

LLM for Dummies: Functionality, Scaling, and HPC Training

HPC Services, NHR@FAU

hpc-support@fau.de








<https://doc.nhr.fau.de>

Agenda

1. Introduction
2. How LLMs work: The Attention Mechanism
3. Scaling Laws
4. Training of LLMs
5. Evaluation of LLMs
6. Parallelization Techniques
7. LLMs on HPC

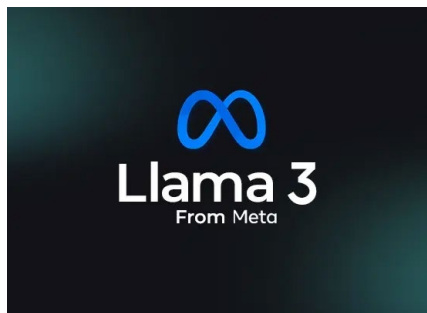
1. Introduction

Leaderboard

	Average 	MMLU (General) 	GPQA (Reasoning) 	HumanEval (Coding) 	Math 	BFCL (Tool Use) 	MGSM (Multilingual) 
OpenAI o1	85.39%	91.8%	75.7%	92.4	96.4%	66.73%	89.3%
DeepSeek-R1	-	90.8%	71.5%	-	97.3%	-	-
GPT-4o	80.5%	88.7%	53.6%	90.2%	76.6%	83.59%	90.5%
Llama 3.1 405b	80.4%	88.6%	51.1%	89%	73.8%	88.5%	91.6%
DeepSeek V3	76.24%	88.5%	59.1%	82.6%	90.2%	57.23%	79.8%
Claude 3.5 Sonnet	84.5%	88.3%	65%	93.7%	78.3%	90.2%	91.6%
Grok-2	-	87.5	56%	88.4%	76.1%	-	-

<https://www.vellum.ai/llm-leaderboard>

Opensource LLMs



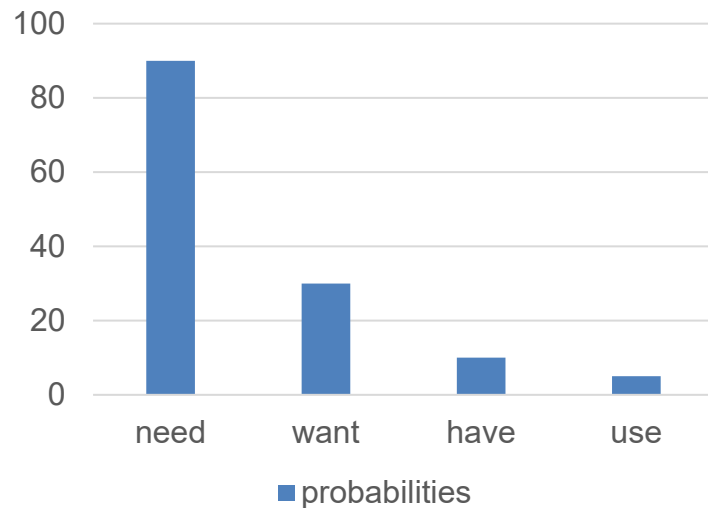
	Rank	Type	Model	Av...	IF...	B...	M...	G...	M...	M...	C...
🏆	1	🏆	CombinHorizon/zetasepic-abliteratedV2-Qwen2.5	46.76 %	83.28 %	56.83 %	58.53 %	15.66 %	14.22 %	52.05 %	7.37 kg
🏆	2	🏆	CombinHorizon/huihui-ai-abliterated-Qwen2.5-32	45.66 %	82.06 %	56.04 %	59.44 %	11.86 %	12.09 %	52.45 %	26.00 kg
🏆	3	🏆	FINGU-AI/RomboUltima-32B	44.73 %	66.72 %	56.67 %	53.85 %	16.22 %	21.72 %	53.21 %	7.80 kg
🏆	4	🏆	EVA-UNIT-01/EVA-Qwen2.5-72B-v0.2	44.22 %	68.79 %	59.07 %	43.13 %	21.14 %	19.73 %	53.48 %	45.91 kg
🏆	5	💬	Cran-May/tempotacilla-cinerea-0308	43.64 %	80.85 %	50.60 %	55.51 %	14.99 %	12.67 %	47.22 %	3.64 kg
🏆	6	🏆	Daemontatox/Llama3.3-70B-CogniLink	42.77 %	69.31 %	52.12 %	41.39 %	26.06 %	21.40 %	46.37 %	32.38 kg

https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard#/

2. How LLMs work: The Attention Mechanism

Attention

HPC is all we _____



Attention

HPC	is	all	we	????
-----	----	-----	----	------



5.8
4.2
9.9
3.7
.
.
.
2.1



6.2
7.1
8.3
5.7
.
.
.
0.1



2.8
3.5
4.2
8.1
.
.
.
1.4



2.0
8.2
5.2
3.9
.
.
.
8.1

Token Embeddings

- By using tokens rather than full words, GPT models can achieve a balance between linguistic flexibility and computational efficiency
- Still problems with numbers and code
- Tokenizer matter a lot

Attention

She decided to **set** the table for dinner before the guests arrived.
(To arrange or place something)

The sun began to **set**, painting the sky with hues of orange and pink. *(To sink below the horizon)*

He bought a **set** of tools to fix the broken chair. *(A group of items)*

Attention

She decided to **set**


$$\begin{bmatrix} 2.8 \\ 3.2 \\ 4.9 \\ 6.7 \\ . \\ . \\ . \\ 3.1 \end{bmatrix}$$

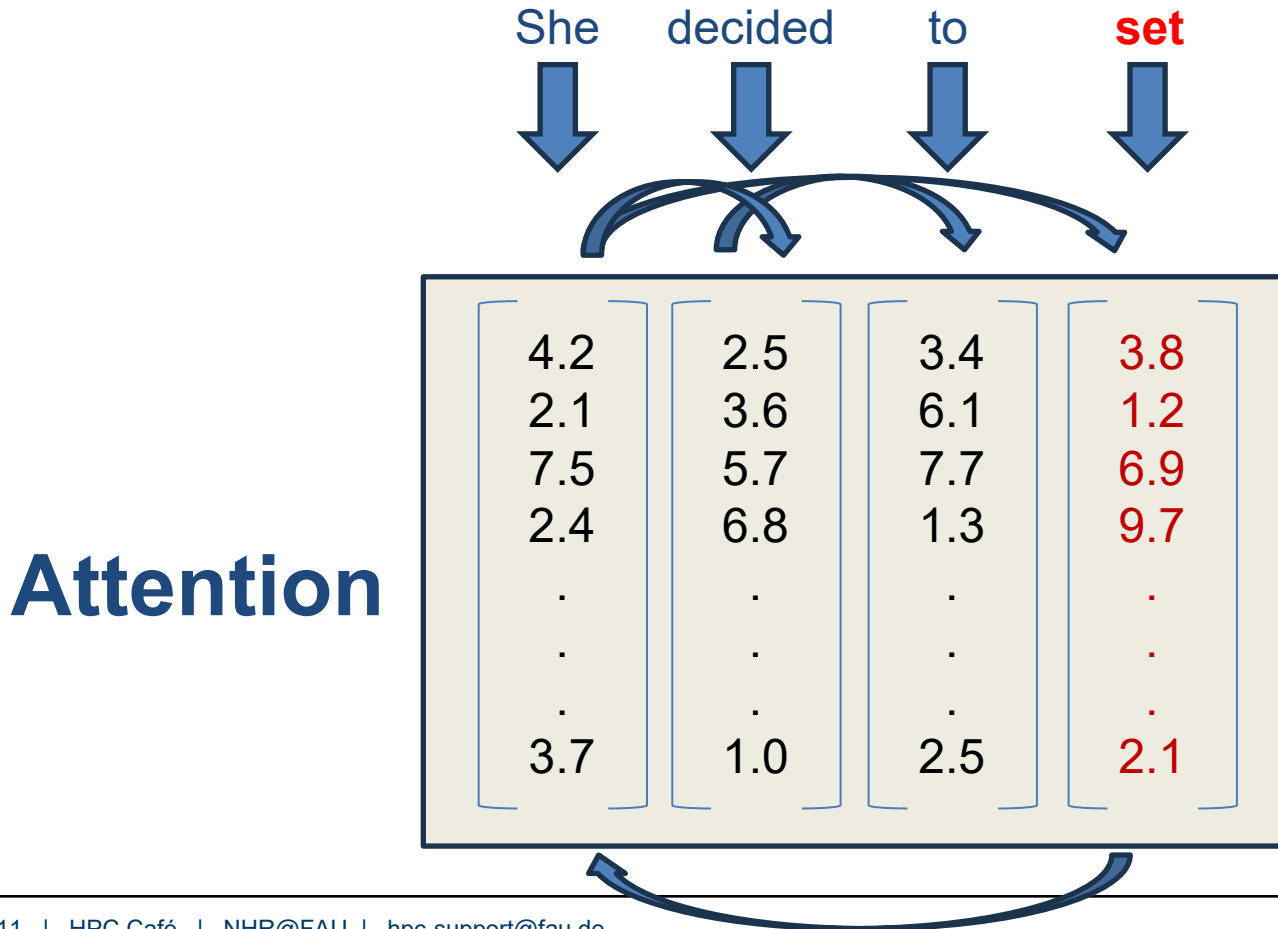
The sun began to **set**, painting


$$\begin{bmatrix} 2.8 \\ 3.2 \\ 4.9 \\ 6.7 \\ . \\ . \\ . \\ 3.1 \end{bmatrix}$$

He bought a **set** of

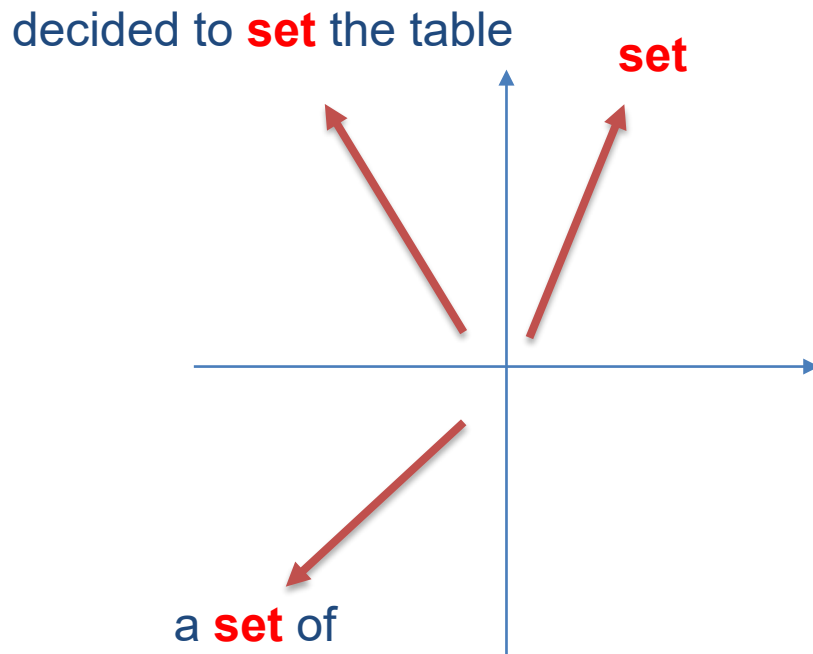

$$\begin{bmatrix} 2.8 \\ 3.2 \\ 4.9 \\ 6.7 \\ . \\ . \\ . \\ 3.1 \end{bmatrix}$$

Attention



Attention

- Embedding vectors get influenced by surrounding words through attention



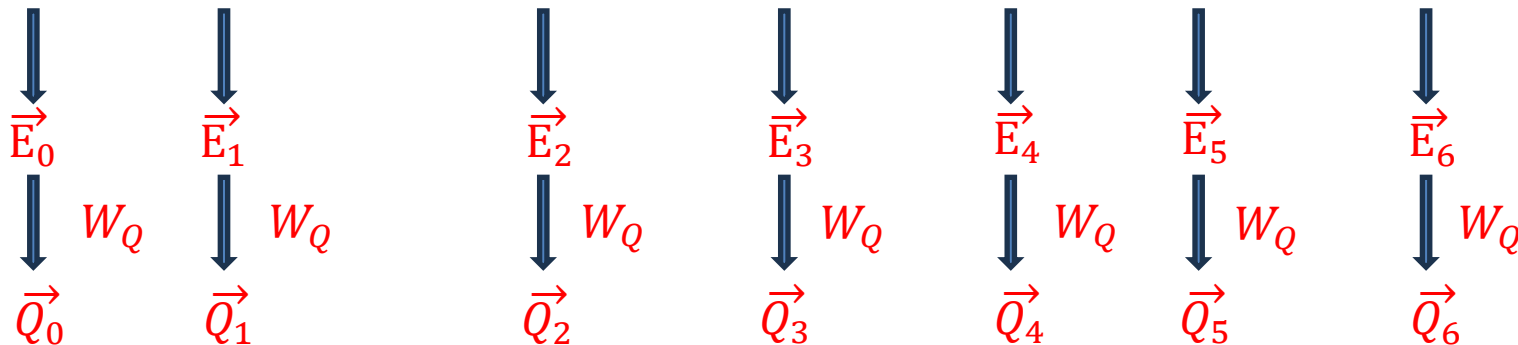
Example

- Imagine the input text is an entire crime story all the way up to a point towards the end. "The reason of his death was ????"
- The final word of the sequence is: **was**
- The embedding vector of "was" has been updated by all the other words before
- Only that's why the model can accurately predict the next word

Attention



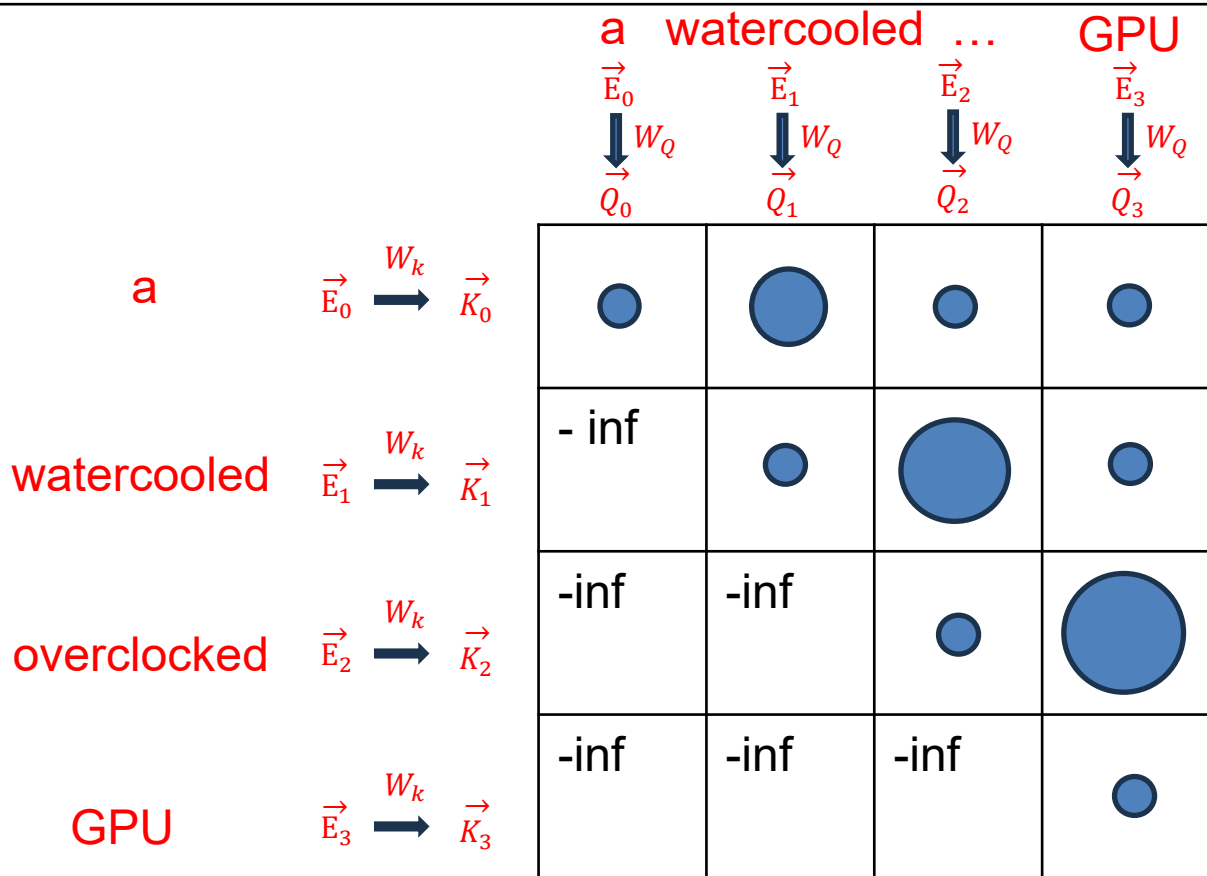
a	watercooled	overclocked	gpu	increased	the	performance
---	-------------	-------------	-----	-----------	-----	-------------



$$\begin{bmatrix} & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \end{bmatrix} W_Q \begin{bmatrix} \vec{E}_i \end{bmatrix} = \begin{bmatrix} \vec{Q}_i \end{bmatrix}$$

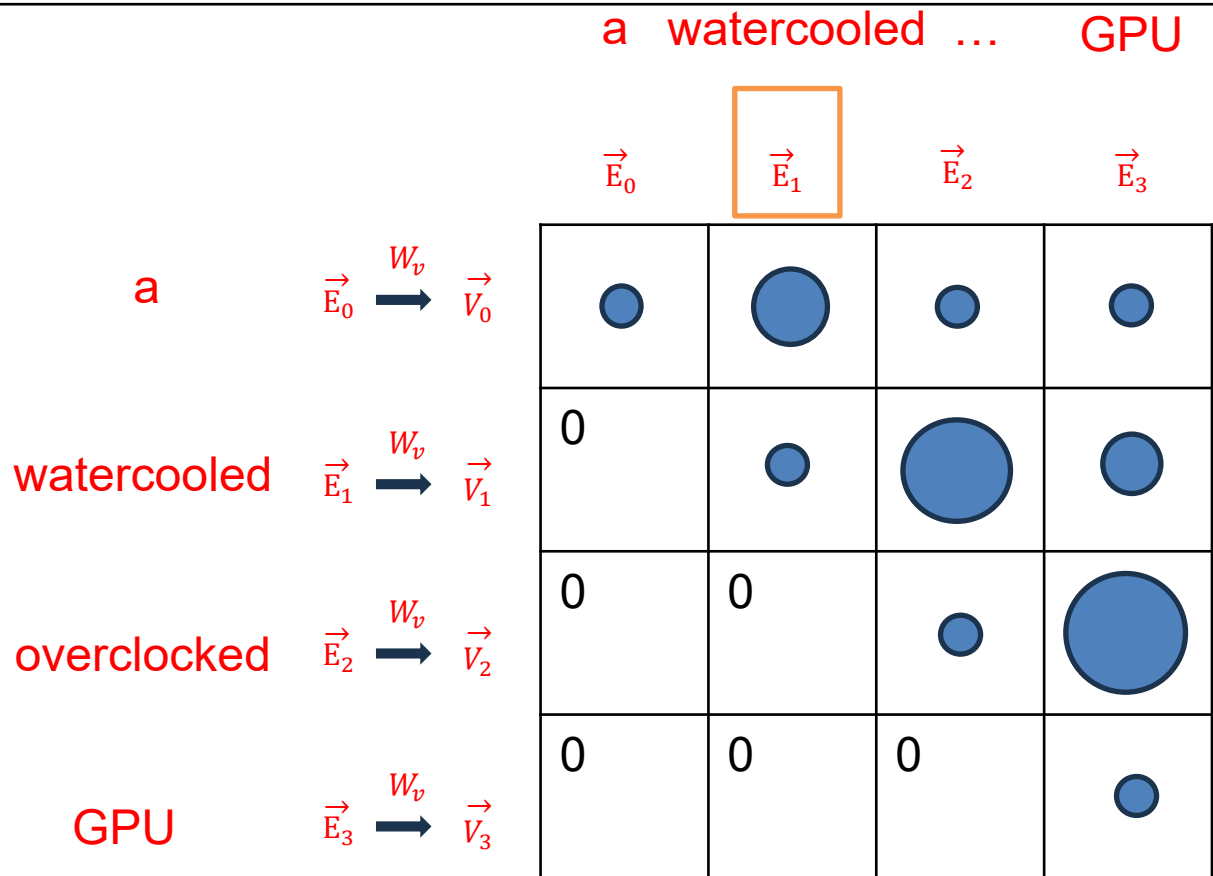
- What comes before me?
- Embeddings get mapped to Q vectors
- Q vector is in a smaller dim. space

Attentionpattern



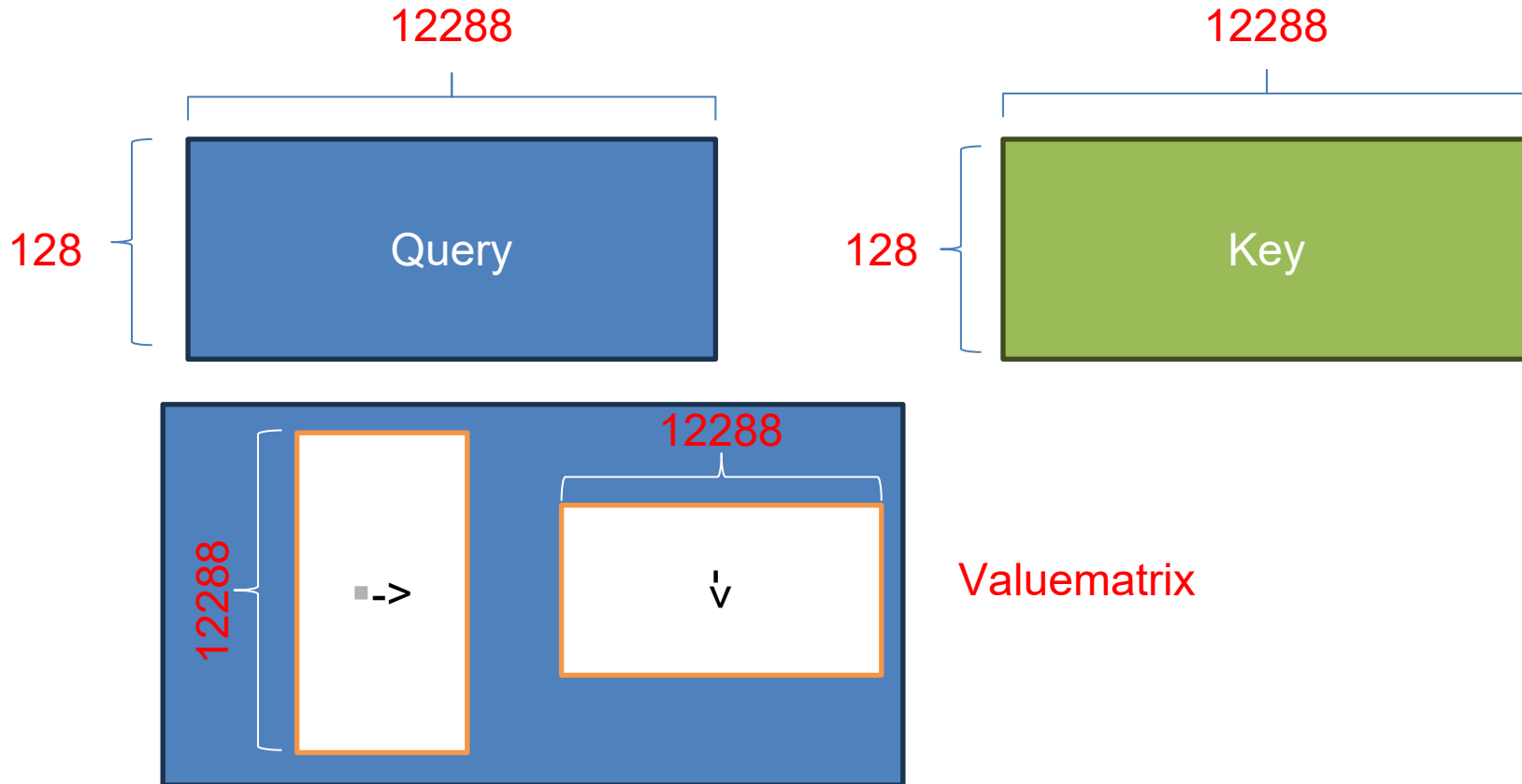
- Dotproduct K and Q
- Big values -> relevance
- **Size of this matrix is equal to the square of context size**
- Apply softmax normalization (normalized exponential function)
- -inf will turn into 0 after normalization
- We want to prevent later tokens to influence earlier ones

Valuevector



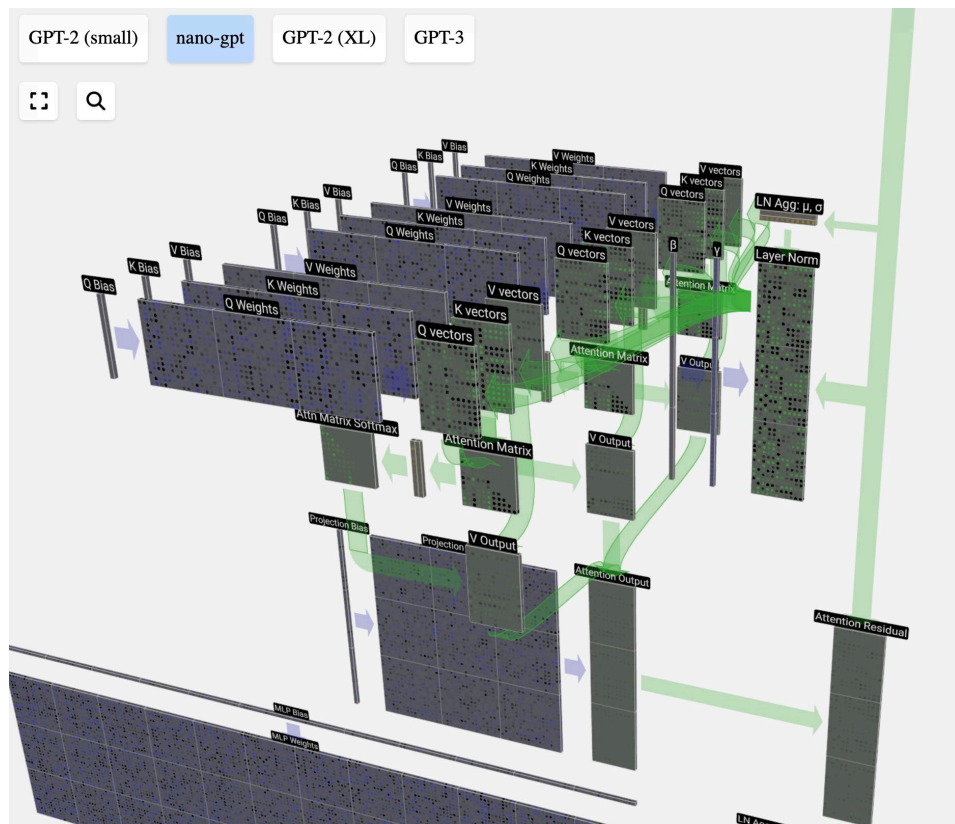
- Which words are relevant to which other words
- Values are getting added to original embedding producing a sequence of changes
- More refined embeddings are the result

Example GPT3



Multi-Head-Attention

- How can we run these operations in parallel?
- GPT3 has 96 attention heads in each attention block
- 96 distinct key and query matrices
- 96 distinct attention patterns
- Each head has his own distinct value matrices producing 96 sequences of value vectors
- They are all added together using the attention patterns as weights
- For each token and position, everyone of these heads proposes a change to the embeddings
- We sum together all the proposed changes and add them together to the embedding



<https://bbycroft.net/llm>

Multi-Head-Attention

- **Each Head Sees All Tokens:** Every attention head processes the entire input sequence rather than just a subset of tokens.
- **Distinct Q/K/V Matrices:** Each head has its own learned linear transformations for Queries, Keys, and Values.
- **Different Specializations:** Because the heads have separate weights, they naturally learn different “roles” or patterns of attention.
- **Random Initialization Leads to Diversity:** Heads start with different random weights, so they converge on diverse solutions.
- **Shared Final Output:** The model sums the contributions from all heads (after concatenation and projection) to produce the final token embeddings.

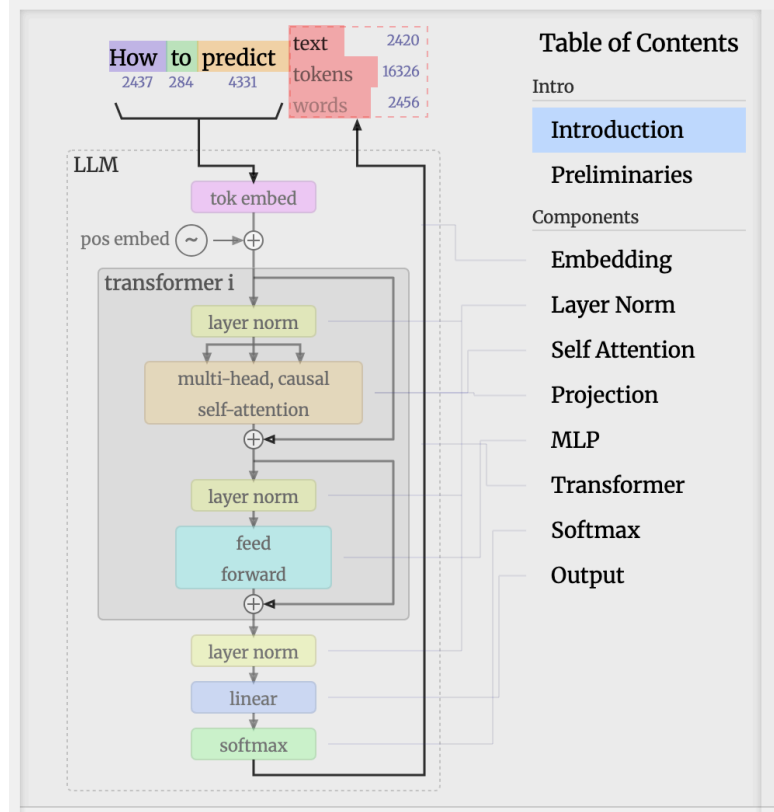


Table of Contents

Intro

Introduction

Preliminaries

Components

Embedding

Layer Norm

Self Attention

Projection

MLP

Transformer

Softmax

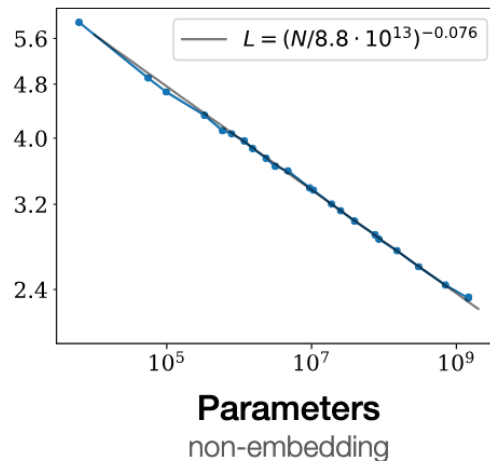
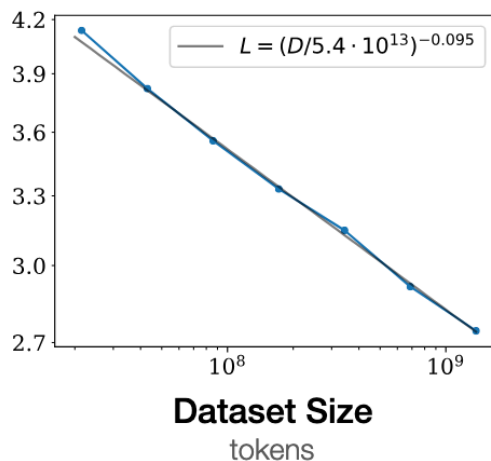
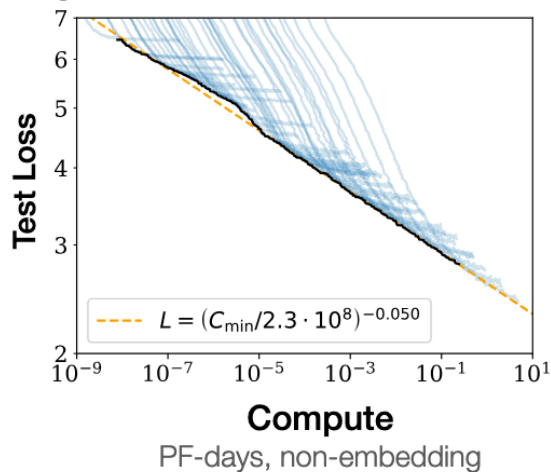
Output

<https://bbycroft.net/llm>

3. Scaling of LLMs

Scaling Laws

- More data, more parameters => better performance
- Idea: predict model performance based on amount of data and parameters
- How well will we perform when we add compute and data?
- No signs of plateau!



Scaling Laws for Neural Language Models

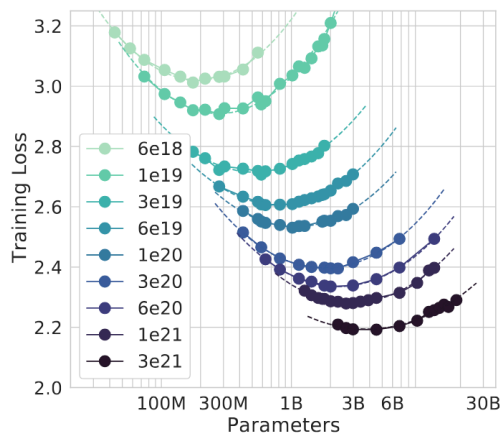
Why are scaling laws important?

Question: you will have 2.000 GPUs next year, what model do you train?

Find scaling recipes:

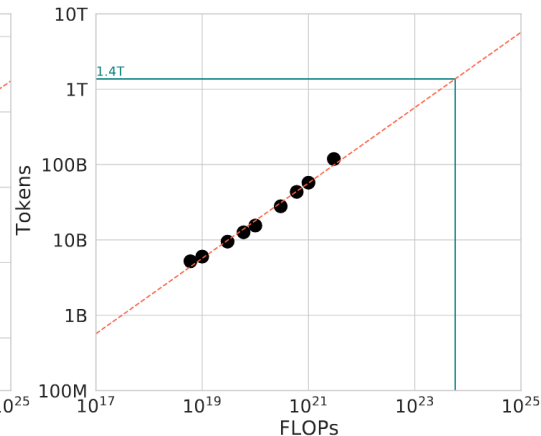
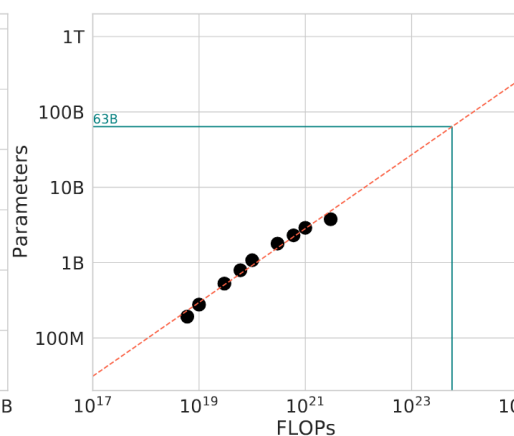
- Compute is constant on the figure
- You adapt number of tokens and parameters

■ Optimal would be
20 tokens per
parameter



Isoflop and vary tokens/parameters

Best parameters for each isoflop



Best tokens for each isoflop

Scaling law

- “The only thing that matters in the long run is the leveraging of computation” Sutton 2019
- Training Llama 405B
 - Data 15T tokens, Parameters 405B
- $\text{Flops } 6 \text{ NP} = 6 \times \text{NP} \times \text{Data} = 6 * 15,6\text{e}12 * 405\text{e}9 = 3.8\text{e}25 \text{ FLOPS}$
- Compute 16k H100 with average of 400 TFLOPS
- $\text{Time } 3.8\text{e}25 / (400\text{e}12 * 3600) = 26\text{M GPU hour} / (16\text{e}3 * 24) = 70 \text{ days}$
 - From paper they said 30M GPU hours
- $\text{Costs rented GPU} + \text{Salary} = 2\$/\text{h } 26\text{Mh} + 500.000\$ \text{ per Employee (50)} = 75\text{M\$}$
- Next model 10x?

4. Training of LLMs

Types of trainings

- Pretraining
- Finetuning
 - Fullfinetuning (SFT)
 - RLHF, DPO, GRPO (Preference Optimization)
 - PEFT (Parameter Efficient Finetuning)
 - Adapters like LoRA

Pretraining

- Idea: Use all data available (internet)
- Download everything
 - Extract from HTML
 - Clean Data from undesired material
 - Deduplication
 - Filtering (NSFW)
 - Datamix
 - Quality ranking (Wikipedia)
 - Format: Parquet files, JSONL
- Right now best models are trained with 15T Tokens
 - Fineweb-edu Dataset
 - Usually Wikipedia is ranked high
 - Entertainment ranked low
 - GPT4 probably trained 13T tokens

Example Llama 2


- 2 trillion tokens of data (good performance–cost trade-off)
- Chunked into fixed size (2048 or 4096 for example)
- Trained with Megatron
- NVIDIA A100 Clusters

		Time (GPU hours)	Power Consumption (W)	Carbon Emitted (tCO ₂ eq)
LLAMA 2	7B	184320	400	31.22
	13B	368640	400	62.44
	34B	1038336	350	153.90
	70B	1720320	400	291.42
Total		3311616		539.00

[Llama 2: Open Foundation and Fine-Tuned Chat Models](#)

Instructiontuning

- Instruction Datasets
- Most of them created synthetically and also filtered afterwards
- Reranked and filtered collection of datasets with a focus on instruction following
- Type: Conversation, Input/Output

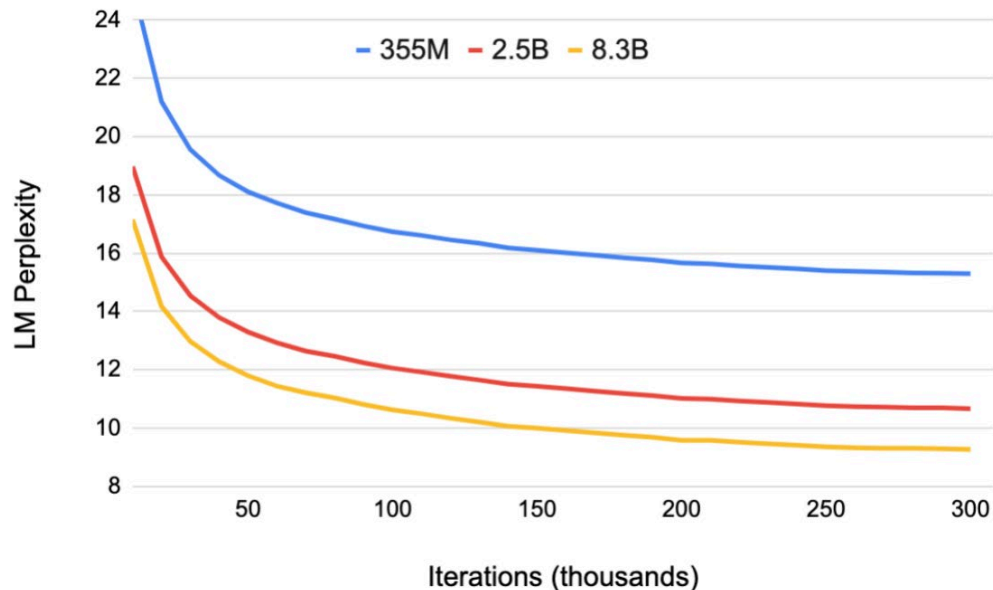
Dataset Viewer	
Auto-converted to Parquet <> API Embed View in Dataset Viewer	
Split (1) train · 1.75M rows	▼
Search this dataset	
conversations list · lengths	
	
[{ "from": "human", "value": "In coding theory and cryptography, how can I efficiently solve modular arithmetic involving powers and primes or non-prime moduli by hand?" }, { "from": "gpt", "value": "To solve such problems ef	
[{ "from": "human", "value": "Let \mathfrak{p} be a prime ideal in a Dedekind domain \mathfrak{A} , and v an absolute value on \mathfrak{A} . Consider a finite, separable field extension \mathfrak{E} of \mathfrak{A} 's quotient field \mathfrak{K} , and let \mathfrak{B} be the integral cl	
[{ "from": "human", "value": "Determine the basis for the set of all 2×2 matrices W defined as follows: $W = \begin{pmatrix} a & b \\ c & a + b + c \end{pmatrix}$ where $a, b, c \in \mathbb{R}$ }]	
[{ "from": "human", "value": "Can we infer that the generalized momentum corresponding to a non-cyclic coordinate is conserved?" }, { "from": "gpt", "value": "Yes, generally, if a coordinate is not cyclic in the Lagrangian, it imp	

5. Evaluation of LLMs

Evaluation of LLMs

- “As the model size increases, the validation perplexity decreases and reaches a validation perplexity of 9.27 for the 8.3B model”
- “We observe the trend that increasing model size also leads to lower perplexity on WikiText103 and higher cloze accuracy on LAMBADA”

* Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism
<https://arxiv.org/pdf/1909.08053>



- Basically Perplexity is the validation loss
 - avg per token loss (independent of length)
 - Exponentiate
 - Perplexity between 1 and size of vocab
 - Number of tokens you are hesitating between

Evaluation of LLMs

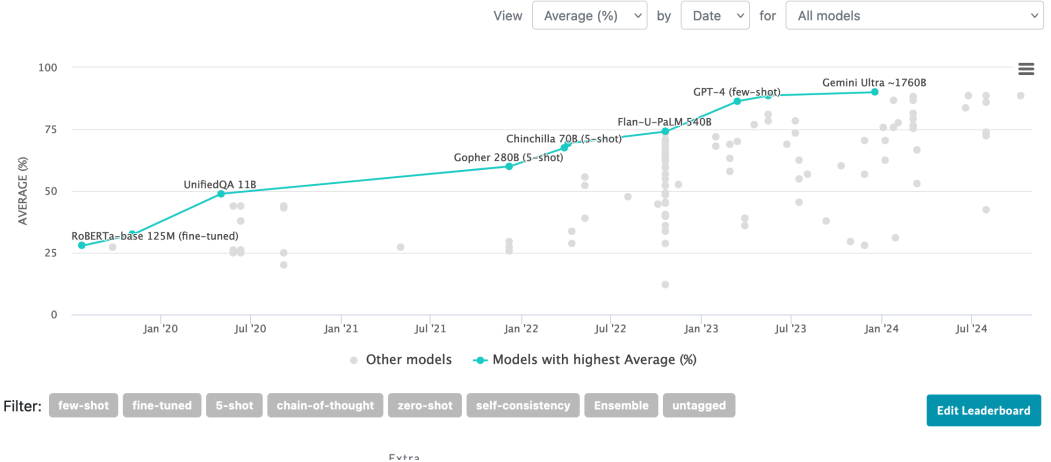
- Example MMLU
- Questions of many different domains
- What is true for a type 1a supernova?
 - a)...
 - b)...

Multi-task Language Understanding

Multi-task Language Understanding on MMLU

Leaderboard

Dataset



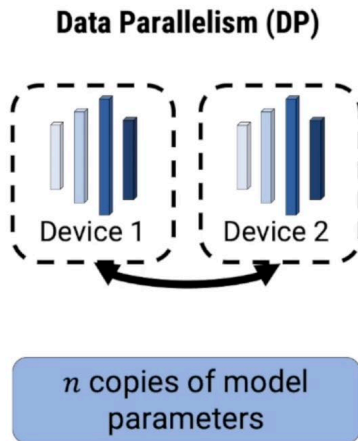
6. Parallelism Techniques

Parallelism

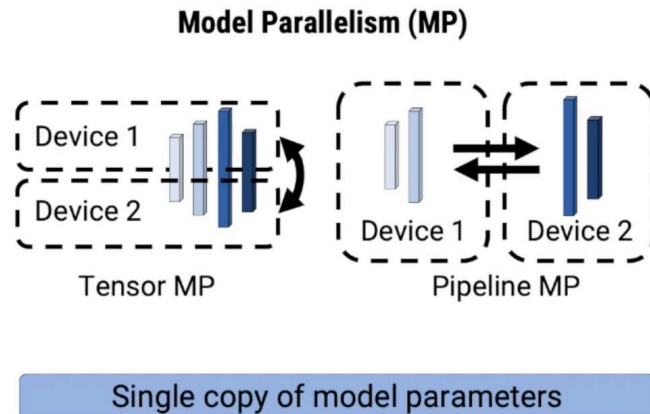
1. Data Parallelism (DP)
2. TensorParallelism (TP)
3. PipelineParallelism (PP)
4. Zero Redundancy Optimizer (ZeRO)
5. Expert Parallelism - Mixture-Of-Experts (MoE)

Data Parallelism

- **Data Parallelism**
 - Split up the batch data into smaller chunks and give each model on device one



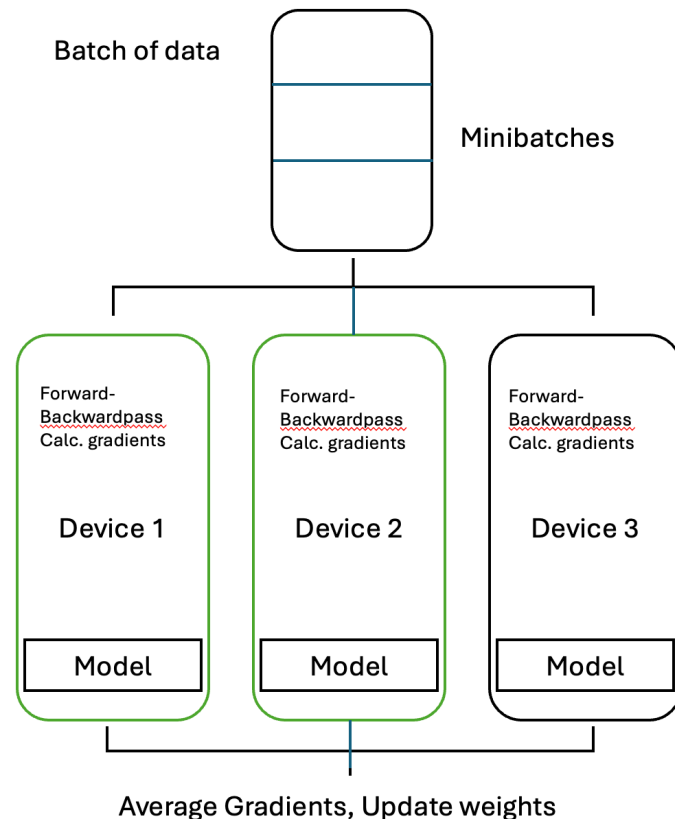
PARALLEL TRAINING



[Megatron-LM](#)

Data Parallelism

- Batch gets split up between models
 - Weight updates combined by Allreduce
 - Shortcomings:
 - Cannot scale infinitively like 3000 gpu -> Batchsize has to be big enough for example
 - Model doesn't fit on a single Device
- ➔ That's why we need Model Level Parallelism
- ➔ Splits up the layer of a model and calculates only a part of the input
 - ➔ That's Tensor level Parallelism



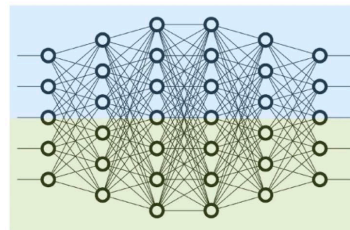
Modelparallelism

TP and PP

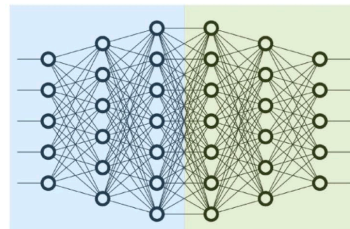
TP - each tensor is split up, each shard of the tensor resides on its designated gpu. This is what one may call horizontal parallelism, as the splitting happens on horizontal level

PP - the model is split up vertically (layer-level) across multiple GPUs, so that only one or several layers of the model are placed on a single gpu. Each gpu processes in parallel different stages of the pipeline and working on a small chunk of the batch

- Tensor (Intra-Layer) Parallelism
 - Split individual layers across multiple devices
 - Both devices compute different parts of Layer 0,1,2,3,4,5

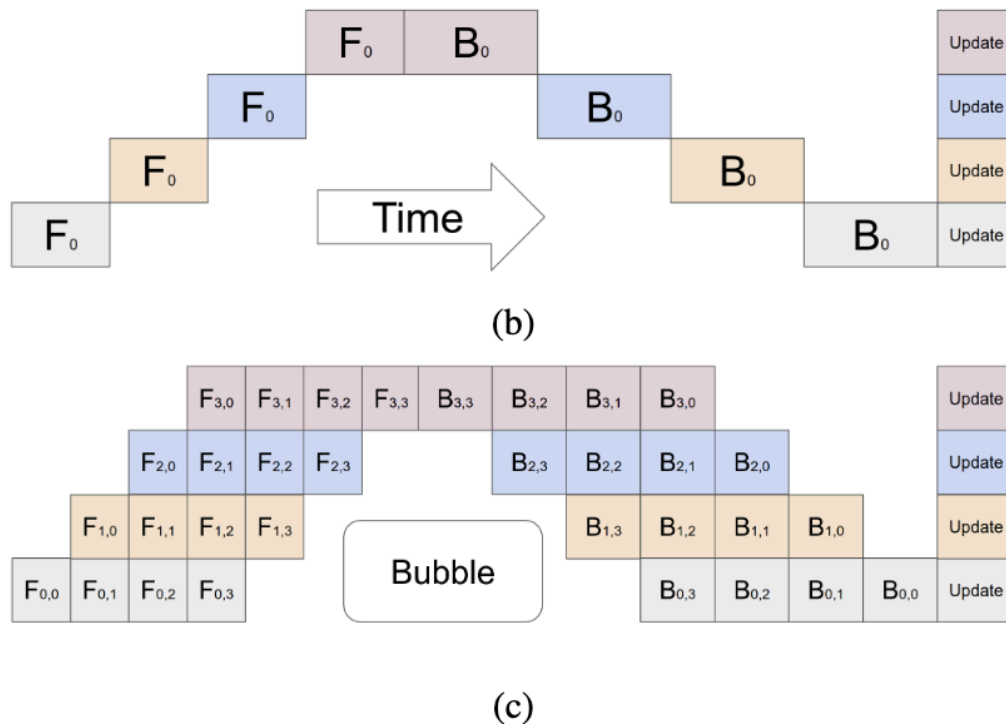
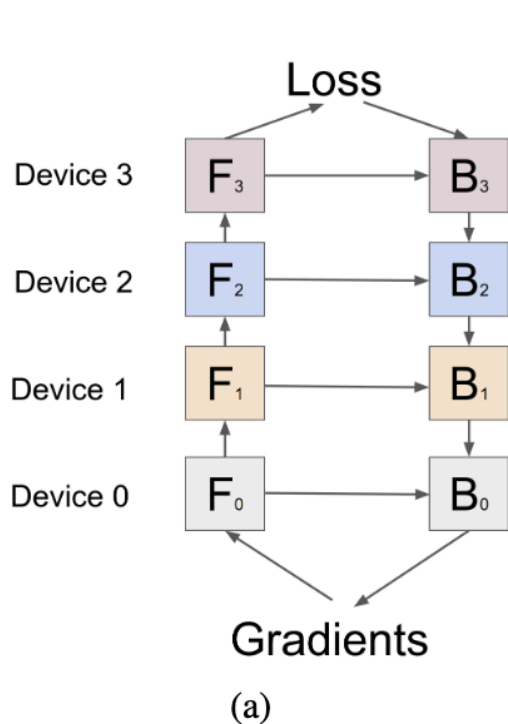


- Pipeline (Inter-Layer) Parallelism
 - Split sets of layers across multiple devices
 - Layer 0,1,2 and layer 3,4,5 are on different devices



[Megatron-LM](#)

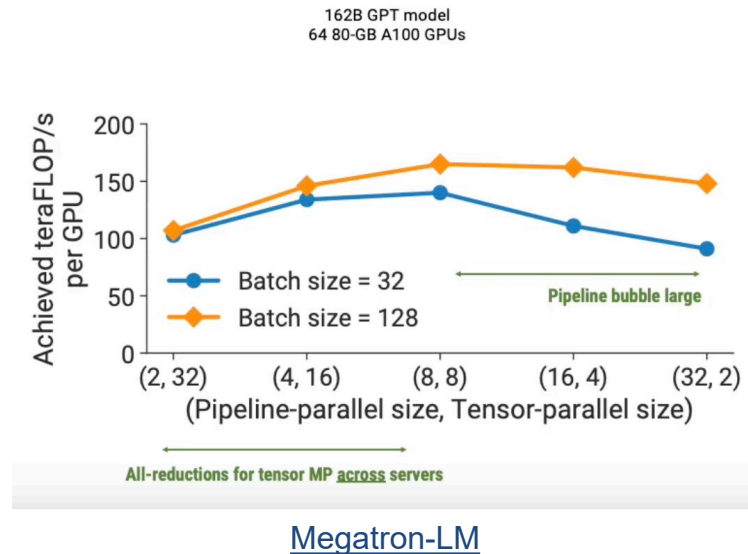
Pipeline Parallelism



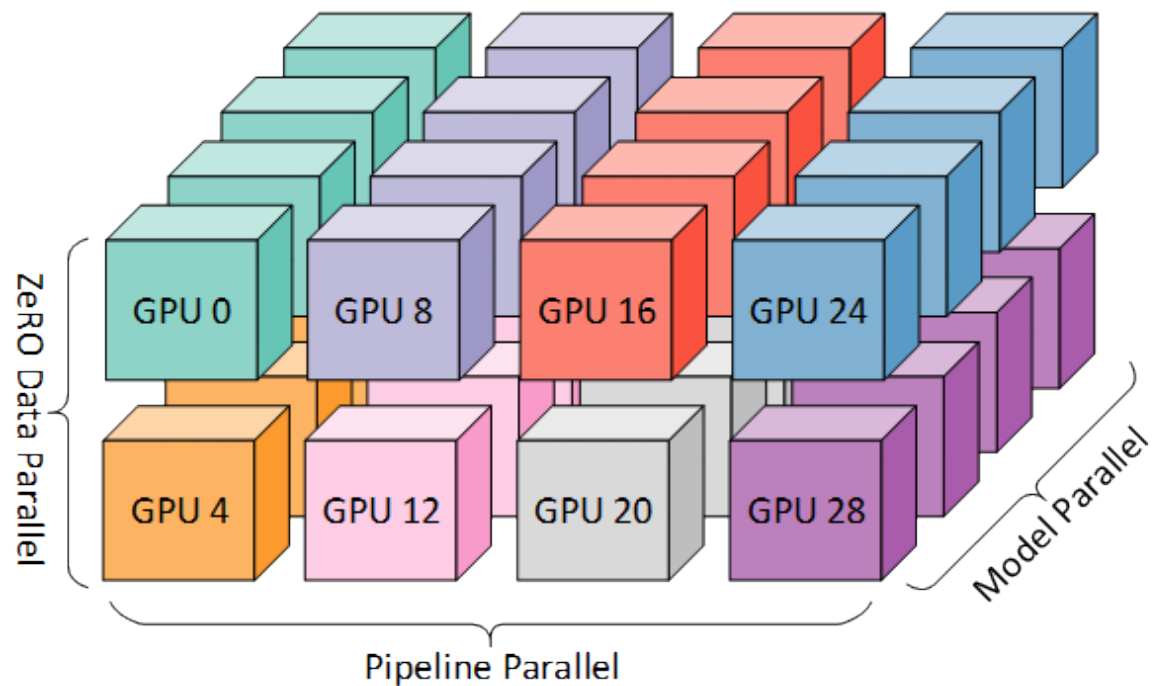
Megatron-LM

TP and PP optimization

- TP is much more communication intensive because of allreduces
- Not want TP across Nodes
- Sweetspot is maximizing TP for a Node (Number of GPUs per Node)
- Pipeline stages shouldn't be too many
 - Bubbles increase
- Schedule of forward and backward passes



3D Parallelism



[Microsoft deepspeed](#)

Zero?

- is just the usual DataParallel (DP)
- except, instead of replicating the full model params, gradients and optimizer states, each GPU stores only a slice of it
- at run-time when the full layer params are needed just for the given layer, all GPUs synchronize to give each other parts that they miss - this is it

MoE Models

- **Mixture of Experts (MoE) in Large Language Models (LLMs)**

- **Purpose:** Optimize compute efficiency for large-scale LLMs by activating only relevant model parts.

- **Advantages:**

- **Compute Efficiency:** Optimizing processing power.
- **Scalability:** Supports large-scale training while reducing memory bandwidth demands.

- **Challenges:**

- **Memory Inefficiency:** Higher VRAM usage compared to dense models.
- **Fine-tuning Complexity:** Trickier to fine-tune.
- **Best Use Cases:** Benefits are maximized in large datacenter environments.

7. LLMs on HPC

Examples single GPU

- Model fits onto a single GPU:
 - Normal use
- Model doesn't fit onto a single GPU:
 - ZeRO + Offload CPU and optionally NVMe



[Nvidia A100](#)

Examples single node

- Model fits onto a single GPU:
 - DDP - Distributed DP
 - ZeRO - may or may not be faster depending on the situation and configuration used
- Model doesn't fit onto a single GPU:
 - PP
 - ZeRO
 - TP
- With very fast intra-node connectivity of NVLINK or NVSwitch all three should be mostly on par, without these PP will be faster than TP or ZeRO. The degree of TP may also make a difference. Best to experiment to find the winner on your particular setup.
- TP is almost always used within a single node. That is TP size \leq gpus per node.

Examples multinode

- If the model fits into a single node first try ZeRO with multiple replicas, because then you will be doing ZeRO over the faster intra-node connectivity, and DDP over slower inter-node
- When you have fast inter-node connectivity:
 - ZeRO - as it requires close to no modifications to the model
 - PP+TP+DP - less communications, but requires massive changes to the model
- when you have slow inter-node connectivity and still low on GPU memory:
 - DP+PP+TP+ZeRO-1

Training-Frameworks

- **NVIDIA Megatron-LM:** This framework is predominantly used for pretraining large language models, leveraging advanced model parallelism techniques to facilitate efficient large-scale training. Often times forked. Other frameworks like HF Nanotron are inspired by Megatron.
- **Lit-GPT:** Designed as a minimalistic implementation in PyTorch, Lit-GPT serves research purposes, allowing for straightforward experimentation with various LLM architectures.
- **Axolotl:** Focused on the fine-tuning aspect of LLMs, Axolotl provides a user-friendly interface that enables users to adapt pre-trained models to specific tasks effectively.
- **Unsloth:** Optimized for single-node setups, Unsloth offers hyper-optimized QLoRA fine-tuning for single GPUs, resulting in improved speed and reduced VRAM usage compared to standard industry baselines.

Inference-Frameworks

- **vLLM:** A high-performance inference engine designed for serving large language models efficiently. It leverages continuous batching and PagedAttention to maximize throughput and reduce latency, making it ideal for production-scale inference.
- **EXL2 (ExLlama):** Optimized for running quantized models, ExLlama and ExLlamaV2 specialize in efficient inference for GGUF and GPTQ models, providing significant speed-ups on consumer GPUs.
- **TGI (Text Generation Inference):** An optimized inference server from Hugging Face for running LLMs efficiently in production, featuring features like dynamic batching, tensor parallelism, and continuous token streaming.
- **Llama.cpp:** A lightweight, C++-based inference framework optimized for CPU and low-resource environments. It supports GGUF models and is commonly used for running models on edge devices and local machines.