Training a Transformer

Mohamed Abbas KONATE

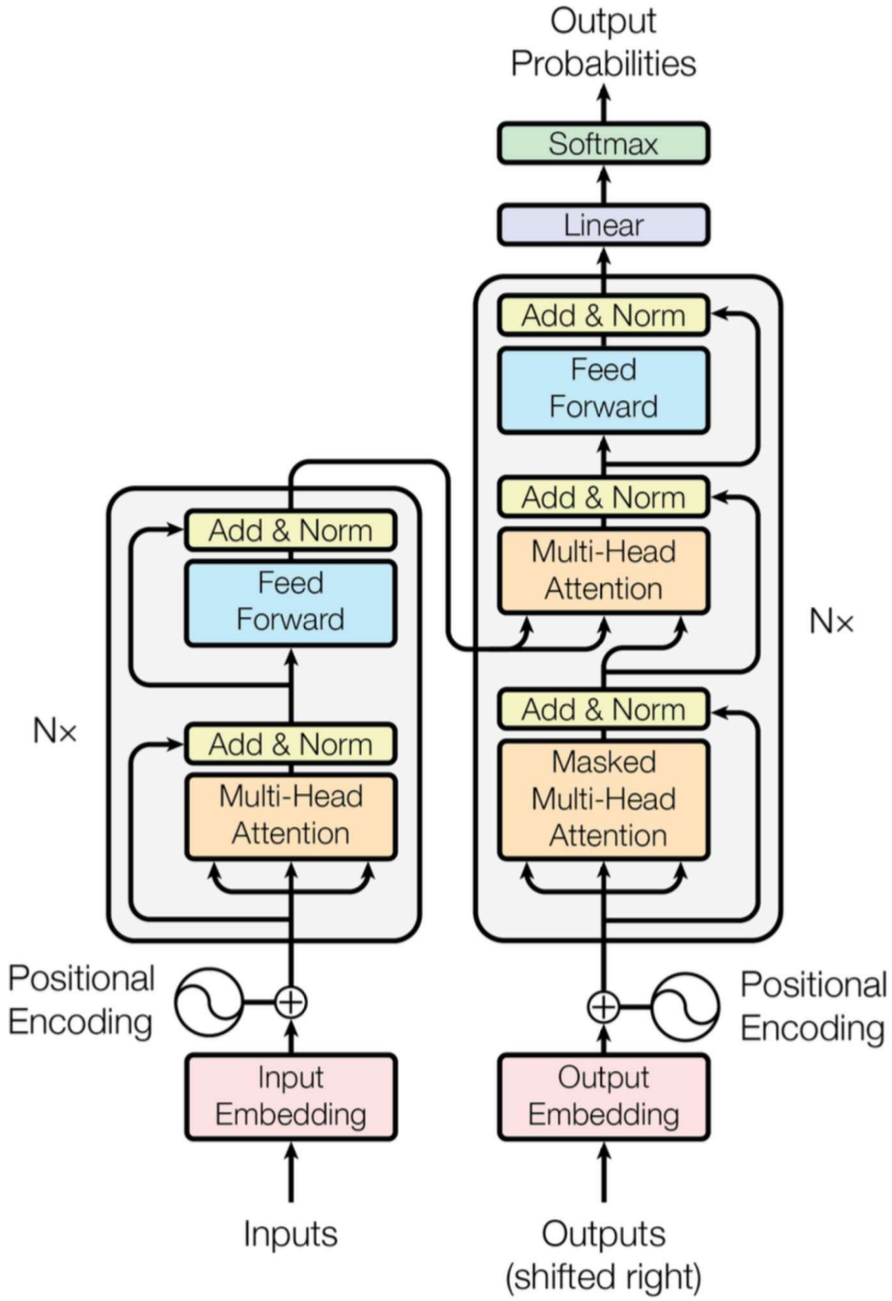
Plan

Rappel sur les transformers
Entrainer un transformer
Scaling Laws of transformer

But du chapitre : Découvrir comment on entraine un transformeur et l'adapte

Rappel

Rappel


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
N×
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
N×
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Transformer-based Language Modeling

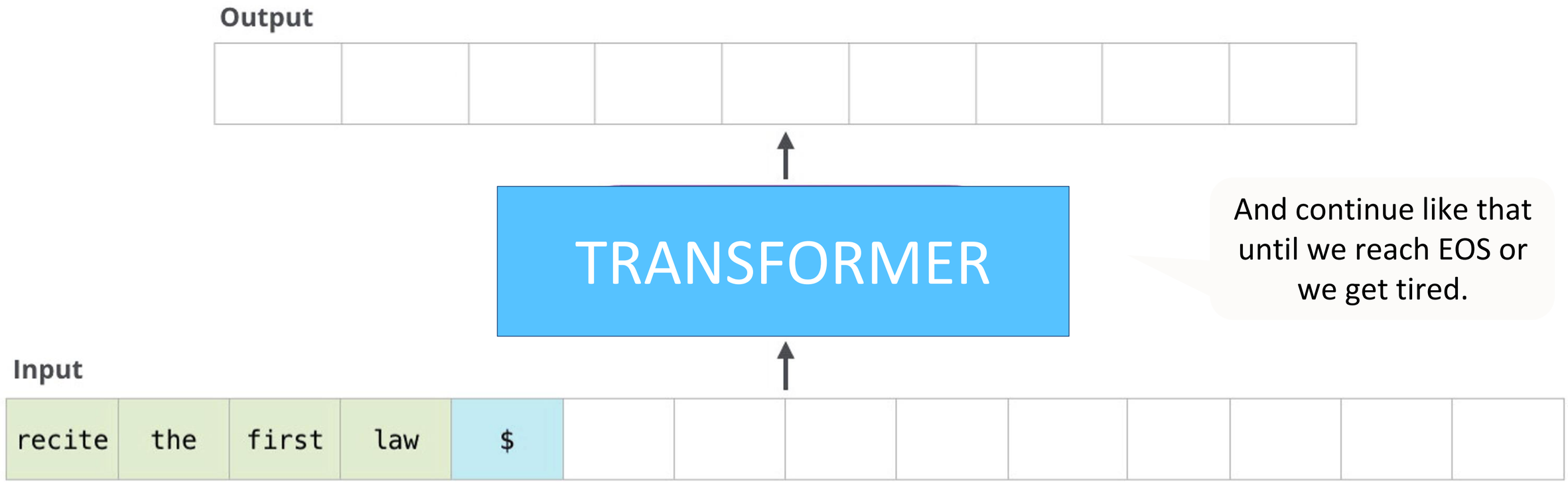
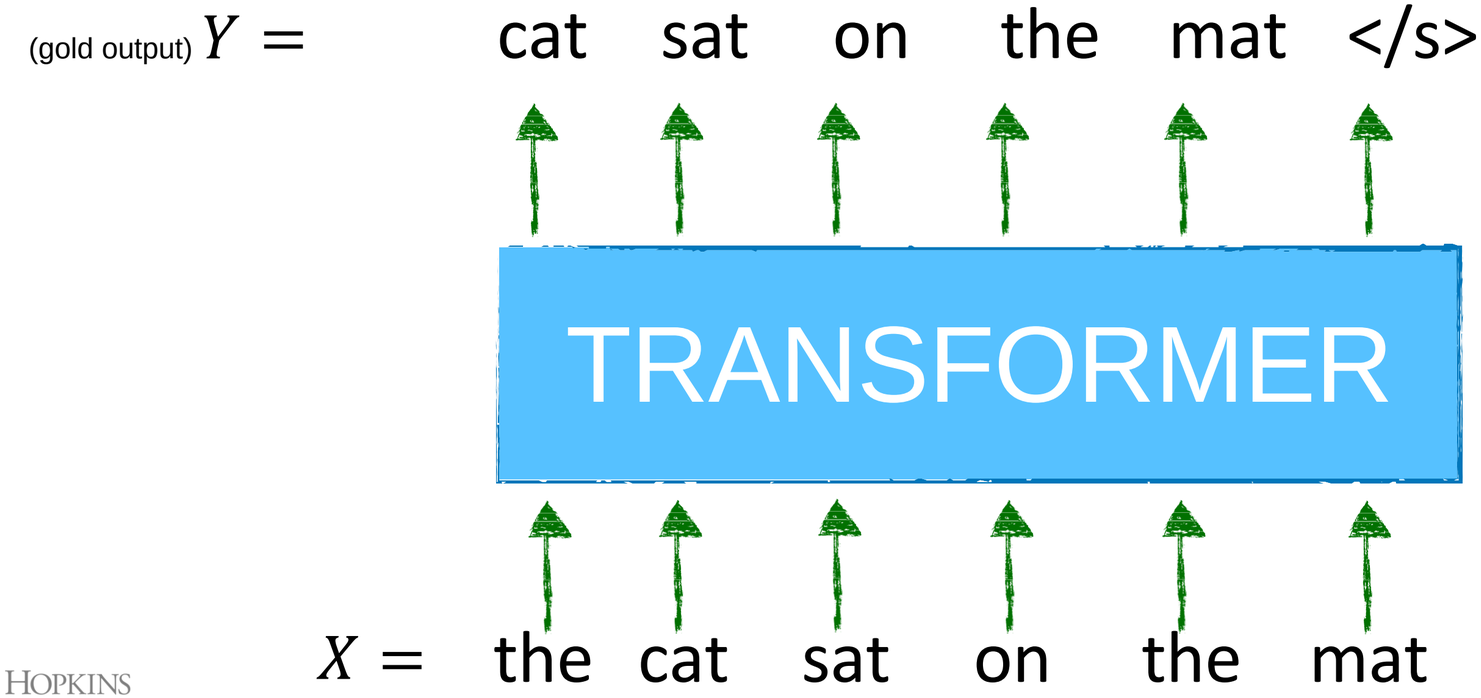


Image by http://jalammar.github.io/illustrated-gpt2/

Training a Transformer Language Model

- **Goal:** Train a Transformer for language modeling (i.e., predicting the next word).
- **Approach:** Train it so that each position is predictor of the next (right) token.
  - We just shift the input to right by one, and use as labels

EOS special token



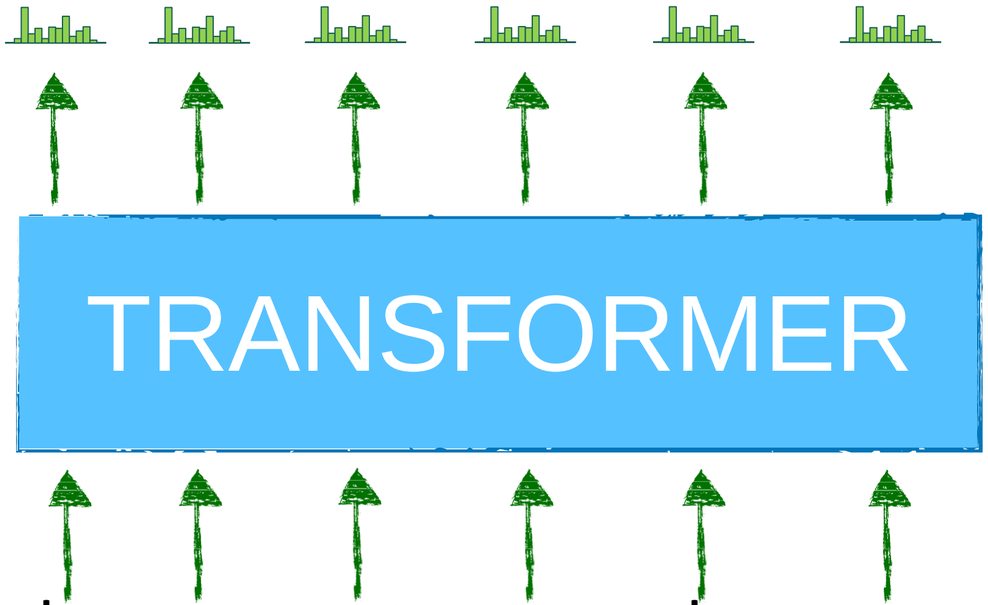

```
X = text[:, :-1]
Y = text[:, 1:]
```

[Slide credit: Arman Cohan]

Training a Transformer Language Model

- For each position, compute their corresponding **distribution** over the whole vocab.

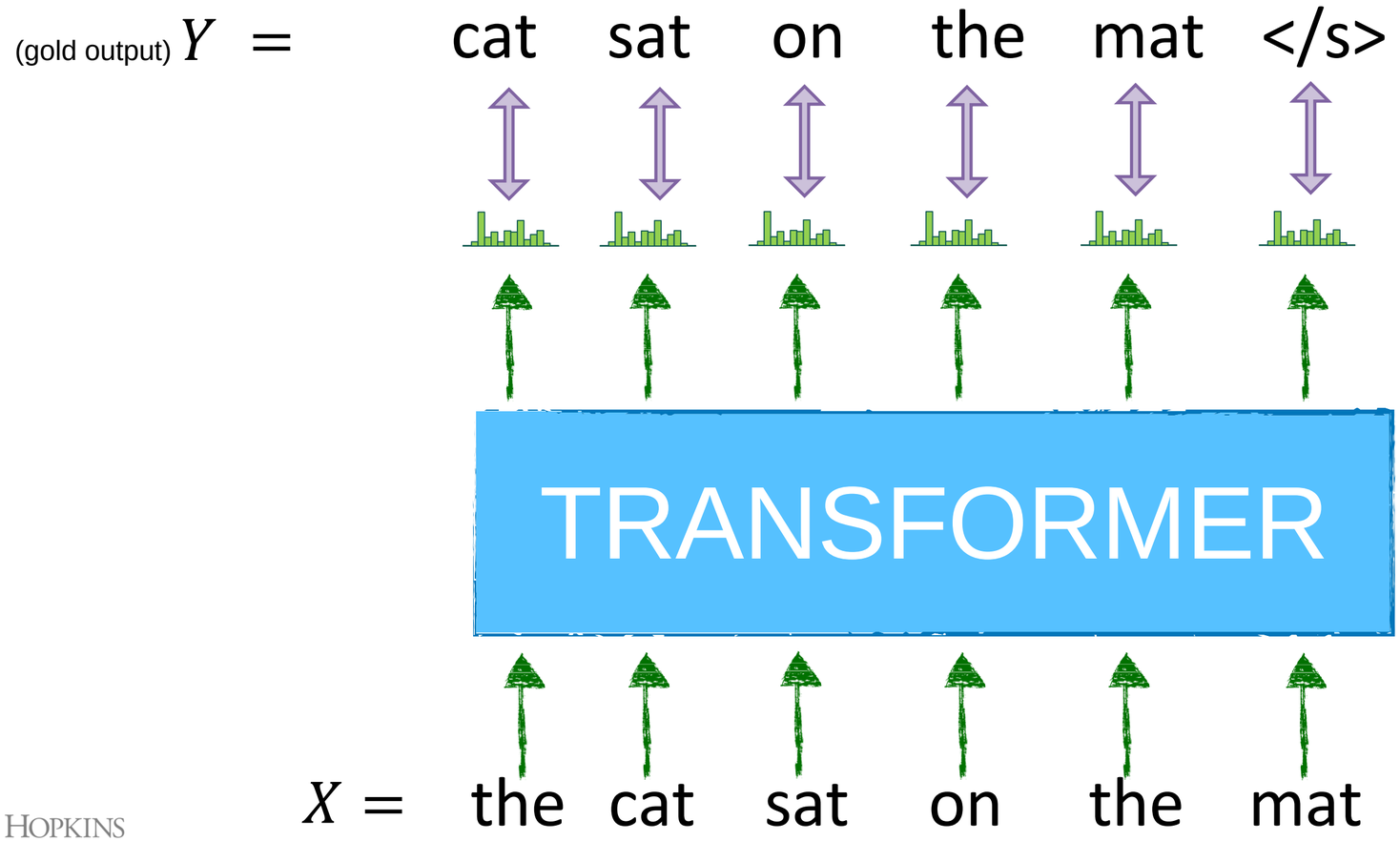
(gold output) $Y =$ cat sat on the mat </s>



$X =$ the cat sat on the mat

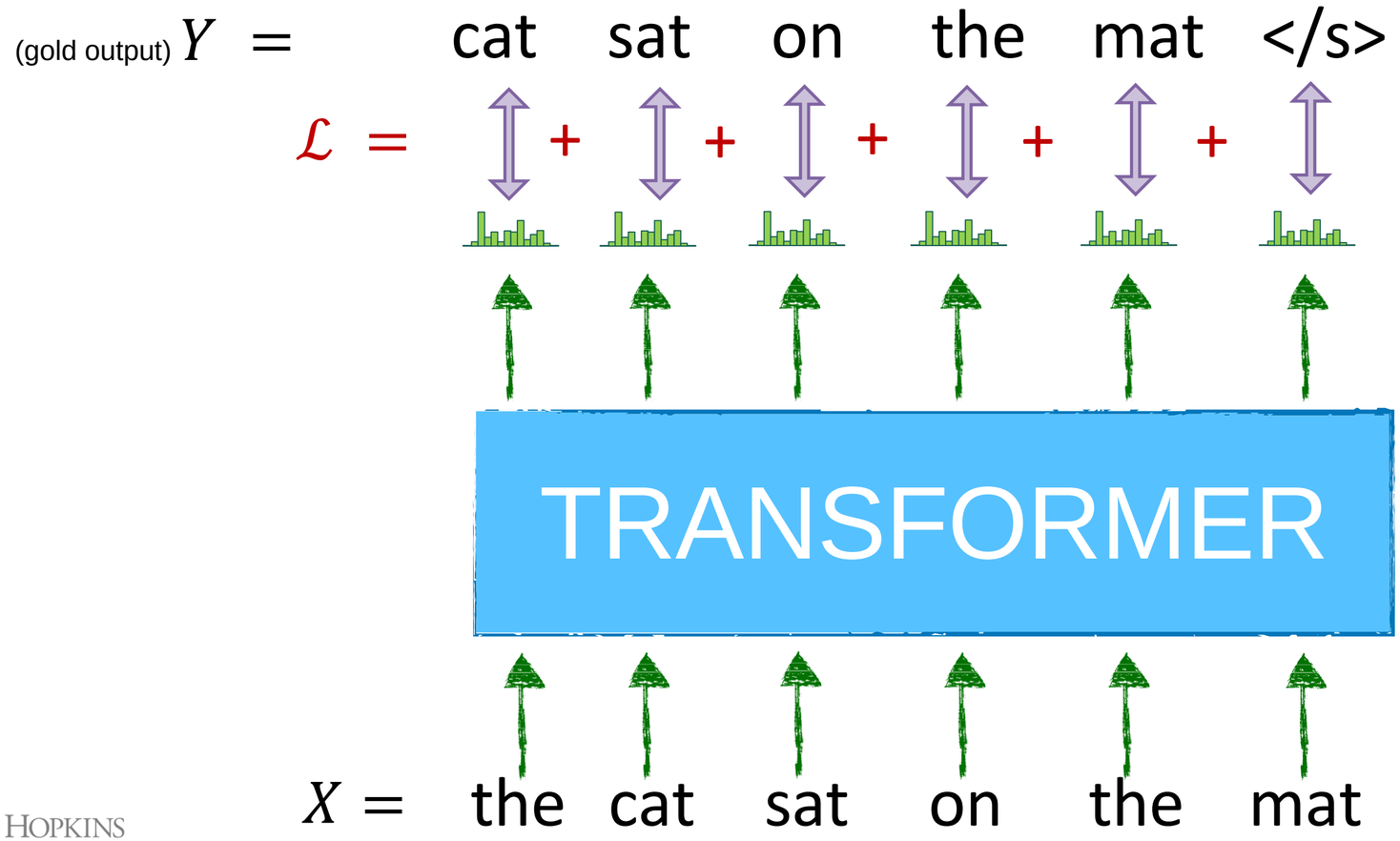
Training a Transformer Language Model

- For each position, compute the **loss** between the distribution and the gold output label.



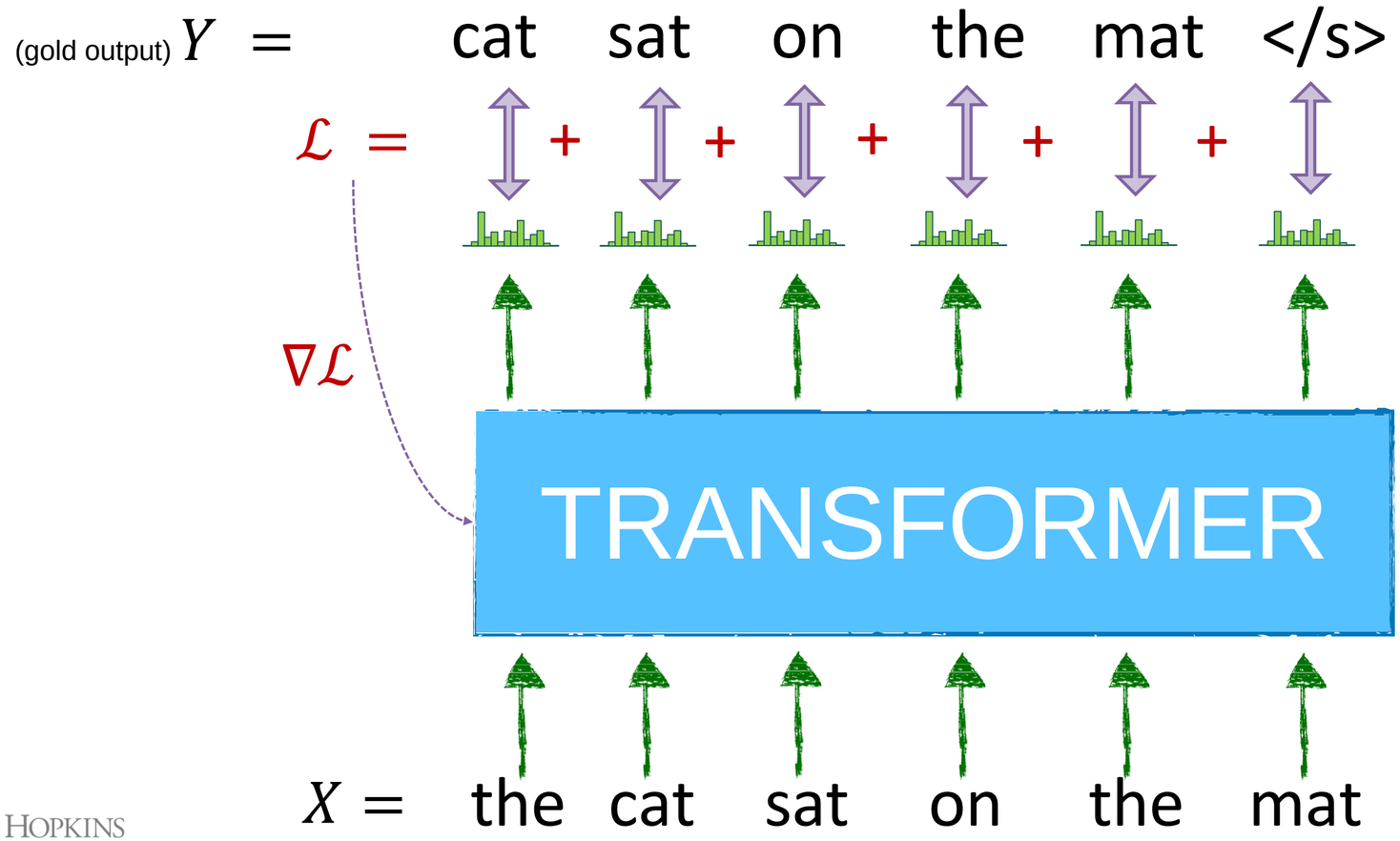
Training a Transformer Language Model

- Sum the position-wise loss values to a obtain a **global loss**.



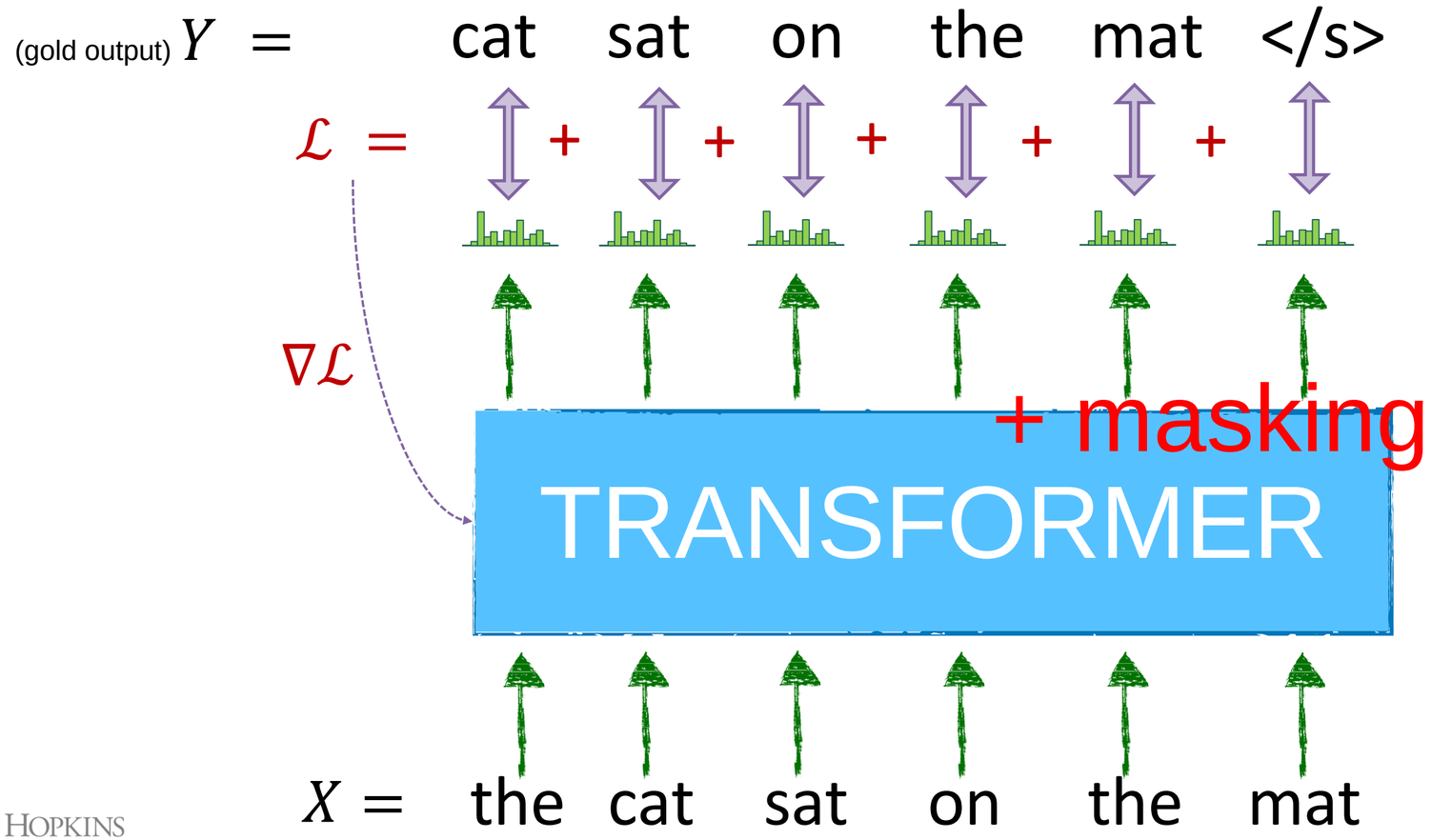
Training a Transformer Language Model

- Using this loss, do **Backprop** and **update** the Transformer parameters.



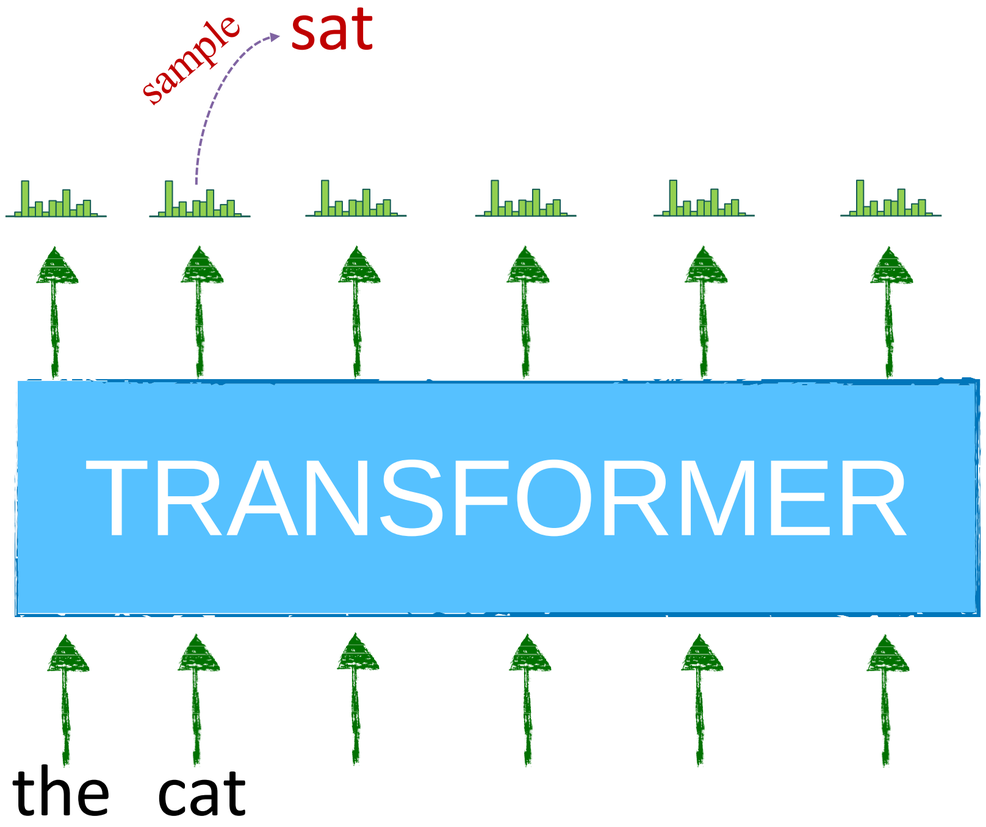
Training a Transformer Language Model

- We need to **prevent information leakage** from future tokens! How?



How to use the model to generate text?

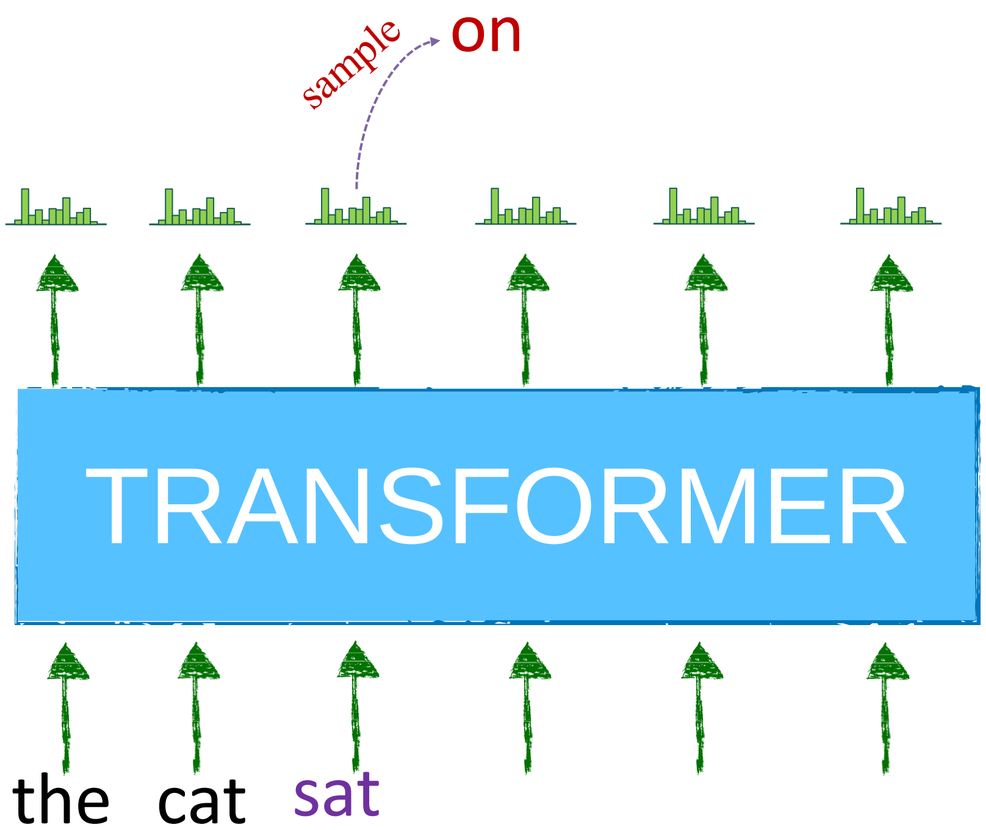
- Use the output of previous step as input to the next step repeatedly



How to use the model to generate text?

- Use the output of previous step as input to the next step repeatedly

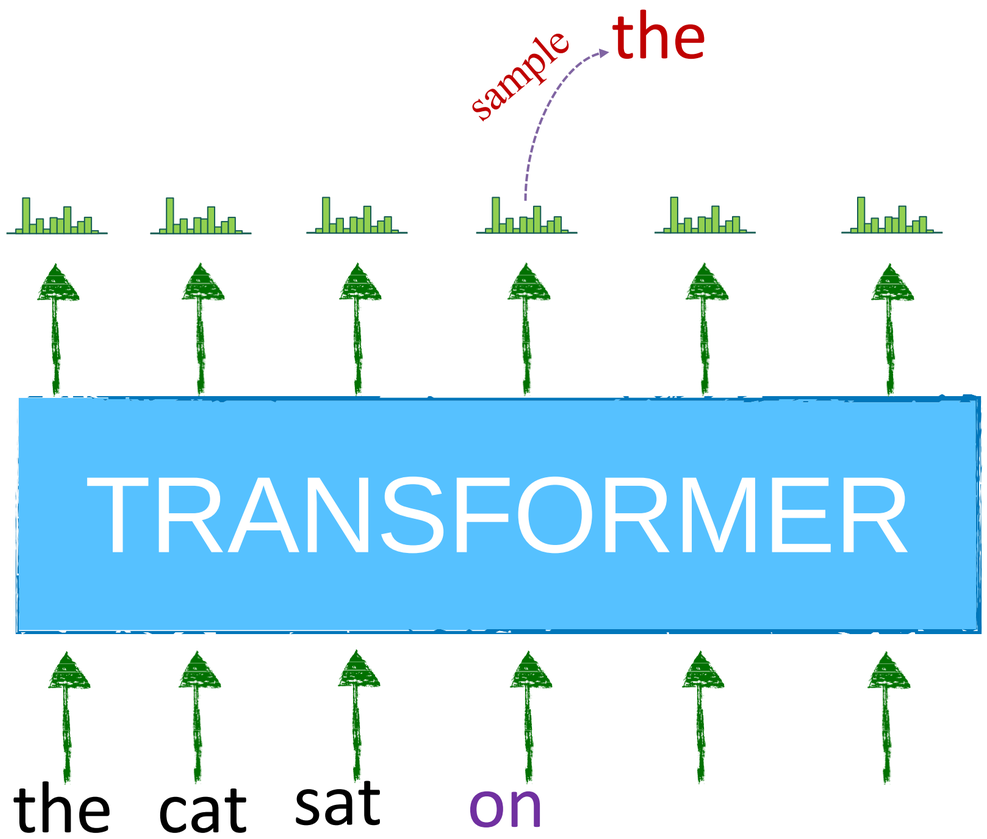
The probabilities get revised upon adding a new token to the input.



How to use the model to generate text?

- Use the output of previous step as input to the next step repeatedly

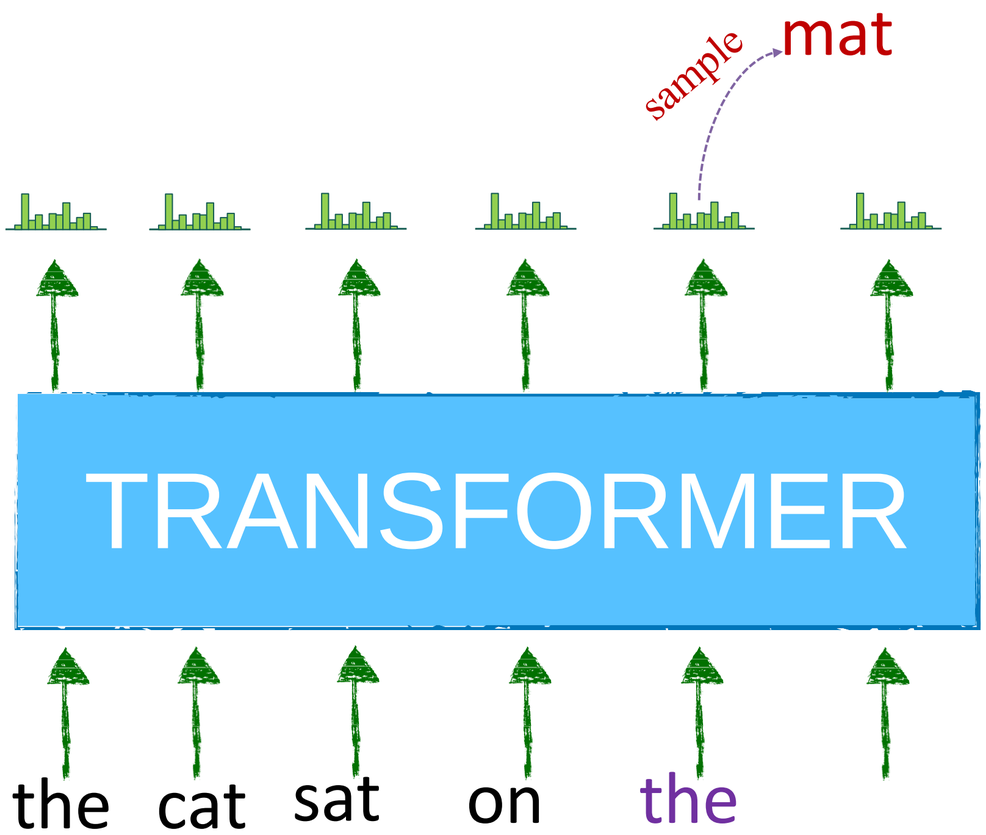
The probabilities get revised upon adding a new token to the input.



How to use the model to generate text?

- Use the output of previous step as input to the next step repeatedly

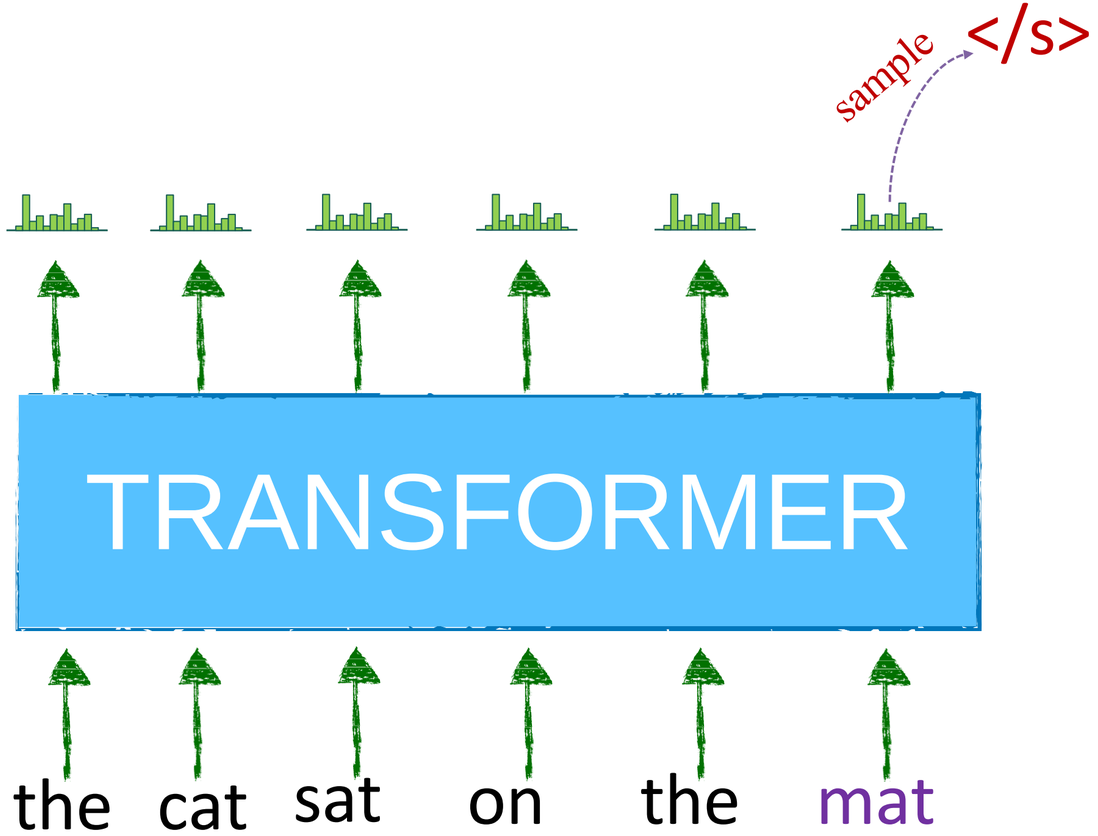
The probabilities get revised upon adding a new token to the input.



How to use the model to generate text?

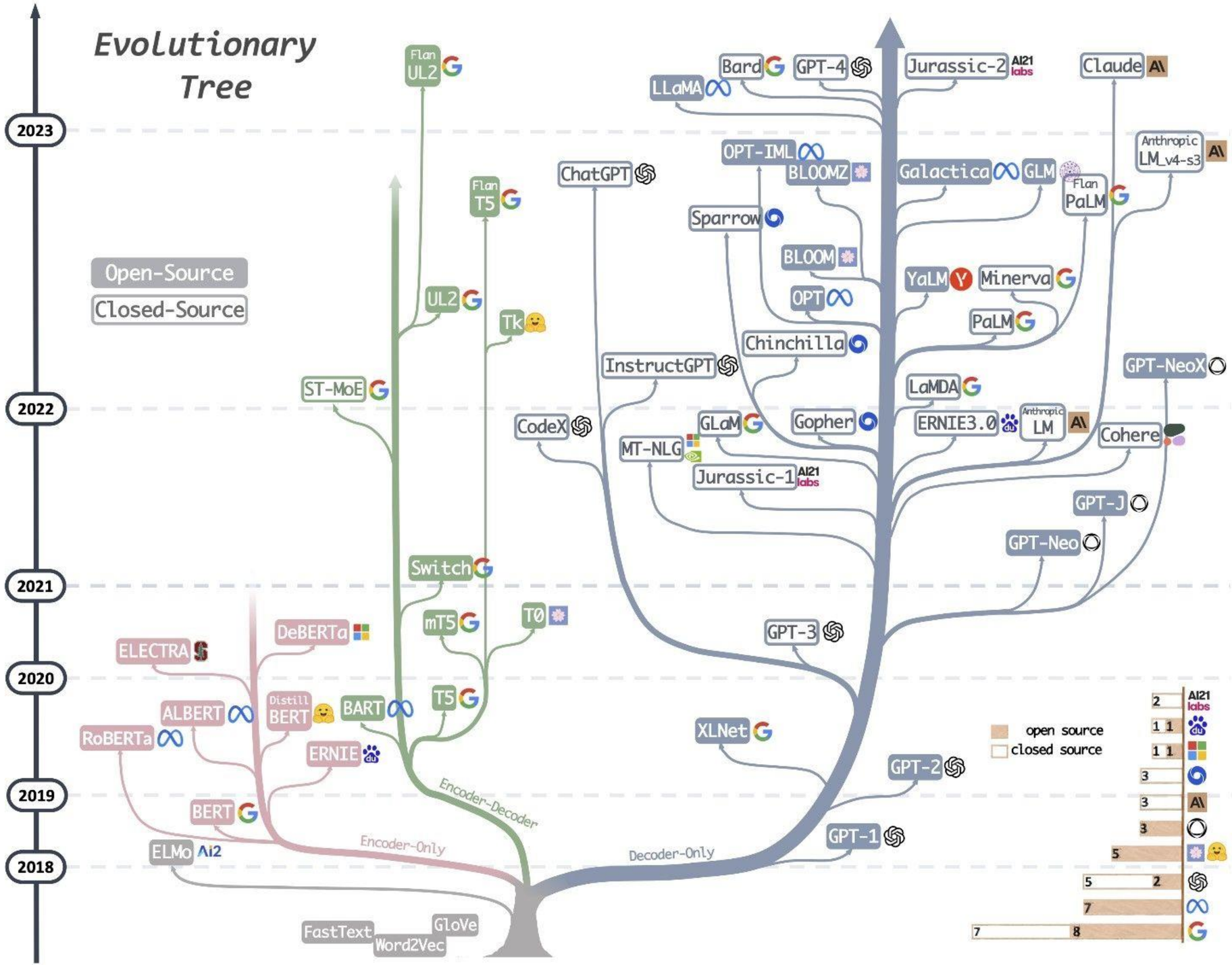
- Use the output of previous step as input to the next step repeatedly

The probabilities get revised upon adding a new token to the input.



Transformer Architectural Variants

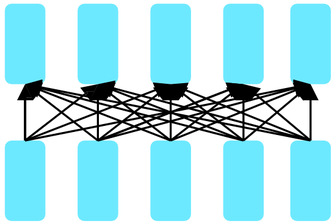
After Transformer …



Yang et al. Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond, 2023

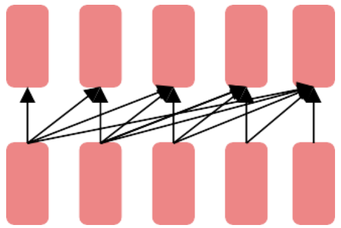
Impact of Transformers

- A building block for a variety of LMs



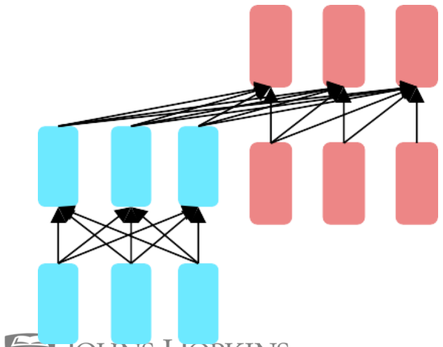
Encoders

- Examples: BERT, RoBERTa, SciBERT.
- Captures bidirectional context. Wait, how do we pretrain them?



Decoders

- Examples: GPT-2, GPT-3, LaMDA
- Other name: causal or auto-regressive language model
- Nice to generate from; can't condition on future words



Encoder-Decoders

- Examples: Transformer, T5, Meena
- What's the best way to pretrain them?

Pre-training objectives

On Pre-training Objectives

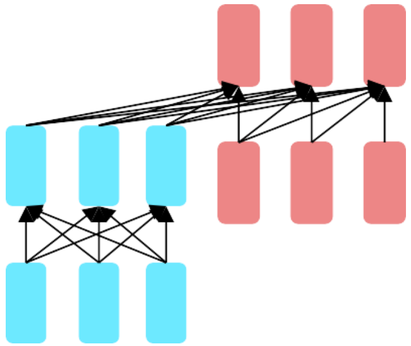
- So far, the dominant objective we have seen is “next-token” prediction.
- In reality any “marginal” observations about language can be a source of supervision.

Objectives

- Prefix language modeling
  - **Input:** Thank you for inviting
  - **Output:** me to your party last week
- BERT-style denoising
  - **Input:** Thank you <M> <M> me to your party apple week
  - **Output:** Thank you for inviting me to your party last week
- Deshuffling
  - **Input:** party me for your to. last fun you inviting week Thanks.
  - **Output:** Thank you for inviting me to your party last week
- IID noise, replace spans
  - **Input:** Thank you <X> me to your party <X> week
  - **Output:** <X> for inviting <Y> last <Z>
- IID noise, drop tokens
  - **Input:** Thank you me to your party week .
  - **Output:** for inviting last

Transformer Language Model Families

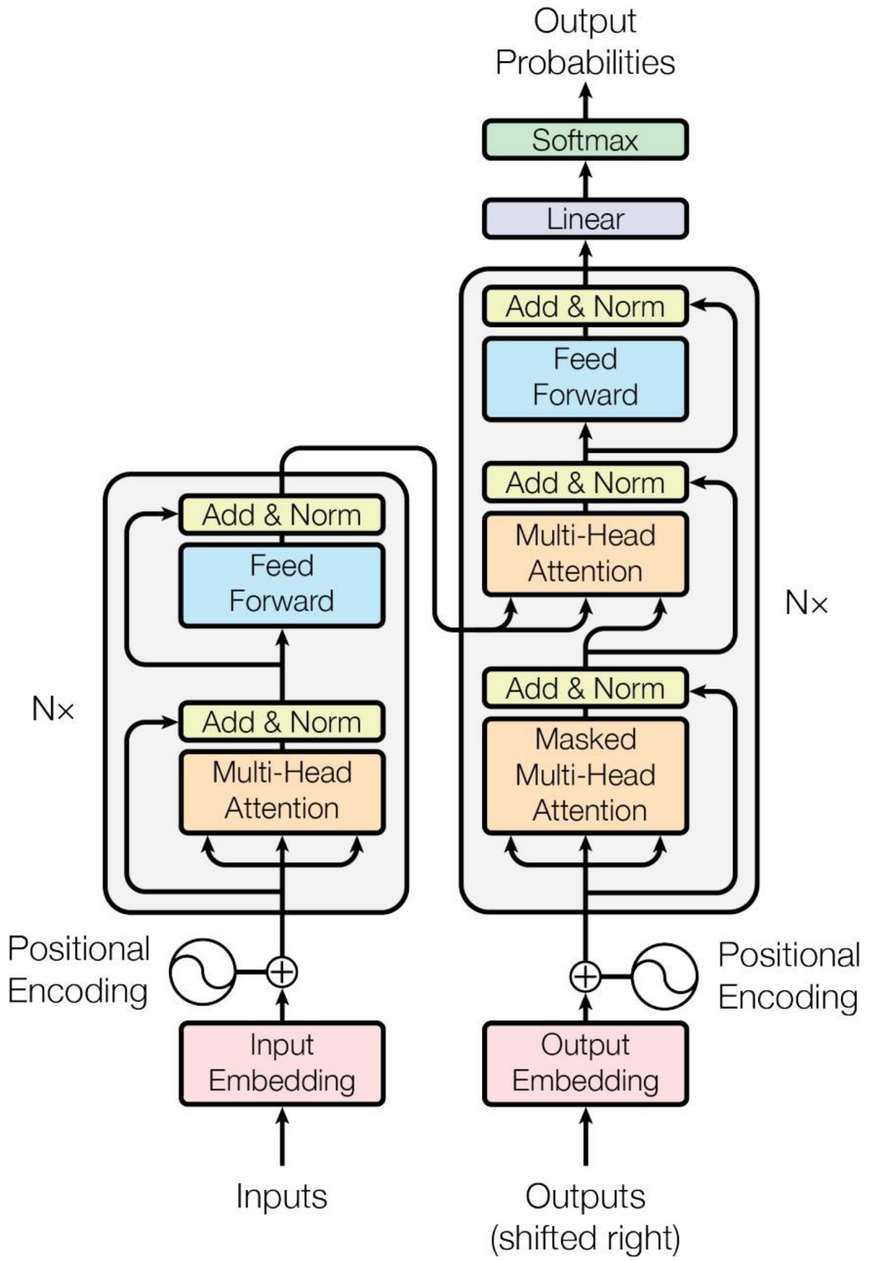
Encoder-Decoder Family of Transformers



Encoder-Decoders

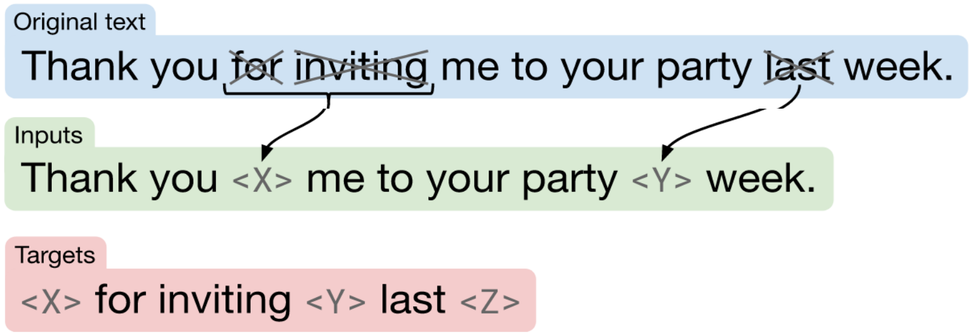
Encoder-decoder Models

- The original transformer architecture was encoder decoder
- Encoder-decoder models are flexible in both generation and classification tasks



T5: Text-To-Text Transfer Transformer

- An encoder-decoder architecture
- Pre-training objective:
  corrupt and reconstruct objective

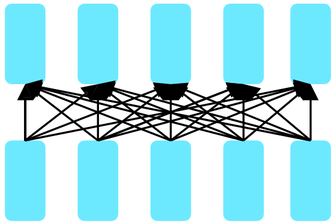


Model	Parameters	No. of layers	d_{model}	d_{ff}	d_{kv}	No. of heads
Small	60M	6	512	2048	64	8
Base	220M	12	768	3072	64	12
Large	770M	24	1024	4096	64	16
3B	3B	24	1024	16384	128	32
11B	11B	24	1024	65536	128	128

- The original paper is an excellent set of in-depth analysis of various parameters of model design. We discuss some of these results in other places.

https://huggingface.co/t5-base

Encoder-only Family of Transformers



BERT

Bidirectional Encoder Representations from Transformers

BERT

Bidirectional Encoder Representations from Transformers

BERT

Bidirectional Encoder Representations from Transformers

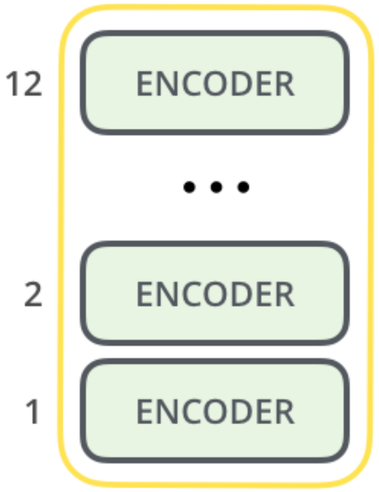
Let's only use Transformer Encoders, no Decoders

Bidirectional Encoder Representations from Transformers

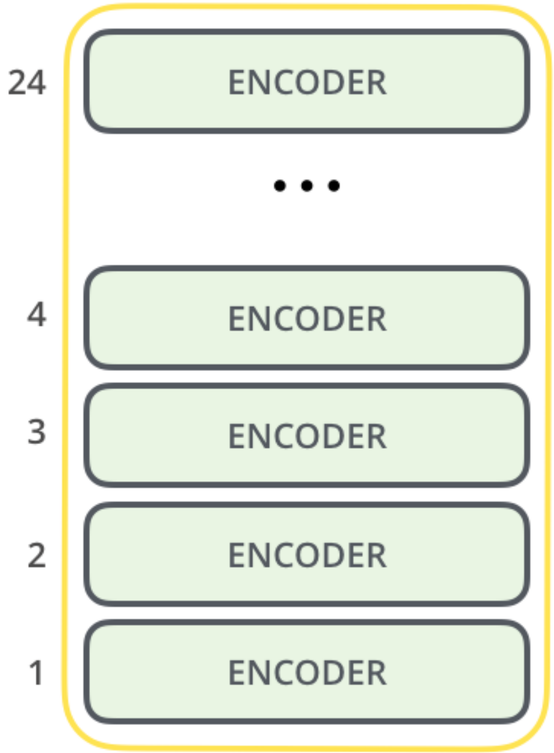
It's a language model that builds rich representations
via self-supervised learning (pre-training)

BERT: Architecture

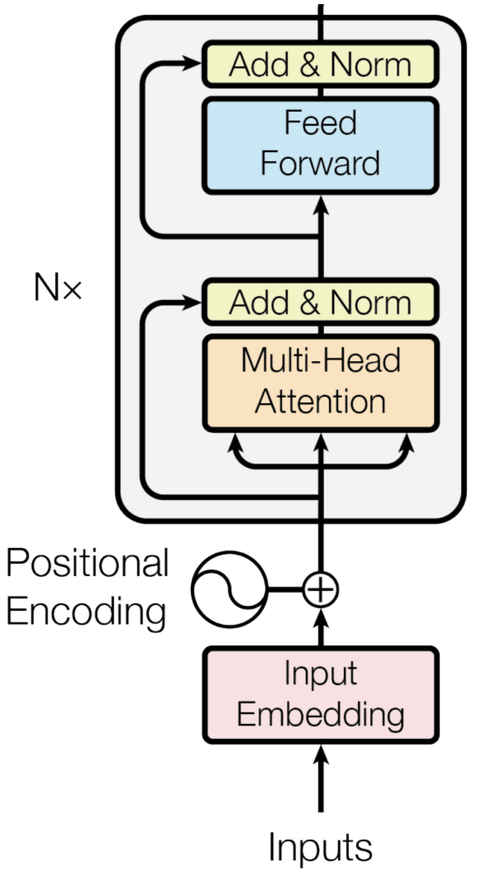
- Stacks of Transformer encoders



$BERT_{BASE}$



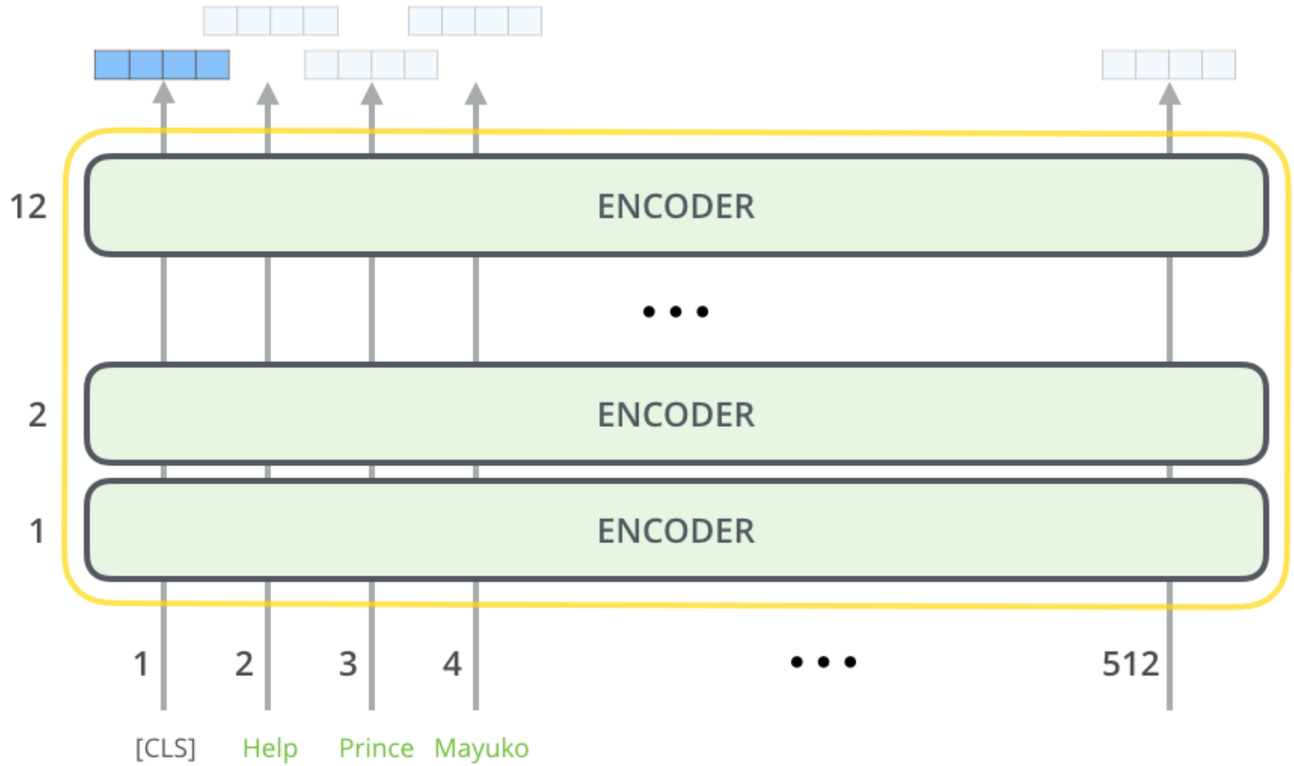
$BERT_{LARGE}$

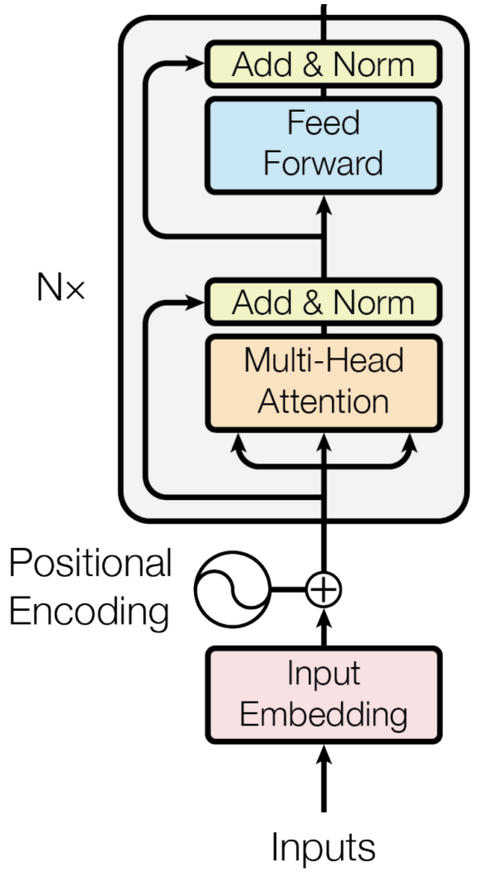


[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]

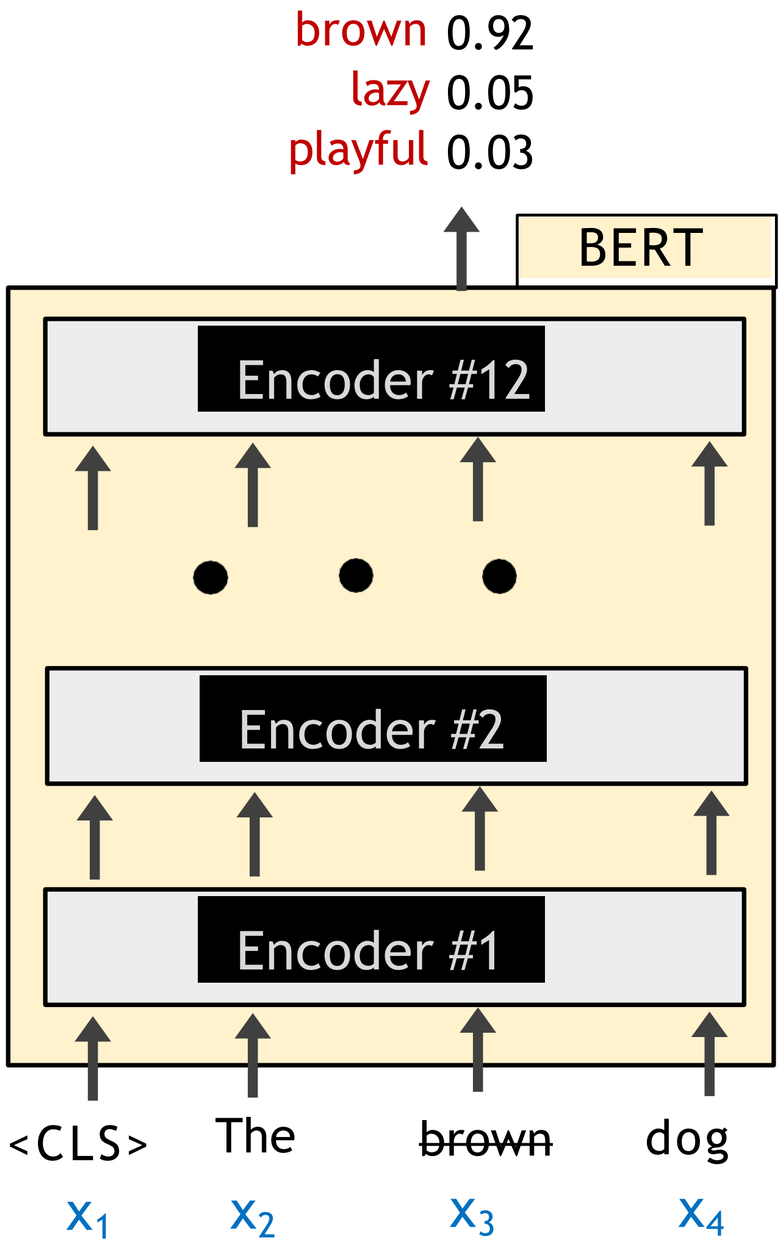
BERT: Architecture

- Model output dimension: 512





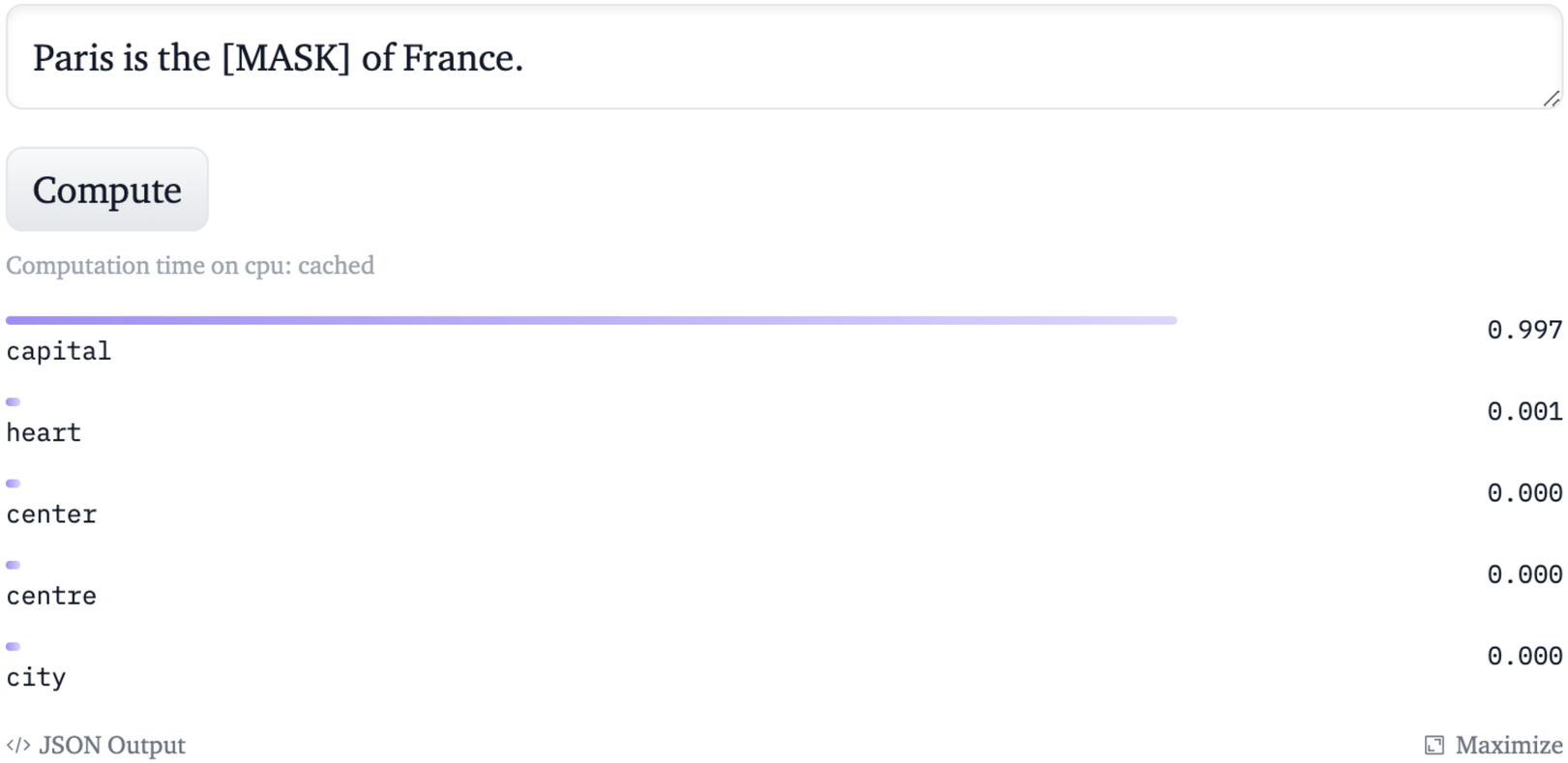
[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]



BERT is trained to uncover masked tokens.

Probing BERT Masked LM

- Masking words forces BERT to use context in both directions to predict the masked word.

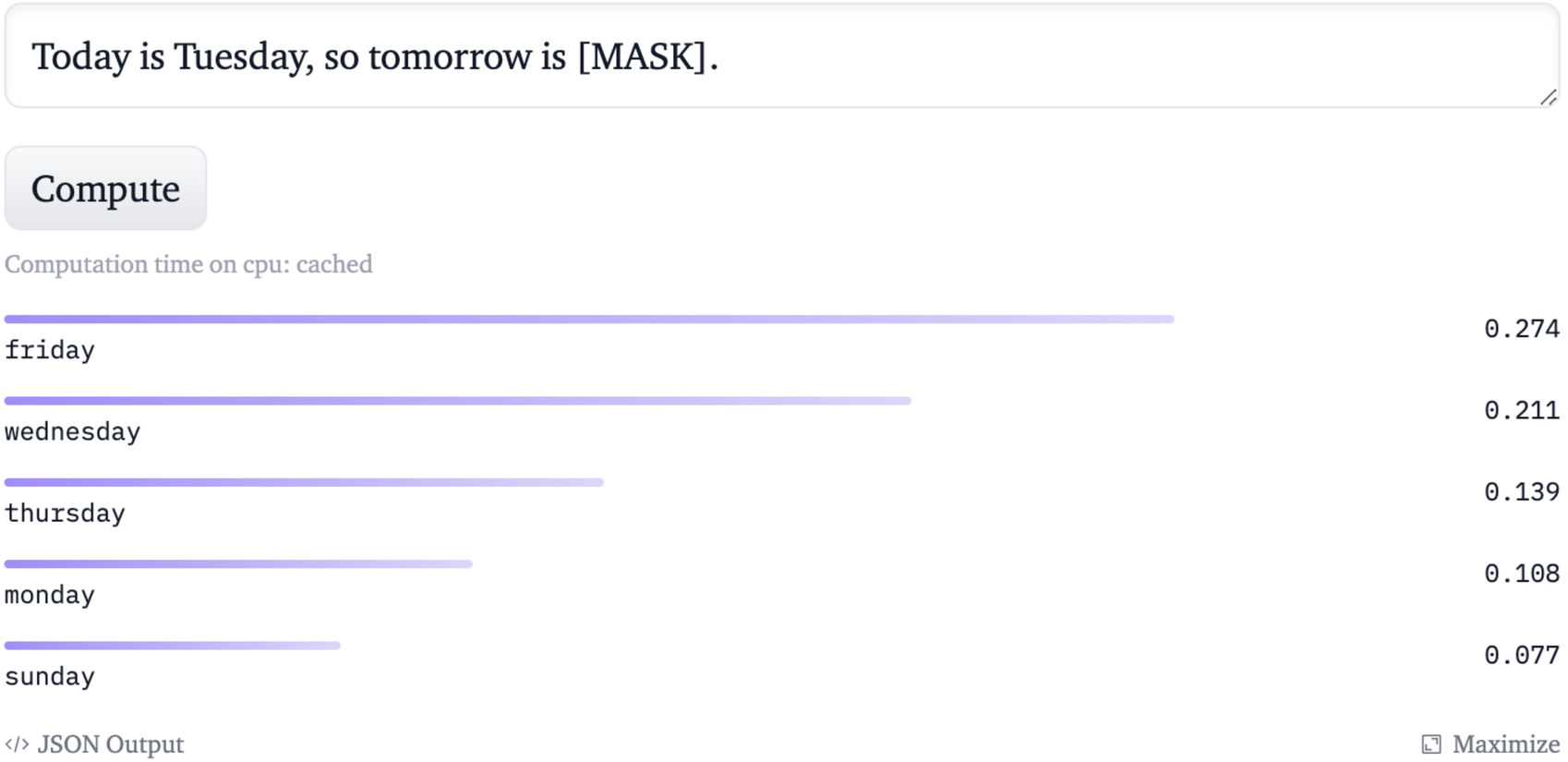


https://huggingface.co/bert-base-uncased

37

Probing BERT Masked LM

- Masking words forces BERT to use context in both directions to predict the masked word.

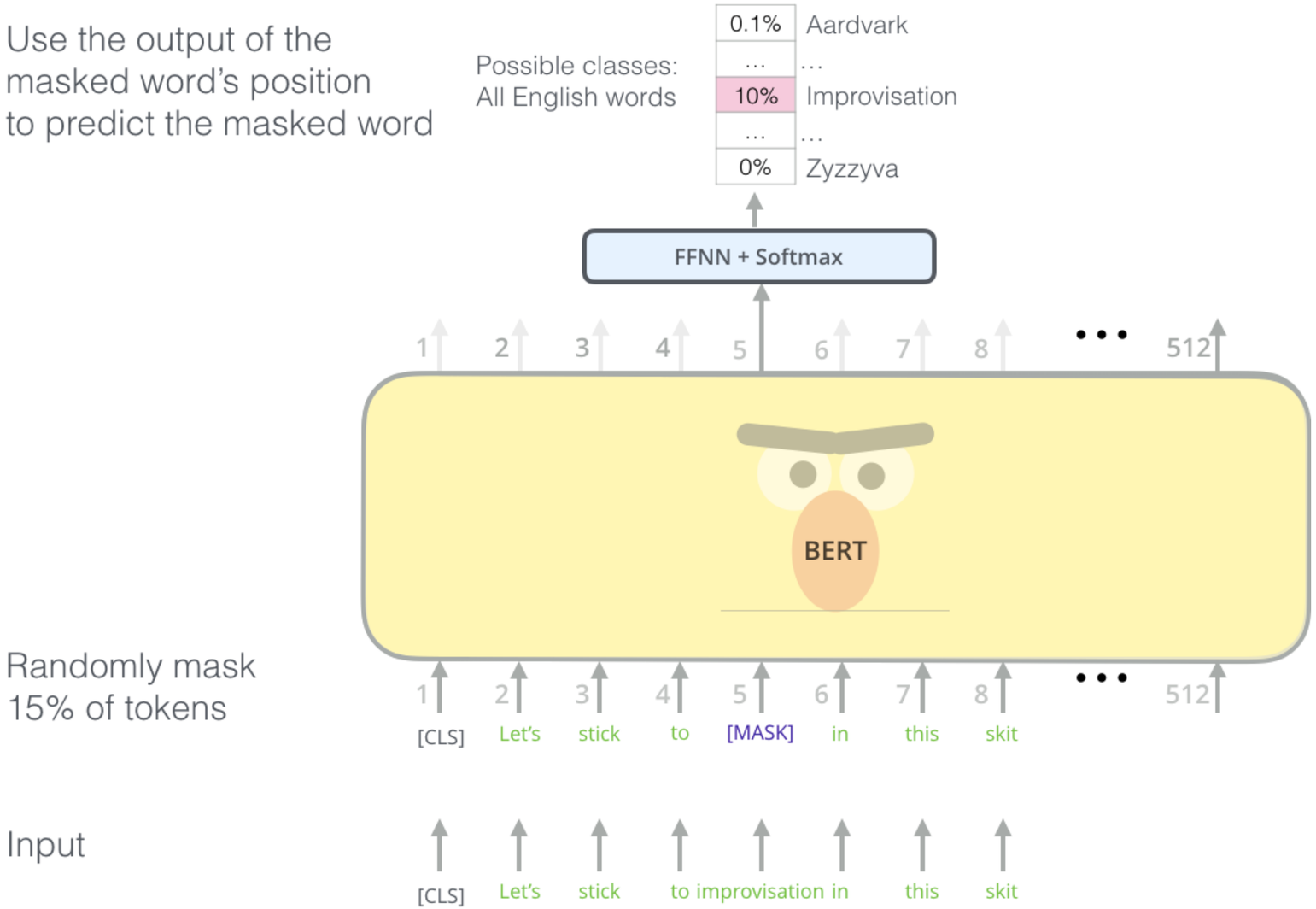


https://huggingface.co/bert-base-uncased

38

BERT: Pre-training Objective : Masked Tokens

- Randomly mask 15% of the tokens and train the model to predict them.



[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]

BERT: Pre-training Objective (1): Masked Tokens

`the man went to the [MASK] to buy a [MASK] of milk`

- Too little masking: Too expensive to train
- Too much masking: Underdefined
  - (not enough info for the model to recover the masked tokens)

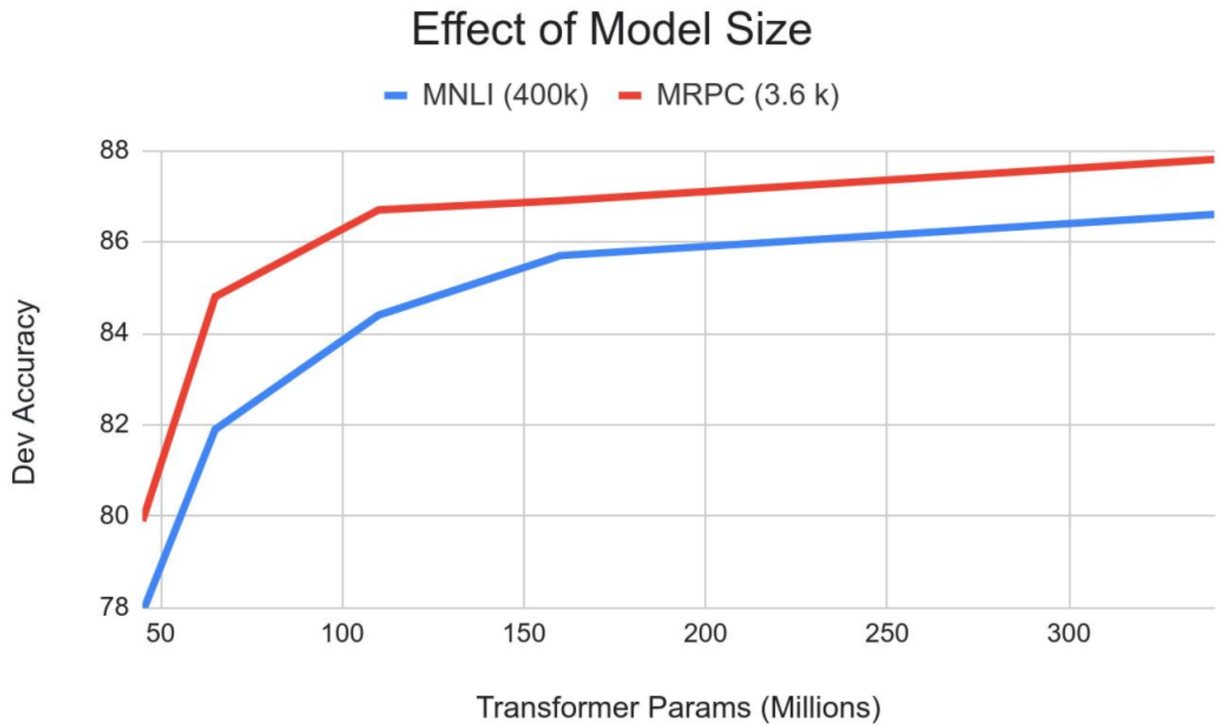
[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]

Training

- Trains model on unlabeled data over different pre-training tasks (self-supervised learning)
- **Data:** Wikipedia (2.5B words) + BookCorpus (0.8B words)
- **Training Time:** 1M steps (~40 epochs)
- **Optimizer:** AdamW, 1e-4 learning rate, linear decay
- **BERT-Base:** 12-layer, 768-hidden, 12-head, sequence length of 512
- **BERT-Large:** 24-layer, 1024-hidden, 16-head, sequence length of 512
- Trained on 4x4 and 8x8 TPUs for 4 days (cost today using cloud TPU: $1.3K and $5K)

[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]

Effect of Model Size



- Big models help a lot
- Going from 110M -> 340M params helps even on datasets with 3,600 labeled examples
- Improvements have **not** plateaued!

[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]

Impact of BERT

- In order to have state-of-the-art performance on different tasks, there is no need for coming up with a novel model architecture
  - End of task-specific model architecture engineering

- An early sign that larger scales and self-supervised learning (language modeling) are the key for future performance improvements

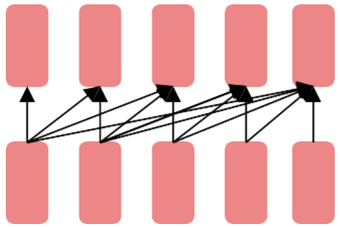
- Better for tasks requiring analysis and questions answering

Summary Thus Far

- BERT and the family
- An encoder; Transformer-based networks trained on massive piles of data.
- Incredible for learning contextualized embeddings of words
- It's very useful to pre-train a large unsupervised/self-supervised LM then fine-tune on your particular task (replace the top layer, so that it can work)
- However, they were not designed to generate text.

https://huggingface.co/datasets/rajpurkar/squad?row=4

Decoder-only Family of Transformers



Decoders

GPT

Generative Pre-trained Transformer

GPT-2: A Big Language Model (2019)

Language Models are Unsupervised Multitask Learners

Alec Radford [*1] Jeffrey Wu [*1] Rewon Child [1] David Luan [1] Dario Amodei [**1] Ilya Sutskever [**1]

GPT: An Auto-Regressive LM (2018)

Improving Language Understanding by Generative Pre-Training

Alec Radford
OpenAI
alec@openai.com

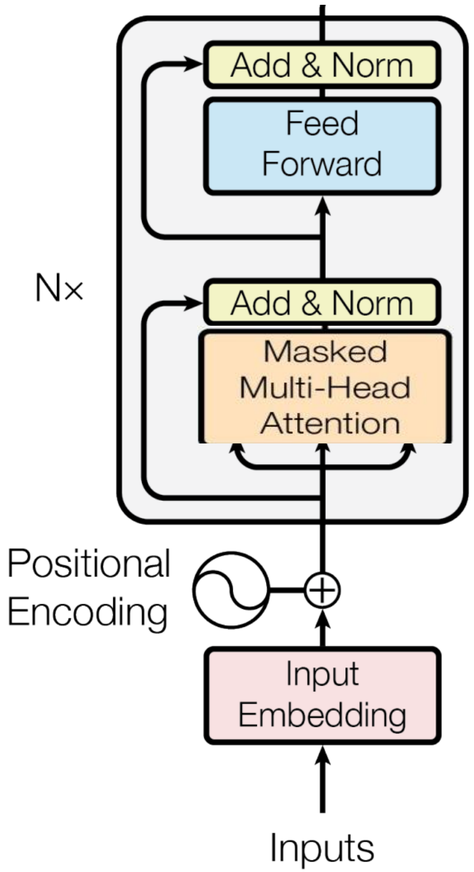
Karthik Narasimhan
OpenAI
karthikn@openai.com

Tim Salimans
OpenAI
tim@openai.com

Ilya Sutskever
OpenAI
ilyasu@openai.com

GPT: Architecture

- Stacks of Transformer decoders



[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018]

GPT-2

- GPT-2 uses only Transformer Decoders (no Encoders) to generate new sequences from scratch or from a starting sequence
- As it processes each subword, it masks the "future" words and conditions on and attends to the previous words

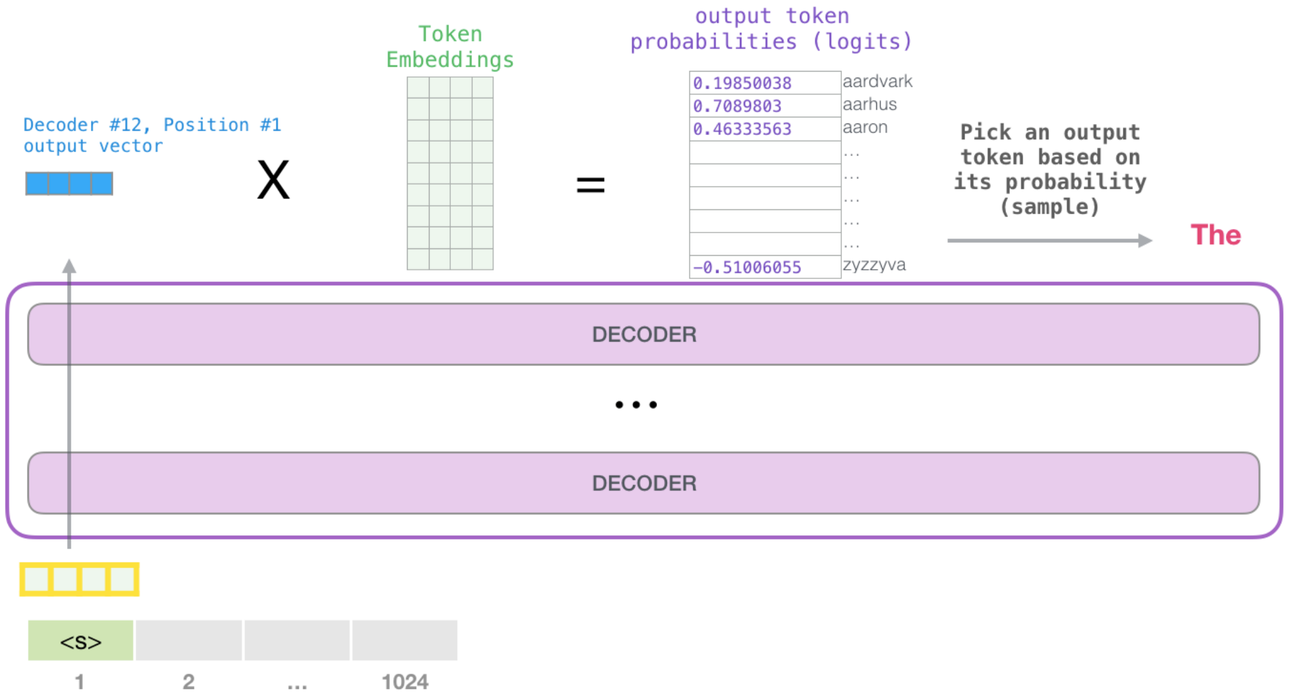
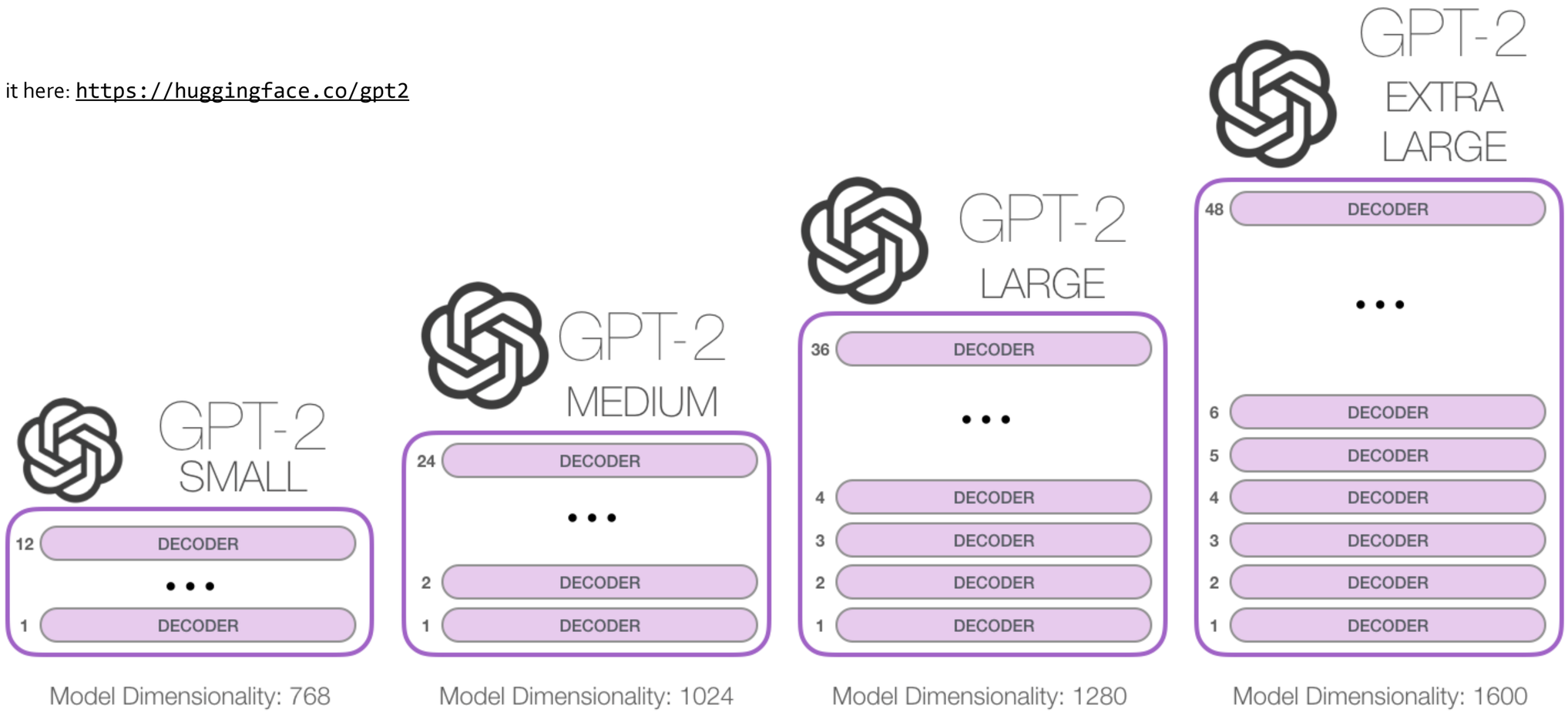


Image by http://jalammar.github.io/illustrated-gpt2/

GPT2: Model Sizes

Play with it here: https://huggingface.co/gpt2



117M parameters

345M

762M

1542M

[Image by http://jalammar.github.io/illustrated-gpt2/]

GPT-2 is identical to GPT-1, but:

- Has Layer normalization in between each sub-block (as we've already seen)
- Vocab extended to 50,257 tokens and context size increased from 512 to 1024

Language Models are Unsupervised Multitask Learners

Alec Radford [*1] Jeffrey Wu [*1] Rewon Child [1] David Luan [1] Dario Amodei [**1] Ilya Sutskever [**1]

Article: Prehistoric man sketched an incredible array of prehistoric beasts on the rough limestone walls of a cave in modern day France 36,000 years ago.
Now, with the help of cutting-edge technology, those works of art in the Chauvet-Pont-d'Arc Cave have been reproduced to create the biggest replica cave in the world.
The manmade cavern named the Caverne du Pont-d'Arc has been built a few miles from the original site in Vallon-Pont-D'arc in Southern France and contains 1,000 painstakingly-reproduced drawings as well as around 450 bones and other features...
Cavemen and women sketched an incredible array of prehistoric beasts on the rough limestone walls of a cave 36,000 years ago and now a replica has been created (pictured)
...

GPT-2: The original site in Vallon-Pont-D'arc in Southern France is a Unesco World Heritage site and is the oldest known and the best preserved cave decorated by man. The replica cave was built a few miles from the original site in Vallon-Pont-D'Arc in Southern France. The cave contains images of 14 different species of animals including woolly rhinoceros, mammoths, and big cats.	**Reference:** Cave mimics famous Caverne du Pont-d'Arc in France, the oldest cave decorated by man and the best preserved. The replica contains all 1,000 paintings which include 425 such as a woolly rhinoceros and mammoths. Minute details were copied using 3D modelling and anamorphic techniques, often used to shoot widescreen images. The modern cave also includes replica paw prints of bears, bones and details preserved in the original cave.

Impact of GPT2

- Zero-shot learning (no use of task-specific supervision) increasingly become a reality.

NMT: "Translate to french," <English text>, <French text>.

QA: "Answer the question," <Document>, <Question>, <Answer>.

SUMM: <Document> "TL; DR:" <Summarization>

GPT-3: A Very Large Language Model (2020)

More layers & parameters
Bigger dataset
Longer training
Larger embedding/hidden dimension
Larger context window

[Slide credit: Sbhya Chhabria & Michael Tang]

Size Comparisons

BERT-Base model has 12 transformer blocks, 12 attention heads,
110M parameters!

BERT-Large model has 24 transformer blocks, 16 attention heads,
340M parameters!

GPT-2 is trained on 40GB of text data (8M webpages)!
1.5B parameters!

GPT-3 is an even bigger version of GPT-2, but isn't open-source
175B parameters!

Impact of GPT3

- Moving away from the fine-tuning paradigm
  - Zero/Few-shot learning and in-context learning
- Massive LM scale makes high zero/few-shot performance possible
- Start of closed source models
  - Not too many details about their model
  - No released code / model checkpoint
- Also revitalized open source efforts:
  - OPT, LLaMA by Meta, BLOOM by Huggingface, etc.

GPT4

Model	Usage
davinci-002	$0.0020 / 1K tokens

Model	Input	Output
gpt-4	$0.03 / 1K tokens	$0.06 / 1K tokens

- Transformer-based
  - The rest is …. mystery! ☺
  - If we're going based on costs, GPT4 is ~15-30 times costlier than GPT3. That should give you an idea how its likely size!

- Note, these language models involve more than just pre-training.
  - Pre-training provides the foundation based on which we build the model.
  - We will discuss the later stages (post hoc alignment) in a 2-3 weeks.

https://openai.com/pricing

Other Available [Decoder] LMs

EleutherAI: GPT-Neo (6.7B), GPT-J (6B), GPT-NeoX (20B)

https://huggingface.co/EleutherAI

https://6b.eleuther.ai/

LLaMA, 65B: https://github.com/facebookresearch/llama

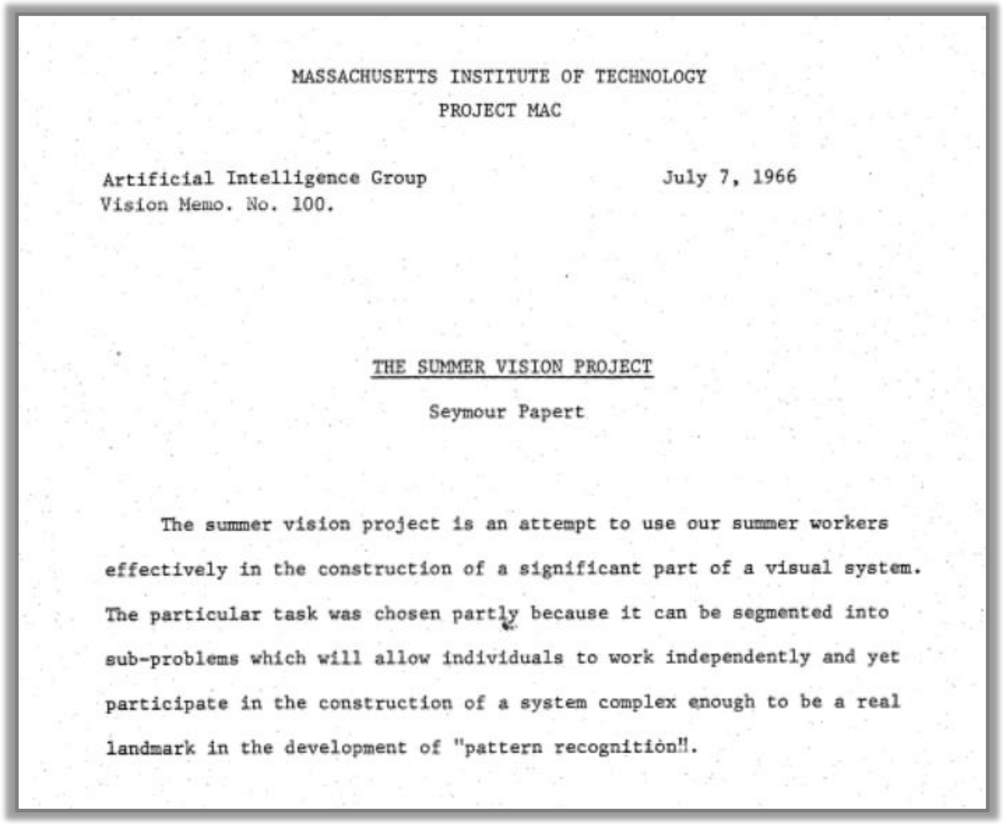
Mistral and Mixtral:

https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2

https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1

Computer Vision Abridged History

- 1960s - First computer vision projects (MIT summer project)



MASSACHUSETTS INSTITUTE OF TECHNOLOGY
PROJECT MAC

Artificial Intelligence Group
Vision Memo. No. 100.

July 7, 1966

THE SUMMER VISION PROJECT

Seymour Papert

The summer vision project is an attempt to use our summer workers effectively in the construction of a significant part of a visual system. The particular task was chosen partly because it can be segmented into sub-problems which will allow individuals to work independently and yet participate in the construction of a system complex enough to be a real landmark in the development of "pattern recognition".

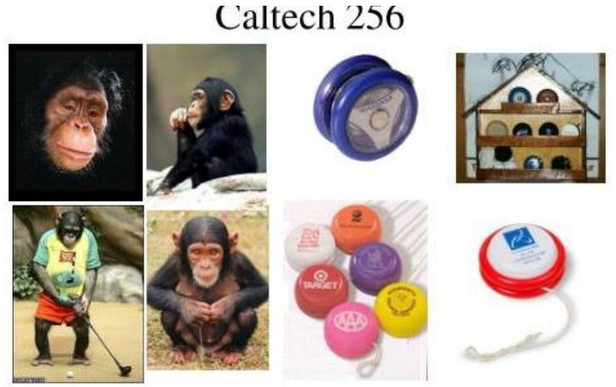
Computer Vision Abridged History

- 1960s - First computer vision projects (MIT summer project)
- 2000s - Emergence of “tasks” and benchmarking in computer vision

PASCAL Visual Object Classes (2005-2012)

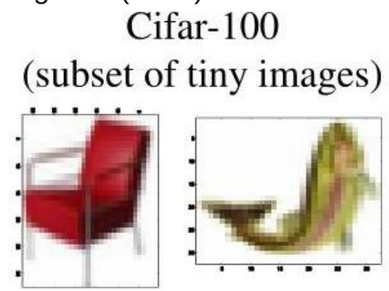
Caltech-101 (2003)

Caltech-256 (2007)



ImageNet (2009)

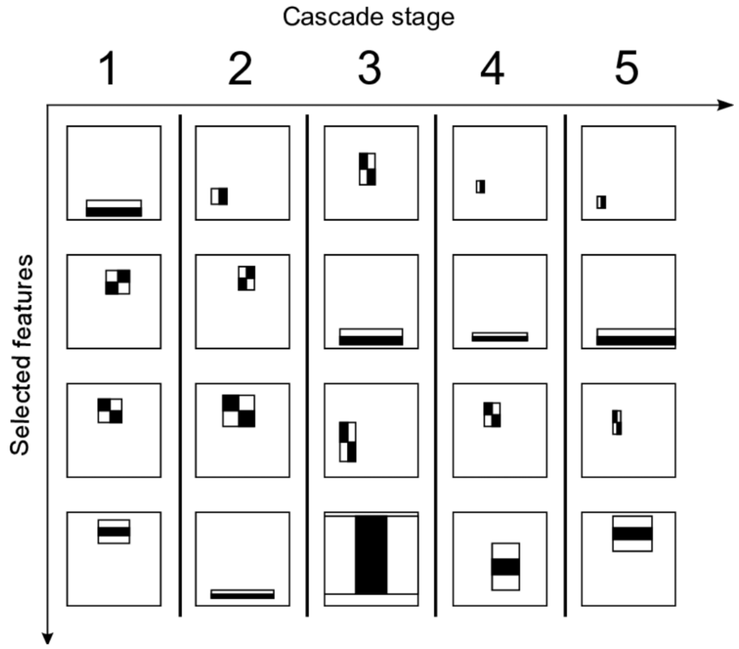
ImageNet (2009)

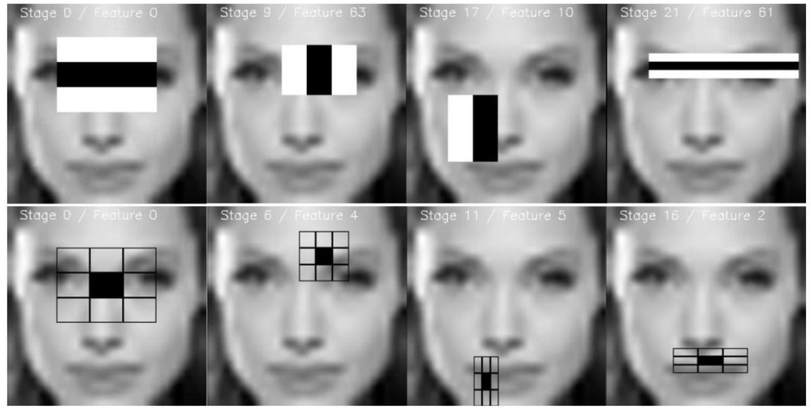


https://en.wikipedia.org/wiki/List_of_datasets_in_computer_vision_and_image_processing

Computer Vision Abridged History

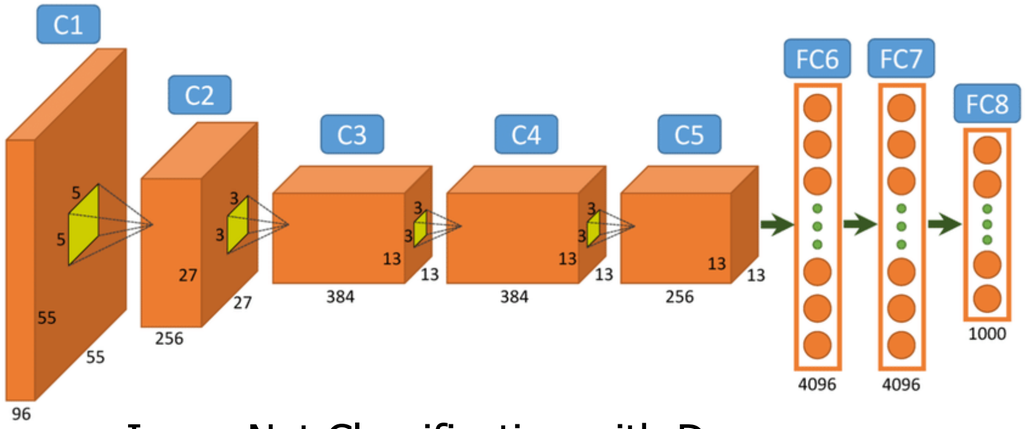
- 1960s - First computer vision projects (MIT summer project)
- 2000s - Emergence of "tasks" and benchmarking in computer vision
- 2000s - Shallow classifiers and feature engineering (e.g., Viola & Jones algorithm)



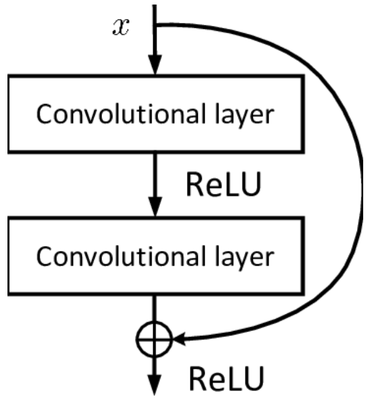


Computer Vision Abridged History

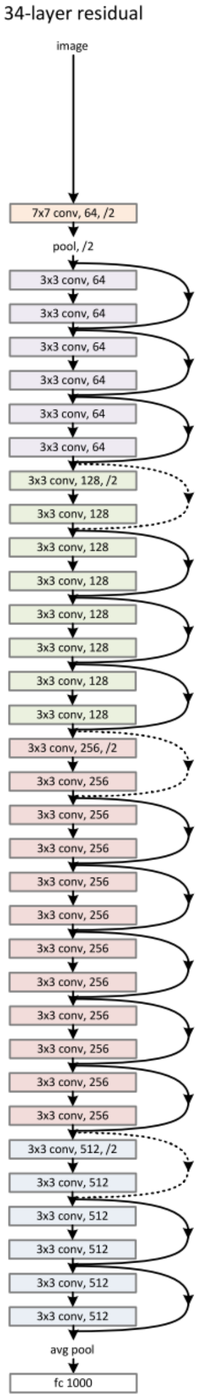
- 1960s - First computer vision projects (MIT summer project)
- 2000s - Emergence of "tasks" and benchmarking in computer vision
- 2000s - Shallow classifiers and feature engineering
- 2012 - Deep Learning revolution:
  - Success of Convolutional neural nets in ImageNet



ImageNet Classification with Deep Convolutional Neural Networks, 2012



Deep Residual Learning for Image Recognition, 2015



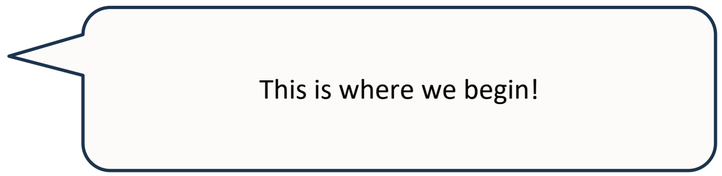
Computer Vision Abridged History

- 1960s - First computer vision projects (MIT summer project)
- 2000s - Emergence of “tasks” and benchmarking in computer vision
- 2000s - Shallow classifiers and feature engineering
- 2012 - Deep Learning revolution:
  - Success of Convolutional neural nets in ImageNet
  - Unification of architectures
  - Rise of image generation (VAEs, GANs, etc.)

Kingma et al. Auto-Encoding Variational Bayes, 2013
Goodfellow et al. Generative Adversarial Networks, 2014

Computer Vision Abridged History

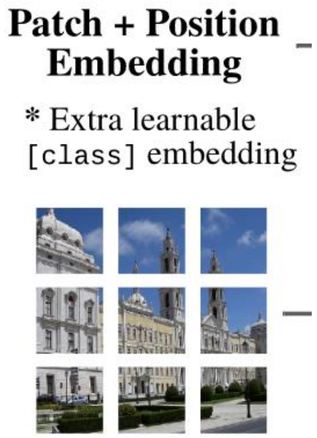
- 1960s - First computer vision projects (MIT summer project)
- 2000s - Emergence of “tasks” and benchmarking in computer vision
- 2000s - Shallow classifiers and feature engineering
- 2012 - Deep Learning revolution:
  - Success of Convolutional neural nets in ImageNet
  - Unification of architectures
  - Rise of image generation (VAEs, GANs, etc.)
- 2020s - Era of Vision Transformer
  - Stronger connection to language
  - Better generative models
  - Further unification of models and tasks



Let's Consider Images – How to Encode?

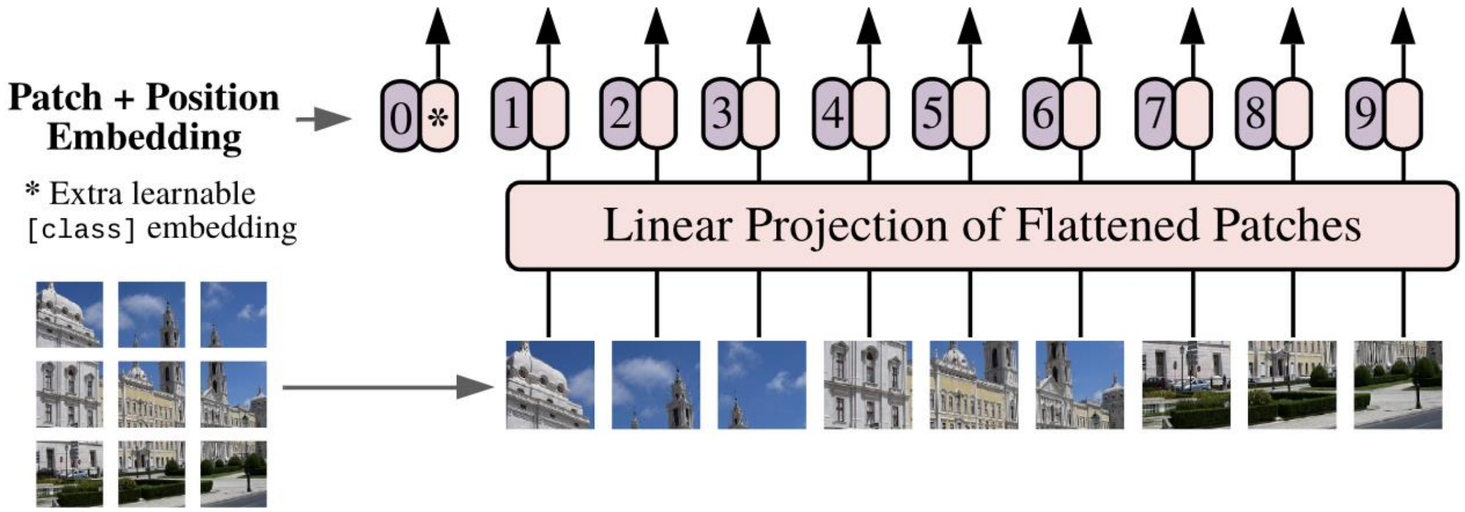
Transformers eats sequences !!!

Vision Transformers

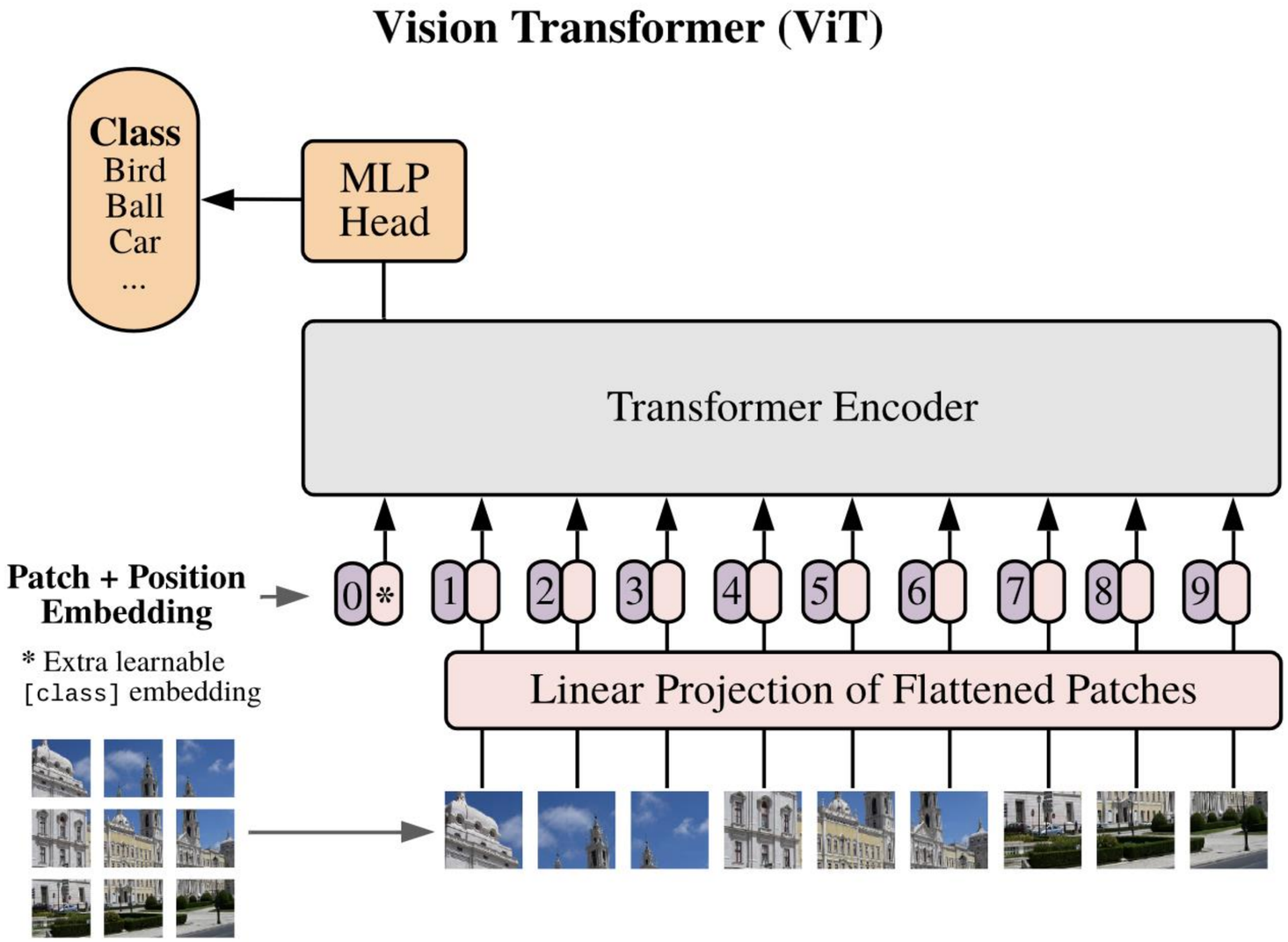


tokenize" the image by cutting it into patches of 16px², and treating each patch as a token, e.g. embedding it into input space

Vision Transformers



Vision Transformers



[An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, 2020]

Training Transformer LMs: Empirical Considerations

Pre-training Transformer LMs

- You have learned about the basics of pre-training Transformer language models.
- There is so much empirical knowledge/experiences that goes into training these models.
- Various empirical issues about:
  - Preparation/pre-processing data
  - Efficient training of models
  - …

C4: The Data

- C4: Colossal Clean Crawled Corpus
  - Web-extracted text
  - English language only
  - 750GB

Data set	Size
★ C4	745GB
C4, unfiltered	6.1TB

Play with the data: https://huggingface.co/datasets/allenai/c4/viewer/fr?row=5/

C4: The Data

Remove any:
- References to Javascript
- "Lorem ipsum" text — placeholder text commonly used to demonstrate the visual form of a document

Menu

Lemon

Introduction

The lemon, Citrus Limon (l.) Osbeck, is a species of small evergreen tree in the flowering plant family rutaceae.
The tree's ellipsoidal yellow fruit is used for culinary and non-culinary purposes throughout the world, primarily for its juice, which has both culinary and cleaning uses.
The juice of the lemon is about 5% to 6% citric acid, with a ph of around 2.2, giving it a sour taste.

Article

The origin of the lemon is [illegible]wn, though [illegible]
orange) and citron.

Please enable JavaScript to use our site.

Home
Products
Shipping
Contact
FAQ

Dried Lemons, $3.59/pound

Organic dried lemons from our farm in California.
Lemons are harvested and sun-dried for maximum flavor.
Good in soups and on popcorn.

The lemon, Citrus Limon (l.) Osbeck, is a species of small evergreen tree in the flowering plant family rutaceae.
The tree's ellipsoidal yellow fruit is used for culinary and non-culinary purposes throughout the world, primarily for its juice, which has both culinary and cleaning uses.
The juice of the lemon is about 5% to 6% citric acid, with a ph of around 2.2, giving it a sour taste.

Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Curabitur in tempus quam. In mollis et ante at consectetur.
Aliquam erat volutpat.
Donec at lacinia est.
Duis semper, magna tempor interdum suscipit, ante elit molestie urna, eget efficitur risus nunc ac elit.
Fusce quis blandit lectus.
Mauris at mauris a turpis tristique lacinia at nec ante.
Aenean in scelerisque tellus, a efficitur ipsum.
Integer justo enim, ornare vitae sem non, mollis fermentum lectus.
Mauris ultrices nisl at libero porta sodales in ac orci.

```
function Ball(r) {
  this.radius = r;
  this.area = pi * r ** 2;
  this.show = function(){
    drawCircle(r);
  }
}
```

Retain:
- Sentences with terminal punctuation mrks
- Pages with at least 5 sentences, sentences with at least 3 words

Pre-training Data: Experiment

- Takeaway:
  - Clean and compact data is better than large, but noisy data.
  - Pre-training on in-domain data helps.

Pre-training Data Duplicates

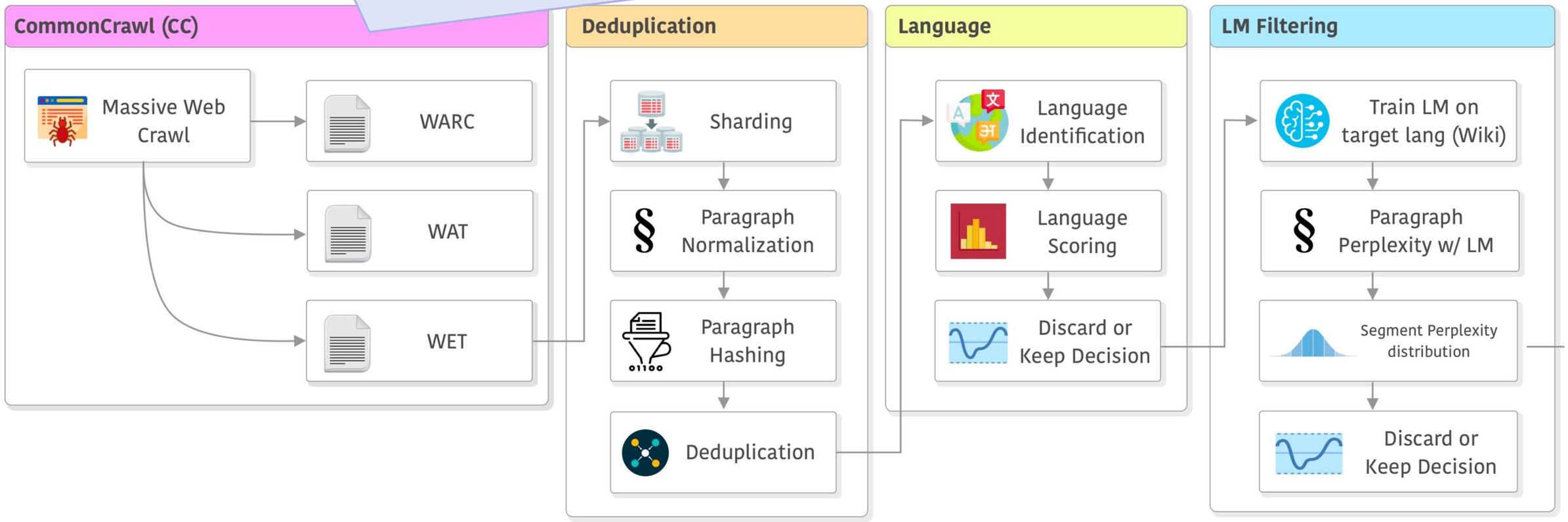
- There is a non-negligible number of duplicates in any pre-training data.

	% train examples with dup in train	% train examples with dup in valid	% valid with dup in train
C4	3.04%	1.59%	4.60%
RealNews	13.63%	1.25%	14.35%
LM1B	4.86%	0.07%	4.92%
Wiki40B	0.39%	0.26%	0.72%

Dataset	Example	Near-Duplicate Example
Wiki-40B	\n_START_ARTICLE_\nHum Award for Most Impactful Character \n_START_SECTION_\nWinners and nominees\n_START_PARAGRAPH_\nIn the list below, winners are listed first in the colored row, followed by the other nominees. [...]	\n_START_ARTICLE_\nHum Award for Best Actor in a Negative Role \n_START_SECTION_\nWinners and nominees\n_START_PARAGRAPH_\nIn the list below, winners are listed first in the colored row, followed by the other nominees. [...]
LM1B	I left for California in 1979 and tracked Cleveland 's changes on trips back to visit my sisters .	I left for California in 1979 , and tracked Cleveland 's changes on trips back to visit my sisters .
C4	Affordable and convenient holiday flights take off from your departure country, "Canada". From May 2019 to October 2019, Condor flights to your dream destination will be roughly 6 a week! Book your Halifax (YHZ) - Basel (BSL) flight now, and look forward to your "Switzerland" destination!	Affordable and convenient holiday flights take off from your departure country, "USA". From April 2019 to October 2019, Condor flights to your dream destination will be roughly 7 a week! Book your Maui Kahului (OGG) - Dubrovnik (DBV) flight now, and look forward to your "Croatia" destination!

LLaMA's Data Pipeline

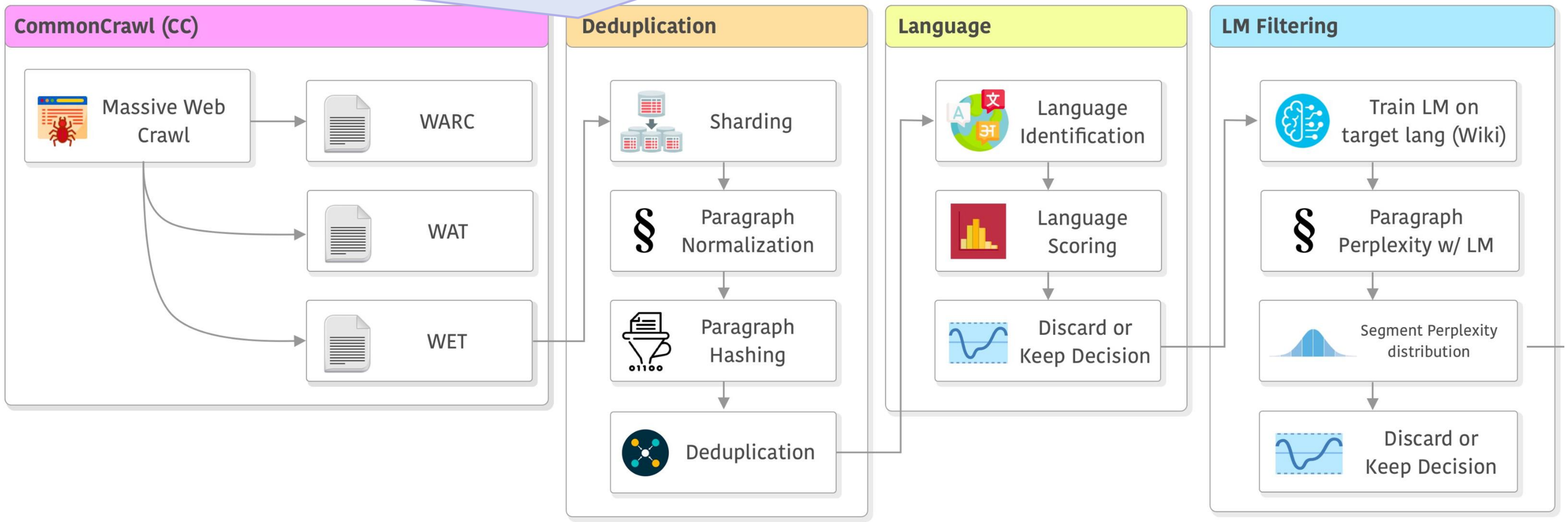
Starts with the massive crawled data by CommonCrawl.
The WET format that contains textual information.
WARC is raw, WAT is metadata, WET is text+some metadata.



https://twitter.com/tarantulae/status/1650170087708454913?t=ncWWY0FI0tYC_dYLs76r5g

LLaMA's Data Pipeline

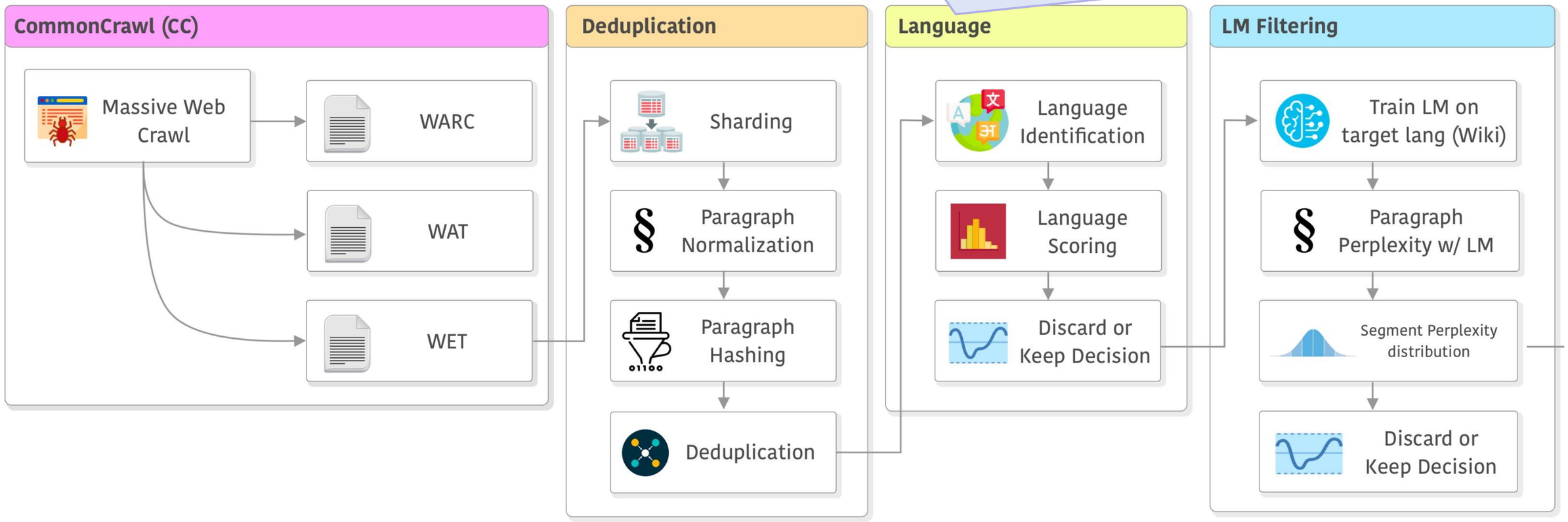
Shard WET content into shards of 5GB each (one CC snapshot can have 30TB).
Then you normalize paragraphs (lowercasing, numbers as placeholders, etc), compute per-paragraph hashes and then duplicate them.



https://twitter.com/tarantulae/status/1650170087708454913?t=ncWWY0FI0tYC_dYLs76r5g

LLaMA's Data Pipeline

Perform language identification and decide whether to keep or discard languages.
The order of when you do this in the pipeline can impact the language discrimination quality.

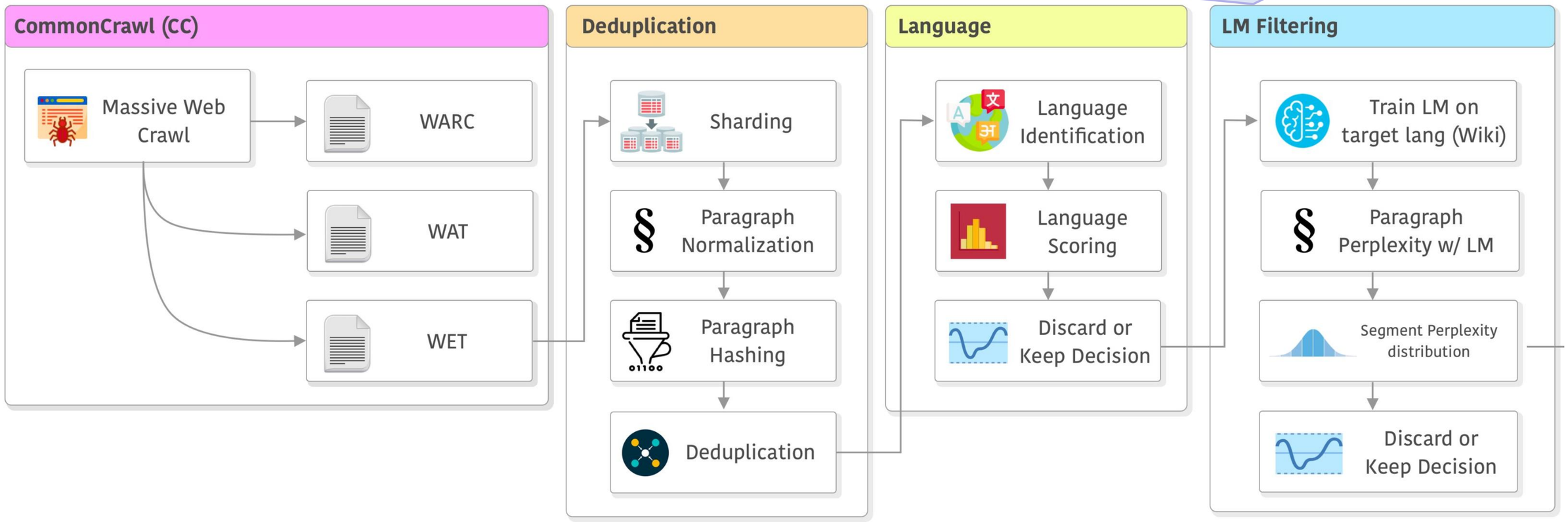


https://twitter.com/tarantulae/status/1650170087708454913?t=ncWWY0FI0tYC_dYLs76r5g

LLaMA's Data Pipeline

Do further quality filtering: Train a simple LM (n-gram) on target languages using Wikipedia, then compute per-paragraph perplexity on the rest of the data:

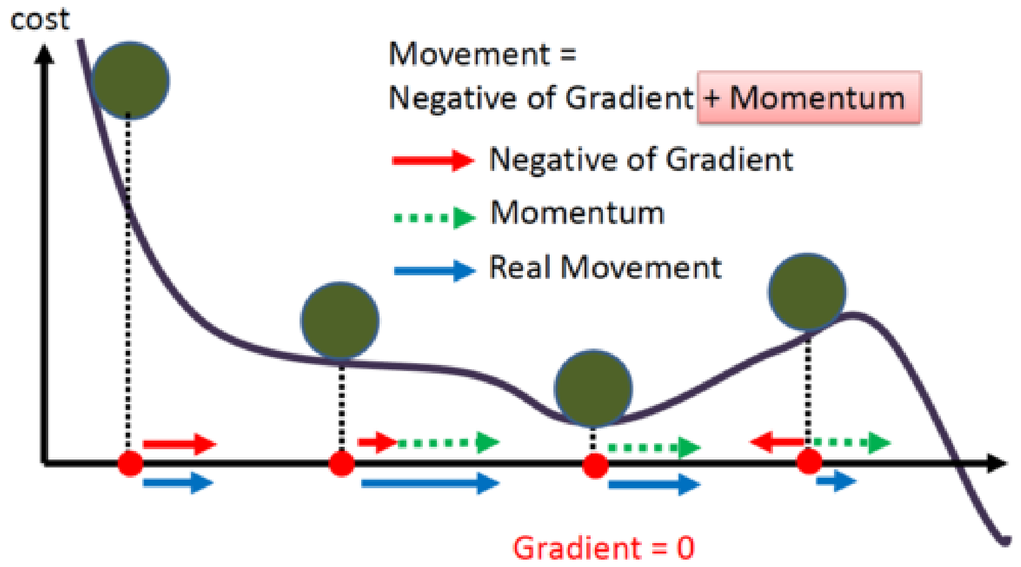
- Very high PPL: Very different than Wiki and likely low-quality → Drop
- Very low PPL: Very similar or near duplicates to Wiki → Drop



CCNet: Extracting High Quality Monolingual Datasets from Web Crawl Data, 2019

Optimizers

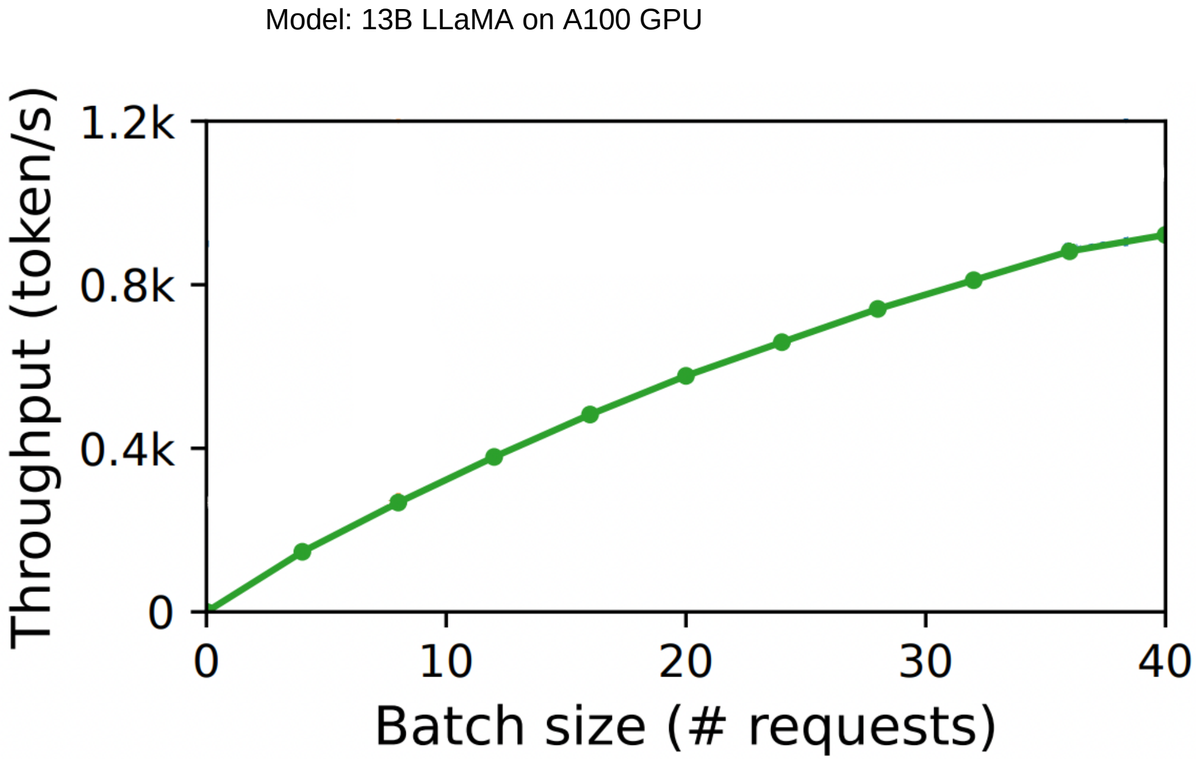
- Most modern models use "AdamW" optimizer (not vanilla Gradient Descent).
  - Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order "momentums".
  - "W" because it decouples "weight decay" from "learning rate". (Details out of scope for us. See the cited paper.)



https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html
https://pytorch.org/docs/stable/generated/torch.optim.Adam.html
[Decoupled Weight Decay Regularization, 2017]

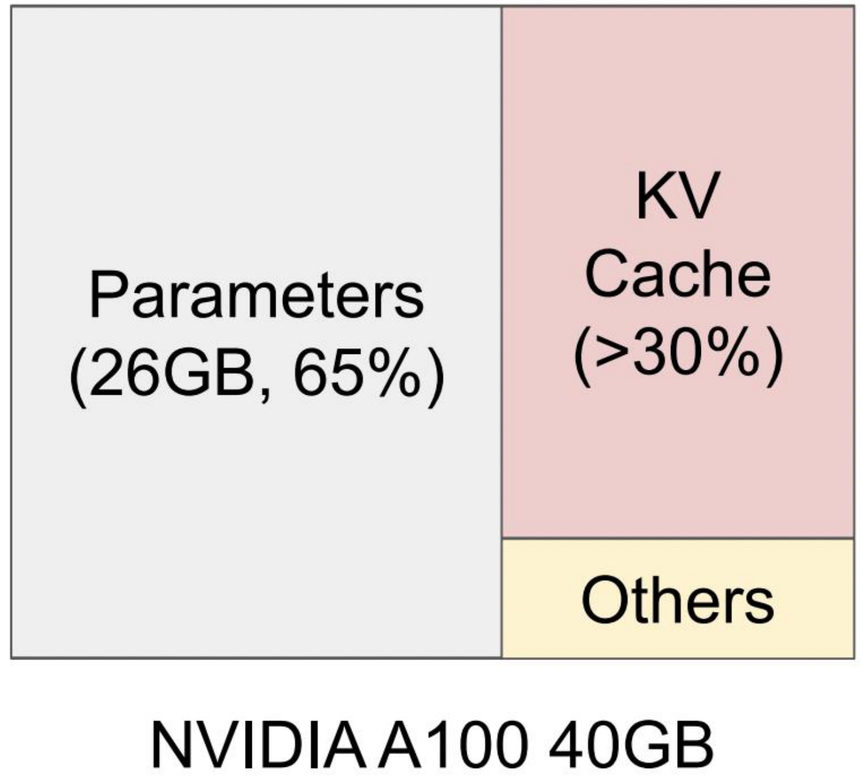
Batching Data

- Previously we talked about the importance of batching data
- GPUs are faster at Tensor operations and hence, we want to do batch processing
- The lager batch of data, the faster they get processed.
- Alas, the speedup is often sub-linear (e.g., 2x larger batch leads to less than 2x speedup).



The Memory Usage

- Here is the memory usage of an NVIDIA A100 when serving (i.e., no training)
  - Model: 13B LLaMA
  - Batch size of 10



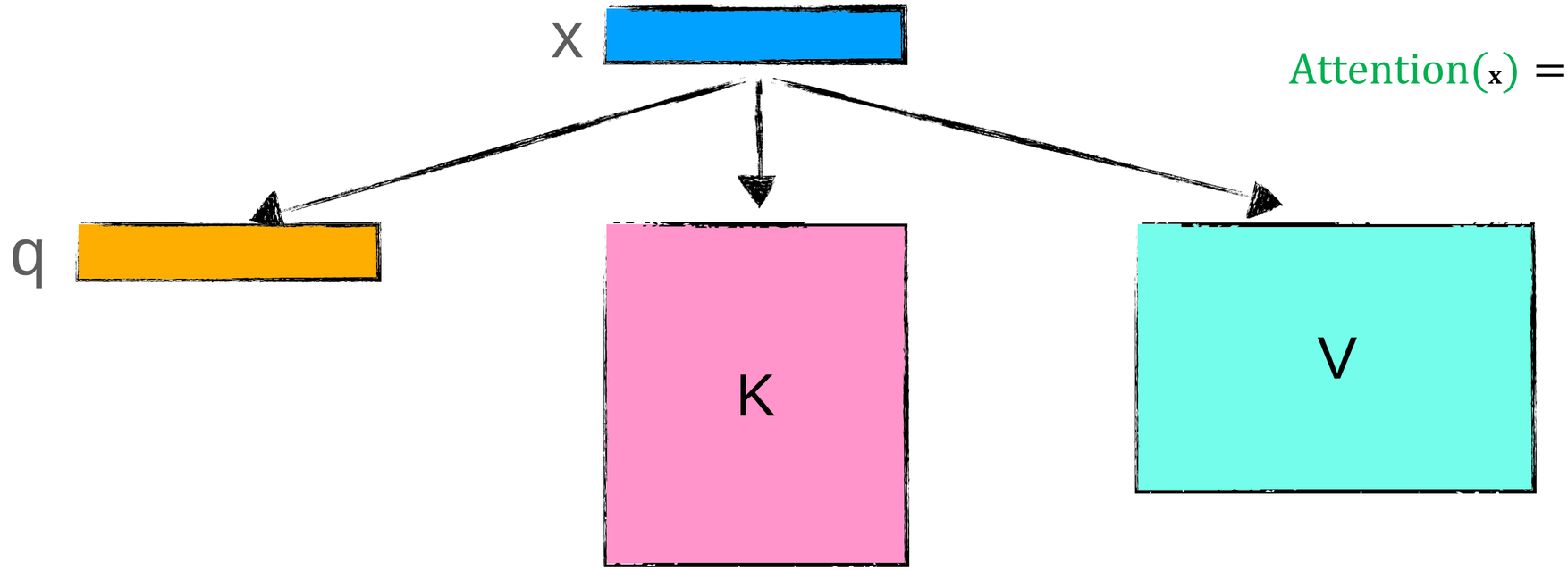
- ~65% of your GPU memory is the model parameters that never change
- ~32% of your memory are KV tensors that change for each input.
  - This KV cache will increase for larger batch sizes.
  - Managing this part of the memory is key for efficient training.

An important efficiency consideration about decoding!

Making decoding more efficient

$$Q = \mathbf{W}^q \mathbf{x}$$
$$K = \mathbf{W}^k \mathbf{x}$$
$$V = \mathbf{W}^v \mathbf{x}$$

$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$

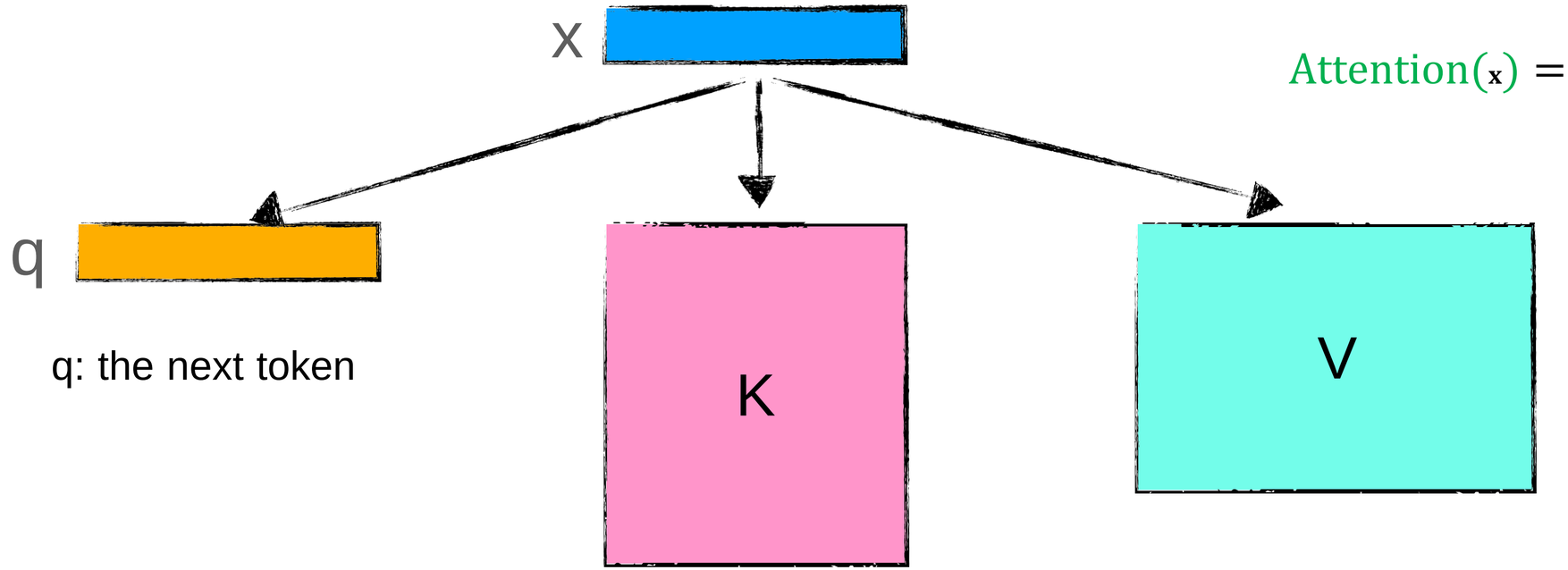


[Slide credit: Arman Cohan]

Making decoding more efficient

$$Q = \mathbf{W}^q\mathbf{x}$$
$$K = \mathbf{W}^k\mathbf{x}$$
$$V = \mathbf{W}^v\mathbf{x}$$

$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$



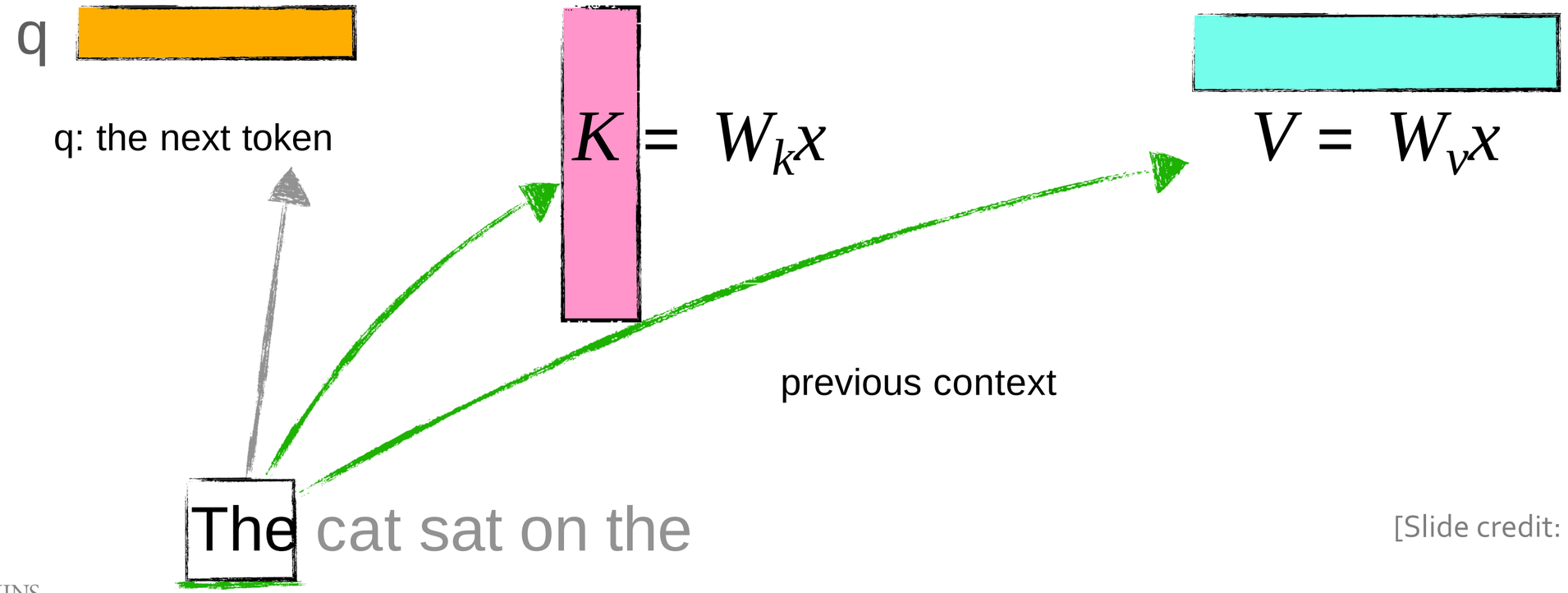
q: the next token

previous context

[Slide credit: Arman Cohan]

Making decoding more efficient

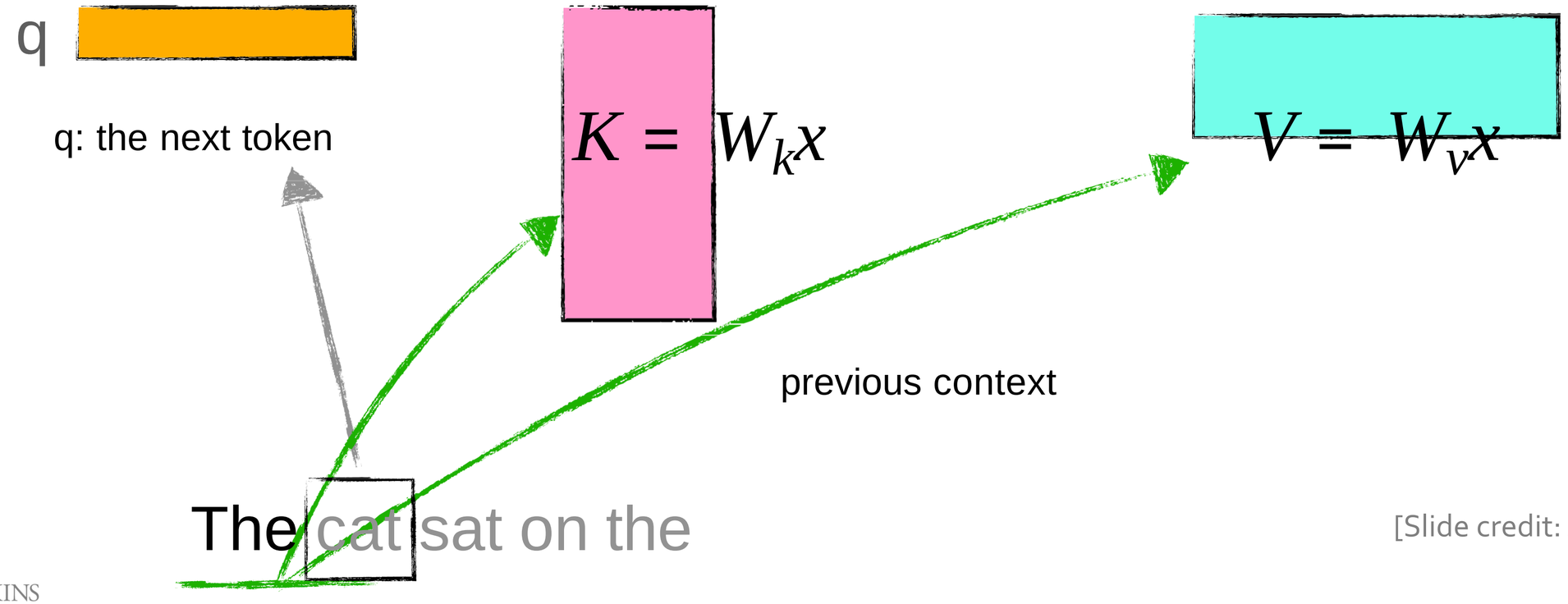
$$Q = \mathbf{W}^q\mathbf{x}$$
$$K = \mathbf{W}^k\mathbf{x}$$
$$V = \mathbf{W}^v\mathbf{x}$$
$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$



[Slide credit: Arman Cohan]

Making decoding more efficient

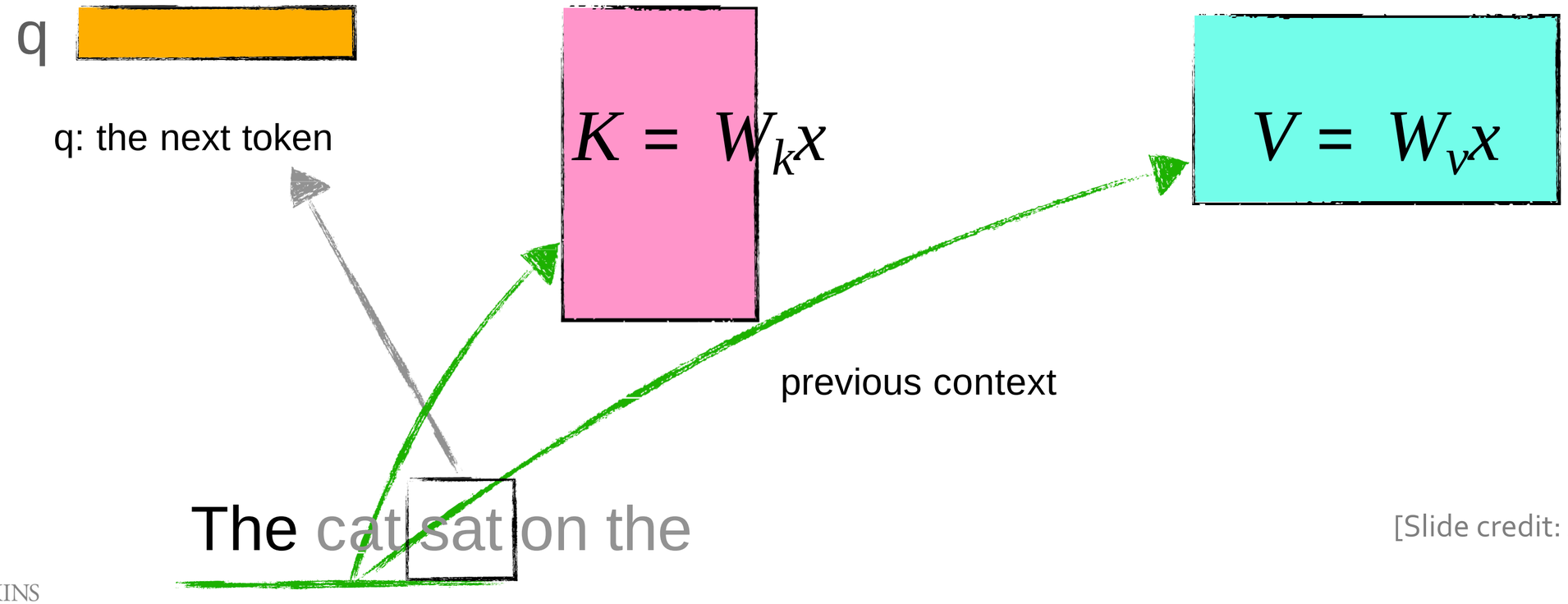
$$Q = \mathbf{W}^q \mathbf{x}$$
$$K = \mathbf{W}^k \mathbf{x}$$
$$V = \mathbf{W}^v \mathbf{x}$$
$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$



[Slide credit: Arman Cohan]

Making decoding more efficient

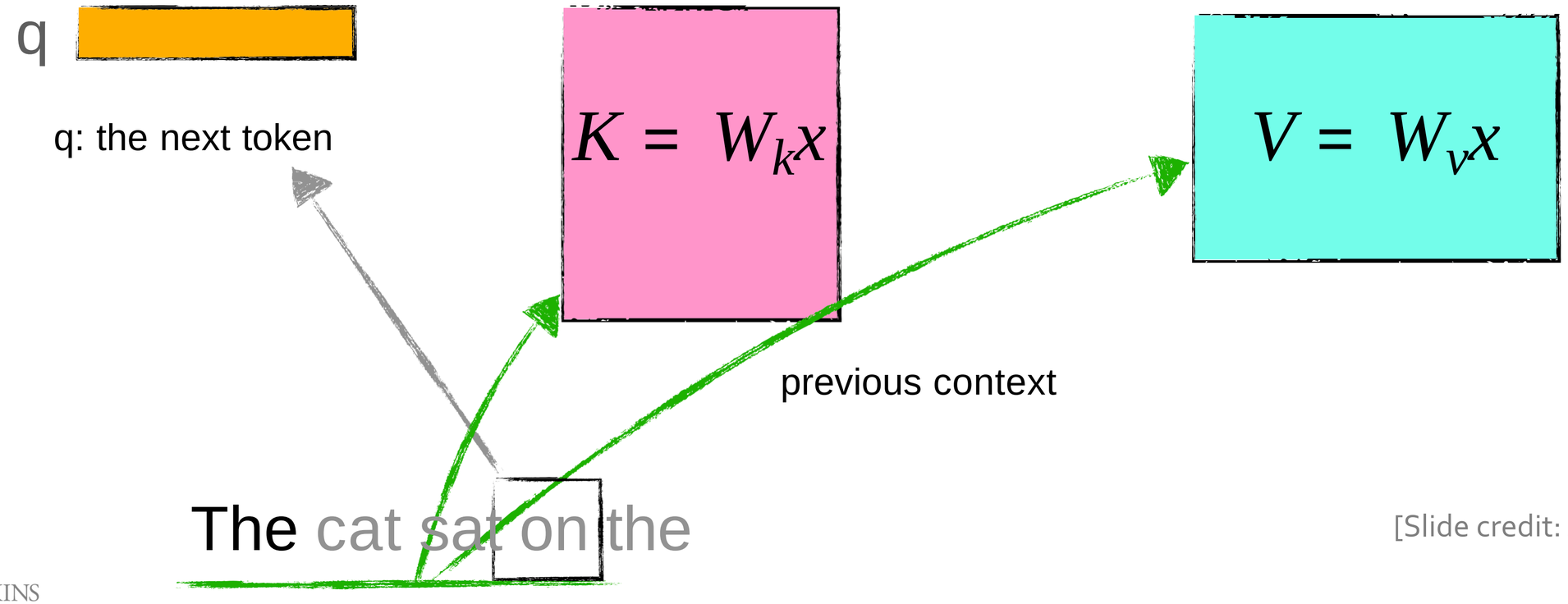
$$Q = \mathbf{W}^q \mathbf{x}$$
$$K = \mathbf{W}^k \mathbf{x}$$
$$V = \mathbf{W}^v \mathbf{x}$$
$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$



[Slide credit: Arman Cohan]

Making decoding more efficient

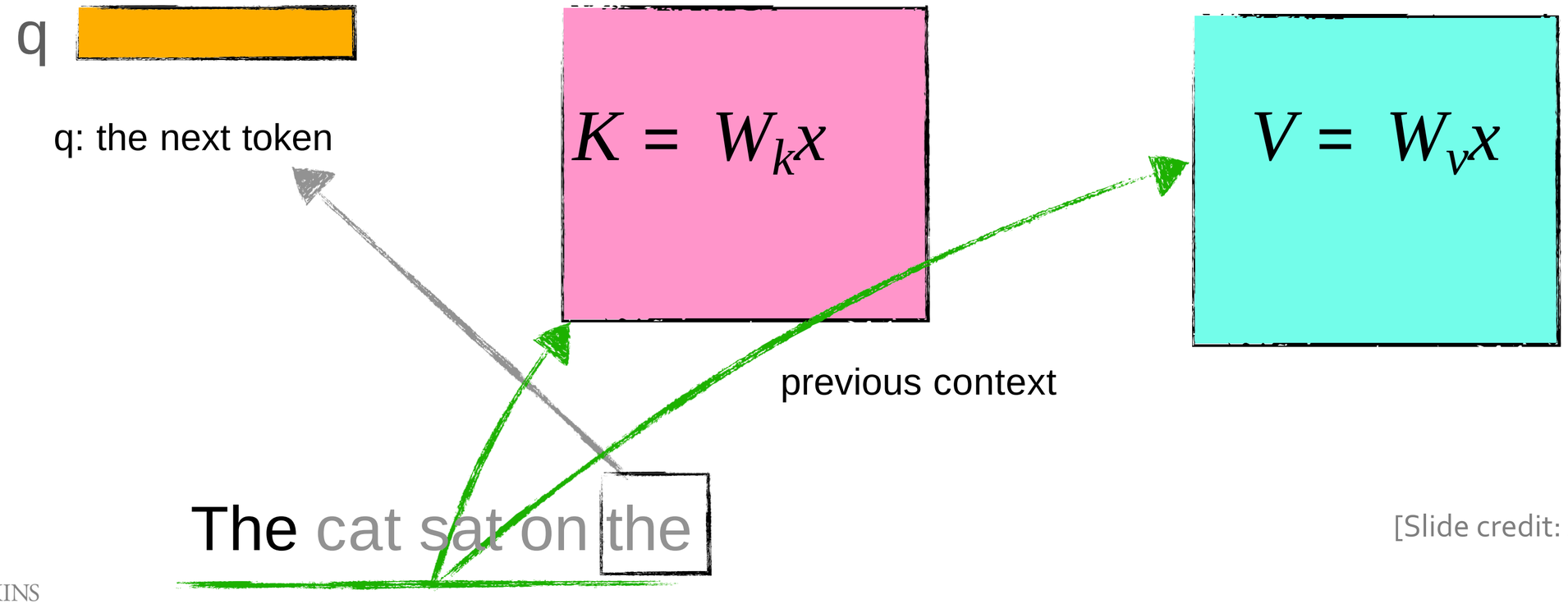
$$Q = \mathbf{W}^q \mathbf{x}$$
$$K = \mathbf{W}^k \mathbf{x}$$
$$V = \mathbf{W}^v \mathbf{x}$$
$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$



[Slide credit: Arman Cohan]

Making decoding more efficient

$$Q = \mathbf{W}^q \mathbf{x}$$
$$K = \mathbf{W}^k \mathbf{x}$$
$$V = \mathbf{W}^v \mathbf{x}$$
$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$



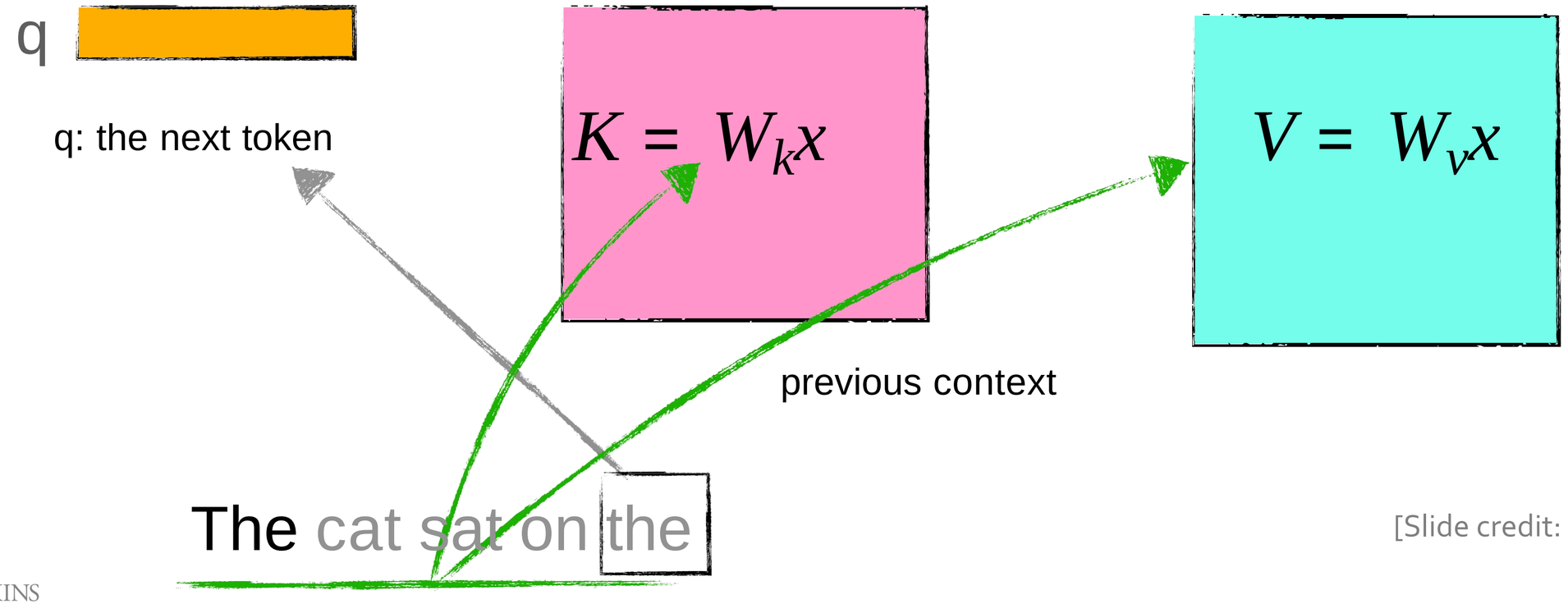
[Slide credit: Arman Cohan]

Making decoding more efficient

$$Q = \mathbf{W}^q \mathbf{x}$$
$$K = \mathbf{W}^k \mathbf{x}$$
$$V = \mathbf{W}^v \mathbf{x}$$

$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$

- We are computing the Keys and Values many times!
  - Let's reduce redundancy! 🥰



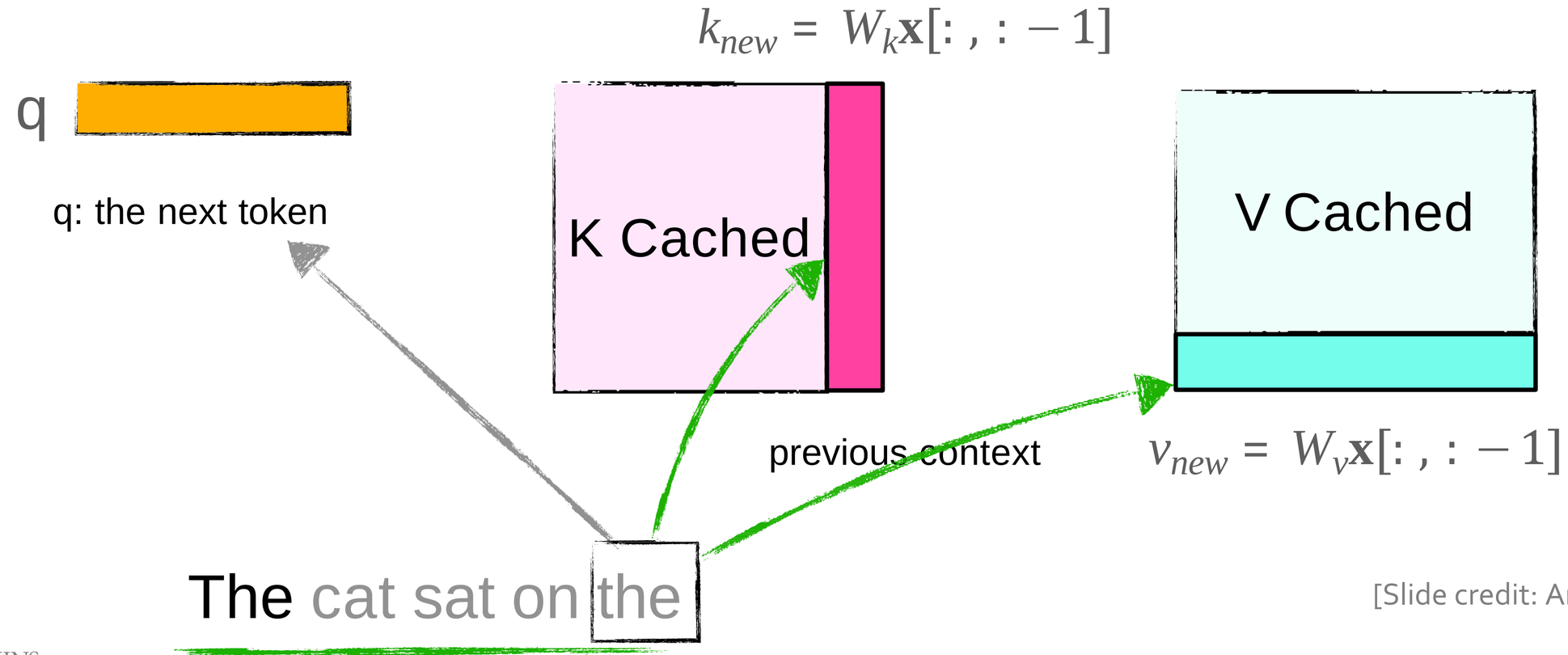
[Slide credit: Arman Cohan]

Making decoding more efficient

$$Q = \mathbf{W}^q\mathbf{x}$$
$$K = \mathbf{W}^k\mathbf{x}$$
$$V = \mathbf{W}^v\mathbf{x}$$

$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$

- We are computing the Keys and Values many times!
  - Let's reduce redundancy! 🥰



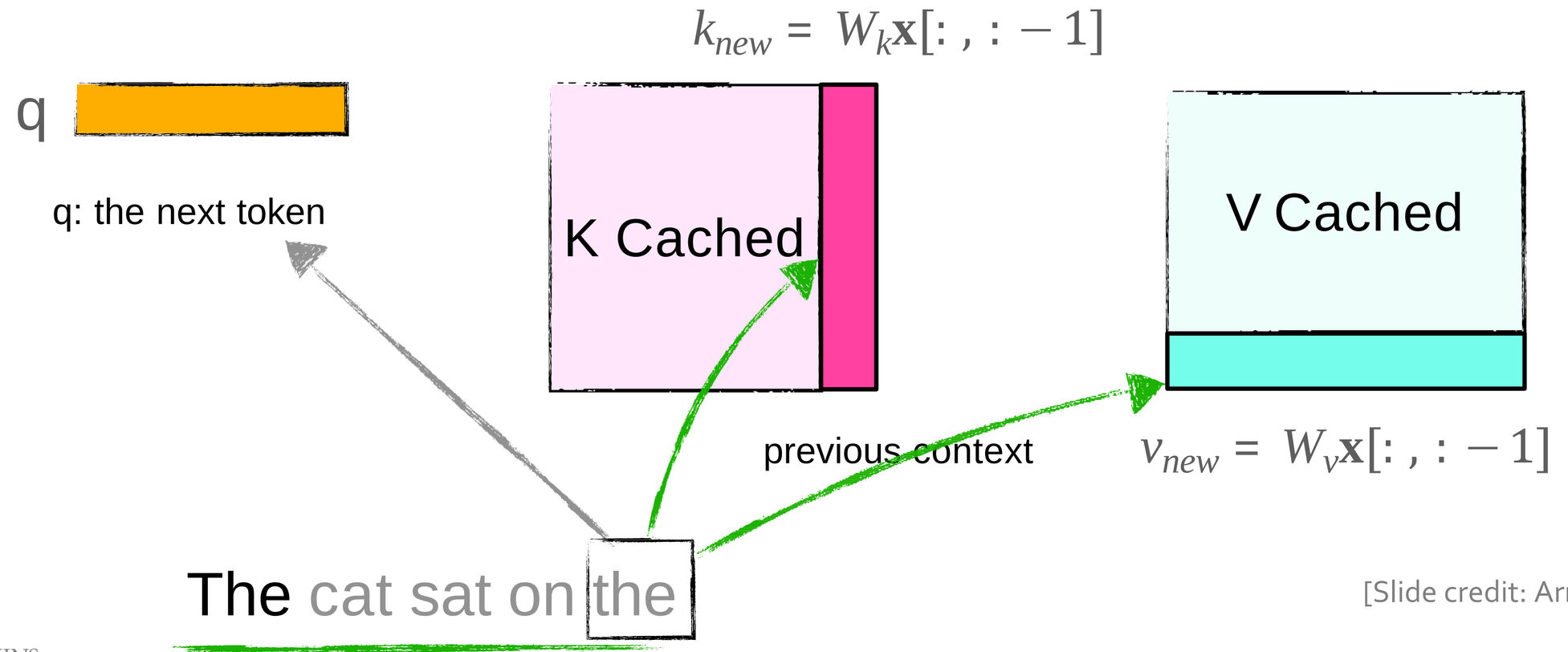
[Slide credit: Arman Cohan]

Making decoding more efficient

$$Q = \mathbf{W}^q\mathbf{x}$$
$$K = \mathbf{W}^k\mathbf{x}$$
$$V = \mathbf{W}^v\mathbf{x}$$

$$\text{Attention}(\mathbf{x}) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}\right)V$$

- **Question:** How much memory does this K, V cache require?



[Slide credit: Arman Cohan]

Scaling model size

- LM are getting larger and more expensive

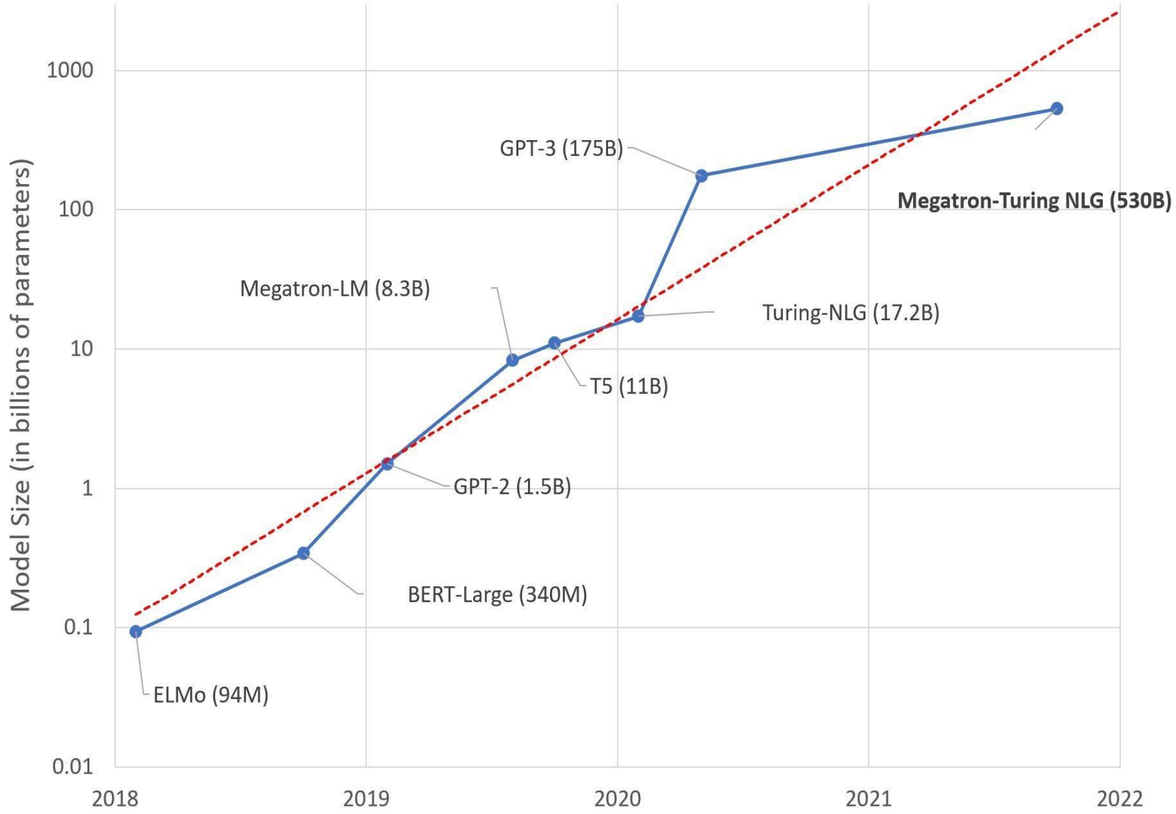
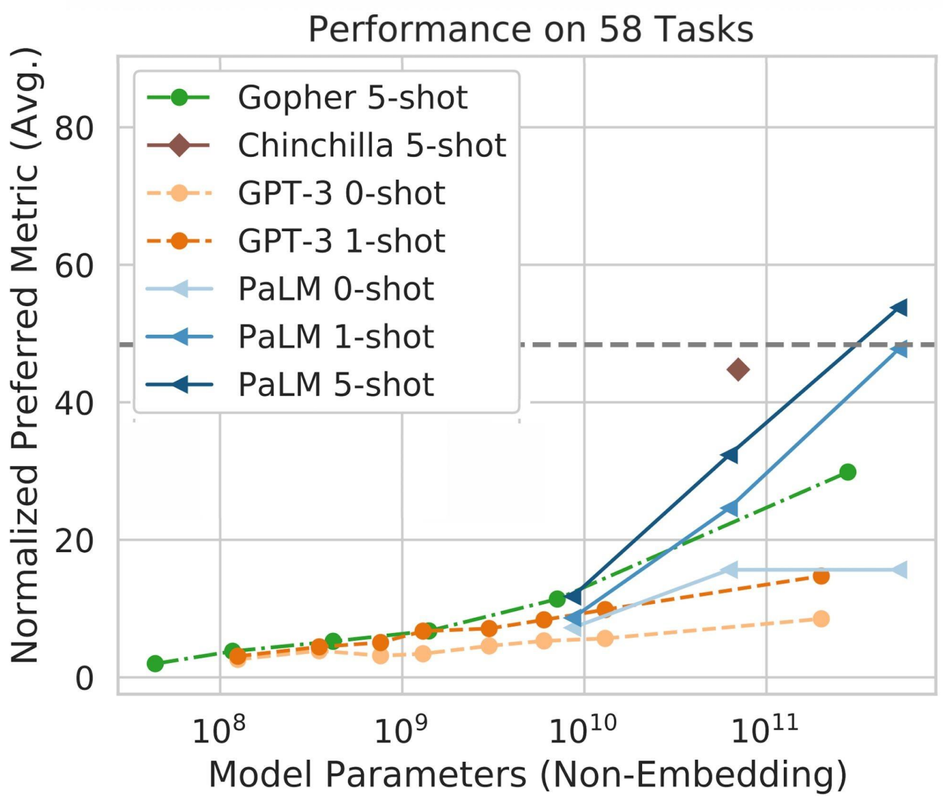
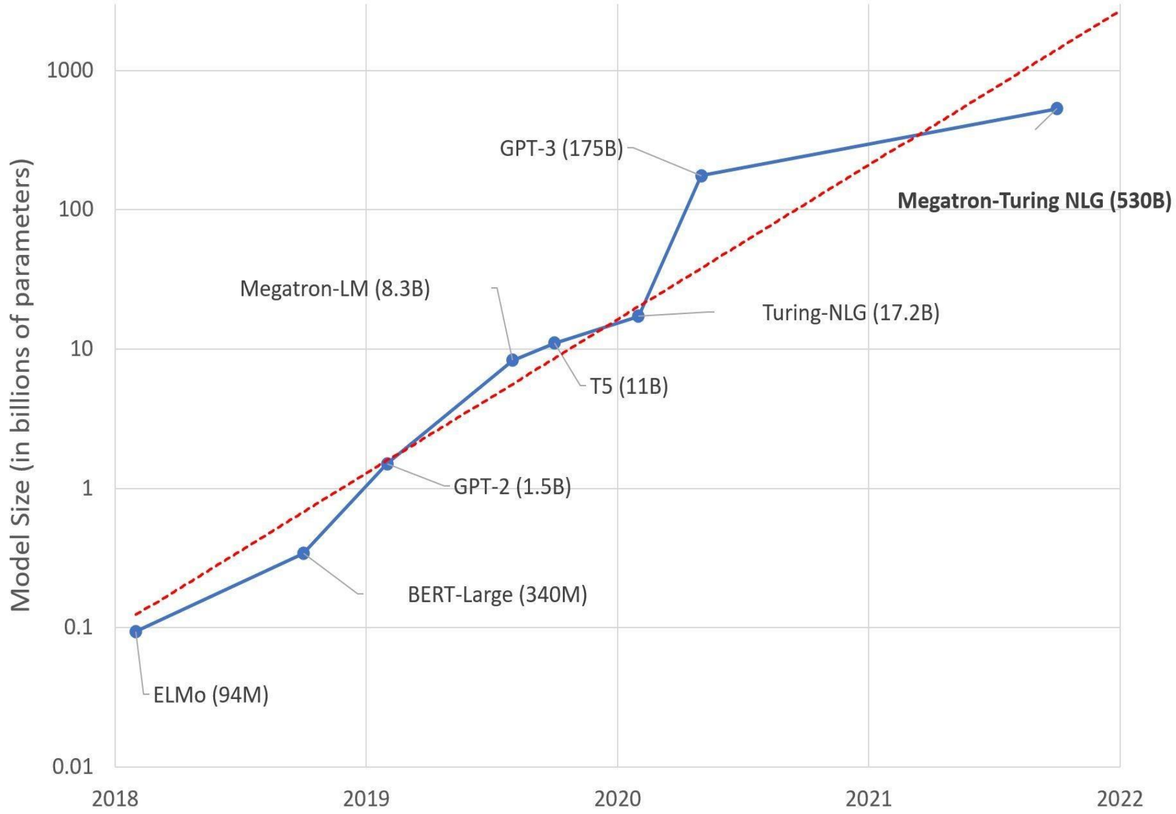


Photo credit: Microsoft Research Blog, Alvi et. al., 2021

Model Size vs. Accuracy

Photo credit: PaLM, Chowdhery et. al., 2022





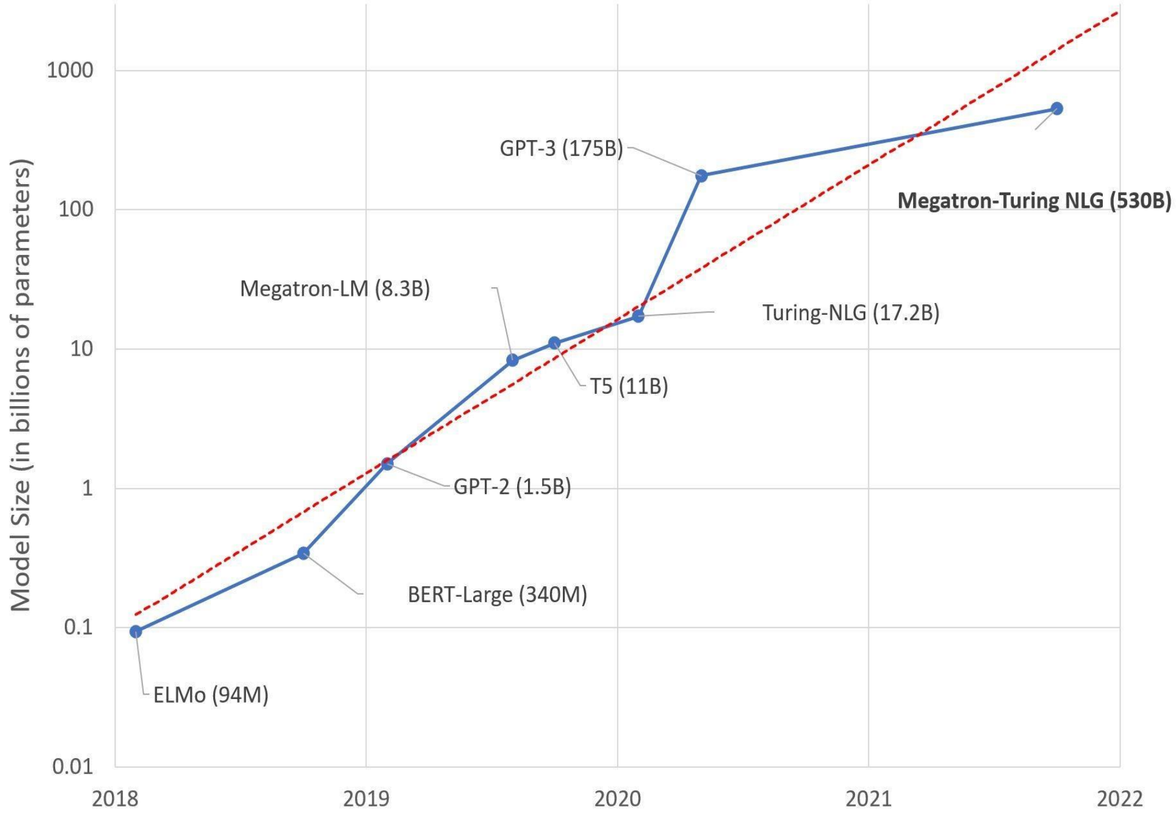
Larger LMs ⇒ better zero/few-shot performance

What is “Scaling”?

- “scaling means **larger model size**”
  - But model parameters may be under-utilized.
- “scaling means **more compute**”
  - But computation may be unnecessarily wasted.
- “scaling means **more data**”
  - But more data might not necessarily contain more information (e.g., duplications)
- Scale means all the above: *effective compression of information*
  - Requires model capacity
  - Requires compute
  - Requires large, rich data

Constraints of Real World

- Even massive companies have their own constraints.
- Examples of constraints:
  - The total amount of data
  - The total computing budget.
  - Time
  - ….
- **Given a set of constraints, how do you choose which LM to train?**
  - Note, trial and error is wasteful.



Scaling Laws

- **Hypothesis:** there are fundamental principles that govern effective scaling
- **Importance:** understanding these "laws" would allow us to find optimal models for a given data/compute budget.
- Think of Newton's laws
  - Provide the basis for understanding and analyzing the motion of objects in the physical world
  - Can be used to calculate the trajectory of a rocket, the speed of a car, or the motion of a planet.

Computation Cost of Models

How do you compute computational cost of
a single-layer NN with one matrix multiplication?

FLOPS

- Floating point operations per second (FLOPS, flops or flop/s)
- Each FLOP can represent an addition, subtraction, multiplication, or division of floating-point numbers,
- The total FLOP of a model (e.g., Transformer) provides a basic approximation of computational costs associated with that model.

FLOPS: Matrix Multiplication

- Matrix-vector multiplication are common in Self-Attention (e.g., QKV projection)
  - Requires $2mn$ (2 x matrix size) operations for multiplying $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^{n}$
  - (2 because 1 for multiplication, 1 for addition)

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 + \cdots + A_{1n}x_n \\ A_{21}x_1 + A_{22}x_2 + \cdots + A_{2n}x_n \\ \vdots \\ A_{m1}x_1 + A_{m2}x_2 + \cdots + A_{mn}x_n \end{bmatrix}$$

FLOPS: Matrix Multiplication

- Matrix-vector multiplication are common in Self-Attention (e.g., QKV projection)
  - Requires $2mn$ (2 x matrix size) operations for multiplying $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^{n}$
  - (2 because 1 for multiplication, 1 for addition)

- For multiplying $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$, one needs $2mnp$ operations.
  - Again, 2 because of 1 for multiplication, 1 for addition

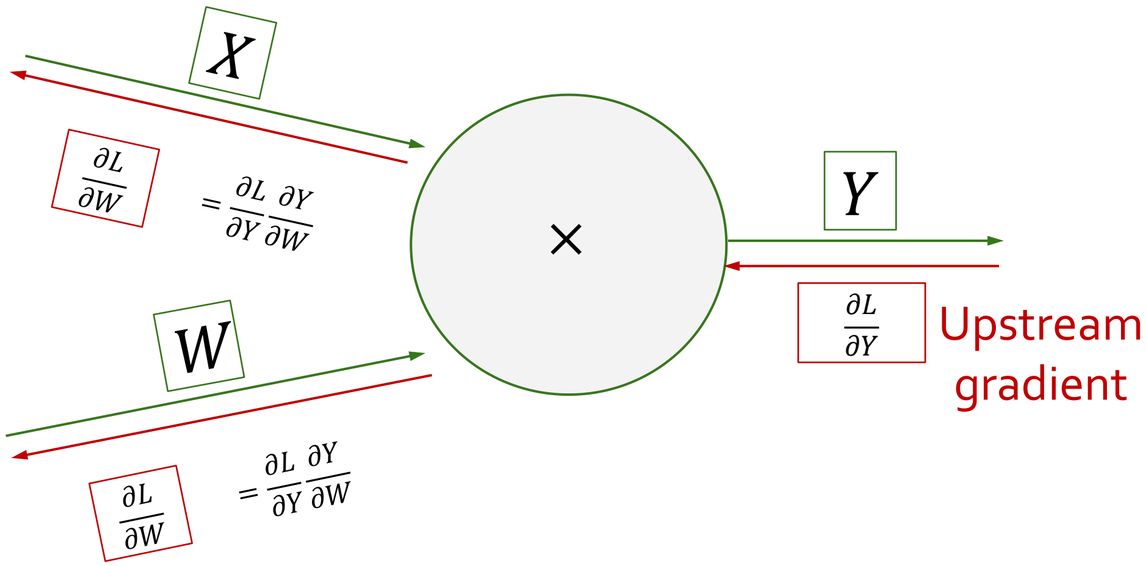
- Now this is just forward propagation in Backprop. What about the **backward** step?

FLOPS: Matrix Multiplication: Backward

- Backward pass needs to calculate the derivative of loss with respect to each hidden state and for each parameter

We also need $\frac{\partial L}{\partial X}$ to continue to pass gradient to the previous layers.

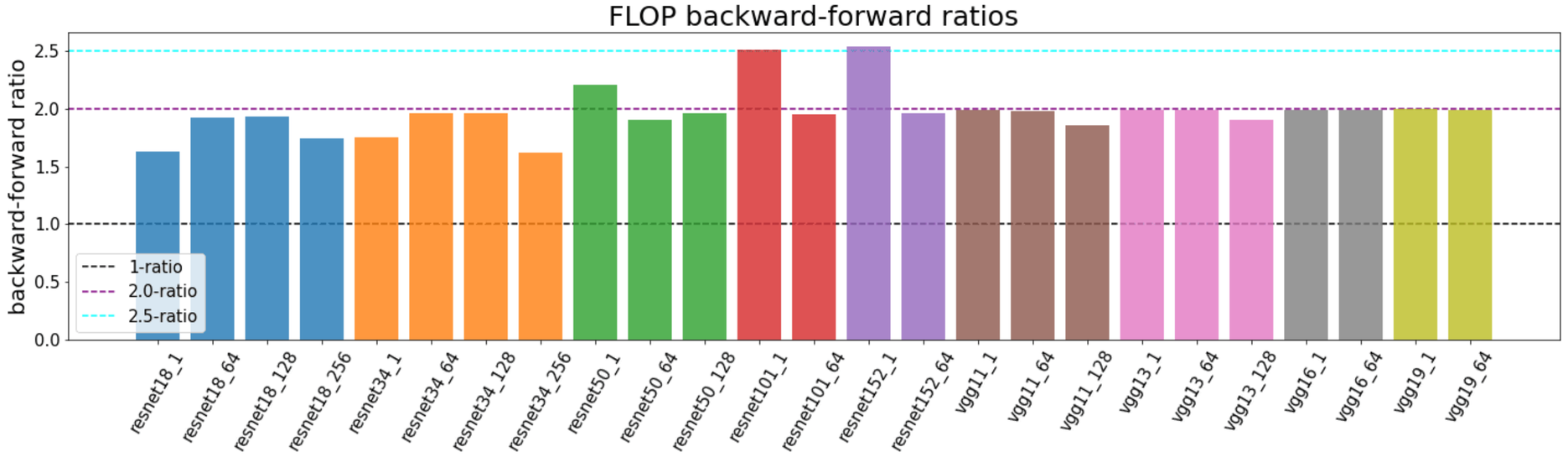
One matrix multiplication for $\frac{\partial L}{\partial W}$



FLOPs for backward pass is roughly twice of forward pass.

FLOPS: Matrix Multiplication: Backward

- FLOPs for backward pass is **roughly twice** of forward pass.
- Note that, this ratio depends on various parameters (architecture, batch size, et).



What's the backward-forward FLOP ratio for Neural Networks? LessWrong, 2021
https://www.lesswrong.com/posts/fnjKpBoWJXcSDwhZk/what-s-the-backward-forward-flop-ratio-for-neural-networks

FLOPS: Matrix Multiplication: Altogether

- Multiplying an input by a weight matrix requires 2x matrix size FLOPS.
- FLOPs for backward pass is **roughly twice** of forward pass.

Training FLOPs for multiplying by a matrix W =
6 x (batch size) x (size of W)

Computing the computational cost of Transformer

Transformer FLOPs: The Quick Estimate

- The Weight FLOPs Assumption
  - The FLOPs that matter the most are weight FLOPs, that is ones performed when intermediate states are multiplied by weight matrices.

  - The weight FLOPs are the majority of Transformer FLOPs
  - We can ignore FLOPs for
    Bias vector addition
    layer normalization
    residual connections
    non-linearities
    Softmax

Transformer FLOPs: The Quick Estimate

- Let N be number of parameters (the sum of size of all matrices)
- Let D be the number of tokens in pre-training dataset.
- **Forward pass:**
  - FLOPs for forward pass on a single token is roughly 2N
  - FLOPs for forward pass for the entire dataset is roughly 2ND
- **Backward pass:**
  - FLOPs for backward pass is roughly twice of forward pass
  - FLOPs for backward pass for the entire dataset is roughly 4ND
- What is the total?

Transformer FLOPs: The Quick Estimate

- Let N be number of parameters (the sum of size of all matrices)
- Let D be the number of tokens in pre-training dataset.
- The total cost of pre-training on this dataset is:

$$C \sim 6ND$$

- You can already see how this relates to our constraints:
  - If you have a fixed compute budget C, increasing D means decreasing N (and vice versa).

Transformer Parameter Count

- One can show that:

Non-embedding params

$$N = 12 \times n_{layers} \times d_{model}^2 + (n_{vocab} + n_{pos}) \times d_{model}$$

Embedding params

- Assuming:
  - the size of MLP hidden layer to be $4.d_{\text{model}}$
  - $n_{\text{heads}}.d_{\text{hidden}} = d_{\text{model}}$

- You will prove this in homework assignment! 😊

Most Transformer LMs make these design assumptions.

Transformer Parameter Count

- One can show that:

Non-embedding params

$$N = 12 \times n_{layers} \times d_{model}^2 + (n_{vocab} + n_{pos}) \times d_{model}$$

Embedding params

n_{layer}	d_{model}	Parameters (N)
4	512	13M
6	768	42M
10	1280	197M
16	2048	810M
24	3072	2.7B
40	5120	13B
64	8192	52B

Table from: A General Language Assistant as a Laboratory for Alignment, 2021

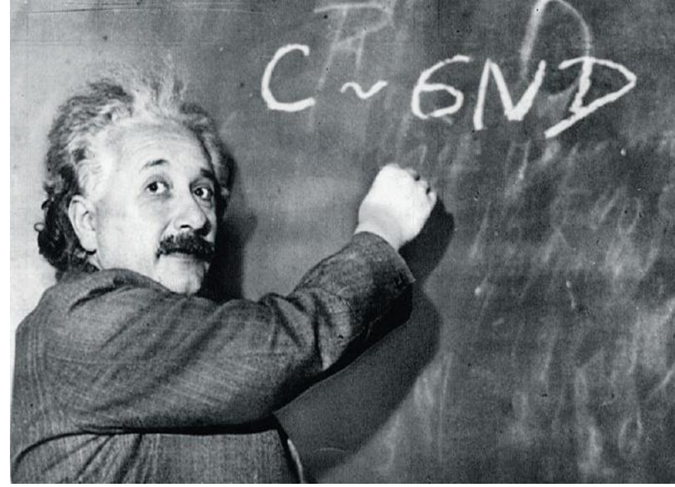
For example, see the models in the following table:

$$N_{\text{non-embedding}} = 12 \times 64 \times 8192^2 = 51.5B$$

Vocab size = 65536
Positional emb size = ?

$$N_{\text{embedding}} = (65536 + ?) \times 8192 = 0.5\text{B} + ?$$

Transformer Parameter Count



- Given the pre-training data with 400B tokens.

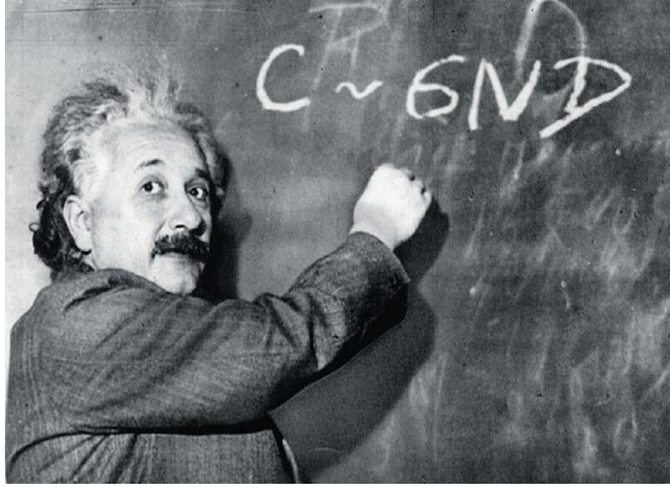
n_{layer}	d_{model}	Parameters (N)	Training FLOPs
4	512	13M	3.0e19
6	768	42M	1.0e20
10	1280	197M	4.7e20
16	2048	810M	1.9e21
24	3072	2.7B	6.5e21
40	5120	13B	3.0e22
64	8192	52B	1.2e23

Table from: A General Language Assistant as a Laboratory for Alignment, 2021

Training cost (FLOPs):

$$C \approx 6ND$$
$$= 6 \times (400 \times 10^9) \times (52 \times 10^9)$$
$$= 1.24 \times 10^{23}$$

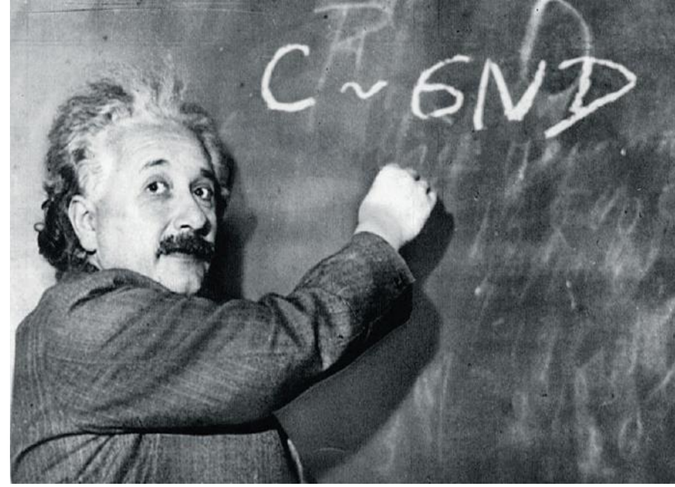
Estimating training time



- This is a very practical question in real world.
- We will use our formula earlier to estimate training time.
- Consider HyperCLOVA, an 82B parameter model that was pre-trained on 150B tokens, using a cluster of 1024 A100 GPUs.

Intensive Study on HyperCLOVA: Billions-scale Korean Generative Pretrained Transformers, 2021 https://arxiv.org/pdf/2109.04650.pdf

Estimating training time



- Consider HyperCLOVA, an 82B parameter model that was pre-trained on 150B tokens, using a cluster of 1024 A100 GPUs.
- Training cost (FLOPs):

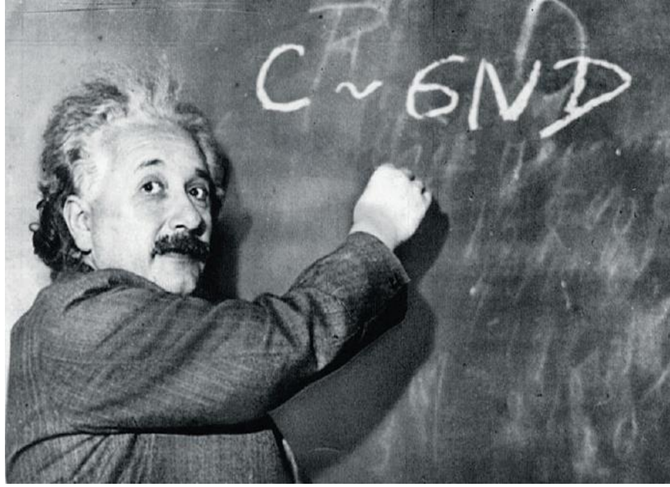
$$C \approx 6ND$$
$$= 6 \times (150 \times 10^9) \times (82 \times 10^9) = 7.3 \times 10^{22}$$

- The peak throughput of A100 GPUs if 312 teraFLOPS or 3.12×10^{14}.
- **How long would this take?**

$$\text{Duration} = \frac{\text{model compute cost}}{\text{cluster throughput}} = \frac{7.3 \times 10^{22}}{3.12 \times 10^{14} \times 1024} = 2.7 \text{ days}$$

Intensive Study on HyperCLOVA: Billions-scale Korean Generative Pretrained Transformers, 2021 https://arxiv.org/pdf/2109.04650.pdf

Estimating training time



- **How long would this take?**

$$\text{Duration} = \frac{\text{model compute cost}}{\text{cluster throughput}} = \frac{7.3 \times 10^{22}}{3.12 \times 10^{14} \times 1024} = 2.7 \text{ days}$$

- According to the white paper, training took 13.4 days. Our estimate is 5 times off (why?), but we did get the order of magnitude right! 🙌

Intensive Study on HyperCLOVA: Billions-scale Korean Generative Pretrained Transformers, 2021 https://arxiv.org/pdf/2109.04650.pdf

Factors We Did Not Consider

- Note that these estimates can be slightly off in practice
  - Theoretical peak throughput is not achievable with distributed training. (unless your model only does large matrix multiplications).
  - We ignored many additional operations like softmax, ReLU/GeLU activations, self-attention, Layer Norm etc.
  - Training divergence and restarting from earlier checkpoints are not uncommon.

- There are various factors that contribute to computation latency
  - Communication latency, memory bandwidth, caching, etc.
  - See https://kipp.ly/transformer-inference-arithmetic/ for an excellent discussion.

Measuring FLOPS Empirically

- There are libraries for computing FLOPS
  - Example: https://github.com/MrYxJ/calculate-flops.pytorch

```
from calflops import calculate_flops_hf

batch_size, max_seq_length = 1, 128
model_name = "meta-llama/Llama-2-7b"
access_token = "" # your application for using llama

flops, macs, params = calculate_flops_hf(model_name=model_name,
                                         access_token=access_token,
                                         input_shape=(batch_size, max_seq_length))
print("%s FLOPs:%s  MACs:%s  Params:%s \n" %(model_name, flops, macs, params))
```

Measuring FLOPS Empirically

- There are libraries for computing FLOPS
  - Example: https://github.com/MrYxJ/calculate-flops.pytorch

Model	Input Shape	Params(B)	Params(Total)	fwd FLOPs(G)	fwd MACs(G)	fwd + bwd FLOPs(G)
bloom-1b7	(1,128)	1.72B	1722408960	310.92	155.42	932.76
bloom-7b1	(1,128)	7.07B	7069016064	1550.39	775.11	4651.18
bloomz-1b7	(1,128)	1.72B	1722408960	310.92	155.42	932.76
baichuan-7B	(1,128)	7B	7000559616	1733.62	866.78	5200.85
chatglm-6b	(1,128)	6.17B	6173286400	1587.66	793.75	4762.97
chatglm2-6b	(1,128)	6.24B	6243584000	1537.68	768.8	4613.03
Qwen-7B	(1,128)	7.72B	7721324544	1825.83	912.88	5477.48
llama-7b	(1,128)	6.74B	6738415616	1700.06	850	5100.19
llama2-7b	(1,128)	6.74B	6738415616	1700.06	850	5100.19

Summary

- One can measure the computational cost of training neural networks in terms of FLOPS.
- Such estimates allow you to estimate the training time of your model, given your GPU specs.
- What else can we do?

Optimal Scaling

Optimal Scaling

- **A real problem:** Your boss gives you a compute budget $$$. What is the best model you can build with this budget?
- We know from the literature that larger models generally lead to better models.
  - Does that mean that you should aim to build the largest model possible?
- Intuitively, if you choose a model that is too large for your budget, you need to cut your training cycles that may reduce its quality.
- **This chapter:** principled approach to selecting optimal data/model scaling.

Scaling

Experimental Setup:

- Pre-train various models of different sizes
- Plot their validation loss throughout training

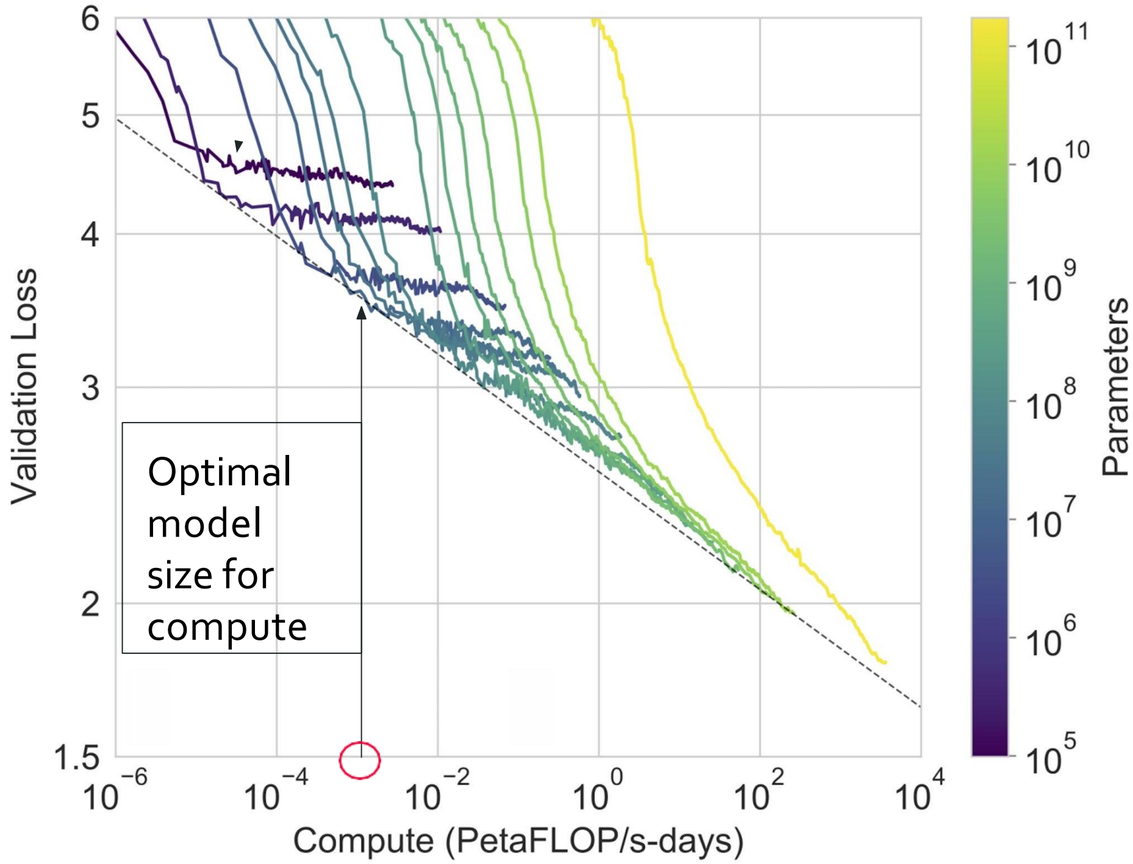


Photo credit: GPT3, Brown et. al., 2020

Scaling

- Smaller models don't have enough capacity to utilize the extra compute. They plateau early.
- Larger models are initially slower to train, but with more compute they reach lower losses.

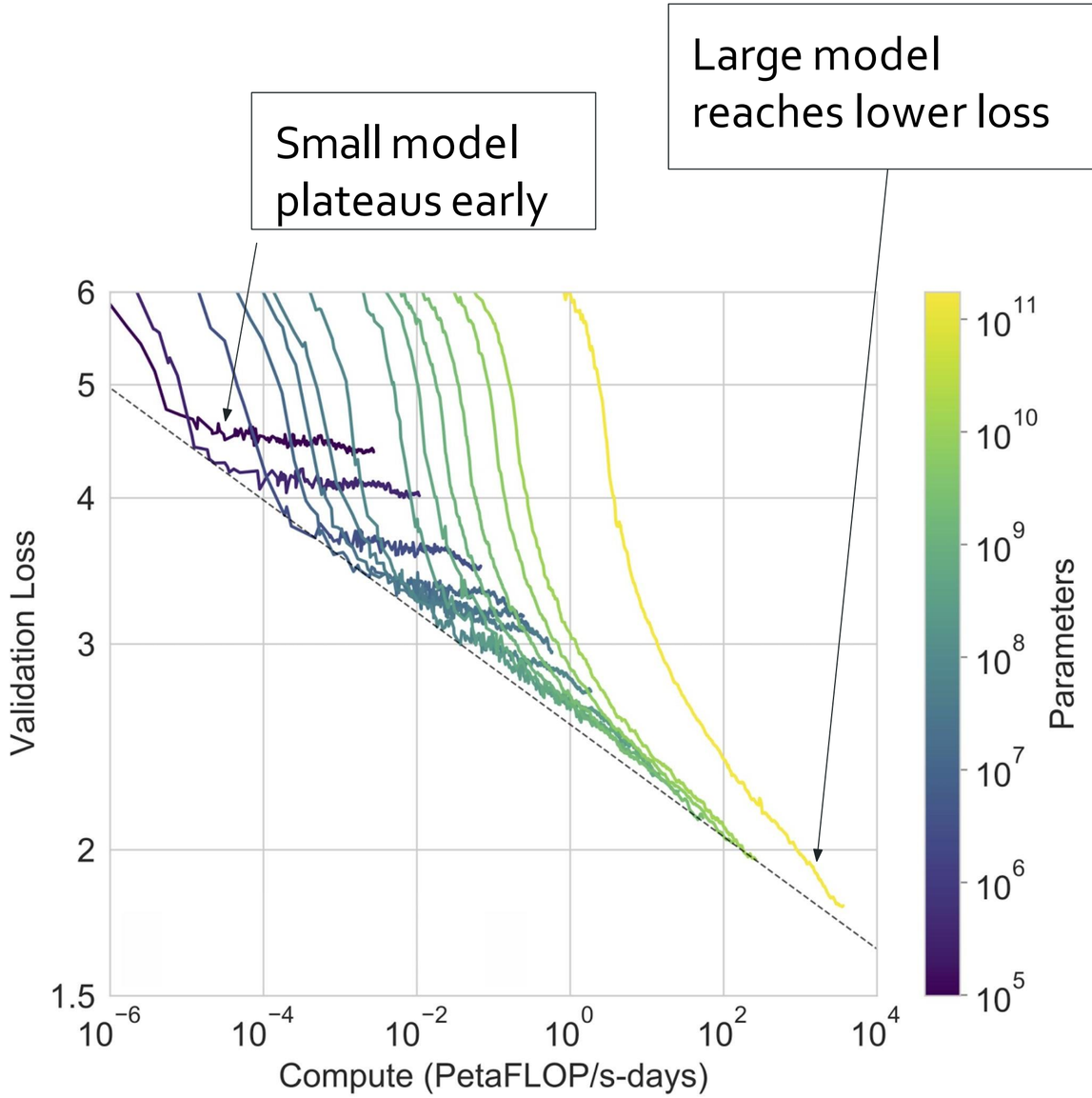


Photo credit: GPT3, Brown et. al., 2020

Scaling - Optimal Model Size

- Let's say our compute budget is $C = 10^{-2}$ PetaFLOPs-days.
- The optimal model is the one that plateaus at exactly C.
- If we train a larger model than optimality point, we won't reach the best performance.
- If we train a smaller model the performance wouldn't be optimal

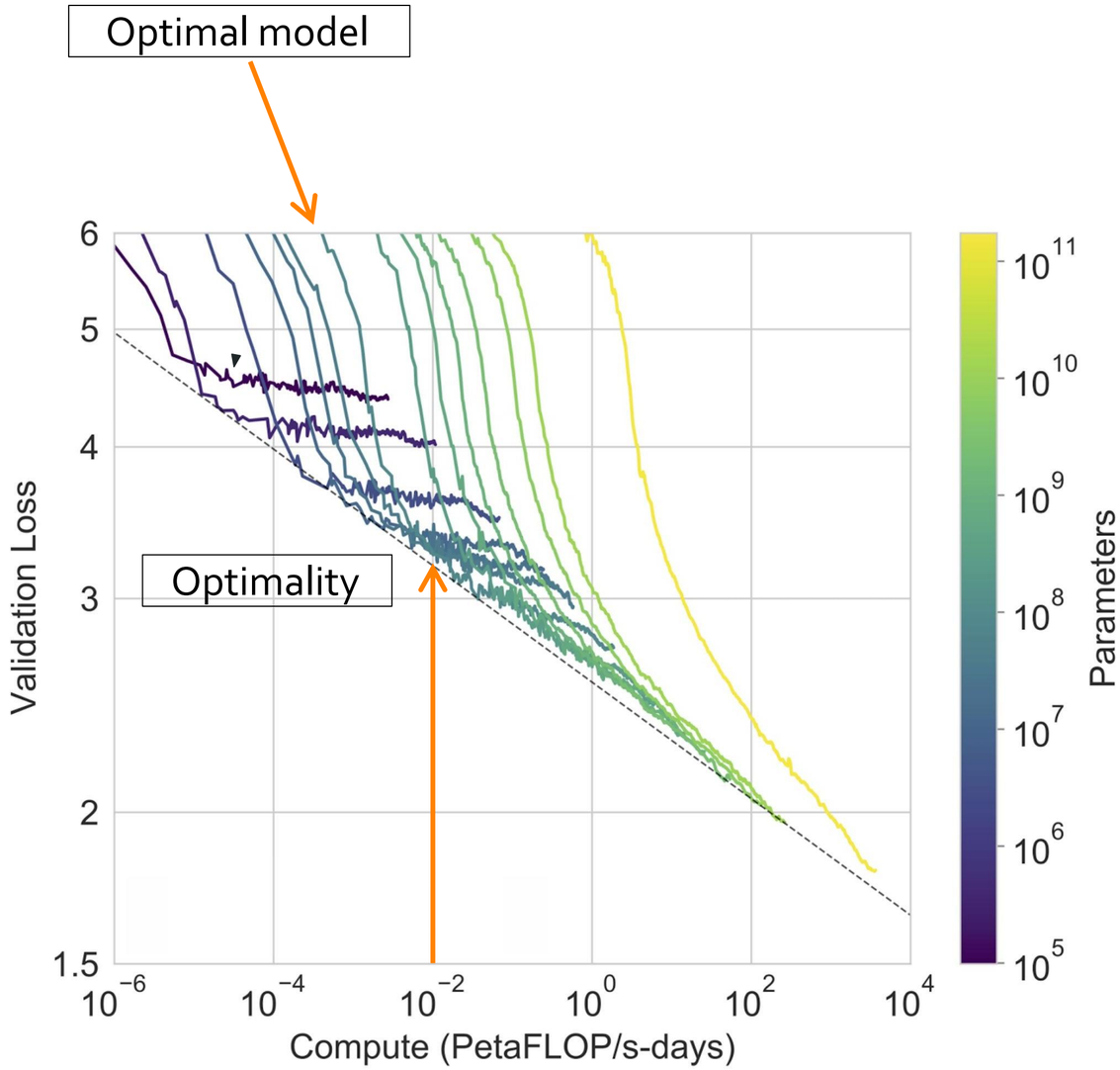


Photo credit: GPT3, Brown et. al., 2020

Scaling - Optimal Model Size

- The idea of "optimal model size for given compute" was introduced by Kaplan et. al.
- In ideal world, we are given lots of compute to train many models to find the optimality.
- Alas not feasible when you have budget to train a single model.
- If we have the equations ("laws") describing the behavior, we can compute it analytically.

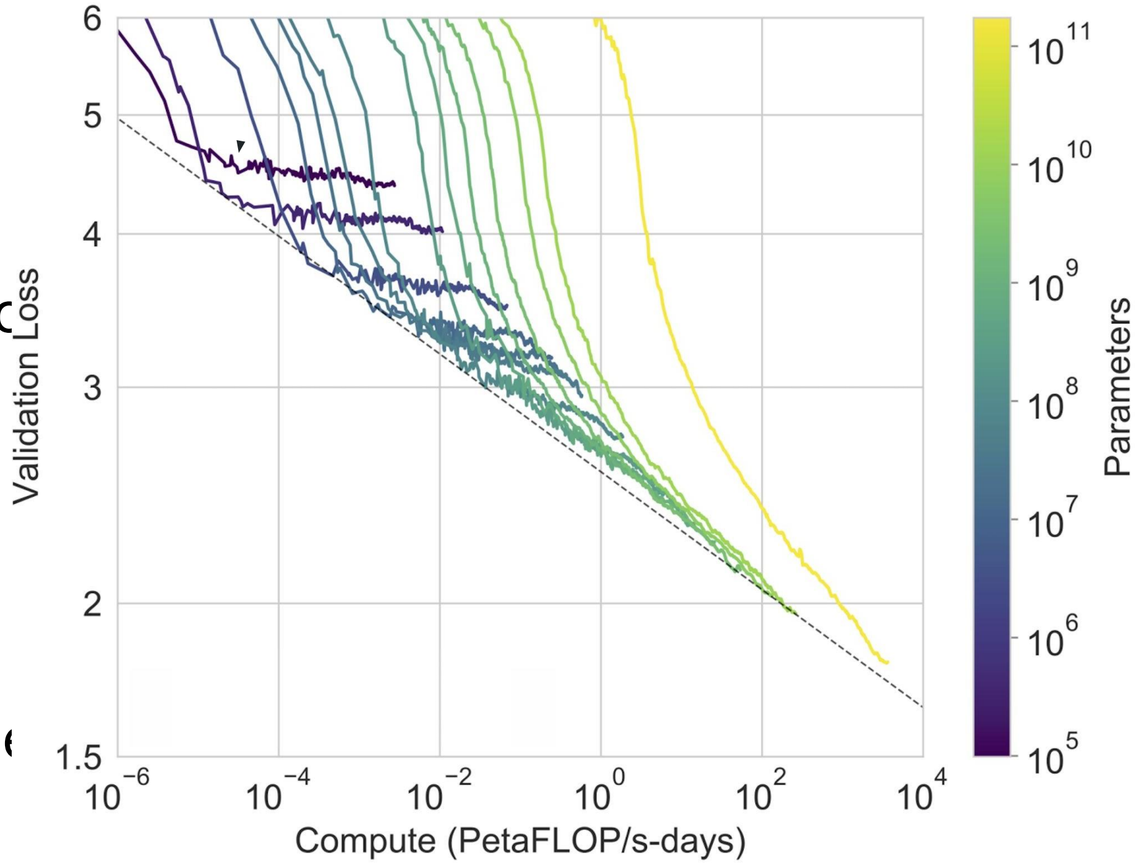


Photo credit: GPT3, Brown et. al., 2020

Scaling - Optimal Model Size

- What is the function that describes this optimality line?

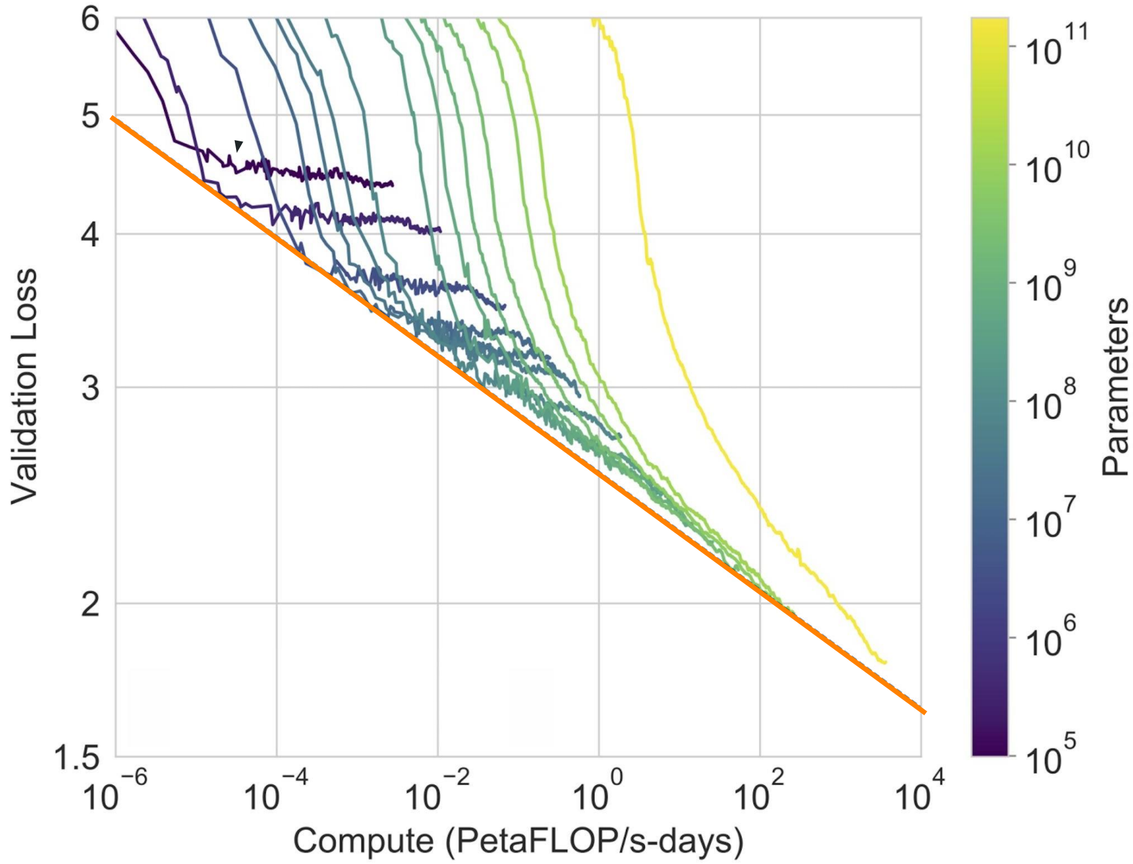


Photo credit: GPT3, Brown et. al., 2020

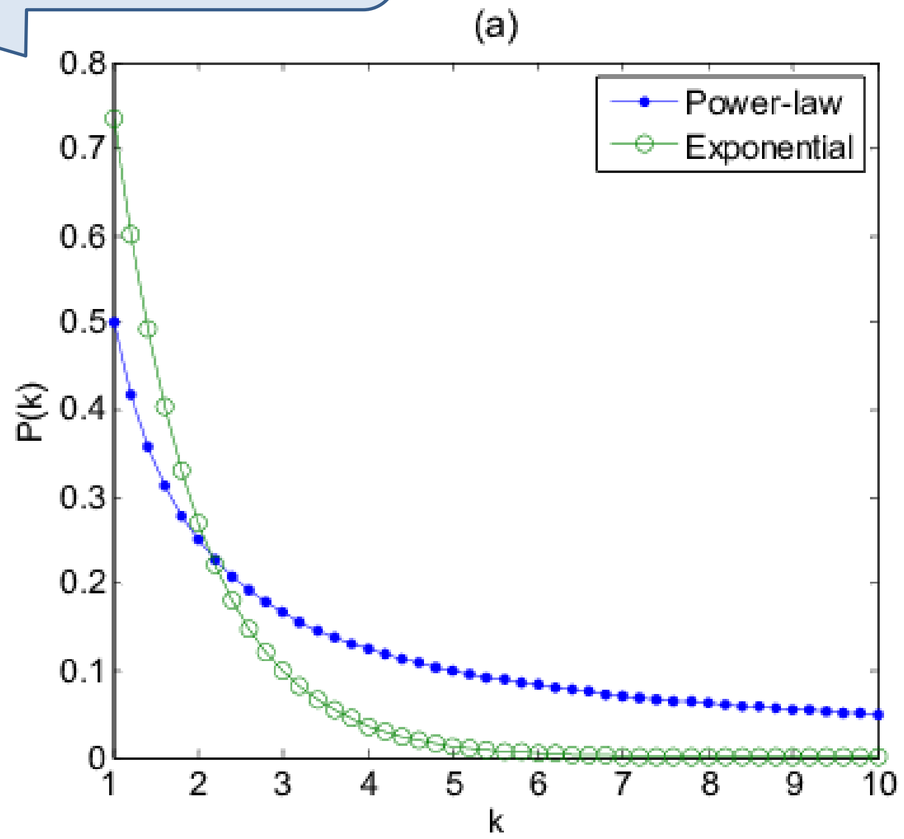
Terminology: Power law vs exponential

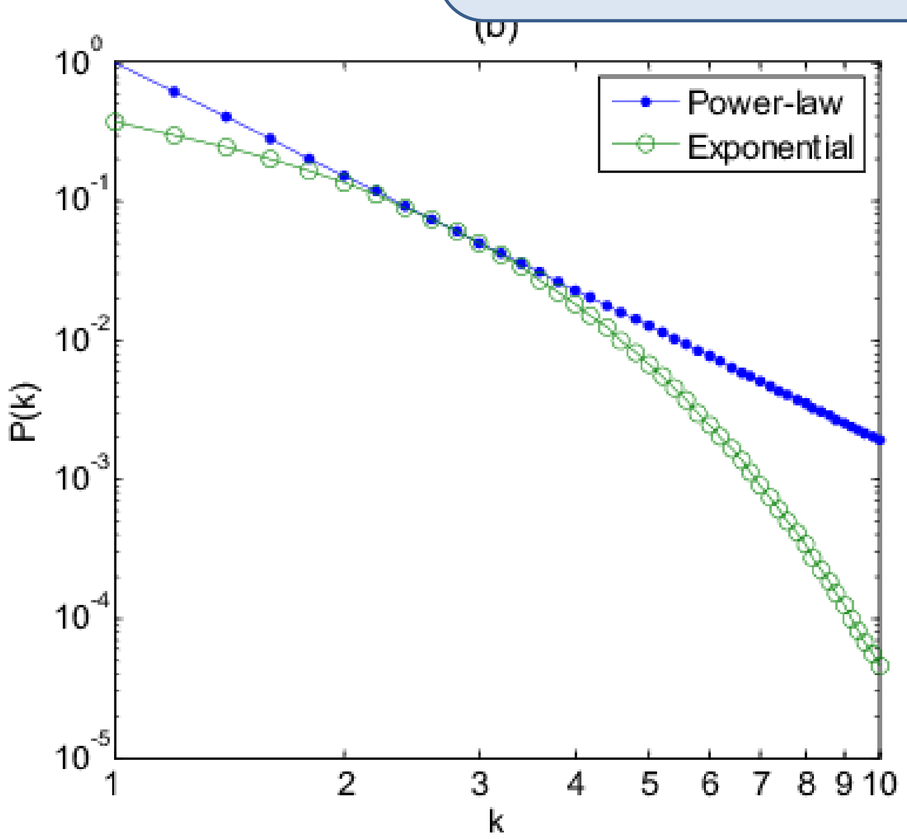
Exponential goes to zero at a faster rate.

Power law $= \text{variable}^{(\text{constant})}$

Exponential $= (\text{constant})^{\text{variable}}$

Power law trend looks linear, when the variable is shown in logarithmic scale.





Scaling Laws of Kaplan et. al., 2020

- A power-law function predicts the "compute efficient" frontier:

$$L \propto C^{-0.48}$$

- Using this (and some other analysis not shown here) we can **analytically** predict the optimal model and data size, for a given amount of compute.

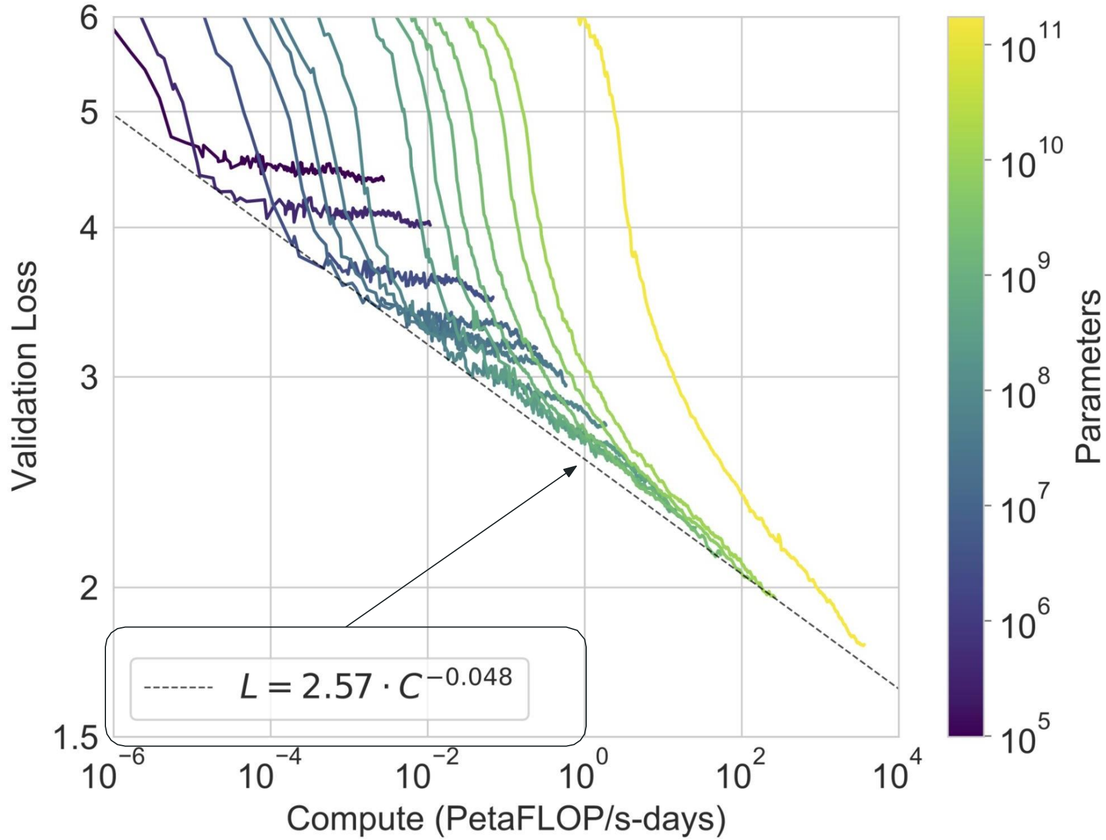


Photo credit: GPT3, Brown et. al., 2020

N: number of model parameters
C: compute
D: dataset size

Scaling Laws: Kaplan et al.

- Optimal model size and optimal number of tokens, for a given compute budget

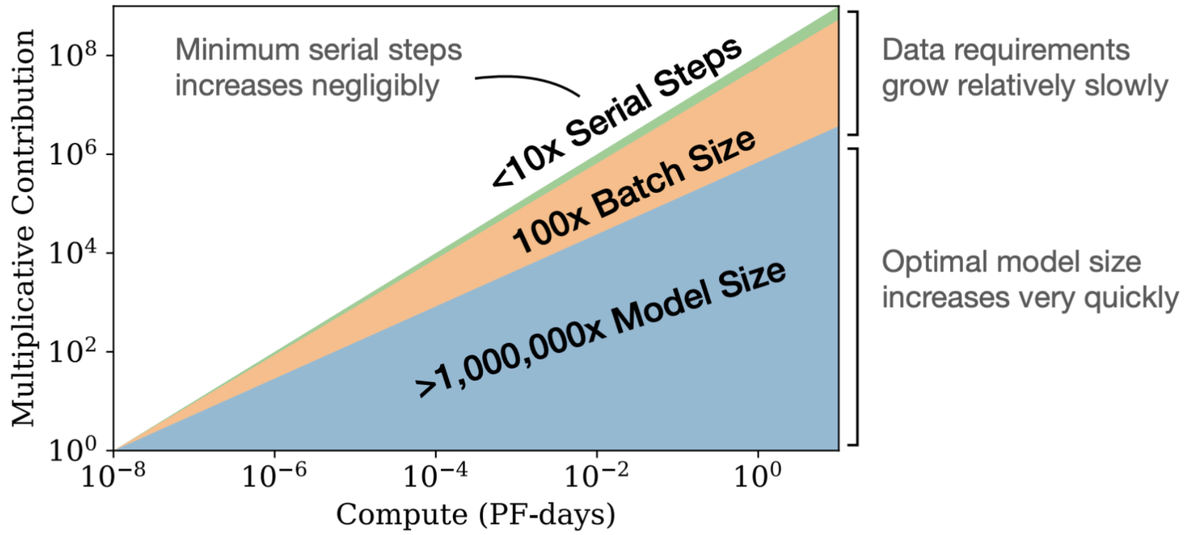
Kaplan et. al. 2020	$N_{\text{opt}} \propto C^{0.73}$	$D_{\text{opt}} \propto C^{0.27}$

N_{opt} exponent $>> D_{\text{opt}}$ exponent

- **Takeaway:** grow the model size faster than growing the number of tokens.
  - **Example:** Given 10x compute, increase N by 5.5x, a[MASK] by 1.8x [MASK]
- GPT3 (and many other followed this recipe) training a 175B model on 300B tokens

Recap: Scaling Laws (Kaplan et al.)

- It appears that there are Precise scaling laws predicting the performance of AI models based on
  - Model size: Number of params
  - Dataset size
  - Total compute used for training
- Scaling Laws: scale model size at a faster rate.
- Given a 10x increase in compute budget,
  - increase the size of the model by 5.5x, and training data size by 1.8x.



Implications of Scaling Laws (Kaplan et al.)

- Subsequent papers tried to engineer larger and larger models

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

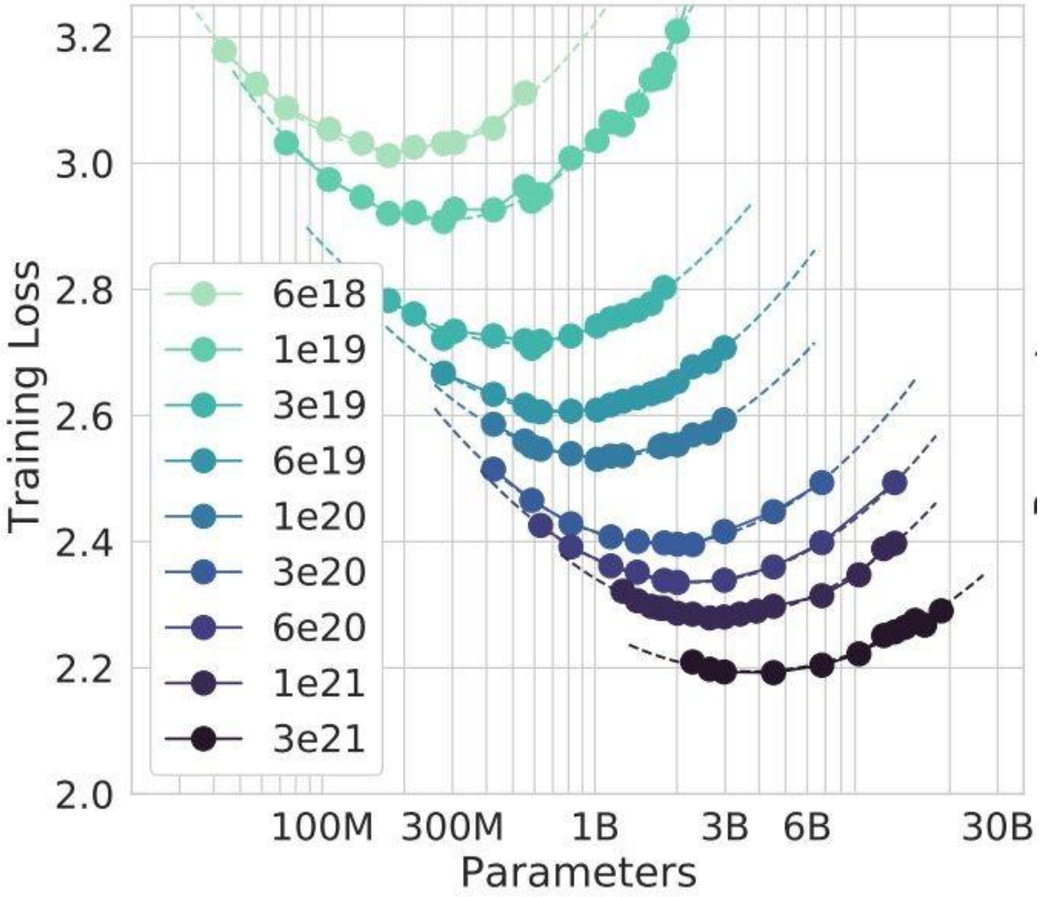
However …

- In 2022 a Hoffmann et al. from DeepMind showed a different set of scaling laws.

Scaling - Kaplan et al. vs. Hoffmann et al.

Experimental setup:

- They chose different compute budgets.
- For each compute budget, train different sized models (varying data or model size)
- They find a clear valley like shape
- For each compute budget there is an optimal model to train



Training Compute-Optimal Large Language Models (2022)
https://arxiv.org/pdf/2203.15556.pdf

N: number of model parameters
C: compute
D: dataset size

Scaling Laws: Hoffmann et al.

- Optimal model size and optimal number of tokens, for a given compute budget

Kaplan et. al. 2020	$N_{\text{opt}} \propto C^{0.73}$	$D_{\text{opt}} \propto C^{0.27}$
Hoffmann et. al., 2021	$N_{\text{opt}} \propto C^{0.5}$	$D_{\text{opt}} \propto C^{0.5}$

$$N_{\text{opt}} \text{ exponent} \cong D_{\text{opt}} \text{ exponent}$$

- Compute and tokens should increase at the same rate.
  - **Example 1:** Given 10x compute, grow N by 3.2x and D by 3.2x
  - **Example 2:** Given 100x compute, grow N by 10x and D by 10x

[ACL 2022 Tutorial Beltagy, Cohan, Logan IV, Min and Singh]

Recap

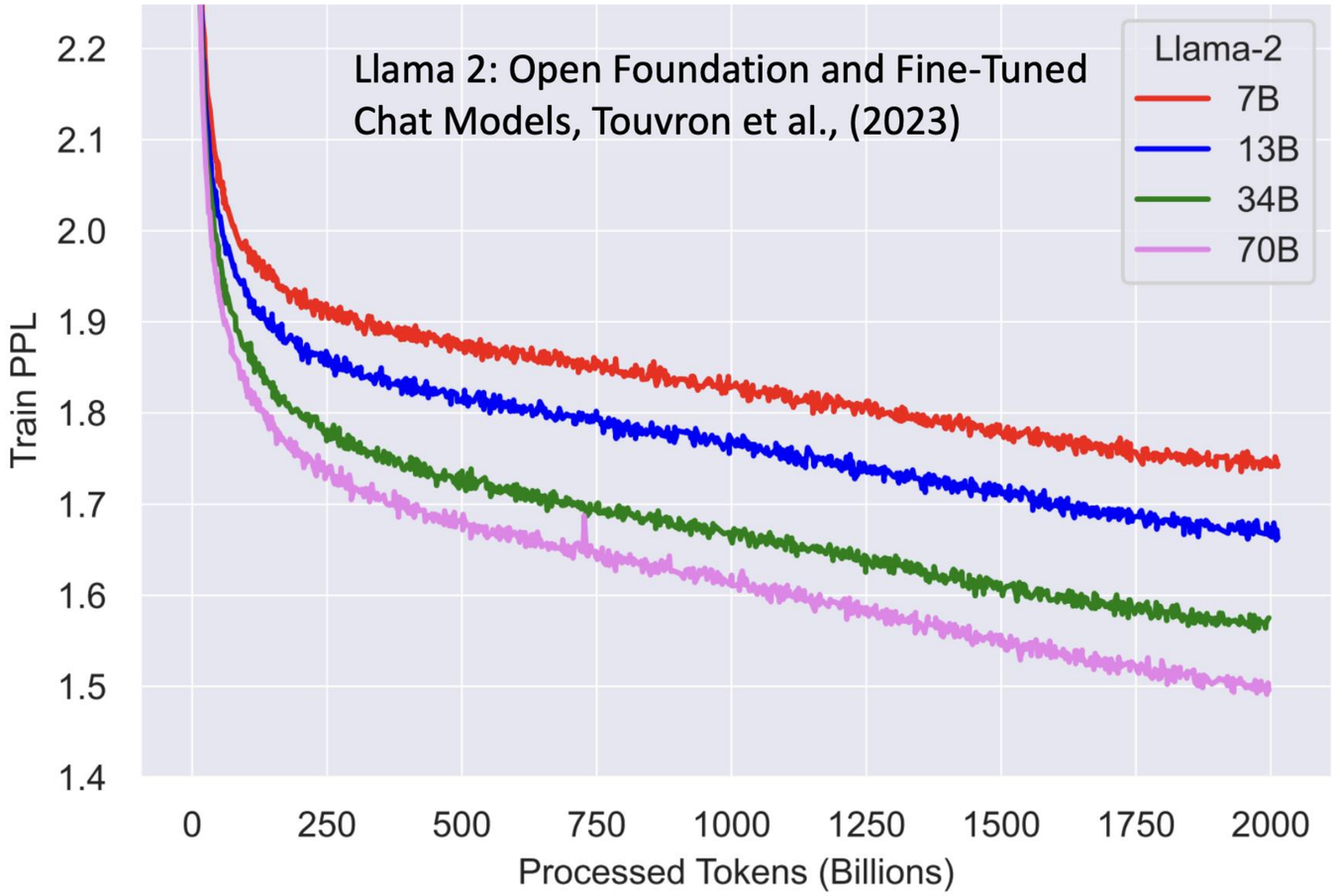
- We used to train “oversized” and “under-trained” models.
- You should scale your model at the same rate as your data.
- For example, if you get a 100x increase in compute,
  - you should make your model 10x bigger and your data 10x bigger.

A Word of Caution

- While we kept referring to these as “law”, one should take them with grain of salt.
- There are various confounding factors here:
  - Different optimizer: AdamW vs. Adam vs. others
  - Different tokenizers
  - Different numerical representation (e.g., bfloat16 vs float32)
  - ….

More Recently …

- Training Loss for Llama 2 models.
- After pretraining on 2T Tokens, the models still did not show any sign of saturation. Why? 🤔
- The scaling laws are usually derived on much smaller scales. Behavior might be different at larger scales.



Data Quality Matters

- There is increasing evidence that data more than just token count.
  - Previously we saw that data duplications and noise hurts LLM performance.
  - There is also evidence that's one can be selective about data diversity.
- These are topics of the ongoing research and there will be more discuss here in coming years.

Beyond neural scaling laws: beating power law scaling via data pruning (Sorscher et al., 2022)

Summary

- Optimality conditions: given a limited budget (compute, data, size) what is the best model you can train.

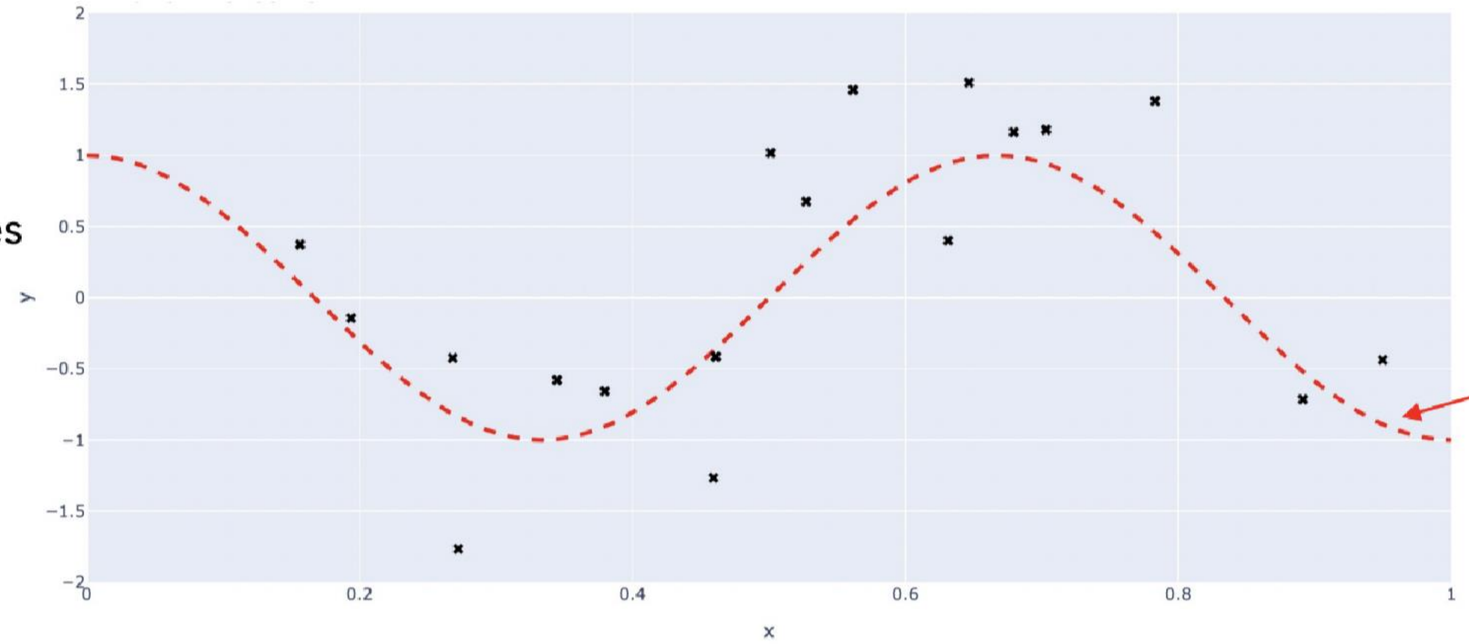
- For now: maintain similar ratio for model size and data size.

- Next: why didn’t we scale earlier?

Why didn’t we scale earlier??

The Old Wisdom: Optimizing Model Capacity

n = 20 samples



✖ random x_i, y_i noisy version of $f^*(x_i)$

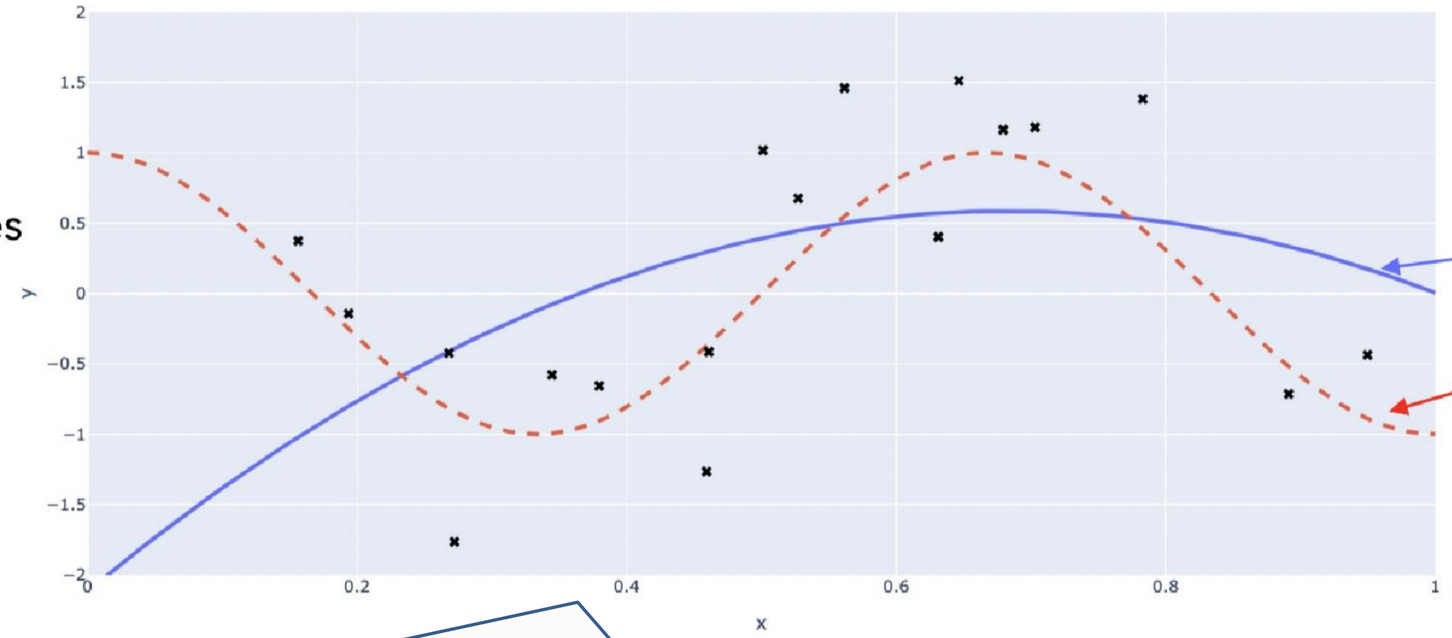
true $f^*(x)$

[Figure credit: NeurIPS '23 Tutorial: Reconsidering Overfitting in the Age of Overparameterized Models]

The Old Wisdom: Optimizing Model Capacity

n = 20 samples

polynomial fit
degree d = 2



random x_i, y_i noisy version of $f^\star(x_i)$

predicted $\hat{f}(x)$

true $f^\star(x)$

Small models cannot fit perfectly.

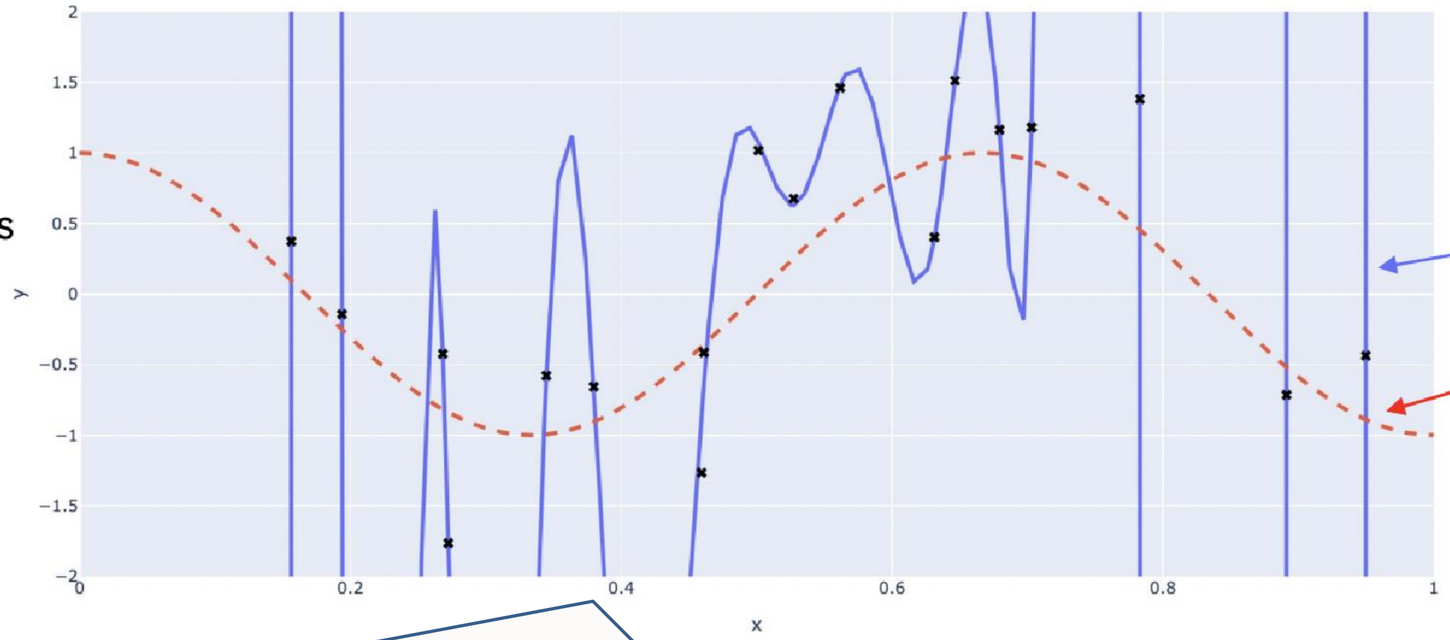
- they cannot express complex functions → high statistical bias.
- largely ignores noise → does not fluctuate a lot (small variance)

[Figure credit: NeurIPS '23 Tutorial: Reconsidering Overfitting in the Age of Overparameterized Models]

The Old Wisdom: Optimizing Model Capacity

n = 20 samples

polynomial fit
degree d = 20



✖ random x_i, y_i noisy version of $f^\star(x_i)$

predicted $\hat{f}(x)$

true $f^\star(x)$

Large models fit perfectly (overfit)

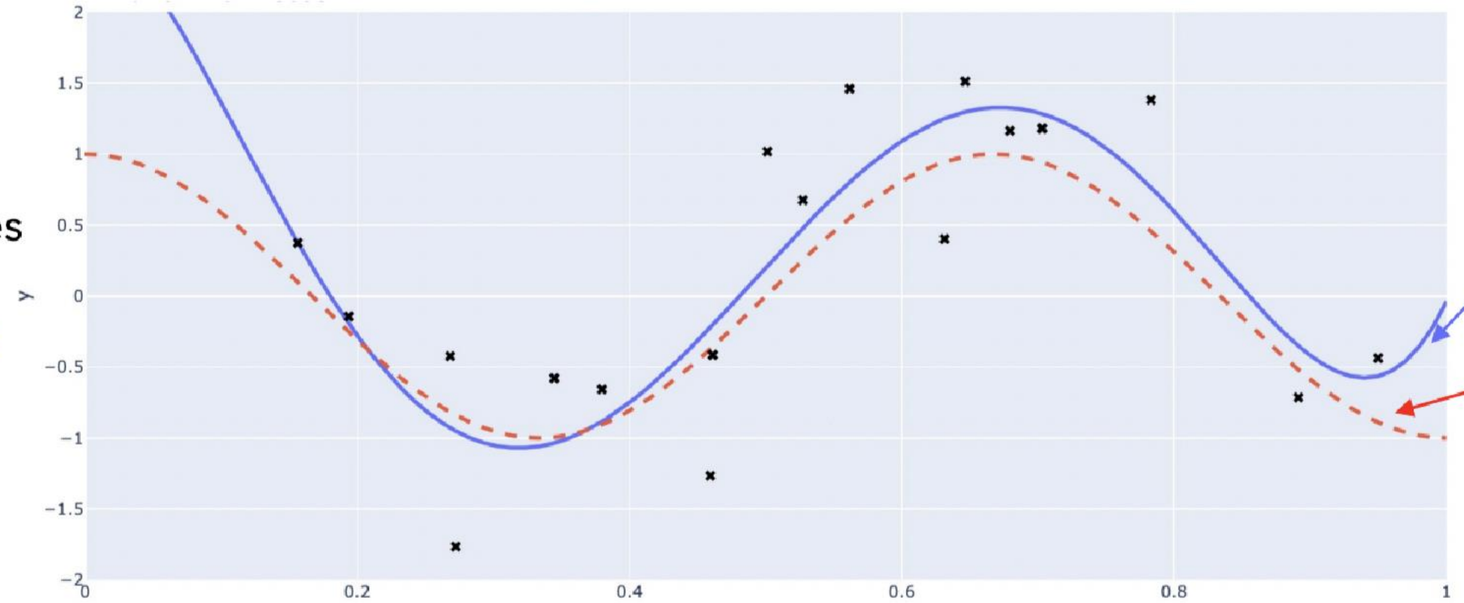
- Can express function of interest → small statistical bias.
- Fits too much of the noise (overfit) → fluctuates a lot (high variance)

[Figure credit: NeurIPS '23 Tutorial: Reconsidering Overfitting in the Age of Overparameterized Models]

The Old Wisdom: Optimizing Model Capacity

n = 20 samples

polynomial fit
degree d = 5



random x_i, y_i noisy version of $f^\star(x_i)$

predicted $\hat{f}(x)$

true $f^\star(x)$

Classical generalization theory — one can get generalization by optimizing for capacity (expressivity) — equivalent to balancing the bias-variance trade-off.

[Figure credit: NeurIPS '23 Tutorial: Reconsidering Overfitting in the Age of Overparameterized Models]

Prochain cours

Parameters Efficient Fine Tuning

Merci !

Mohamed Abbas KONATE

mohamed-abbas.konate@michelin.com