Training Multi-lingual Transformer Models in a Resource Constrained Environment

John Che, Ashish Priyadarshi, Sree Kandasamy

1 Introduction

In recent years, the proliferation of large language models (LLMs) has significantly advanced the field of Natural Language Processing (NLP), enabling state-of-the-art performance across a wide range of tasks such as translation, summarization, and question answering. While these models have shown remarkable results on high-resource languages, particularly English, their performance and accessibility in multilingual contexts remain limited. This gap arises primarily from two systemic challenges: the underrepresentation of low-resource languages in training corpora and the high computational demands of training and deploying multilingual LLMs (MLLMs) [1].

Multilingual language models hold the potential to democratize access to NLP technology across linguistic boundaries, thereby fostering equitable information access and communication. However, low-resource languages often lack extensive parallel corpora, making it difficult to train high-quality multilingual models without significant overfitting or loss in generalization [8]. As a result, most state-of-the-art models, such as mBERT and XLM-R, show diminishing returns in performance as the number of supported languages increases [9]. Additionally, these models are often not optimized for generative tasks or cross-lingual transfer in resource-constrained environments.

To address these limitations, the mT5 model was introduced as a multilingual extension of the T5 architecture, trained on the mC4 corpus comprising 101 languages [14]. While mT5 demonstrates improved generalization across sentence-pair classification, structured prediction, and multilingual QA tasks, the full-scale version of the model requires over 50 GB of memory and substantial compute resources, rendering it impractical for many researchers and institutions operating with limited hardware budgets.

This project investigates the feasibility of training performant multilingual generative models using low-cost, widely accessible hardware. We explore the use of model parallelism and data parallelism to scale training on clusters of 12GB GPUs, leveraging PyTorch's Distributed Data Parallel (DDP) framework. By clustering four low-memory nodes, we achieve significant run-time reductions and competitive evaluation metrics on multilingual translation tasks. Our approach demonstrates that, with careful architectural and systems-level optimizations, it is possible to train compact and efficient MLLMs capable of covering over 70 languages and 19 NLP tasks, even in resource-constrained settings.

2 Background

Natural Language Processing (NLP) research has increasingly relied on computationally intensive training procedures to achieve state-of-the-art (SOTA) performance. For instance, training large-scale language models such as GPT-3 REPLACE (Brown et al., 2020) requires thousands of high-end GPUs, incurring substantial financial and environmental costs. This reliance on expensive hardware creates a significant barrier to entry for researchers with limited access to computational resources both in the US and in other parts of the world.

Moreover, even for researchers with sufficient funding, the availability of high-performance GPUs (e.g., NVIDIA A100 or H100) remains constrained, both inside and outside the United States [5]. This scarcity necessitates innovative approaches to leverage more accessible, lower-cost GPUs (e.g., NVIDIA P100) in configurations that can approximate the training throughput of high-end hardware.

2.1 Project Objective

Through our work, we aim to develop an efficient distributed training architecture that integrates low-cost, readily available GPUs (such as the P100) to train multilingual language models (MLLMs) with moderate to competitive performance. By optimizing the use of modest, low-cost and readily available hardware through advanced parallelism techniques, we seek to democratize access to State-of-the-art (SOTA) NLP research, enabling broader participation from resource-constrained institutions and environments.

2.2 Technical Background

Transformer Models

Transformer models are neural networks that make use of self attention mechanisms, mapping every input token to every other input token to capture dependencies within text. The transformer architecture has largely replaced RNNs when it comes to natural language tasks and is the foundation for LLMs. BERT, t5, and more are all based on this architecture. Transformer models stack multiple encoder and/or decoder layers. Each layer consists of multiple multi-headed attention blocks along with feed-forward networks that have intermediate connections and normalizations.

Distributed Data Parallelism (DDP)

Distributed Data Parallelism (DDP) is a machine learning training paradigm that parallelizes model training across multiple computational nodes. In this framework, the dataset is partitioned, and each node containing either a GPU or TPU processes a distinct subset of the data. DDP is implemented in PyTorch via the DistributedDataParallel module, which ensures synchronized training through gradient aggregation.

DistributedSampler

The DistributedSampler library in PyTorch handles critical aspects of data distribution in a Distributed Data Parallel (DDP) framework. It partitions datasets into non-overlapping shards, ensuring each processing node receives unique data subsets to prevent redundancy during training. The sampler maintains epoch consistency through a shared random seed, enabling reproducible shuffling while regenerating data permutations at each epoch. Additionally, it implements load balancing mechanisms to handle cases where the dataset size is not evenly divisible by the number of processes, ensuring efficient resource utilization across all nodes.

DistributedDataParallel Library

PyTorch's DistributedDataParallel (DDP) library provides an efficient implementation for scaling deep learning training across multiple GPUs with near-linear acceleration. The framework operates by maintaining identical model replicas across all computational nodes while synchronizing their training states. Gradient synchronization is achieved through optimized collective communication operations, particularly the AllReduce primitive, which aggregates and distributes gradient updates REPLACE (Li et al., 2020). Unlike other the basic DataParallel (DP) approaches limited to multiple GPUs/TPUs on a single-machine deployment, DDP supports true distributed training across multiple nodes, making it suitable for large-scale experiments.

Dataset Choice: M2Lingual

We chose to work with the M2Lingual dataset because it provides an extensive framework for evaluating multi turn conversations between a human and an AI agent. The M2Lingual [9] dataset provides the conversational flow that we would like to fine tune the mt5_small model on.

Although the mt5 model is already pretrained on the mC4 corpus which covers 101 languages [14], fine tuning is required to have it behave more conversationally. Through the mC4 corpus, mt5 never sees the workflow of a human asking a question and the AI producing a reply. Additionally, the M2Lingual dataset introduces the Evol taxonomy which creates a much more robust dataset.

The success of LLMs is driven through instruction-fine-tuning (IFT) datasets written primarily in English. The M2Lingual dataset expands coverage by focusing on low resource languages and covers 77 languages making it an effective dataset to train for the multilingual use case. Additionally, the M2Lignual dataset addresses concerns that arise via human-involved datasets. Typical multilingual datasets require native speakers which can lead to annotation errors, uneven data distribution, and can lead to privacy concerns.

Thus, the M2Lignual dataset is fully synthetically generated. They first select diverse multilingual seeds from the Aya dataset which has a high approval rating from annotators. They combine this with one of 17 NLP tasks (summarization, cross lingual summary, joke explanation, etc.) and feed it to an LLM (GPT-4) which develops an evolved prompt. The LLM then answers this evolved prompt giving us the assistant reply. This model also accounts for multi turn prompting, thus giving us a rich dataset to train on conversational flow.

CloudLab

CloudLab serves as a distributed research infrastructure hosted across three academic institutions: the University of Utah, Clemson University, and the University of Wisconsin-Madison REPLACE (Duplyakin et al., 2019). This platform provides researchers with flexible access to customizable hardware configurations through rapid provisioning capabilities. A key feature of CloudLab is its performance isolation mechanism, which ensures consistent experimental conditions for reproducible systems research. The platform's accessibility and control features make it particularly valuable for investigating distributed training architectures, serving as an ideal testbed for our proposed research framework.

2.3 Significance and Contribution

By optimizing DDP for low-cost GPUs, our work addresses two identified critical challenges in NLP research:

1. Cost Efficiency: Reducing reliance on prohibitively expensive hardware.

2. Accessibility: Enabling researchers with limited infrastructure to conduct scalable experiments.

Our architecture demonstrates that SOTA-comparable performance can be achieved through intelligent resource allocation and parallelism, broadening participation in cutting-edge NLP research. Future work will explore trade-offs between hardware constraints and model performance, as well as extensions to other resource-intensive domains.

3 Design and Architecture

3.1 Architecture Setup and Requirements

Our experimental architecture integrates four NVIDIA P100 GPUs, collectively providing 48GB of GPU memory (12GB per GPU), a configuration designed to approximate the computational capacity of a single state-of-the-art A100 GPU. To accomplish our primary objective, we plan on training the multilingual language model Google/mT5-small on this distributed setup. Then, we will benchmark its performance against the same model trained on an A100 GPU. The P100 GPUs were provisioned through CloudLab, with a total allocation of 30 training hours. In contrast, the A100 GPU was accessed via Google Colab credits.

Notably, the cost efficiency of our approach is significant: while each P100 GPU costs approximately \$300 (totaling \$1200 for the four-GPU cluster), a single A100 GPU retails for around \$8,300, representing a substantial reduction in resource expenditure for comparable memory capacity.

3.2 Model Selection: Google/mT5-small

For our experiments, we selected the Google/mT5-small model, a pretrained multilingual text-to-text transformer developed as part of the mT5 (Multilingual T5) model family. This model extends the original T5 (Text-to-Text Transfer Transformer) architecture to support 101 languages, making it highly versatile for multilingual applications. The mT5-small variant employs an encoder-decoder transformer architecture with relative position embeddings and tokenization. With approximately 300 million parameters, it is computationally feasible for deployment on individual P100 nodes, unlike larger variants such as mT5-base (580M parameters) or mT5-large (1.2B parameters). Additionally, the model's compatibility with contemporary checkpointing techniques enhances fault tolerance during extended training sessions.

Below is an illustration of our distributed training architecture.

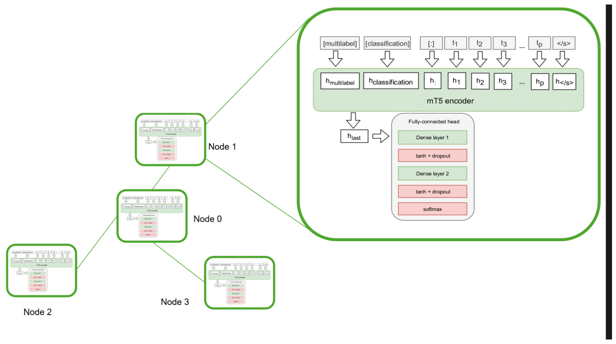

Figure 1: P100 Distributed Architecture containing 4 nodes

This configuration demonstrates how distributed training can be leveraged to approximate high-end GPU performance using cost-effective, readily available hardware. The following sections detail our experimental methodology and benchmarking results.

4 Evaluation

4.1 Evaluation setup

Measuring the performance of transformer models and LLMs have always provided challenges especially in the context where multiple potential answers are acceptable. Generally, the predicted text should be close to the reference text provided and should cover the important context of the reference text. In our case, we compare the generated text from our fine tuned MT5 model to the "output assistant reply" from the M2Lingual dataset. In the M2Lingual dataset, the "output assistant reply" column represents the AI agent's reply to the evolved prompt thus serving as the benchmark for evaluating generated text against.

4.2 Evaluation metrics for Transformer models

Evaluation for transformer models is built around two main metrics: BLEU and ROUGE. BLEU measures precision in the context of comparing n-grams of the machine generated text to the n-gram of human translated text. In our case, we measure the n-grams of our mt5 model's generated text to the generated text from the M2Lingual dataset. Meanwhile, ROUGE acts as the recall metric for transformer models. ROUGE-n measures the precision of n-gram overlap between the generated text and the reference text. For both metrics, we evaluated and considered the unigram scores.

4.3 Runtime comparisons

In addition to evaluating the model's performance itself, we compared runtimes of running an inference model (training time and generation time) in a distributed environment with PyTorch's DDP versus a non-distributed setup. Comparing runtimes will show the time savings that DDP offers.

4.4 Performance per dollar

The other biggest point of evaluation we considered was the performance of the model relative to the cost of the hardware it was being run on. Our main focus is to show that the DDP environment can negate the runtime gains from more expensive and compute heavy GPUs as by providing equal performance for a far lower cost. Therefore, we consider the performance of our DDP environment on the NVIDIA Tesla P100 hardware against a baseline performance of the same model running without the distributed setup on an NVIDIA A100 GPU.

4.5 Training setup and parameters

For our evaluation, we ran the mt5_small model (300 million parameters) first on a single node and two node setup to provide initial performance results on the seed dataset to gain a baseline for what heavier versions of our training and generation could look like when running it on the full dataset. The seed dataset contains 13,700 examples and the full dataset is expanded to 174,000 examples. We took a train-test split of 0.9 to 0.1 and ran training for three epochs.

We then performed training on the full dataset which included additional features like turns, language, task, evolved_user_prompt, and we used the output_assistant_reply as the target text for comparison and evaluation purposes. We again first ran training on only 13,000 examples from the full dataset before finally running training on all examples.

4.6 Runtime evaluation and takeaways

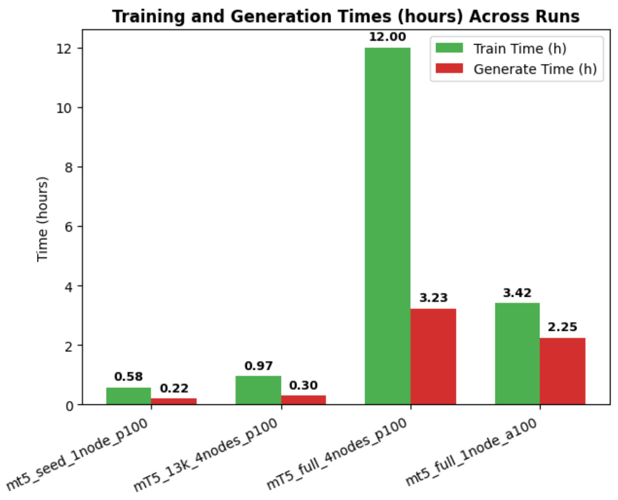

Figure 2: Comparison of training and generation times across different workloads

From the runtime graphs (figure 2), we gain key insights into the savings that DDP provides. The one node setup for the seed dataset shows that training and generation times are very small, however, increasing the parallelization of the model to a four node setup would reduce training by 75%. These differences aren't large for smaller datasets and could be seen as more complex to configure or add additional network latency. The differences are much larger when comparing run times on the full dataset.

However, the time savings scale as training time increases (Figure 3).

Therefore, for larger models and larger datasets, scaling to DDP offers a very effective way of reducing runtime and therefore saving GPU load and utilization.

4.7 BLEU & ROUGE Evaluation and Takeaways

We can also see that DDP does not show a performance drop when it comes to the evaluation metrics (Figure 4).

State of the art MLLMs achieve BLEU and ROUGE scores of 40-60% while scores between 10-30 indicate the gist of the generated text is clear while there

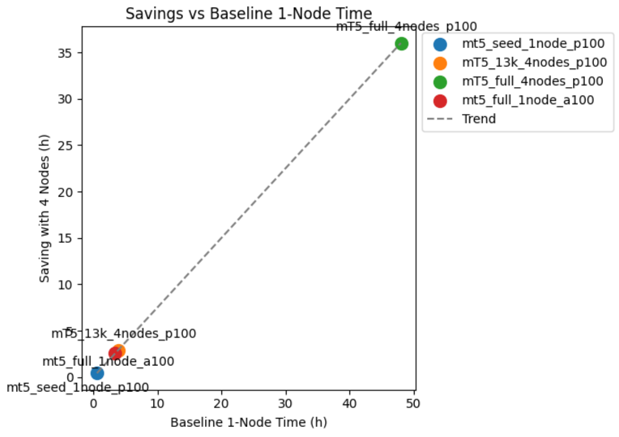

Figure 3: Saving vs Baseline across different distributed configurations

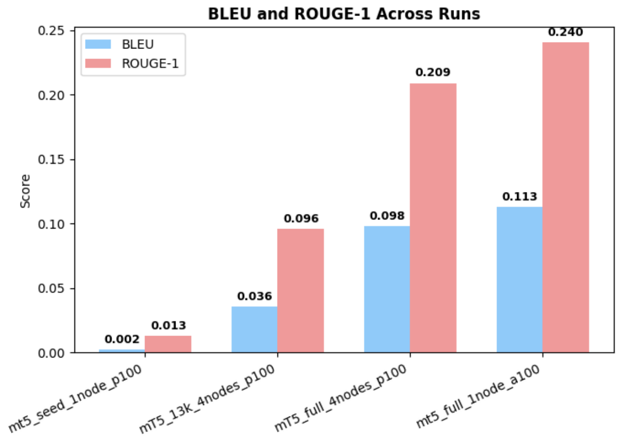

Figure 4: BLEU and ROUGE scores across different workload samples

are grammatical errors [4], and our model was able to achieve a BLEU of 10% and a ROUGE of 21% when ran in the DDP setup. Keeping in mind that training was only done for 3 epochs, our performance expects to scale as we increase the epochs and reach understandable to good translations and even high quality translations.

This is especially seen when we compare the performance of the distributed four GPU setup utilizing the P100 GPU compared to the single node setup on the A100 GPU. This is a 15.3% increase in BLEU score and 14.8% increase in ROUGE-1 score.

4.8 Performance per dollar evaluation and takeaways

However, the cost of a P100 GPU is roughly in the $100-300 while an A100 costs around $8,0000. We can see the benefit in performance is not comparable to the runtime gains or the performance gains. This is illustrated in the following figure (figure 5). For more than six times the cost, the performance benefits don't nearly come close to matching that of the inflated cost. Additionally, run time gains are minimal when comparing to a single node setup when compared to cost inflation. Distributing an A100 setup to gain runtime benefits would exaggerate this cost even more.

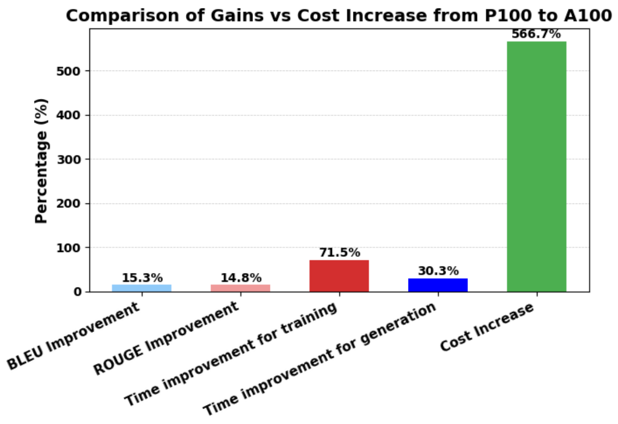

Figure 5: Comparison of gains between the P100 distributed framework and the A100 GPU

This figure shows the improvements in each metric going from a distributed four node setup utilizing P100 hardware when training on the full dataset against the increase in price for each setup. The major gain comes in training time which is expected as the computational speed of the A100 and memory is far greater. The P100 has 12GB memory and has an operational speed of 9.3 TeraFLOPS. Thus, we see that for lower resource utilization and in research fields for graduate students and those without access to high cost GPUs, using PyTorch DDP with cheap hardware can offer runtime gains while showing no drop in performance.

5 Related Work

Previous work was done to incorporate multilingual pretraining on LLM models. Models like XLM-R which improves on the mBERT model [3] showcases fine tuning methods for multilingual use cases. In both these models, there are limitations on handling generative, cross-lingual applicability, and multilingual QA.

This is because mBERT and XLM-R are limited as they are encoder only models which means they are not capable of generation tasks and are thus better suited for classification tasks. Therefore, these models struggle with cross-lingual sequence-to-sequence tasks like translation or question answering which require decoding.

mBART[8] was introduced to combat these problems as it featured an encoder-decoder architecture making it suitable for generation tasks. mBART was also trained on a subset of data which handled 25 languages which enabled it to perform multilingual translation, summarization, and generation. However, this can be scaled up to incorporate more languages [8], and extending support to low resource languages remains an issue. Extending to more languages would require expensive retraining and could come at the expense of performance benefits. XLM-R, for example, showcased performance decreases as the number of languages increased [3].

mT5 was developed as a multilingual pre-trained text-to-text transfer model which took T5's recipe and fine tuned it for the multilingual use case. Using a variant of the C4 dataset, mT5 was trained on the mC4 dataset which comprises 101 languages drawn from the Common Crawl web scrape [14]. This model provided performance benefits in sentence-pair sentence-pair classification, structured prediction and question answering tasks.

However, the model demands heavy pre-training compute and extensive memory requirements to finetune and train. The mt5_XXL model contains 13 billion parameters and occupies 57.1 GB of memory, making it relatively inaccessible for research in resource constrained environments. The mt5_small model is far more lightweight at 300 million parameters and 1.2 GB of memory.

6 Future work

As we have seen, training a lightweight model with PyTorch DDP can boost runtime performance as well as providing equivalent performance to running the same on a single node. Further, combining cheaper hardware together can really leverage the benefits of DDP while providing a cost effective method to run training and generation with MLLMs.

However, the performance gains and cost savings we were able to achieve came largely from the fact that the mt5_small model could fit comfortably into the memory of the P100 GPU. For larger models with far more parameters, this approach may not be as suitable. For example, the mt5_XXL is 57.1 GB which far exceeds the 12 GB memory the P100 provides. Therefore, alternate methods like model parallelism are required to run larger models.

One such approach is given by MegatronLM which slices every layer's weight matrices across multiple GPUs/nodes making use of tensor parallelism. MegatronLM also makes use pipeline parallelism by sequentially splitting the network and compute stages into cohesive blocks across devices. This ensures that MegatronLM maximizes hardware utilization while also minimizing and hiding communication latency. This forms a natural next step for combining DDP runtime gains with models that won't fit on a single node.

Additionally, in the current setup, there is no measure for fault tolerance. In this way, DDP doesn't provide any safeguards against node failure. You would need to manually restart the job. Different methods could be implemented to protect against node failure. One such way could be checkpointing the model and on failure restart the job from the last checkpoint. Another way would be leveraging PyTorch DDP's TorchElastic which uses torchrun to run DDP. TorchElastic has automatic worker restarts as well as checkpoint integration. Implementing this measures could add overhead, so evaluation would need to be done on how much extra resources are required.

7 Code

We make our code publicly accesible through the link provided: `https://github.com/ashishp03/CS744-Final-Project`.

8 Contributions

This was essentially a two-person project. Sree attended the first two meetings, made minor edits to the proposal introduction, and contributed little beyond that.
John Che: 47.5%
Ashish Priyadarshi: 47.5%
Sree Kandasamy: 5%

References

[1] N. Arivazhagan, A. Bapna, O. Firat, D. Lepikhin, M. Johnson, M. Krikun, M. X. Chen, Y. Cao, G. Foster, C. Cherry, W. Macherey, Z. Chen, and Y. Wu. Massively multilingual neural machine translation in the wild: Findings and challenges, 2019.

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[3] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov. Unsupervised cross-lingual representation learning at scale. *CoRR*, abs/1911.02116, 2019.

[4] Google Cloud. Evaluate models — cloud translation api (advanced). `https://cloud.google.com/translate/docs/advanced/automl-evaluate`. Accessed: 2025-05-08.

[5] A. Khandelwal, T. Yun, N. V. Nayak, J. Merullo, S. H. Bach, C. Sun, and E. Pavlick. 100K or 100 Days: Trade-offs when Pre-Training with Academic Resources. *CoRR*, abs/2410.23261, 2024.

[6] H. Kudiabor. Ai's computing gap: academics lack access to powerful chips needed for research. *Nature*, 636:16–17, 2024.

[7] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and S. Chintala. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. In *Proceedings of the VLDB Endowment*, volume 13, pages 3005–3018, 2020.

[8] Y. Liu, J. Gu, N. Goyal, X. Li, S. Edunov, M. Ghazvininejad, M. Lewis, and L. Zettlemoyer. Multilingual denoising pre-training for neural machine translation. *CoRR*, abs/2001.08210, 2020.

[9] R. Maheshwary, V. Yadav, H. Nguyen, K. Mahajan, and S. T. Madhusudhan. M2lingual: Enhancing multilingual, multi-turn instruction alignment in large language models, 2024.

[10] A. Moreno-Cediel, J. A. del Hoyo-Gabaldon, E. Garcia-Lopez, et al. Evaluating the performance of multilingual models in answer extraction and question generation. *Scientific Reports*, 14:15477, 2024.

[11] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2019.

[12] J. Wei, M. Bosma, V. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022.

[13] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush. Transformers: State-of-the-art natural language processing. In Q. Liu and D. Schlangen, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics.

[14] L. Xue, N. Constant, A. Roberts, M. Kale, R. Al-Rfou, A. Siddhant, A. Barua, and C. Raffel. mt5: A massively multilingual pre-trained text-to-text transformer. *CoRR*, abs/2010.11934, 2020.