RESEARCH

SHARE

by Abhinav Venigalla, Linden Li
on September 29, 2022

Mosaic LLMs (Part 2): GPT-3 quality for <$500k

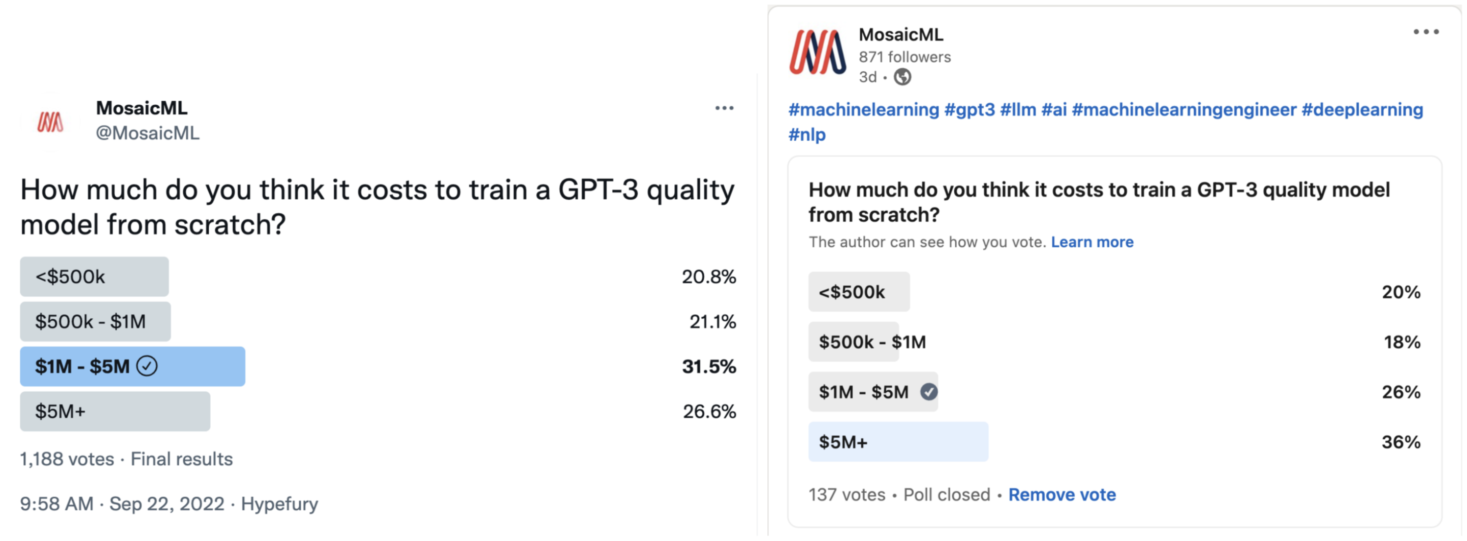
Training large language models (LLMs) costs less than you think. Using the MosaicML platform, we show how fast, cheap, and easy it is to train these models at scale (1B -> 70B parameters). With new training recipes and infrastructure designed for large workloads, we enable you to train LLMs while maintaining total customizability over your model and dataset.

Large language models (LLMs) are exploding in popularity, but the ability to train huge models like GPT-3 from scratch has been limited to a few organizations with vast resources and deep technical expertise. When the rest of us want to use LLMs, we have to rent them from API providers that charge a premium for inference and impose content restrictions on the model's outputs. The models they provide are one-size-fits-all, and are trained on generic, crowd-sourced data that are not necessarily suited for a specific domain. In some cases, this generic data may unintentionally amplify human biases.

At MosaicML, our mission is to make deep learning efficient for everyone. In the context of LLMs, that means driving down costs and speeding up training so that anyone can create a custom LLM that fits their needs (e.g., a LLM for a healthcare organization that is pre-trained on medical journals rather than Reddit). We imagine a world where millions of unique models are trained on diverse datasets for use in specific applications.

Today, we are announcing our next step along this journey: first-class support for training LLMs on the MosaicML platform and transparent data on the costs of doing so. **The bottom line: it costs about $450K to train a model that reaches GPT-3 quality*, which is 2x-10x less than people think.** And this is just the start. Over the coming months, we will be developing MosaicML training recipes that reduce that price tag even more. Even though many of the reference models we share below are now within reach of enterprises, we truly want LLMs to be accessible to *everyone*: any business, researcher, or student should be able to independently analyze and train these models.

** Quality == predicted eval loss using the Chinchilla scaling law; see recipe and details below*



MosaicML
@MosaicML

How much do you think it costs to train a GPT-3 quality model from scratch?

<$500k	20.8%
$500k - $1M	21.1%
$1M - $5M ✓	**31.5%**
$5M+	26.6%

1,188 votes · Final results

9:58 AM · Sep 22, 2022 · Hypefury

MosaicML
871 followers
3d ·

#machinelearning #gpt3 #llm #ai #machinelearningengineer #deeplearning #nlp

How much do you think it costs to train a GPT-3 quality model from scratch?

The author can see how you vote. Learn more

<$500k	20%
$500k - $1M	18%
$1M - $5M ✓	26%
$5M+	36%

137 votes • Poll closed • Remove vote

***Figure 1**: Twitter and LinkedIn users predict how long it would take to train a GPT-3 quality model (collected over 9/22/22 - 9/23/22). Around 60% of users believed the cost was over $1M, and 80% of users believed the cost was over $500k.*

We want to bust that myth. Here, for the first time, are times and costs to train compute-optimal LLMs, all measured on the MosaicML platform. Our profiling shows that a compute-optimal GPT-30B can be trained on 610B tokens for less than $500k. And as we'll explain below, based on the new Chinchilla scaling law, we expect this model recipe to reach the same quality as the original GPT-3, which had more parameters (175 billion params) but was trained on less data (300 billion tokens).

LLM Training Costs on MosaicML Cloud

Model	Billions of Tokens (Compute-optimal)	Days to Train on MosaicML Cloud	Approx. Cost on MosaicML Cloud
GPT-1.3B	26B	0.14	$2,000
GPT-2.7B	54B	0.48	$6,000
GPT-6.7B	134B	2.32	$30,000
GPT-13B	260B	7.43	$100,000
GPT-30B *	**610B**	**35.98**	**$450,000**
GPT-70B **	1400B	176.55	$2,500,000

***Figure 2**: Times and Costs to train GPT models ranging from 1.3B to 70B params. Each model is paired with a compute-optimal token budget based on 'Training Compute-Optimal Language Models'. All training runs were profiled with Composer on a 256xA100-40GB cluster with 1600Gbps RoCE interconnect, using a global batch size of 2048 sequences ~= 4M tokens.*

** This recipe is predicted to reach the same quality as the original GPT3-175B recipe.*
*** This recipe is the same as Chinchilla-70B.*

According to **Figure 2,** the cost to train most of these LLMs is in the range of thousands to hundreds of thousands of dollars, not millions. Around 80% of polled users expected the costs to be higher! For smaller models in the 1.3B -> 6.7B range, training costs are now well within reach of startups and academic labs, and models can be delivered in days, not months.‡

If you want to train your own custom LLMs with MosaicML, sign up for a demo of our platform.

If you want to dive right into the starter code, check it out here.

If you're an ML engineer and wondering, 'How did we even get here?!', read on!

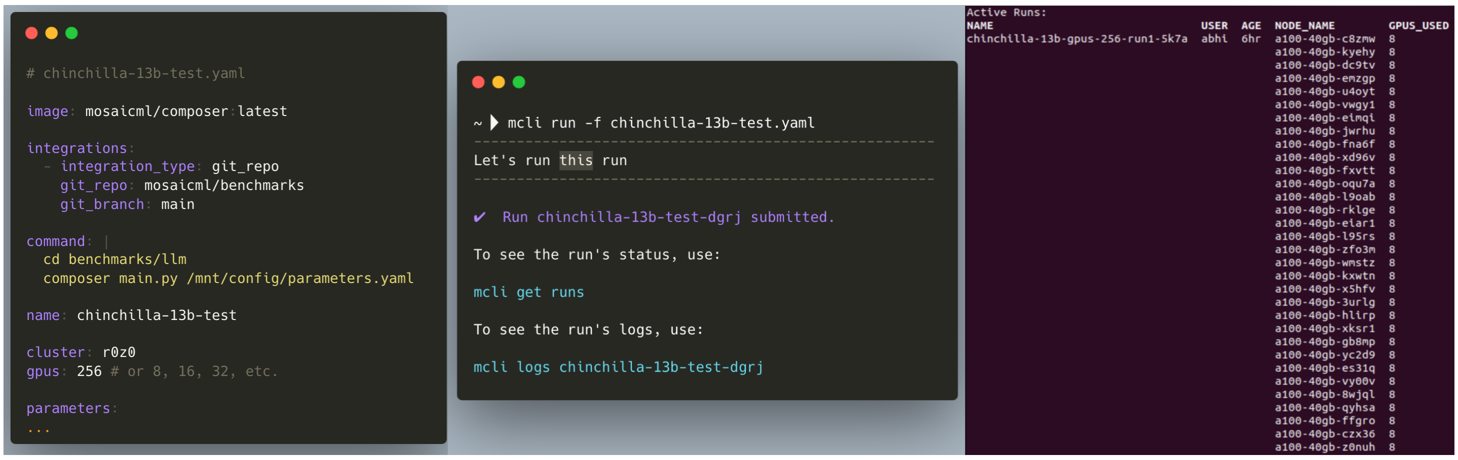
Component #1: MosaicML Platform

When you want to train a custom LLM, one of the biggest challenges is sourcing your compute infrastructure. As we know from open source efforts such as BLOOM and OPT, this is an important decision: orchestrating multi-node jobs on hundreds of GPUs is tricky and can surface errors that don't happen at a smaller scale. Hardware failures and loss spikes are common and require constant monitoring and resumption during training. And even if everything works, it can still be painfully slow to initialize jobs, store/load checkpoints, and resume dataloaders. On top of that, the model may not converge as expected, leading to expensive re-runs and hyperparameter tuning.

The MosaicML platform was built from the ground up to take care of these challenges. Our platform enables you to source compute, orchestrate and monitor training jobs, and share resources with your teammates. Best of all, our research team provides battle-tested training code and recipes that are optimized for LLM workloads. It is cloud infrastructure-agnostic, compatible with data stored in any cloud provider (AWS, OCI, Google, Azure).

Let's dig into some key features:

Multi-Node Scaling



```
# chinchilla-13b-test.yaml

image: mosaicml/composer:latest

integrations:
  - integration_type: git_repo
    git_repo: mosaicml/benchmarks
    git_branch: main

command: |
  cd benchmarks/llm
  composer main.py /mnt/config/parameters.yaml

name: chinchilla-13b-test

cluster: r0z0
gpus: 256 # or 8, 16, 32, etc.

parameters:
  ...
```

```
~ ❯ mcli run -f chinchilla-13b-test.yaml
------------------------------------------------------------
Let's run this run
------------------------------------------------------------

✔  Run chinchilla-13b-test-dgrj submitted.

To see the run's status, use:

mcli get runs

To see the run's logs, use:

mcli logs chinchilla-13b-test-dgrj
```

```
Active Runs:
NAME                             USER  AGE  NODE_NAME          GPUS_USED
chinchilla-13b-gpus-256-run1-5k7a  abhi  6hr  a100-40gb-c8zmw  8
                                                a100-40gb-kyehy  8
                                                a100-40gb-dc9tv  8
                                                a100-40gb-emzgp  8
                                                a100-40gb-u4oyt  8
                                                a100-40gb-vwgy1  8
                                                a100-40gb-eimqi  8
                                                a100-40gb-jwrhu  8
                                                a100-40gb-fna6f  8
                                                a100-40gb-xd96v  8
                                                a100-40gb-fxvtt  8
                                                a100-40gb-oqu7a  8
                                                a100-40gb-l9oab  8
                                                a100-40gb-rklge  8
                                                a100-40gb-eiar1  8
                                                a100-40gb-l95rs  8
                                                a100-40gb-zfo3m  8
                                                a100-40gb-wmstz  8
                                                a100-40gb-kxwtn  8
                                                a100-40gb-x5hfv  8
                                                a100-40gb-3urlg  8
                                                a100-40gb-hlirp  8
                                                a100-40gb-xksr1  8
                                                a100-40gb-gb8mp  8
                                                a100-40gb-yc2d9  8
                                                a100-40gb-es31q  8
                                                a100-40gb-vy00v  8
                                                a100-40gb-8wjql  8
                                                a100-40gb-qyhsa  8
                                                a100-40gb-ffgro  8
                                                a100-40gb-czx36  8
                                                a100-40gb-z0nuh  8
```

***Figure 3**: Running multi-node jobs is as easy as changing the `gpus` field. **Left:** An example of a MosaicML run config. **Center:** Launching a run using the`mcli` (Mosaic Command Line Interface) tool. **Right:** A screenshot from our cluster utilization tool displaying a 256xA100 multi-node job.*

First, the MosaicML platform provides a job scheduler that works on any cluster (yours, ours, or a cloud provider's) and makes **launching and scaling jobs feel like magic.** Accelerating a workload is as easy as changing `gpus: 8` to `gpus: 256`; the MosaicML platform and our Composer library take care of all the distributed training details. In **Figure 3,** you can see what it looks like when we launch a GPT-13B training run from our laptop and monitor our cluster (that's a single run on 256 GPUs!). As we demonstrated in Part 1 of our LLM series, multi-node training with the right infrastructure incurs very little overhead: on our platform, we see near linear scaling when training LLMs on up to hundreds of GPUs. That means you get your work done a lot faster, with minimal increase in total cost.

Fast, Automatic, Graceful Resumption

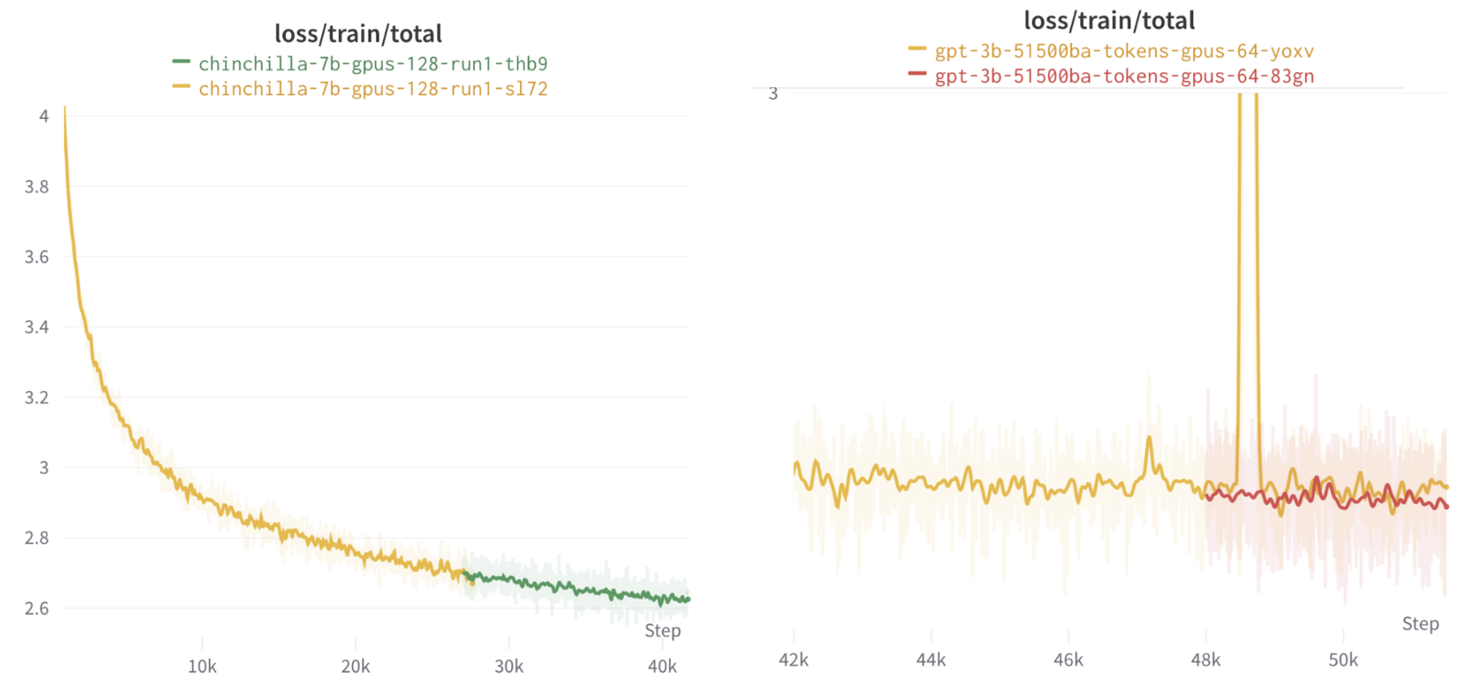


***Figure 4**: The MosaicML platform gracefully resumes runs that fail due to hardware or convergence issues. **Left**: A GPT-6.7B training run on 128xA100 hangs due to a hardware failure, so the job is stopped and launched again, auto-resuming from the last stable checkpoint. **Right**: a GPT-2.7B training run on 64xA100 encounters a loss spike. By resuming from a checkpoint before the loss spike and using a different data shuffle order, the run succeeds without any spikes.*

Next, as we launch and wait for these LLMs to train, we inevitably encounter hardware issues and loss spikes, even when we use bfloat16 mixed-precision training. To address these issues, the MosaicML platform works seamlessly with Composer to support automatic graceful resumption. When runs crash or loss spikes are detected, our platform automatically stops, relaunches, and resumes the run from the latest stable checkpoint. And since Composer natively supports both filesystem and object-store-based checkpointing, you can stop and resume jobs from any cluster and keep your model artifacts in one central location.

In **Figure 4,** you can see how we resumed a GPT-6.7B training run that encountered a 'hang' (likely a failed communication op), and a GPT-2.7B that encountered a loss spike. In both cases, we were able to seamlessly download and resume from the last stable checkpoint, and do so either automatically or by specifying a checkpoint path like `load_path: s3://my-bucket/my-run/ep0-ba48000.pt`.

One last detail: When resuming from a checkpoint deep into a training run, it can take a long time to reload the dataloader state. For example, in Meta's recent OPT-175B training logs, it says that around 30 minutes was spent waiting for the dataloader to "fast-forward" back to its checkpointed state. That's 30 minutes during which all 1024 GPUs were sitting idle, wasting resources and adding frustrating latency. To avoid these problems, we've built our own streaming dataset implementation that features deterministic and near-instant resumption, and works seamlessly with Composer so that no fast-forwarding is needed. It's also built from the ground up to support streaming datasets from object stores like S3, GCS, etc. with no impact on LLM training throughput. Stay tuned for an upcoming blog post with more info on streaming datasets!

Component #2: Composer + PyTorch FSDP

Once you have your compute infrastructure, the next major hurdle for LLMs is fitting and parallelizing these huge models on GPU clusters. Out-of-Memory (OOM) errors and slow training throughput are common. To mitigate these problems, one would usually adopt a framework like Megatron-LM and apply some exotic form of 3D parallelism (pipeline-, model-, and/or tensor-parallelism) to distribute the model across the devices. These frameworks are performant but hard to customize, with strong assumptions about how the model should be structured, and they often use non-standard checkpoint formats and precision strategies. Working with them, let alone modifying them to use custom training recipes, can be an ordeal. (We know. We've tried).

That's why at MosaicML, our team set out to build an LLM stack that was as simple and flexible as possible. We made a set of design choices in Composer that allows us to sidestep many of the above hurdles, and allows users to build models in pure PyTorch and train them with **data-parallelism only** without compromising performance. In particular, we chose to integrate PyTorch FSDP (FullyShardedDataParallel) as our engine for large workloads such as LLMs.

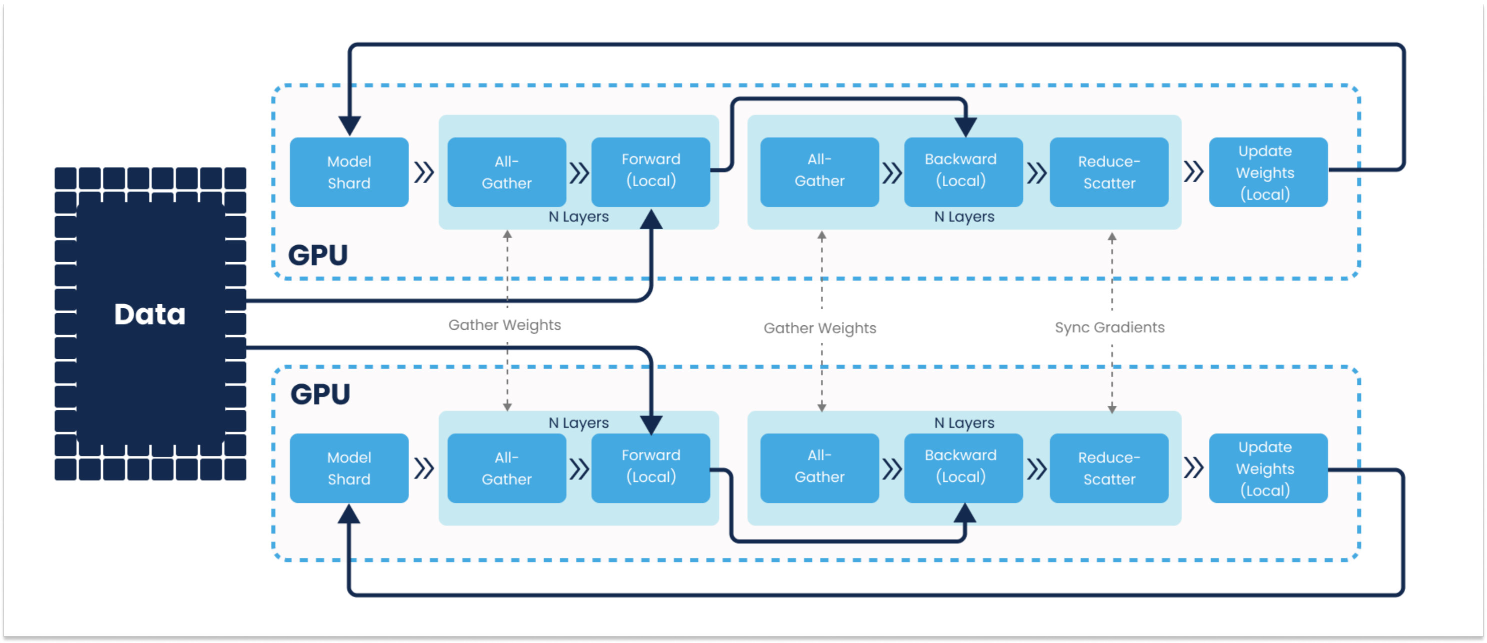


***Figure 5**: A diagram of the PyTorch FullyShardedDataParallel strategy. Model weights, gradients, and optimizer state are sharded across devices, and fetched (all-gather) when needed during the forward and backward pass. Apart from these communication ops, the distribution of work and averaging of gradients across devices is identical to standard distributed data parallelism. Source: Diagram adapted from Meta*

PyTorch FSDP is a strategy similar to PyTorch DDP (DistributedDataParallel) that defines how a model's parameters are sharded and how gradients are calculated across devices (See **Figure 5**). The important detail is that FSDP enables you to train large models like LLMs **without pipeline-, model-, or tensor-parallelism**, which can be difficult to integrate into model code and make it complicated to reason about performance. FSDP instead distributes work using only data-parallelism, the simplest and most familiar strategy for ML users, and doesn't impose any restrictions on the model architecture, optimizer, or algorithms being used.

To emphasize the flexibility of FSDP, all training results in this blog post were measured using a vanilla handwritten GPT model (inspired by Andrej Karpathy's minGPT!) that is essentially pure PyTorch, making it dead simple for users to customize as they wish. For example, we substituted the `torch.nn.MultiheadAttention` layer with the newly released `FlashMHA` module from Dao et. al, and Composer had no trouble training the custom model with FSDP. Check out the code and try customizing it yourself!

How is this possible? FSDP is a PyTorch-native implementation of the ZERO sharding strategies. ZERO shards the model + gradient + optimizer state across the whole cluster rather than replicate it on each GPU. This means that even a large 70B param model with ~ 1 TB of state only has to fit within *the total cluster memory*, not within each GPU. This can be accomplished with a 32xA100-40GB cluster, or a 16xA100-80GB cluster, etc. Enabling CPU offloading with Composer (work in progress!) will unlock even smaller systems.

Composer + FSDP also has some sweet quality-of-life features:

- Checkpoints are saved as a single Pytorch `.pt` file from global device0, instead of a collection of sharded files from N different devices. This makes it easier to resume runs on different cluster sizes and share models for downstream tasks.
- FSDP is built by PyTorch and works seamlessly with standard mixed precision training via `torch.autocast(...)` while also supporting advanced features such as low-precision gradient communication and low-precision master weights + optimizer state.
- FSDP can be used with Composer's auto micro-batching engine to catch and adjust for OOM errors dynamically during training, and enable hardware-agnostic training configs.

Now this is all great, but what about performance? We are happy to report that when using the MosaicML platform, we are able to train these LLMs at equal- or better- utilization to highly optimized frameworks such as Megatron-LM. We regularly see 40%+ MFU (Model FLOPS Utilization) when training with real-world settings such as a GPT-13B model on 256xA100 GPUs. This matches reports from other frameworks while using lower-tier cards (40GB vs. 80GB), using larger device counts (256 GPUs vs. 8-64 GPUs), and providing way more flexibility.

Component #3: Chinchilla Compute-Optimal Training

We've talked about infrastructure and code, but what about the actual model recipe? Should we be training dense 100B+ or even trillion parameter models? We don't think so. In recent years, the research community has made incredible progress in developing scaling laws that govern the quality and compute requirements for LLMs. Recently, the paper Training Compute-Optimal Language Models by Hoffmann et. al (a.k.a. the Chinchilla paper) made the exciting discovery that large models such as GPT3-175B and Gopher-280B could have been trained far more efficiently by **using fewer parameters and more data.** By carefully remeasuring the scaling laws that were used to design these models, the authors updated the function (Eq. 10) for estimating the pre-training loss of a model given N := the parameter count and D := the number of tokens of data:

$$L(N, D) \approx 1.69 + \frac{406.4}{N^{0.34}} + \frac{410.7}{D^{0.27}}$$

This updated scaling law led to a proposal for a model called Chinchilla-70B, that was trained with the same compute budget as Gopher-280B but achieved much better loss and downstream results.

We used the same scaling law and plugged in the original GPT3-175B recipe of (N=175e9, D=300e9) to get a predicted loss value of L = 2.078. If we instead follow the recommendations from the paper by scaling data proportionally with model size (approx. D = 20 * N), and keep hardware-efficient layer sizes in mind, then we arrive at a GPT-30B model and recipe (N=30e9, D=610e9) that is predicted to have a slightly lower loss value of L = 2.071. This new recipe uses 65% less compute than before (See **Figure 6**), leading to a huge reduction in training time and cost!

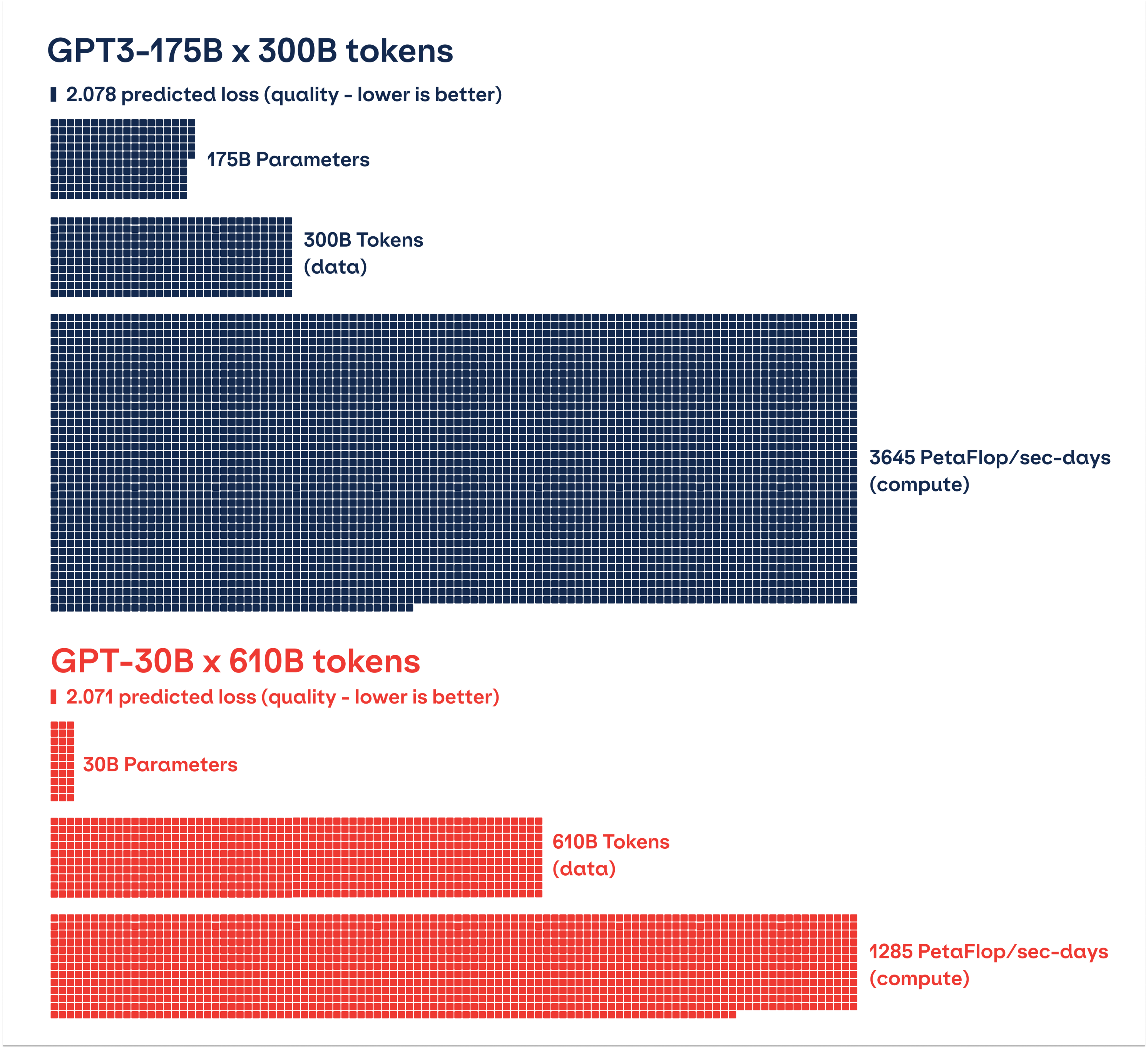


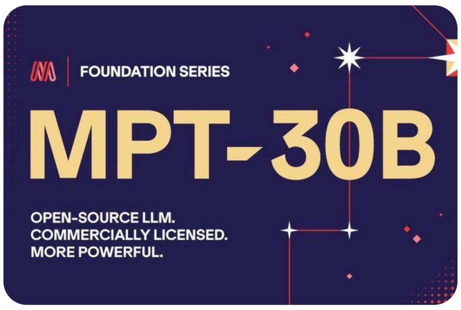
***Figure 6**: We compare the original GPT3-175B recipe with a new compute-optimal GPT-30B recipe. We substitute 175B -> 30B parameters, and 300B -> 610B tokens. Both models have nearly equal predicted loss values based on the Chinchilla scaling law (Eq 10), but the GPT-30B recipe uses 65% less compute as the GPT3-175B recipe. Compute is measured using the standard approximation `FLOPS = 6 * N * D`.*

It's worth noting that **the final quality of GPT-30B is still a projection**, and though we have promising results from smaller runs, we are still training GPT-30B to completion and will have an update soon. But the Chinchilla work and recent results like AlexaLM-20B (which outperforms GPT3-175B and was also trained using these same recommendations) provide overwhelming evidence that better scaling laws dramatically reduce the cost of training GPT-3 quality models. As our training runs continue, stay tuned for a follow-up LLM blog where we will dig deep into training details, results and evaluation!

What's Next?

In this blog post, we showed how fast, cheap, and easy it is to train custom LLMs with the MosaicML platform. We also projected that a GPT-3 quality model could be trained with compute-optimal recipes for a final cost of less than $500k. If these results interest you, stay tuned for upcoming LLM blogs where we will describe improved training recipes by joining our Community Slack or following us on Twitter. If your organization wants to start training LLMs today, sign up for a demo!

†Please note that specific pricing may vary depending on customer commitments.

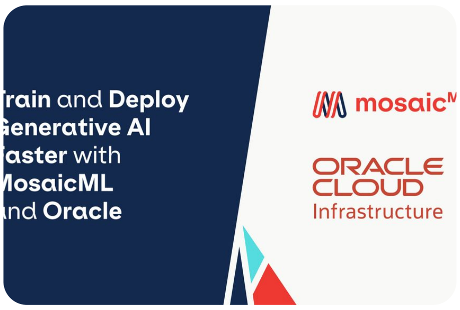


RESEARCH

MPT-30B: Raising the bar for open-source foundation models

Introducing MPT-30B, a new, more powerful member of our Foundation Series of open-source models, trained with an 8k context length on NVIDIA H100 Tensor Core GPUs.

Sep 8, 2023



ECOSYSTEM

Train and Deploy Generative AI Faster with MosaicML and Oracle

Generative AI models have taken the world by storm—but their use in enterprise environments is still limited. Why? This blog post explains the obstacles to adoption and discusses why the MosaicML platform running on Oracle Cloud Infrastructure (OCI) is the best solution for enterprises that want to operationalize generative AI.

Sep 8, 2023

RESEARCH

Training Stable Diffusion from Scratch Costs <$160k

We wanted to know how much time (and money) it would cost to train a Stable Diffusion model from scratch using our Streaming datasets, Composer, and MosaicML platform. Our results: it would take us 79,000 A100-hours in 13 days, for a total training cost of less than $160,000. Our tooling not only reduces time and cost by 2.5x, but it is also extensible and simple to use.

Sep 8, 2023