Scale

Lecture 8

EN.705.743: ChatGPT from Scratch

This Course, Part II

Part II of the Course

Congrats! You now know how to make a language model, train it, and sample text from it!

This means we are now entering part II of the course: all the techniques to turn a base LLM into a magnificent, state-of-the-art AI.

If you felt that the first part of the course was a bit of a grind, I think you will like this part better :)

Part II of the Course

A few things will change going forward:

- Weekly programming assignments are now over, replaced with a final project which will be due on our last day of class. (We will devote time to this today)
- Lectures are now somewhat independent of each other.

Going forward, each lecture will focus on one way that a base LLM can be improved for specific purposes.

A state-of-the-art system like ChatGPT uses many of these working together.

Part II of the Course

Upcoming lectures:

- Scale (Today)
  - Making LLMs more capable by increasing their size
- Fine-tuning and RAG
  - Making LLMs really good at specific tasks
- PEFT and Quantization
  - Increasing the efficiency of tuning and deploying an LLM
- RLHF
  - Adding behaviors, preferences, and a sense of right and wrong to a chatbot
- Multimodality
  - LLMs that also understand images and audio.

Lecture Outline

- Results from our homegrown models
- Short History of LLM Research
- Parallelization Techniques
- Final Project Overview

Results from Our Models

Thomas Jefferson was…

Thomas Jefferson was the command in two First World War. In which event had become a senator, and many were particularly good friends as president.Both of Manipiros began to continue work on how American agents go on regular debateders they could use Malcolm Belgradous about his life rights towards her mother's best belonging to the former officers too poor.

Homework default values:

35M parameters
105M tokens

Thomas Jefferson was ultimately elected as the first Democrat to the Virginia House of Representatives, having raised a large stockpile of common land in his district. Williams enjoyed strenuous adventures during these years, becoming known as one of their " master masonesses's manuals " by practicing with them.<|endoftext|>

I think I got lucky with the quality of this generation.

Scaled up:

900M parameters **(30x)**
1B tokens **(10x)**

A Modern-ish LLM

Using GPT3.5 Turbo (175B params, trillions of tokens) I get:

Thomas Jefferson was not above buying land. In fact, he was an avid land speculator and acquired large tracts of land throughout his lifetime. He saw land ownership as a way to expand his personal wealth and to support his vision for a strong agrarian society in America.

At this point we would need a more complicated example to show how good this model really is compared to the previous ones.

Another Modern LLM

Using Phi 1.5 (1.3B params, 150B tokens) I get:

Thomas Jefferson was a great leader who believed in the power of democracy. He was the third President of the United States

Honestly the most straightforward answer of the bunch. This is what someone actually would want to know about Thomas Jefferson.

This is the only model that mentioned he was President!

To Summarize:

Our small model (0.035B): Almost English

Larger version of the same model (0.9B): A full sentence here and there.

GPT 3.5 Turbo (175B): Coherent summary of TJ’s views on land ownership (?)

Microsoft Phi 1.5 (1.3B): Actually said who TJ was.

What is going on here?

Scaling

There are three independent ways an LLM can be improved:

1) Parameter Count
2) Quantity of Training Data
3) Quality of Training Data

= Training time and/or more compute

I don’t know if it’s fair to call this “scaling”, but I think it applies at least in an abstract sense.

To Summarize:

Our small model (0.035B): Almost English

Larger version of the same model (0.9B): A full sentence here and there.

GPT 3.5 Turbo (175B): Coherent summary of TJ's views on land ownership (?)

More parameters, more data.

Microsoft Phi 1.5 (1.3B): Actually said who TJ was.

Really high quality data.

(also, some synthetic data which is interesting. More on this later.)

A (Brief, Not at all Exhaustive) History of LLMs

A History of LLMs

The effects of parameter count, data quantity, and data quality were not all discovered at the same time.

By looking at research from the last few years we can learn a lot about the evolution of language models, and clearly see the impact of key discoveries.

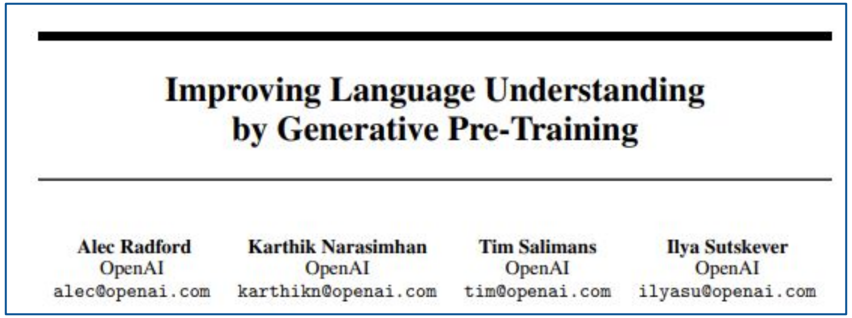
Timeline

GPT (June 2018)
GPT2 (Feb 2019)
T5 (Oct 2019)
Scaling Laws (Jan 2020)
GPT3 (May 2020)
T5-XXL (Jan 2021)
Chinchilla (Mar 2022)
PaLM (Apr 2022)
ChatGPT (Nov 2022)
LLaMA (Feb 2023)
Alpaca (March 2023)
Phi (June 2023)
Mistral (Sep 2023)
OLMo / Dolma (Feb 2024)
LLaMA 3 (Apr 2024)
…

GPT 1 and GPT 2

The early GPT models were pioneering works. GPT 1 was released by OpenAI in 2018 (For reference, the transformers paper was released in 2017).

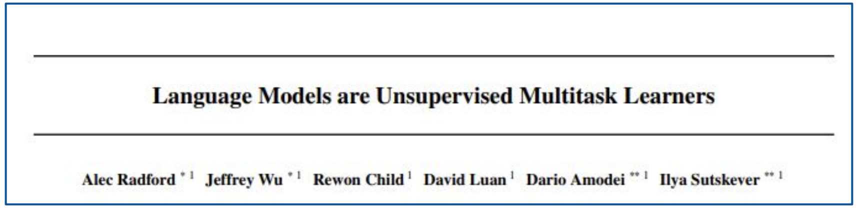
GPT 2 was released in early 2019, and scaled GPT1 by more than 10x. GPT2 has 1.5B parameters (this was considered extremely huge at the time.)



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 1: 117M Params, maybe 1B tokens (100 epochs)



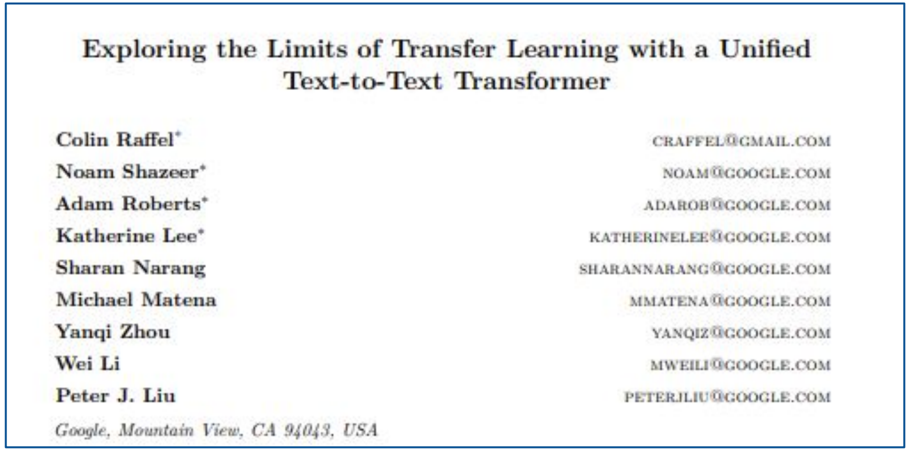
Language Models are Unsupervised Multitask Learners

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

GPT 2: 1.5B Params, 10B tokens

T5

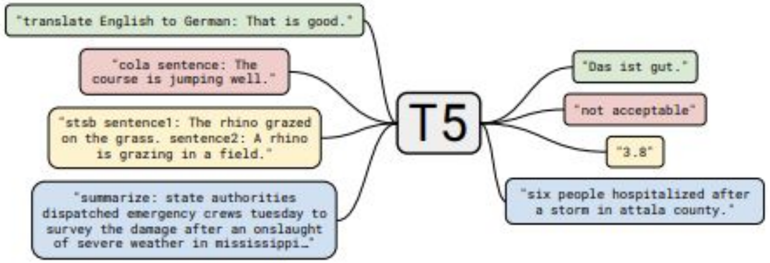
Google released T5 in late 2019 which used the encoder-decoder architecture and laid groundwork for many future models. They also introduce C4, a huge dataset of text scraped from the web.



Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

Colin Raffel*	CRAFFEL@GMAIL.COM
Noam Shazeer*	NOAM@GOOGLE.COM
Adam Roberts*	ADAROB@GOOGLE.COM
Katherine Lee*	KATHERINELEE@GOOGLE.COM
Sharan Narang	SHARANNARANG@GOOGLE.COM
Michael Matena	MMATENA@GOOGLE.COM
Yanqi Zhou	YANQIZ@GOOGLE.COM
Wei Li	MWEILI@GOOGLE.COM
Peter J. Liu	PETERJLIU@GOOGLE.COM

Google, Mountain View, CA 94043, USA


"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

T5: 11B Params, 34B tokens

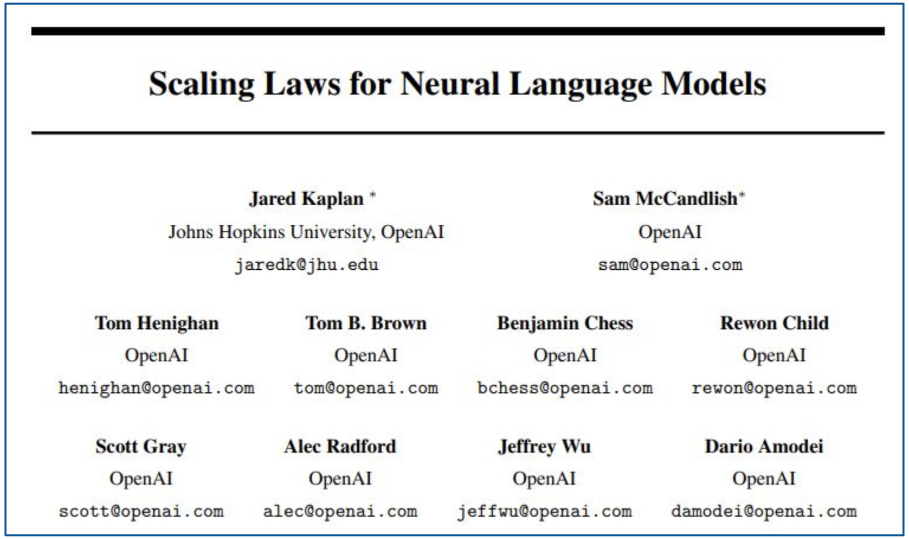
In less than 2 years we have gone up ~100x

Scaling Laws

In early 2020, OpenAI releases a study of model scale. It is clear that models are becoming larger (and more expensive), and researchers want to understand what the benefits are for an Nx increase in size.

They test models from 1k to 1B parameters, and train them for 100B tokens each.

There are number of key insights made, see the next few slides.



Scaling Laws for Neural Language Models

Jared Kaplan [*]
Johns Hopkins University, OpenAI
jaredk@jhu.edu

Sam McCandlish[*]
OpenAI
sam@openai.com

Tom Henighan
OpenAI
henighan@openai.com

Tom B. Brown
OpenAI
tom@openai.com

Benjamin Chess
OpenAI
bchess@openai.com

Rewon Child
OpenAI
rewon@openai.com

Scott Gray
OpenAI
scott@openai.com

Alec Radford
OpenAI
alec@openai.com

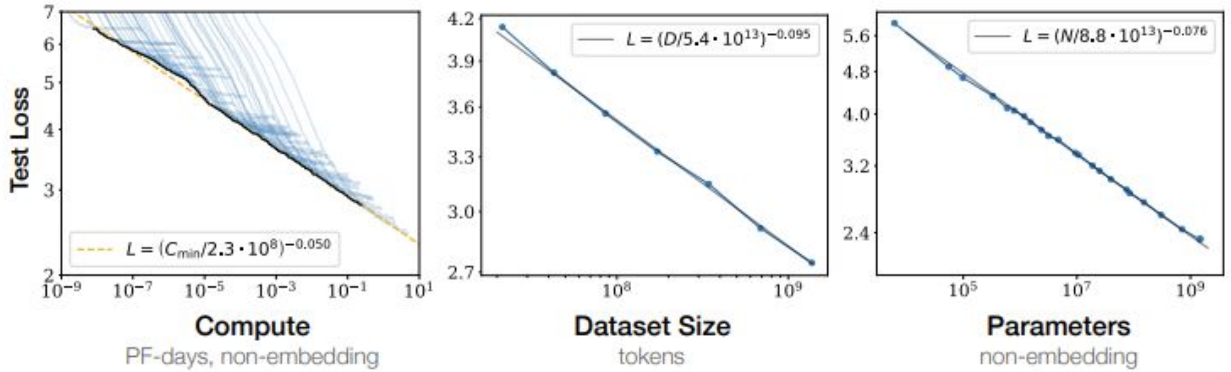
Jeffrey Wu
OpenAI
jeffwu@openai.com

Dario Amodei
OpenAI
damodei@openai.com

Scaling Laws

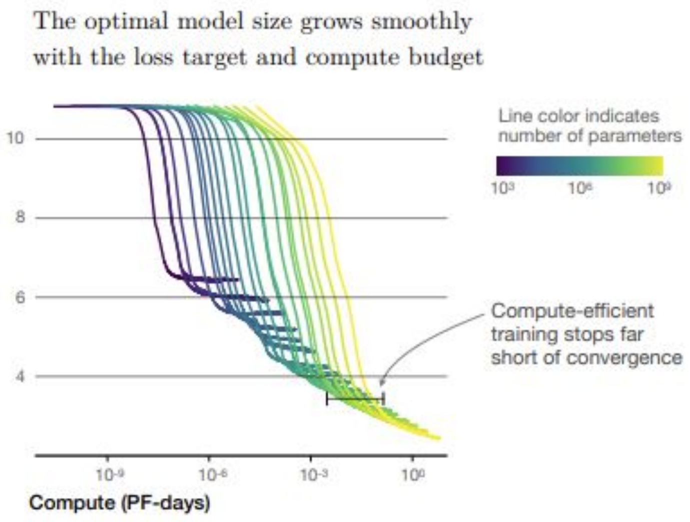
The main contribution of the paper is to experimentally prove what everyone seemed to “know”- a larger model performs better. They create formulas that will predict, based on model size and amount of data, how good the final model will be!

(This is extremely helpful for companies now looking to invest $M in AI: they can predict what they will get out of it.)

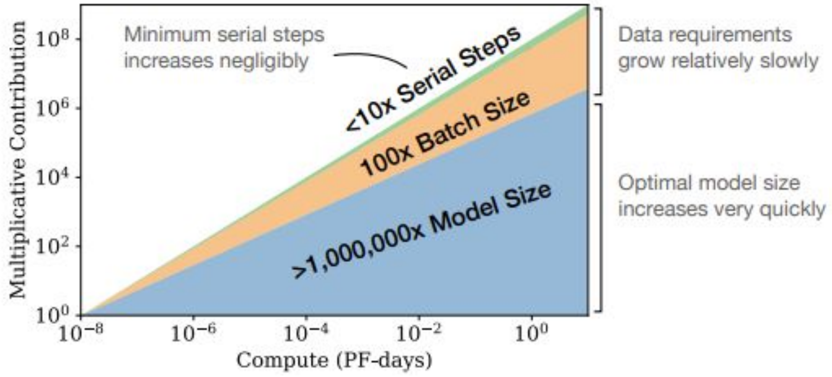


Scaling Laws

A few takeaways:



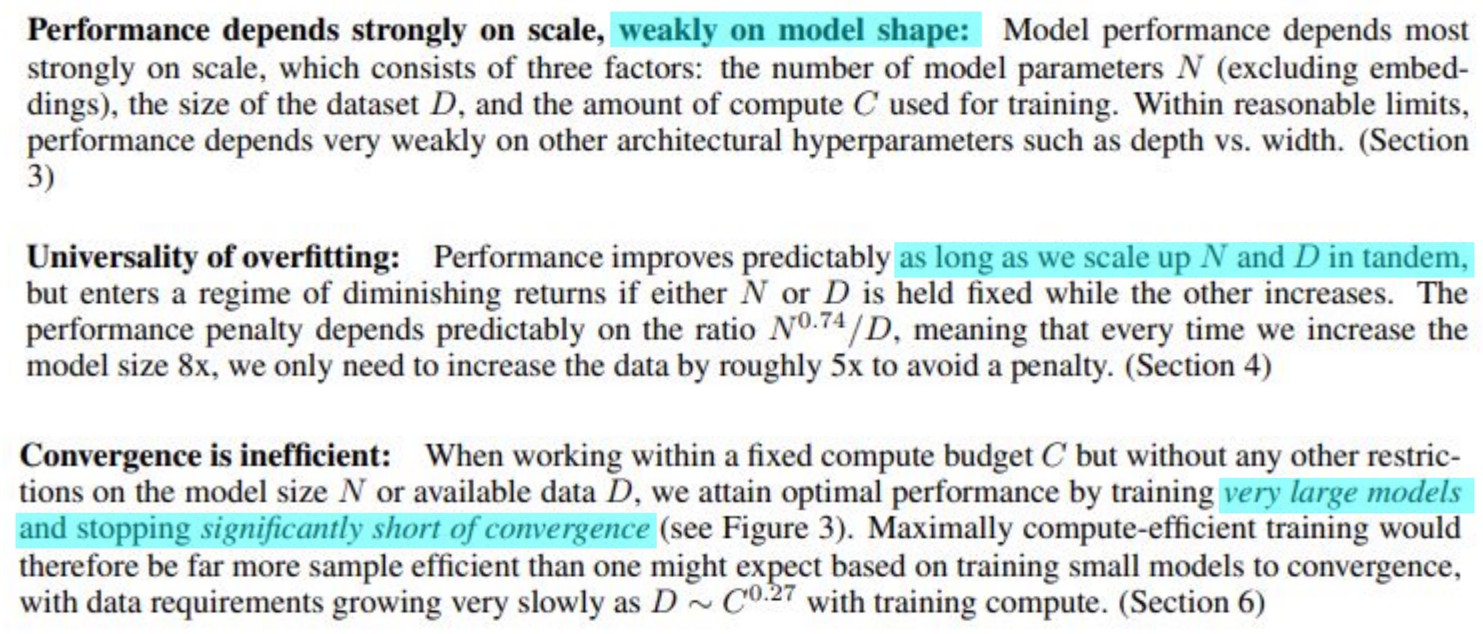
A smaller model converges faster, but to a worse loss.



They argue that given increased compute, you should use this to support larger and larger models.

Scaling Laws

A few more takeaways (from page 3 of the paper, highlighting mine):



Performance depends strongly on scale, weakly on model shape: Model performance depends most strongly on scale, which consists of three factors: the number of model parameters N (excluding embeddings), the size of the dataset D, and the amount of compute C used for training. Within reasonable limits, performance depends very weakly on other architectural hyperparameters such as depth vs. width. (Section 3)

Universality of overfitting: Performance improves predictably as long as we scale up N and D in tandem, but enters a regime of diminishing returns if either N or D is held fixed while the other increases. The performance penalty depends predictably on the ratio $N^{0.74}/D$, meaning that every time we increase the model size 8x, we only need to increase the data by roughly 5x to avoid a penalty. (Section 4)

Convergence is inefficient: When working within a fixed compute budget C but without any other restrictions on the model size N or available data D, we attain optimal performance by training *very large models* and stopping *significantly short of convergence* (see Figure 3). Maximally compute-efficient training would therefore be far more sample efficient than one might expect based on training small models to convergence, with data requirements growing very slowly as $D \sim C^{0.27}$ with training compute. (Section 6)

Note on Units

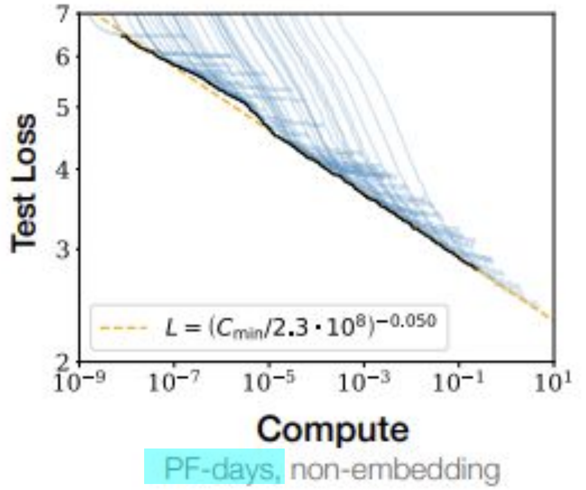
This is kind of like “light years”. Just as light years is not a measurement of time but instead a measurement of something accomplished in that time (distance travelled), petaflop-days is a measure of calculations performed in an amount of time.

The units for scaling research (and measuring compute in general) can be really foreign.

A common unit is “petaflop-days”, which is a measure of a quantity of calculations performed.

A FLOP (floating-point operation) is a low-level operation performed by a computer on float data (addition, multiplication, etc). *FLOPs* is short for “FLOP per second”.

One PF-day is the amount of FLOPs performed by a computer running at 1 petaflop/sec for one day. Peta=10^{15} and there are 86,400 seconds in a day, so this is roughly **10^{20} low-level computations.**



Diatribe on Units

I personally hate the unit “petaflop-days”. It is really confusing and somewhat awkward to calculate. You have to count the operations performed.

A GPU will have an advertised number of FLOPs that it can perform, but this is in an ideal setting at 100% continuous utilization. This is helpful for comparing two GPUs, but doesn’t tell you much about using the GPU in practice.

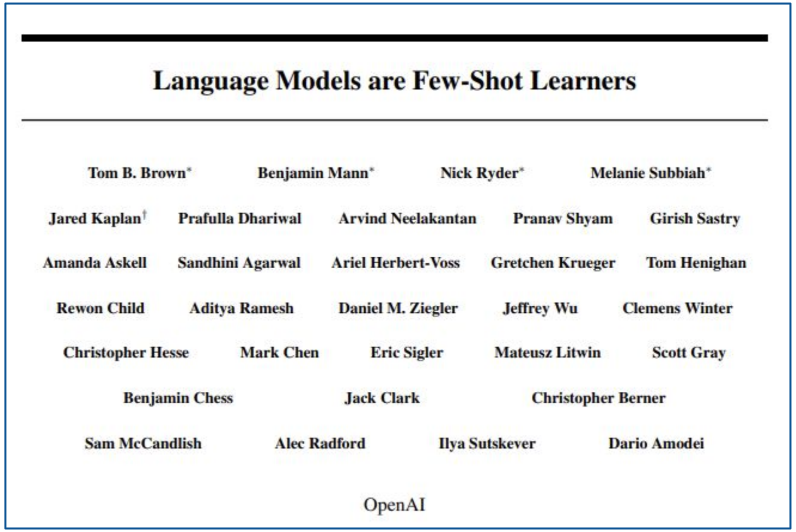
Some papers will report training time in “GPU-days” instead, which is tied the actual reality of running on hardware.

This is really intuitive. 7 GPU days means either training on 1 GPU for a week or parallelized across 7 GPUs for 1 day or some other arrangement that adds up to 7.

GPT3

Several months after “Scaling Laws”, OpenAI released GPT3, coming in at 175B parameters and outperforming other models due to sheer model scale.

Most estimates I have seen conclude that they probably used around 1000 V100 GPUs to train the model, about $20-30M worth of chips. (Fortunately, Microsoft loaned them to OpenAI).



Language Models are Few-Shot Learners

Tom B. Brown* Benjamin Mann* Nick Ryder* Melanie Subbiah*

Jared Kaplan† Prafulla Dhariwal Arvind Neelakantan Pranav Shyam Girish Sastry

Amanda Askell Sandhini Agarwal Ariel Herbert-Voss Gretchen Krueger Tom Henighan

Rewon Child Aditya Ramesh Daniel M. Ziegler Jeffrey Wu Clemens Winter

Christopher Hesse Mark Chen Eric Sigler Mateusz Litwin Scott Gray

Benjamin Chess Jack Clark Christopher Berner

Sam McCandlish Alec Radford Ilya Sutskever Dario Amodei

OpenAI

GPT 3: 175B Params, 300B Tokens

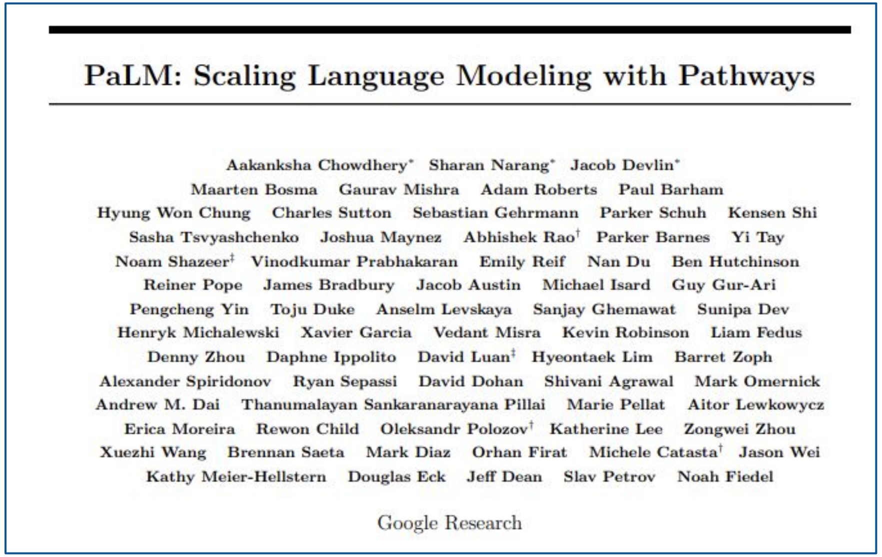
Apex of Model Size (PaLM)

Note: GPT1 had four authors.

In 2022 things just started to get silly. The only way for organizations to outdo each other was to make bigger and bigger models with increasingly absurd costs.

In April 2022 Google announced PaLM, a 540B parameter model.

Even larger models were created by ensembling other models (>1T parameters altogether).



PaLM: Scaling Language Modeling with Pathways

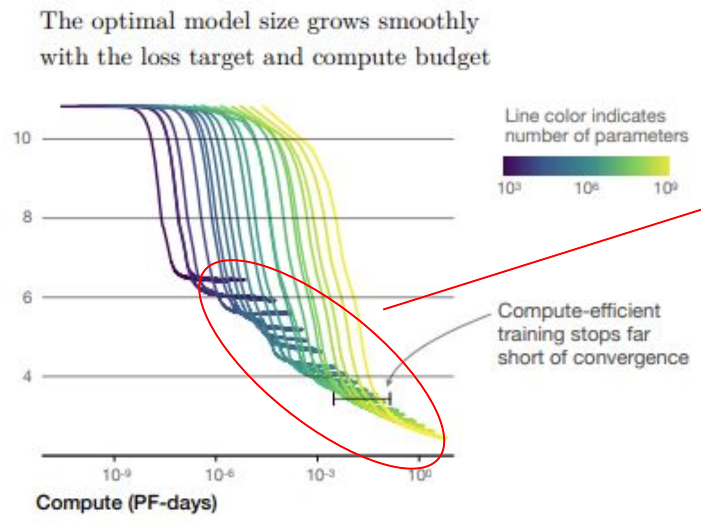
Aakanksha Chowdhery[*] Sharan Narang[*] Jacob Devlin[*]
Maarten Bosma Gaurav Mishra Adam Roberts Paul Barham
Hyung Won Chung Charles Sutton Sebastian Gehrmann Parker Schuh Kensen Shi
Sasha Tsvyashchenko Joshua Maynez Abhishek Rao[†] Parker Barnes Yi Tay
Noam Shazeer[‡] Vinodkumar Prabhakaran Emily Reif Nan Du Ben Hutchinson
Reiner Pope James Bradbury Jacob Austin Michael Isard Guy Gur-Ari
Pengcheng Yin Toju Duke Anselm Levskaya Sanjay Ghemawat Sunipa Dev
Henryk Michalewski Xavier Garcia Vedant Misra Kevin Robinson Liam Fedus
Denny Zhou Daphne Ippolito David Luan[‡] Hyeontaek Lim Barret Zoph
Alexander Spiridonov Ryan Sepassi David Dohan Shivani Agrawal Mark Omernick
Andrew M. Dai Thanumalayan Sankaranarayana Pillai Marie Pellat Aitor Lewkowycz
Erica Moreira Rewon Child Oleksandr Polozov[†] Katherine Lee Zongwei Zhou
Xuezhi Wang Brennan Saeta Mark Diaz Orhan Firat Michele Catasta[†] Jason Wei
Kathy Meier-Hellstern Douglas Eck Jeff Dean Slav Petrov Noah Fiedel

Google Research

PaLM: 540B Params, 780B Tokens,
the “paper” is 87 pages long.

Chinchilla

Separately, Google DeepMind (not yet merged with Google Brain, which happened the following year), makes a critical discovery in 2022: you can make smaller models really good by training them for a long time.

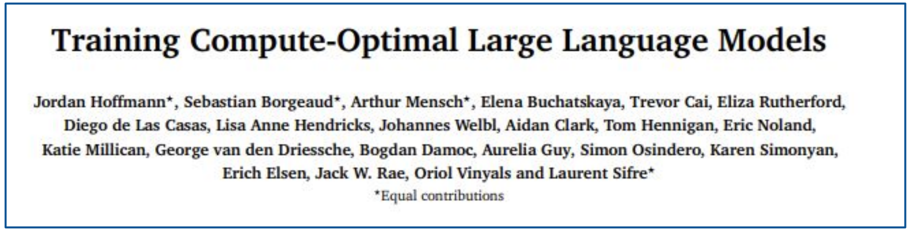


This section of the loss curve is not flat! It just needs to go for a lonnnnnng time.

Chinchilla

They release a model called Chinchilla based on their findings, which is much smaller than other models but trained many times longer.

Given a fixed amount of compute, they argue you should prioritize data throughput over model size.



Training Compute-Optimal Large Language Models

Jordan Hoffmann*, Sebastian Borgeaud*, Arthur Mensch*, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and Laurent Sifre*

*Equal contributions

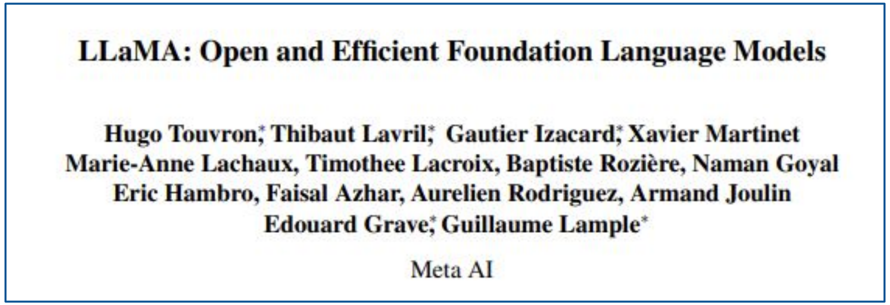
Chinchilla: 70B Params, 1.4T Tokens

LLaMA

Following Chinchilla, the most powerful language models in 2022 and 2023 were smaller, and more accessible to the community.

Meta released LLaMA in early 2023, which came in sizes up to 70B.

However, many third parties began using LLaMA 7B and 13B for their projects since these could run on their resources.

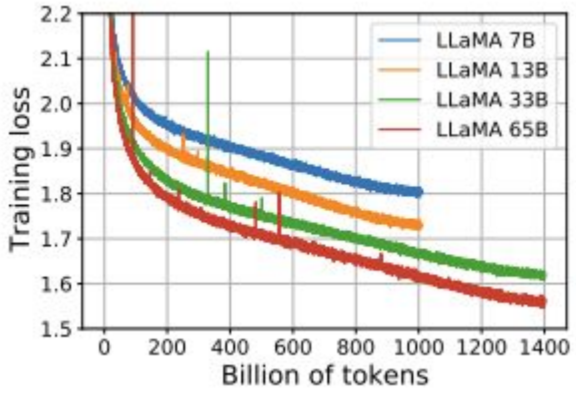


LLaMA: Open and Efficient Foundation Language Models

Hugo Touvron*, Thibaut Lavril*, Gautier Izacard*, Xavier Martinet
Marie-Anne Lachaux, Timothee Lacroix, Baptiste Rozière, Naman Goyal
Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin
Edouard Grave*, Guillaume Lample*

Meta AI

LLaMA: 70B Params, 1.4T Tokens



Small Again (Mistral)

Given the trend for longer training, smaller models, and most organizations being limited in what they are able to run, lots of work went into creating small but powerful models (and still continues).

Ex-Meta and DeepMind Employees started Mistral AI in 2023, and in September 2023 released Mistral 7B.

Mistral now offers several models which are ensembles- their 8x22B model has 176B parameters (same as GPT3), but only uses a small fraction on each forward pass.

Mistral 7B

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, William El Sayed

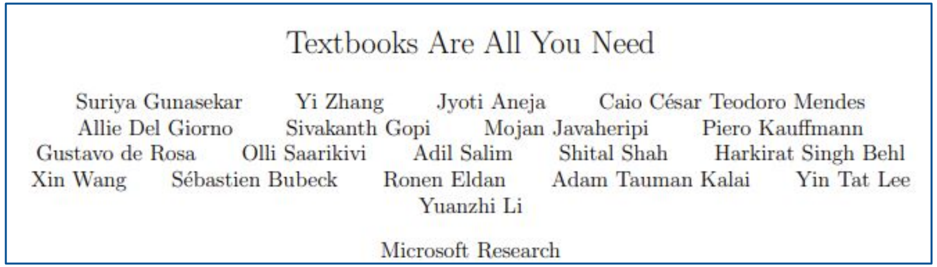
Mistral 7B: 7B Params, Unknown Tokens

Small Again (Phi)

Microsoft has been releasing a series of models called Phi, which are noteworthy for being small and using less data.

They discovered that in addition to model size and data quantity, data quality is important. Phi trains on textbooks and other high-quality data (compared to, i.e. reddit forums).

Interesting Note: Starting with Phi v 1.5, the training data was partially synthetic, produced by a larger model that could produce “textbook-like” exampels.



Textbooks Are All You Need

Suriya Gunasekar Yi Zhang Jyoti Aneja Caio César Teodoro Mendes Allie Del Giorno Sivakanth Gopi Mojan Javaheripi Piero Kauffmann Gustavo de Rosa Olli Saarikivi Adil Salim Shital Shah Harkirat Singh Behl Xin Wang Sébastien Bubeck Ronen Eldan Adam Tauman Kalai Yin Tat Lee Yuanzhi Li

Microsoft Research

Phi 1: 1.3B params, 7B tokens x 8 epochs
Only trained for 4 days on 8 GPUs

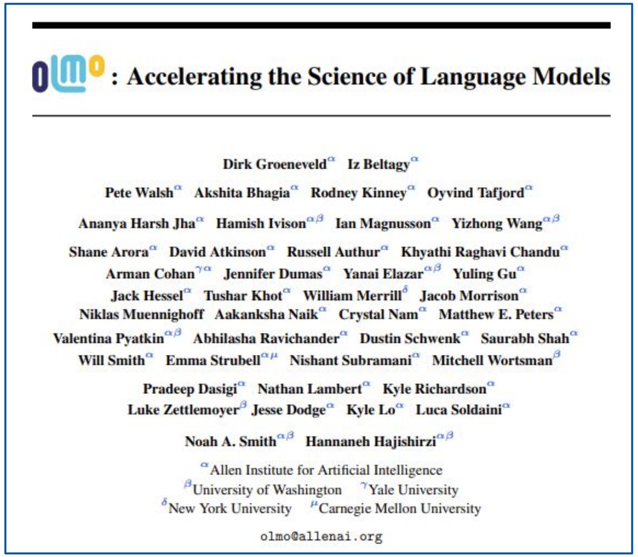
Even More Data

Things are again entering “silly” territory as datasets get bigger. It is unclear how to balance data quality and quantity; this is an active area of research.

AllenAI released OLMo in February 2024, alongside a 3T token dataset called Dolma.

In April 2024, Meta announced LLaMA 3, trained on 15T tokens.

And of course, OpenAI is no longer “open”, but they presumably are training GPT4+ on similarly absurd amounts of data.



OLMo: Accelerating the Science of Language Models

Dirk Groeneveld[α] Iz Beltagy[α]
Pete Walsh[α] Akshita Bhagia[α] Rodney Kinney[α] Oyvind Tafjord[α]
Ananya Harsh Jha[α] Hamish Ivison[αβ] Ian Magnusson[α] Yizhong Wang[αβ]
Shane Arora[α] David Atkinson[α] Russell Authur[α] Khyathi Raghavi Chandu[α]
Arman Cohan[γα] Jennifer Dumas[α] Yanai Elazar[αβ] Yuling Gu[α]
Jack Hessel[α] Tushar Khot[α] William Merrill[δ] Jacob Morrison[α]
Niklas Muennighoff Aakanksha Naik[α] Crystal Nam[α] Matthew E. Peters[α]
Valentina Pyatkin[αβ] Abhilasha Ravichander[α] Dustin Schwenk[α] Saurabh Shah[α]
Will Smith[α] Emma Strubell[αμ] Nishant Subramani[α] Mitchell Wortsman[β]
Pradeep Dasigi[α] Nathan Lambert[α] Kyle Richardson[α]
Luke Zettlemoyer[β] Jesse Dodge[α] Kyle Lo[α] Luca Soldaini[α]
Noah A. Smith[αβ] Hannaneh Hajishirzi[αβ]

[α]Allen Institute for Artificial Intelligence
[β]University of Washington [γ]Yale University
[δ]New York University [μ]Carnegie Mellon University

olmo@allenai.org

OLMo: 65B Params, 3T Tokens

Jonathan Richard Schwar @schwarzjn_ · 12h
When can we expect the Llama 3 research paper?
1 2 51 10K

Yann LeCun @ylecun · 9h
Soon.

LLaMA 3: 70B Params, 15T Tokens

Parallelization

Parallelization

Pretty quickly we have models that cannot realistically be trained on a single computer. There two problems:

Training Time: The amount of time to process billions or trillions of tokens for training on single machine becomes impractical.

Model Size: As we increase our model’s size, it will no longer fit in memory on a single GPU.

Training Time

When training our homegrown LLM, I get around 100M tokens per hour on my desktop.

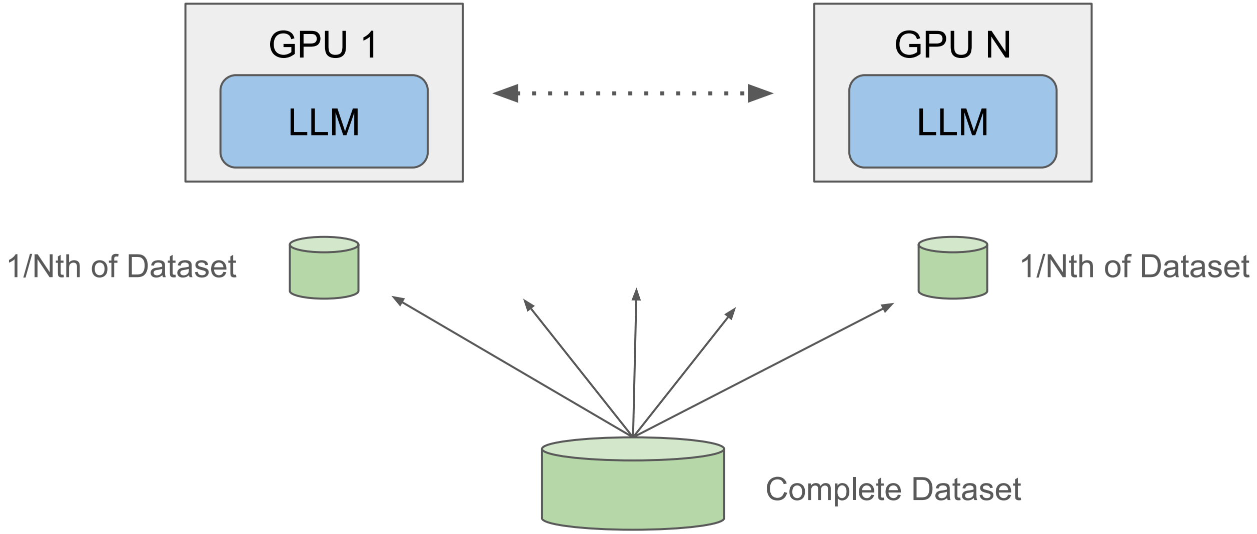
Time to train on 1T tokens: ~1.1 years

This problems gets much worse as our models get larger. If I increase the size of the model 4x to ~110M parameters, I get around 3x speed reduction.

Remember, 110M is still incredibly tiny. Training a big model on my desktop would take several decades.

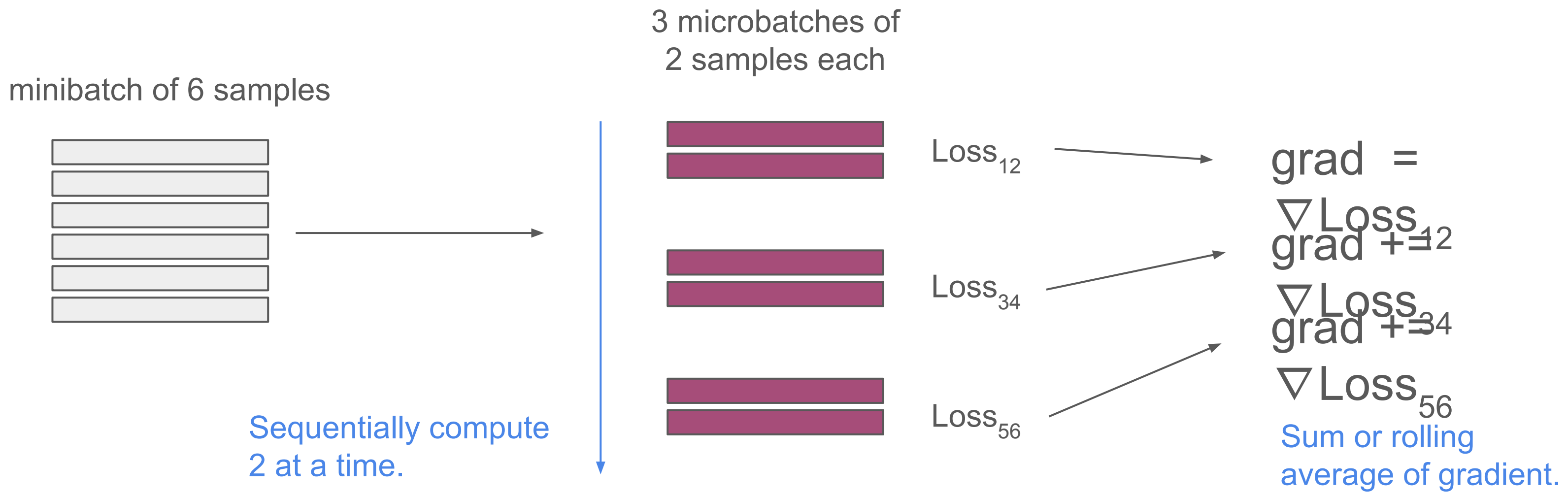
Data Parallelism

The simplest way to speed-up training is **data parallelism**. In data parallel training, each GPU has a complete copy of the LLM. Each copy gets a fraction of the dataset, and updates are shared between copies.



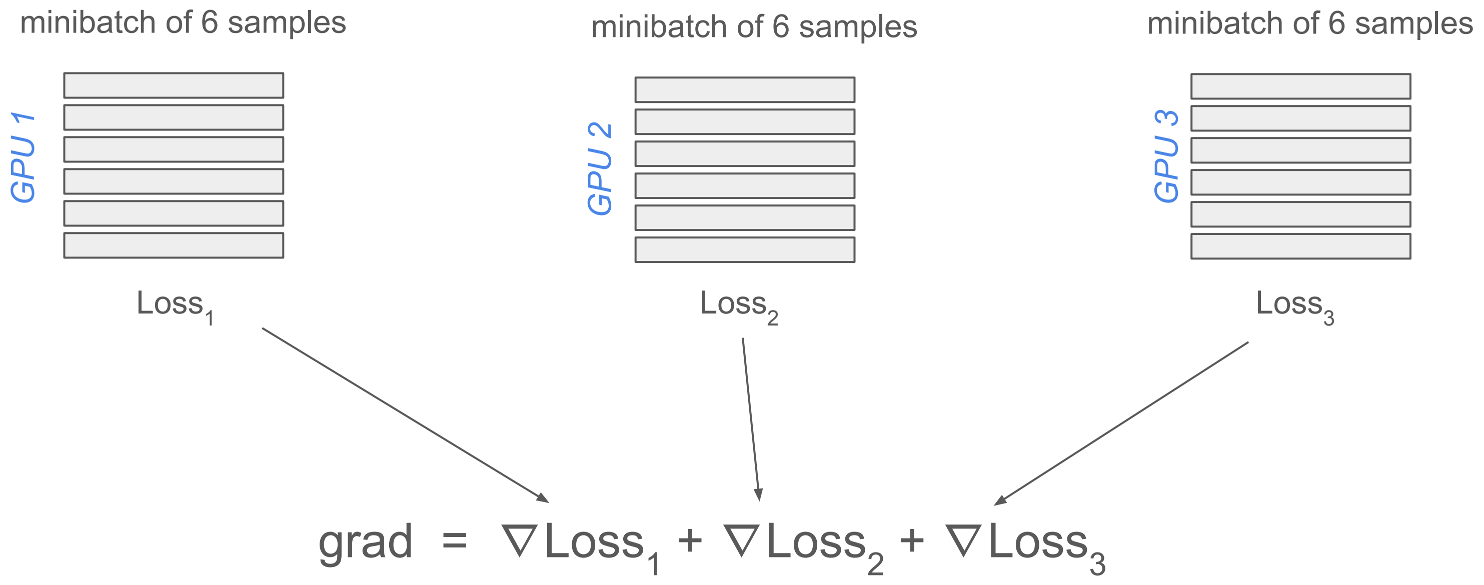
Data Parallel vs. Gradient Accumulation

Recall in *gradient accumulation*, the we may backpropagate on multiple microbatches before applying an update. We average the gradients from each microbatch.


minibatch of 6 samples
3 microbatches of 2 samples each
$Loss_{12}$
$Loss_{34}$
$Loss_{56}$
grad = $\nabla Loss_{12}$
grad += $\nabla Loss_{34}$
grad += $\nabla Loss_{56}$
Sequentially compute 2 at a time.
Sum or rolling average of gradient.

Data Parallel vs. Gradient Accumulation

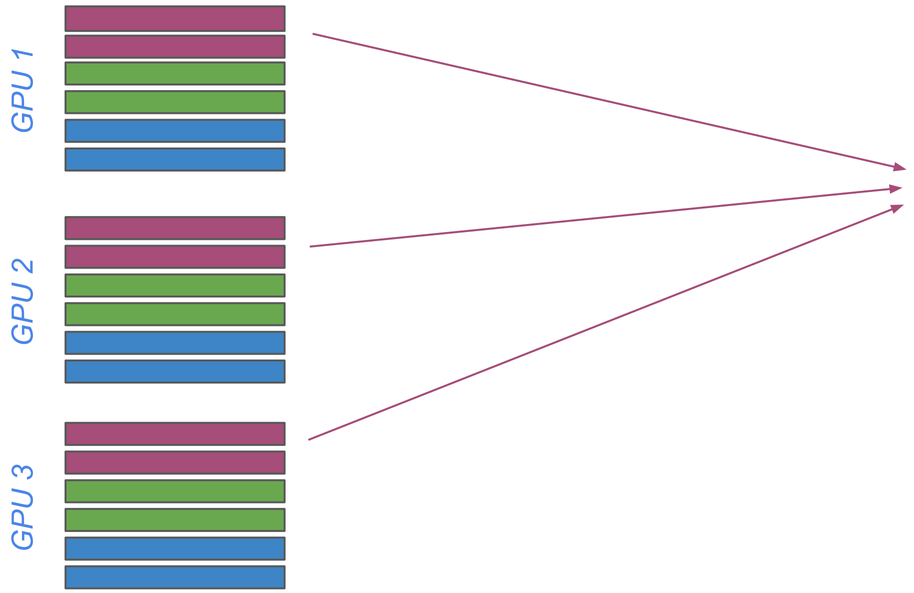
In data parallelism, we are using the same idea, but the batches are processed simultaneously (on separate machines) rather than sequentially (on one machine).



Data Parallel vs. Gradient Accumulation

Of course, we can do both at the same time if we want to:

3 x minibatches of 6 samples ⟶ 3 x (3 x microbatches of 2 samples each)



$$\text{grad} = \nabla \text{Loss}_{1\_12} + \nabla \text{Loss}_{2\_12} + \nabla \text{Loss}_{3\_12}$$

$$\text{grad} \mathrel{+}= \nabla \text{Loss}_{1\_34} + \nabla \text{Loss}_{2\_34} + \nabla \text{Loss}_{3\_34}$$

$$\text{grad} \mathrel{+}= \nabla \text{Loss}_{1\_56} + \nabla \text{Loss}_{2\_56} + \nabla \text{Loss}_{3\_56}$$

Memory Limits

If we have a smaller model and lots of training data, we can use data parallelism.

What if our model is too large to fit on a single GPU? We cannot have one copy per GPU anymore.

For example, GPT3 (175B parameters) requires about 700GB of memory just to hold the model (and substantially more to train or run inference).

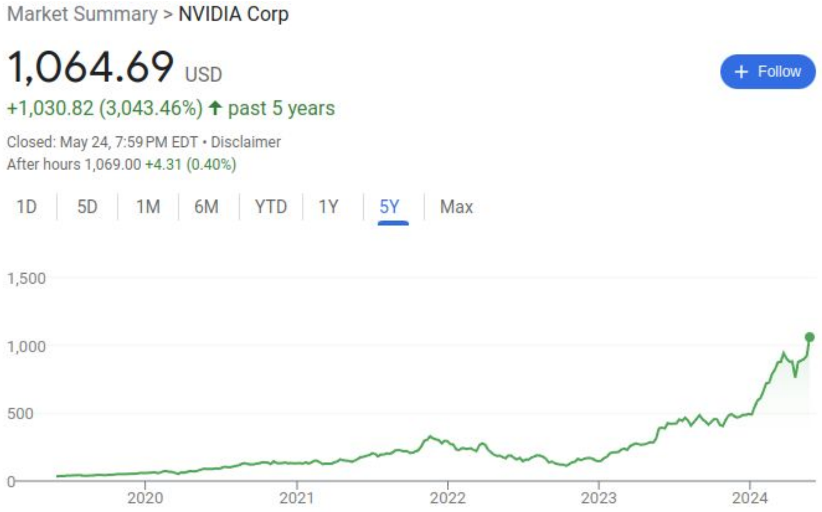
Aside: GPUs

NVIDIA

The large majority of AI applications uses NVIDIA GPUs.

This has proven extremely profitable for NVIDIA, as they are pretty much supplying the entire industry with compute ~~(and are now the 4th most valuable company on earth).~~

second?



Market Summary > NVIDIA Corp

1,064.69 USD

+1,030.82 (3,043.46%) ↑ past 5 years

Closed: May 24, 7:59 PM EDT · Disclaimer

After hours 1,069.00 +4.31 (0.40%)

Jensen Huang, NVIDIA CEO, worth about $100B

NVIDIA GPUs

NVIDIA sells a range of GPUs, generally split into two “grades”:

Consumer Grade: Meant for a single person / single computer. Generally cost in the hundreds or low thousands.

Professional Grade: Meant for compute clusters / servers. If you are in the business of LLMs this is what you use to train. Costs range from high thousands to 10s of thousands.

At any time a limiting factor on the parallelization strategy of LLM practitioners is the VRAM of the highest-end NVIDIA GPUs.

NVIDIA High-End GPUs

Architecture	**Release Date**	**Highest-End Offering**	**Max VRAM**
Pascal	2016	P100	16 GB
Volta	2017-2018	V100	32 GB
Ampere	2021-2022	A100	80 GB
Hopper	2023	H100	96 GB
Blackwell	2024 TBD	B200 (2 x B100)	192 GB

NVIDIA is constantly pushing updates and mid-cycle upgrades so this is just a rough outline.

Aside Over

Moving Targets

All of the above really boils down to one thing when discussing parallelism:

The size of a model that can fit on a single GPU is increasing exponentially. It is becoming easier and easier to get really far only using data parallelism.

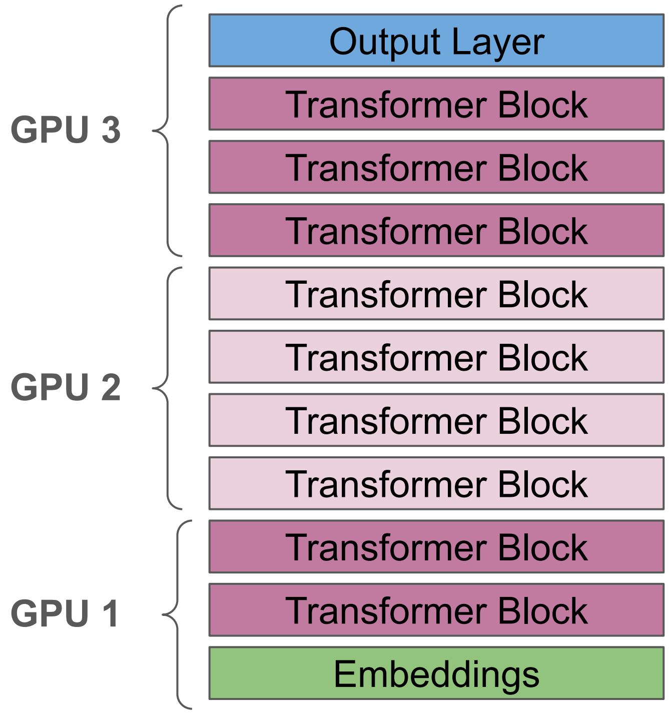
But, for the biggest models we still need other options (or looking back, the options were different at the time of release). GPT3 was trained on V100s (32GB).

Pipeline Parallelism (also called “Model Parallelism”)

If a model cannot fit on a single GPU, it can be sliced up and distributed in chunks of sequential layers (transformer blocks).

This is called pipeline parallelism.

Since our models are many layers stacked, there are lots of places we can slice them. Generally we try to load-balance across our GPUs (similar sized chunk on each GPU).



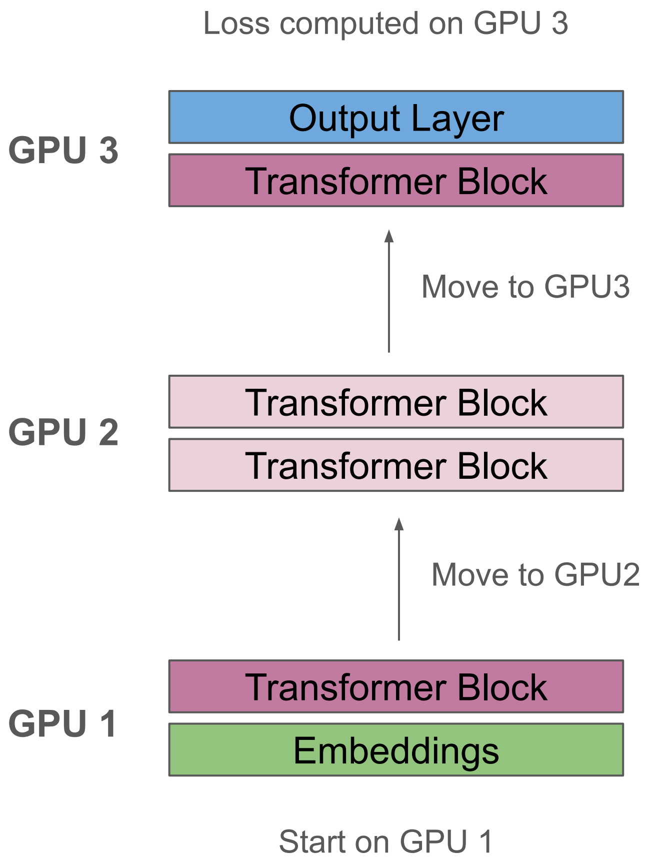
Pipeline Parallelism

When passing data through the model, it is moved between devices as needed (and similarly, backpropagation needs to pass through a few devices).

One “gotcha” is that the loss will be computed on a different GPU than where the data is loaded!

If you construct a batch of (**`input, target`**) pairs, **`target`** needs to be sent to the end.

These data movements are very low-level in the training loop and model. It is very awkward to add this to an existing model (should be built into the model from the beginning).

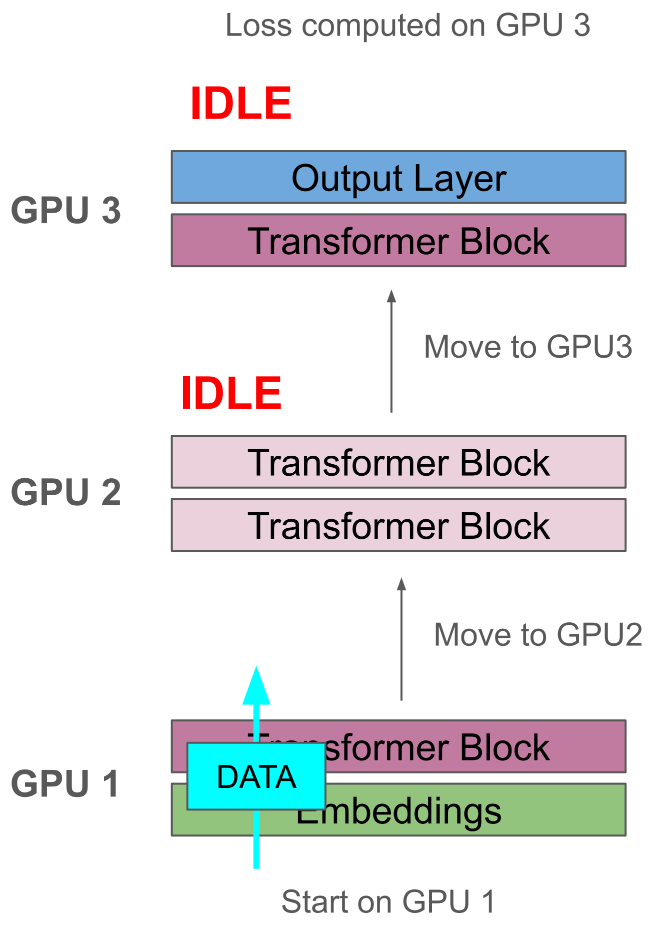


Pipeline Parallelism Inefficiencies

Since each GPU only holds a fraction of a model, you could also use this to increase your batch size and only have one model copy.

However, a lot of time the GPUs are just "waiting for data".

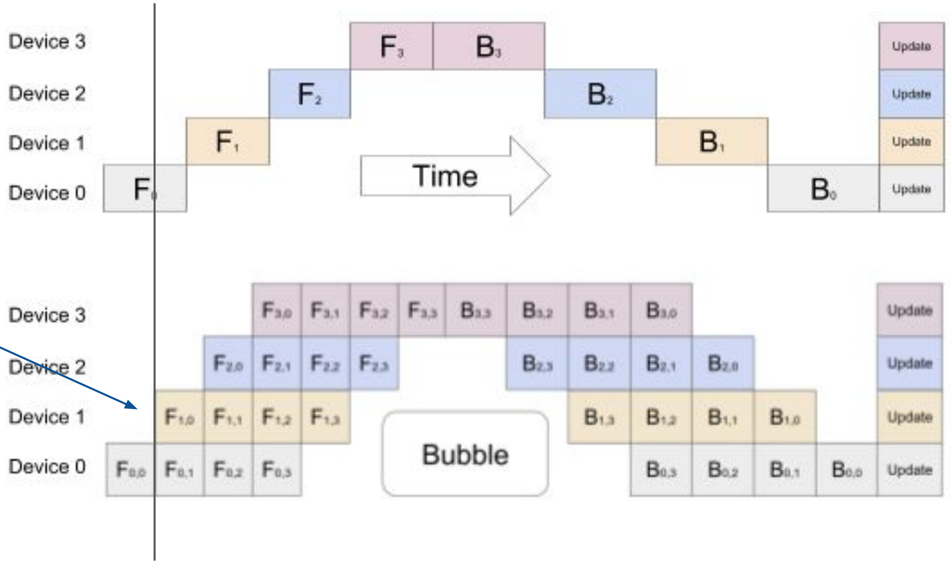
For this reason, 1 model across N GPUs is <u>much slower</u> than N copies of the model, even if each copy has a tiny batch size.



Fancy Pipeline Parallelism

There are some ways to make this a little better. A method called GPipe considers a “streaming” type of approach. Instead of doing an entire batch, a batch is subdivided so that part of it can be passed forward while the rest is computed:

Device 1 starts early because device 0 only processed a small portion of the batch. For the next portion, both devices run.



Device 3
Device 2
Device 1
Device 0
F_0
F_1
F_2
F_3
B_3
B_2
B_1
B_0
Update
Time
$F_{0,0}$ $F_{0,1}$ $F_{0,2}$ $F_{0,3}$
$F_{1,0}$ $F_{1,1}$ $F_{1,2}$ $F_{1,3}$
$F_{2,0}$ $F_{2,1}$ $F_{2,2}$ $F_{2,3}$
$F_{3,0}$ $F_{3,1}$ $F_{3,2}$ $F_{3,3}$
$B_{3,3}$ $B_{3,2}$ $B_{3,1}$ $B_{3,0}$
$B_{2,3}$ $B_{2,2}$ $B_{2,1}$ $B_{2,0}$
$B_{1,3}$ $B_{1,2}$ $B_{1,1}$ $B_{1,0}$
$B_{0,3}$ $B_{0,2}$ $B_{0,1}$ $B_{0,0}$
Bubble

Nodes

Important note on server configurations. Generally, 4 or 8 GPUs are housed together in a single “node”. Within a node, data transmission is very fast.

You can think of a node as a single machine with multiple GPUs.

If you start using more than 8 GPUs, you enter the territory of “multi-node training”, which will add more complexities, especially in moving data around.

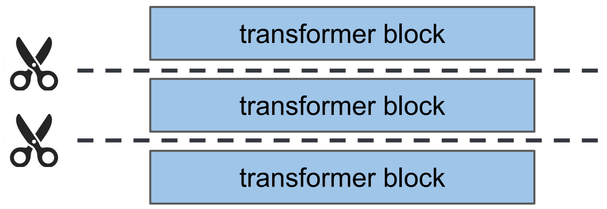
Pipeline parallel over 8 GPUs - Model is contained on one node.

Pipeline parallel over 9 GPUs - Model split across two nodes, possible bottleneck.

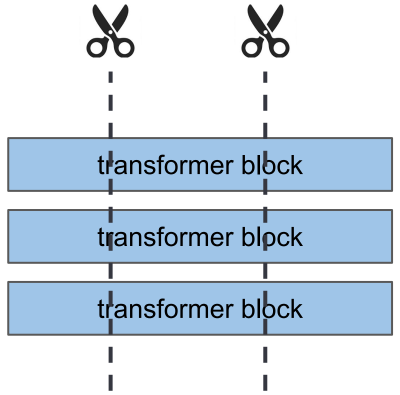
Tensor Parallelism

An alternative to pipeline parallelism is tensor parallelism, which chops up individual layers. When operating, the matrix multiplication inside the layer is distributed across several devices.

This is clearly more complex to implement than the other methods, but we do not have GPU’s waiting for data!



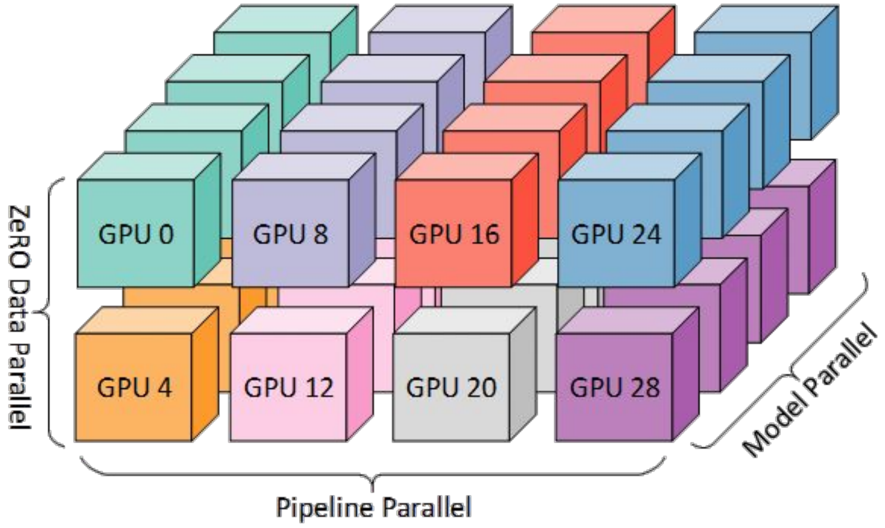
Pipeline Parallel



Tensor Parallel

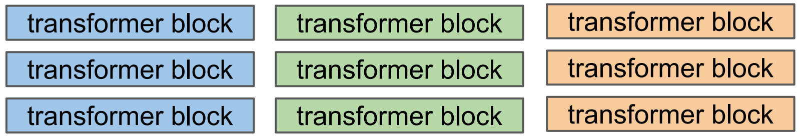
3D Parallelism (All Together)

If you have a giant compute cluster you can of course do all of this together. This is called “3D parallelism”.

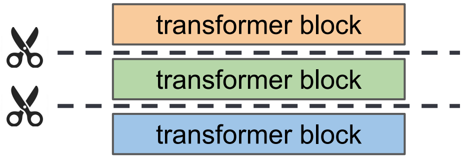


Summary

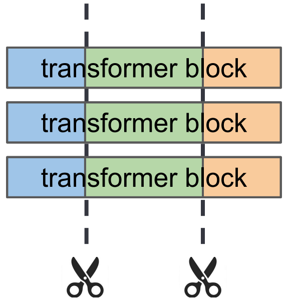
Data Parallel - Copies of the model (or part of the model if combined with other methods) each process different samples. By far the easiest method.



Pipeline Parallel - A single model bridges multiple GPUs, slicing between layers.



Tensor Parallel - A single model bridges multiple GPUs, slicing the layers themselves.



3D Parallel - All of the above.

Software

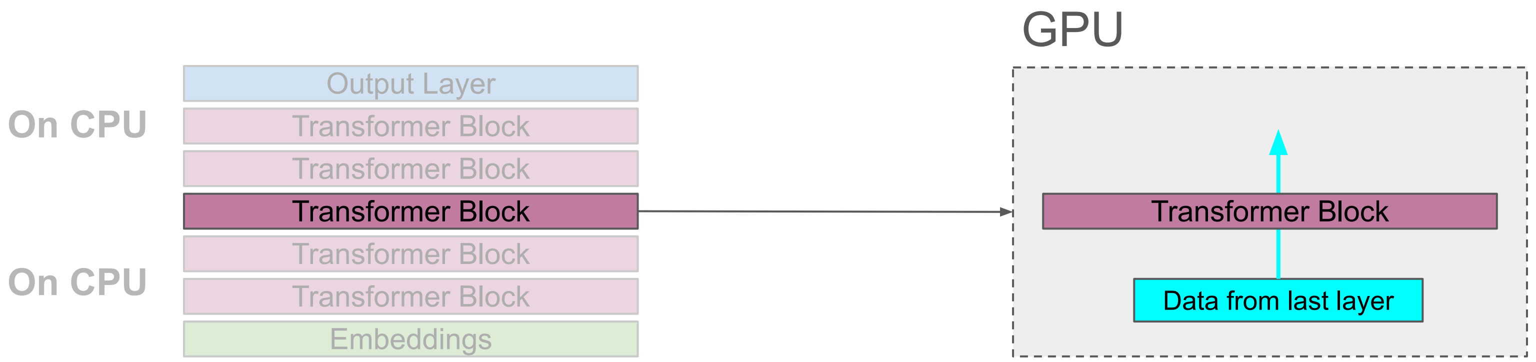
Parallelism can be a real headache to get working, and there are a handful of software packages that do the work for you (and even they can be hard to get working).

A great resource for parallelism is this HuggingFace article: https://huggingface.co/docs/transformers/v4.15.0/en/parallelism

CPU Offloading

Another method when dealing with resource constraints is “offloading” - only loading part of the model on the GPU at one time (or part of the gradient, part of the layer, etc).

This is not a type of parallelism, but can be used with parallelism (or instead of parallelism) to stretch existing compute resources.



Only part of the model is active in VRAM at a time.

Final Projects
(switch to instruction document)