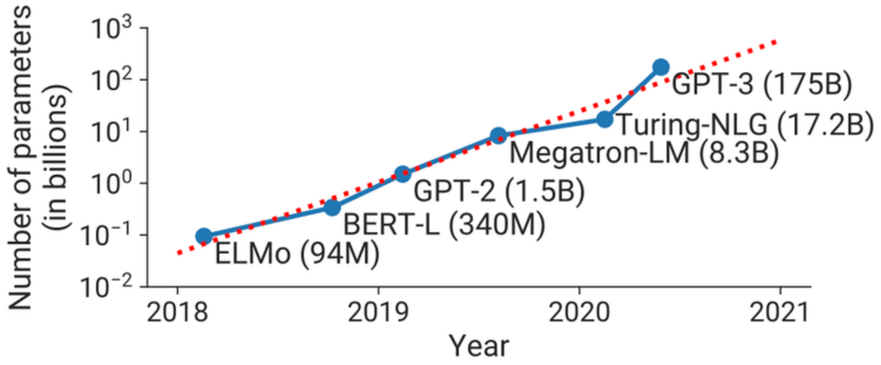
Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM

Research Question:

"How should parallelism techniques be combined to maximize the training throughput of large models given a batch size while retaining strict optimizer semantics?"

Relevance:

- NLP model sizes growing exponentially (from ELMo 94M to GPT-3 175B in 3 years)
- Individual parallelism techniques have scaling limitations
- Naive combinations of parallelism techniques lead to suboptimal performance
- Need practical training times (months, not years)



Technical Challenges

- Large models don't fit in GPU memory. Nvidia 80GB A100s chips are insufficient. GPT-3 (175B parameters) requires 350GB just for weights.

- Training GPT-3 on a single V100 chip would take 288 years.

- Naive implementations of parallelism works for smaller models (20B parameters on NVIDIA DGX A100 servers), but breaks down for larger models. Larger models need to be split across multiple multi-GPU servers.

Types of parallelism

Data Parallelism

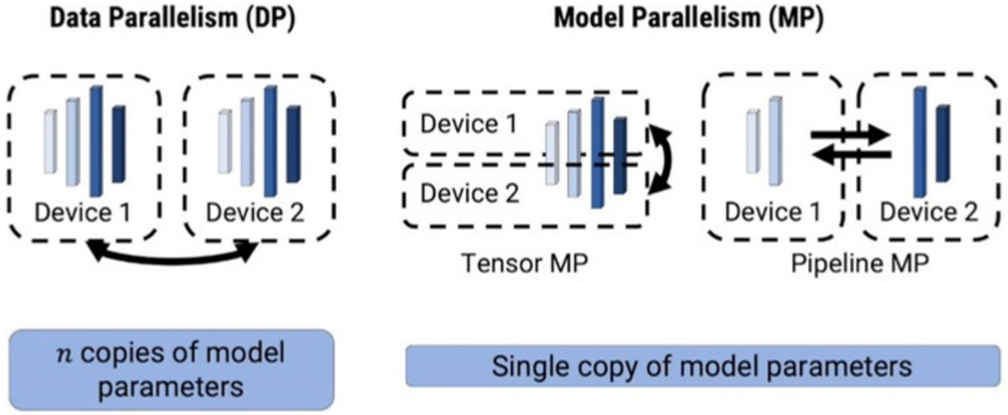
- Each worker has full model copy
- Dataset is sharded across workers
- Limitation: Can't train models larger than single GPU memory

Tensor (Intra-layer) Model Parallelism

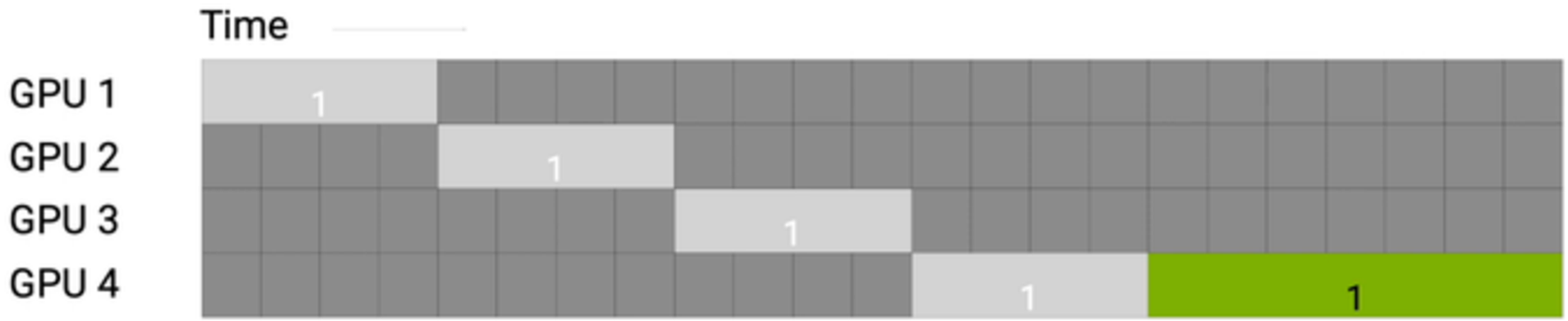
- Individual layers partitioned across GPUs
- Matrix multiplications split
- Best within a node due to high communication requirements

Pipeline (Inter-layer) Model Parallelism

- Sets of layers distributed across GPUs in stages
- Batch split into microbatches, execution pipelined
- Cheaper point-to-point communication
- Challenge: Pipeline bubble (idle time during flush)



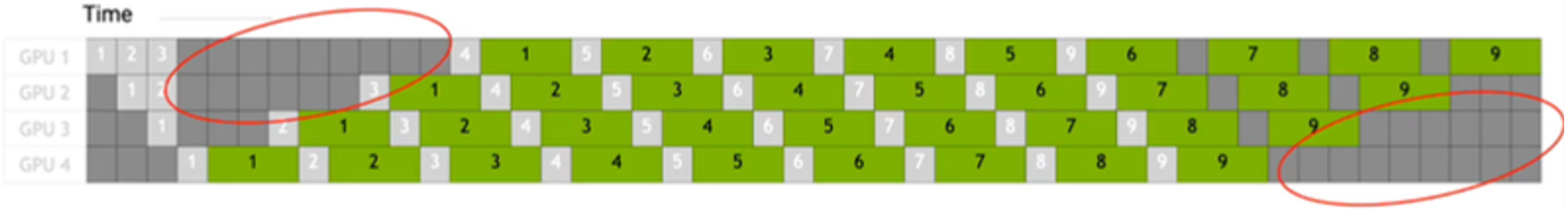
Tensor and Pipelining



Split batch into microbatches and pipeline execution

NVIDIA

Pipeline Bubbles



p : number of pipeline stages

m : number of micro batches

t_f : forward step time

t_b : backward step time

$$\text{total time} = (m + p - 1) \times (t_f + t_b)$$

$$\text{ideal time} = m \times (t_f + t_b)$$

$$\text{bubble time} = (p - 1) \times (t_f + t_b)$$

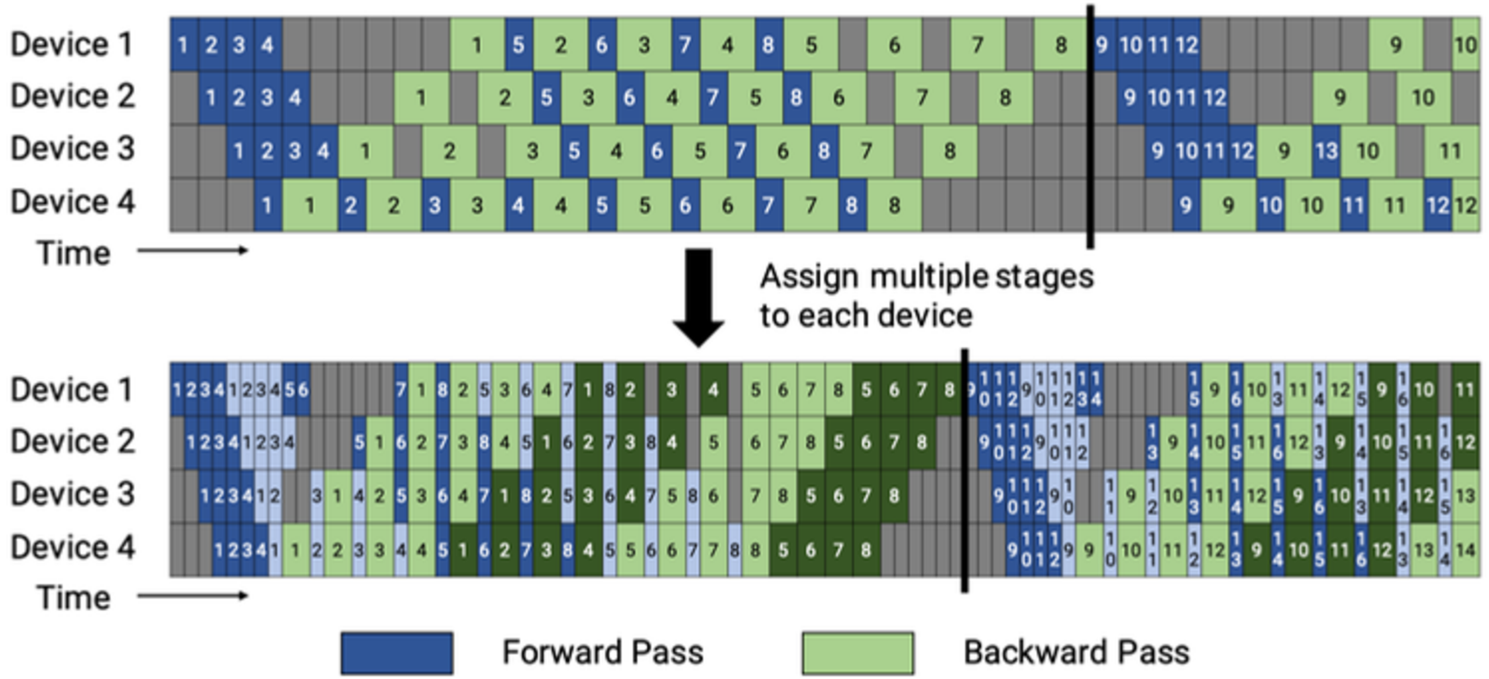
$$\text{bubble time overhead} = \frac{\text{bubble time}}{\text{ideal time}} = \frac{p-1}{m}$$

NVIDIA

- With more pipeline batches, bubble size decreases.
- As number of micro batches increases, there is higher communication overhead relative to computation, as the system must initiate more frequent data transfers between stages.
- Each parallelism mode makes tradeoffs, and determining the optimal degrees of parallelism requires reasoning through these tradeoffs

Pipeline Bubbles

Another way to reduce pipeline bubbles is by adjusting the pipeline schedules of the forward and backward pass



Factors (Heuristics) that influence training efficiency, including pipeline bubbles and memory footprint

- **Tensor parallel size (t)** = GPUs per node (8 for DGX A100)
- **Pipeline parallel size (p)** = Minimum needed to fit model in memory
- **Data parallel size (d)** = Use remaining GPUs (n / (t × p))
- **Microbatch size (b)** = Model-dependent, use analytical model
- **Schedule** - Interleaved (with scatter/gather) for smaller batches
- **Activation recomputation** - Enable for large models / large batches

Setup


312 TFLOP/s
Peak per GPU (FP16)
200 Gbps
HDR InfiniBand
850
Switches
3-Level
Fat-Tree Topology
Core Layer
Core
Core
Core
Core
Spine Layer
Spine
Spine
Spine
Spine
Spine
Spine
Spine
Spine
Leaf Layer
Leaf
Leaf
Leaf
Leaf
Leaf
Leaf
Leaf
Leaf
Leaf
Leaf
Leaf
Leaf
Compute Nodes (8× A100 GPUs each)
Node 1
NVLink + NVSwitch
A100 80GB
HCA
8× App HCAs (200Gbps)
2× Storage
Node 2
NVLink + NVSwitch
Claude content
Node 3
NVLink + NVSwitch
Node N
NVLink + NVSwitch

Combining Parallelism techniques

1. How well does PTD-P (Pipeline, tensor and data parallelism) perform? Does it result in realistic end-to-end training times?
2. How well does pipeline parallelism scale for a given model and batch size?
3. How much impact does the interleaved schedule have on performance?
4. How do different parallelization dimensions interact with each other?
5. What is the impact of hyperparameters such as microbatch size?
6. What is the impact of the scatter-gather communication optimization? What types of limits do we put on hardware when running training iterations at scale?

Evaluation

We consider the end-to-end performance of our system on GPT models ranging from a billion to a trillion parameters, using tensor, pipeline, and data parallelism. In particular, we use the interleaved pipeline schedule with the scatter/gather optimization enabled. All models use a vocabulary size (denoted by V) of 51,200 (multiple of 1024) and a sequence length (s) of 2048. We vary hidden size (h), number of attention heads, and number of layers (l). The number of parameters in a model, P, can be computed as:

$$P = 12lh^2 \left(1 + \frac{13}{12h} + \frac{V+s}{12lh}\right).$$

PTD-P vs ZeRO-3

- ZeRO-3 (from Microsoft DeepSpeed) was the state-of-the-art in 2021.
- Main competing approach for training large models

Results:

- PTD-P trains GPT-3 (175B) in 43-90 days depending on configuration. ZeRO-3 takes 74-169 days for the same model
- PTD-P is 2 times faster than the ZeRO-3 (the previous best approach)

Scheme	Number of parameters (billion)	Model-parallel size	Batch size	Number of GPUs	Microbatch size	Achieved teraFlOP/s per GPU	Training time for 300B tokens (days)
ZeRO-3 without Model Parallelism	174.6	1	1536	384	4	144	90
				768	2	88	74
				1536	1	44	74
	529.6	1	2560*	640	4	138	169
			2240	1120	2	98	137
				2240	1	48	140
PTD Parallelism	174.6	96	1536	384	1	153	84
				768	1	149	43
				1536	1	141	23
	529.6	280	2240	560	1	171	156
				1120	1	167	80
				2240	1	159	42

Evaluate the weak-scaling performance of pipeline parallelism in isolation

Results:

- At batch size 8: Throughput drops from ~175 to ~50 TFLOP/s as p increases from 1 to 8
- At batch size 128: Throughput stays stable (~150-175 TFLOP/s)
- Poor scaling: With small batches, bubbles dominate
- Good scaling: With large batches, bubbles are amortized

Conclusion:

- Pipeline parallelism has a fundamental tradeoff where large batches are needed to hide bubbles
- Pipeline parallelism alone doesn't scale efficiently. Data parallelism shoudl be used first.

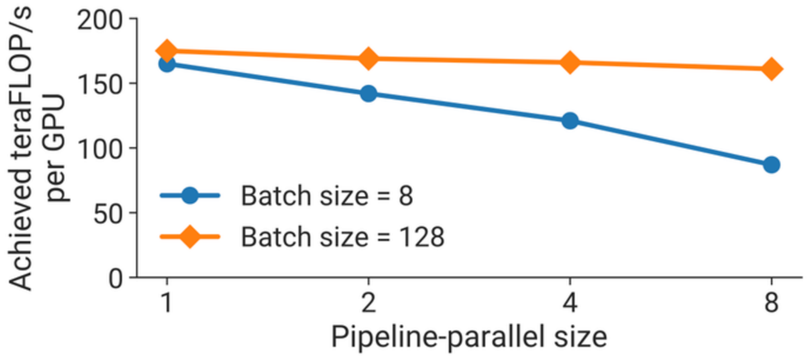


Figure 11: Throughput per GPU of pipeline parallelism using two different batch sizes in a weak-scaling experiment setup (model size increases with the pipeline-parallel size).

Pipeline versus Data Parallelism - Comparison of Parallel Configurations

Results:

- For every batch size tested, throughput decreases as pipeline-parallel size increases
- Data parallelism consistently outperforms pipeline parallelism when model fits in memory
- At batch size 512: (p=2, d=32) achieves ~175 TFLOP/s vs (p=32, d=2) achieves only ~25 TFLOP/s

Conclusion:
Pipeline parallelism should be used only when necessary (model doesn't fit on fewer GPUs). Data parallelism is preferred for scaling when possible.

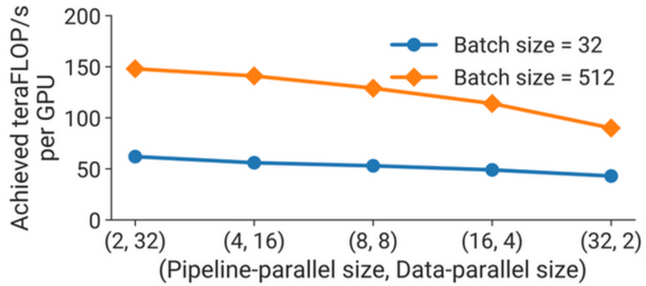


Figure 14: Throughput per GPU of various parallel configurations that combine data and pipeline model parallelism using a GPT model with 5.9 billion parameters, three different batch sizes, microbatch size of 1, and 64 A100 GPUs.

Tensor versus Data Parallelism

Results:

- Performance degrades significantly when t > 8
  - Every layer requires all-reduce across all t GPUs
  - When t > 8, spans multiple nodes which results in slower InfiniBand instead of fast NVLink
  - Higher t = smaller matrix multiplications per GPU = lower GPU utilization

Conclusion:
Use tensor parallelism only up to t=8 (one node), then switch to pipeline parallelism

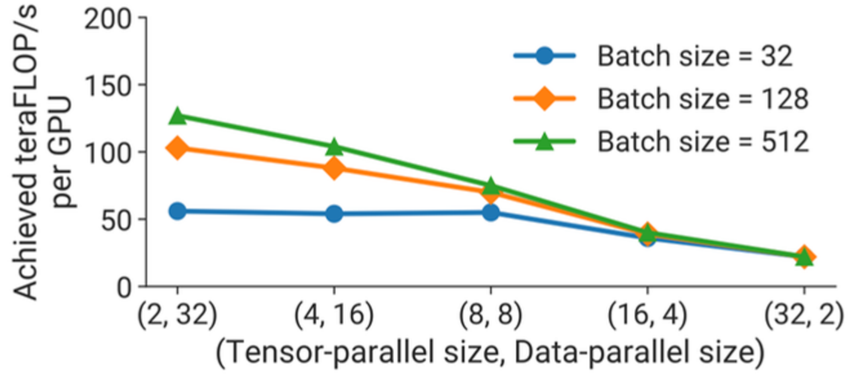


Figure 15: Throughput per GPU of various parallel configurations that combine data and tensor model parallelism using a GPT model with 5.9 billion parameters, three different batch sizes, microbatch size of 1, and 64 A100 GPUs.

Tensor versus Pipeline Parallelism

Key finding: Peak performance at (t=8, p=8) — tensor parallelism within a node, pipeline parallelism across nodes.
Why?

- t < 8: Not utilizing all intra-node NVLink bandwidth
- t > 8: Tensor parallelism crosses node boundaries → expensive inter-node all-reduce
- p too high: Large pipeline bubble
- p too low: Model doesn't fit / underutilizes available GPUs

Takeaway: Use tensor parallelism = number of GPUs per node (8 for DGX A100), then add pipeline parallelism for larger models.

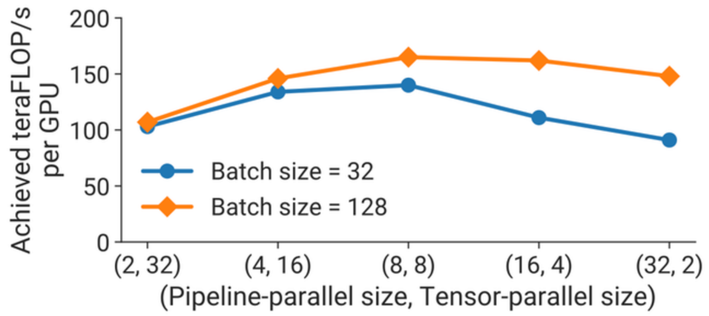


Figure 13: Throughput per GPU of various parallel configurations that combine pipeline and tensor model parallelism using a GPT model with 162.2 billion parameters and 64 A100 GPUs.

Impact of microbatch size

Results:

- Too small (b=1): Large pipeline bubbles (bubble overhead = (p-1)/m where m = B/(b×d))
- Too large: Fewer microbatches → can't hide communication latency
- Just right: Balances computation/communication overlap with bubble size

Conclusion:
Use analytical model (below) to predict optimal microbatch size

3.3.1 Pipeline Model Parallelism. Let $t = 1$ (tensor-model-parallel size). The number of microbatches per pipeline is $m = B/(d \cdot b) = b'/d$, where $b' := B/b$. With total number of GPUs n, the number of pipeline stages is $p = n/(t \cdot d) = n/d$. The pipeline bubble size is:

$$\frac{p-1}{m} = \frac{n/d-1}{b'/d} = \frac{n-d}{b'}.$$

Activation Recomputation

- Understand the tradeoff between memory and compute, and when it helps.

Results:

- Smaller bubbles from more microbatches, which increases the cost of recomputation
- Bubble overhead = (p-1)/m. Doubling batch means doubles m, which halves the bubble

Conclusion:
Enable activation recomputation for large models to support larger batch sizes

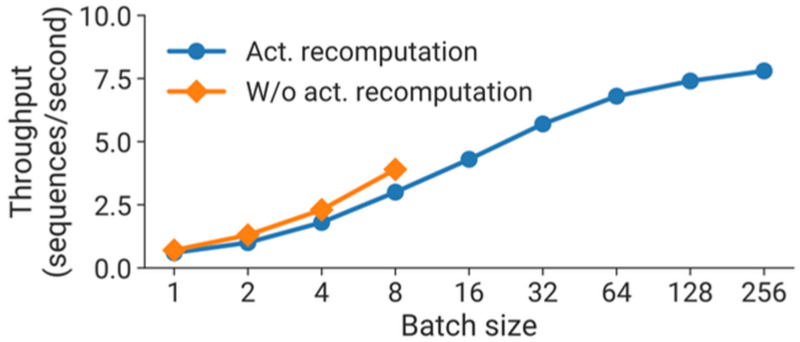


Figure 17: Throughput (in sequences per second) with and without activation recomputation for a GPT model with 145 billion parameters using 128 A100 GPUs ($(t, p) = (8, 16)$).

LLaMA: Open and Efficient Foundation Language Models

Research Objective:
The focus of this work is to train a series of language models that achieve the best possible performance at various inference budgets, by training on more tokens than what is typically used.

Relevance:
can we achieve what openai acheived with open source data

Approach:

Train large transformers on a large quantity of textual data using a standard optimizer. The models were trained only using data that is publicly available, and compatible with open sourcing.

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

Table 1: **Pre-training data.** Data mixtures used for pre-training, for each subset we list the sampling proportion, number of epochs performed on the subset when training on 1.4T tokens, and disk size. The pre-training runs on 1T tokens have the same sampling proportion.

Training and Architecture

- Using the RMSNorm normalising function, the input of each transformer sub-layer is normalised, instead of the output.
- ReLU non-linearity activation function was replaced by the SwiGLU activation function
- AdamW optimizer was used with hyper-parameters: β1 = 0.9,β2 = 0.95.

params	dimension	n heads	n layers	learning rate	batch size	n tokens
6.7B	4096	32	32	$3.0e^{-4}$	4M	1.0T
13.0B	5120	40	40	$3.0e^{-4}$	4M	1.0T
32.5B	6656	52	60	$1.5e^{-4}$	4M	1.4T
65.2B	8192	64	80	$1.5e^{-4}$	4M	1.4T

Table 2: **Model sizes, architectures, and optimization hyper-parameters.**

Results

		BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA
GPT-3	175B	60.5	81.0	-	78.9	70.2	68.8	51.4	57.6
Gopher	280B	79.3	81.8	50.6	79.2	70.1	-	-	-
Chinchilla	70B	83.7	81.8	51.3	80.8	74.9	-	-	-
PaLM	62B	84.8	80.5	-	79.7	77.0	75.2	52.5	50.4
PaLM-cont	62B	83.9	81.4	-	80.6	77.0	-	-	-
PaLM	540B	**88.0**	82.3	-	83.4	**81.1**	76.6	53.0	53.4
LLaMA	7B	76.5	79.8	48.9	76.1	70.1	72.8	47.6	57.2
	13B	78.1	80.1	50.4	79.2	73.0	74.8	52.7	56.4
	33B	83.1	82.3	50.4	82.8	76.0	**80.0**	**57.8**	58.6
	65B	85.3	**82.8**	**52.3**	**84.2**	77.0	78.9	56.0	**60.2**

Table 3: **Zero-shot performance on Common Sense Reasoning tasks.**

		0-shot	1-shot	5-shot	64-shot
GPT-3	175B	14.6	23.0	-	29.9
Gopher	280B	10.1	-	24.5	28.2
Chinchilla	70B	16.6	-	31.5	35.5
PaLM	8B	8.4	10.6	-	14.6
	62B	18.1	26.5	-	27.6
	540B	21.2	29.3	-	39.6
LLaMA	7B	16.8	18.7	22.0	26.1
	13B	20.1	23.4	28.1	31.9
	33B	**24.9**	28.3	32.9	36.0
	65B	23.8	**31.0**	**35.0**	**39.9**

Table 4: **NaturalQuestions.** Exact match performance.

Results

		0-shot	1-shot	5-shot	64-shot
Gopher	280B	43.5	-	57.0	57.2
Chinchilla	70B	55.4	-	64.1	64.6
LLaMA	7B	50.0	53.4	56.3	57.6
	13B	56.6	60.5	63.1	64.0
	33B	65.1	67.9	69.9	70.4
	65B	**68.2**	**71.6**	**72.6**	**73.0**

Table 5: **TriviaQA.** Zero-shot and few-shot exact match performance on the filtered dev set.

		RACE-middle	RACE-high
GPT-3	175B	58.4	45.5
PaLM	8B	57.9	42.3
	62B	64.3	47.5
	540B	**68.1**	49.1
LLaMA	7B	61.1	46.9
	13B	61.6	47.2
	33B	64.1	48.3
	65B	67.9	**51.6**

Table 6: **Reading Comprehension.** Zero-shot accuracy.

Results

		MATH	+maj1@k	GSM8k	+maj1@k
PaLM	8B	1.5	-	4.1	-
	62B	4.4	-	33.0	-
	540B	8.8	-	56.5	-
Minerva	8B	14.1	25.4	16.2	28.4
	62B	27.6	43.4	52.4	68.5
	540B	**33.6**	**50.3**	**68.5**	**78.5**
LLaMA	7B	2.9	6.9	11.0	18.1
	13B	3.9	8.8	17.8	29.3
	33B	7.1	15.2	35.6	53.1
	65B	10.6	20.5	50.9	69.7

Table 7: **Model performance on quantitative reasoning datasets.** For majority voting, we use the same setup as Minerva, with $k = 256$ samples for MATH and $k = 100$ for GSM8k (Minerva 540B uses $k = 64$ for MATH and and $k = 40$ for GSM8k). LLaMA-65B outperforms Minerva 62B on GSM8k, although it has not been fine-tuned on mathematical data.

	Params	HumanEval		MBPP	
pass@		@1	@100	@1	@80
LaMDA	137B	14.0	47.3	14.8	62.4
PaLM	8B	3.6*	18.7*	5.0*	35.7*
PaLM	62B	15.9	46.3*	21.4	63.2*
PaLM-cont	62B	23.7	-	31.2	-
PaLM	540B	**26.2**	76.2	36.8	75.0
LLaMA	7B	10.5	36.5	17.7	56.2
	13B	15.8	52.5	22.0	64.0
	33B	21.7	70.7	30.2	73.4
	65B	23.7	**79.3**	**37.7**	**76.8**

Table 8: **Model performance for code generation.** We report the pass@ score on HumanEval and MBPP. HumanEval generations are done in zero-shot and MBBP with 3-shot prompts similar to Austin et al. (2021). The values marked with * are read from figures in Chowdhery et al. (2022).

Results

		Humanities	STEM	Social Sciences	Other	Average
GPT-NeoX	20B	29.8	34.9	33.7	37.7	33.6
GPT-3	175B	40.8	36.7	50.4	48.8	43.9
Gopher	280B	56.2	47.4	71.9	66.1	60.0
Chinchilla	70B	63.6	54.9	79.3	**73.9**	67.5
PaLM	8B	25.6	23.8	24.1	27.8	25.4
	62B	59.5	41.9	62.7	55.8	53.7
	540B	**77.0**	**55.6**	**81.0**	69.6	**69.3**
LLaMA	7B	34.0	30.5	38.3	38.1	35.1
	13B	45.0	35.8	53.8	53.3	46.9
	33B	55.8	46.0	66.7	63.4	57.8
	65B	61.8	51.7	72.9	67.4	63.4

Table 9: **Massive Multitask Language Understanding (MMLU).** Five-shot accuracy.

OPT	30B	26.1
GLM	120B	44.8
PaLM	62B	55.1
PaLM-cont	62B	62.8
Chinchilla	70B	67.5
LLaMA	65B	63.4
OPT-IML-Max	30B	43.2
Flan-T5-XXL	11B	55.1
Flan-PaLM	62B	59.6
Flan-PaLM-cont	62B	66.1
LLaMA-I	65B	**68.9**

Table 10: **Instruction finetuning – MMLU (5-shot).** Comparison of models of moderate size with and without instruction finetuning on MMLU.

Results

		Truthful	Truthful*Inf
GPT-3	1.3B	0.31	0.19
	6B	0.22	0.19
	175B	0.28	0.25
LLaMA	7B	0.33	0.29
	13B	0.47	0.41
	33B	0.52	0.48
	65B	0.57	0.53

Table 14: **TruthfulQA.** We report the fraction of truthful and truthful*informative answers, as scored by specially trained models via the OpenAI API. We follow the QA prompt style used in Ouyang et al. (2022), and report the performance of GPT-3 from the same paper.