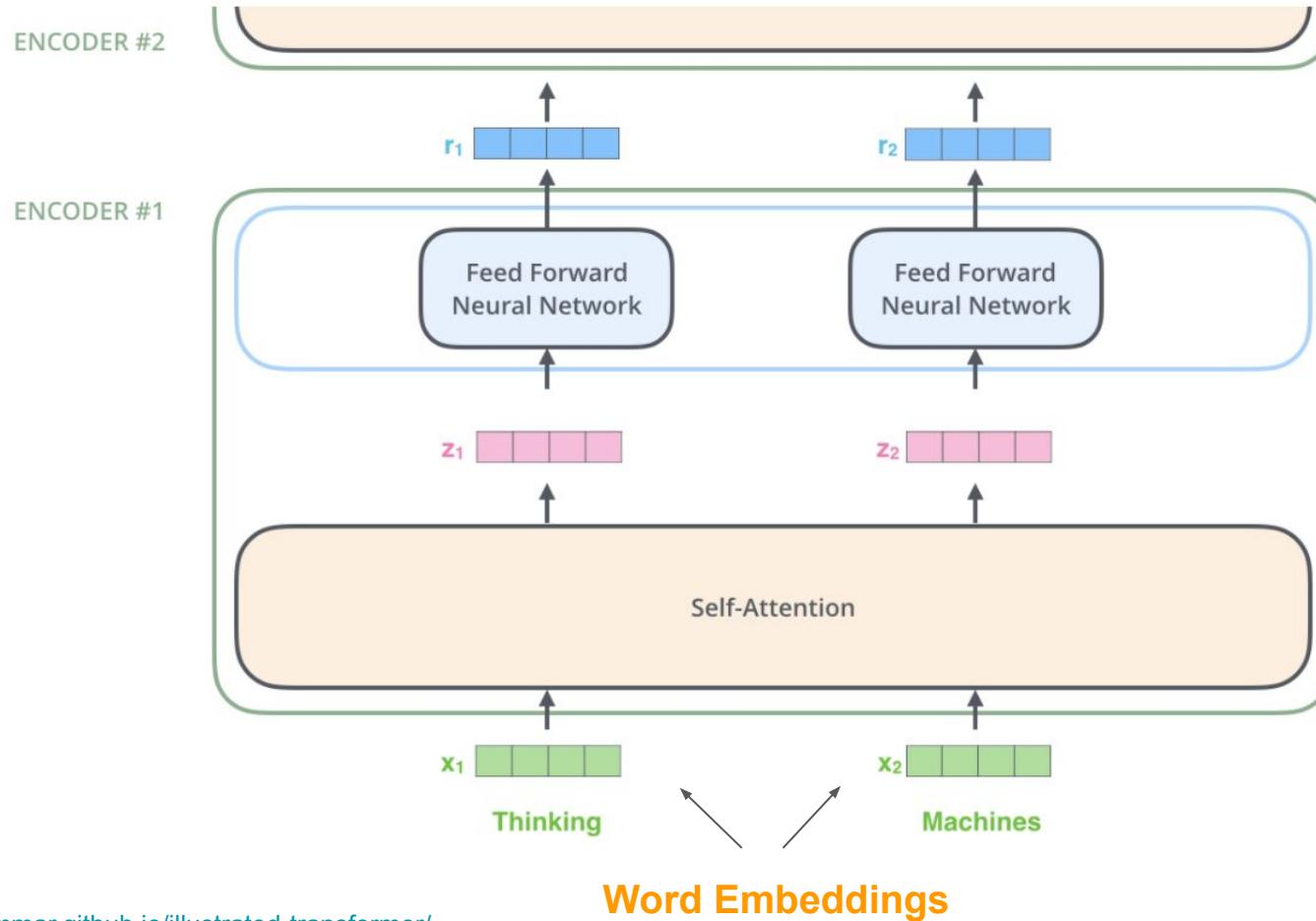
- More parallelizable -> Faster training time
- Better at longer sequence





In each layer..

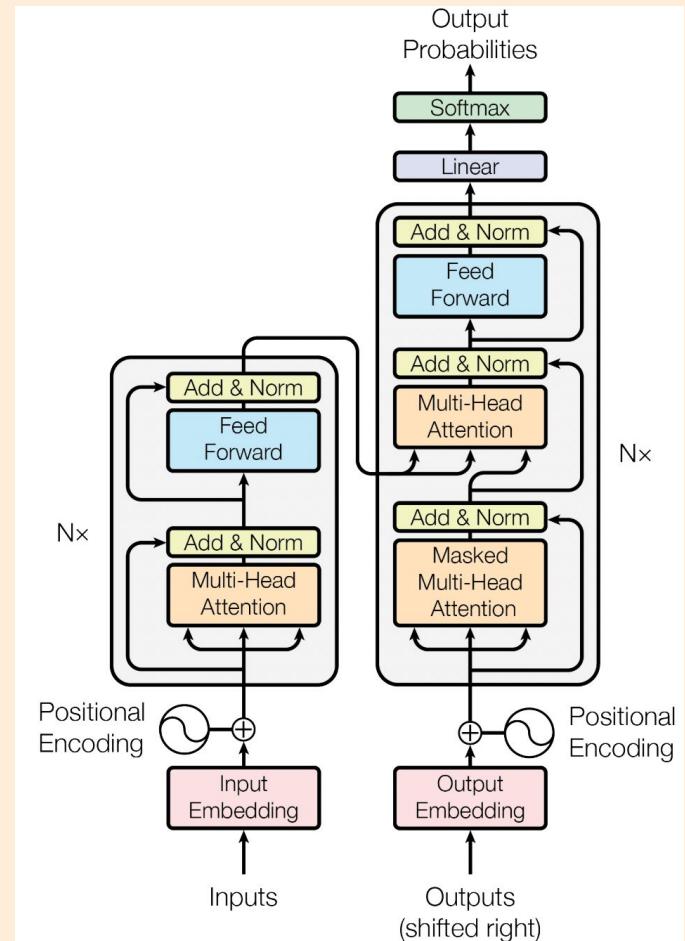




Scaled Dot-Product Attention

There are many variations used in the Transformer:

1. Self Attention
2. Encoder-decoder Attention
3. Masked Self Attention
4. Multi-headed Self Attention

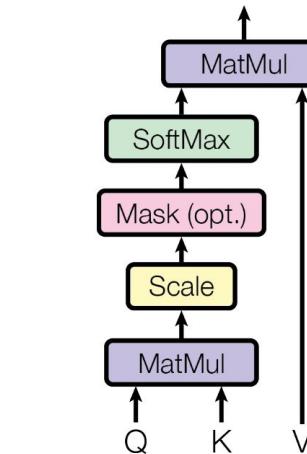


1) Self Attention

Q , K , and V are all from the same input.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where the query, keys, values, and output are all vectors and d_k is a number.



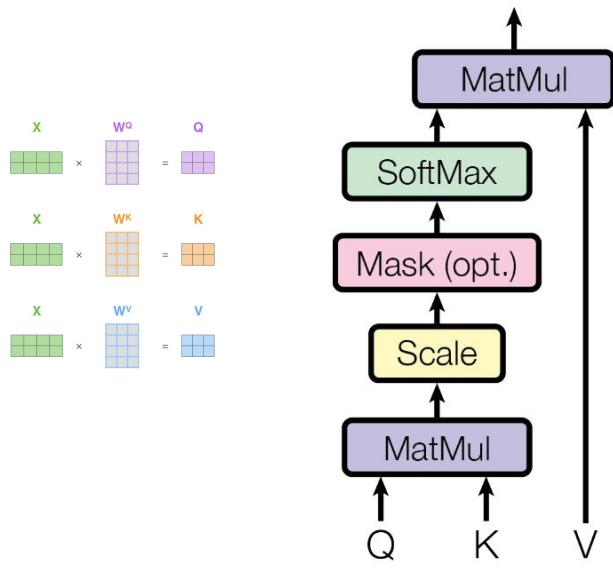
$$\begin{array}{ccc} x & \times & w^q \\ \begin{matrix} \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \\ \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \\ \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \end{matrix} & \times & \begin{matrix} \textcolor{purple}{\square} & \textcolor{purple}{\square} & \textcolor{purple}{\square} \\ \textcolor{purple}{\square} & \textcolor{purple}{\square} & \textcolor{purple}{\square} \\ \textcolor{purple}{\square} & \textcolor{purple}{\square} & \textcolor{purple}{\square} \end{matrix} \\ = & & \begin{matrix} \textcolor{purple}{\square} & \textcolor{purple}{\square} & \textcolor{purple}{\square} \\ \textcolor{purple}{\square} & \textcolor{purple}{\square} & \textcolor{purple}{\square} \\ \textcolor{purple}{\square} & \textcolor{purple}{\square} & \textcolor{purple}{\square} \end{matrix} \end{array}$$

$$\begin{array}{ccc} x & \times & w^k \\ \begin{matrix} \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \\ \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \\ \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \end{matrix} & \times & \begin{matrix} \textcolor{orange}{\square} & \textcolor{orange}{\square} & \textcolor{orange}{\square} \\ \textcolor{orange}{\square} & \textcolor{orange}{\square} & \textcolor{orange}{\square} \\ \textcolor{orange}{\square} & \textcolor{orange}{\square} & \textcolor{orange}{\square} \end{matrix} \\ = & & \begin{matrix} \textcolor{orange}{\square} & \textcolor{orange}{\square} & \textcolor{orange}{\square} \\ \textcolor{orange}{\square} & \textcolor{orange}{\square} & \textcolor{orange}{\square} \\ \textcolor{orange}{\square} & \textcolor{orange}{\square} & \textcolor{orange}{\square} \end{matrix} \end{array}$$

$$\begin{array}{ccc} x & \times & w^v \\ \begin{matrix} \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \\ \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \\ \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \end{matrix} & \times & \begin{matrix} \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \\ \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \\ \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \end{matrix} \\ = & & \begin{matrix} \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \\ \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \\ \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \end{matrix} \end{array}$$

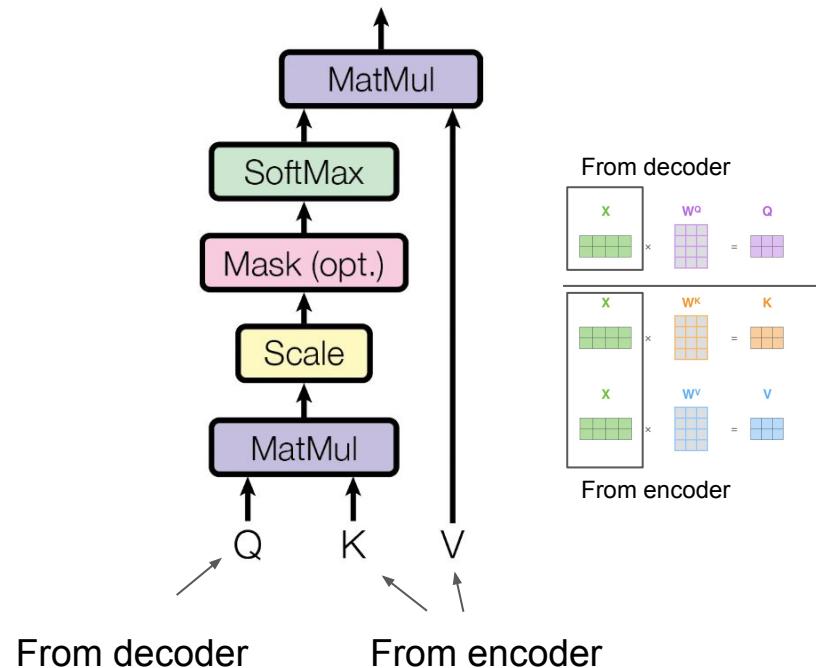
2) Encoder-Decoder Attention

Self Attention



ALL from same input

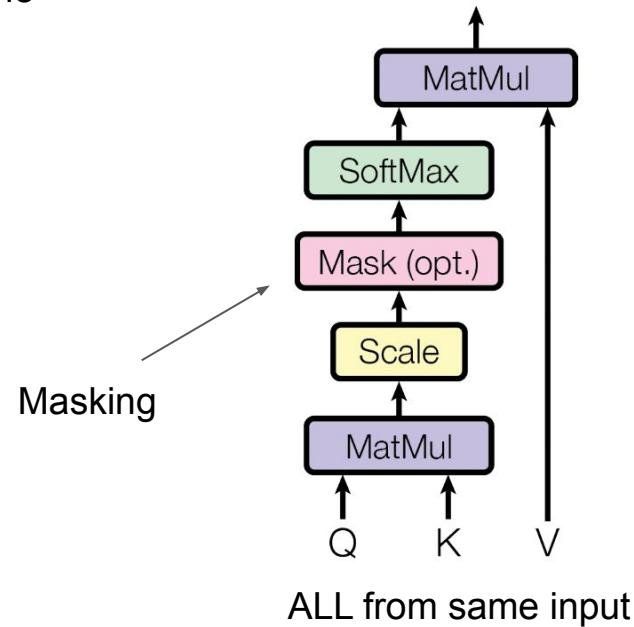
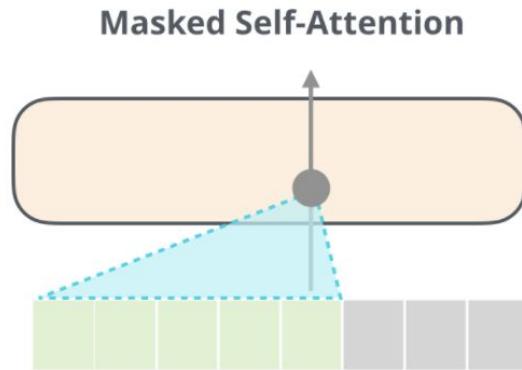
Encoder-Decoder Attention
(or Cross Attention)



3) Masked Self Attention

Prevent the model from **seeing into the future** to preserve the autoregressive property.

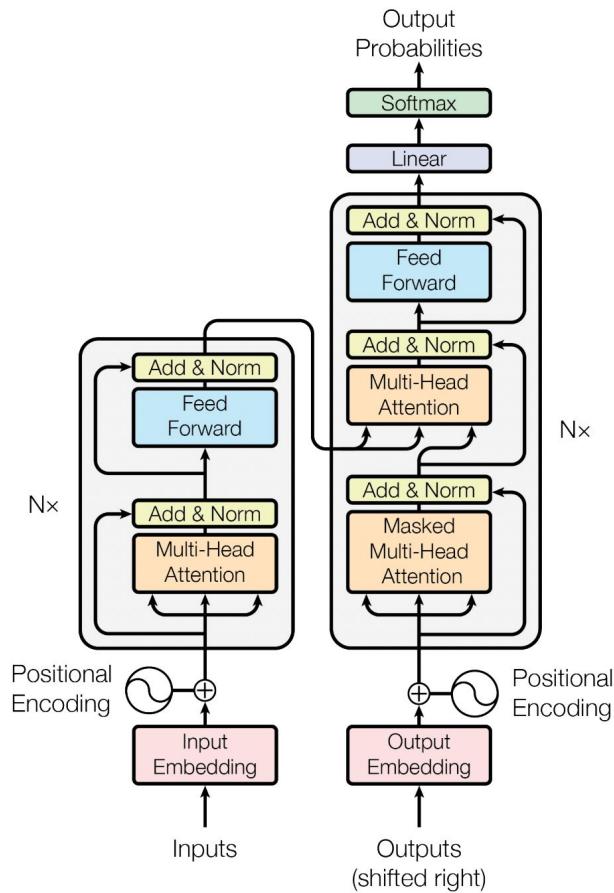
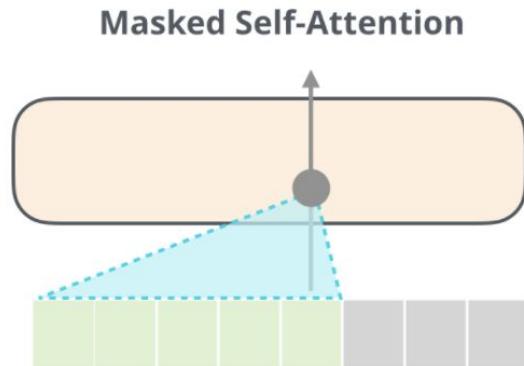
Used in *decoder* only (so the model can't see the answer).



Masked Self Attention

Prevent the model from **seeing into the future** to preserve the autoregressive property.

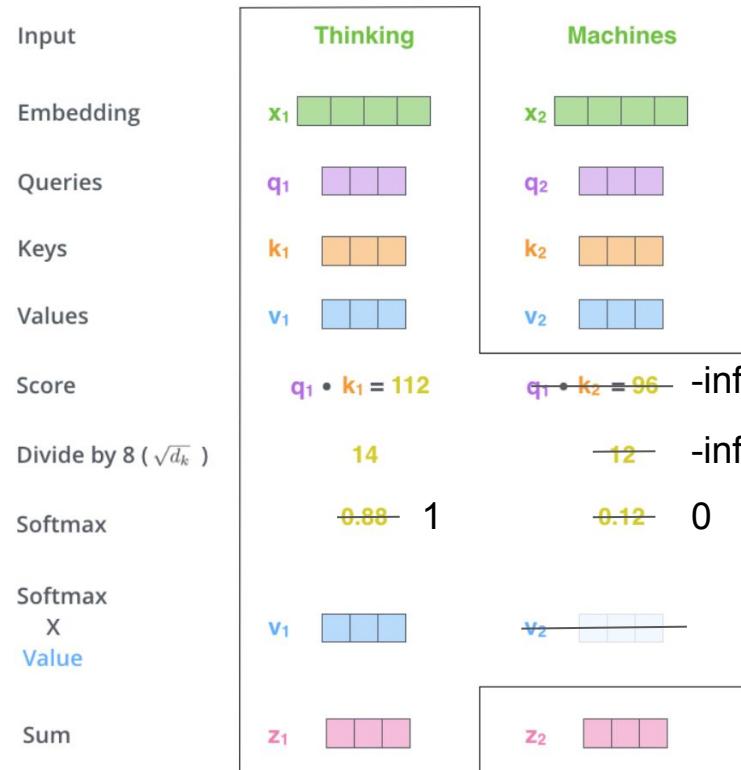
Used in *decoder* only (so the model can't see the answer).



I like dogs <eos>

<bos> ຂັນ ຈວບ ສັນຍາ

Masked Self Attention



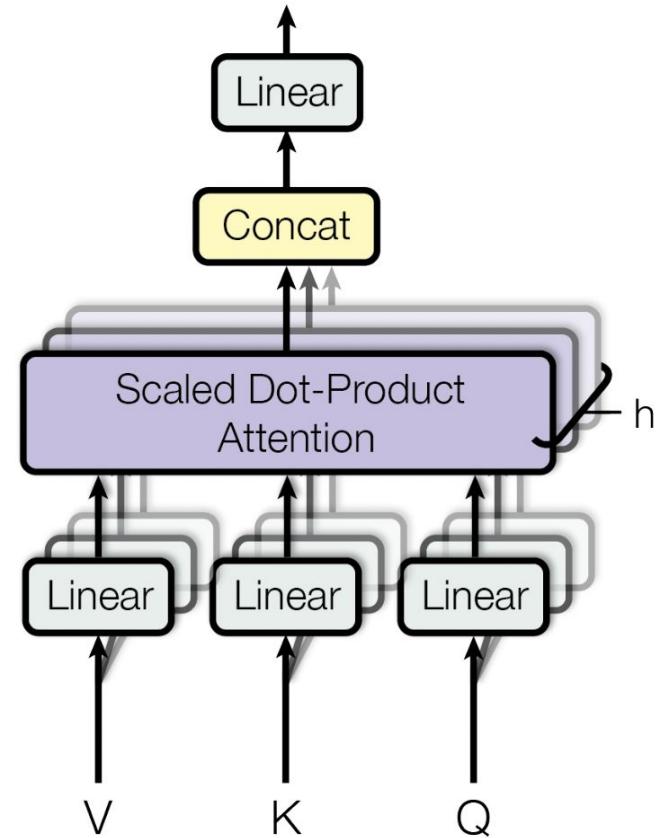
4) Multi-head Self Attention

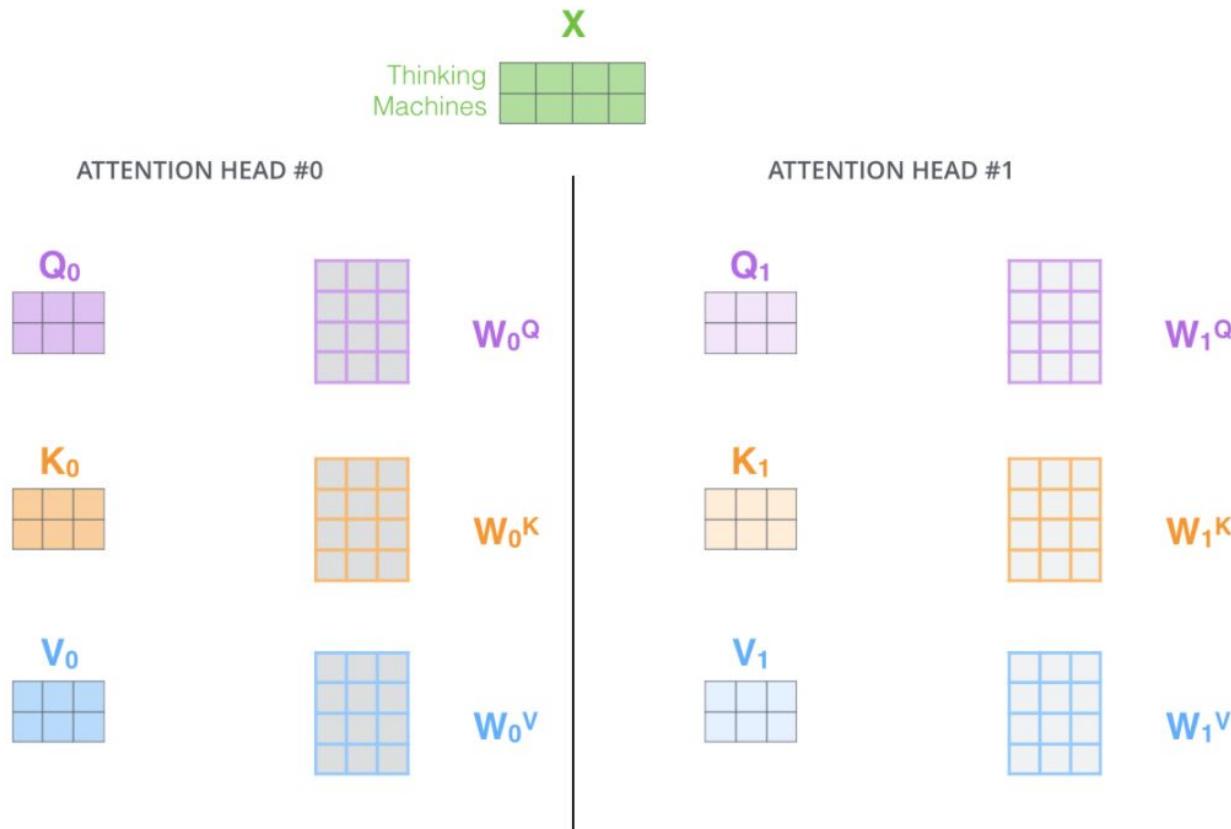
Multi-head attention allows the model to jointly attend to information from different representations.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

Multi-Head Attention

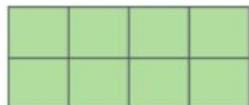




With multi-headed attention, we maintain separate Q/K/V weight matrices for each head resulting in different Q/K/V matrices. As we did before, we multiply X by the WQ/WK/WV matrices to produce Q/K/V matrices.

X

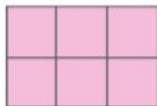
Thinking
Machines



Calculating attention separately in
eight different attention heads

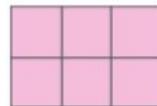
ATTENTION
HEAD #0

Z_0



ATTENTION
HEAD #1

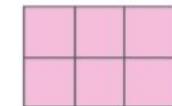
Z_1



...

ATTENTION
HEAD #7

Z_7



1) Concatenate all the attention heads



2) Multiply with a weight matrix W^o that was trained jointly with the model

\times

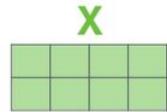


3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

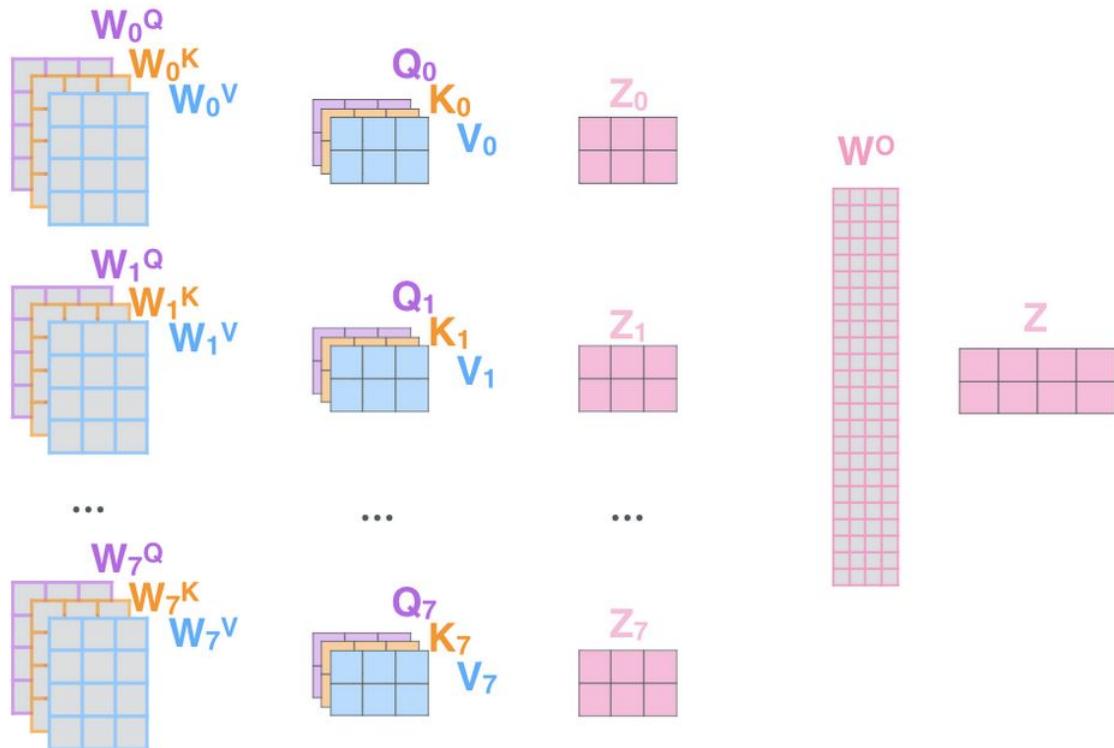
$$= \begin{matrix} Z \\ \hline \text{---} \\ \boxed{\quad \quad \quad} \end{matrix}$$

- 1) This is our input sentence* X
- 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

Thinking
Machines

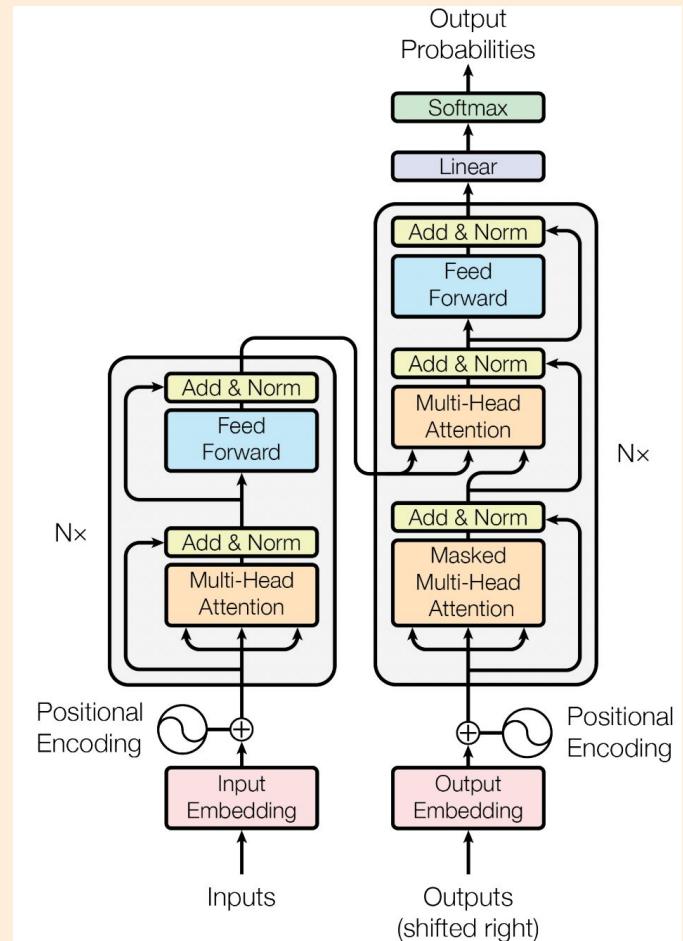


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



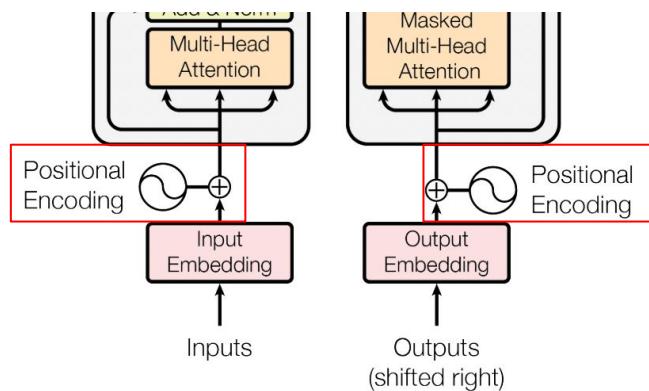
Positional Encodings

The black cat fights the white cat.



Positional Encodings

Without RNN, the model **cannot** make use of the *order* of the input sequence, e.g. the first, second, or third token.



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

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

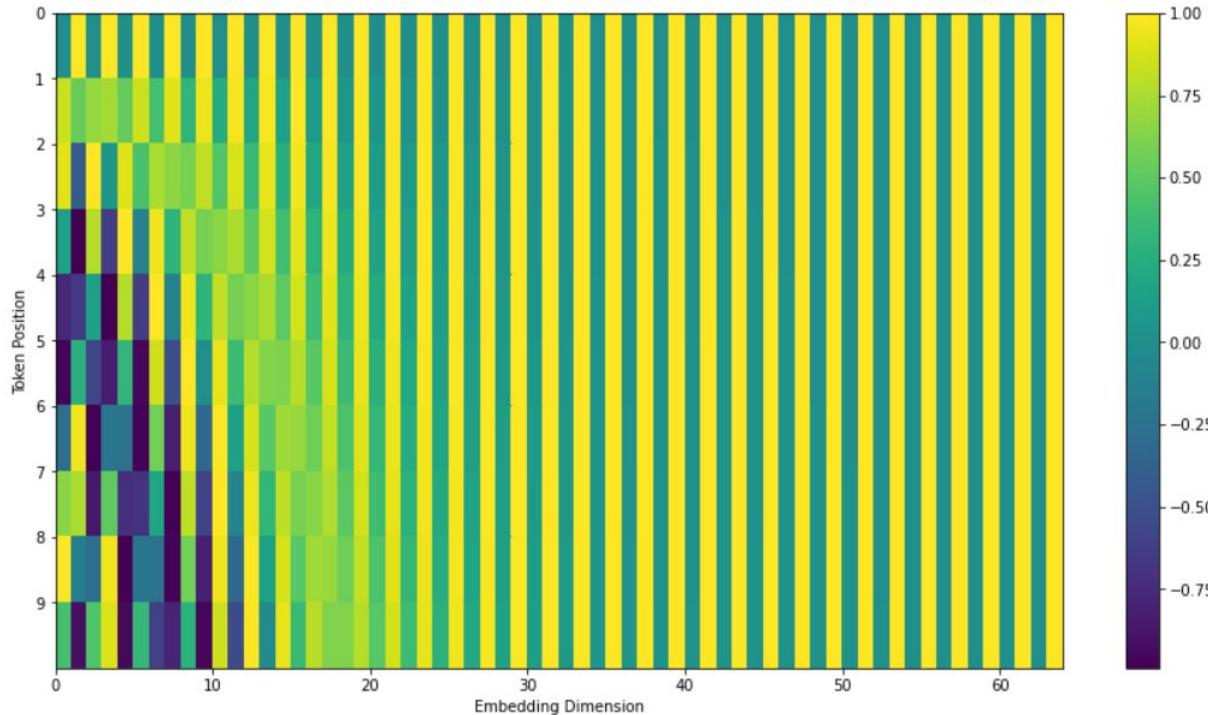
pos is the token position, *i* is the dimension

Figure 1: The Transformer - model architecture.

The black cat fights the white cat.

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

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



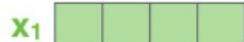
$i = 4$

POSITIONAL
ENCODING

0	0	1	1
---	---	---	---

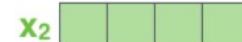
+

EMBEDDINGS



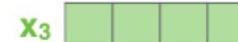
0.84	0.0001	0.54	1
------	--------	------	---

+



0.91	0.0002	-0.42	1
------	--------	-------	---

+

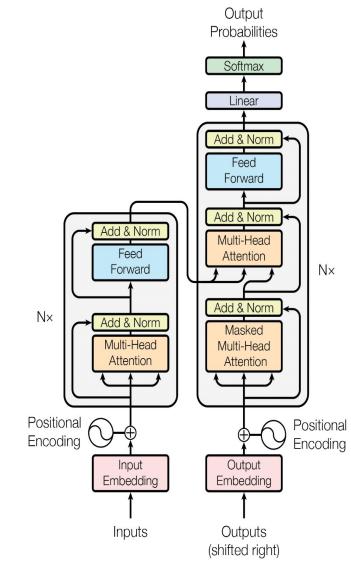


INPUT

Je

suis

étudiant

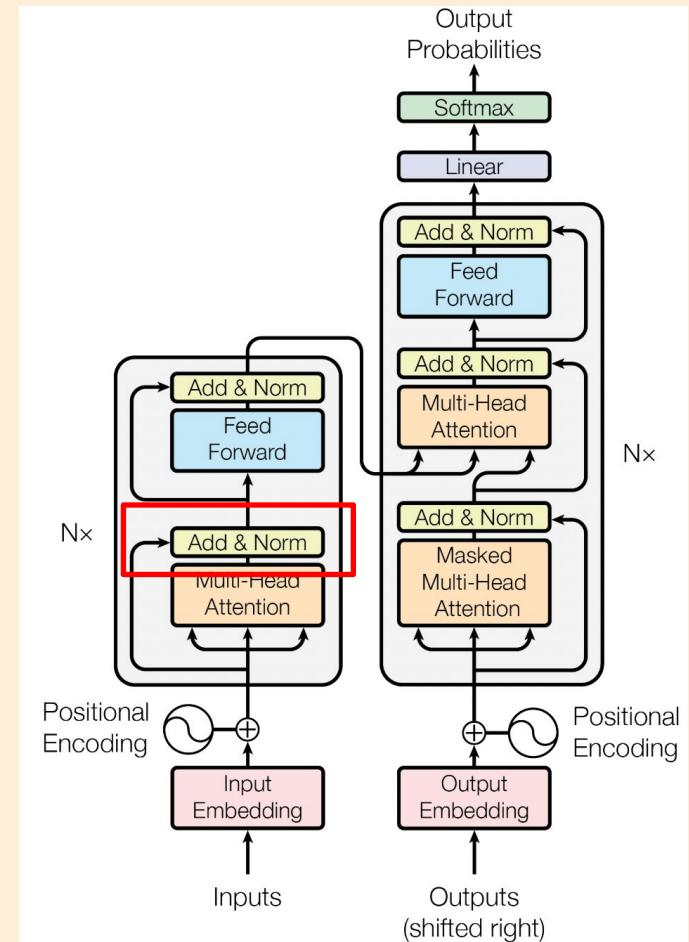


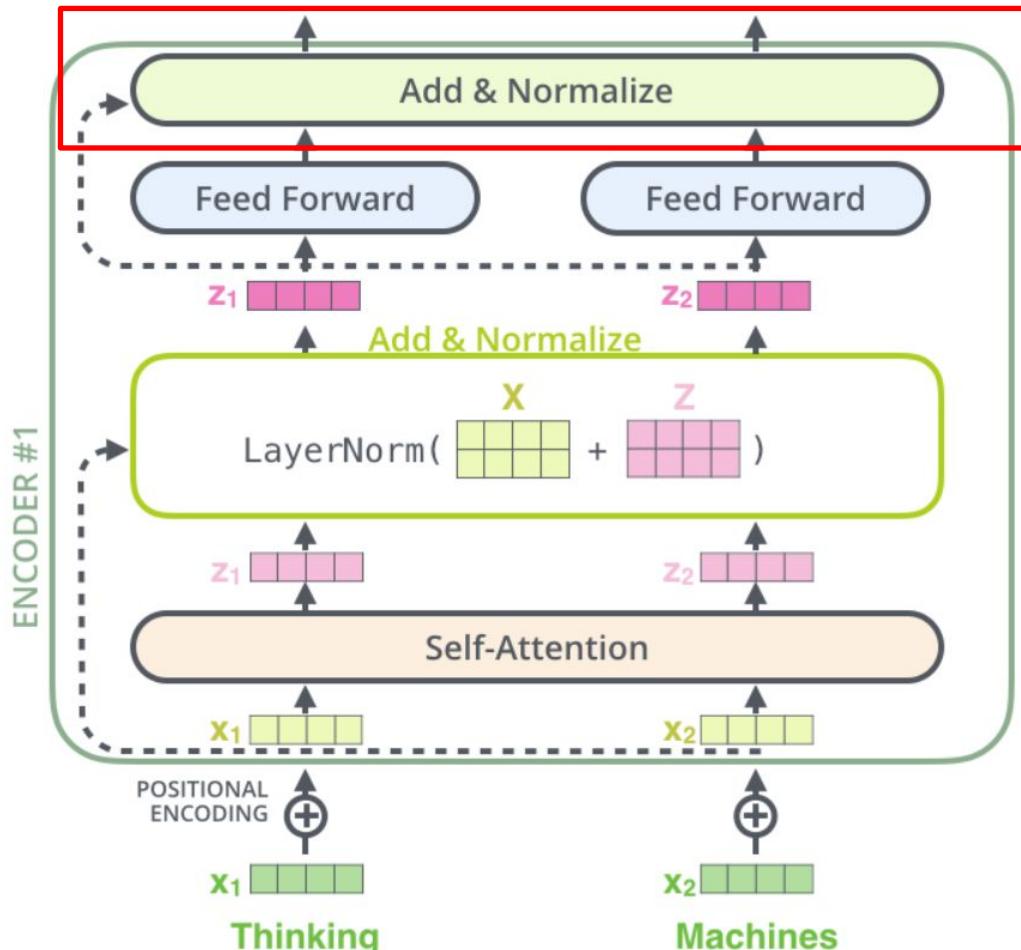
Types of positional encoding

1. Absolute Positional encoding
 - 1.1) Fixed encoding (Transformer)
 - E.g. sinusoidal forms
 - 1.2) Learned encoding (GPT)
2. Relative Positional encoding (GPT NeoX)

	Token 1	Token 2	Token 3	Token 4	Token 5	Token 6
Absolute	1	2	3	4	5	6
Relative	-2	-1	0	1	2	3

Layer Normalization





Batch norm. (each feature (channel)) vs Layer norm. (each word)

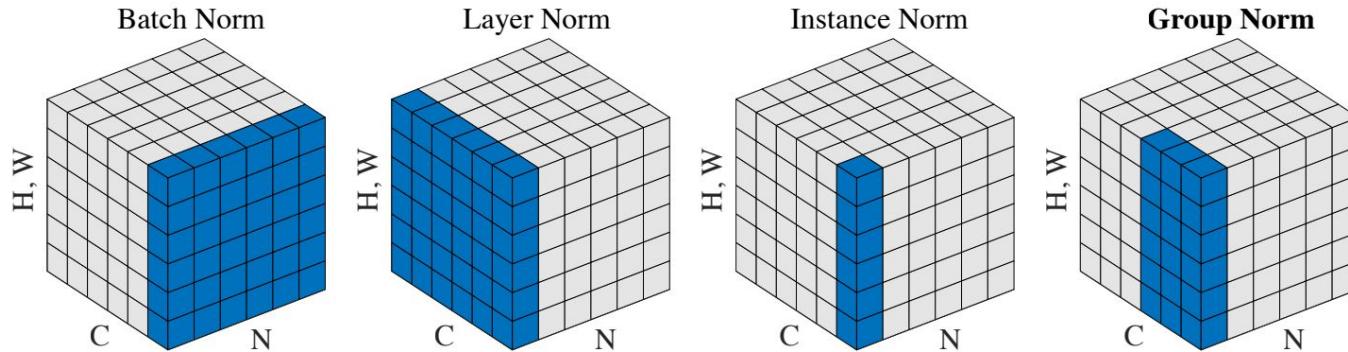
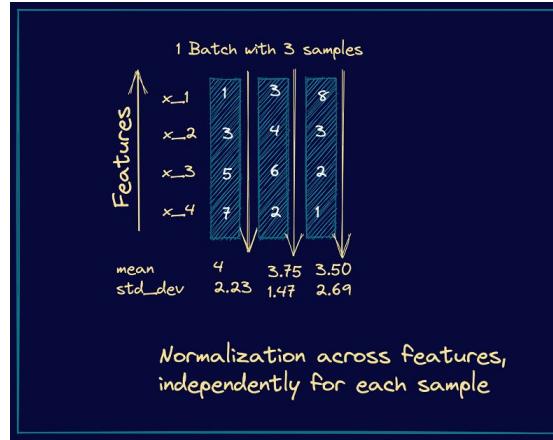
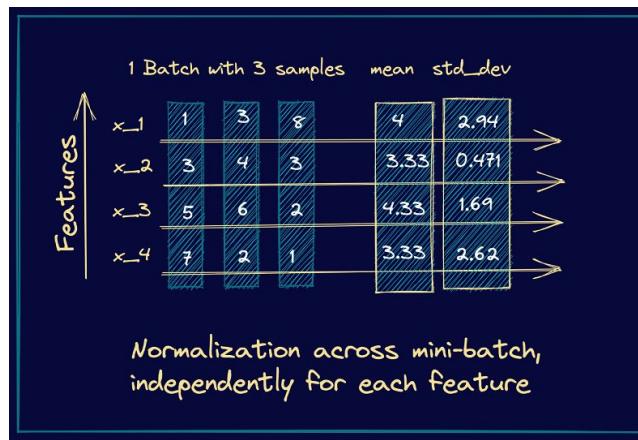
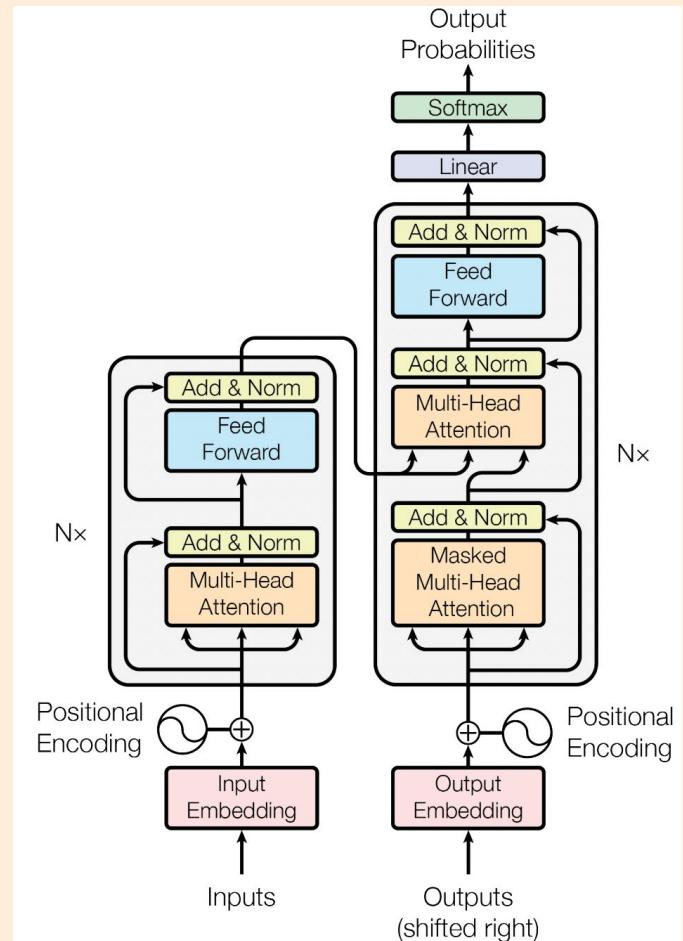


Figure 2. **Normalization methods.** Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.



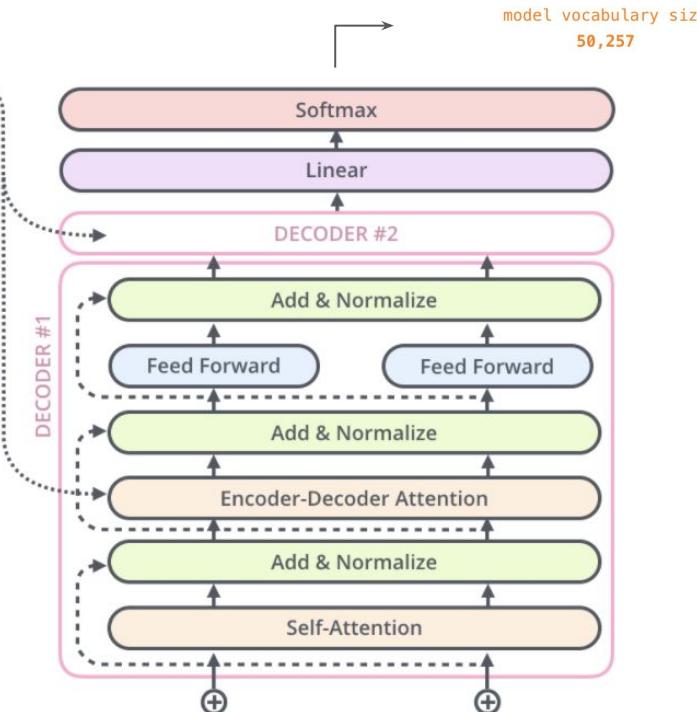
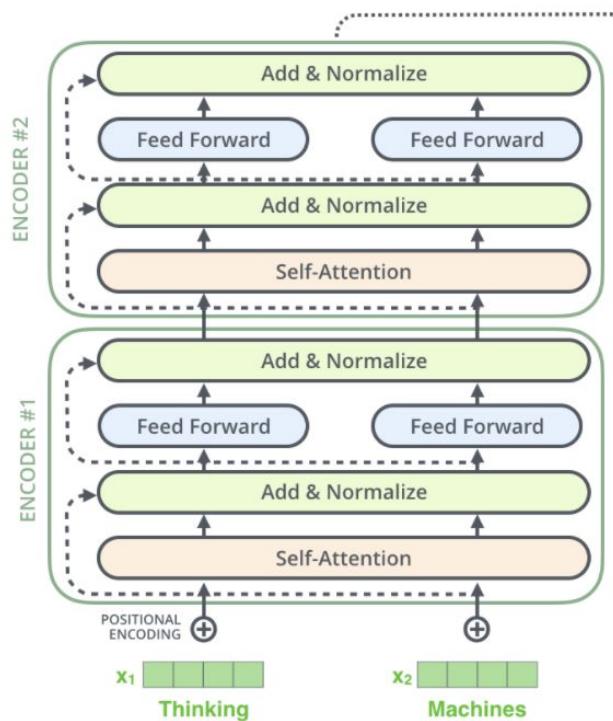
Done:

- Let's link encoder to decoder
- Then, generate **final output**

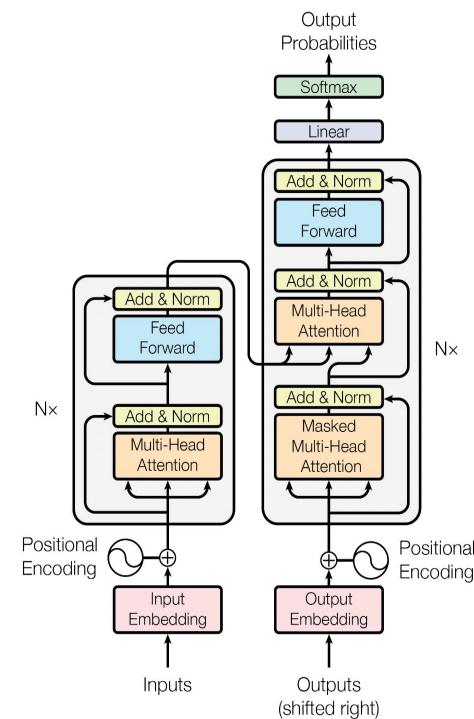


output token probabilities (logits)

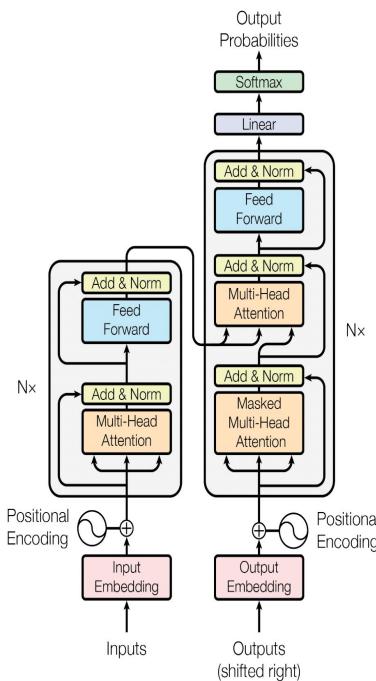
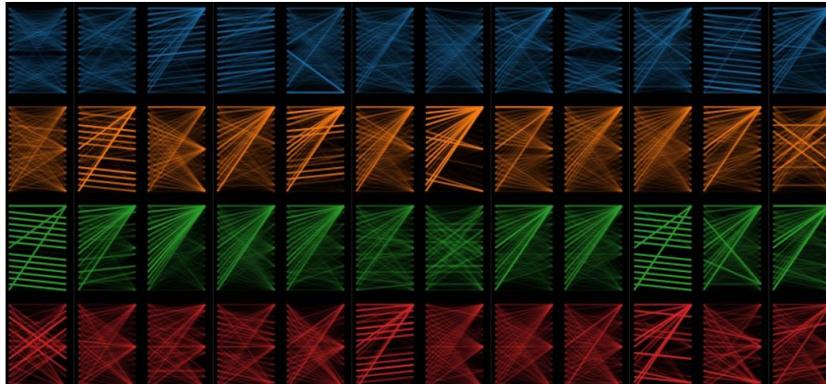
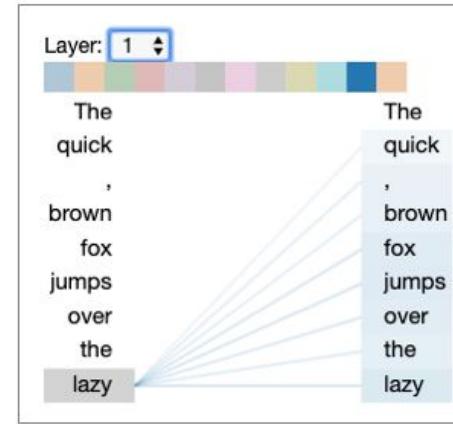
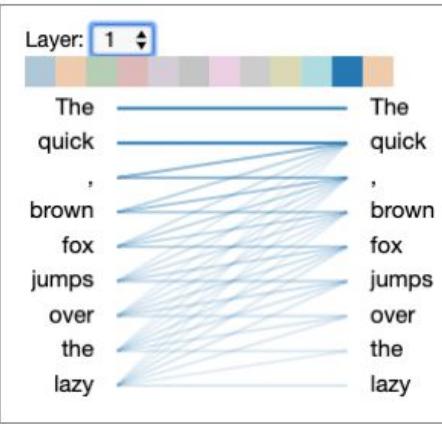
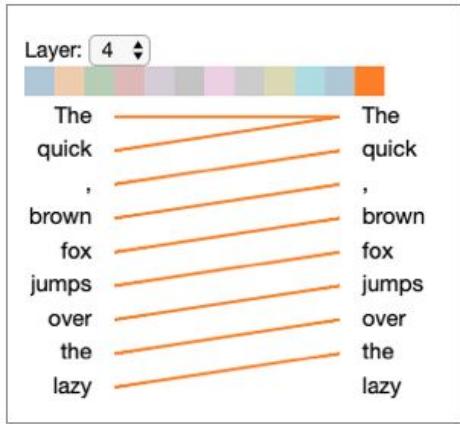
Final output



0.19850038	aardvark
0.7089803	aarhus
0.46333563	aaron
...	...
...	...
...	...
-0.51006055	zyzzyva



Visualizing Attention (N encoders & N decoders)



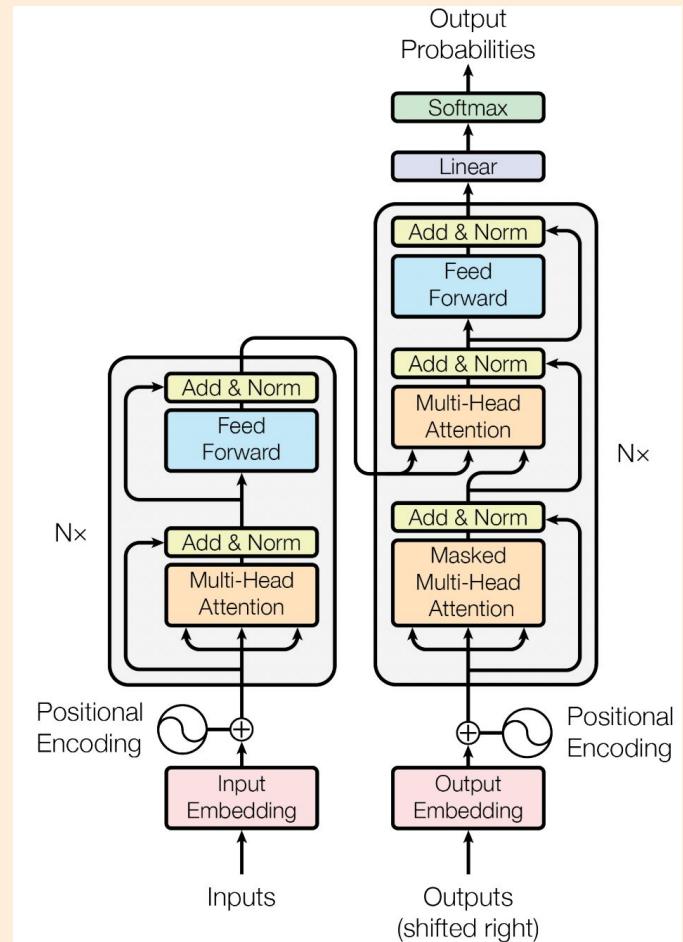
Large Language Models

Pretrained Language Models

All Transformer based

1. Decoder-based model: GPT
2. Encoder-based model: BERT
3. Encoder and Decoder: BART

Generative Pre-Training (OpenAI GPT)





OpenAI GPT (Generative Pre-Training) [Radford, 2018]

Improving Language Understanding by Generative Pre-Training

Alec Radford

OpenAI

alec@openai.com

Karthik Narasimhan

OpenAI

karthikn@openai.com

Tim Salimans

OpenAI

tim@openai.com

Ilya Sutskever

OpenAI

ilyasu@openai.com

Abstract

Natural language understanding comprises a wide range of diverse tasks such as textual entailment, question answering, semantic similarity assessment, and document classification. Although large unlabeled text corpora are abundant, labeled data for learning these specific tasks is scarce, making it challenging for discriminatively trained models to perform adequately. We demonstrate that large gains on these tasks can be realized by *generative pre-training* of a language model on a diverse corpus of unlabeled text, followed by *discriminative fine-tuning* on each specific task. In contrast to previous approaches, we make use of task-aware input transformations during fine-tuning to achieve effective transfer while requiring minimal changes to the model architecture. We demonstrate the effectiveness of our approach on a wide range of benchmarks for natural language understanding. Our general task-agnostic model outperforms discriminatively trained models that use architectures specifically crafted for each task, significantly improving upon the state of the art in 9 out of the 12 tasks studied. For instance, we achieve absolute improvements of 8.9% on commonsense reasoning (Stories Cloze Test), 5.7% on question answering (RACE), and 1.5% on textual entailment (MultiNLI).

Generative Pre-Training (OpenAI GPT)

A model comprises of Transformer decoders only.

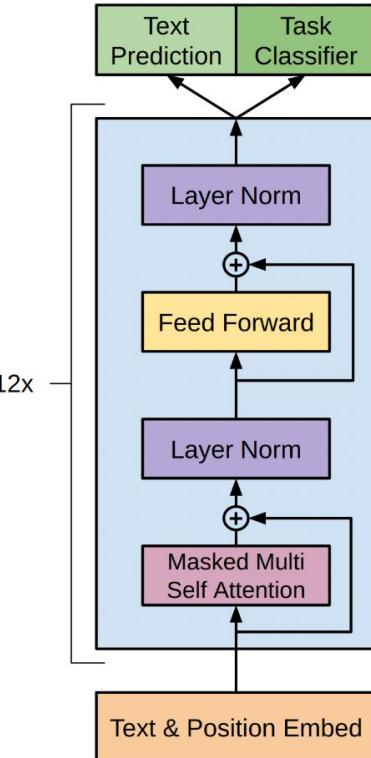
The framework consists of two stages:

1. Unsupervised pre-training
2. Supervised finetuning

Excels at text generation tasks

Try it out:

<https://transformer.huggingface.co/>



Generative Pre-Training (OpenAI GPT)

1) Unsupervised pre-training

Given a large unlabelled corpus $\mathcal{U} = \{u_1, \dots, u_n\}$, the model's objective is to maximize the following likelihood (also called “Language Modelling”).

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \dots, u_{i-1}; \Theta)$$

where k is the context length and the conditional probability P is modeled using a neural network with parameters Θ .

Generative Pre-Training (OpenAI GPT)

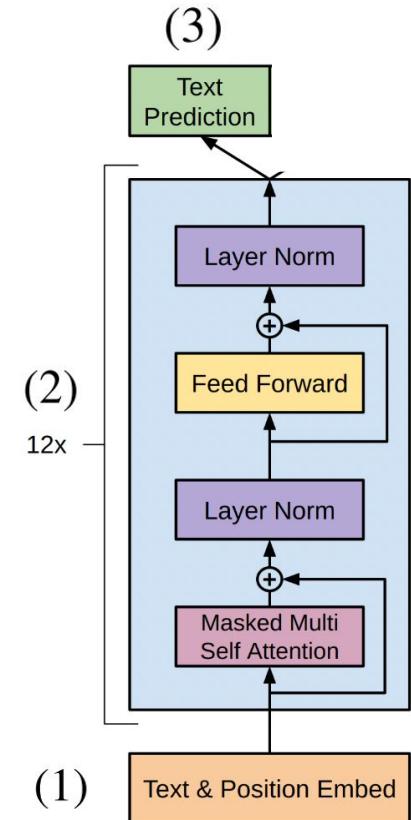
1) Unsupervised pre-training

$$(1) \quad h_0 = UW_e + W_p$$

$$(2) \quad h_l = \text{transformer_block}(h_{l-1}) \forall i \in [1, n]$$

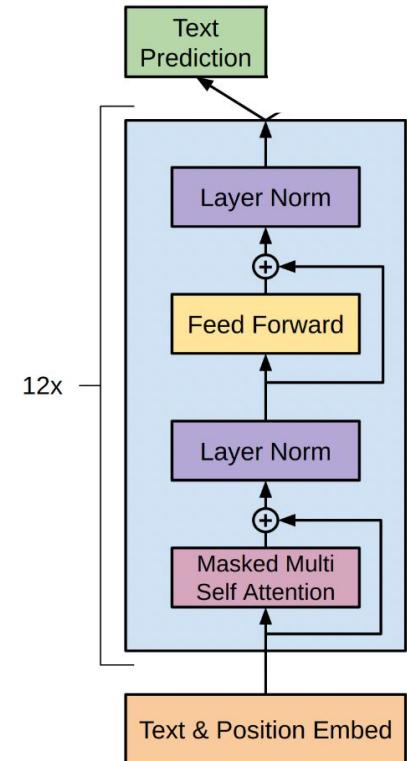
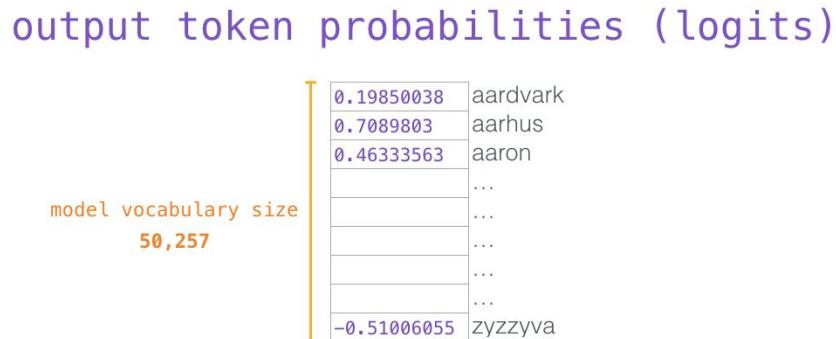
$$(3) \quad P(u) = \text{softmax}(h_n W_e^T)$$

where $U = (u_{-k}, \dots, u_{-1})$ is the context vector of tokens, n is the number of layers, W_e is the token embedding matrix, and W_p is the position embedding matrix.



Generative Pre-Training (OpenAI GPT)

1) Unsupervised pre-training



Generative Pre-Training (OpenAI GPT)

2) Supervised finetuning

We assume a labeled dataset C, where each instance consists of a sequence of input tokens (x^1, \dots, x^m) , along with a label y.

$$P(y|x^1, \dots, x^m) = \text{softmax}(h_l^m W_y).$$

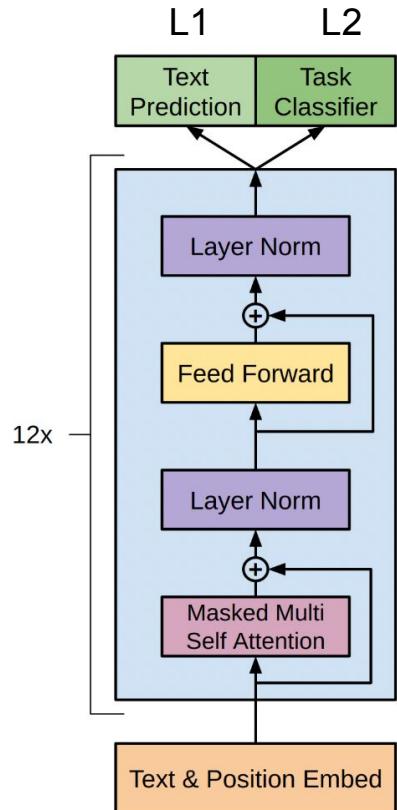
$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^1, \dots, x^m).$$

Total finetuning loss

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * L_1(\mathcal{C})$$

classification

LM
(unsupervised)



Generative Pre-Training (OpenAI GPT)

2) Supervised finetuning

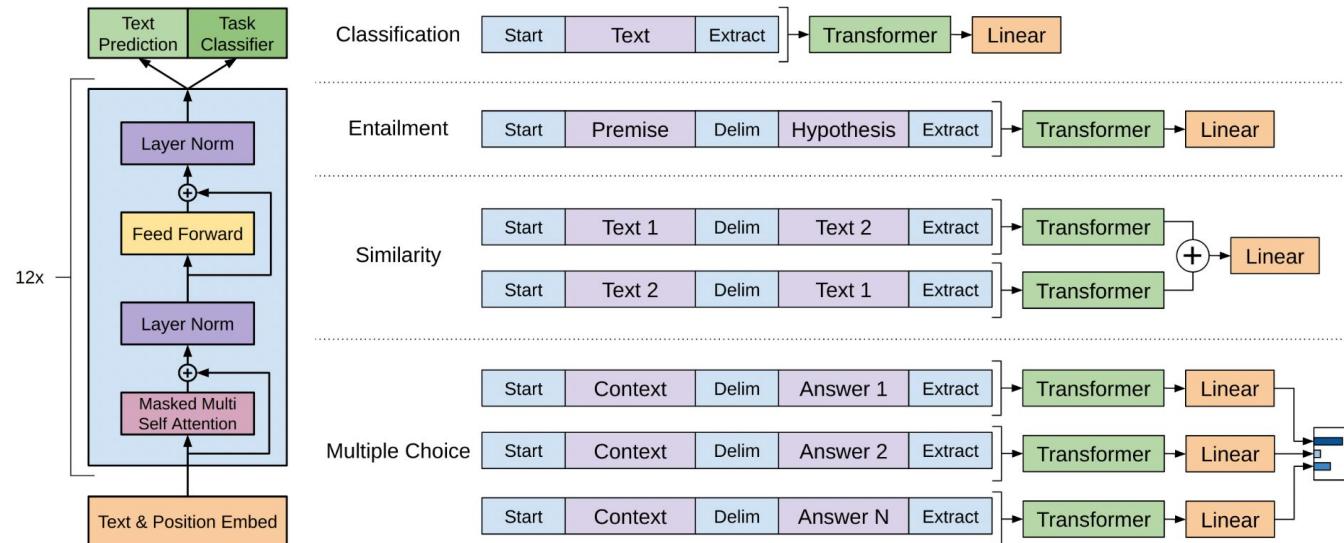


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

GPT2 & GPT3

Bigger Models + More data = Better Results

Zero-shot

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



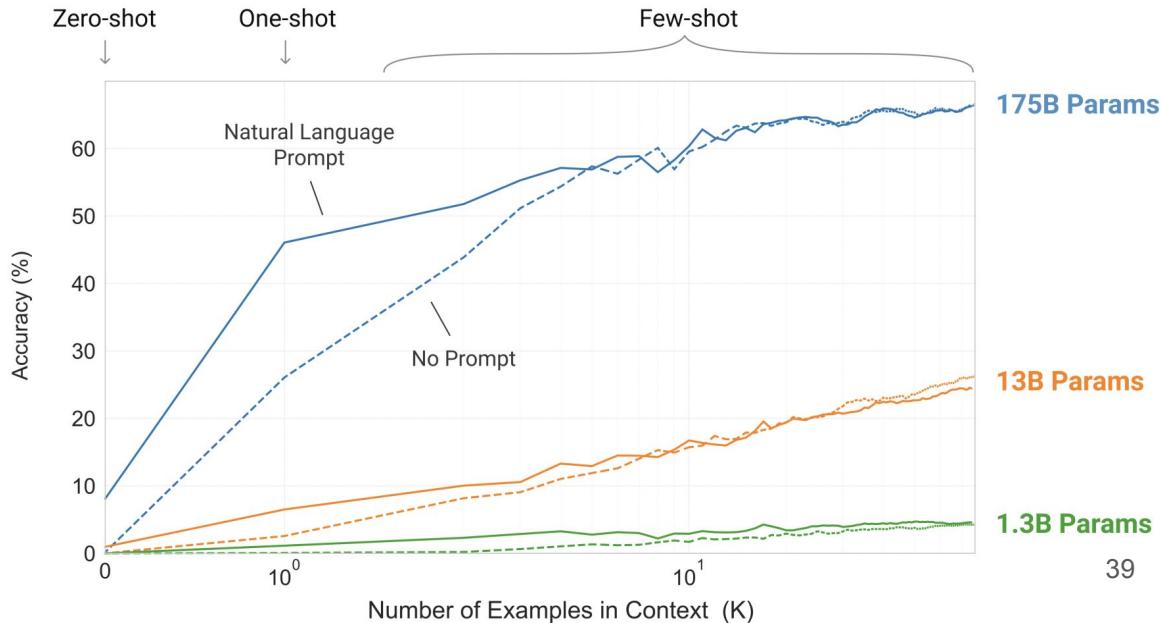
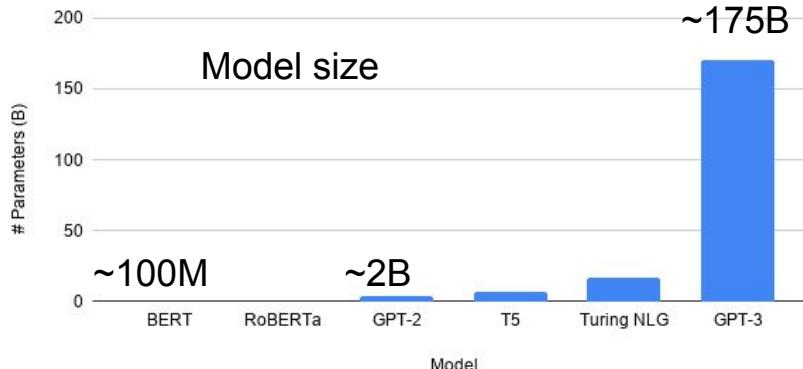
One-shot

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



Few-shot

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



OpenAI API

Models referred to as "GPT 3.5"

GPT-3.5 series is a series of models that was trained on a blend of text and code from before Q4 2021. The following models are in the GPT-3.5 series:

- 1 `code-davinci-002` is a base model, so good for pure code-completion tasks
- 2 `text-davinci-002` is an InstructGPT model based on `code-davinci-002`
- 3 `text-davinci-003` is an improvement on `text-davinci-002`

GPT-3.5

GPT-3.5 models can understand and generate natural language or code. Our most capable and cost effective model is `gpt-3.5-turbo` which is optimized for chat but works well for traditional completions tasks as well.

LATEST MODEL	DESCRIPTION	MAX REQUEST	TRAINING DATA
gpt-3.5-turbo	Most capable GPT-3.5 model and optimized for chat at 1/10th the cost of <code>text-davinci-003</code> . Will be updated with our latest model iteration.	4,096 tokens	Up to Sep 2021
gpt-3.5-turbo-0301	Snapshot of <code>gpt-3.5-turbo</code> from March 1st 2023. Unlike <code>gpt-3.5-turbo</code> , this model will not receive updates, and will only be supported for a three month period ending on June 1st 2023.	4,096 tokens	Up to Sep 2021
text-davinci-003	Can do any language task with better quality, longer output, and consistent instruction-following than the curie, babbage, or ada models. Also supports inserting completions within text.	4,000 tokens	Up to Jun 2021
text-davinci-002	Similar capabilities to <code>text-davinci-003</code> but trained with supervised fine-tuning instead of reinforcement learning	4,000 tokens	Up to Jun 2021
code-davinci-002	Optimized for code-completion tasks	4,000 tokens	Up to Jun 2021

Codex Limited beta

The Codex models are descendants of our GPT-3 models that can understand and generate code. Their training data contains both natural language and billions of lines of public code from GitHub. [Learn more](#).

They're most capable in Python and proficient in over a dozen languages including JavaScript, Go, Perl, PHP, Ruby, Swift, TypeScript, SQL, and even Shell.

We currently offer two Codex models:

LATEST MODEL	DESCRIPTION	MAX REQUEST	TRAINING DATA
code-davinci-002	Most capable Codex model. Particularly good at translating natural language to code. In addition to completing code, also supports inserting completions within code.	8,000 tokens	Up to Jun 2021
code-cushman-001	Almost as capable as Davinci Codex, but slightly faster. This speed advantage may make it preferable for real-time applications.	Up to 2,048 tokens	

For more, visit our guide to [working with Codex](#).

code



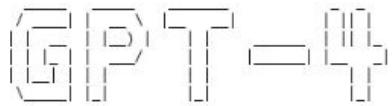
Ange Loron

Dec 14, 2022 · 3 min read · Member-only · Listen



GPT-4-100X More Powerful than GPT-3

The world's most powerful AI is about to be overruled



GPT-4 (Generative Pretrained Transformer 4) is the latest and most advanced version of the GPT (Generative Pretrained Transformer) model developed by OpenAI. It is a large-scale machine learning model that has been trained on a massive amount of data to generate human-like text.

GPT-4 builds on the success of GPT-3, which was released in May 2020 and quickly became one of the most widely used natural language processing models. GPT-4 is significantly larger and more powerful than GPT-3, with **170 trillion parameters** compared to GPT-3's **175 billion parameters**. This allows GPT-4 to process and generate text with greater accuracy and fluency.

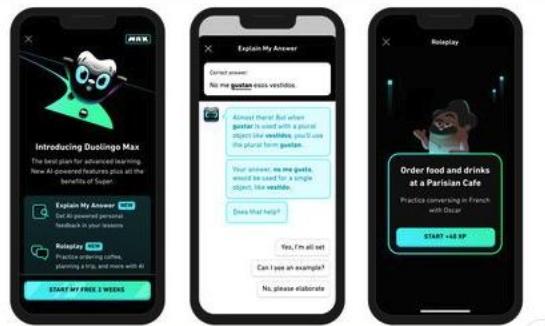
พร้อม
มาก!

OpenAI เปิดตัว GPT-4

มืออาชีวะบ้างมาดูกัน



OpenAI เปิดตัว GPT-4 ฉลาดกว่า ChatGPT เช่นไร รูปภาพได้ ทำข้อสอบระดับสูงได้หลายวิชา



BLOGNONE.COM

OpenAI เปิดตัว GPT-4 ฉลาดกว่า ChatGPT เช่นไร รูปภาพได้ ทำข้อสอบระดับสูงได้หลายวิชา | Blognone

พร้อม
มาก!

ใส่ข้อความได้
มากกว่า ChatGPT
8 เท่า!

พร้อม
มาก!

GPT-4 ไม่จำกัด
แค่ภาษาอังกฤษ



What happens when the glove drops?
It will hit the wood plank and the ball will fly up.

ลุ้นไปเข้าไปในงาน AI ก็ได้แล้วนะ

สมาชิก ChatGPT PlusTM
ลองใช้ GPT-4 ได้ก่อน



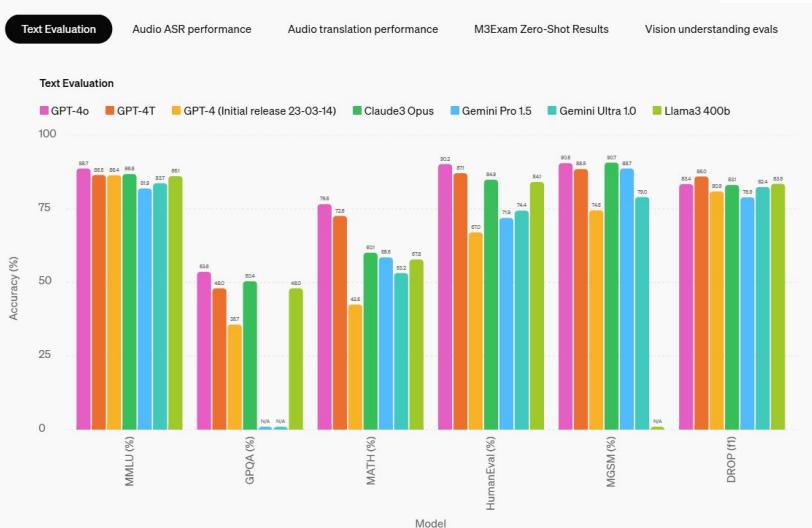
ค่าสมาชิก \$20 ต่อเดือน
จำกัดที่ 100 ข้อความทุก 4 ชั่วโมง
ยังไงรู้บ้างได้ในวันเปิดตัว

May 13, 2024

Hello GPT-4o

We're announcing GPT-4o, our new flagship model that can reason across audio, vision, and text in real time.

Contributions > Try on ChatGPT ↗



BUSINESS / TECH

OpenAI เปิดตัวโมเดล o1 ที่คิดก่อนตอบ และผลสอบเกียบเท่าผู้เรียน ปรับข้อมูลเอกสารในวิชาฟิสิกส์ เคมี และชีววิทยา

โดย สารสิน สิตาภูมิ
13.09.2024



830



Models

Copy page

Flagship models

GPT-4o

Our versatile, high-intelligence flagship model

Text and image input, text output

128k context length

Smarter model, higher price per token

GPT-4o mini

Our fast, affordable small model for focused tasks

Text and image input, text output

128k context length

Faster model, lower price per token

o1 & o1-mini

Reasoning models that excel at complex, multi-step tasks

Text and image input, text output

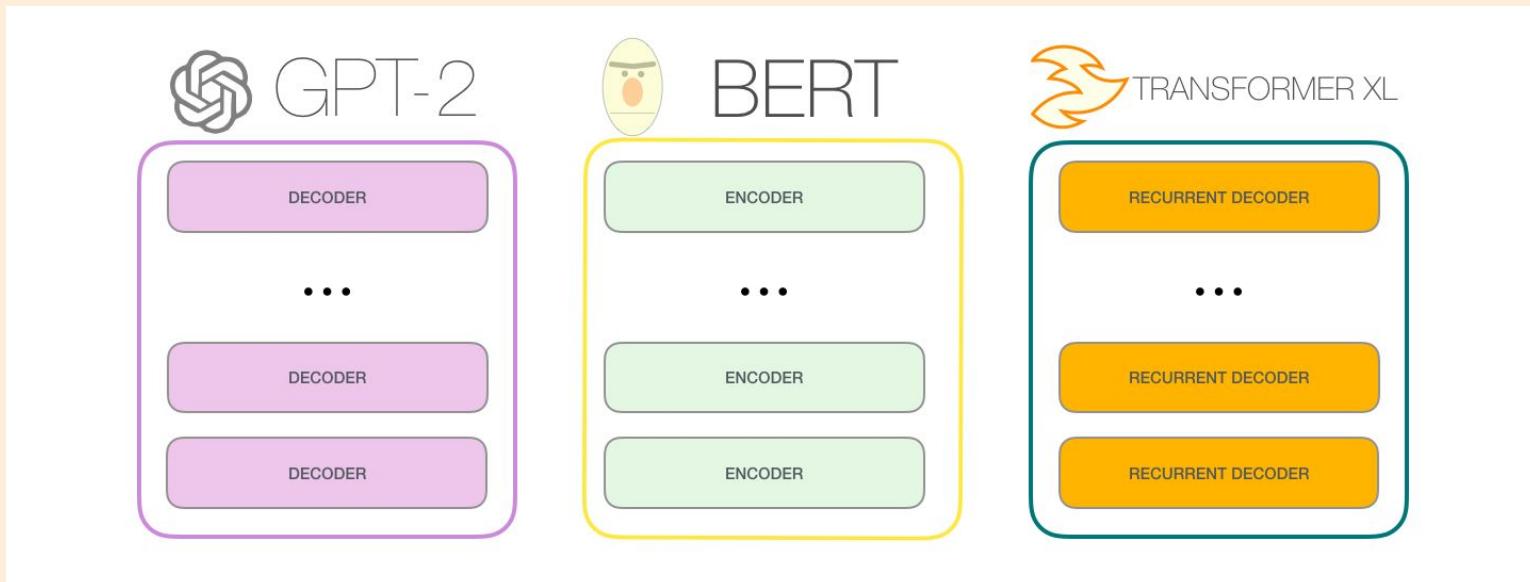
128k context length

Uses additional tokens for reasoning

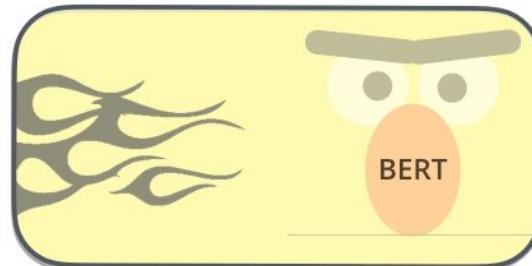
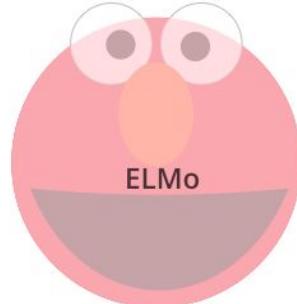
As of 2025-01-17

42

Bidirectional Encoder Representation from Transformers (BERT)



Year 2018: NLP “Pretrained” Models





BERT [Devlin, et al, 2018]: Bidirectional Encoder Representation from Transformers

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

Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova

Google AI Language

{jacobdevlin, mingweichang, kentonl, kristout}@google.com

Abstract

We introduce a new language representation model called **BERT**, which stands for Bidirectional Encoder Representations from Transformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-

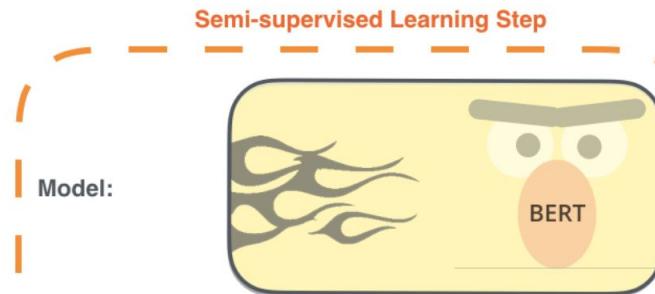
There are two existing strategies for applying pre-trained language representations to downstream tasks: *feature-based* and *fine-tuning*. The feature-based approach, such as ELMo (Peters et al., 2018a), uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning *all* pre-



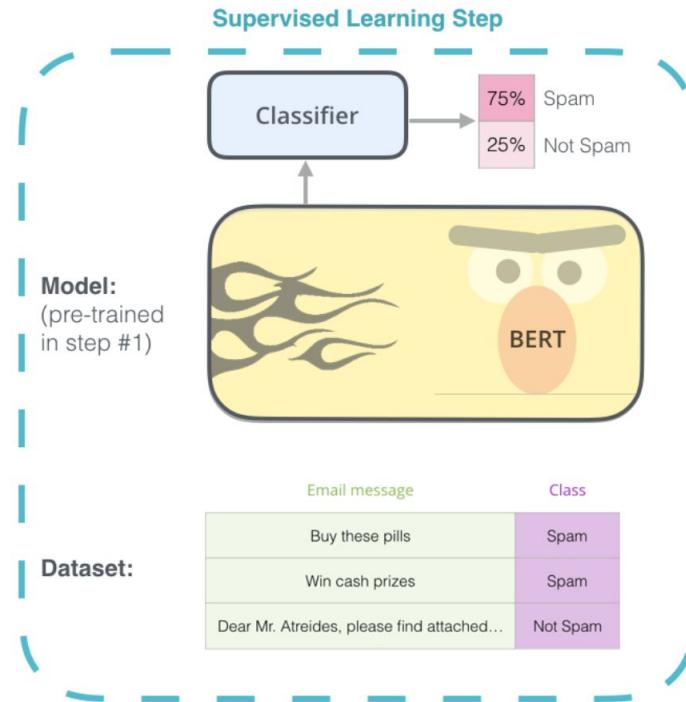
BERT [Devlin, et al, 2018]: Bidirectional Encoder Representation from Transformers

1 - Unsupervised training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.



2 - Supervised training on a specific task with a labeled dataset.

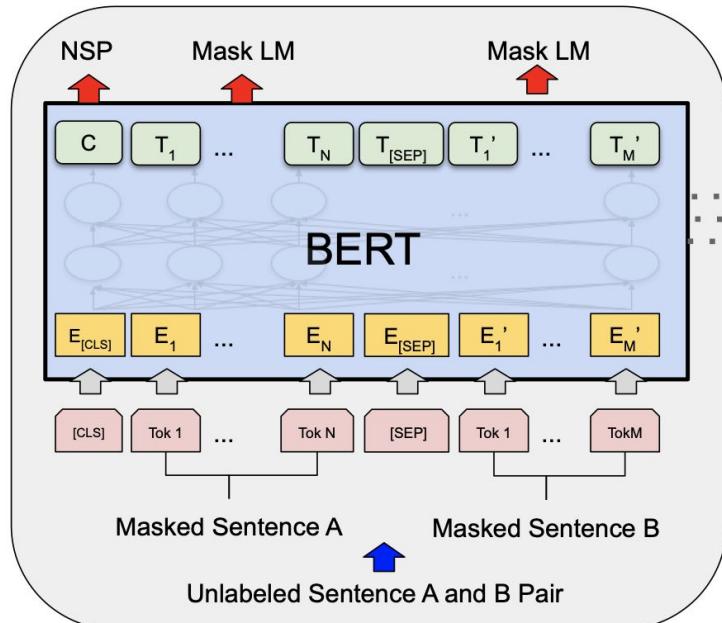




BERT (cont.): Overall idea

Phase1: Unsupervised Learning

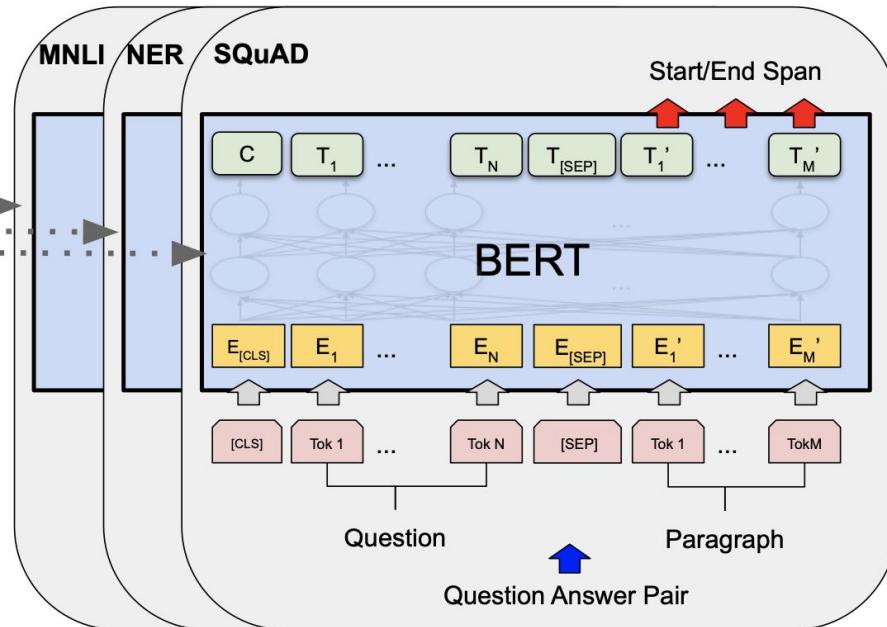
- Mask LM
- Next sentence prediction



Pre-training

Phase2: Supervised learning

- E.g. Finetune on QA, Text classification, etc.



Fine-Tuning



Phase1: Unsupervised Phase

Semi-supervised training, using 2 prediction tasks:

1.1) Mask language modeling

- represents the word using both its left and right context, so called deeply bidirectional.
- Mask out 15% of the words in the input, run the entire sequence through a deep bidirectional encoder, and then predict only the masked words.

Input: the man went to the [MASK1] . he bought a [MASK2] of milk.

Labels: [MASK1] = store; [MASK2] = gallon

1.2) Next sentence prediction (NSP)

- to learn relationships between sentences.

Sentence A: the man went to the store .

Sentence B: he bought a gallon of milk .

Label: IsNextSentence

Sentence A: the man went to the store .

Sentence B: penguins are flightless .

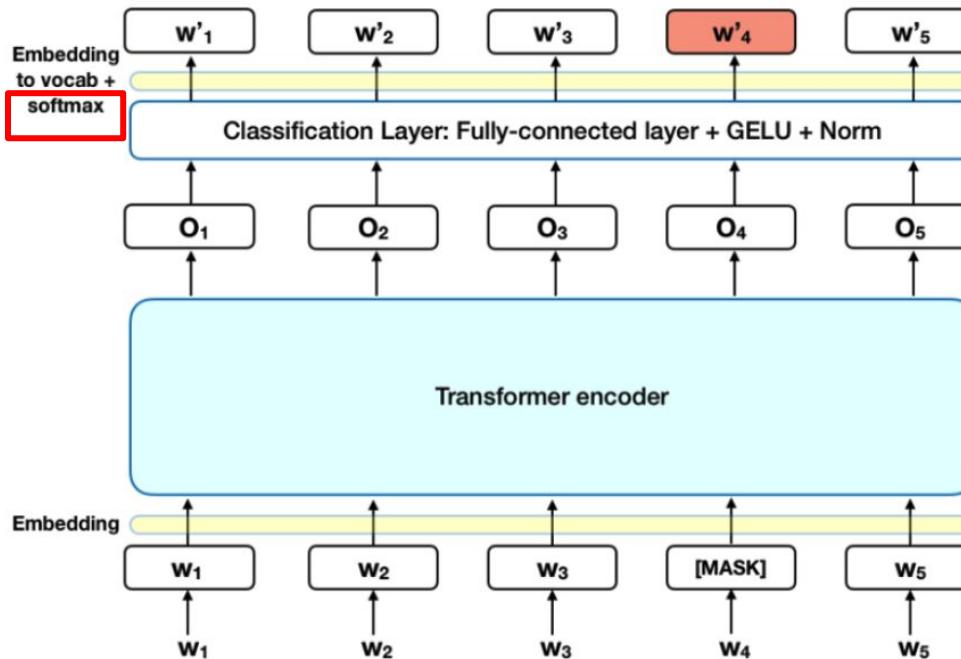
Label: NotNextSentence



Phase1: Unsupervised Phase

(1.1. Masked LM)

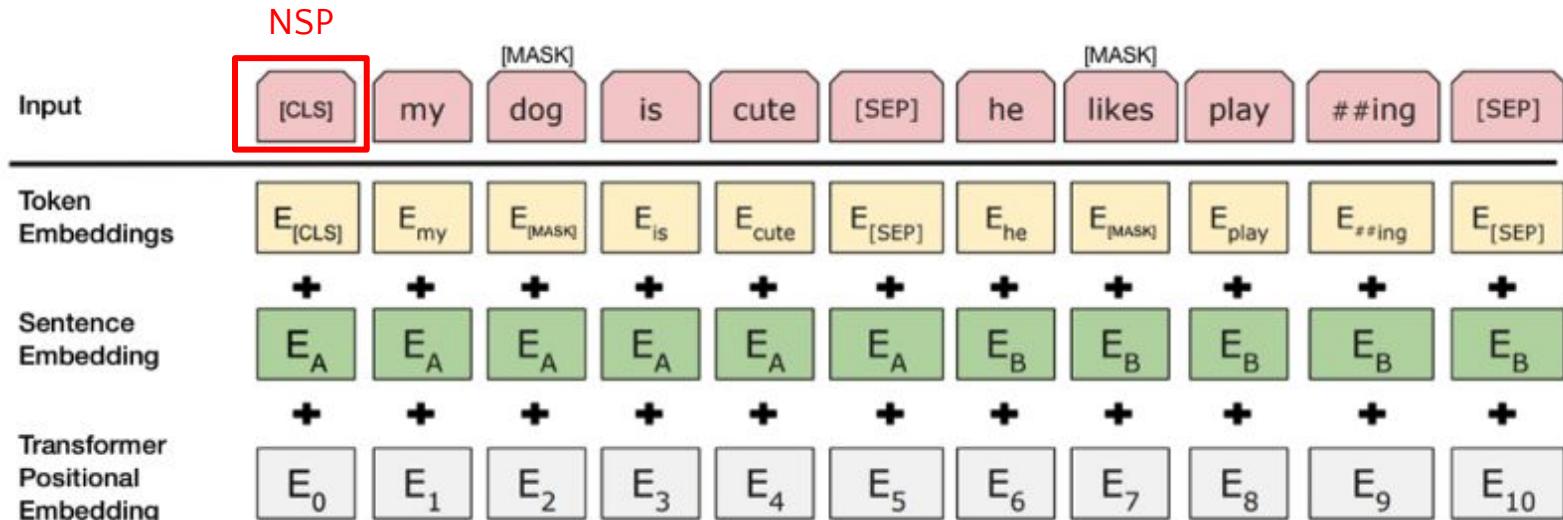
Mask language modeling:





Phase1: Unsupervised Phase

(1.2. Next Sentence Prediction)



CLS = Classification, which is used for NSP (Next Sentence Prediction) during this phase

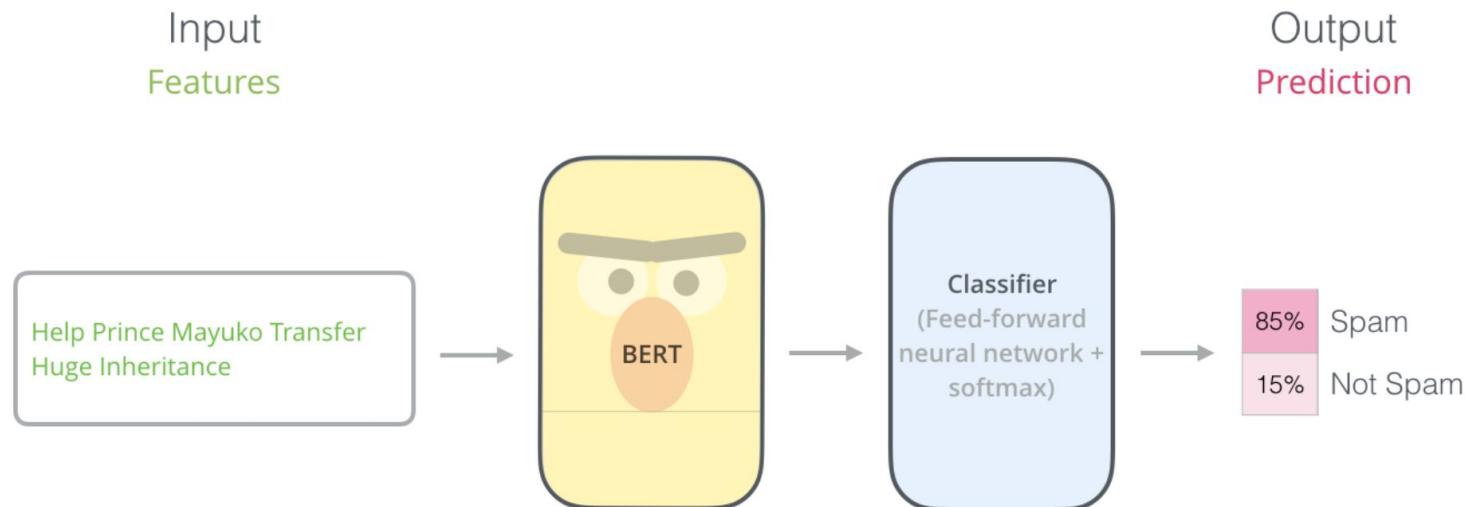
SEP = a special separator token (e.g. separating questions/answers).



Phase2: Supervised Phase

Supervised training (e.g. Email sentence classification)

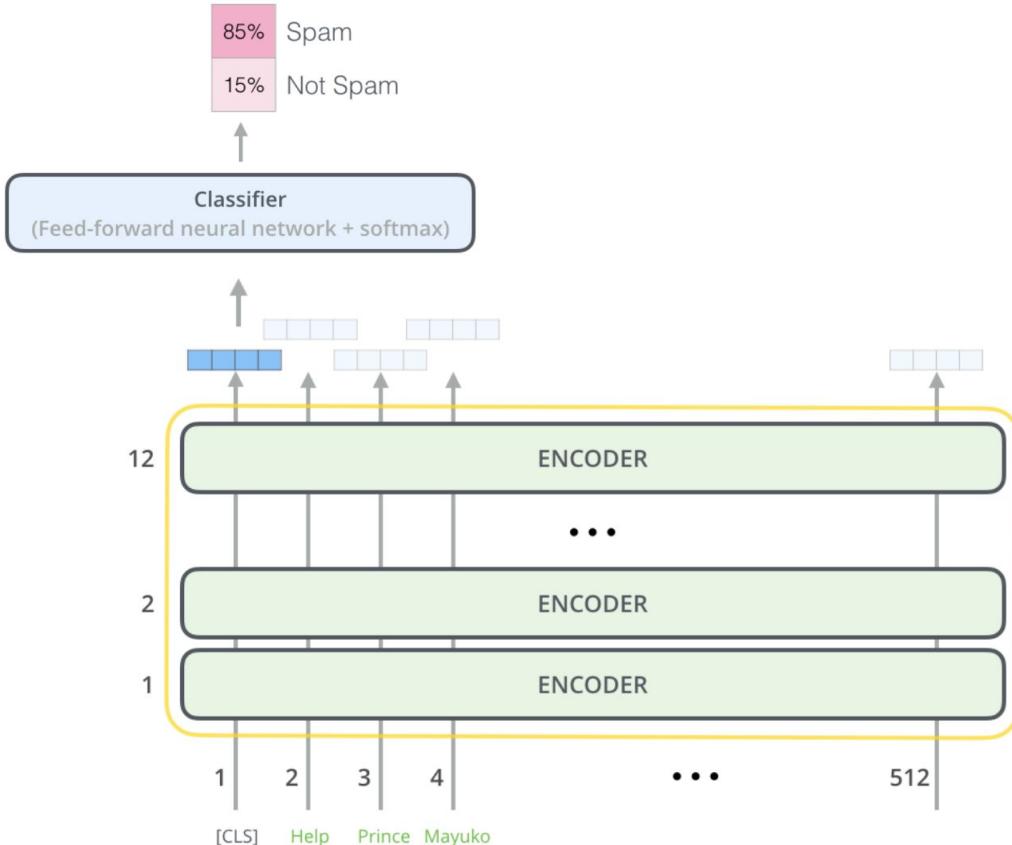
- you mainly have to train the classifier, with minimal changes happening to the pre-trained model during the training phase (**fine-tuning approach**).





Phase2: Supervised Phase E.g., Spam Email Classification

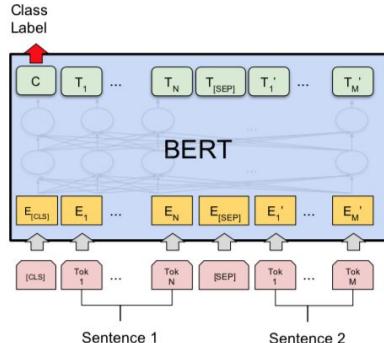
- BERT is basically a trained Transformer Encoder¹ stack.
- The first input token is supplied with a special (classification embedding) [CLS] token for classification task to represent the entire sentence.
- The output is generated from only this special [CLS] token.



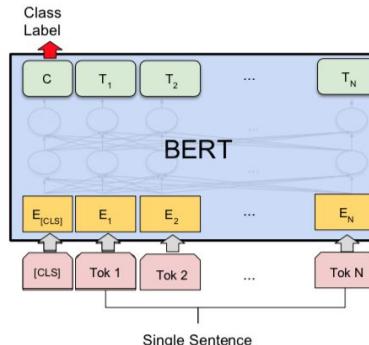
¹Vaswani et al., Attention Is All You Need, NIPS 2017, <https://arxiv.org/pdf/1706.03762.pdf>

Phase2: Supervised Phase

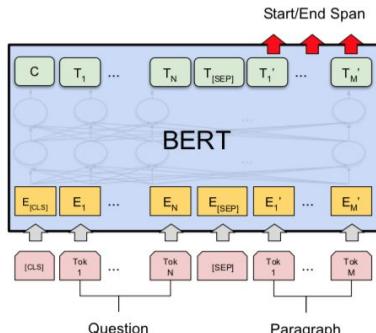
E.g., other tasks



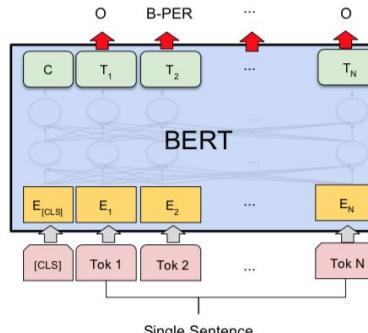
(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC,
RTE, SWAG



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



(c) Question Answering Tasks:
SQuAD v1.1



(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER



Pretrained BERT (English)

<https://github.com/google-research/bert>

BERT

***** New March 11th, 2020: Smaller BERT Models *****

This is a release of 24 smaller BERT models (English only, uncased, trained with WordPiece masking) referenced in [Well-Read Students Learn Better: On the Importance of Pre-training Compact Models](#).

We have shown that the standard BERT recipe (including model architecture and training objective) is effective on a wide range of model sizes, beyond BERT-Base and BERT-Large. The smaller BERT models are intended for environments with restricted computational resources. They can be fine-tuned in the same manner as the original BERT models. However, they are most effective in the context of knowledge distillation, where the fine-tuning labels are produced by a larger and more accurate teacher.

Our goal is to enable research in institutions with fewer computational resources and encourage the community to seek directions of innovation alternative to increasing model capacity.

You can download all 24 from [here](#), or individually from the table below:

	H=128	H=256	H=512	H=768
L=2	2/128 (BERT-Tiny)	2/256	2/512	2/768
L=4	4/128	4/256 (BERT-Mini)	4/512 (BERT-Small)	4/768
L=6	6/128	6/256	6/512	6/768
L=8	8/128	8/256	8/512 (BERT-Medium)	8/768
L=10	10/128	10/256	10/512	10/768
L=12	12/128	12/256	12/512	12/768 (BERT-Base)

New bert model



Pretrained BERT (Multilingual)

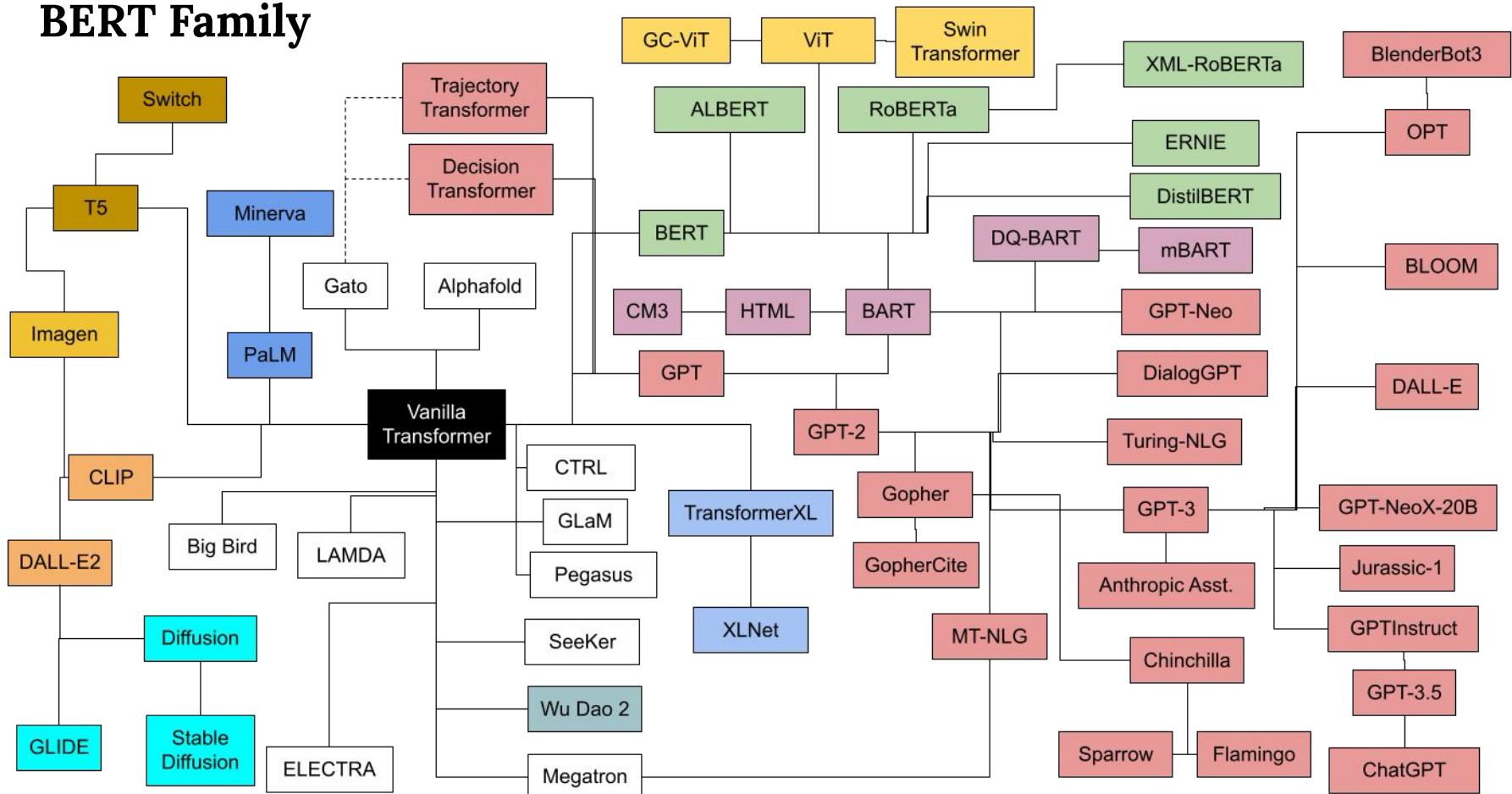
<https://github.com/google-research/bert/blob/master/multilingual.md>

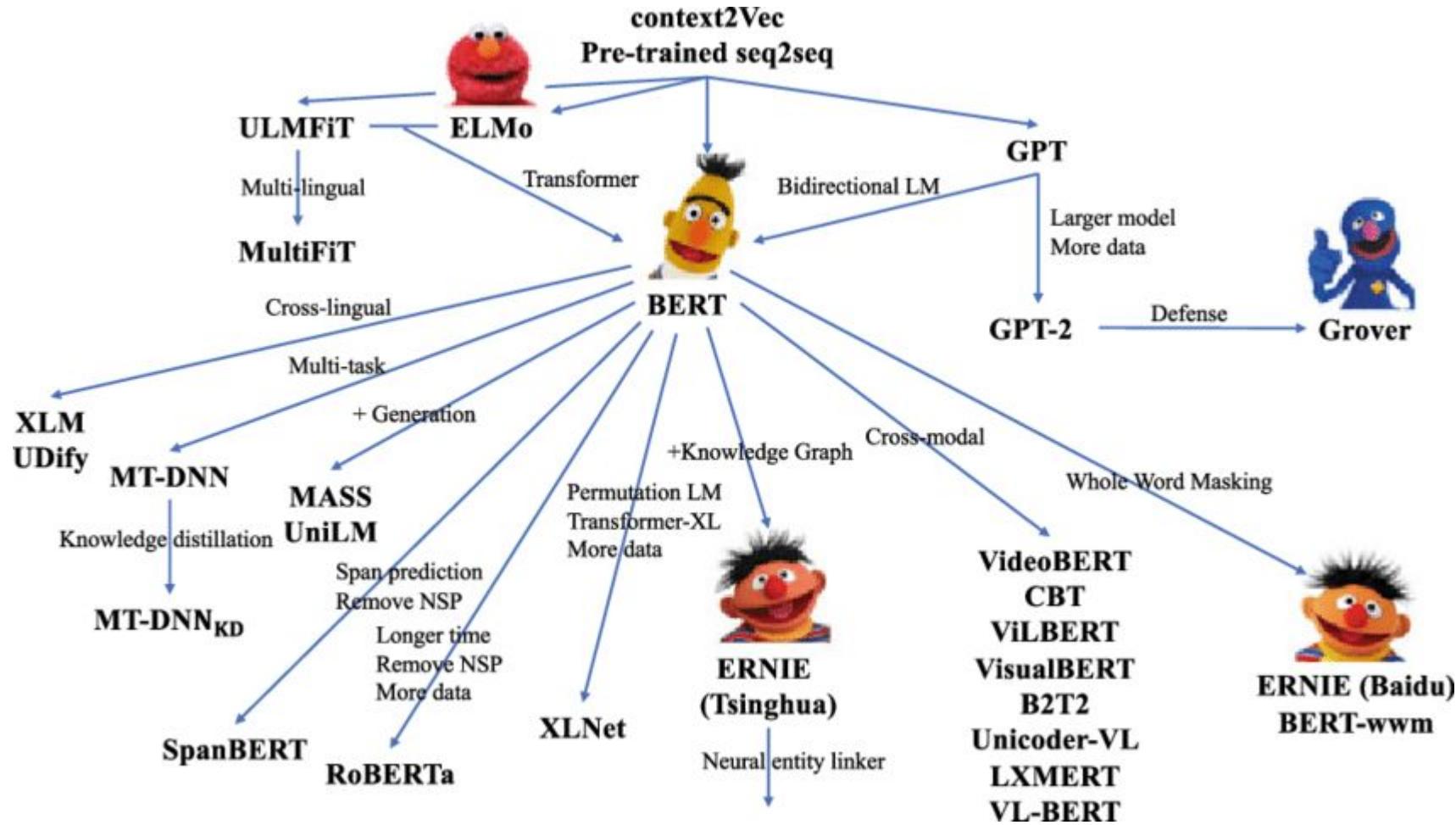
Models

There are two multilingual models currently available. We do not plan to release more single-language models, but we may release BERT-Large versions of these two in the future:

- **BERT-Base, Multilingual Cased (New, recommended)** : 104 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- **BERT-Base, Multilingual Uncased (Orig, not recommended)** : 102 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- **BERT-Base, Chinese** : Chinese Simplified and Traditional, 12-layer, 768-hidden, 12-heads, 110M parameters

BERT Family





Roberta (Robustly optimized BERT approach) 2019

A trick and tuning study

Dynamic masking > static

Next sentence prediction is not optimal

Larger batch + higher learning rate

Masking	SQuAD 2.0	MNLI-m	SST-2
reference	76.3	84.3	92.8
<i>Our reimplementation:</i>			
static	78.3	84.3	92.5
dynamic	78.7	84.0	92.9
bsz	steps	lr	ppl
256	1M	1e-4	3.99
2K	125K	7e-4	3.68
8K	31K	1e-3	3.77
			MNLI-m
			SST-2

RoBERTa: A Robustly Optimized BERT Pretraining Approach

Used in WangchanBERTa

Highly recommended to go watch <https://www.youtube.com/watch?v=kXPMXLX0vfYU>
for more info on WangchanBERTa

Based on Aj.Ekapol's slide

<https://arxiv.org/abs/1907.11692>

DistilBERT 2019

Knowledge distillation to get smaller models

Reduce the # of transformer layers by half. Use tricks in Roberta.

Use KL-divergence between teacher and student model

“Cheaper training”
eight 16GB V100 GPUs for approximately three and a half days

	Nb of parameters (millions)	Inference Time (s)
GLUE BASELINE (ELMo + BiLSTMs)	180	895
BERT base	110	668
DistilBERT	66	410

	Macro Score	CoLA	MNLI	MNLI-MM	MRPC		QNLI	QQP		RTE	SST-2	STS-B		WNLI
		mcc	acc	acc	acc	f1	acc	acc	f1	acc	acc	pearson	spearmanr	acc
GLUE BASELINE (ELMo + BiLSTMs)	68.7	44.1	68.6 (avg)		70.8	82.3	71.1	88.0	84.3	53.4	91.5	70.3	70.5	56.3
BERT base	78.0	55.8	83.7	84.1	86.3	90.5	91.1	90.9	87.7	68.6	92.1	89.0	88.6	43.7
DistilBERT	75.2	42.5	81.6	81.1	82.4	88.3	85.5	90.6	87.7	60.0	92.7	84.5	85.0	55.6

10000

Millions of parameter

7500

5000

2500

ELMo

94

April 2018



GPT

110

July 2018



BERT-Large

340

October 2018



GPT-2

1500



Transformer

ELMo

465

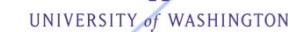
January 2019



MT-DNN

330

April 2019



Grover-Mega

1500



XLM

665



RoBERTa

355



XLNET

340



Carnegie

Mellon

University

MegatronLM

8300



DistilBERT

0

ALBERT 2019

Want higher hidden units without growing the model. Factorized embedding matrix

$$O(V \times H) \text{ to } O(\sqrt{V} \times E + E \times \sqrt{H})$$

Share attention layer parameters across layers. More stable training as a side effect.

V = Vocab, H = Hidden, E = Lower Dimensional Embedding space

Model	Parameters	Layers	Hidden	Embedding	Parameter-sharing	Avg	Speedup
BERT	base	108M	12	768	768	False	82.1
	large	334M	24	1024	1024	False	85.1
	xlarge	1270M	24	2048	2048	False	76.7
ALBERT	base	12M	12	768	128	True	80.1
	large	18M	24	1024	128	True	82.4
	xlarge	59M	24	2048	128	True	85.5
	xxlarge	233M	12	4096	128	True	88.7

Some experiments show dropout hurt performance

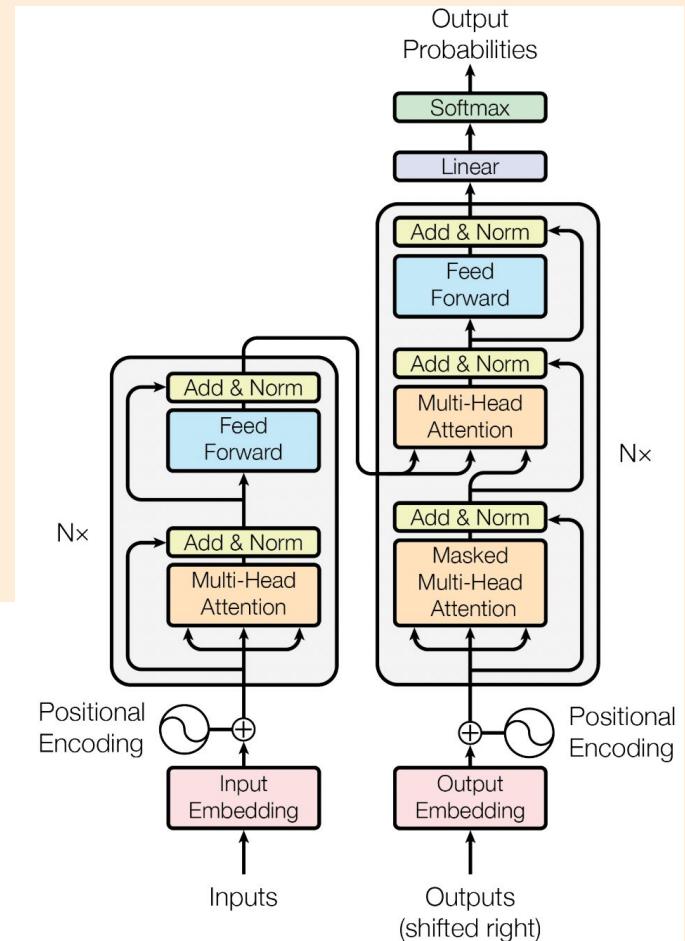
<https://arxiv.org/abs/1909.11942>

BART [2019]

BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, Luke Zettlemoyer

We present BART, a denoising autoencoder for pretraining sequence-to-sequence models. BART is trained by (1) corrupting text with an arbitrary noising function, and (2) learning a model to reconstruct the original text. It uses a standard Transformer-based neural machine translation architecture which, despite its simplicity, can be seen as generalizing BERT (due to the bidirectional encoder), GPT (with the left-to-right decoder), and many other more recent pretraining schemes. We evaluate a number of noising approaches, finding the best performance by both randomly shuffling the order of the original sentences and using a novel in-filling scheme, where spans of text are replaced with a single mask token. BART is particularly effective when fine tuned for text generation but also works well for comprehension tasks. It matches the performance of RoBERTa with comparable training resources on GLUE and SQuAD, achieves new state-of-the-art results on a range of abstractive dialogue, question answering, and summarization tasks, with gains of up to 6 ROUGE. BART also provides a 1.1 BLEU increase over a back-translation system for machine translation, with only target language pretraining. We also report ablation experiments that replicate other pretraining schemes within the BART framework, to better measure which factors most influence end-task performance.

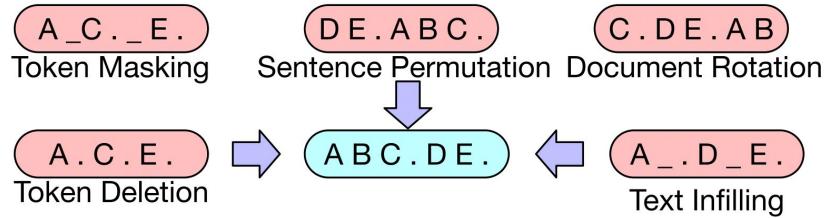


BART 2019

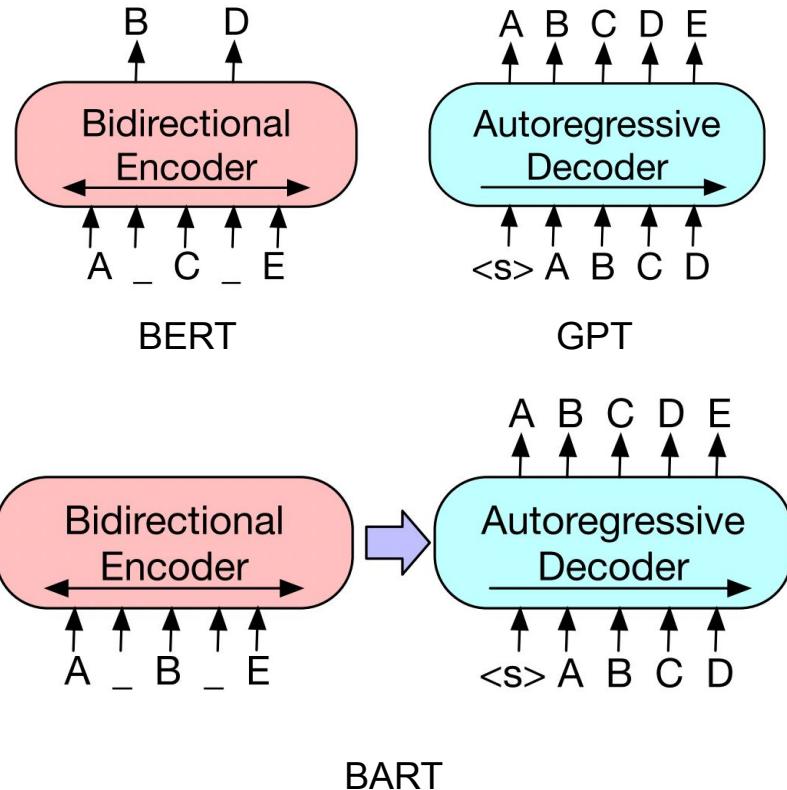
A model comprises of both Transformer encoders *and* decoders

BART optimizes on denoising corrupted inputs

By using both encoders and decoders, BART combines the bidirectionality of BERT and autoregressive ability of GPT.



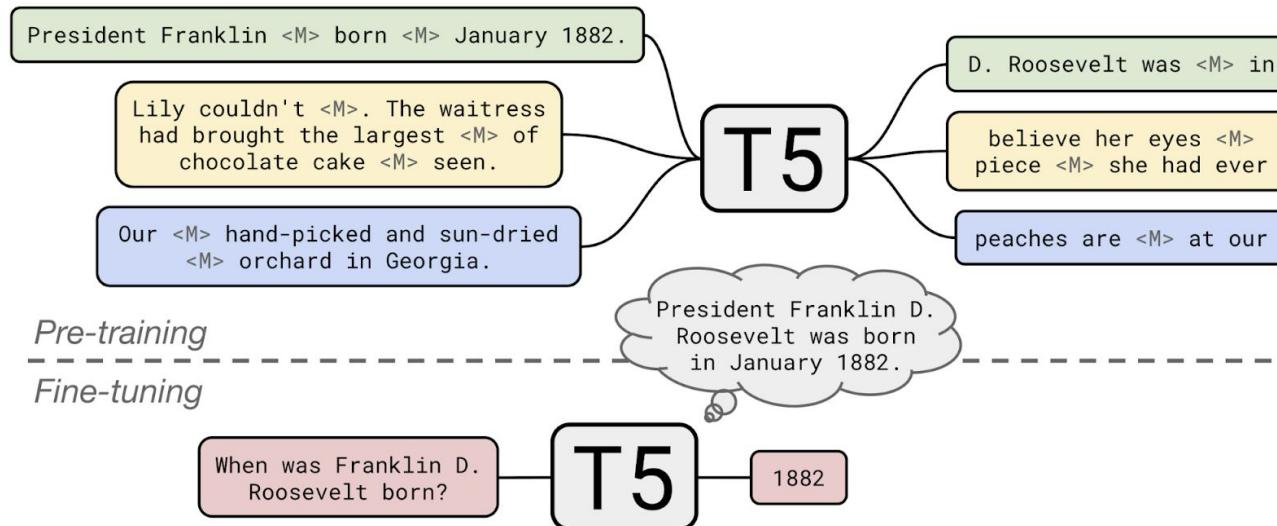
Input noising scheme



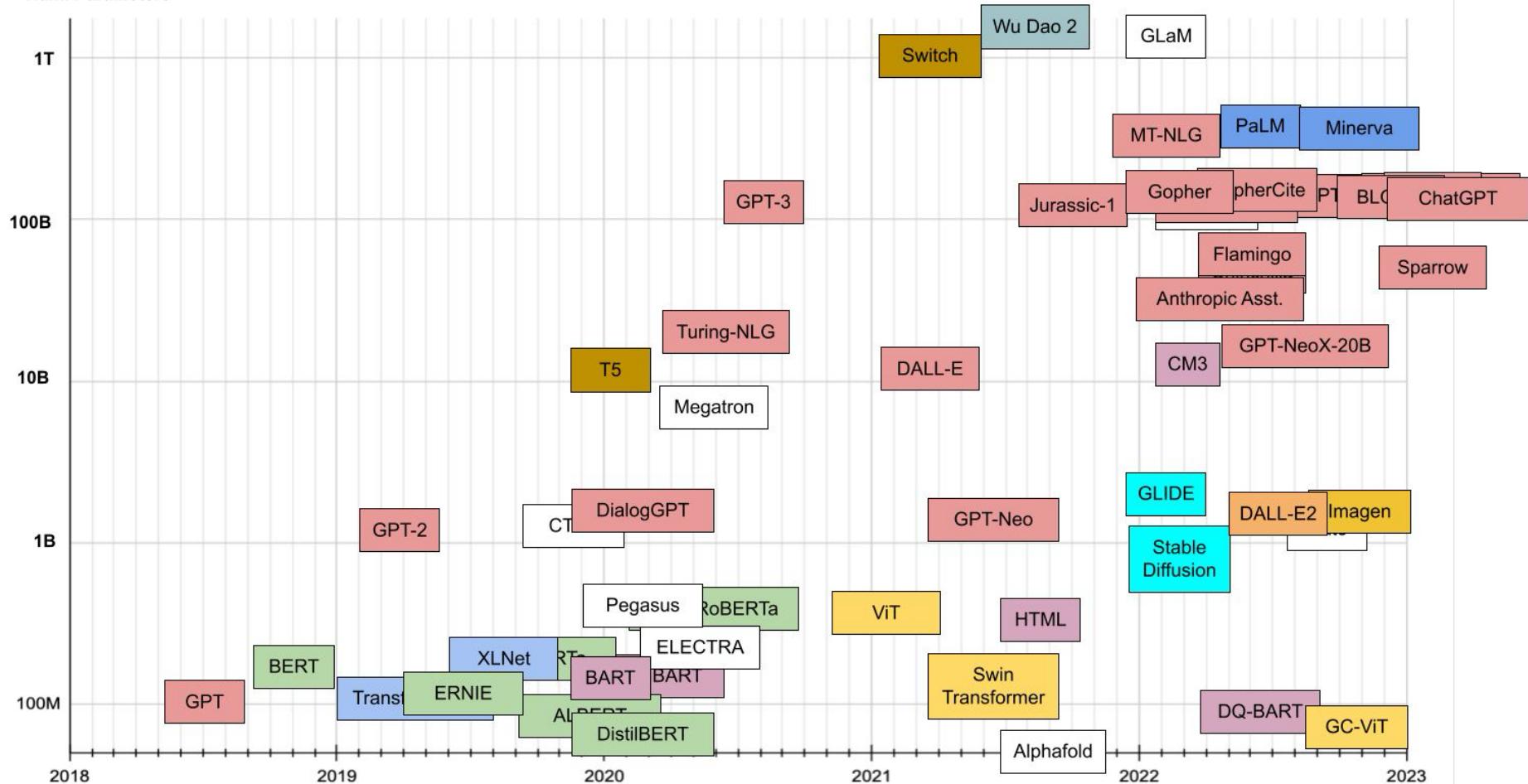
Text-to-text Transfer Transformer (T5) 2019

T5 is also an encoder-decoder model

It pretrains on the **span-corruption** objective.



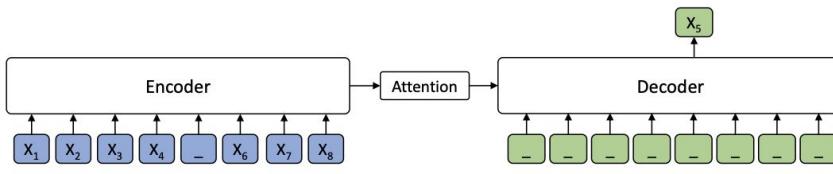
Num. Parameters



Pre-trained transformer types (by training method)

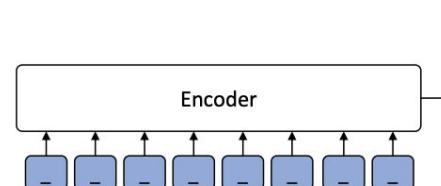
Encoder only (autoencoder)

- BERT, ALBERT, RoBERTa
- Seq classification, token classification
- Masked words and predict



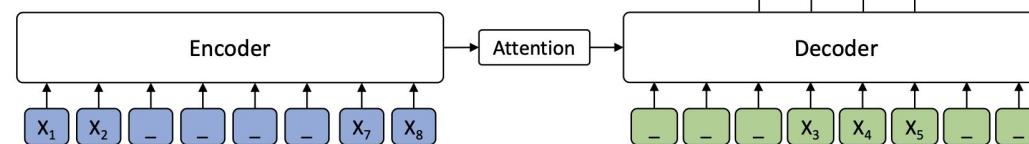
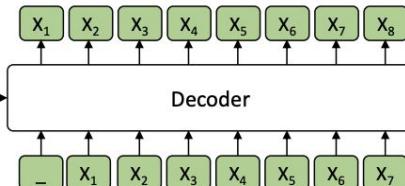
Encoder-decoder (seq2seq)

- MASS, BART, T5
- Machine translation, text summary
- Mask phrases



Decoder only (autoregressive)

- GPT, CTRL
- Text generation
- Predict next word



When to use which?

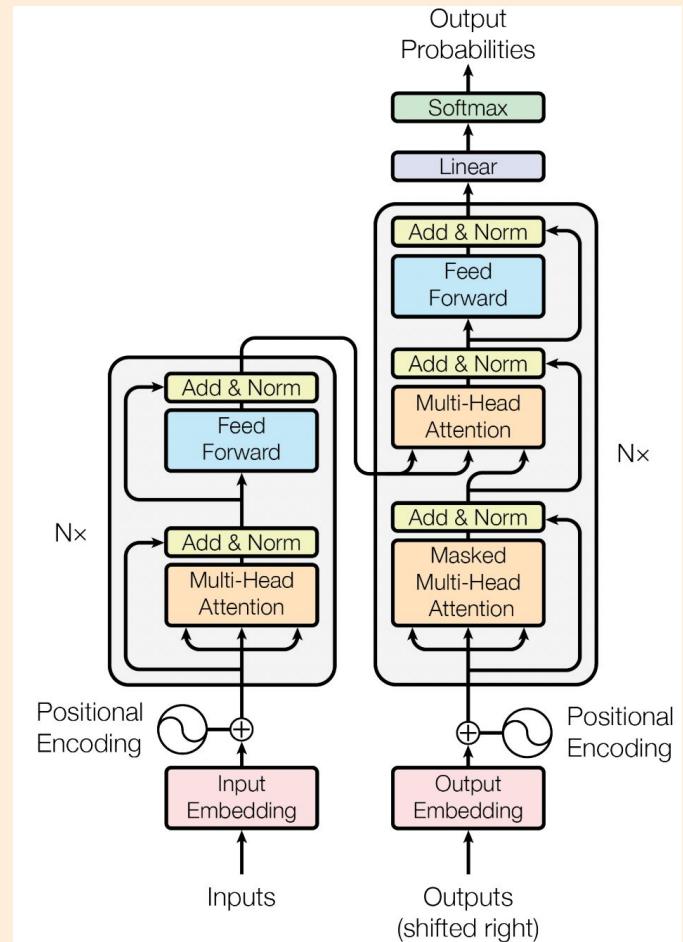
Text Generation: GPT models

Sequence-to-sequence (e.g. text summarization, translation): BART, T5, or similar

Text classification/ Token classification: RoBERTa

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	94.6/89.4	90.2	96.4

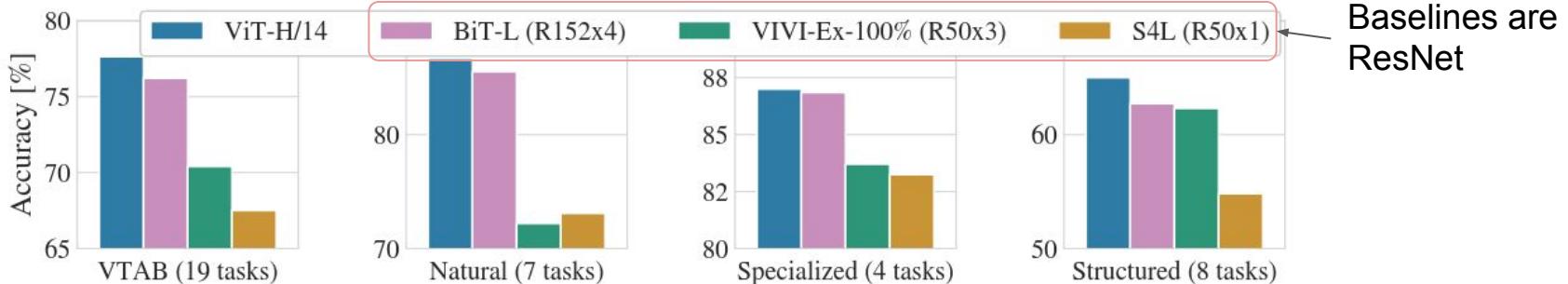
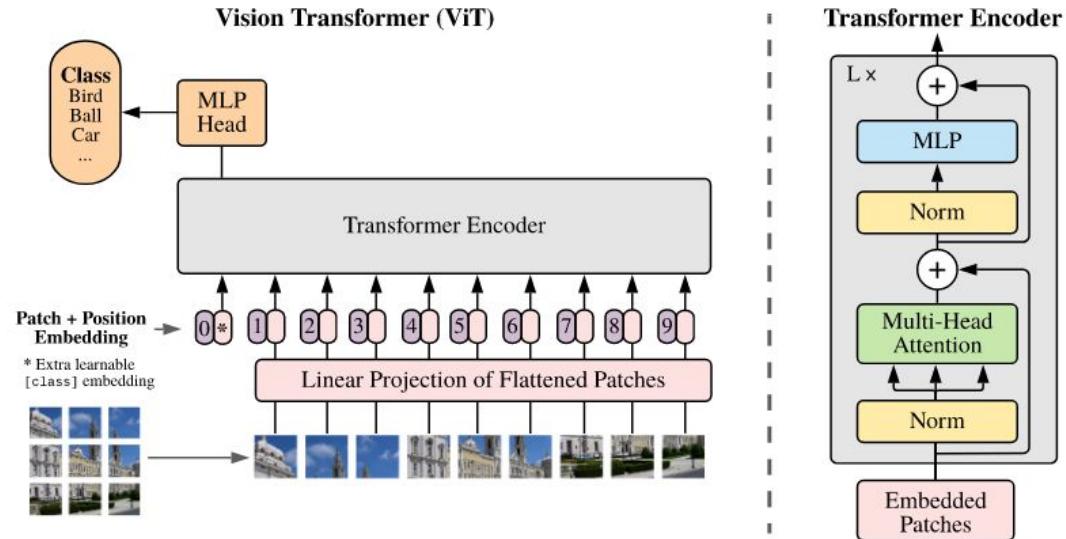
Beyond NLP



Beyond NLP

1) Computer Vision

ViT uses only encoder.

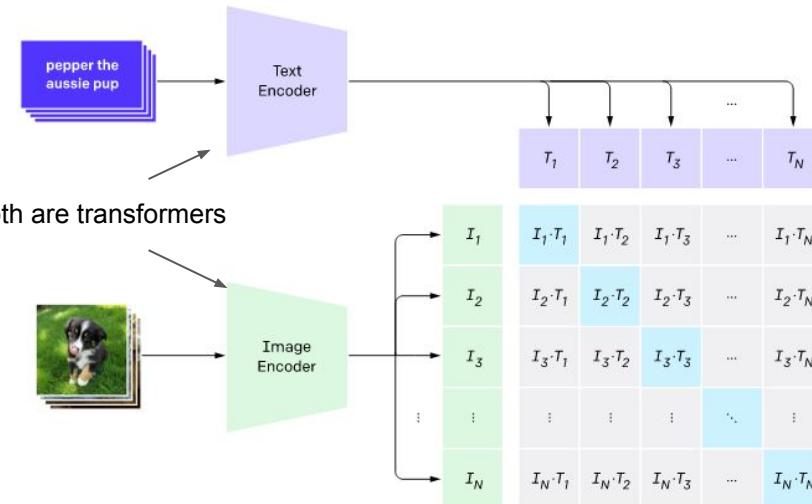


Beyond NLP

2) Text-Image (Multimodal)

Image search, i.e., CLIP

1. Contrastive pre-training



FOOD101

guacamole (90.1%) Ranked 1 out of 101 labels



- ✓ a photo of **guacamole**, a type of food.
- ✗ a photo of **ceviche**, a type of food.
- ✗ a photo of **edamame**, a type of food.
- ✗ a photo of **tuna tartare**, a type of food.
- ✗ a photo of **hummus**, a type of food.

Beyond NLP

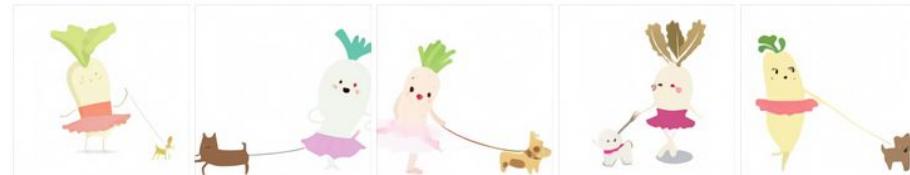
3) Text-to-Image

Diffusion, e.g.,

- DALL-E
- DALL-E3
- Stable Diffusion
- Midjourney

TEXT PROMPT an illustration of a baby daikon radish in a tutu walking a dog

AI-GENERATED IMAGES



Edit prompt or view more images↓

TEXT PROMPT an armchair in the shape of an avocado....

AI-GENERATED IMAGES



Edit prompt or view more images↓

TEXT PROMPT a store front that has the word 'openai' written on it....

AI-GENERATED IMAGES



Edit prompt or view more images↓