Parallel Computing and AI, A love only relationship

YIFAN LI

December 25, 2022

1 Introduction

Inspired by Professor Liu's last lecture subtopic Deep Reinforcement Learning in Go, I was deeply intrigued by the power of combined AI (Artificial Intelligence) and Big Data. Back in 1997, chess was tackled by IBM's Deep Blue using Brute Force algorithm. Once considered as impossible game for a computer to beat an elite human go player, Recently, AlphaGo has scored landslide winning records against the two best go players in the world. Because unlike chess, Go is a game of extremely profound search space, the sheer combinatorial possibilities of every move is a number the scale of all atoms in the universe [Nor]. Implemented by the advanced deep learning methods, notable the Monte Carlo Tree Search, Alpha Go is able to learn from others, train itself, and eventually beat humans! None of these is possible without the High Performance Computing (HPC) platform that tirelessly enumerate all possibilities in the defined search space and return the optimized move.

Because most emphasis of the class are focused on the Machine Learning (ML) methods, which we implemented often with our programming assignments Deep Neural Networks, Clustering, Random Forests, Auto Encoder, and such. I would like to use this opportunity to explore more on the back end of the HPC side.

Why are HPC necessary for AI and Big Data? How are HPC implemented? How can a task be distributed to multiple processors to work individually? How can each individual's results be combined to produce a final output? In the case of Alpha Go, the HPC cluster consisted of 1,202 CPUs, 176 GPUs [SHM+16]. How can so many processors work seamlessly together without fighting with each other? What are some of the algorithms out there to implement parallel computing.

I would like to provide some high-level overview to help answer the above questions. With that purpose, the first half of the review is dedicated to HPC, and the second half presents the state-of-art GPT-3 modeling with HPC implementations on AI.

2 What is HPC?

Being no strangers to personal computers, we all have a good understanding of how powerful a computer is. A CPU with higher frequency, a RAM with higher capacity, a separate Graphics card instead of an integrated graphics processor, or an SSD instead of Hard Drive. A PC with a 3 GHz processor can make around 3 billion calculations in a second [net]. While an already very impressive speed, a HPC cluster can perform quadrillions of calculations per second [net]. Figure 1 shows what a typical commercial sized HPC cluster looks like.

As seen in Figure 2, an HPC cluster architecture has a network of high-speed computer servers called nodes. The head node serves like an administrator that control the users' interactions with HPC applications [Jet21]. The compute nodes serve like workers that perform calculations. With the effective and efficient parallel algorithms and cluster management, they can work together to produce results with super speed. During class, we have explored quite a bit of how tasks are scheduled for each of the worker nodes using algorithms such as MapReduce and Hadoop, and how node failure is managed such as using strategies like "fencing" by isolating the malfunctioned node or by denying access to the shared resources. I will continue to the next question of why HPC is necessary for AI, ML, or deep learning.

Figure 1: Commercial size HPC cluster [HPC]

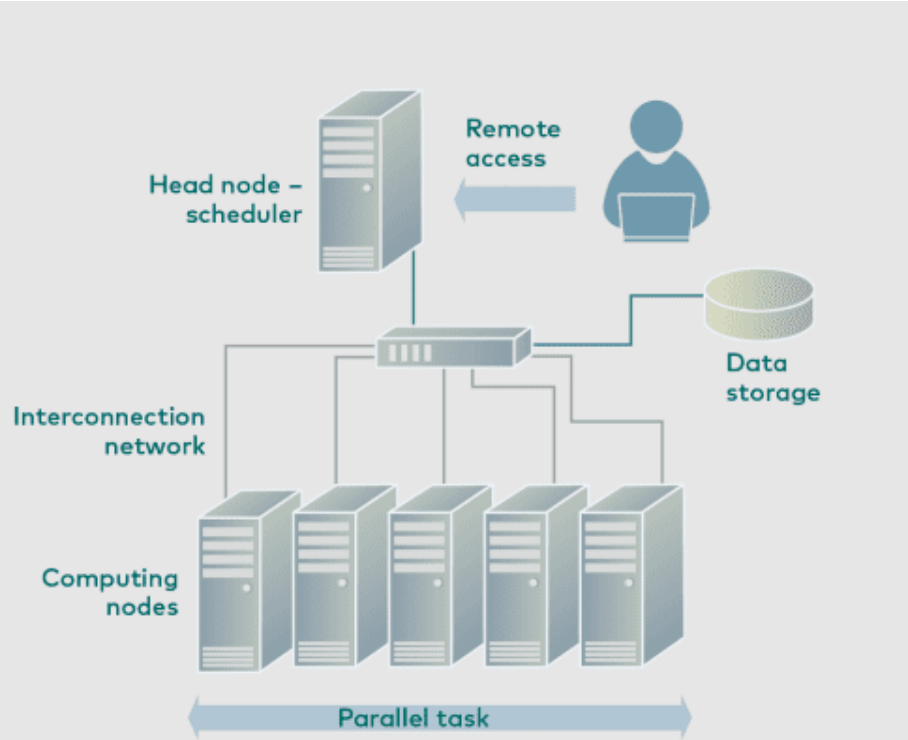


Figure 2: HPC Cluster Architecture [Jet21]

3 Why HPC?

With my own experience of training a simple neural network for the MNIST data set with only one hidden layer of size 30 using Stochastic Gradient Descent (SGD), that's $784 \times 30 + 30 = 23,550$ parameters to train, it took about one to two hours with CPU only to achieve prediction accuracy above 85%. To train a ML model with commercial impact, the number of parameters need to be trained on a much larger scale. Take the example of extreme case of the world's largest neural network GPT-3 ever-built, a super-intelligent NLP (Natural Language Processing) that even knows how to write code. It has 175 billion parameters. If I were to run it on my PC, it would take 848 years to finish. According to a paper presented by Sébastien Bubeck of Microsoft Research and Mark Sellke of Stanford University, they seemed to prove that "neural networks that are much larger than conventionally expected are necessary to avoid certain basic problems" [Ror22] and why bigger neural networks are necessary to do better.

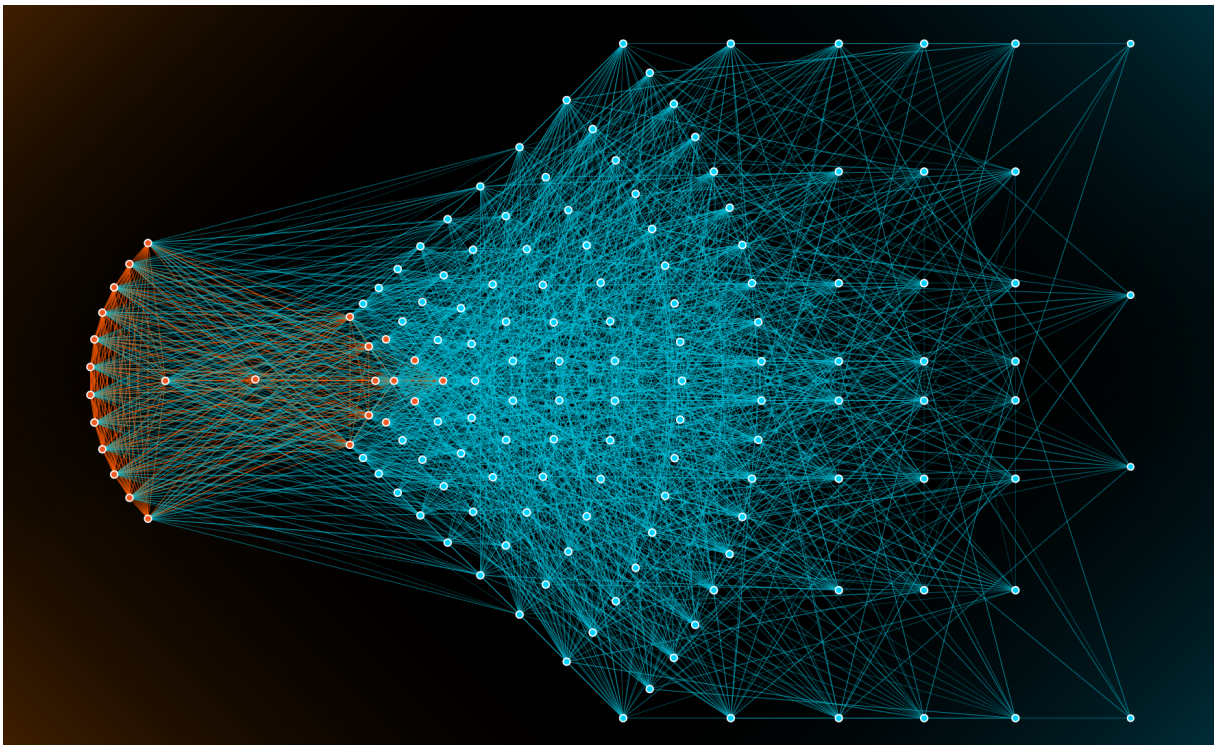


Figure 3: An Example Neural Network [Ror22]

There's no doubt that standard single sequential computer could not fit for the purpose of such big computing demand because sequential computation is ultimately bounded by the speed of light. In order to access the data stored in the memory, processor spends at least the time equal to its distance divided by the speed of light. "The only way to make a computer faster is to make its component closer" [Alu]. Faster computers also require much larger memory capacity to keep up with the processors. But there are only so many transistors you can pile up in one chip, as explained by Moore's law , which is why we need HPC and parallel computing to overcome this bottleneck.

Even with the hardware infrastructure available, we need efficient parallel algorithms to achieve parallel computing.

4 Parallel Algorithm

4.1 Goal

The goal of parallel algorithm is to speed up problem solving compared to just using sequential computers. To measure this performance gain, we can define speedup $S(p)$ as the ratio of running time of the best sequential algorithm to running time of the parallel algorithm using p processors. It has been proved that $S(p) \leq p$ and often

less than p because of some overhead cost in coordination among the processors [Alu]. Although in some cases superlinear speedup could happen, $S(p) > p$, such as in the case of tree searching, some processor may happen to search in the region where the solution lies.

On top of speedup, we need to consider the efficiency of the parallel algorithm. It's a crucial factor because the total cost of a parallel computer grows linearly as the number of processors increase. Economically it does not make sense to have an overly large number of processors to only achieve small gain in the running speed. We want a highly efficient parallel algorithm to utilizes as many cores as it has and leave as less cores in an idle state [Alu].

4.2 Model

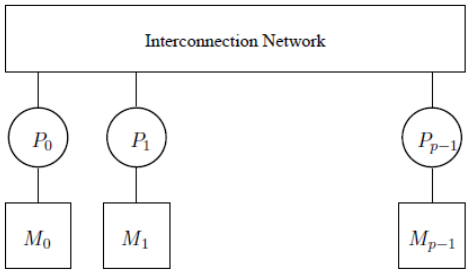


Figure 4: Parallel Networked Distributed Memory model [Alu]

As shown in the above figure, we construct a parallel computation model with p processors interconnected with a network. Each processor p_i can access its own memory M_i as fast as in a sequential computer, and can perform computations simultaneously. The communication between each processor can also be parallelized with additional time costs. They can be arranged in a permutation network such that each processor p_i will send one message to another processor p_j but no p_j will be same, meaning they all send to a different destination (see Figure 4.2. They can also be arranged in the multistage networks such as the Benes network as seen in Figure 5.

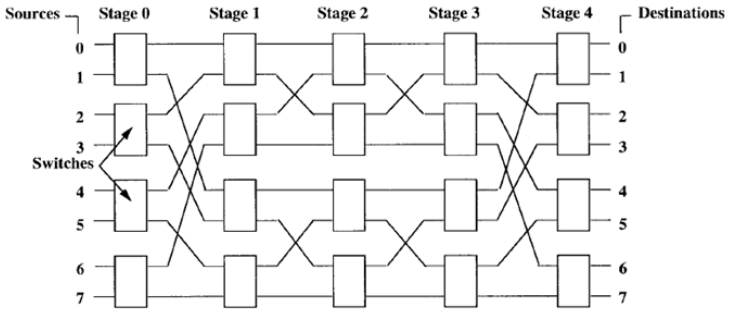


Figure 5: Benes network for $N = 8$ [JD86]

4.3 Two Example Parallel Algorithms

4.3.1 Example 1: parallel sum

I will show a simple instance of parallel algorithms using OpenMP API using C programming language. Let's say we want to calculate the sum from 1 to 1000. A typical sequential code would be:

```
#include <stdio.h>
void main()
{
    int i = 0;
    int sum = 0;
    for(i = 1; i <= 1000; i++){
        sum += i;
    }
    printf("The sum of 1 to 1000 is: %d", sum);
}
```

Let's say we have four threads with id 0, 1, 2, 3 available. A simple strategy would be let iterator i starts with id + 1, and increment step be the number of threads 4, then thread 0 will get the portion of iterations 1, 5, 9, 13, ...; thread 1 will get 2, 6, 10, 14, ...; thread 2 will get 3, 7, 11, 15, ...; and thread 3 will get 4, 8, 12, 16, ... In the last for loop, we can sum up all four thread's individual sum so that we can get the final result. (FYI, both scripts are written on my own).

```
#include <stdio.h>
void main2()
{
    int thrd_sum[4]; //create an empty array of size 4
    omp_set_num_threads(4); // set number of threads as 4
    int i; // shared variable
    int sum = 0; // shared variable

    #pragma omp parallel
    {
        int id = omp_get_thread_num(); //0,1,2,3
        int nthrds = omp_get_num_threads(); //will return 4
        int i; // private variable

        for (i = id; i <= 1000; i += nthrds){
            thrd_sum[id] += i;
        }
    }
    // end of parallel
    for(i = 0; i < 4; i++){
        sum += thrd_sum[i];
    }
    printf("The sum of 1 to 1000 is: %d\n", sum);
}
```

4.3.2 Example 2: Canon's method of Matrix Multiplication

Let's say we have two $n \times n$ matrices A and B and want to parallel compute multiplication on p processors. We can partition each matrix into p blocks of size $n/\sqrt{p} \times n/\sqrt{p}$ and we denote $A_{i,j}$ and $B_{i,j}$ as the submatrix for $i, j \in [0, \sqrt{p}]$. We then let processor $P_{i,j}$ perform the calculations of $A_{i,j}B_{i,j} = C_{i,j}$ (see step one). Next, we shift all $A_{i,j}$ to the left and $B_{i,j}$ upward one grid so that processor $P_{i,j}$ can perform multiplication for a new combination of submatrices and the result to the partial sum $C_{i,j}$. And repeat the process p times until all blocks have been multiplied [Sri18]. In the example shown in Figure 6. to Figure 9., a large matrix is partitioned in to 4×4 blocks and after 3 shifts, every block has been multiplied and the sum of all 4 partial sums is the final results.

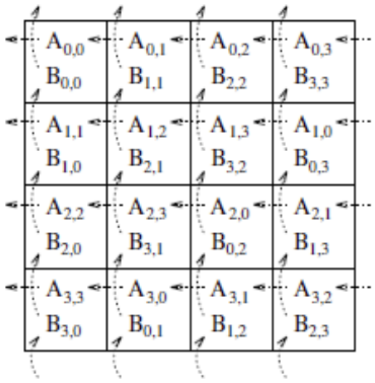


Figure 6: Step 1 [Sri18]

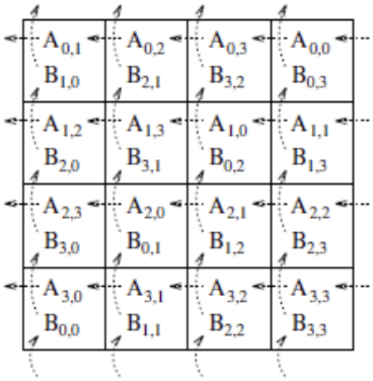


Figure 7: Step 2 [Sri18]

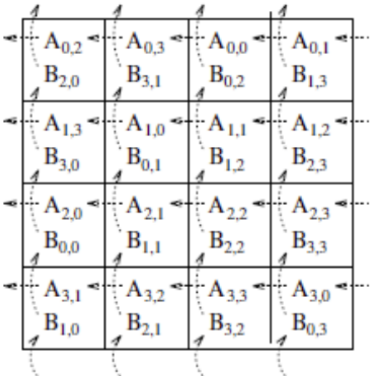


Figure 8: Step 3 [Sri18]

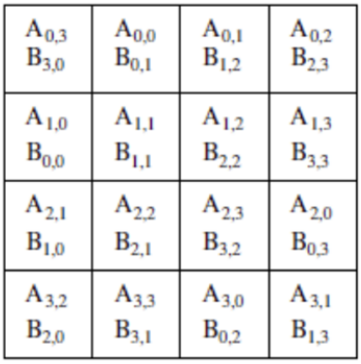


Figure 9: Last step [Sri18]

Machine Learning methods often involves million by million size matrix calculations, parallel algorithms such as Canon's method can significantly improve the running speed of such scale of calculations. The overall Speed up $S_p = \frac{n^2}{n^2/p} = p$ and Efficiency $E_p = \frac{n^2}{p \cdot (n^2/p)} = 1$ are considered ideal measures for parallel computing [Sri18].

4.4 Divide and Conquer

Another important strategy is the Divide and Conquer approach. A great example would be merge-sort (see Figure 10). As the name suggested, first a problem is divided into a number of subproblems of smaller size and solved recursively, then the results from the subproblems are combined to obtain the solution of the original problem, resulting in and $\mathcal{O}(n \log n)$ running time. It already sounded attractive for parallel computing since we can let each processor work on a subproblem concurrently. The divide step is not so hard but the merge part could be tricky. If we were to run this algorithm in parallel, we could easily divide the problem of size n in half and assign p/2 processors each. However, the merging step seems to be inherently sequential and usually replaced by bitonic sorting strategy [Alu].

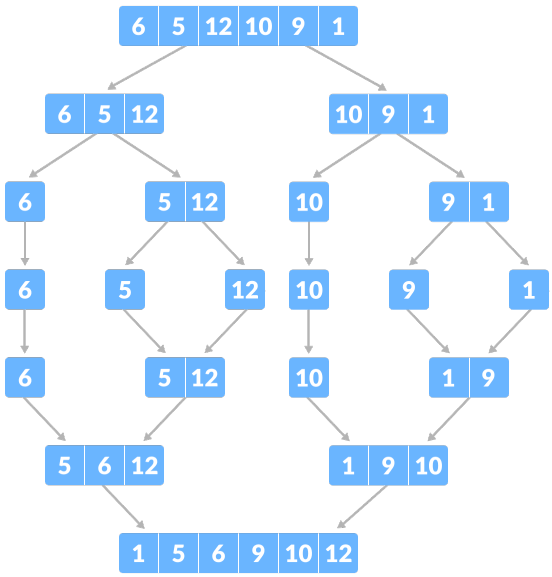


Figure 10: Merge Sort Example [pro]

4.5 Parallel implementation in Convolution Neural Network

Now that we understand the parallel calculation for matrix along with divide and conquer approach, we can implement convolution neural network training parallelly. For one kernel to slide over the whole image matrix, it's equivalent to perform matrix-vector multiplication. To perform convolution for multiple kernels in a single layer at a same time, the flattened vectors of kernels are stacked next to each other to form a matrix, hence equivalent to matrix-matrix multiplication, the most computationally intensive part of the process [HU16]. We are able to distribute that amount of data across multiple servers and individually compute such multiplications and it was not hard to do that with algorithms like the Canon's method. The model training part is

more complex as we know forward and backward propagation are inherently sequential so it cannot be just divided and perform individually. But we can still achieve some sort of parallelism from letting each worker node conducts full sequential training of the network for one portion of the training, the head node can then perform averaging procedure on results to extract final weights [GS].

5 State-of-the-Art GPT

5.1 GPT background

NLP (Natural Language Processing) has been advancing forward in a fast pace in recent years as we enjoy the convenience it brought to us such as getting immediate answers from conversations with AI, like Apple's Siri, Amazon's Alexa, or Microsoft's Cortana, etc. With evidence based on experiences and observations, NLP models trained with large data sizes on the order of millions proved much more useful to tackle tasks such as composing articles, answering questions, and inferring hypotheses [DCLT18, RWC+19]. One of the State-of-the-Art of such models is the GPT-3. GPT stands for Generative Pre-trained Transformer. It is an autoregressive language prediction model developed by OpenAI using deep learning. As of today, the third generation GPT-3 is the largest non-sparse language model (LM), trained with a whopping size of 175 billion parameters that require 800 GB of data storage and with token size of 2048 . It is capable of producing human-like texts [BMR+20, Ben22] that are so sophisticated such as rhyming poetry, lyrics, translation, dealing with occasions that require "on-the-fly reasoning or domain adaptation". It becomes hard to tell from real human produced texts.

5.2 Model Architecture

The GPT-3 adapted multitask neural-net based Transformer architecture that focuses on Attention from paper *Attention is all you need* [VSP+17]. The Transformer architecture is known to handle sequential text data well and excels at performance compared to recurrent neural network (RNN) [Dos21a]. Similar to the RNN based encoder decoder process, the transformer-based encoder part maps the input sequence to a sequence of hidden states, then the decoder maps the hidden states and all previous target vectors to define the conditional probability distribution of the output sequence from which it can be auto-regressively generated [tra]. Note that the distribution of the target vector is explicitly conditioned on all previous target vectors in Transformer but implicitly conditioned on in the case of RNN due to its sequential nature [tra]. The gist of Transformer lies in the extra bi-directional self-attention layers that projects each input vector of an input sequence to a key vector, value vector, and query vector through trainable weight matrices [Dos21a]. To help understand, I tend to think of query and key vector as conveyor of information among tokens, value as the information extracted by the current token. In summary, the self-attention function uses query and key to compute the similarities among tokens in the current layer, and the weighted sum of value is

the token for the next self-attention layer [VSP+17]. As mentioned, in the RNN based model, the computation of the hidden state must be done sequentially, in contrast to the direct matrix multiplication in the Transformer [tra]. Therefore, Transformer has significant advantages with the ability of parallelization and long-range contextual representations [tra].

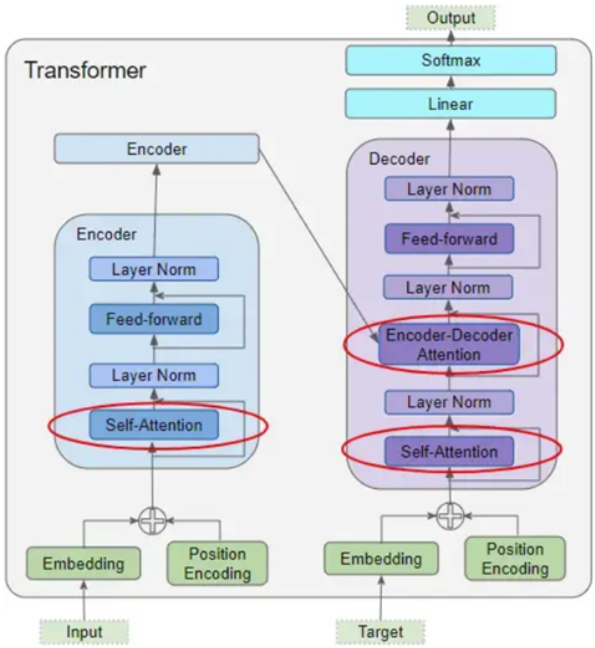


Figure 11: Transformer Layout [Dos21b]

The model generally followed the steps detailed in [RWC+19] of a combination of unsupervised pre-training and supervised fine-tuning [BMR+20, RWC+19]. The pre-training follows a standard LM to maximize the likelihood of conditional probability of every token in the neural network model using stochastic gradient descent (SGD) [RWC+19]. The neural network model follows a multi-layer Transformer decoder (no encoder) [RNS+18]. In the multi-head attention sub-layer in GPT-3, the attention patterns were alternated between dense and locally banded sparse way [BMR+20]. The decoder applies position-wise feedforward layers to produce an output distribution [18]. Simply speaking, it is auto-regressive process that takes context vectors of tokens as input and generates probability estimates of the next word as output. Next step is to adapt the trained parameters to the supervised target task using a labeled dataset [RNS+18].

On top of feeding the extra inputs into previously pre-trained model and obtain the final transfer block's activation layer, an extra linear output layer is added to help predict the labels, which is called an auxiliary objective function to maximize the

labels' conditional probabilities over labeled training data set [18]. This step is the fine-tuning supervised part of the modeling process that could help accelerate convergence [RNS+18]. This step is also necessary and easily modifiable to cope with tasks more than just text classification such as "question answering or text entailment" without big changes to the existing architecture [RNS+18].

Since GPT-3 cares more about the performance in task-agnostic fashion, it skipped the fine-tuning part instead was given training examples in a Few-Shot (FS), One-Shot (1S), and Zero-Shot(0S) way. However, OpenAI team looks forward to fine-tuning for future improvement. As opposed to the fine-tuning method that uses a big training data set to pursue high accuracy, FS only feeds the model with few training data to guide its predictions at inference time. Similarly for 1S and 0S, only one example or no example is demonstrated for the model in addition to the task description in natural language [BMR+20]. Figure 12 list one example for each setting. The model was also later evaluated at these three settings given new test data sets.

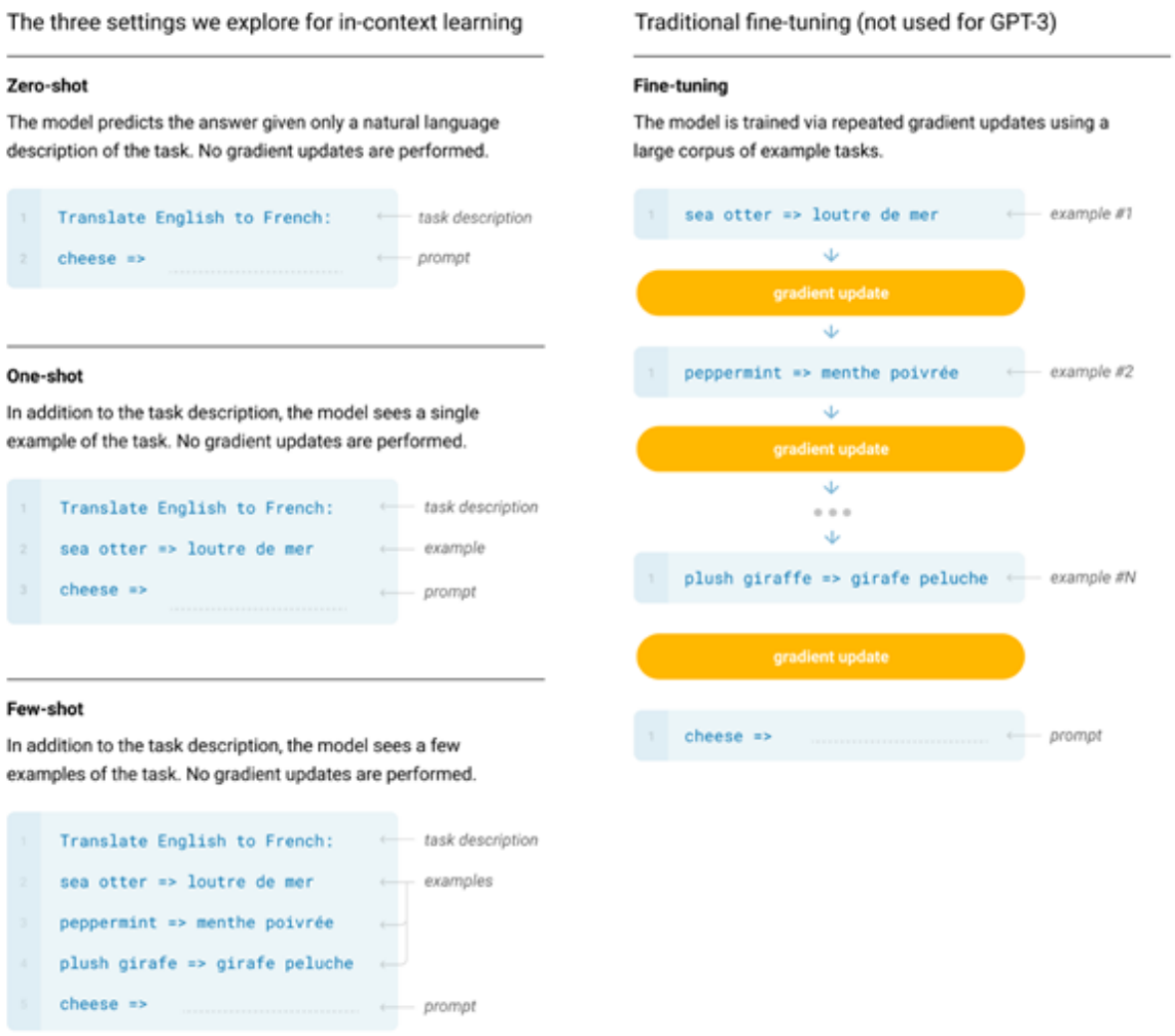


Figure 12: Four cases of in-context learning [BMR+20]

5.3 Training Datasets

The largest training dataset GPT-3 consists of 60% filtered Common Crawl constituting nearly a trillion words or 410 billion byte-pair-encoded tokens equivalent, "which process structured text input as a single contiguous sequence of tokens" [BMR+20, SHB15, RWC+19]. On top of that, several high-quality curated datasets are added, including 19 billion tokens from WebText, 67 billion tokens from two internet based Book1 and Book2, and 3 billion tokens from Wikipedia [BMR+20, RWC+19].

5.4 Parallel Computing

Model parallelism is essential to train the above models with such scale of sizes because they exceed the memory limit of all modern processors. Since GPT used ADAM optimizer, additional memory per parameter is required, resulting even further reduction the size of models that can be well trained [BMR+20, SPP+]. OpenAI team partitioned the model across V 100 GPUs both in width dimension and depth dimension provided by Microsoft's high-bandwidth cluster [BMR+20]. The parallel approach used in this case is the intra-layer model-parallelism [SPP+].

The two essential paradigms for scaling out model training through hardware accelerator such as GPU rendering are data parallelism and model parallelism [SPP+]. The former splits a training mini-batch and the latter splits memory usage and computation across multiple workers [SPP+]. The team furthered the model parallelism by utilizing distributed tensor computation from insights leveraged in Mesh-TensorFlow [SCP+18]. As mentioned before, Transformer Model has the structure advantage to implement parallelism and that's exactly what the team had exploited. As briefly introduced in the model architecture section, the transformer layer consists of two sublayers of an Attention block and a multi-layer perceptron (MLP) layer and both of them can apply parallelism [SPP+].

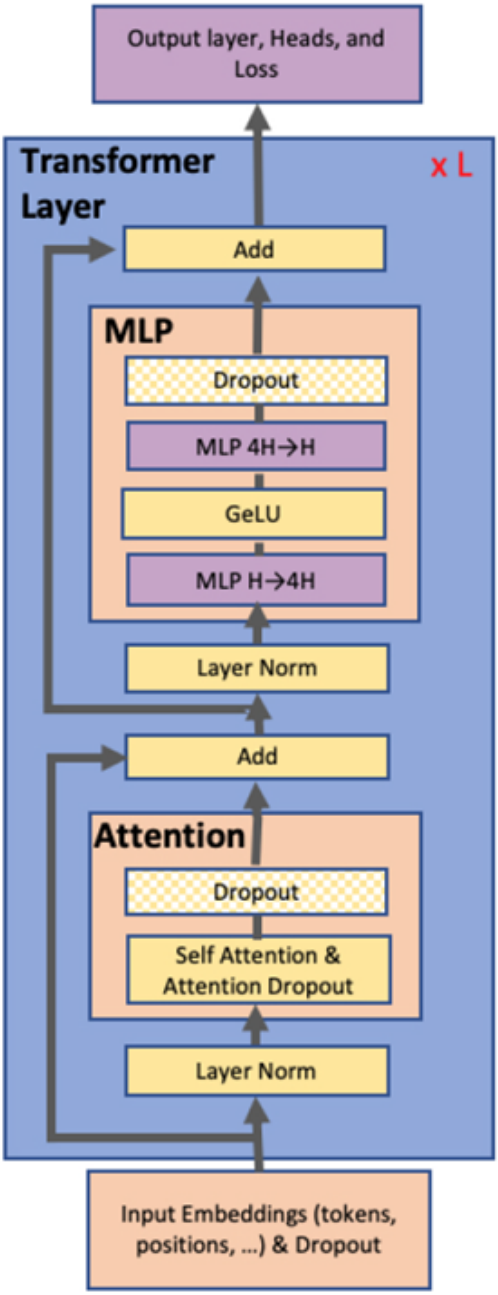
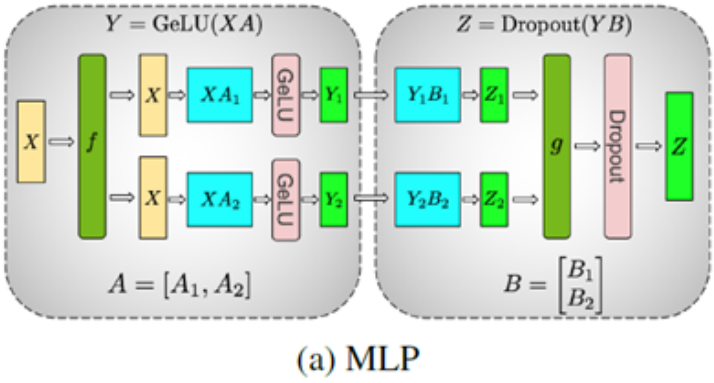


Figure 13: Parallel Networked Distributed Memory model [VSP+17]

The team started to tackle the MLP block that consists of general matrix multiplication (GEMM) followed by GeLU (Gaussian Error Linear Unit) activation function [SPP+]. They partitioned the first GEMM column wise ($A = [A_1, A_2]$, apply GeLU and then partition the second GEMM row wise, with no need to communicate. Therefore, the whole process can be reduced

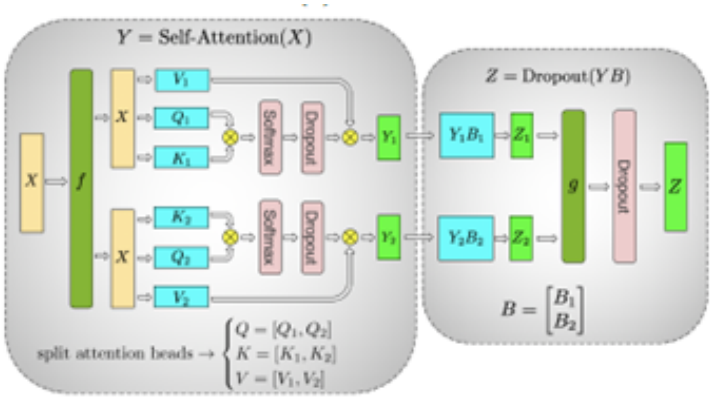
across GPUs and only requires a single all-reduce operation in both the forward pass (g operator) and backward pass (f operator) (see Figure 14).

The self-attention block can be set up in similar way by partitioning the first GEMMs column wise to multiply by key (K), query (Q), and value (V) such that each GEMM can be done locally on one GPU [SPP+]. And the subsequent GEMMS are split row wise to feed into the final Dropout layer. Further parallel efforts were also done in the input and output embedding GEMMs. The team especially focused on the technique to engineer the parallel structure with as few communication needs as possible and to keep each GPU memory performance to the expected maximum (see Figure 15).



(a) MLP

Figure 14: Multi Layer Perceptron [SPP+]



(b) Self-Attention

Figure 15: Self-attention Block [SPP+]

A baseline was established to demonstrate the scalability of the aforementioned approach. On a single NVIDIA V100 32GB GPU, 39 TeraFLOPS (39×10^{12} floating point operations per second) is achieved to train a model with 1.2 billion parameters, which

serves as a strong baseline [SPP+]. The GPT-2 model trained with 8.3 billion parameters with 8 GPUs parallelism achieved 77%, with 512 GPUs achieved 74% of linear scaling efficiencies compare to the baseline [SPP+]. The results are really promising.

5.5 Model Results

The largest NLP model so far, GPT-3 was trained with 175 billion parameters with 96 total number of layers in the neural network, 12,288 units in each bottleneck layer, and a batch size of 3.2 million samples using Adam Optimizer. To illustrate the scalability, 7 models with smaller size of parameters were also trained (see table below) [BMR+20].

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Figure 16: Table of Models Trained [BMR+20]

Model was evaluated on different task settings including Cloze tasks like filling in the blanks in a sentence or paragraph context, "closed book" questions by answering given information stored in the model's parameters, language translation tasks, Winograd schema challenge (WSC) type of tasks [wiki], reading comprehension tasks, on-the-fly reasoning, etc. [BMR+20]. Here I will only select a few test cases from the paper to demonstrate the model's test results.

The LAMBADA data set [PKL+16] was used to test the model's ability to predict the last word of sentences given reading contexts. GPT-3 achieved 8% and 18% performance gain on previous state-of-art (SOTA) models with 76% and 86.4% accuracy in the 0S and FS settings respectively [BMR+20]. The HellaSwag dataset [ZHB+19] also demonstrated GPT-3 superior modeling in long-range dependencies in text since this data set was intentionally made hard for testers to choose the best ending to a story [BMR+20]. GPT-3 achieved 78.1% accuracy with respect to humans' 95.6% accuracy, which is quite impressive [BMR+20].

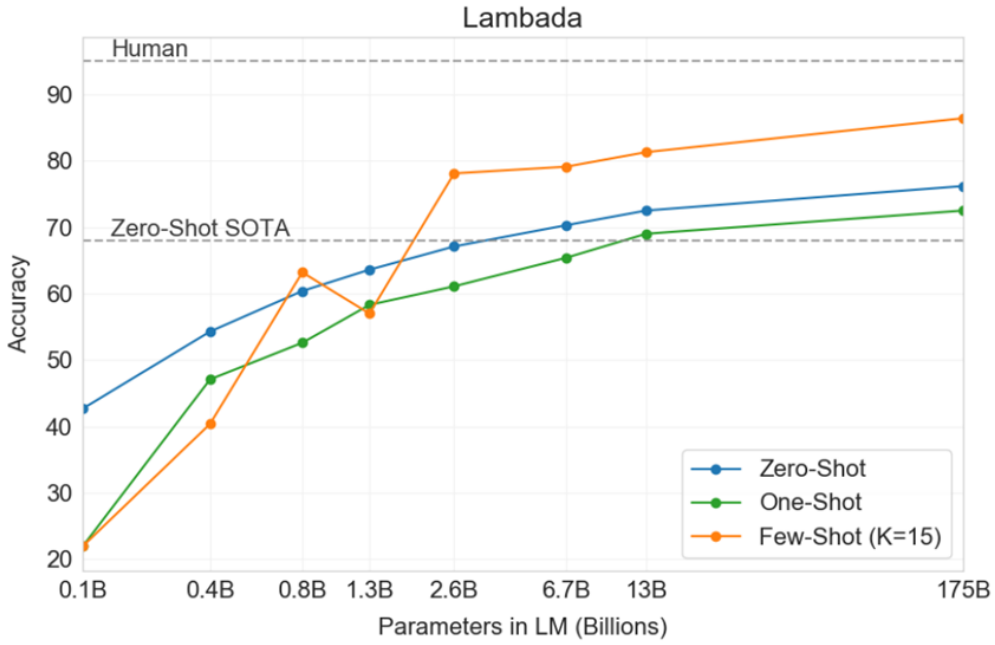


Figure 17: Lambada Test Results [BMR+20]

The WSC task [LDM12] is to test the machine to identify the grammatical antecedent of an ambiguous pronoun but semantically unequivocal for humans. For example:

The cake doesn't fit in the box because it's too big. What is too big?

- Answer 1: the cake
- Answer 2: the box

WinoGrande example [SBBC21]:

The turkey is warmer than the chicken because **it** was in the oven for a *longer* amount of time?

The turkey is warmer than the chicken because **it** was in the oven for a *shorter* amount of time?

- Answer 1: **turkey**/chicken
- Answer 2: **chicken**/turkey

The GPT-3 was tested on two Winograd dataset, one is the original, the other called WinoGrande, inspired by Winograd, is the more difficult version as it contains a lot more misleading datapoints [SBBC21]. GPT-3 achieved great accuracy as opposed to the SOTA models (see Figure 18).

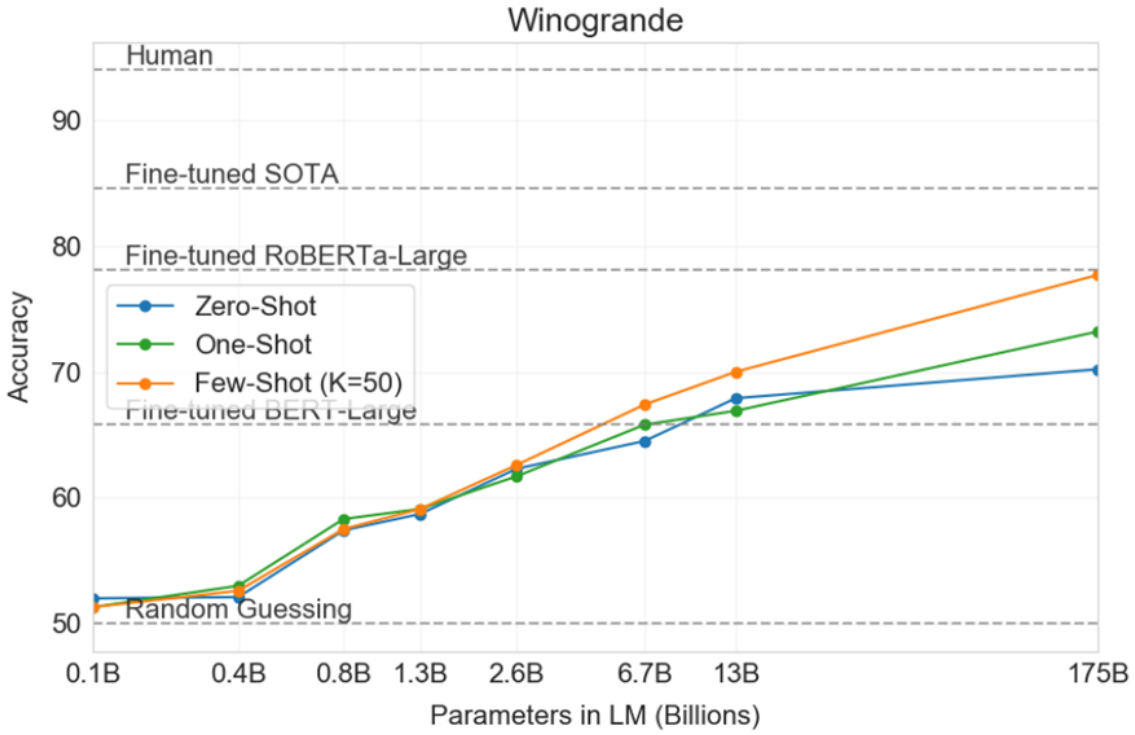


Figure 18: Winogrande Test Results [BMR+20]

All the experiment results suggest a great leap forward on AI capabilities in natural languages. As we can see from all the trendlines, the more parameters a model has, the more accuracies improved by orders of magnitude, certainly within the limit of not overfitting. These obviously cannot be achieved without expansion of hardware and data sizes, as well as efficient parallel algorithms that allow the models to be trained in a reasonable timespan.

6 Summary

There is no doubt AI technology has already become part our essential life improvement needs. It is not possible to achieve the sophistication of what we have today without HPC with Parallel Computing. No need to stress more and I would like to conclude my report with a poem compposed by GPT-3 describing such an important relationship between AI and Parallel computing.

Parallel computing and AI,
A perfect match, as all can see.
They work together, hand in glove,
To solve problems with speed and love.

With parallel processing, they can fly,
Testing and refining algorithms on the fly.
And with the power of AI,
Parallel computing can reach the sky.

They're a dynamic duo, no doubt,
Advancing technology without a doubt.
Their partnership will only grow stronger,
Bringing us closer to a smarter, brighter future.

References

[Alu] Srinivas Aluru. Cse 6220 / cx 4220 class notes.

[Ben22] Edwards Benj. Openai upgrades gpt-3, stunning with rhyming poetry and lyrics, Nov 2022. URL: https://arstechnica.com/information-technology/2022/11/openai-conquers-rhyming-poetry-with-new-gpt-3-update/.

[BMR+20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[Dos21a] Ketan Doshi. Transformers explained visually (part 1): Overview of functionality, Jun 2021. URL: https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452.

[Dos21b] Ketan Doshi. Transformers explained visually (part 3), Jun 2021. URL: https://towardsdatascience.com/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853.

[GS] Dan Grau and Nick Sereni. Parallel computing for neural networks. URL: http://meseec.ce.rit.edu/756-projects/spring2013/1-4.pdf.

[HPC] Hpc and the future of seismic. https://www.pgs.com/company/newsroom/news/hpc-and-the-future-of-seismic/.

[HU16] Vishakh Hegde and Sheema Usmani. Parallel and distributed deep learning. *May*, 31:1–8, 2016.

[JD86] Hemant K Jain and Amitava Dutta. Concepts, theory, and techniques distributed computer system design: A multicriteria decision-making methodology. *Decision Sciences*, 17(4):437–453, 1986.

[Jet21] Hitesh Jethva. What is hpc? high performance computing and how it works, Dec 2021. URL: https://cloudinfrastructureservices.co.uk/what-is-hpc-high-performance-computing-and-how-it-works/.

[LDM12] Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Thirteenth international conference on the principles of knowledge representation and reasoning*, 2012.

[net] What is high performance computing. https://www.netapp.com/data-storage/high-performance-computing/what-is-hpc/.

[Nor] Peter Norvig. http://norvig.com/atoms.html.

[PKL+16] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

[pro] Merge sort algorithm. https://www.programiz.com/dsa/merge-sort.

[RNS+18] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[Ror22] Rorvig. Computer scientists prove why bigger neural networks do better, Feb 2022. https://www.quantamagazine.org/computer-scientists-prove-why-bigger-neural-networks-do-better-20220210/.

[RWC+19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[SBBC21] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

[SCP+18] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. Mesh-tensorflow: Deep learning for supercomputers. *Advances in neural information processing systems*, 31, 2018.

[SHB15] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[SHM+16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[SPP+] M Shoeybi, M Patwary, R Puri, P LeGresley, J Casper, and B Megatron-LM Catanzaro. Training multi-billion parameter language models using model parallelism. arxiv 2019. *arXiv preprint arXiv:1909.08053*.

[Sri18] Kyatham Srikanth. Cannon's algorithm for distributed matrix multiplication, Oct 2018. https://iq.opengenus.org/cannon-algorithm-distributed-matrix-multiplication/.

[tra] Transformer-based encoder-decoder models. URL: https://huggingface.co/blog/encoder-decoder.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[ZHB+19] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.