Pretraining Large Language Models to Fill in the Middle

Hailey Schoelkopf,[1,2] Stella Biderman,[1] and Dragomir Radev[2]

[1]EleutherAI [2]LILY Lab, Department of Computer Science, Yale University, New Haven, CT

Introduction + Methods

Large language models (LLMs) in recent years have exhibited impressive performance that scales well with model data and size on a variety of NLP tasks including NL/code generation. Typical LMs are autoregressive, meaning they only generate text left-to-right. However, one desirable use case is to improve model-human collaboration using models which *edit* text and fill in new tokens at a specific cursor location. This is a useful step toward more collaborative use of LLMs by humans.

A recent technique published by OpenAI is FIM (Fill-In-the-Middle), a simple method to enable language models to perform text infilling. A "FIM-for-free" property is reported, in which this FIM capability is gained without a significant autoregressive perplexity/loss/generation performance hit, but all models in the paper are private so the claim is unverifiable. I have replicated the setup from Bavarian et al. 2022 and released a FIM model, but FIM comes at a cost to autoregressive performance.

Figure 1. The FIM Data Augmentation

$$\texttt{<PRE>} \circ \mathrm{Enc}(\mathrm{prefix}) \circ \texttt{<SUF>} \circ \mathrm{Enc}(\mathrm{suffix}) \circ \texttt{<MID>} \circ \mathrm{Enc}(\mathrm{middle}),$$

Distributed Training for LLMs

Training large language models requires significant engineering for proper performance and hardware utilization, measured in MFU (Model Flop Utilization), in terms of % of peak achievable performance on a single GPU (e.g. 312 TFLOPs/gpu for an A100).

Models are trained across many GPU accelerators with high-speed interconnect and often separated into separate nodes in a cluster.

For efficient large model and large-batch training, 3D parallelism is used: data, pipeline, and tensor (model) parallelism are used to shard model states and batches across devices efficiently.

In this project, I used and developed on top of EleutherAI's GPT-NeoX library, which uses Nvidia Megatron and Microsoft Deepspeed libraries to support 3D parallel training, and achieved up to 50% MFU in training 1.3B parameter models.

Table 1. Main Evaluations

Model	HumanEval AR Pass@1	HumanEval AR Pass@10	HumanEval-infilling Pass@1
GPT-Neo-1.3B	4.79	7.41	---
AR-1.3B	4.15	6.71	0.06
FIM-1.3B-rotary	3.66	6.71	3.90
FIM-1.3B-alibi	2.26	4.88	0.06
TokenFIM-1.3B-rotary	--	--	1.03

Table 1. Final Test PPL

Model	Final Test Loss
AR-1.3B	2.007
FIM-1.3B-rotary	2.096
FIM-1.3B-alibi	---
TokenFIM-1.3B-rotary	2.146

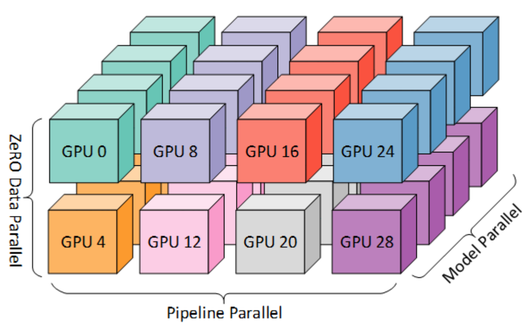

Figure 2. 3D Parallelism Techniques

(https://www.microsoft.com/en-us/research/blog/deepspeed-extreme-scale-model-training-for-everyone/)

Figure 3. A sample HumanEval AR and infilling example + solution.

```
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) ==>12
    solution([3, 3, 3, 3, 3]) ==>9
    solution([30, 13, 24, 321]) ==>0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

```
def unique(l: list):
    """Return sorted unique elements in a list
    >>> unique([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [0, 2, 3, 5, 9, 123]
    """
    return sorted(list(set(l)))
```

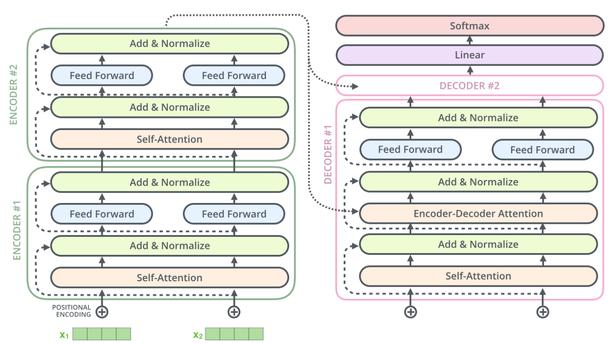

Figure 4. Sample Transformer Architecture Diagram

(https://jalammar.github.io/illustrated-transformer/)

Results

I implemented the FIM method in GPT-NeoX and trained over 5 large language models of 1.3B floating point parameters or larger, controlling for batch size, data, model size, and other hyperparameters.

I evaluated these models against a number of classification and code generation tasks as well as perplexity, and found that the claims in Bavarian et al. 2022 were not replicable by myself or other. This may be due to subtle differences in data preprocessing, and follow-up work will be looking into where the divergence arises, as well as further attempting to perform infilling with autoregressive models and prompting.

Code is available at https://github.com/EleutherAI/gpt-neox/tree/FIM-clean/
and
https://github.com/EleutherAI/lm-evaluation-harness/tree/add_humaneval/ .

Conclusion

In the course of this semester, I learned how to scale large language model pretraining to dozens of nodes and hundreds of GPU accelerators.

I was able to replicate recent state-of-the-art work on LLM pretraining and publicly release said replication at https://huggingface.co/CarperAI/FIM-NeoX-1.3B/tree/main . I was also able to contribute to the open-source LLM ecosystem via code implementations in current popular infrastructure developed by EleutherAI.

I also had the opportunity to share intermediate results and discuss findings with other groups attempting replications who had similar difficulty achieving the paper's claims.

Acknowledgements

I would like to thank both my advisors, Drago and Stella. I'd also like to thank everyone who helped me out on learning infrastructure and engineering including Shivanshu Purohit and Quentin Anthony, and also everyone who shared advice and experience on their own (re)implementations, including Sid Black, Mohammad Bavarian, and others!

Compute for this project was provided by Stability AI.