



# Transformer and Applications

Yuan YAO

HKUST

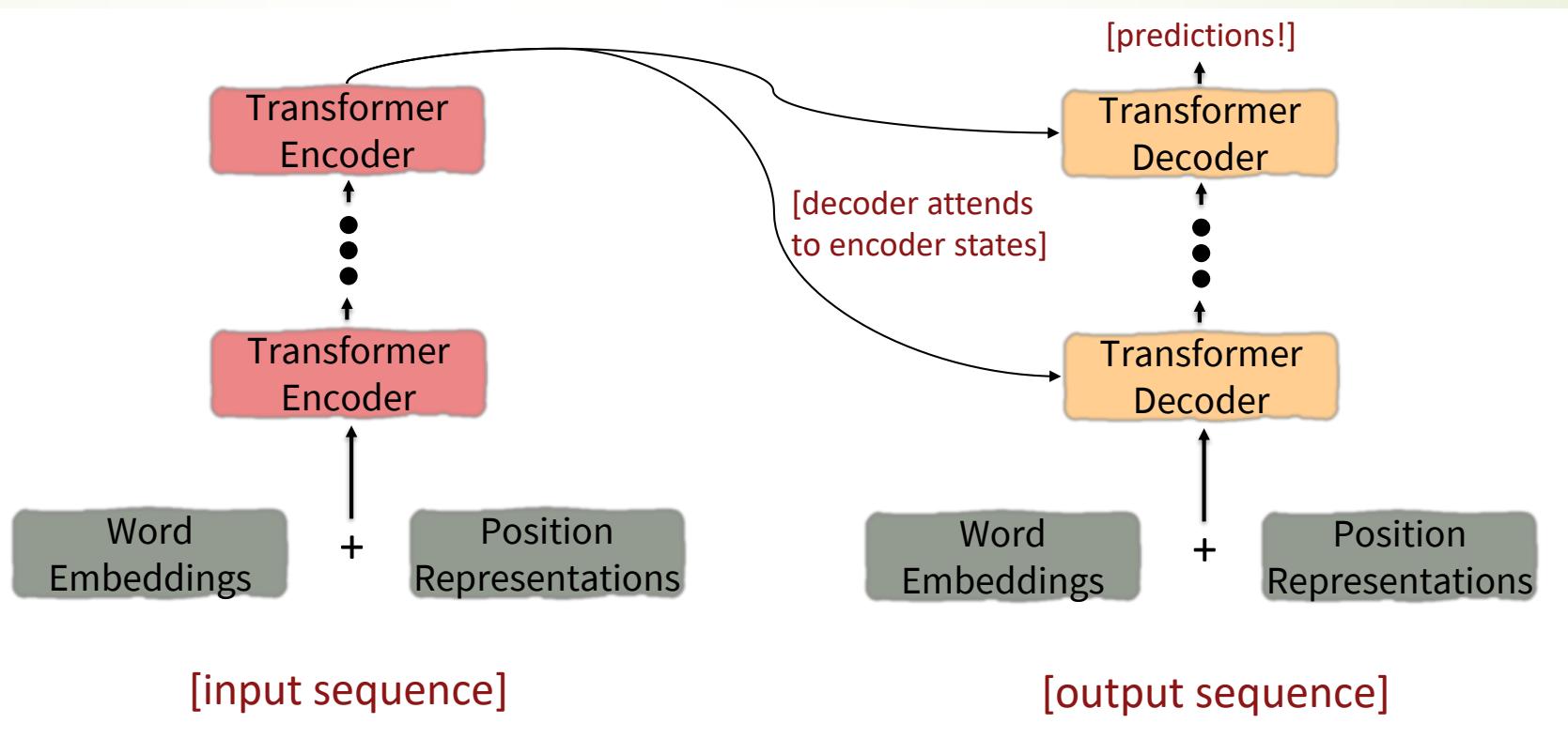
# Summary

- ▶ We have shown:
  - ▶ CNN Architectures: LeNet5, Alexnet, VGG, GoogleNet, Resnet
  - ▶ Recurrent Neural Networks and LSTM (GRU)
  - ▶ Attention and Transformer
- ▶ Today:
  - ▶ **Applications of Transformer**
    - ▶ BERT, GPT, and ViT, etc.
- ▶ Reference:
  - ▶ Feifei Li, Stanford cs231n
  - ▶ Chris Manning, Stanford cs224n

# The Transformer Encoder-Decoder

[Vaswani et al. 2017]

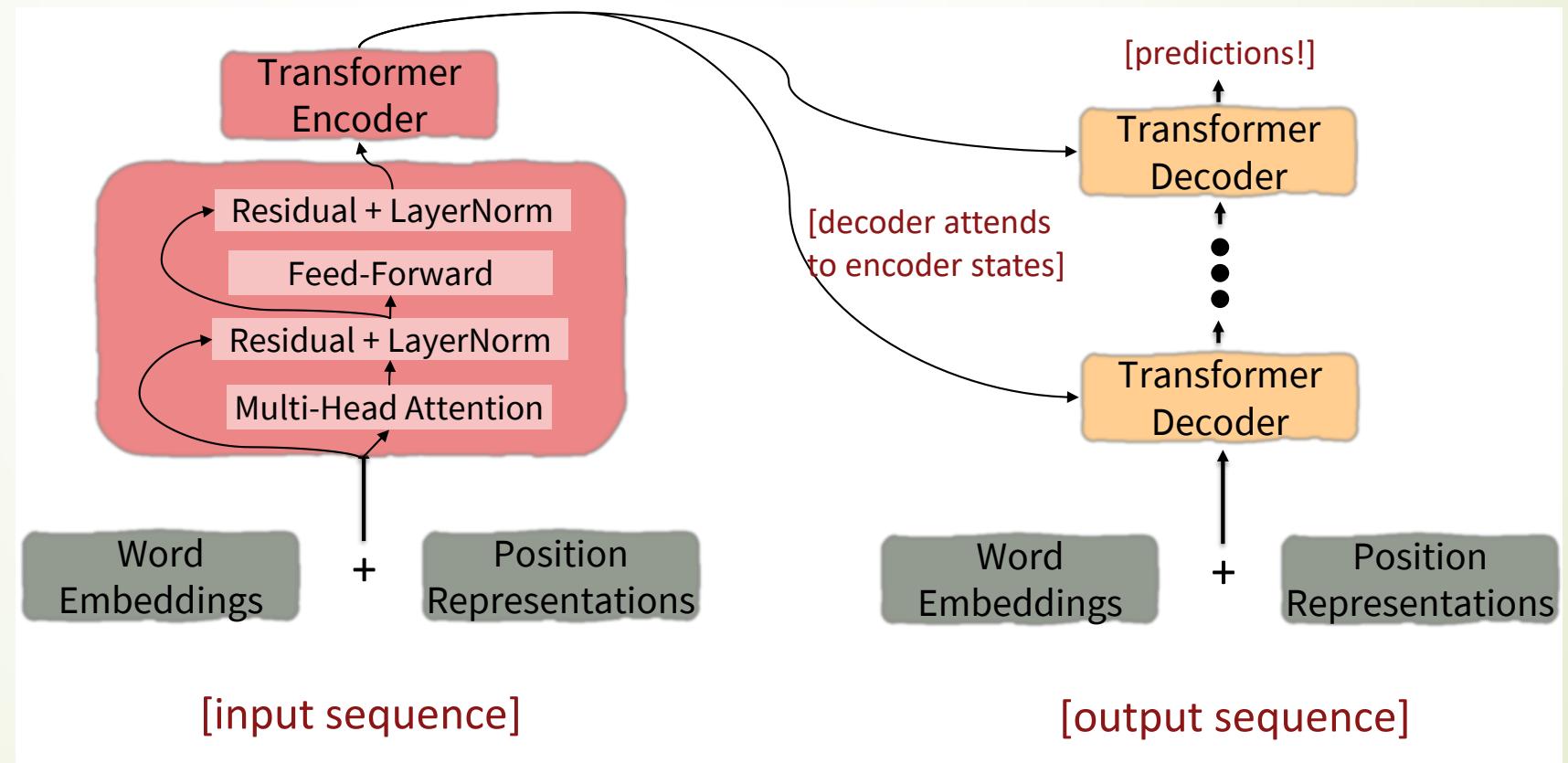
- ▶ Looking back at the whole model



# The Transformer Encoder-Decoder

[Vaswani et al. 2017]

- ▶ Looking back at the whole model



# The Transformer Encoder-Decoder

[Vaswani et al. 2017]

- ▶ Looking back at the whole model

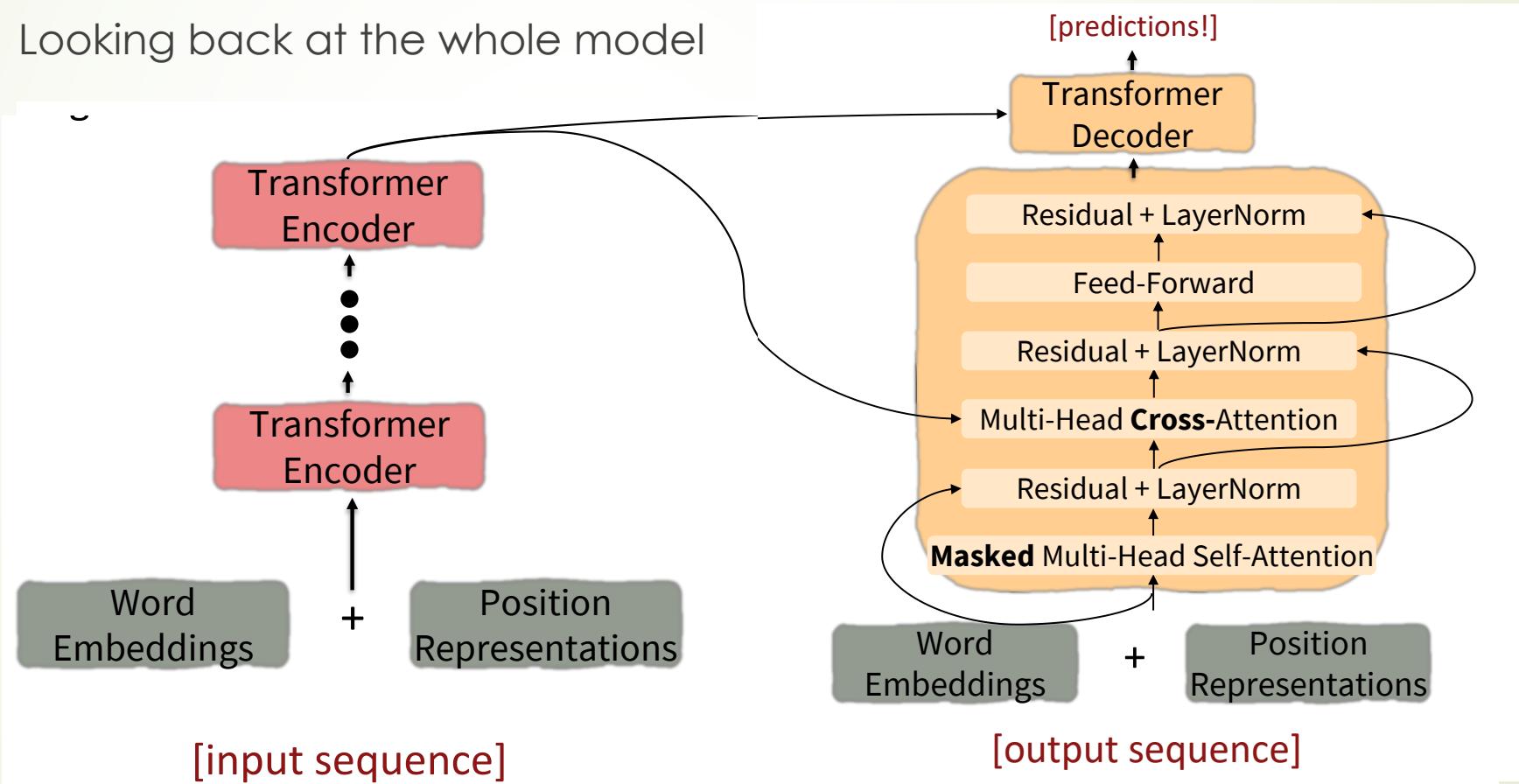


Table 2.1: In our network configurations, *Sublayer* refers to either a feed-forward neural network (FFN) or a self-attention module within a Transformer layer. The symbol  $d$  represents the size of the hidden states in the network. The position embedding at a specific position  $i$  is denoted by  $p_i$ . In the attention mechanism,  $A_{ij}$  signifies the attention score computed between a given query and its corresponding key. The difference in positions between the query and the key is represented by  $r_{i-j}$ , a learnable scalar value. Finally, the term  $R_{\theta,t}$  refers to a rotary matrix, which rotates by an angle determined by multiplying  $t$  by  $\theta$ .

Configuration	Method	Equation
Normalization position	Post Norm [1]	$\text{Norm}(\mathbf{x} + \text{Sublayer}(\mathbf{x}))$
	Pre Norm [2]	$\mathbf{x} + \text{Sublayer}(\text{Norm}(\mathbf{x}))$
	Sandwich Norm [3]	$\mathbf{x} + \text{Norm}(\text{Sublayer}(\text{Norm}(\mathbf{x})))$
Normalization method	LayerNorm [4]	$\frac{\mathbf{x} - \mu}{\sqrt{\sigma}} \cdot \gamma + \beta, \mu = \frac{1}{d} \sum_{i=1}^d \mathbf{x}_i, \sigma = \sqrt{\frac{1}{d} \sum_{i=1}^d (\mathbf{x}_i - \mu)^2}$
	RMSNorm [5]	$\frac{\mathbf{x}}{\text{RMS}(\mathbf{x})} \cdot \gamma, \text{RMS}(\mathbf{x}) = \sqrt{\frac{1}{d} \sum_{i=1}^d \mathbf{x}_i^2}$
	DeepNorm [6]	$\text{LayerNorm}(\alpha \cdot \mathbf{x} + \text{Sublayer}(\mathbf{x}))$
Activation function	ReLU [7]	$\text{ReLU}(\mathbf{x}) = \max(0, \mathbf{x})$
	GeLU [8]	$\text{GeLU}(\mathbf{x}) = 0.5\mathbf{x} \otimes \left( 1 + \tanh \left( \sqrt{\frac{2}{\pi}} (\mathbf{x} + 0.044715\mathbf{x}^3) \right) \right)$
	Swish [9]	$f(x) = x \cdot \frac{1}{1+e^{-x}}$
	SwiGLU [10]	$f(x) = x \odot \sigma(Wx + b)$
	GeGLU [10]	Similar to SwiGLU with GeLU
Positional embeddings	Absolute [1]	$x_i = x_i + p_i$
	Relative [11]	$A_{ij} = W_q x_i x_j^T W_k + r_{i-j}$
	RoPE [12]	$A_{ij} = W_q x_i R_{\theta,i-j} x_j^T W_k$
	Alibi [13]	$A_{ij} = W_q x_i x_j^T W_k - m(i-j)$

Key: [1] ([Vaswani et al., 2017](#)), [2] ([Radford et al., 2019](#)), [3] ([Ding et al., 2021](#)), [4] ([Ba et al., 2016](#)),  
 [5] ([Zhang and Sennrich, 2019](#)), [6] ([Wang et al., 2022](#)), [7] ([Nair and Hinton, 2010](#)), [8] ([Wang et al., 2019](#)),  
 [9] ([Ramachandran et al., 2017](#)), [10] ([Shazeer, 2020](#)), [11] ([Raffel et al., 2020](#)), [12] ([Su et al., 2021](#)),  
 [13] ([Press et al., 2021](#))

# Empirical advantages of Transformer vs. LSTM

- ▶ One major benefit of self-attention over recurrence was that it's highly **parallelizable**.
- ▶ 1. Self-attention == no locality bias
  - ▶ Long-distance context has “equal opportunity”
- ▶ 2. Single multiplication per layer == efficiency on TPU

**Transformer**

x_0_0	x_0_1	x_0_2	x_0_3
x_1_0	x_1_1	x_1_2	x_1_3

$\times$  W

**LSTM**

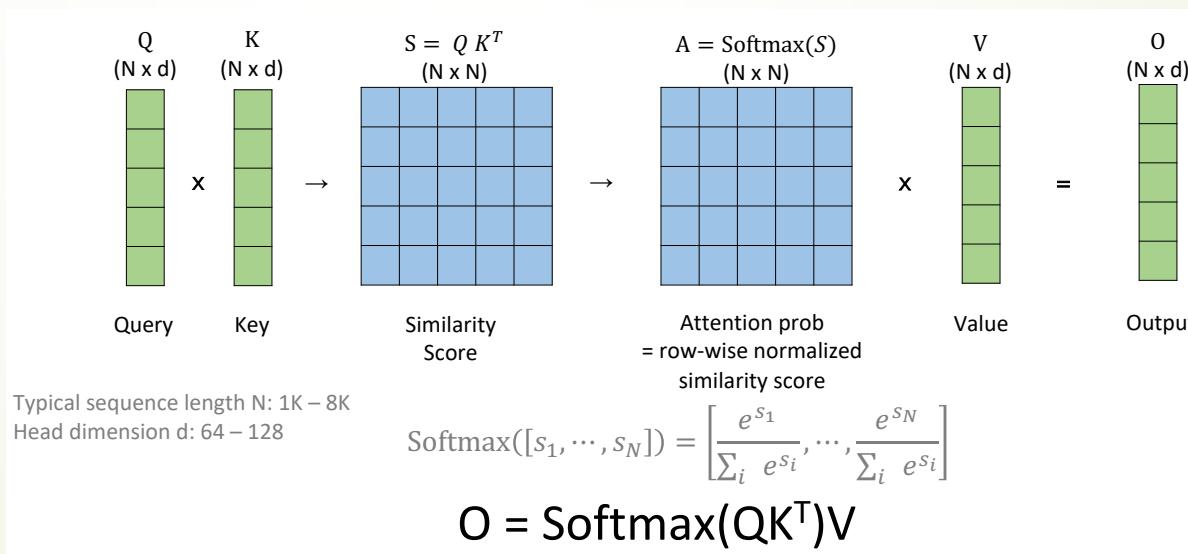
x_0_0	x_0_1	x_0_2	x_0_3
x_1_0	x_1_1	x_1_2	x_1_3

$\times$  W

# Major disadvantage of Transformer

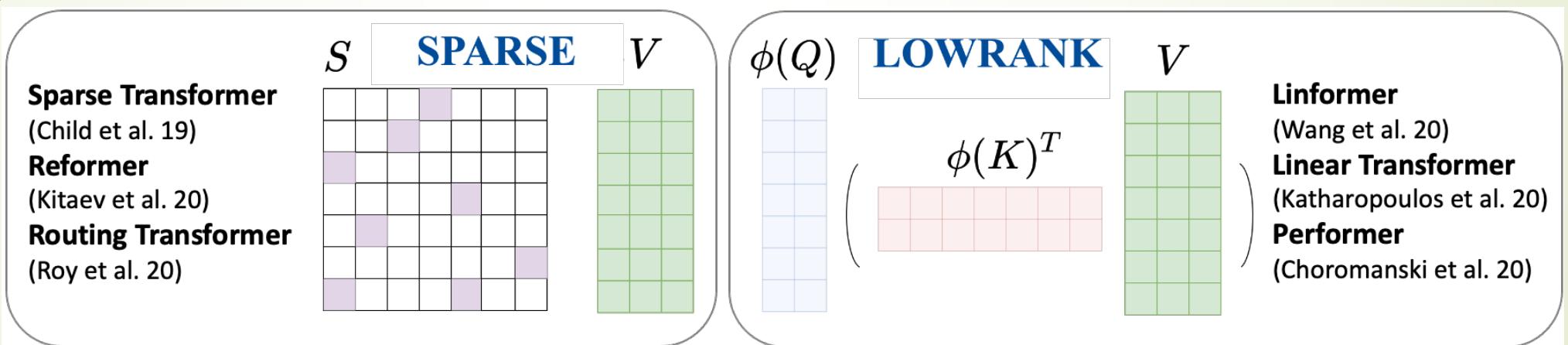
## ► Quadratic compute in self-attention:

- ▶ However, its total number of operations grows as  $O(N^2 d)$ , quadratic with N, where N is the sequence length, and d is the dimensionality.
- ▶ Think of  $d$  as around **1,000** (though for large language models it's much larger!).
  - So, for a single (short) sentence,  $N \leq 30$ ;  $N^2 \leq 900$ .
  - In practice, we set a bound like  $N = 512$ .
  - **But what if we'd like  $N \geq 50,000$ ?** For example, to work on long documents?
- ▶ In contrast, for recurrent models, it only grew linearly!



# Approximate Attention: sparse or low-rank

- ▶ Approximate attention exploits sparse or low-rank approximations of full attention, as a tradeoff of quality for fewer operations.



# Efficient Transformers

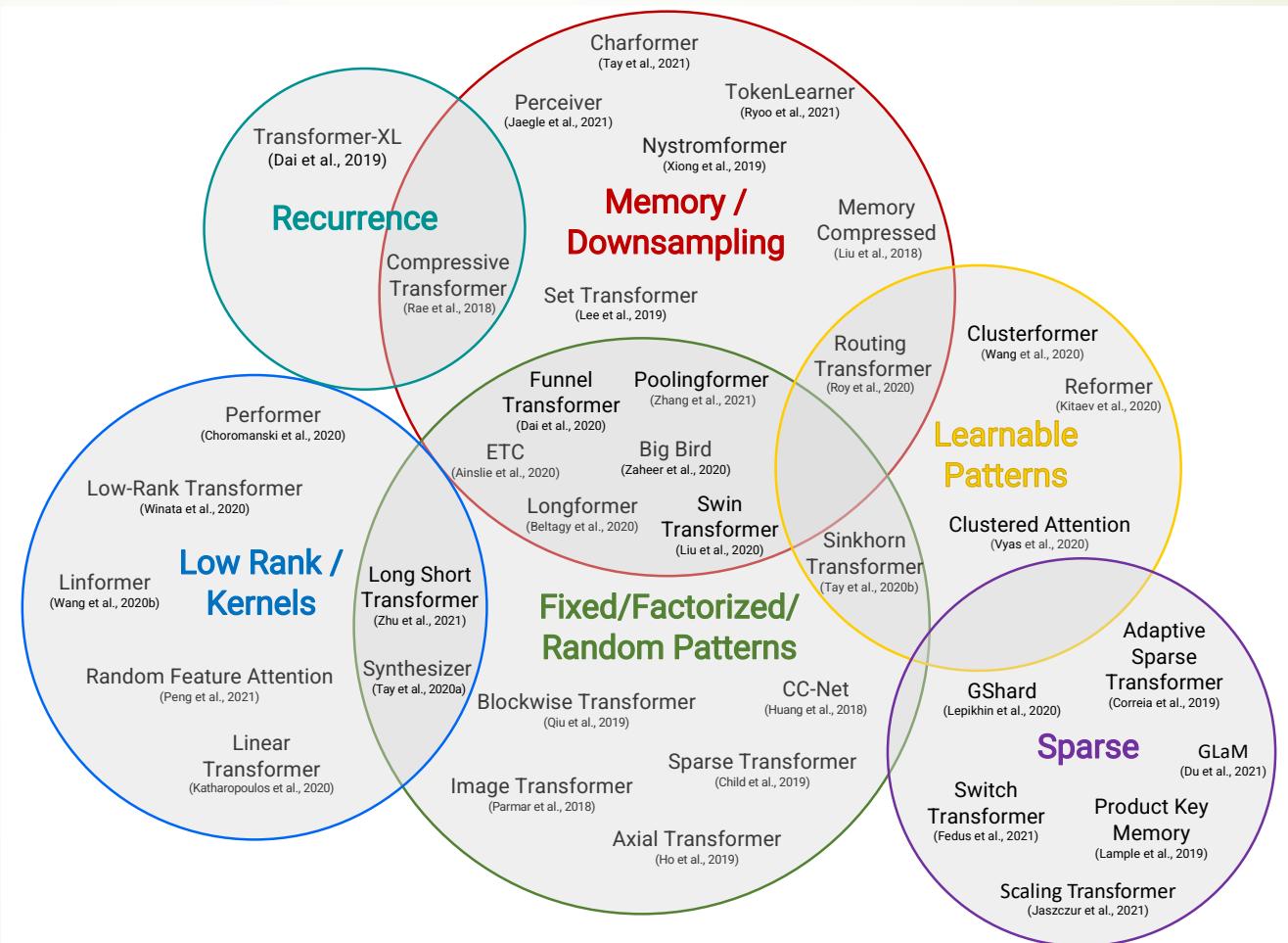
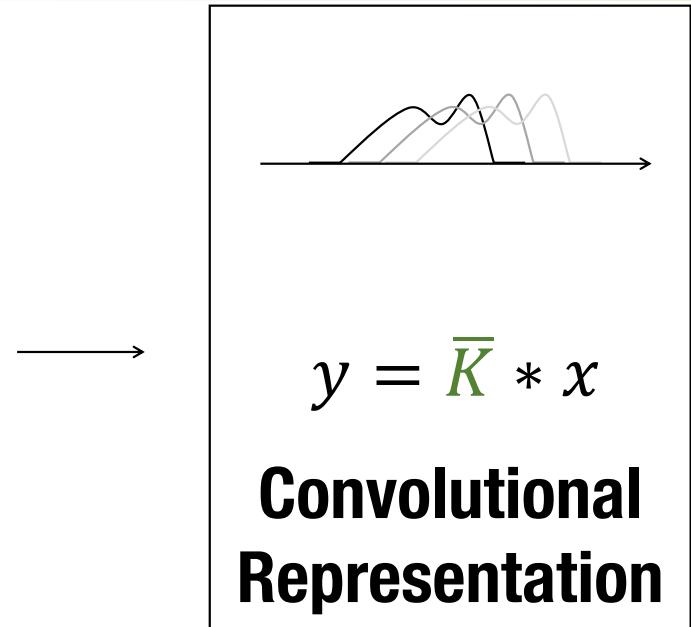
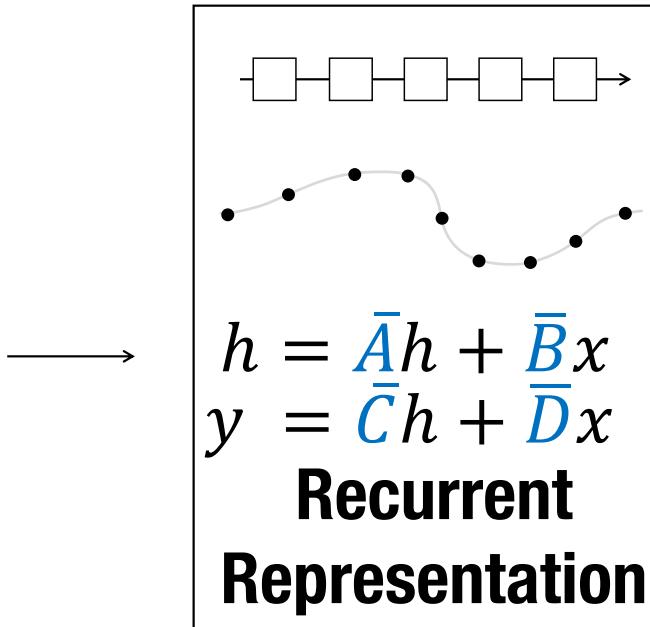
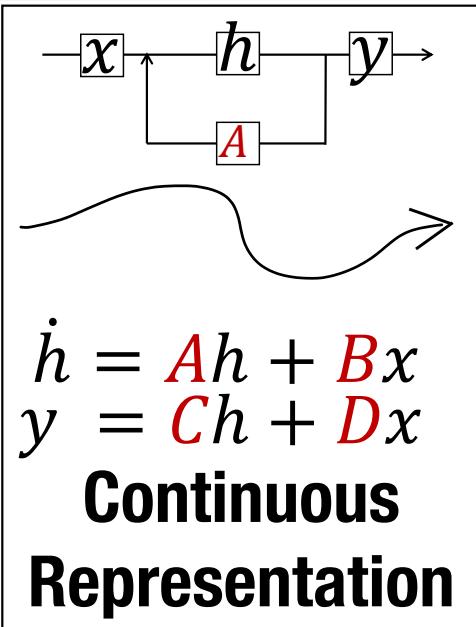


Figure 2: Taxonomy of Efficient Transformer Architectures.

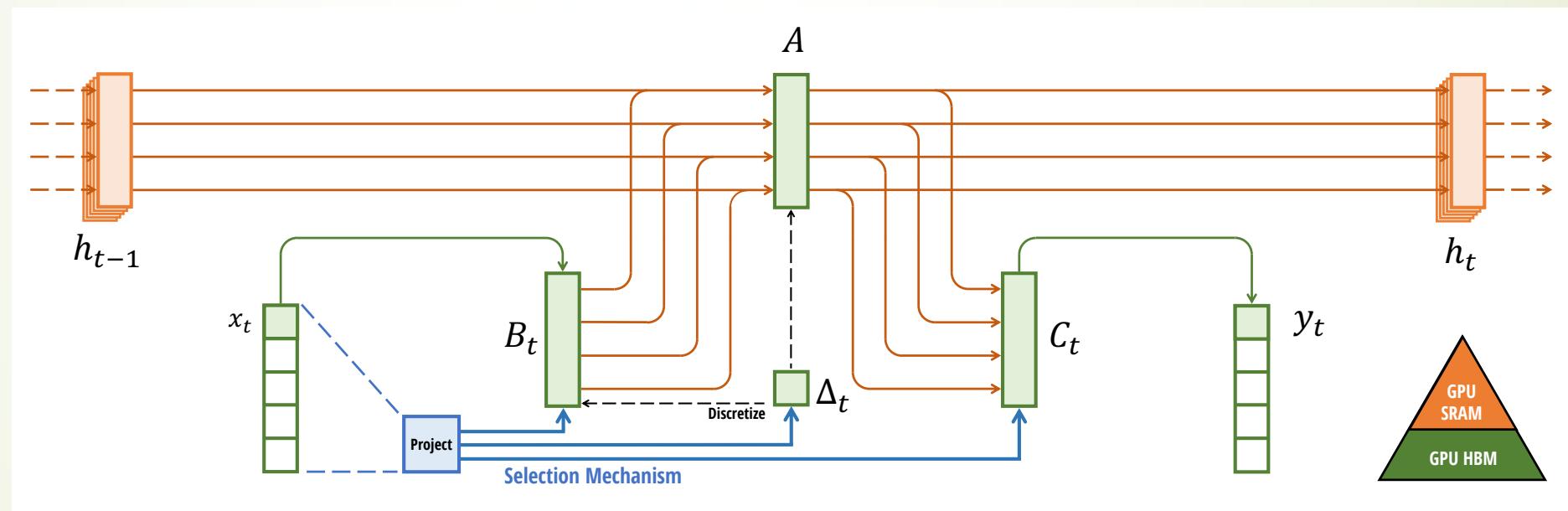
# Return of Linear Dynamical System as State-Space Models

- ▶ Albert Gu. Modeling Sequences with Structured State Spaces, CMU PhD Dissertation.
- ▶ SSMs is a special case of RNN and 1-D CNN
  - ▶ Recurrent models operate **linearly** with sequence length
  - ▶ Convolution can be implemented in a **parallel** way
  - ▶ Yet, it lacks “attention” as a **selection mechanism depending on inputs?**



# Mamba: Selection Mechanism for State-Space Models

- ▶ Parameters depend on input!
- ▶ It can handle long context sequences (**N~1M** for DNA)!



# Mixture of Experts (MoE)

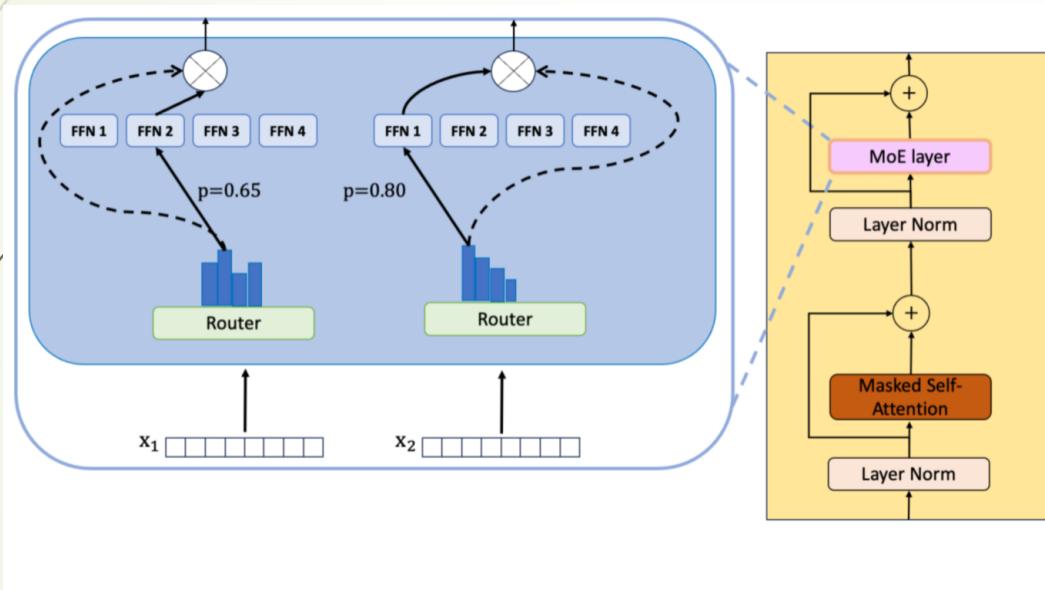
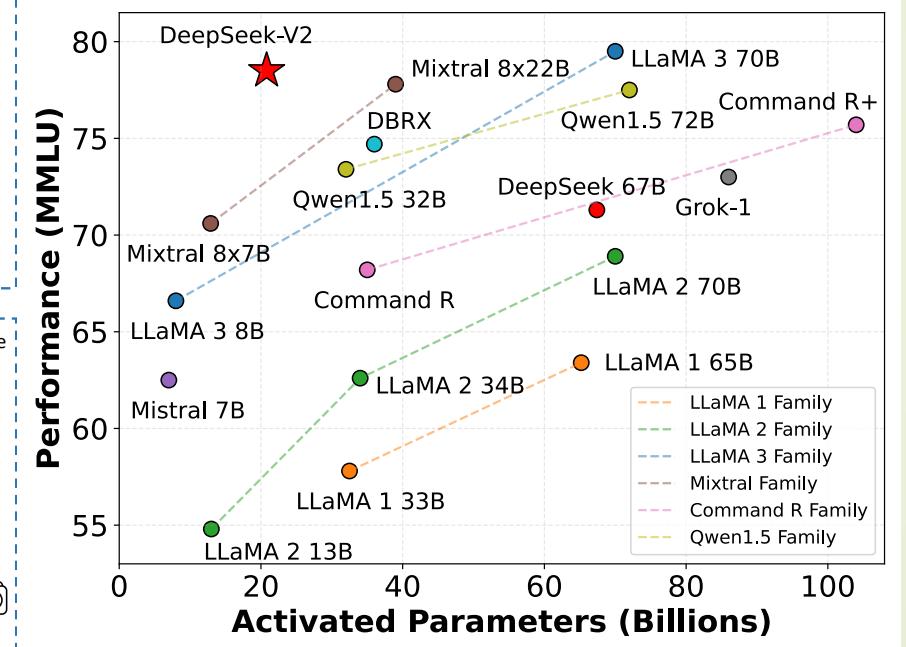
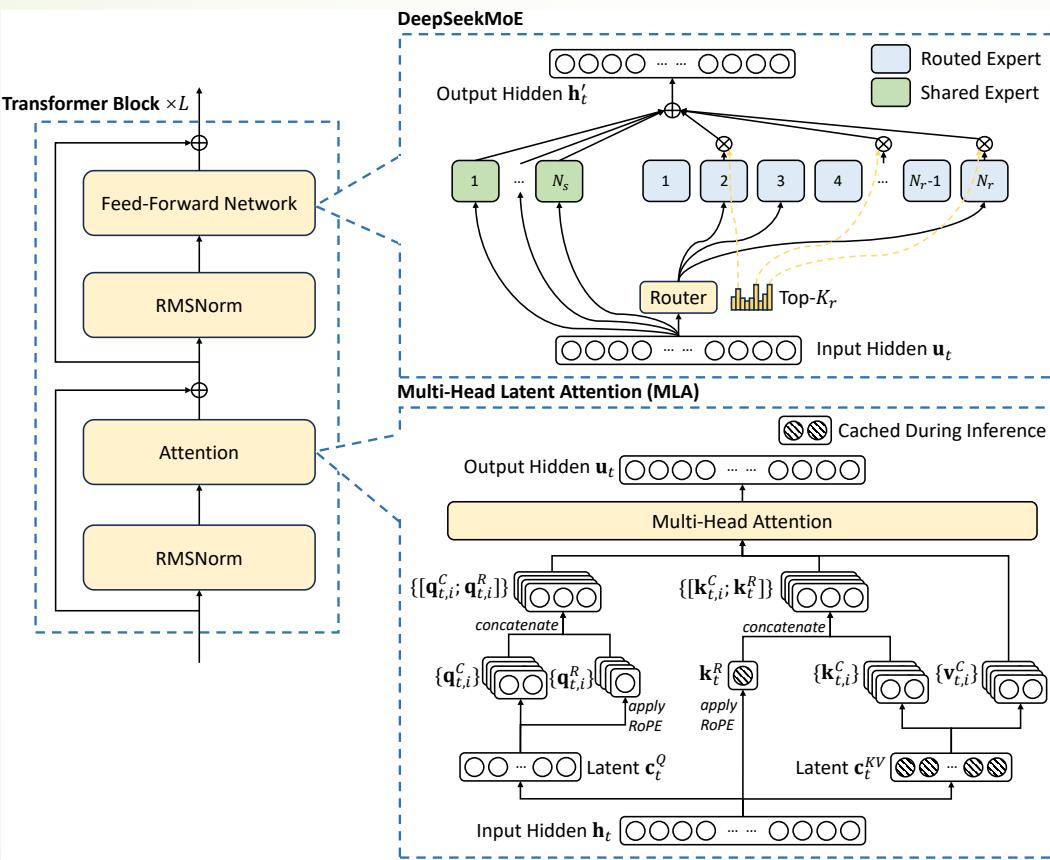


Fig. 2.9: Mixture-of-experts variant of the Transformer architecture.

- MoE layer replaces the standard feed-forward blocks by multiple parallel ‘experts’ as feed-forward blocks weighted by probability gates.
- MoE architecture simultaneously activates only a few experts. This sparse activation allows the architecture to support larger model sizes without a proportional increase in computational demand, maintaining efficient performance.

Shazeer, Noam; Mirhoseini, Azalia; Maziarz, Krzysztof; Davis, Andy; Le, Quoc; Hinton, Geoffrey; Dean, Jeff (2017). "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer". [arXiv:1701.06538](https://arxiv.org/abs/1701.06538)

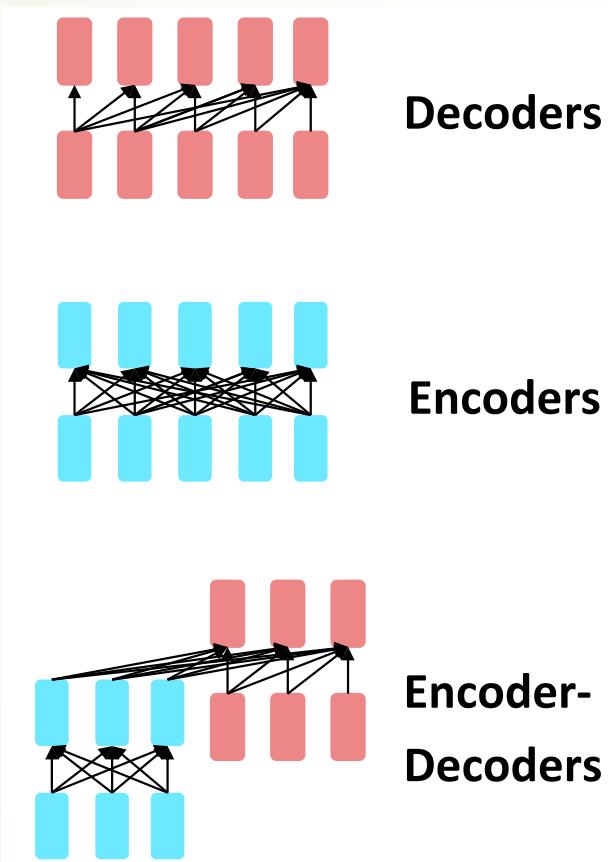
# Deepseek (v-2 and v-3): Multi-head Latent Attention (MLA) and MoE



Deepseek-V2, arXiv:2405.04434  
Deepseek-V3, arXiv:2412.19437

# Pretraining for three types of architectures in Transformers

The transformer architecture influences the type of pretraining:

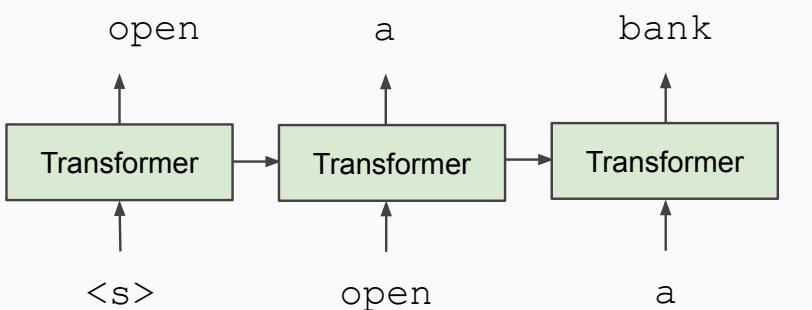


- ▶ Decoders:
  - ▶ **Unidirectional** Language models! What we've seen so far.
  - ▶ Nice to generate from; can't condition on future words: **GPT**
- ▶ Encoders:
  - ▶ Gets **bidirectional** context – can condition on future!
  - ▶ Wait, how do we pretrain them? -- **BERT**
- ▶ Encoder-Decoders:
  - ▶ Good parts of decoders and encoders?
  - ▶ What's the best way to pretrain them? -- **T5**

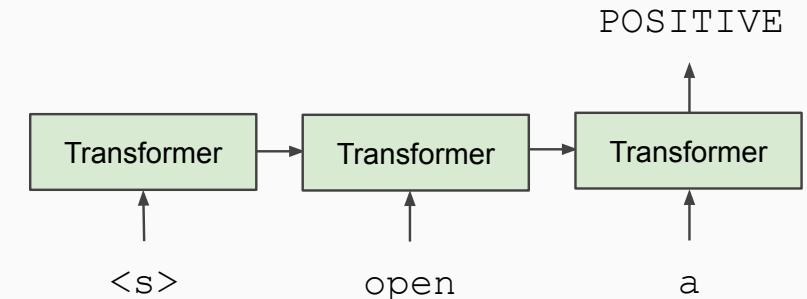
# GPT (Generative Pre-Training): uni-directional transformer decoder

- *Improving Language Understanding by Generative Pre-Training*, OpenAI, 2018

## Train Deep (12-layer) Transformer LM



## Fine-tune on Classification Task



# Pretraining decoders

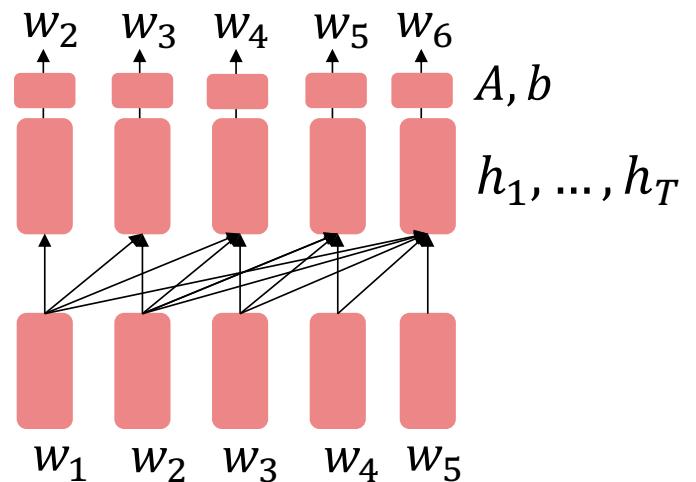
It's natural to pretrain decoders as language models and then use them as generators, finetuning their  $p_\theta(w_t|w_{1:t-1})$ !

This is helpful in tasks **where the output is a sequence** with a vocabulary like that at pretraining time!

- Dialogue (context=dialogue history)
- Summarization (context=document)

$$\begin{aligned} h_1, \dots, h_T &= \text{Decoder}(w_1, \dots, w_T) \\ w_t &\sim Ah_{t-1} + b \end{aligned}$$

Where  $A, b$  were pretrained in the language model!



[Note how the linear layer has been pretrained.]

# Finetuning decoders

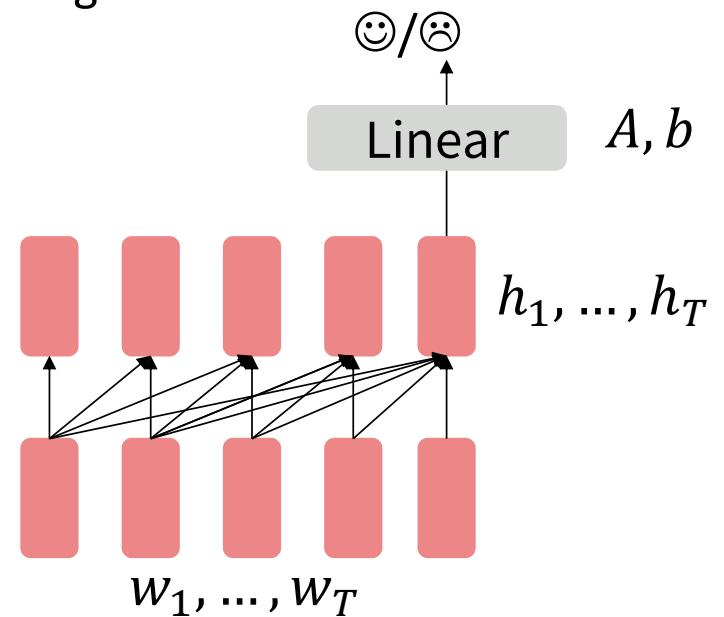
When using language model pretrained decoders, we can ignore that they were trained to model  $p(w_t|w_{1:t-1})$ .

We can finetune them by training a classifier on the last word's hidden state.

$$h_1, \dots, h_T = \text{Decoder}(w_1, \dots, w_T)$$
$$y \sim Ah_T + b$$

Where  $A$  and  $b$  are randomly initialized and specified by the downstream task.

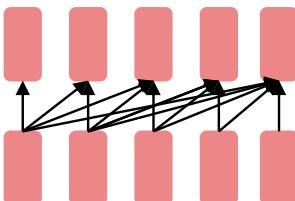
Gradients backpropagate through the whole network.



[Note how the linear layer hasn't been pretrained and must be learned from scratch.]

# GPT (Generative Pre-Trained Transformer): uni-directional transformer-decoder

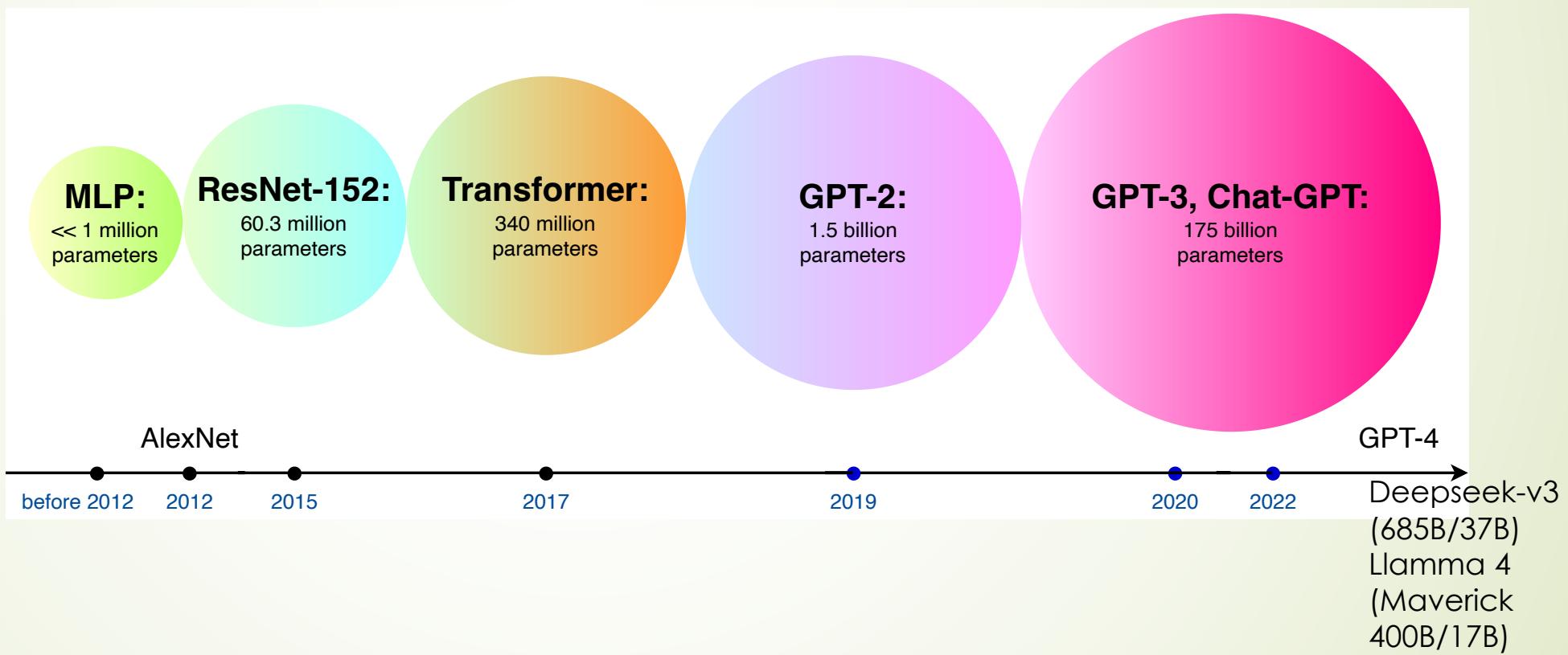
- ▶ 2018's GPT was a big success in pretraining a decoder!
  - Transformer decoder with 12 layers.
  - 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
  - Byte-pair encoding with 40,000 merges
  - Trained on BooksCorpus: over 7000 unique books.
    - Contains long spans of contiguous text, for learning long-distance dependencies.
- GPT-3 (2020) and GPT-3.5 (ChatGPT 2022) has 175 billion parameters or more
- Llama3.1: > 400 billion parameters
- Deepseek-v3: > 600 billion parameters



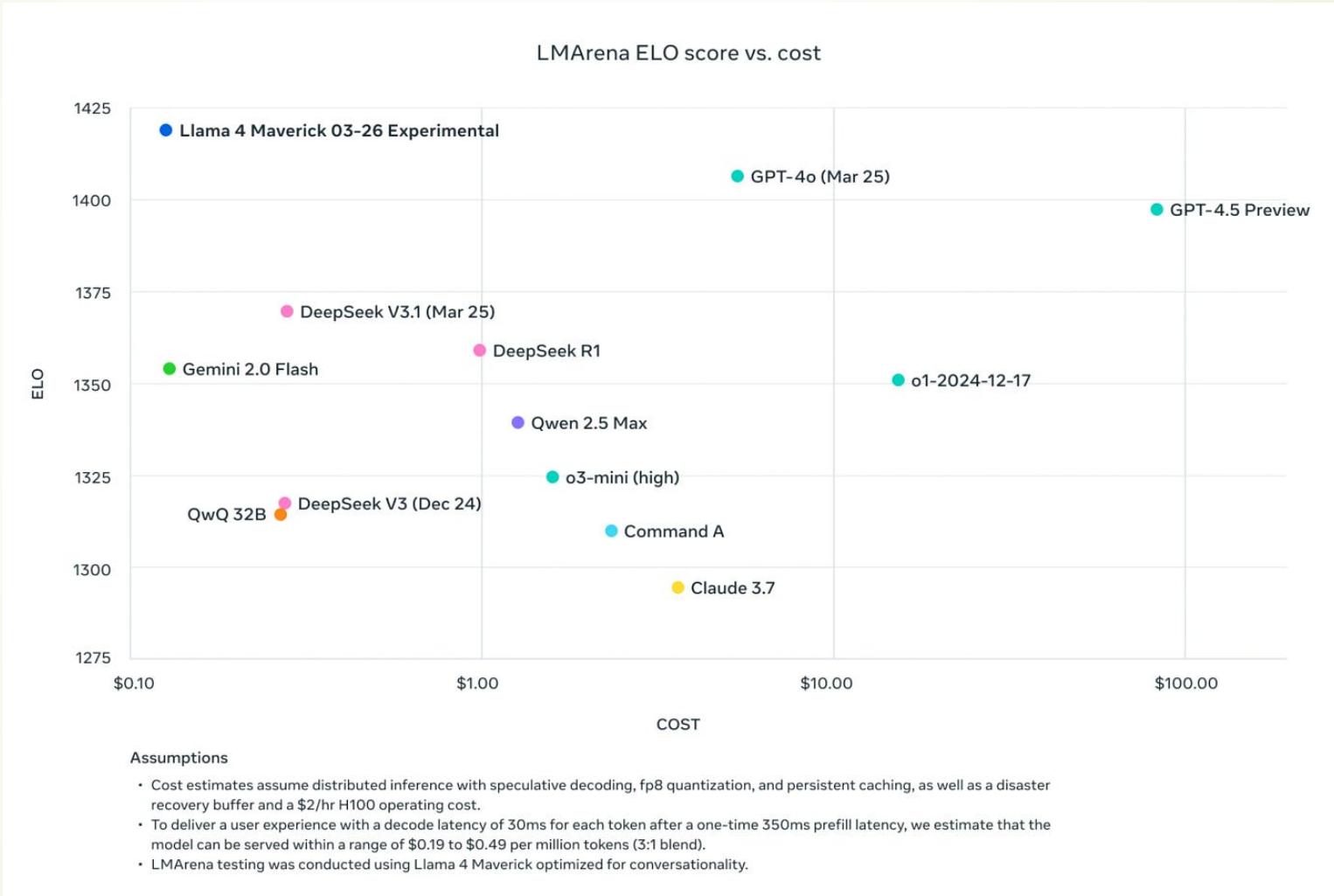
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

# Model size increases



# LMArena score vs. Cost





# Transformers, In-context learning, and very large models

- ▶ So far, we've interacted with pretrained models in two ways:
  - ▶ Sample from the distributions they define (maybe providing a prompt)
  - ▶ Fine-tune them on a task we care about, and take their predictions.
- ▶ Very large language models seem to perform some kind of learning **without gradient steps** simply from examples you provide within their contexts.
- ▶ GPT-3 is the canonical example of this (Brown et al. NeurIPS 2020).
- ▶ Researchers try to interpret **in-context learning** of transformers as **nearest neighbor matching**.

Brown et al. Language models are few-shot learners. NeurIPS 2020.

Bai et al. Transformers as Statisticians. NeurIPS 2023.

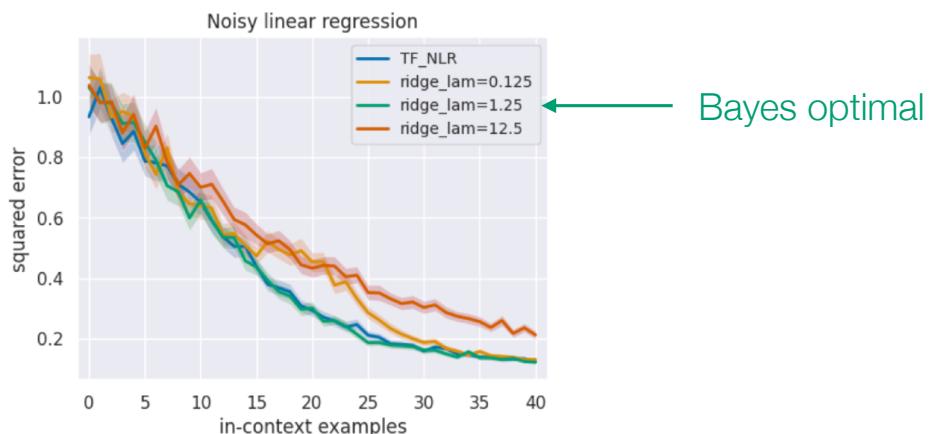
Collins et al. In-Context Learning with Transformers: Softmax Attention Adapts to Function Lipschitzness. NeurIPS 2024.

# Illustration of In-Context Learning

A Task :  $\{(x_i, y_i)\}_{i \in [N]}, \beta \sim \mathcal{N}(0, I_d/d),$

$$x_i \sim \mathcal{N}(0, I_d), \quad y_i = x_i^\top \beta + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma^2)$$

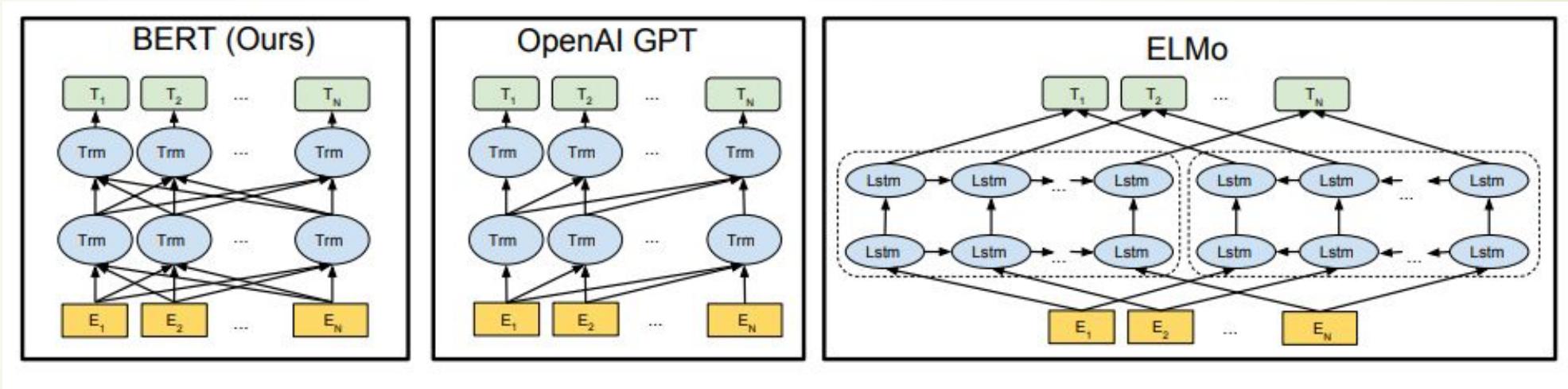
- A dataset of (size  $N$ ) is a meta-d datapoint:  $H = [x_1, y_1, x_2, y_2, \dots, x_N, y_N]$ .
- A meta-dataset (size  $n$ ):  $\{H^{(j)} = [x_1^{(j)}, y_1^{(j)}, x_2^{(j)}, y_2^{(j)}, \dots, x_N^{(j)}, y_N^{(j)}]\}_{j \in [n]}$ .
- Train the GPT2 model using  $\{H^{(j)}\}_{j \in [n]}$  (a smaller version of ChatGPT).
- $d = 5, N = 40, n = 19,200,000$
- Evaluate the test performance of GPT2 on a new independent task.



- Trained GPT2 performs as good as Bayesian predictor!

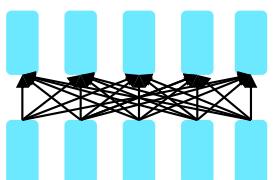
# How about bi-directional transformers (Encoders)?

## – BERT



# BERT: Devlin, Chang, Lee, Toutanova (2018)

- ▶ BERT (**Bidirectional Encoder Representations from Transformers**):
- ▶ Pre-training of Deep Bidirectional Transformers for Language Understanding, which is then fine-tuned for a task
- ▶ Want: truly bidirectional information flow without leakage in a deep model



Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?

# Masked Language Model

- ▶ **Problem:** How the words see each other in bi-directions?
  - ▶ **Solution:** Mask out  $k\%$  of the input words, and then predict the masked words
    - ▶ We always use  $k = 15\%$

- ▶ Too little masking: Too expensive to train
  - ▶ Too much masking: Not enough context

# Masked LM

- ▶ **Problem:** Masked token never seen at fine-tuning
- ▶ **Solution:** 15% of the words to predict, but don't replace with [MASK] 100% of the time. Instead:
  - ▶ 80% of the time, replace with [MASK]
    - ▶ went to the store → went to the [MASK]
  - ▶ 10% of the time, replace random word
    - ▶ went to the store → went to the running
  - ▶ 10% of the time, keep same
    - ▶ went to the store → went to the store

# Next Sentence Prediction

- To learn *relationships* between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

**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

# BERT sentence pair encoding

- ▶ Token embeddings are word pieces (30k)
- ▶ Learned segmented embedding represents each sentence
- ▶ Positional embedding is as for other Transformer architectures

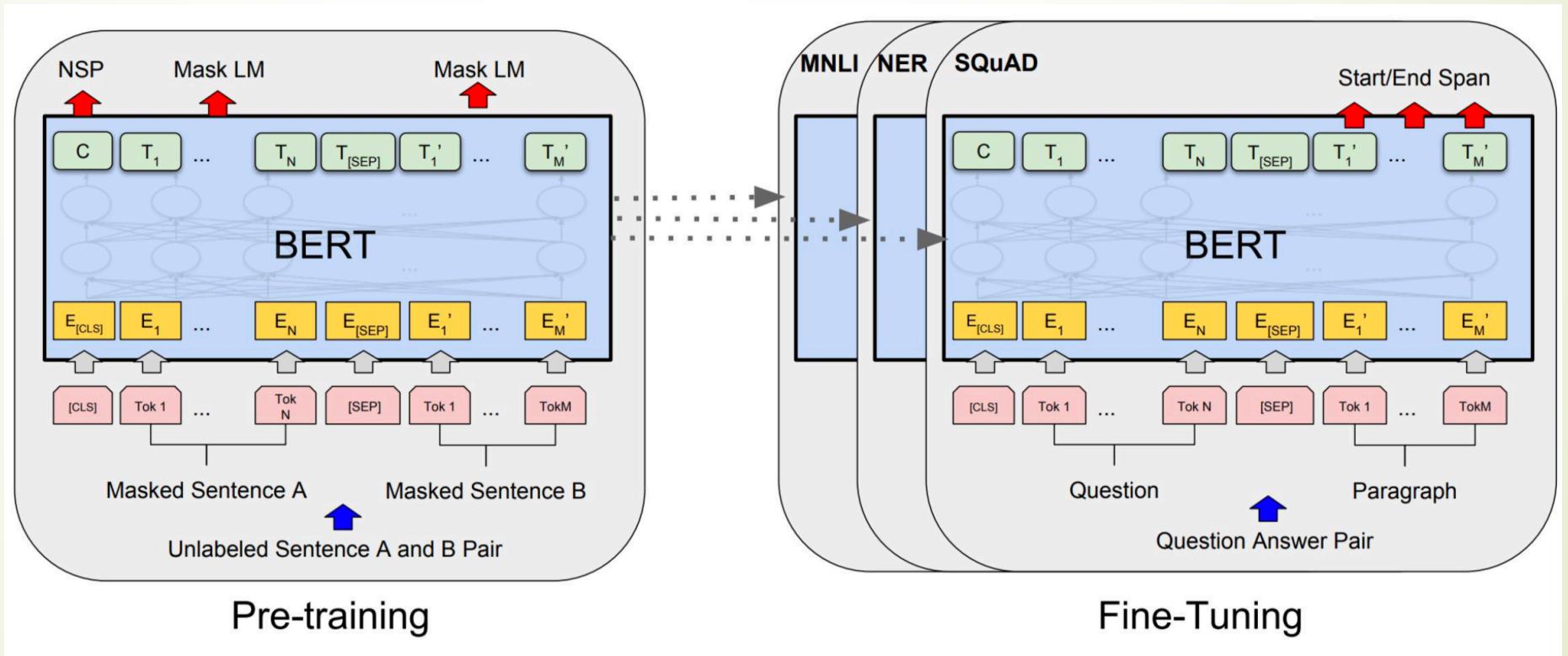
Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	# #ing	[SEP]
Token Embeddings	$E_{[CLS]}$	$E_{my}$	$E_{dog}$	$E_{is}$	$E_{cute}$	$E_{[SEP]}$	$E_{he}$	$E_{likes}$	$E_{play}$	$E_{##ing}$	$E_{[SEP]}$
Segment Embeddings	$E_A$	$E_A$	$E_A$	$E_A$	$E_A$	$E_A$	$E_B$	$E_B$	$E_B$	$E_B$	$E_B$
Position Embeddings	$E_0$	$E_1$	$E_2$	$E_3$	$E_4$	$E_5$	$E_6$	$E_7$	$E_8$	$E_9$	$E_{10}$

# PreTraining

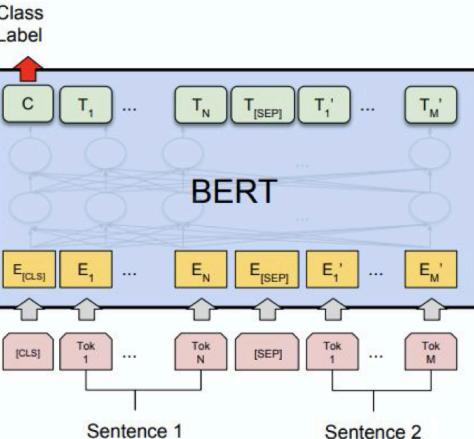
- ▶ 2 model released:
  - ▶ BERT-Base: 12-layer, 768-hidden, 12-head, 110 million params.
  - ▶ BERT-Large: 24-layer, 1024-hidden, 16-head, 340 million params.
- ▶ Training Data:
  - ▶ BookCorpus (800M words)
  - ▶ English Wikipedia (2.5B words)
- ▶ Batch Size: 131,072 words
  - ▶ (1024 sequences \* 128 length or 256 sequences \* 512 length)
- ▶ Training Time: 1M steps (~40 epochs)
- ▶ Optimizer: AdamW, 1e-4 learning rate, linear decay
- ▶ Trained on 4x4 or 8x8 TPU slice for 4 days
- ▶ Pretraining is expensive and impractical on a single GPU; Finetuning is practical and common on a single GPU

# BERT model fine tuning

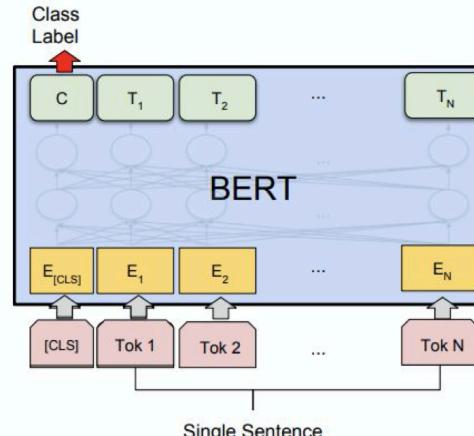
- ▶ Simply learn a classifier built on the top layer for each task that you fine tune for



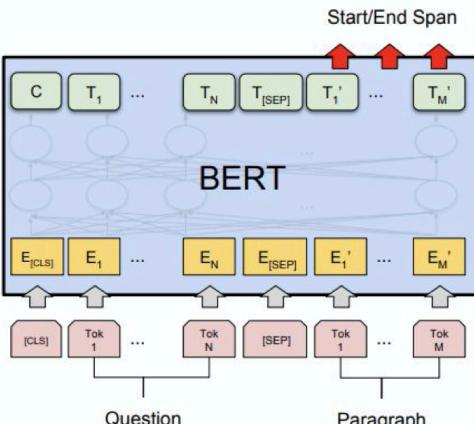
# BERT model fine tuning



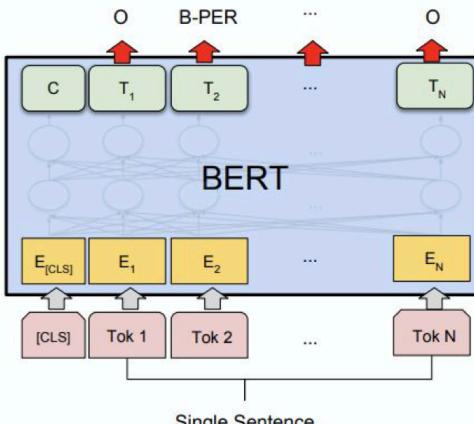
(a) Sentence Pair Classification Tasks:  
MNLI, QQP, QNLI, STS-B, MRPC,  
RTE, SWAG



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

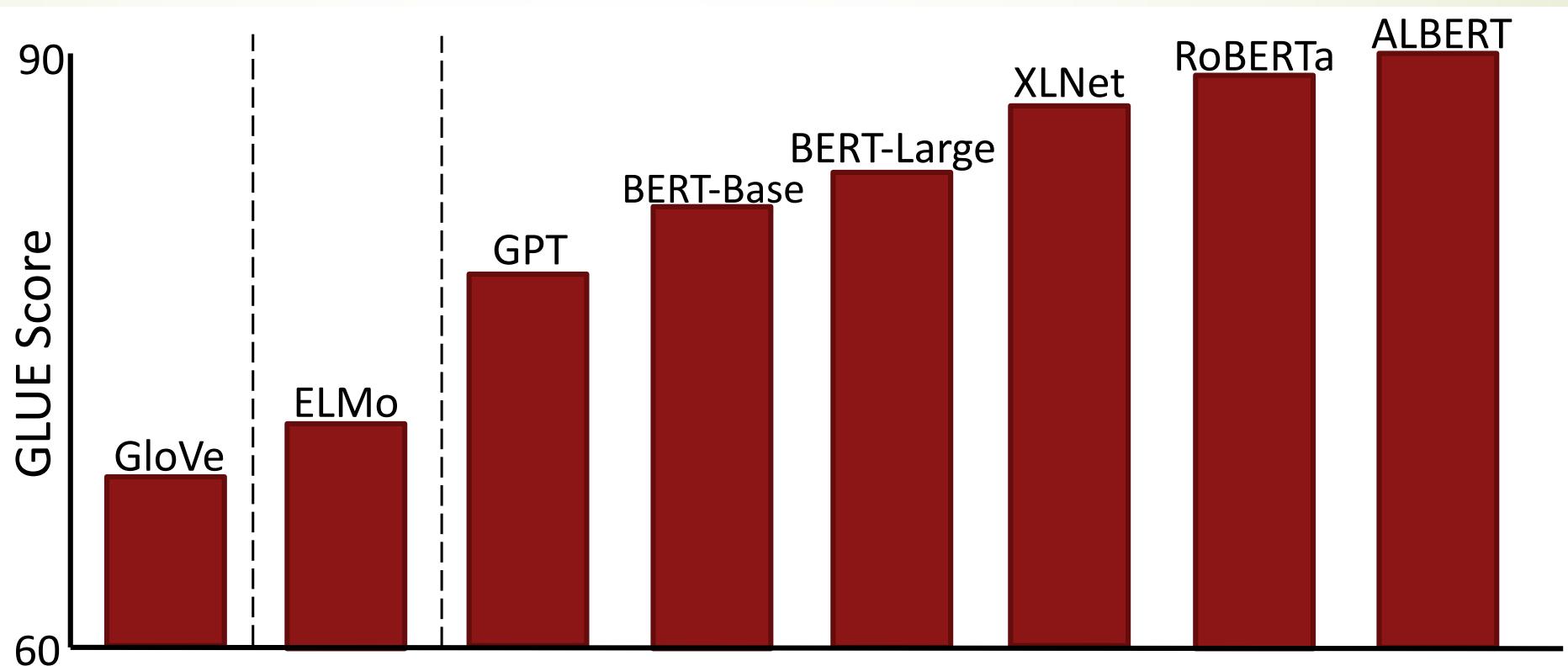


(c) Question Answering Tasks:  
SQuAD v1.1



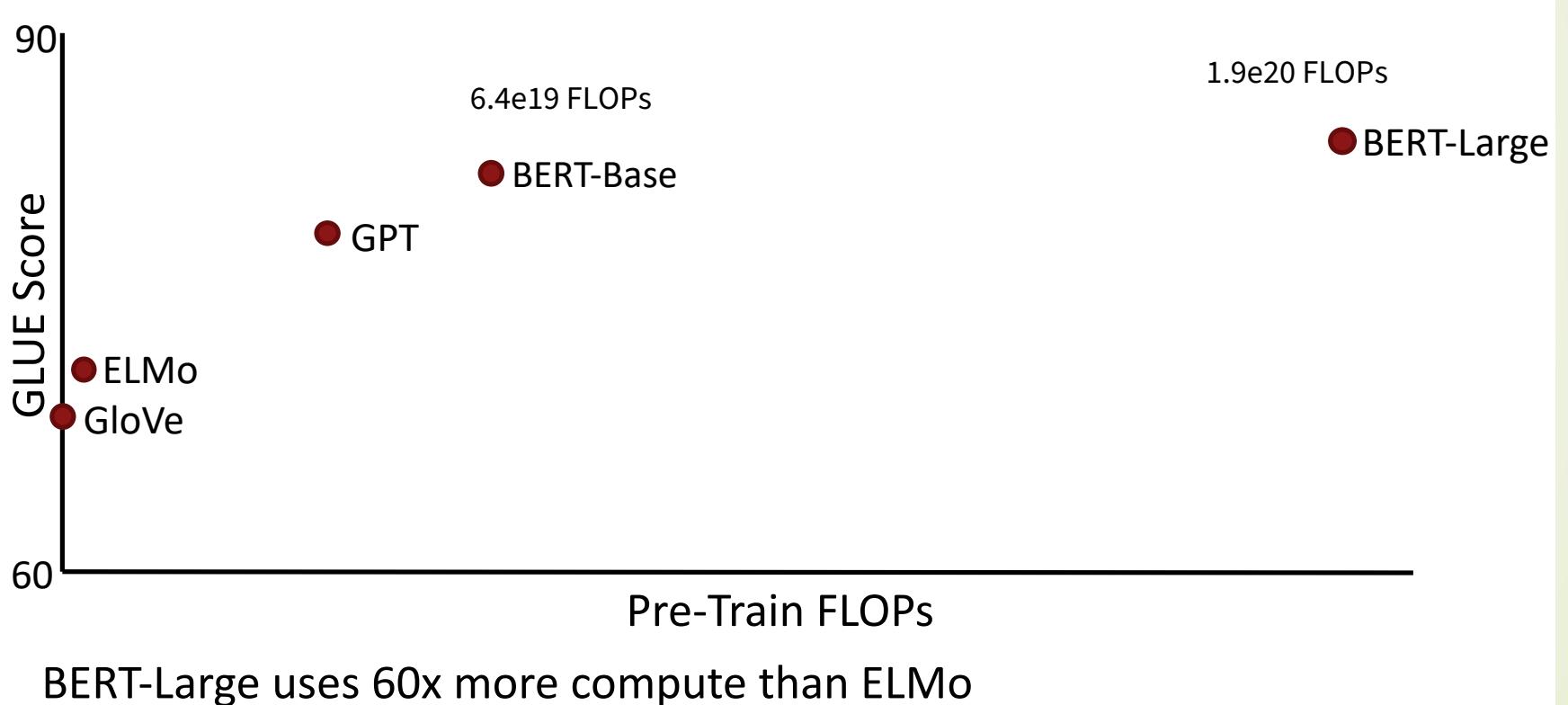
(d) Single Sentence Tagging Tasks:  
CoNLL-2003 NER

# Rapid Progress for Pre-training (GLUE Benchmark)

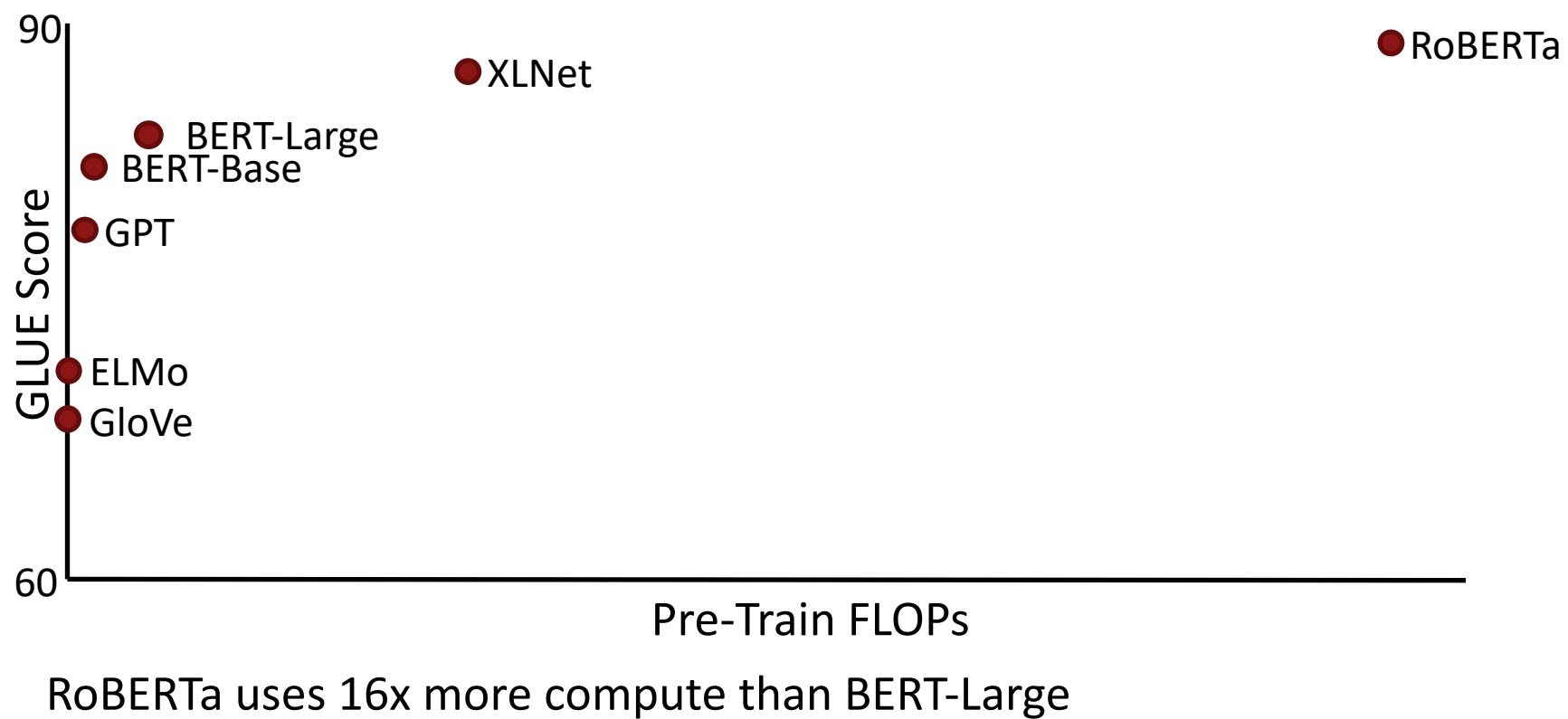


Over 3x reduction in error in 2 years, “superhuman” performance

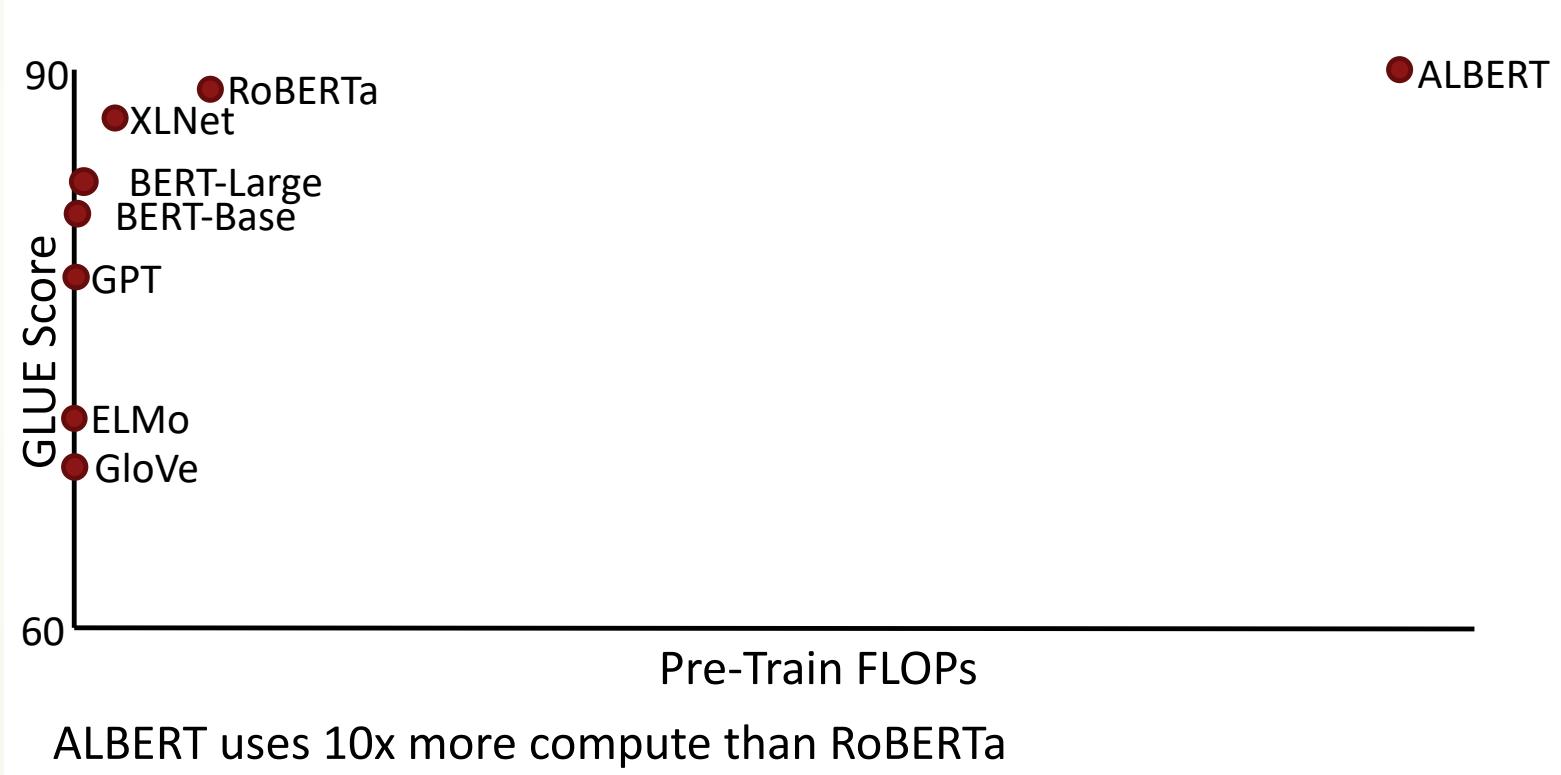
But let's change the x-axis to computational cost...



But let's change the x-axis to computational cost...

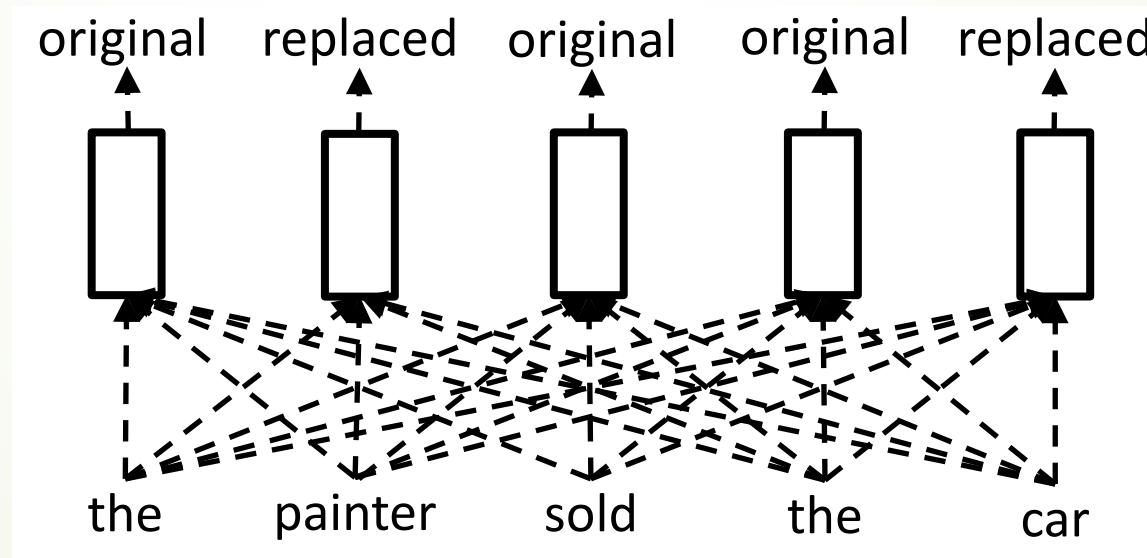


# More compute, more better?

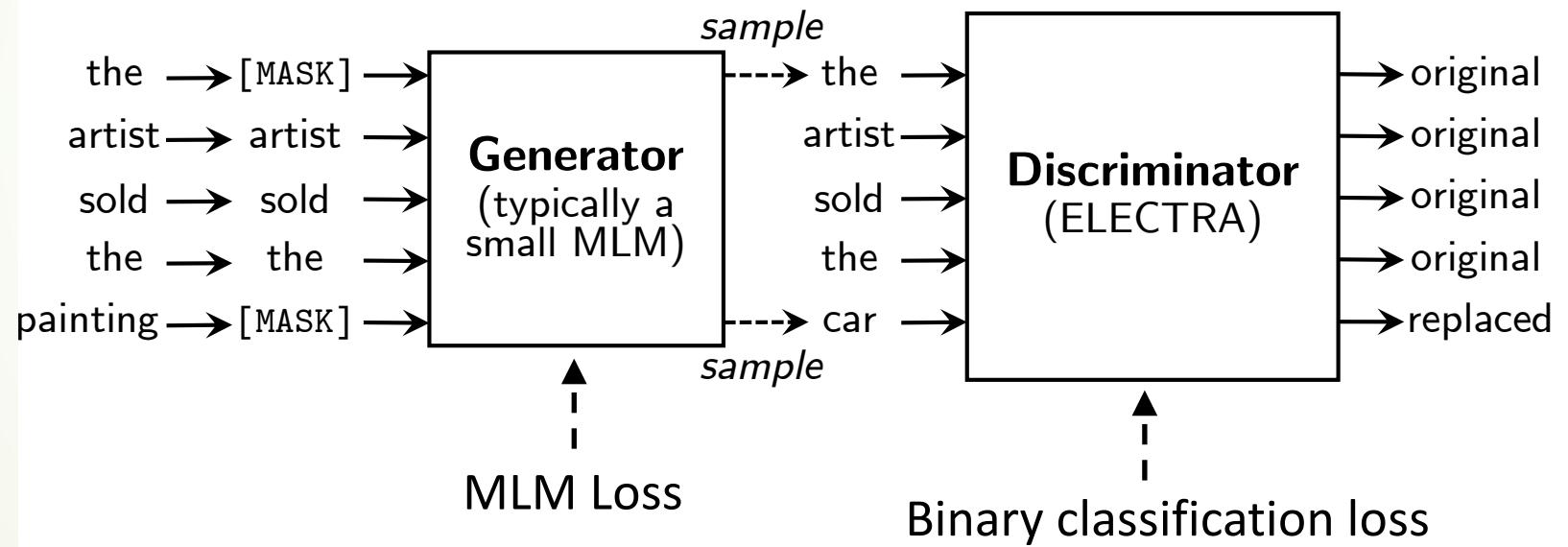


# ELECTRA: “Efficiently Learning an Encoder to Classify Token Replacements Accurately”

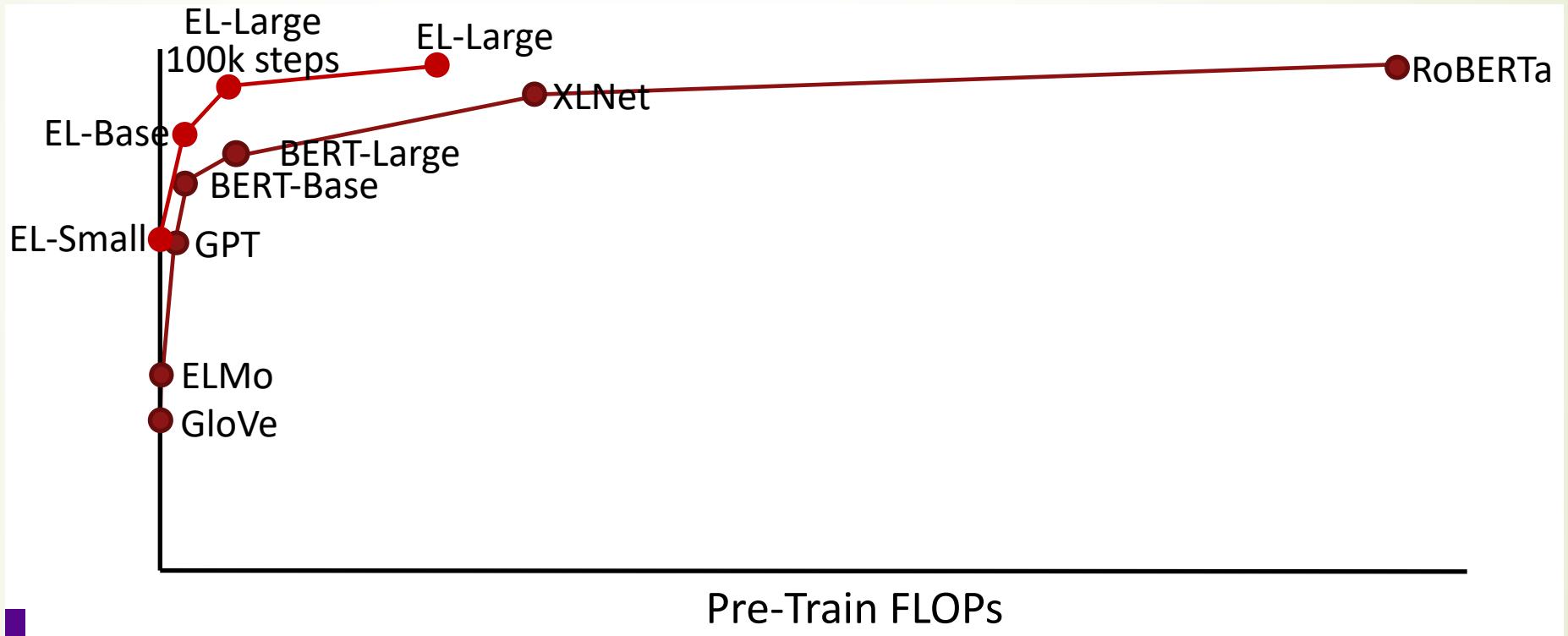
- ▶ Clark, Luong, Le, and Manning, ICLR 2020.  
<https://openreview.net/pdf?id=r1xMH1BtvB>
- ▶ Bidirectional model but learn from all tokens



# Generating Replacements

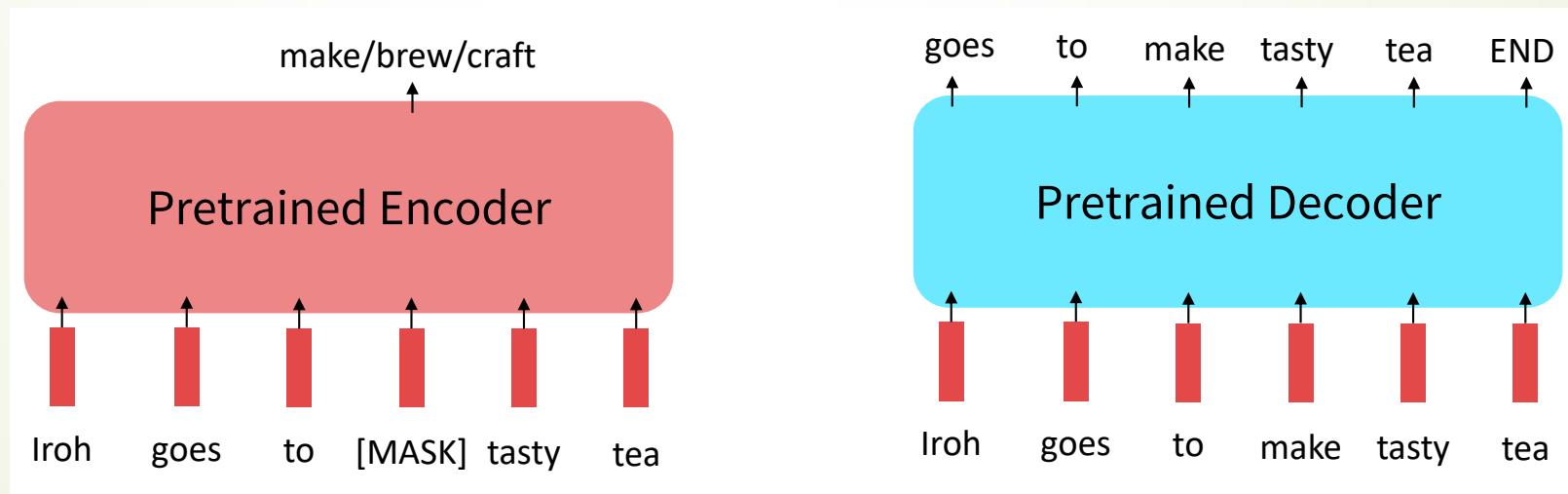


# Results: GLUE Score vs Compute



# Limitations of Pretrained Encoders vs. Decoders

- ▶ BERT and other pretrained **encoders** are good for classifications, but don't naturally lead to nice autoregressive (1-word-at-a-time) generative methods.
- ▶ **Decoders** like GPT are good at generating sequences in autoregressive way.

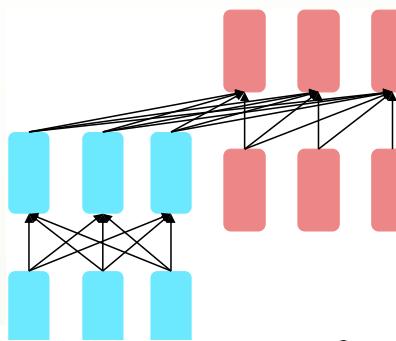


# Pretraining encoders-decoders: T5

- ▶ Pretraining encoder-decoders: what pretraining objective to use?
- ▶ What Raffel et al., 2018 found to work best was **span corruption: T5**.
- ▶ Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!
- ▶ The largest T5 model had 11 billion parameters.

Original text

Thank you for inviting me to your party last week.



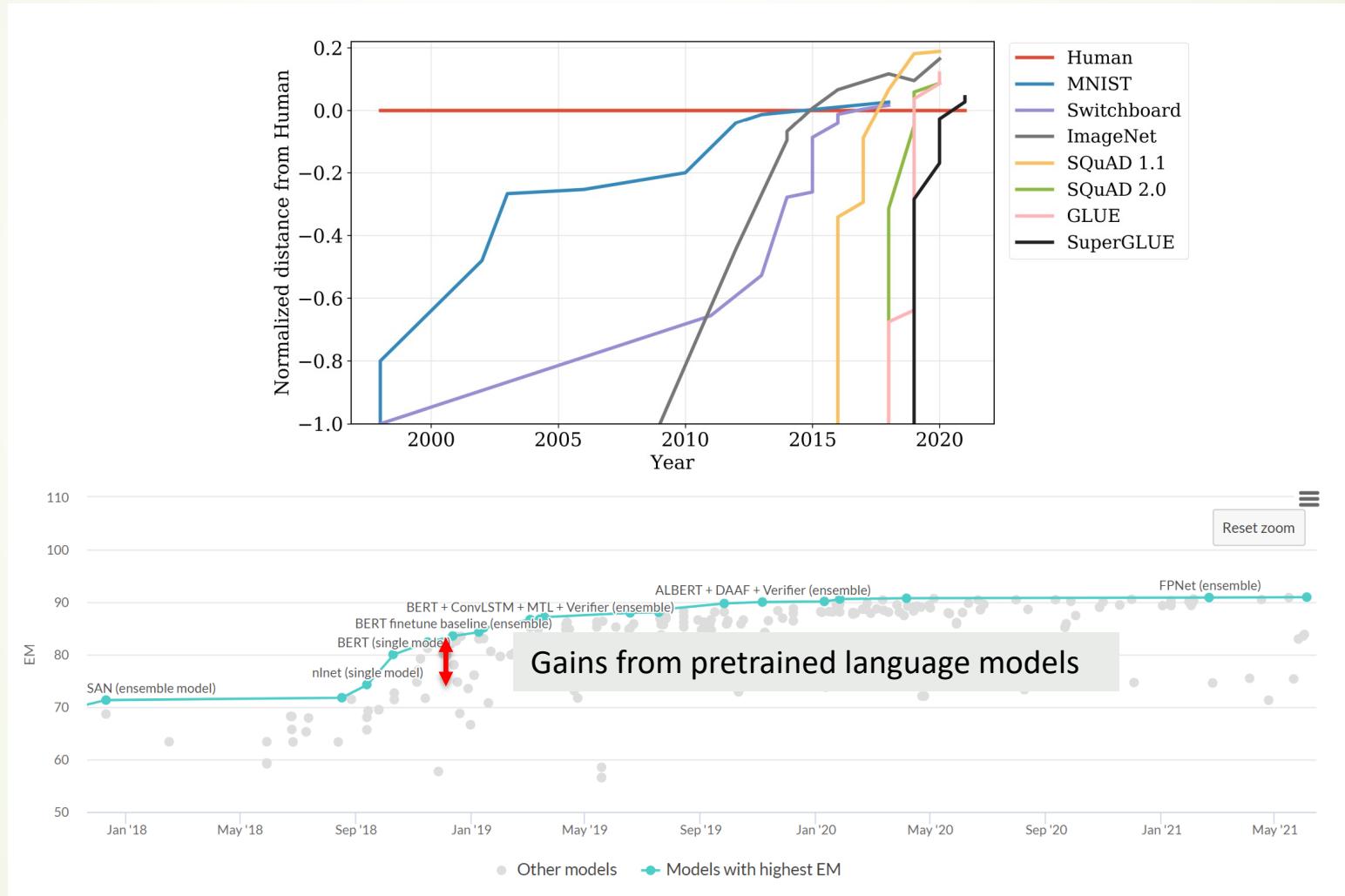
Targets

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

Inputs

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

# Pretraining revolution

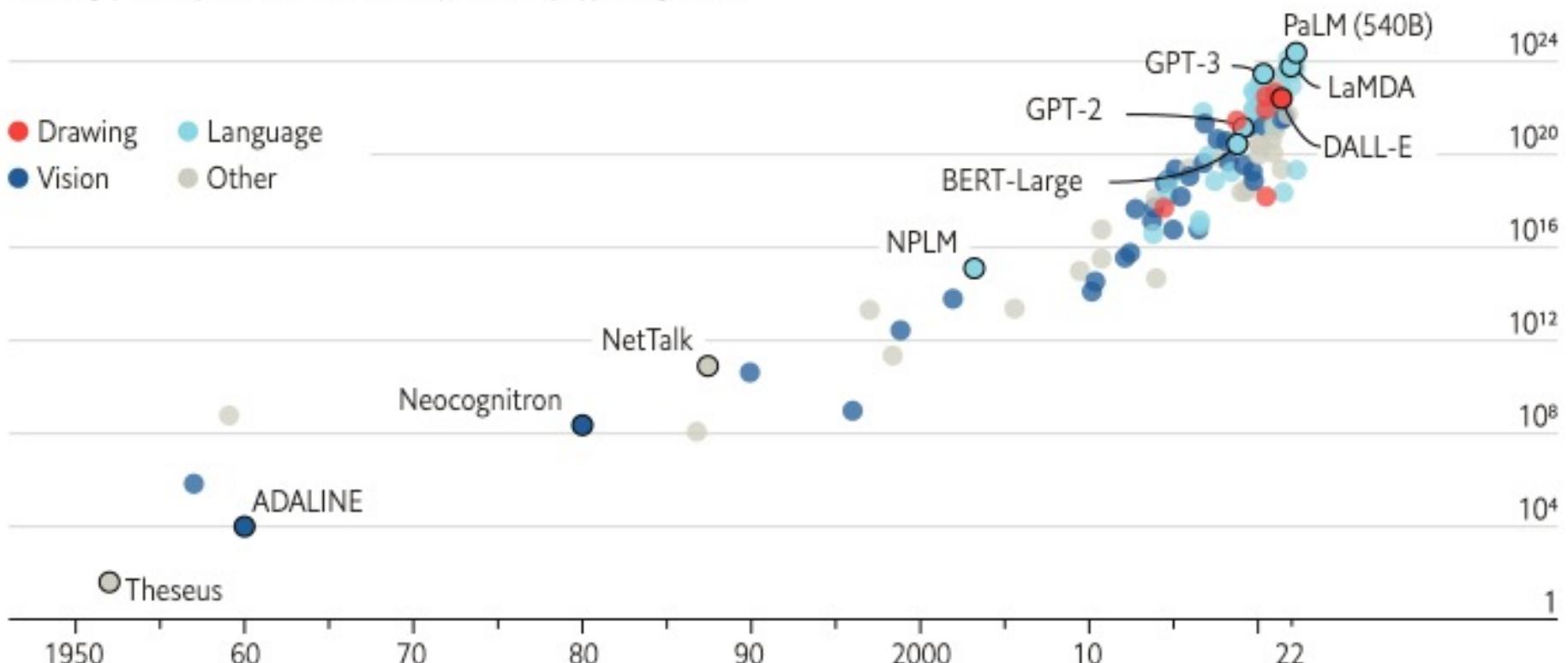


# Exponential increase of computing

## The blessings of scale

AI training runs, estimated computing resources used

Floating-point operations, selected systems, by type, log scale

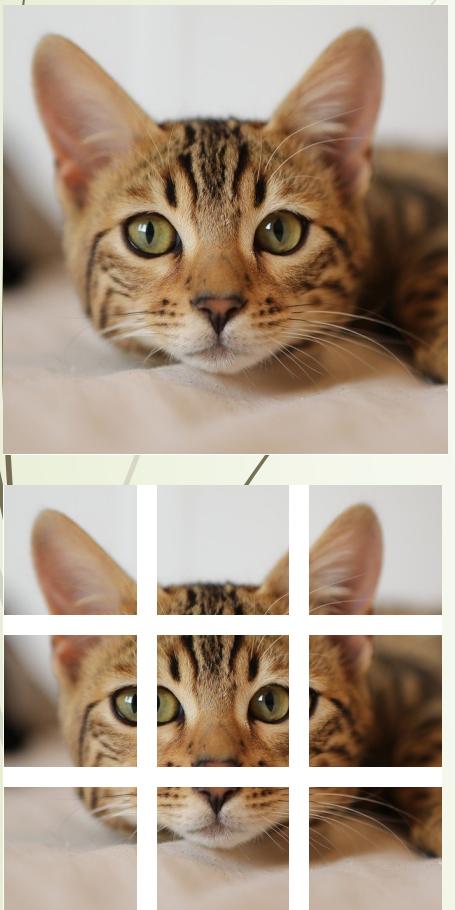


Sources: "Compute trends across three eras of machine learning", by J. Sevilla et al., arXiv, 2022; Our World in Data



# Vision Transformer

Transformer for images?



# Vision Transformer (ViT)

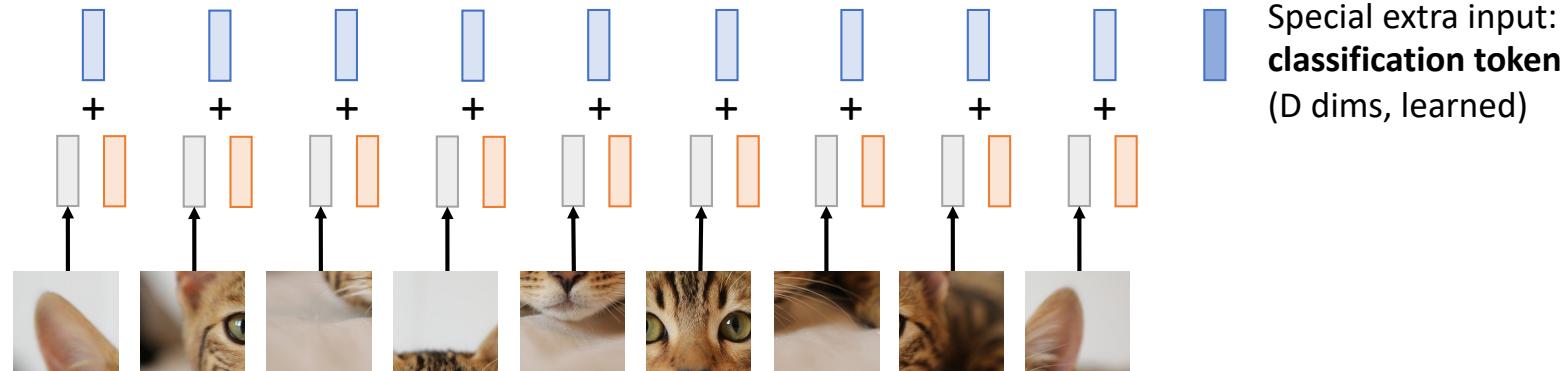
Computer vision model  
with no convolutions!

Output vectors



Exact same as  
NLP Transformer!

Add positional  
embedding: learned D-  
dim vector per position



Linear projection to  
D-dimensional vector

N input patches, each  
of shape 3x16x16

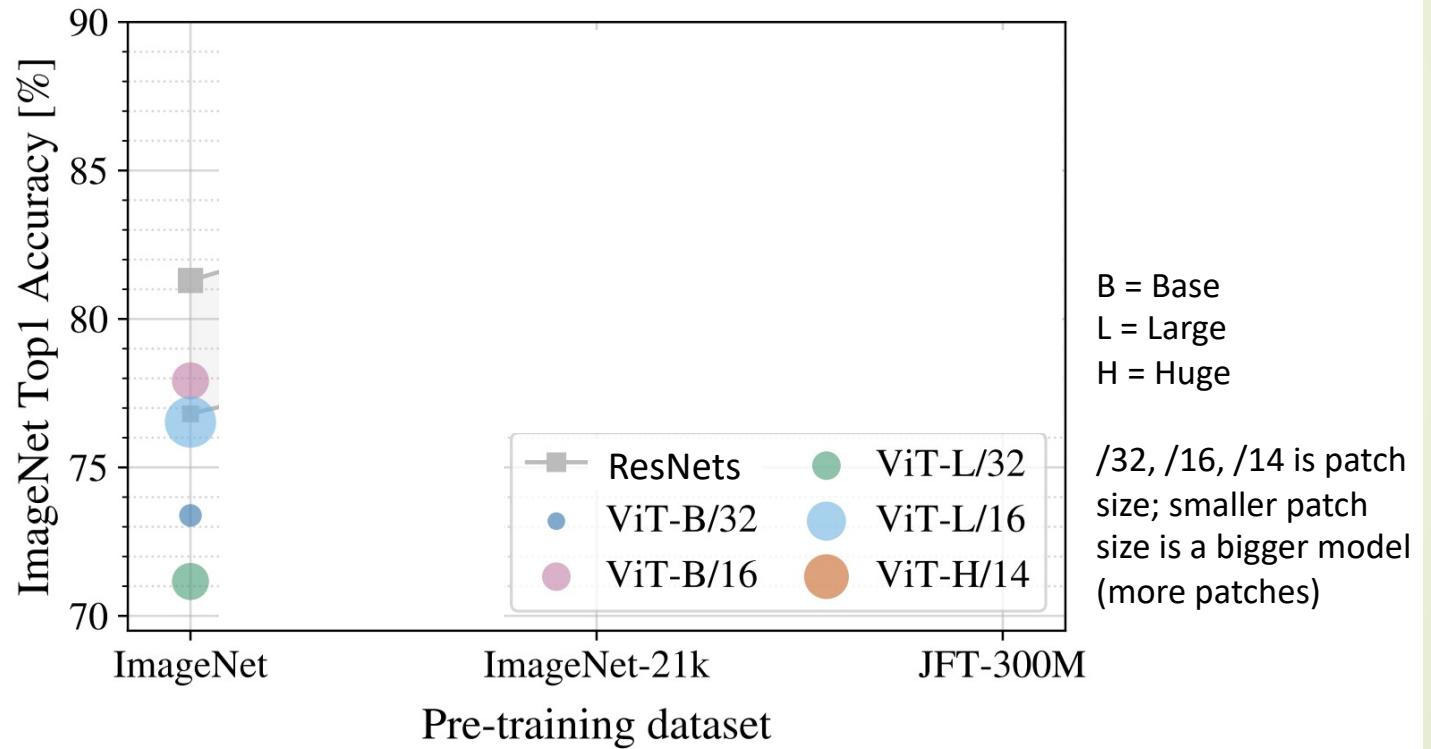
Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

[Cat image](#) is free for commercial  
use under a [Pixabay license](#)

## Vision Transformer (ViT) vs ResNets

Recall: ImageNet dataset has 1k categories, 1.2M images

When trained on ImageNet, ViT models perform worse than ResNets

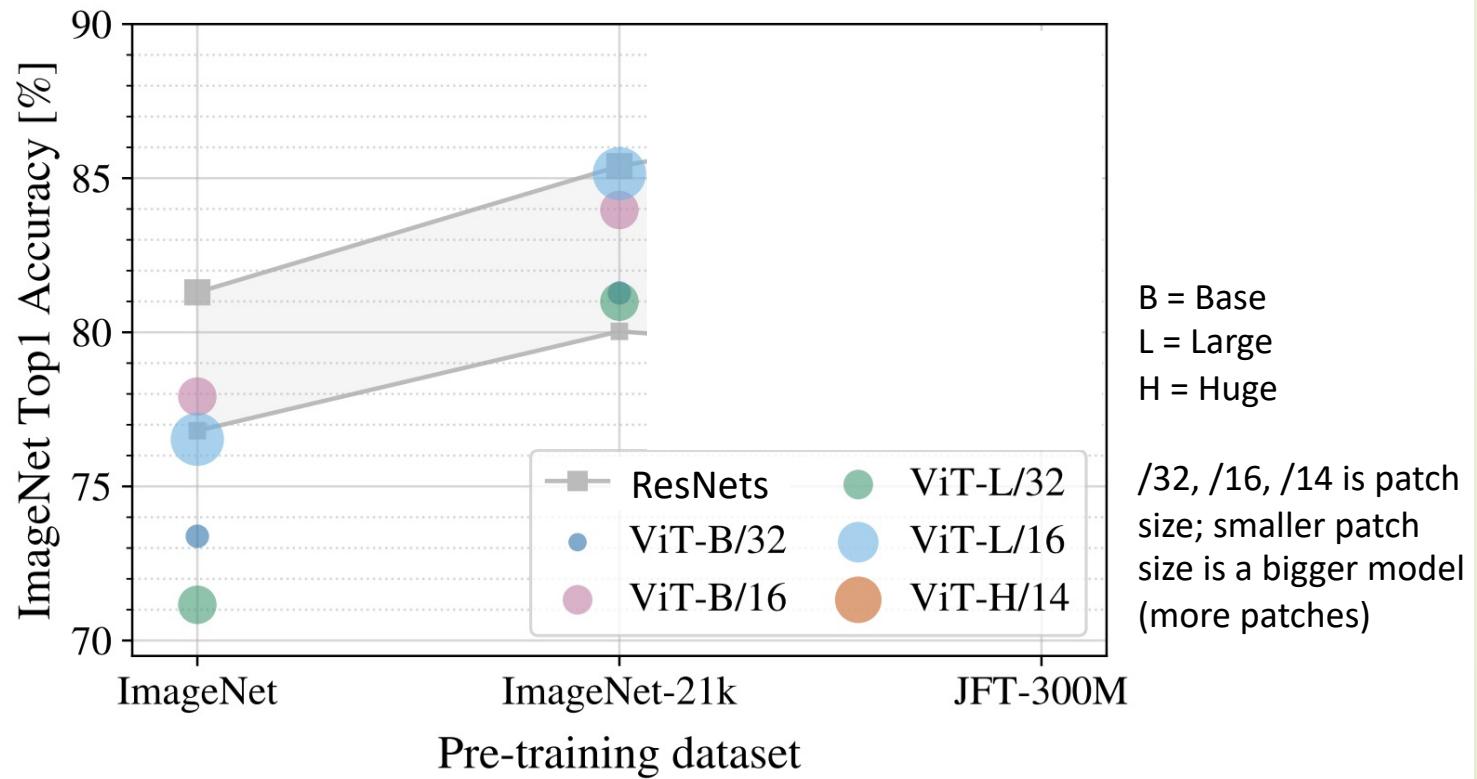


Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

# Vision Transformer (ViT) vs ResNets

ImageNet-21k has 14M images with 21k categories

If you pretrain on ImageNet-21k and fine-tune on ImageNet, ViT does better: big ViTs match big ResNets

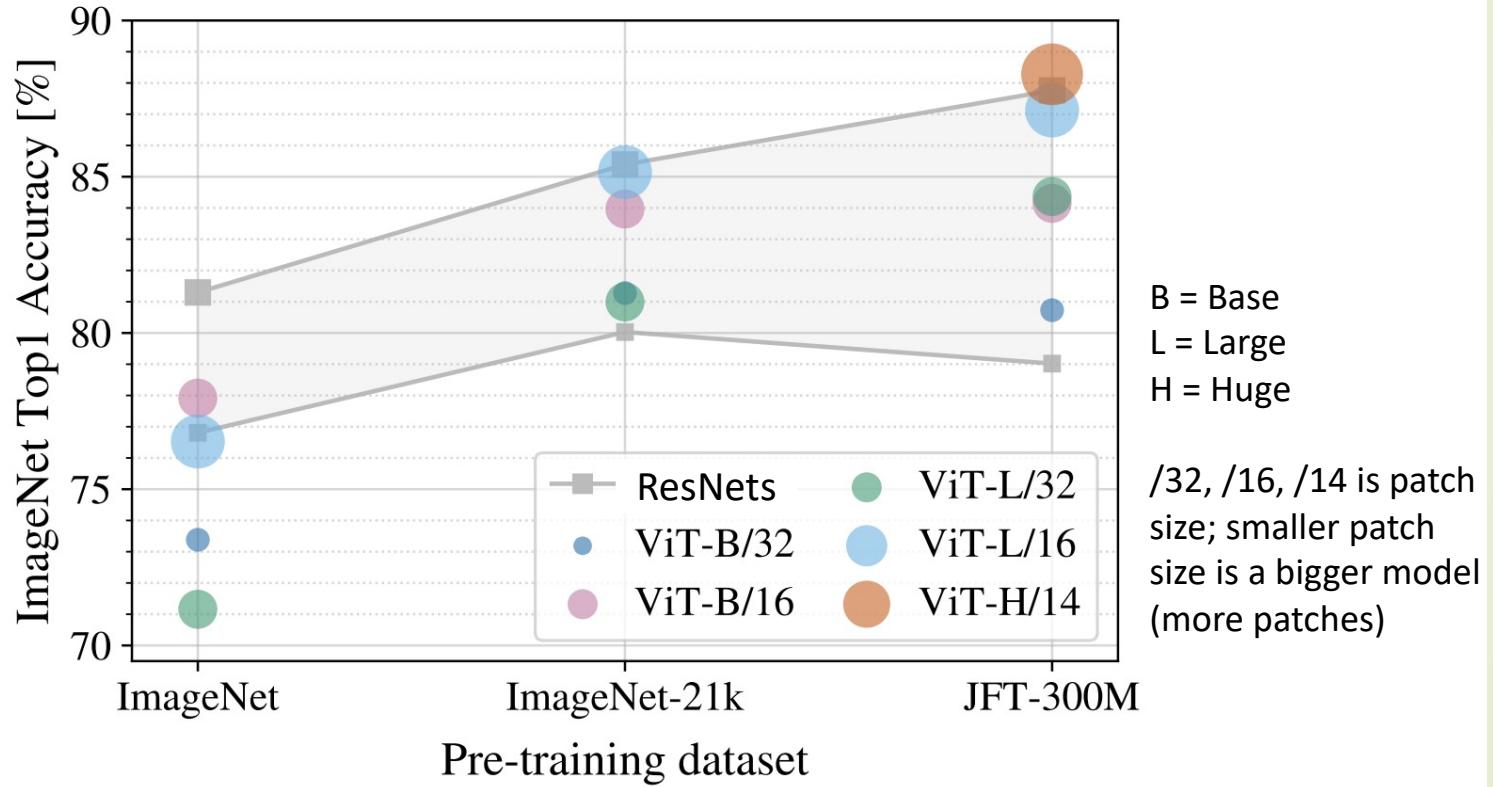


Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

# Vision Transformer (ViT) vs ResNets

JFT-300M is an internal Google dataset with 300M labeled images

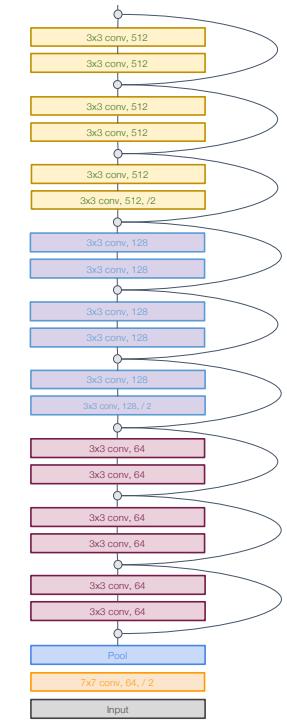
If you pretrain on JFT and finetune on ImageNet, large ViTs outperform large ResNets



Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

# ViT vs CNN

Stage 3:  
256 x 14 x 14

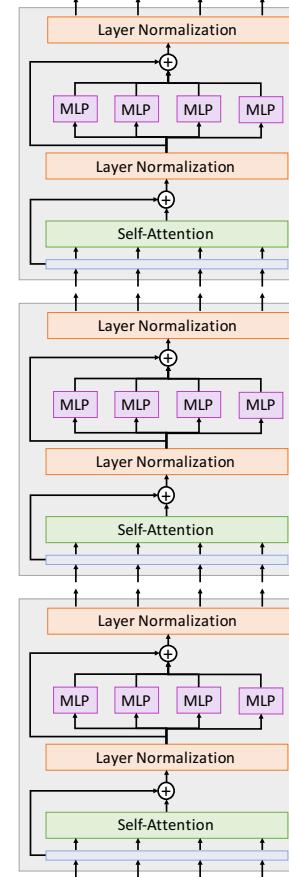


In most CNNs (including ResNets), **decrease** resolution and **increase** channels as you go deeper in the network (Hierarchical architecture)

Useful since objects in images can occur at various scales

In a ViT, all blocks have same resolution and number of channels (Isotropic architecture)

Can we build a **hierarchical** ViT model?

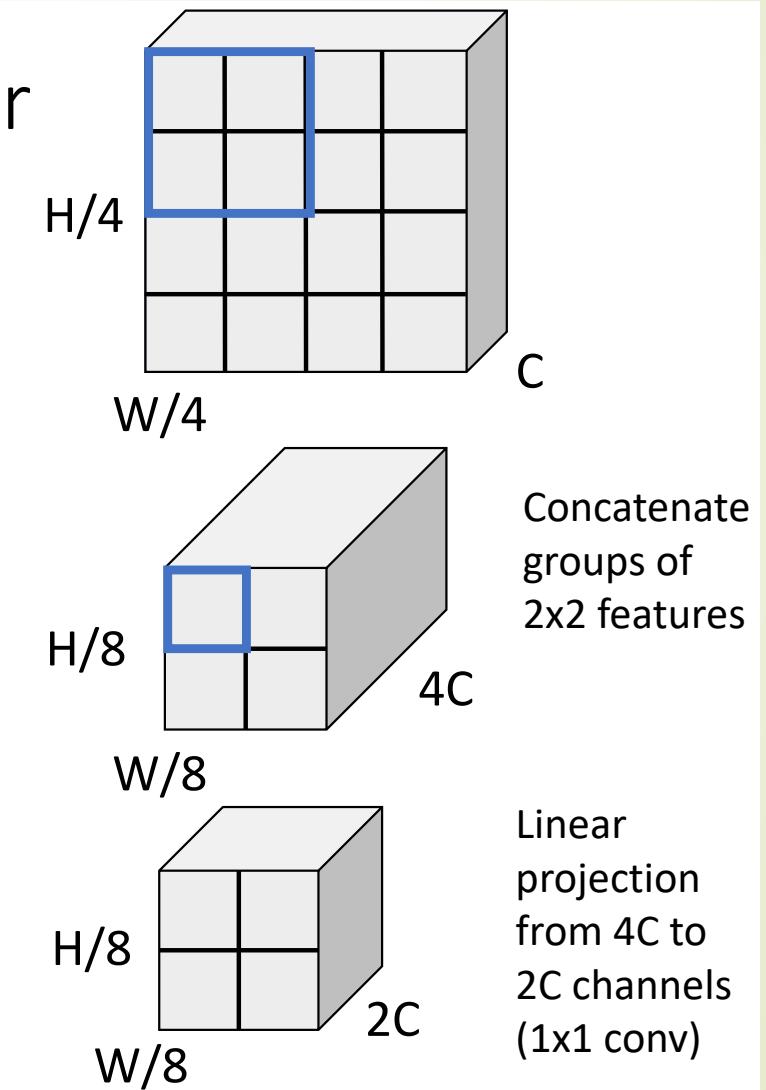
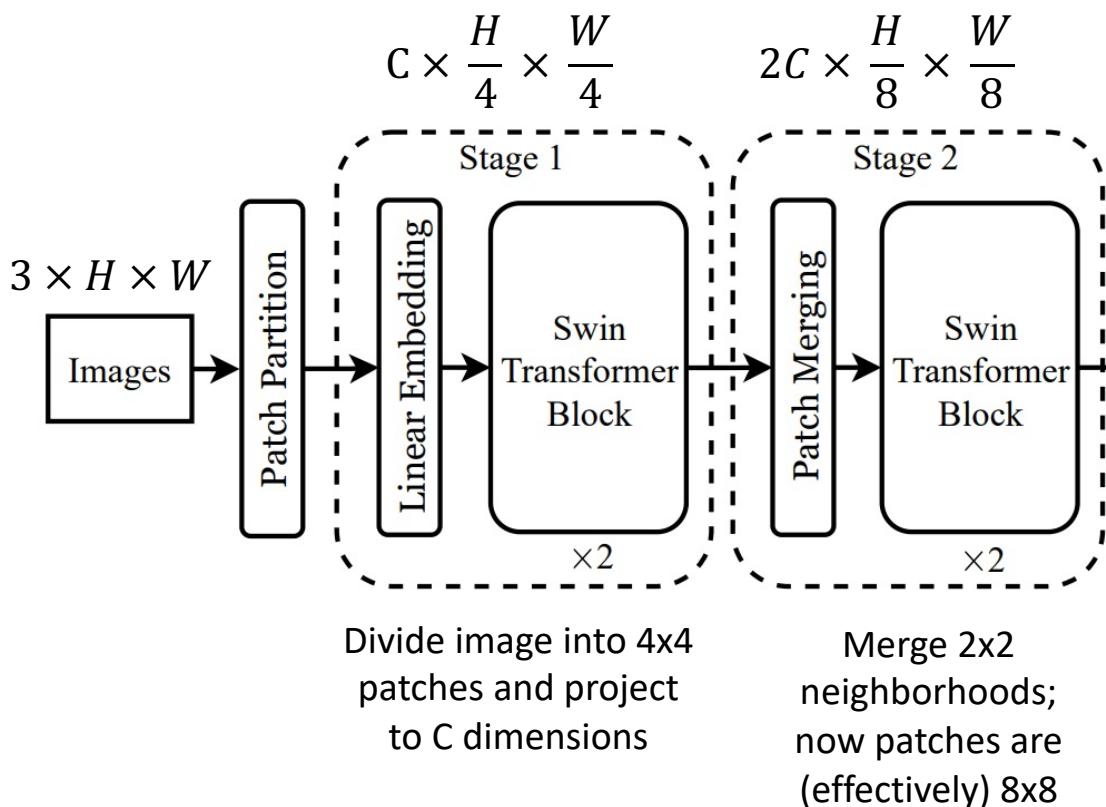


3<sup>rd</sup> block:  
768 x 14 x 14

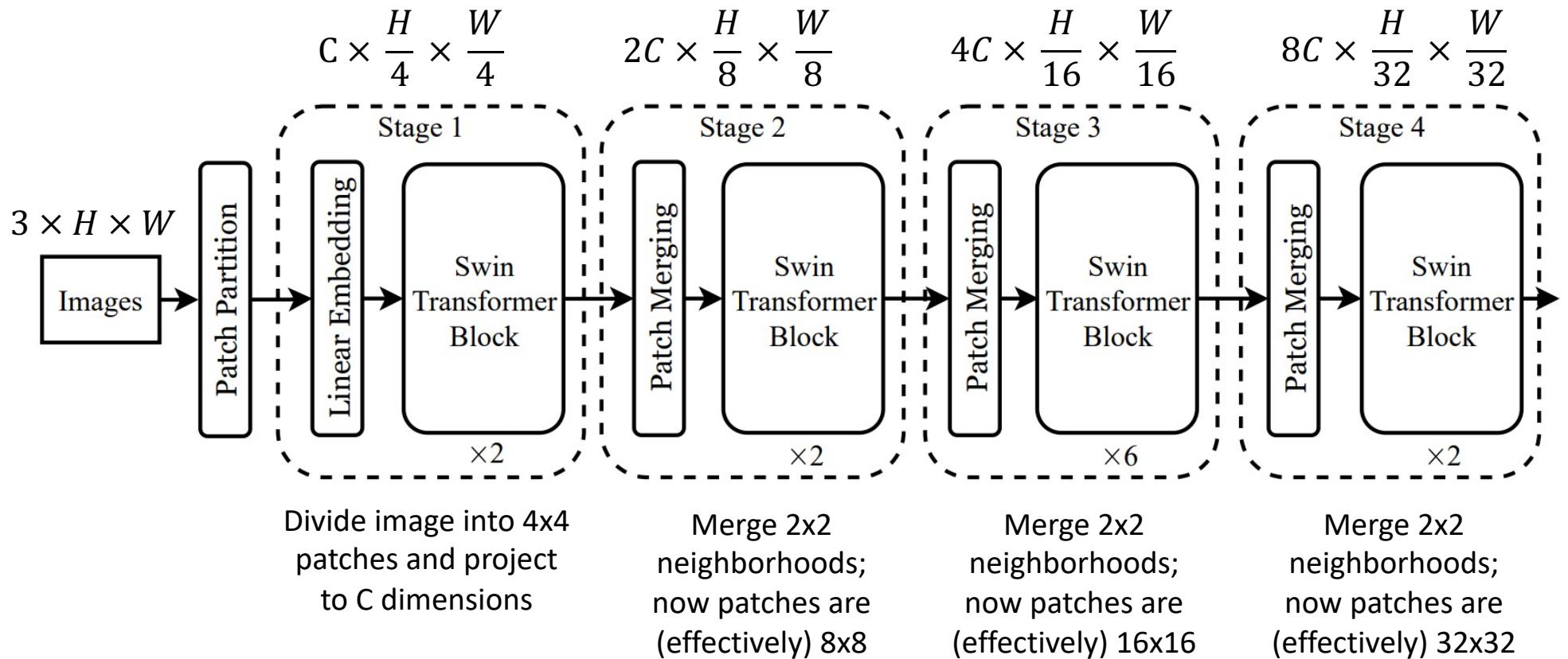
2<sup>nd</sup> block:  
768 x 14 x 14

1<sup>st</sup> block:  
768 x 14 x 14

# Hierarchical ViT: Swin Transformer



# Hierarchical ViT: Swin Transformer



# Swin Transformer: Window Attention



With  $H \times W$  grid of **tokens**, each attention matrix is  $H^2W^2$  – **quadratic** in image size

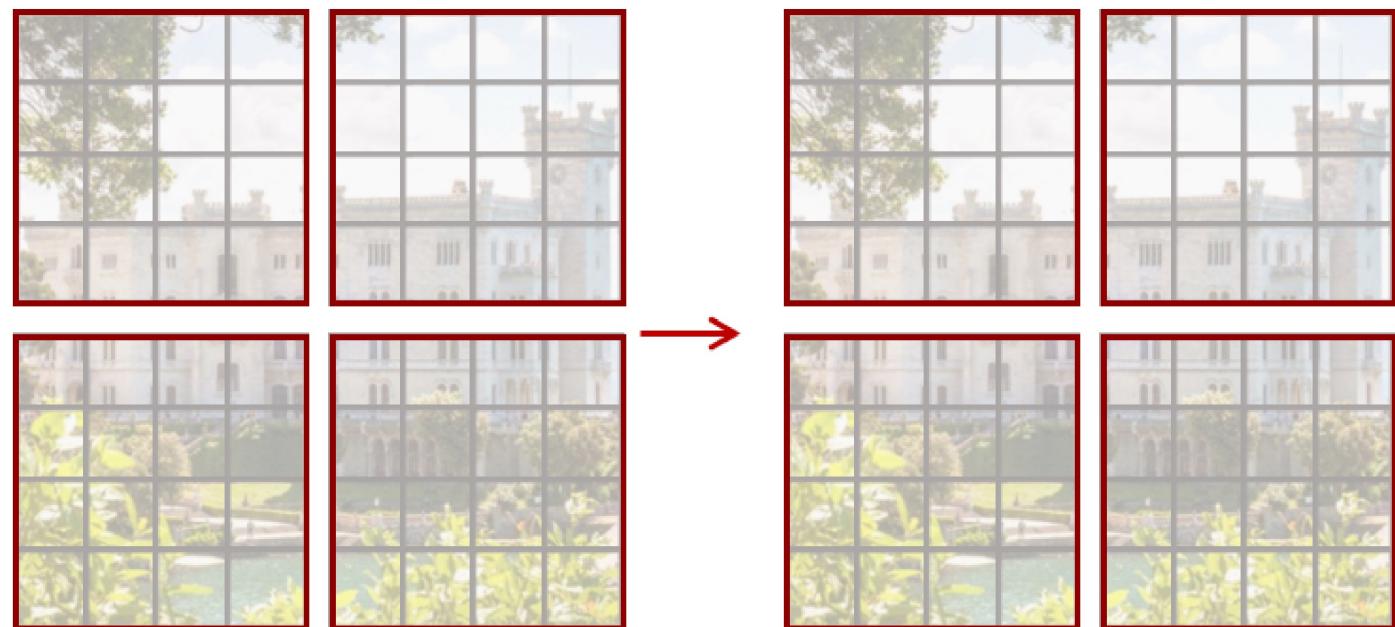
Rather than allowing each **token** to attend to all other tokens, instead divide into **windows** of  $M \times M$  tokens (here  $M=4$ ); only compute attention within each window

Total size of all attention matrices is now:  
 $M^4(H/M)(W/M) = M^2HW$

**Linear** in image size for fixed  $M$ !  
Swin uses  $M=7$  throughout the network

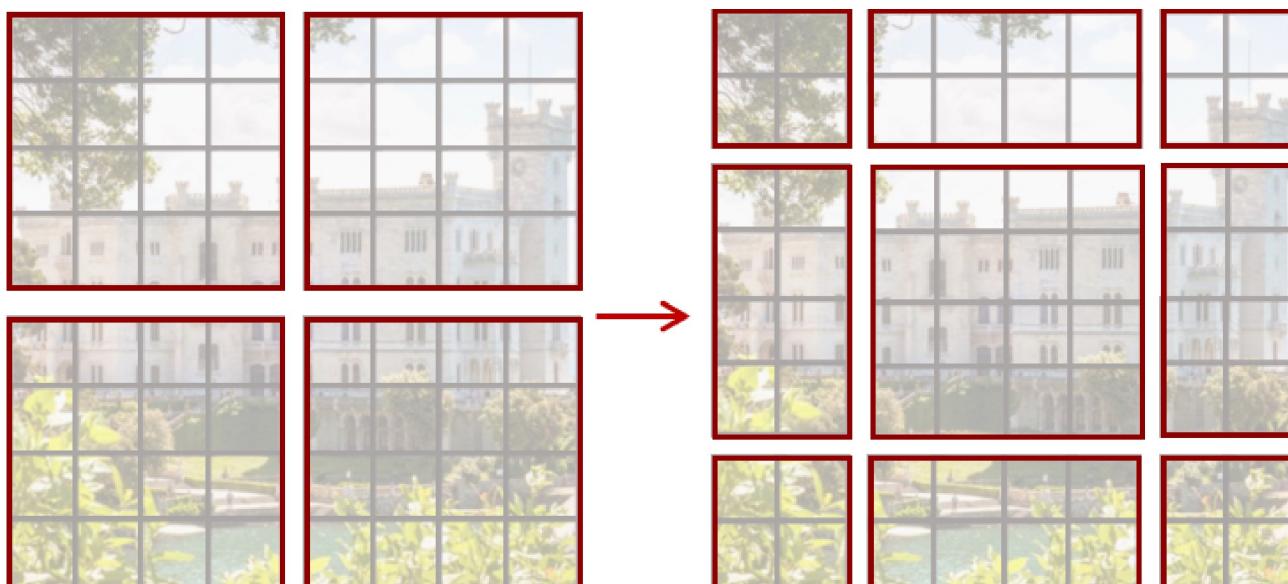
## Swin Transformer: Window Attention

**Problem:** tokens only interact with other tokens within the same window; no communication across windows



# Swin Transformer: Shifted Window Attention

**Solution:** Alternate between normal windows and shifted windows in successive Transformer blocks



Block L: Normal windows

Block L+1: Shifted Windows

Detail: Relative Positional Bias

ViT adds positional embedding to input tokens, encodes *absolute position* of each token in the image

Swin does not use positional embeddings, instead encodes *relative position* between patches when computing attention:

Attention with relative bias:

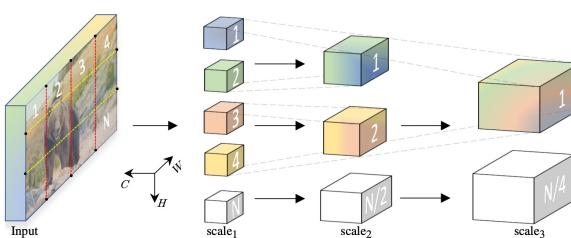
$$A = \text{Softmax} \left( \frac{QK^T}{\sqrt{D}} + B \right) V$$

$Q, K, V: M^2 \times D$  (Query, Key, Value)

$B: M^2 \times M^2$  (learned biases)

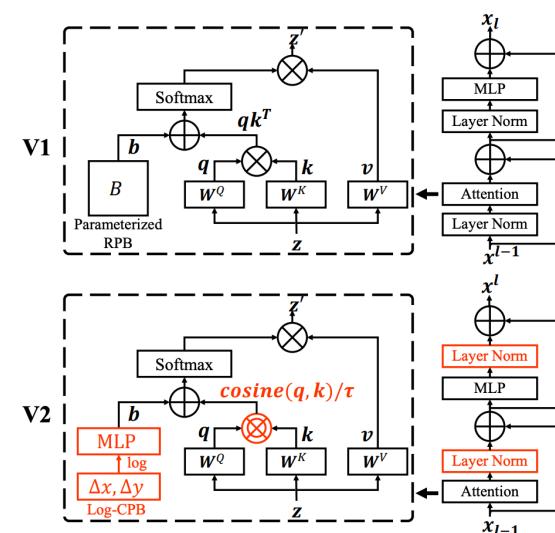
# Other Hierarchical Vision Transformers

MViT



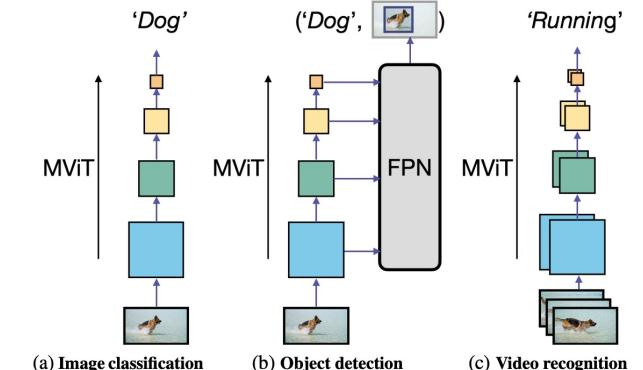
Fan et al, "Multiscale Vision  
Transformers", ICCV 2021

Swin-V2



Liu et al, "Swin Transformer V2: Scaling  
up Capacity and Resolution", CVPR 2022

Improved MViT



Li et al, "Improved Multiscale Vision Transformers  
for Classification and Detection", arXiv 2021

Thank you!

