

October 10 – 12, 2023



ALCF Hands-on HPC Workshop

Status of Large Language Models

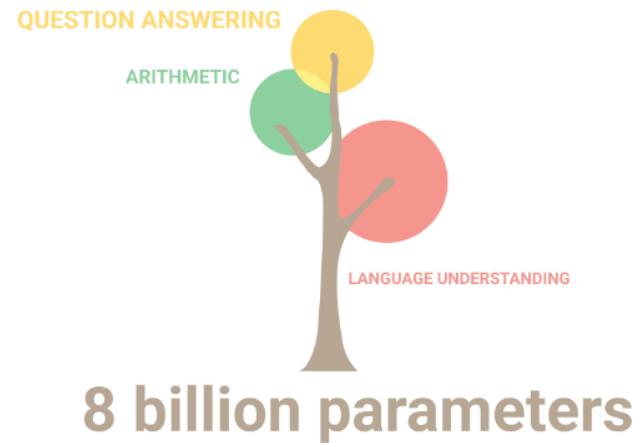
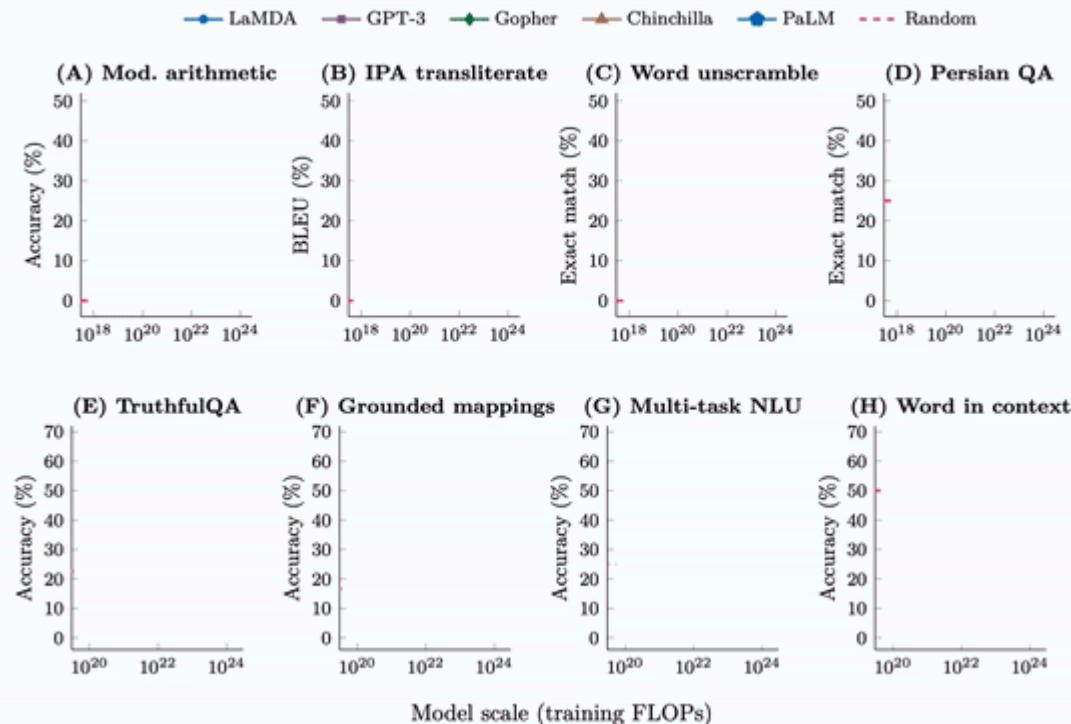


Figure 1: Large Language Models have (LLM)s have taken the ~~NLP community~~ world by storm¹

1. [Hannibal046/Awesome-LLM](#)

Emergent Abilities



[Emergent abilities of Large Language Models](#) Yao et al. (2023)

Training LLMs

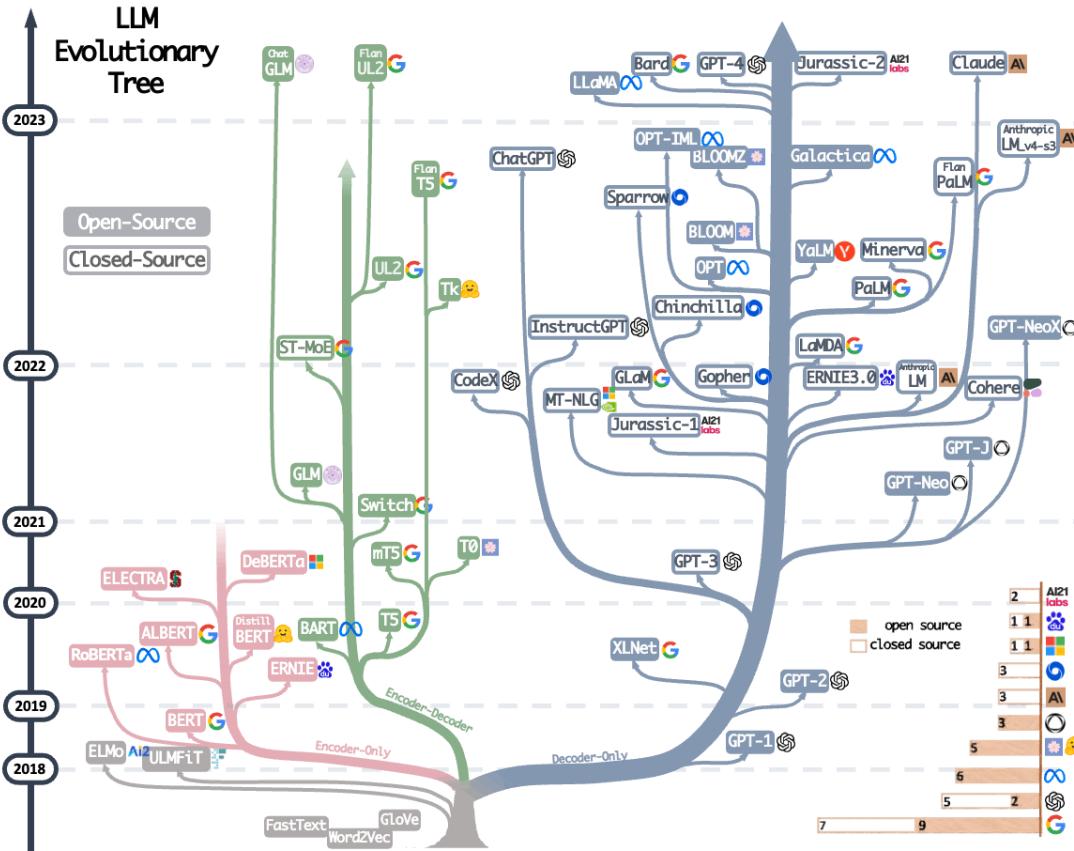


Figure 2: Visualization from Yang et al. (2023)

May God forgive us for what we have done



Training LLMs

It hungers

O'RLY?

Lovecraft

Recent Work (2017 – Now)

► Recent Work

Life-Cycle of the LLM

1. Data collection + preprocessing

2. Pre-training

- Architecture decisions:
`{model_size,
hyperparameters,
parallelism,
lr_schedule, ...}`

3. Supervised Fine-Tuning

- Instruction Tuning
- Alignment

4. Deploy (+ monitor, re-evaluate, etc.)

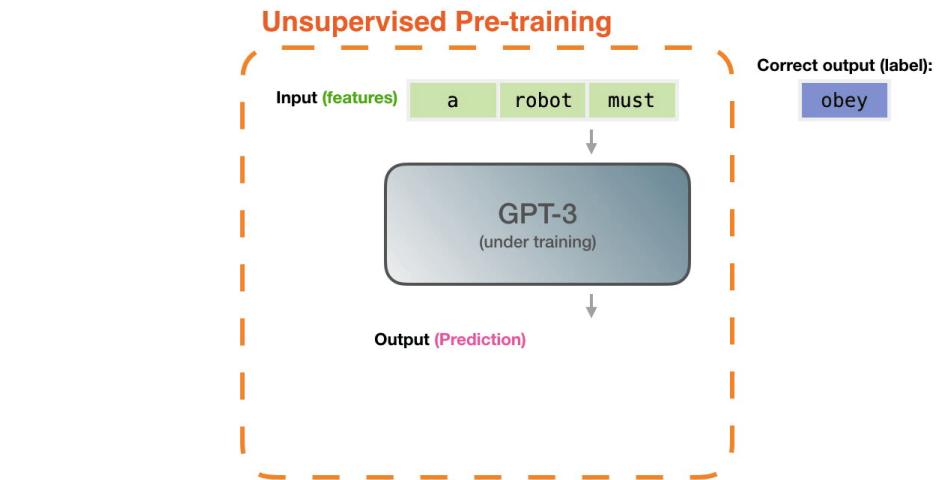


Figure 3: **Pre-training:** Virtually all of the compute used during pretraining phase¹.

1. Figure from [The Illustrated Transformer](#)

Life-Cycle of the LLM: Pre-training

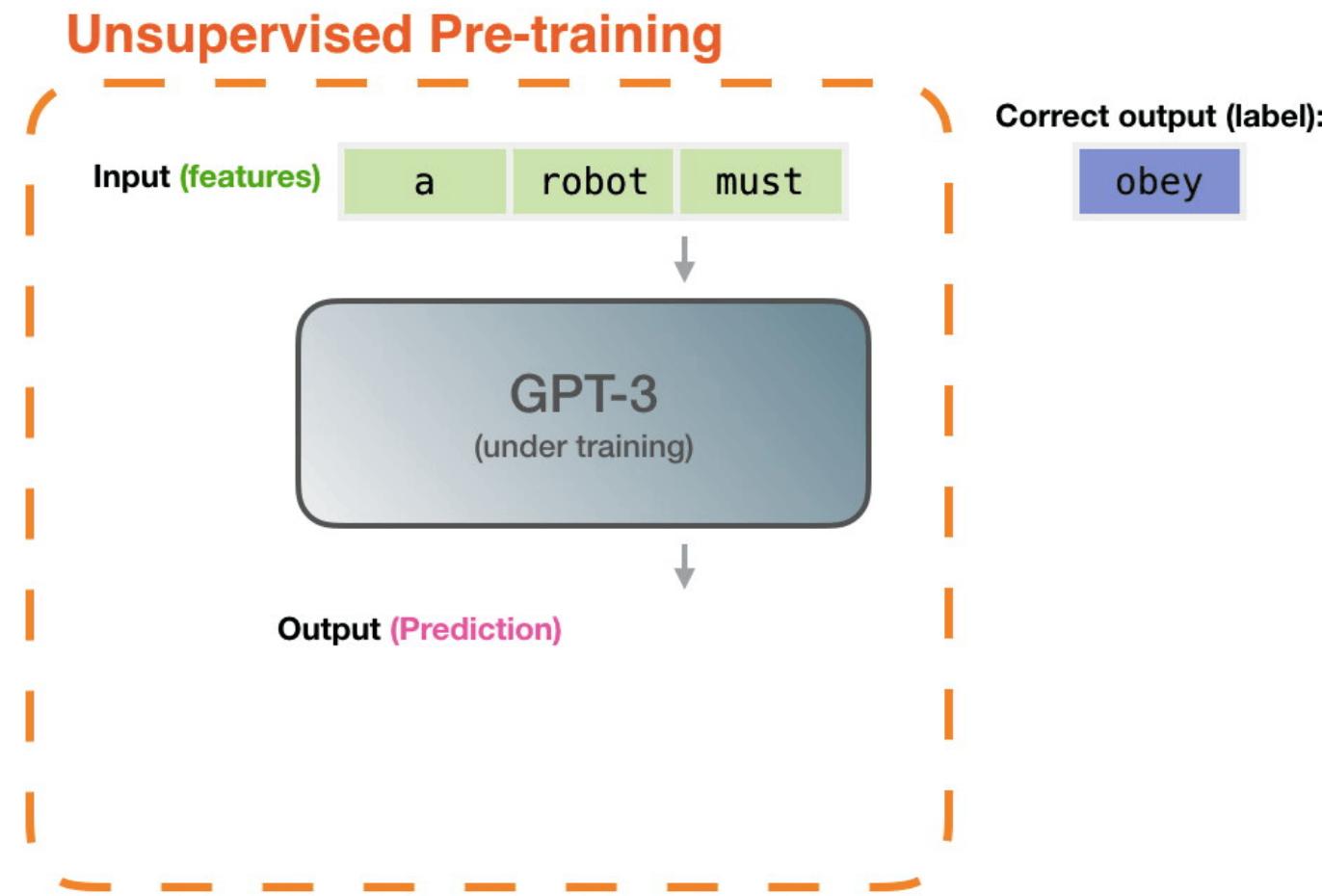
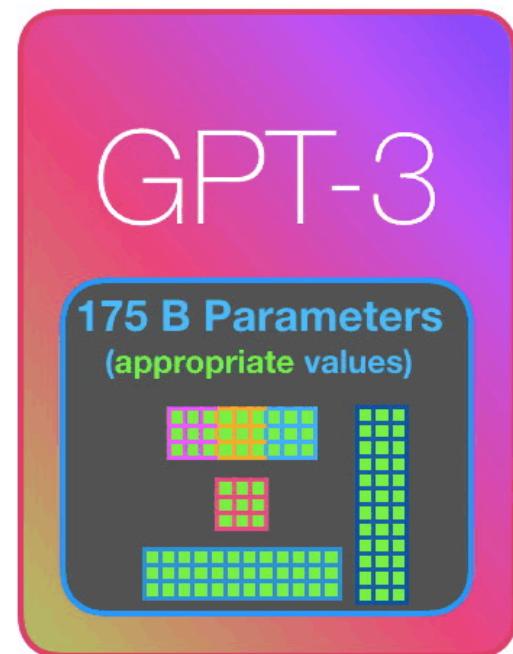


Figure 4: **Pre-training:** Virtually all of the compute used during pretraining phase

Life-Cycle of the LLM: Fine-Tuning

Pre-training



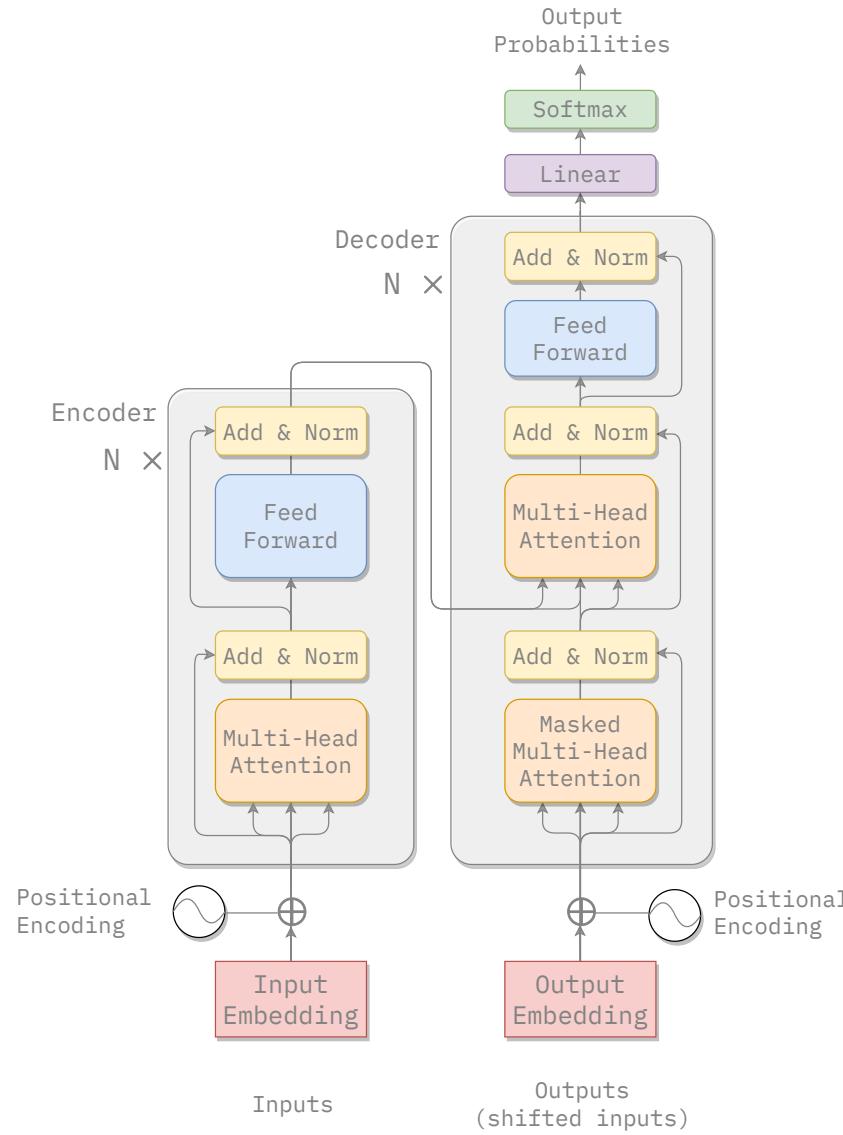
Fine-tuning

Additional training to become better at a certain task

Example: English to French Translation

Figure 5: **Fine-tuning:** Fine-tuning actually updates the model's weights to make the model better at a certain task.

Transformer Architecture



Vaswani et al. (2017)

Forward Pass

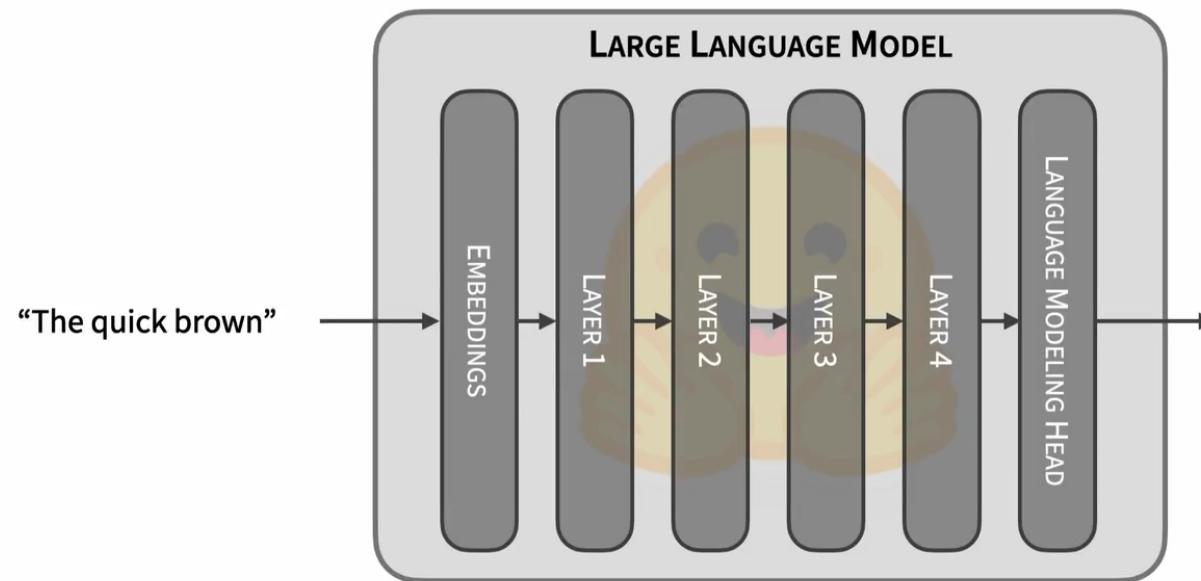


Figure 6: Language Model trained for causal language modeling. Video from: 😊 [Generation with LLMs](#)

Generating Text

Figure 7: Language Model trained for causal language modeling. Video from:  [Generation with LLMs](#)

Parallelism Overview

Modern parallelism techniques enable the training of large language models

Parallelism Concepts¹

- **DataParallel (DP):**

- The same setup is replicated multiple times, and each being fed a slice of the data.
- The processing is done in parallel and all setups are synchronized at the end of each training step.

- **TensorParallel (TP):**

- Each tensor is split up into multiple chunks.
- So, instead of having the whole tensor reside on a single gpu, each shard of the tensor resides on its designated gpu.
 - During processing each shard gets processed separately and in parallel on different GPUs and the results are synced at the end of the step.
 - This is what one may call horizontal parallelism, as the splitting happens on horizontal level.

1. 😊 [Model Parallelism](#)

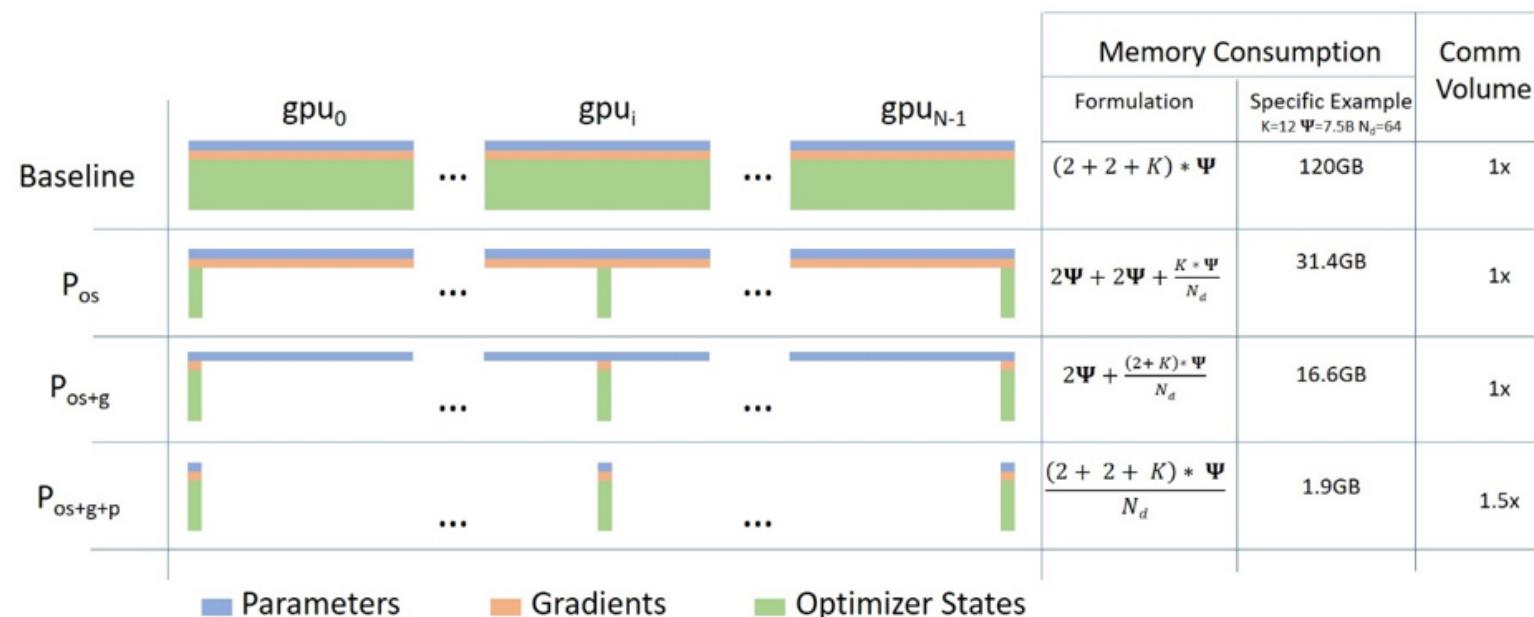
Parallelism Concepts¹

- **Pipeline Parallel (PP):**
 - Model is split up vertically (layer-level) across multiple GPUs, so that only one or several layers of the model are places on a single gpu.
 - Each gpu processes in parallel different stages of the pipeline and working on a small chunk of the batch.
- **Zero Redundancy Optimizer (ZeRO):**
 - Also performs sharding of the tensors somewhat similar to TP, except the whole tensor gets reconstructed in time for a forward or backward computation, therefore the model doesn't need to be modified.
 - It also supports various offloading techniques to compensate for limited GPU memory.
- **Sharded DDP:**
 - Another name for the foundational ZeRO concept as used by various other implementations of ZeRO.

1. 😊 [Model Parallelism](#)

Data Parallelism

- Data Parallelism:
 - The simplest and most common parallelism technique. Workers maintain *identical copies* of the *complete* model and work on a *subset of the data*.
 - DDP supported in PyTorch native.
- ZeRO Data Parallel
 - ZeRO powered data parallelism is shown below¹



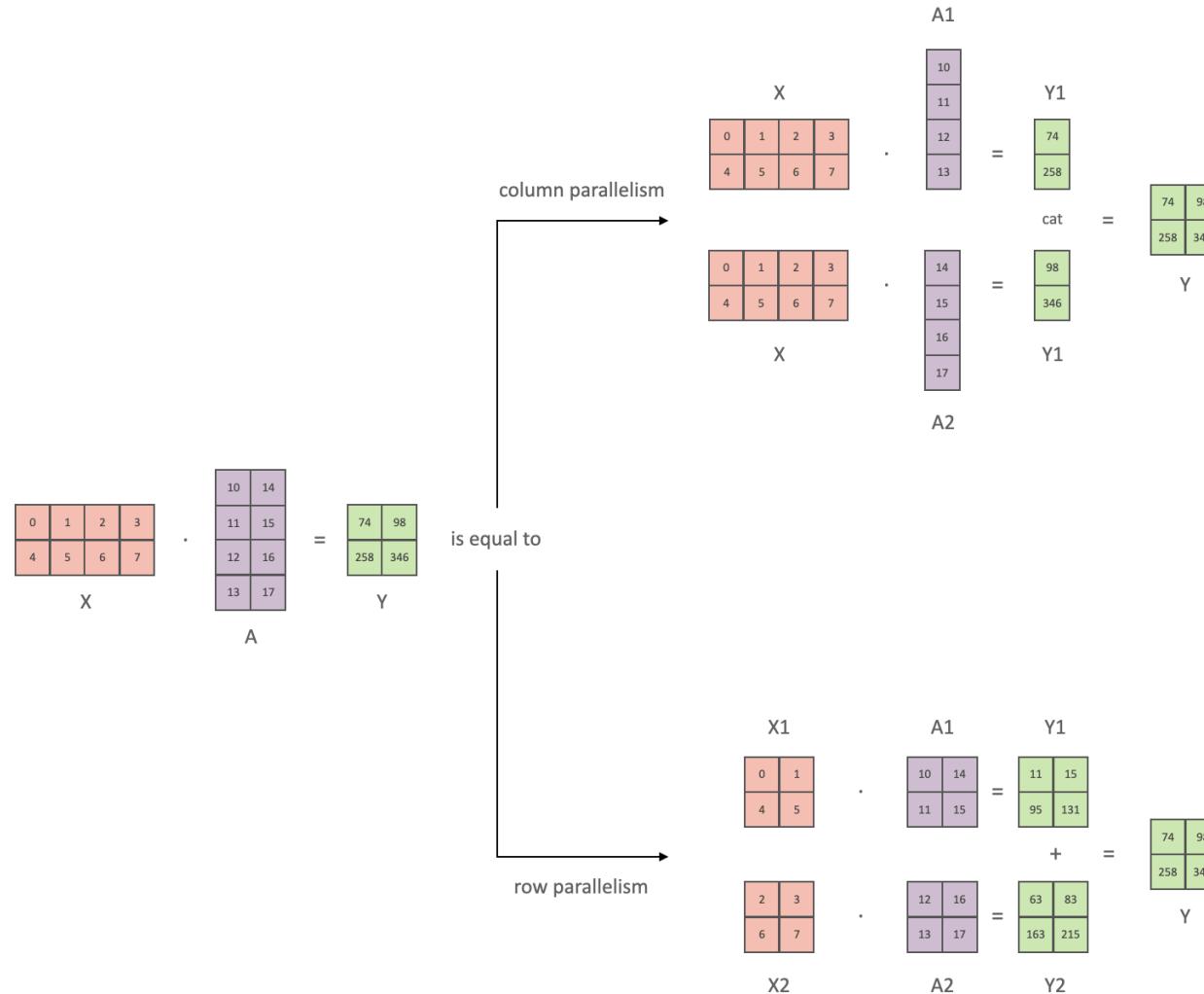
1. [Blog Post](#)

Tensor Parallelism¹

- In **Tensor Parallelism** each GPU processes only a slice of a tensor and only aggregates the full tensor for operations that require the whole thing.
 - The main building block of any transformer is a fully connected nn.Linear followed by a nonlinear activation GeLU.
 - $Y = \text{GeLU}(XA)$, where X and Y are the input and output vectors, and A is the weight matrix.
 - If we look at the computation in matrix form, it's easy to see how the matrix multiplication can be split between multiple GPUs:

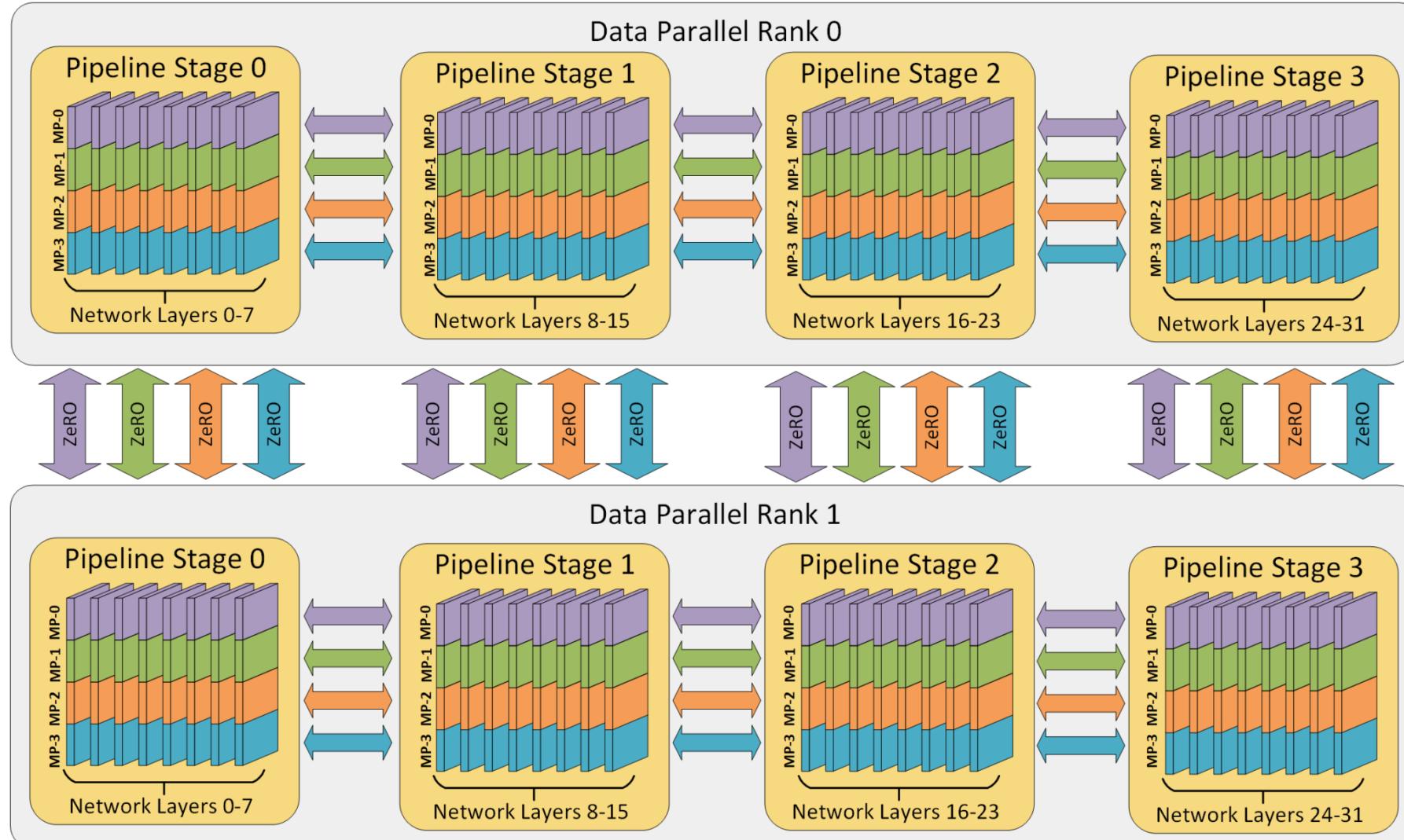
1. [Efficient Large-Scale Language Model Training on GPU Clusters](#)

Tensor Parallelism



3D Parallelism

- DP + TP + PP (3D) Parallelism



3D Parallelism illustration. Figure from: <https://www.deepspeed.ai/>

3D Parallelism

- DP + TP + PP (3D) Parallelism

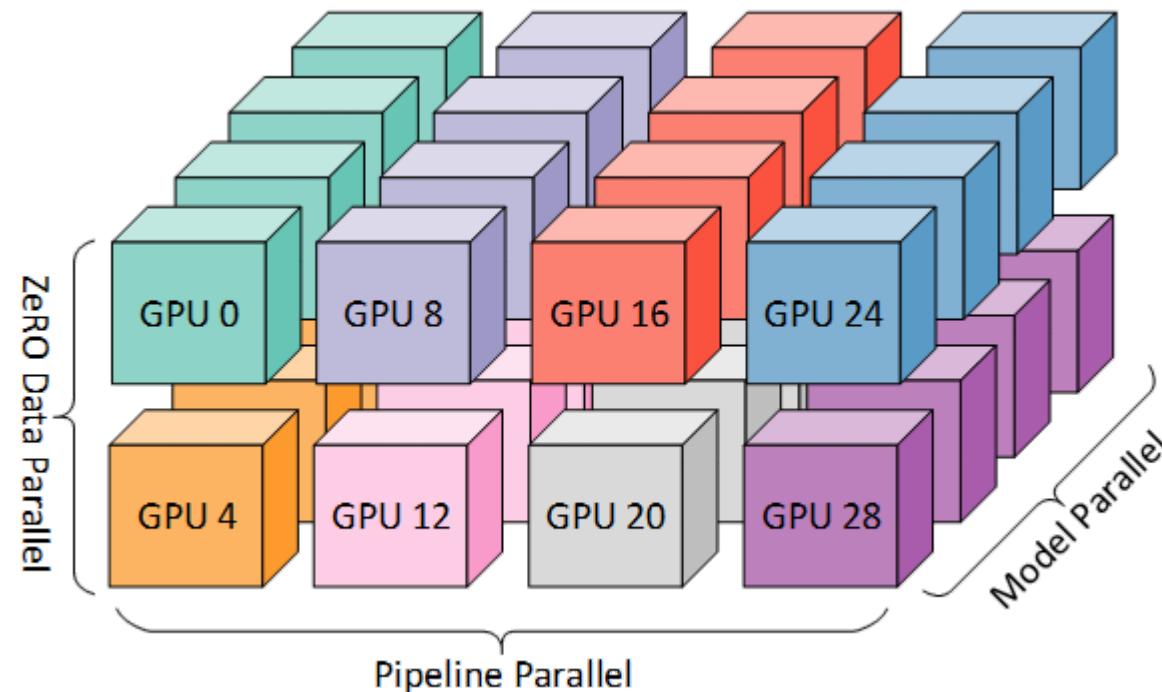


Figure taken from [3D parallelism: Scaling to trillion-parameter models](#)

Running on ALCF

- We've provided a virtual environment complete with all dependencies for running
 [argonne-lcf/Megatron-DeepSpeed](#)

```
# navigate to directory
WORKSHOP_DIR="/lus/grand/projects/fallwkshp23/"
PROJECTS_DIR="${WORKSHOP_DIR}/foremans/locations/polaris/projects/"
PROJECT_DIR="${PROJECTS_DIR}/argonne-lcf/Megatron-DeepSpeed"
cd "${PROJECT_DIR}"

# load conda module and activate venv
module load conda/2023-10-04; conda activate base
source venvs/polaris/2023-10-04/bin/activate

# set runtime environment variables
export IBV_FORK_SAFE=1
export CUDA_DEVICE_MAX_CONNECTIONS=1

# set environment variables for running
MODEL_SIZE_KEY="GPT1_5B"
SEQ_LEN=1024
MICRO_BATCH=1
SP_TYPE="megatron"

# launch training
./ALCF/train-gpt3.sh
```

Running on ALCF

- Executable:

```
MODEL_SIZE_KEY="GPT1_5B" SEQ_LEN=1024 MICRO_BATCH=1 SP_TYPE="megatron" ./ALCF/train-gpt3.sh
```

▼ Output

```
+++++
ALCF_DIR: /lus/grand/projects/fallwkshp23/foremans/locations/polaris/projects/argonne-lcf
+++++
source-ing /lus/grand/projects/fallwkshp23/foremans/locations/polaris/projects/argonne-lc
Setting up MPI on Polaris from x3210c0s1b0n0
++ SetupMPI() ++++++
Using HOSTFILE: /var/spool/pbs/aux/1126584.polaris-pbs-01.hsn.cm.polaris.alcf.anl.gov
NHOSTS: 2
NGPU_PER_HOST: 4
NGPUS: 8
+++++
Skipping setupThetaGPU() on x3210c0s1b0n0
Setting up MPI on Polaris from x3210c0s1b0n0
USING PYTHON: /lus/grand/projects/fallwkshp23/foremans/locations/polaris/projects/argonne
[...]
```

Running on ALCF

Once the text has *finally* stopped printing, you should see output similar to the following:

```
Job started at: 2023-10-11-092906 on x3210c0s1b0n0
[...]
Writing logs to: /lus/grand/projects/fallwkshp23/foremans/locations/polaris/projects/argonne-lcf/Megatron-DeepSpeed
to view output: tail -f $(tail -1 logfiles)
i.e. tail -f /lus/grand/projects/fallwkshp23/foremans/locations/polaris/projects/argonne-lcf/Megatron-DeepSpeed
```

- To watch / view the output:

```
tail -fn 1000 $(tail -1 logfiles) | less
```

- will look like¹:

```
Job started at: 2023-10-11-092906 on x3210c0s1b0n0
Training GPT-3 with GPT13B parameters
Writing logs to: /lus/grand/projects/fallwkshp23/foremans/locations/polaris/projects/argonne-lcf/Megatron-DeepSpeed
to view output: tail -f $(tail -1 logfiles)
i.e. tail -f /lus/grand/projects/fallwkshp23/foremans/locations/polaris/projects/argonne-lcf/Megatron-DeepSpeed
using: /lus/grand/projects/fallwkshp23/foremans/locations/polaris/projects/argonne-lcf/Megatron-DeepSpeed/venvs
[...]
```

1.  W&B Run: [soft-wave-264](#)

Getting Started at ALCF

- We provide below the **details** for installing / getting started on ALCF (Polaris)

- Installation:

1.  Clone GitHub repo:

```
git clone https://github.com/argonne-lcf/Megatron-DeepSpeed
```

2. Load Conda module:

- Polaris:

```
if [[ "$hostname"==x3* ]]; then
    export MACHINE="Polaris"
    export CONDA_DATE="2023-10-04"
    module load conda/${CONDA_DATE}
    conda activate base
fi
```

- ThetaGPU:

```
if [[ "$hostname"==theta* ]]; then
    export MACHINE="ThetaGPU"
    export CONDA_DATE="2023-01-10"
    module load conda/${CONDA_DATE}
    conda activate base
fi
```

Getting Started

3. Setup virtual environment¹:

```
cd Megatron-DeepSpeed
# create a new virtual environment
mkdir -p "venvs/${MACHINE}/${CONDA_DATE}"
python3 -m venv "venvs/${MACHINE}/${CONDA_DATE}" --system-site-packages
source "venvs/${MACHINE}/${CONDA_DATE}/bin/activate"
```

4. Create a new folder where we'll install dependencies:

```
mkdir -p "deps/${MACHINE}"
cd "deps/${MACHINE}"
```

1. On-top of the base `conda` environment (`--system-site-packages`)

Install Dependencies

 Dao-AILab/flash-attention

 saforem2/ezpz

 NVIDIA/apex

- The [new release](#) supports three different implementations of FlashAttention: (`v1.0.4`, `v2.x`, `triton`)
- FlashAttention `v2.x` may have numerical instability issues. For the best performance, we recommend using FlashAttention + Triton
- [!\[\]\(21836c01843165e8dc035337f541a17f_img.jpg\) Dao-AILab/flash-attention](#):

- `v1.0.4`:

```
python3 -m pip install flash-attn==1.0.4
```

- `v2.x`:

```
git clone https://github.com/Dao-AILab/flash-attention
cd flash-attention
python3 setup.py install
```

- `openai/triton`:

```
git clone -b legacy-backend https://github.com/openai/triton
cd triton/python
python3 -m pip install cmake pybind11
python3 -m pip install .
```

Running

- The  [ALCF/](#) directory contains shell scripts for setting up the environment and specifying options to be used for training.
-  [ALCF/](#)
 - └ [args.sh](#)
 - └ [launch.sh](#)
 - └ [model.sh](#)
 - └ [setup.sh](#)
 - └ [submit-pbs.sh](#)
 - └ [submit.sh](#)
 - └ [train-gpt3.sh](#)

- Various options can be specified dynamically at runtime by setting them in your environment, e.g.:

```
# Set env. vars to use:  
MODEL_SIZE_KEY="GPT25B"  
SEQ_LEN=1024  
USE_FLASH_ATTN=1  
MICRO_BATCH=1  
GAS=1  
SP_TYPE="megatron"  
ZERO_STAGE=1  
# Launch training:  
. ./ALCF/train-gpt3.sh
```

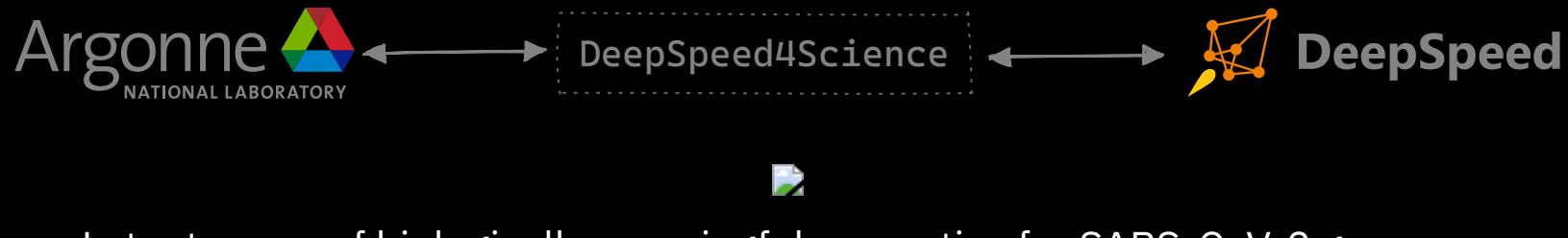
Details

Explicitly:

- `ALCF/train-gpt3.sh`: **Main entry point for training.** This script will:
 - Source the rest of the required `ALCF/*.sh` scripts below
- `ALCF/models.sh`: Contains some example model architectures for GPT3-style models
- `ALCF/args.sh`: Logic for parsing / setting up runtime options for Megatron and DeepSpeed
- `ALCF/setup.sh`: Locate and activate virtual environment to be used, ensure MPI variables are set properly
- `ALCF/launch.sh`: Identify available resources and build the command to be executed
 - i.e. figure out how many: `{nodes, GPUs per node, GPUs total}`, to pass to `mpi{run,exec}`
 - then, use this to launch `mpiexec <mpiexec-args> python3 pretrain_gpt.py <gpt-args>`

DeepSpeed4Science

- Long Sequence Support for GenSLM Model



Latent space of biologically meaningful properties for SARS-CoV-2 genomes

Looooooooong Sequence Lengths

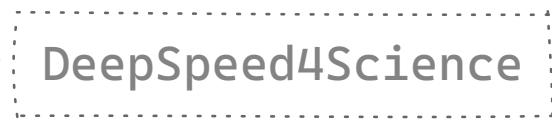


Table 2: Long sequence length support from microsoft/Megatron-DeepSpeed

Sequence Length	Old Megatron-DeepSpeed (TFLOPS)	New Megatron-DeepSpeed (TFLOPS)
2k	25	68
4k	28	80
8k	OOM	86
16k	OOM	92
32k	OOM	100
64k	OOM	106
128k	OOM	119
256k	OOM	94

Looooooooooooong Sequence Lengths

- Working with [Microsoft DeepSpeed](#) team to enable longer sequence lengths (context windows) for LLMs¹
 - [Release: DeepSpeed4Science Overview and Tutorial](#)

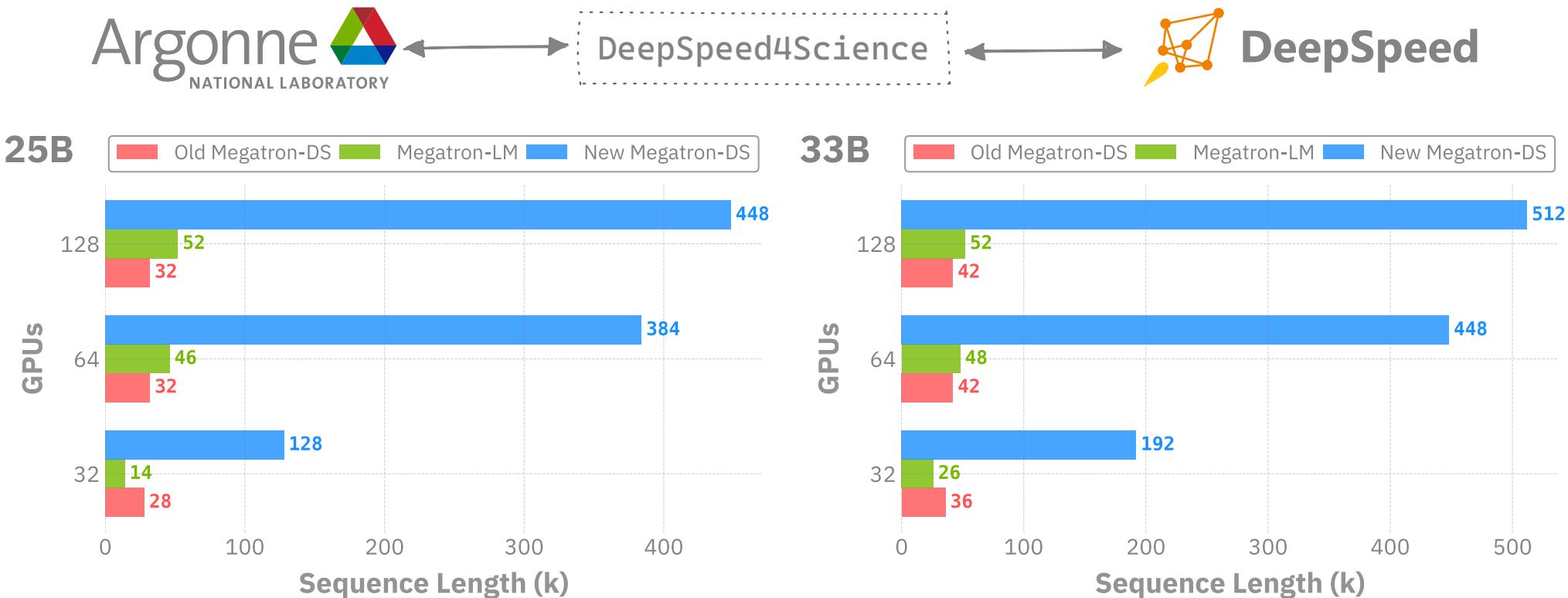


Figure 8: Maximum (achievable) SEQ_LEN for both 25B and 33B models [WIP]

1. The described experiments were performed on 4 NVIDIA DGX A100-40GB nodes, all using TPSIZE=32[^tpsize], connected through 8 HDR InfiniBand (200Gb/s per HDR). ↩

Looooooooong Sequence Lengths

- We can evaluate the performance of our model by looking at two different metrics for throughput: `samples_per_sec` and `TFLOPS`.
 - Explicitly, we see that we are able to scale up to significantly longer sequences: (`420k / 128k ~ 3.3x`) with only a minimal impact on throughput performance: (`81 / 105 ~ 77%`)¹.

Table 3: Impact on TFLOPS as a function of increasing sequence length. Table from:
[throughput/TFLOPS](#)

Name	Sequence Length (k)	(<code>seq_len / min_seq_len</code>)	TFLOPS	TFLOPS (% of peak)
GPT25B	420	3.28125	81.77225	77.867
GPT25B	400	3.125	90.62	86.297
GPT25B	360	2.8125	81.6325	77.7348
GPT25B	360	2.8125	82.6824	78.7346
GPT25B	192	1.5	115.8228	110.2927
GPT25B	128	1	106.672	101.5788
GPT25B	128	1	105.014	100.00

1. [throughput/TFLOPS](#)

Links

1. [!\[\]\(649a5212cc35ddc44f431622e9980b1a_img.jpg\) Hannibal046/Awesome-LLM](#) 
2. [!\[\]\(59526174b9cdf0d7b5d9f6af61b3d00d_img.jpg\) Mooler0410/LLMsPracticalGuide](#)
3. [Large Language Models \(in 2023\)](#)
4. [The Illustrated Transformer](#)
5. [Generative AI Exists because of the Transformer](#)
6. [GPT in 60 Lines of Numpy](#)
7. [Better Language Models and their Implications](#)
8. [!\[\]\(e44f5257037790c35b3082daa50e909a_img.jpg\) Progress / Artefacts / Outcomes from 🌸 Bloom BigScience](#)

Acknowledgements

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

References

- Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. “Attention Is All You Need.” <https://arxiv.org/abs/1706.03762>.
- Yang, Jingfeng, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Bing Yin, and Xia Hu. 2023. “Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond.” <https://arxiv.org/abs/2304.13712>.
- Yao, Shunyu, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. “Tree of Thoughts: Deliberate Problem Solving with Large Language Models.” <https://arxiv.org/abs/2305.10601>.

