LLM360: Towards Fully Transparent Open-Source LLMs

Zhengzhong Liu Petuum & MBZUAI
Aurick Qiao Petuum
Willie Neiswanger USC & Petuum
Hongyi Wang CMU
Bowen Tan CMU

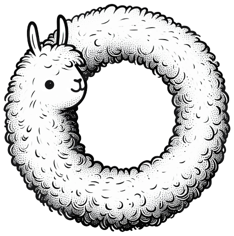
Tianhua Tao UIUC
Junbo Li MBZUAI
Yuqi Wang Petuum
Suqi Sun Petuum
Omkar Pangarkar Petuum
Richard Fan Petuum

Yi Gu UCSD
Victor Miller Petuum
Yonghao Zhuang CMU
Guowei He MBZUAI
Haonan Li MBZUAI
Fajri Koto MBZUAI

Liping Tang MBZUAI
Nikhil Ranjan MBZUAI
Zhiqiang Shen MBZUAI
Xuguang Ren MBZUAI
Roberto Iriondo MBZUAI

Cun Mu MBZUAI
Zhiting Hu UCSD
Mark Schulze Petuum
Preslav Nakov MBZUAI
Timothy Baldwin MBZUAI

Eric P. Xing MBZUAI



Abstract

The recent surge in open-source Large Language Models (LLMs), such as LLaMA, Falcon, and Mistral, provides diverse options for AI practitioners and researchers. However, most LLMs have only released partial artifacts, such as the final model weights or inference code, and technical reports increasingly limit their scope to high-level design choices and surface statistics. These choices hinder progress in the field by degrading transparency into the training of LLMs and forcing teams to rediscover many details in the training process. We present **LLM360**, an initiative to fully open-source LLMs, which advocates for all training code and data, model checkpoints, and intermediate results to be made available to the community. The goal of LLM360 is to support open and collaborative AI research by making the end-to-end LLM training process transparent and reproducible by everyone. As a first step of LLM360, we release two 7B parameter LLMs pre-trained from scratch, Amber and CrystalCoder, including their training code, data, intermediate checkpoints, and analyses (at `llm360.ai`). We are committed to continually pushing the boundaries of LLMs through this open-source effort. More large-scale and stronger models are underway and will be released in the future.

A technical report of the LLM 360 project.

1 Introduction

The landscape of Large Language Models (LLMs) has experienced a remarkable transformation in the past one year, witnessing an unprecedented surge in both the popularity and capabilities of these models. At the forefront of this evolution are proprietary LLMs such as GPT-4 [1] and Claude [2], which have captured the attention of the AI community due to their power and versatility. At the same time, the recent emergence of openly accessible yet highly capable LLMs such as LLaMA [3, 4], Falcon [5], and Mistral [6] allow researchers and practitioners at large to easily obtain, customize, and deploy LLMs in more diverse environments and for more diverse use cases.

Despite the growing influence and accessibility of open-source LLMs, a notable trend has been to restrict visibility and access to their training, fine-tuning, and evaluation processes, including crucial components such as their training code and data. This practice limits the ability of the broader AI research community to study, replicate, and innovate upon advanced LLMs. A more transparent approach to sharing not just the final model but also training details and artifacts is crucial for fostering a more inclusive and collaborative research environment.

Motivated by the above, we note the following specific challenges in LLM research today.

Data Provenance. Understanding the origins and characteristics of the training data is crucial for assessing the reliability and biases inherent in LLMs. A lack of transparency about data sources and composition hinders the ability to identify and mitigate biases which can be perpetuated in model outputs. Simultaneously, data leakage—where training datasets overlap with benchmark datasets—can lead to misleading performance metrics that obscure a model's general effectiveness (studied in [7, 8]). These issues highlight the need for clear documentation of data origins and usage in LLM development.

Reproducibility. Even with full disclosure of data sources, the lack of access to complete training code, configuration details, and specific datasets can make it challenging to reproduce the results reported in studies. For example, although the training data mixtures are disclosed by LLaMA [3], the data processing and training code are not released. Yet, LLMs known to be trained using an open reproduction of LLaMA's data (*e.g.,* RedPajama [9, 10]) still do not fully reproduce its benchmark evaluations [11], indicating that additional data processing or training procedures may be necessary.

Open Collaboration. The practice of only releasing final model weights not only leads to redundant efforts but also poses uniques challenges in conducting certain research. For instance, research into the emergent abilities of LLMs [12, 13] or the investigation of how different training data affects model behavior [14, 15] becomes more challenging without access to intermediate training checkpoints. Researchers are often forced to either work with the final model, which offers limited insights into its developmental nuances, or start from scratch, leading to unnecessary duplication of work and expenditure of compute.

LLM360[1] aims to address the issues above through a comprehensive open-source LLM effort. Models in LLM360 are published with all training and model details (*e.g.,* hyperparameters, schedules, architecture, and designs), all intermediate model checkpoints saved during training, and full disclosure of the exact pre-training data used.

Our contributions are:

- We outline the LLM360 framework, focusing on its design principles and the rationale for fully open-sourcing LLMs. We detail the components of the framework, including datasets, code and configurations, model checkpoints, and training metrics. This framework provides a target for transparency that all present and future LLM360 models strive to meet.
- We pretrain two new LLMs from scratch and release them under the LLM360 framework. AMBER is a 7B English LLM pretrained on 1.3T tokens. CRYSTALCODER is a 7B English and code LLM pretrained on 1.4T tokens. We discuss the development details, preliminary evaluations, observations, and lessons we learned from AMBER and CRYSTALCODER.
- We release all training code, pretraining data, model checkpoints, and evaluation metrics collected during pretraining for both AMBER and CRYSTALCODER. Notably, AMBER is released with 360 model checkpoints saved during training, and CRYSTALCODER with 143.

[1] The name LLM360 signifies open-sourcing LLMs from all angles, and that 360 data points (*i.e.,* checkpoints, data chunks, evaluation results) are released for many of our models.

We aim to make a continuous commitment to fully open-source LLMs by releasing multiple LLMs at various scales. As the first step, in this technical report, we discuss Amber and CrystalCoder, the first open-source LLMs in the LLM360 series. In the future, we plan to release more pre-trained LLMs that are larger in scale, exhibit better performance, and focus on various domains.

The rest of this report is organized as follows. In §2, we discuss related works and the predecessors that inspired LLM360. In §3, we provide a description of the LLM360 framework and the release artifacts that fall into its purview. In §4, we discuss the first two LLMs released under LLM360, Amber (§4.1) and CrystalCoder (§4.1.5), and preliminary analyses of both. §6 concludes.

2 Related Work

The closest project to LLM360 is Pythia, which also aims at full reproducibility of LLMs [16]. The Pythia project provided 154 checkpoints for model sizes from 70M to 12B to better support research on the scaling behavior and learning dynamics of LLMs. While Pythia is a pioneering work, it no longer reflects many recent LLM practices, such as training over trillion-token datasets or training on language and code in different stages. On the other hand, LLM360 defines a release framework prioritizing transparency and reproducibility under which up-to-date models can continue to be released, and our 7B Amber model surpasses the 12B Pythia model in public benchmarks [17]. Overall, Pythia set an early precedent for transparency and reproducibility of LLMs that we aim to perpetuate and expand in LLM360 to modern LLM pretraining regimes.

LLM	**Release**	**Pretraining**		**Checkpoints**		**Pretraining Dataset**			**Tokens**
Name	**Date**	Code	Config	Model	Optim	Data Mix	Ordering	Available	(T)
GPT-J [18]	May’21	✓	✓	✓	✓	✓	✓	✓	0.40
GPT-NeoX [19]	Apr’22	✓	✓	✓	✓	✓	✓	✓	0.40
OPT [20]	May’22	✓	✓	✓		✓			0.18
BLOOM [21]	Nov’22	✓	✓	✓	✓	✓	✓	✓	0.34
Pythia [16]	Feb’23	✓	✓	✓	✓	✓	✓	✓	0.30
LLaMA [3]	Feb’23		✓			✓			1.0
OpenLLaMA [11]	May’23	✓	✓	✓		✓		✓	1.0
INCITE [10]	May’23	✓	✓	✓		✓		✓	1.0
MPT [22]	May’23	✓	✓			✓			1.0
Falcon [23]	May’23		✓			✓			1.5
Llama 2 [4]	Jul’23		✓						2.0
Qwen [24]	Aug’23		✓						2.4
Mistral [6]	Sep’23								?
Yi [25]	Nov’23								?
Amber	Dec’23	✓	✓	✓	*	✓	✓	✓	1.3
CrystalCoder	Dec’23	✓	✓	✓	✓	✓	✓	✓	1.4

Table 1: Summary of notable open-source LLMs. We note a trend of progressively less disclosure of important pretraining details over time: (1) availability of pretraining code, (2) disclosure of training configurations and hyperparameters, (3) intermediate checkpoints of model weights, (4) intermediate checkpoints of optimizer states, (5) disclosure of data mixture and sources, (6) reproducibility of pretraining data sequence, and (7) availability (or reconstruction scripts) of the pretraining data.
* *Amber optimizer states are lost due to errors in early implementation.*

In general, open-source LLMs span a wide spectrum of transparency and reproducibility when it comes to their release artifacts. Many recent LLMs only release their final model architecture and weights, keeping their data sources and most training details undisclosed [4, 24, 6, 25]. Some are trained on publicly available datasets [18, 19, 21, 16, 11, 10, 26], whereas others disclosed their data mixtures but do not make training-ready data available to the public [20, 3, 22, 23]. Several LLMs of note have been released with substantially more transparent details and artifacts. For example, EleutherAI models such as GPT-J [18] and GPT-NeoX [27] included training code, datasets, and up to 150 intermediate model checkpoints. The value of the open-source GPT-NeoX training code was demonstrated by its use in subsequent LLM pretraining by others in the community [10, 22]. INCITE [10], MPT [22], and OpenLLaMA [11] were released with training code and training dataset, with RedPajama also releasing 10 intermediate model checkpoints.

Overall, we observe a trend that more recent and capable LLMs are becoming more closed in their release artifacts. In contrast, the goal of LLM360 is to release modern and high-quality models while maintaining a high degree of release transparency.

3 The LLM360 Framework

In this section we present LLM360, a framework for releasing LLMs that promotes open-source transparency, reproducibility, data/model provenance, and collaborative research. LLM360 provides guidance and recommendations for release artifacts that are collected during LLM pre-training and subsequently made publicly available to the community.

As part of the launch of LLM360, we also release two new pre-trained LLMs, which we hope will foster immediate interest and collaboration in the open-source research community. First, AMBER, an English language LLM with 6.7B parameters trained on 1.25 trillion tokens. Second, CRYSTALCODER, an English and code LLM, also with 6.7B parameters, trained on 1.4 trillion tokens. Details on AMBER and CRYSTALCODER are reported in §4.

Training Dataset and Data Processing Code The pretraining dataset is the main ingredient of an LLM and significantly impacts its capabilities. Thus, it is important for users and adopters to have visibility into pretraining data to assess potential behavior issues and biases. For example, recent concerns about benchmark data leakage into LLM pretraining is much easier to study when pretraining datasets are available for exploration [8, 7].

Furthermore, visible pretraining data improves the extensibility of LLMs in later fine-tuning and domain adaptation. Recent work suggests that training on repeated data disproportionately degrades final model performance [28]. Given the breadth of data modern pretraining is performed on, visibility into the original pretraining data is essential for avoiding repeated data in downstream fine-tuning or continued pretraining on specialized domains.

LLM360 advocates for the public release of the data LLMs are pretrained on. When applicable, details about data filtering, processing, and training order should be released as well. Doing so equips the community with better tools to assess the capabilities and risks of LLMs and to reproduce and build upon existing LLMs for future use cases.

Training Code, Hyperparameters, and Configurations These code and settings have a significant impact on the performance and quality of LLM training, and are not always publicly disclosed. For example, we observed that a carefully balanced hybrid data-model-pipeline (3D) parallelism [29] can outperform the standard FSDP in PyTorch by up to 15% on our Nvidia A100 clusters. Another example we observed is that it is essential to keep the inverse frequency matrix in RoPE positional embedding in FP32 [30], which aligns with the observation in Qwen [24].

In LLM360, we open-source all our LLM pre-training frameworks, hyperparameters, as well as the configurations. These include the entire training source code, training parameters such as learning rates and batch sizes, and system configurations such as parallelism dimensions.

Model Checkpoints It is typical during LLM training to periodically save checkpoints of the model to persistent storage. These checkpoints are not only crucial for recovery from faults during training, but also useful in post-training research such as studying different data and/or hyperparameter schedules, or reproducing infrequently-occurring training faults (*e.g.,* loss spikes, NaN results). Recent research on model quantization and compression heavily relies on analysis of model weights and the dynamics during training [31, 32].

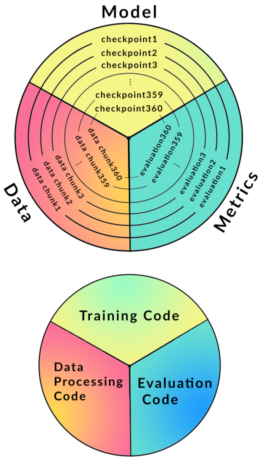


Figure 1: Artifacts released by the LLM360 project include data chunks, model checkpoints, and metrics, at over 360 time stamps of training (and code for all parts).

LLM360 models are published with all intermediate checkpoints saved during their training, including model weights and optimizer states (when applicable, *e.g.,* Adam [33] moving averages). These checkpoints enable continued training from a range of starting points without training from scratch, making it easier to study and reproduce a wider variety of effects during training.

Metrics LLMs undergo training over weeks to months, and the trends and evolution patterns over this training period can offer valuable information. However, access to detailed logs and intermediate metrics for LLMs is currently limited to groups involved in pretraining, hindering a comprehensive study of LLMs. These statistics often contain key insights that cannot be directly derived otherwise, and even a simple analysis on the metrics, such as computing metric variances or norms, can reveal significant findings. For instance, the team behind GLM proposed an effective gradient shrinking algorithm for handling loss spikes and NaN losses by analyzing gradient norm behaviors [34].

Our aim with LLM360 is to alleviate this problem by completely open sourcing the logs and metrics we collect. This includes system statistics (e.g., GPU workload), training logs (e.g., loss, gradient norm), and evaluation metrics (e.g., perplexity, downstream tasks). Access to these logs may facilitate a deeper understanding of the whole training process, including how LLMs evolve during various training scenarios. We provide easy access to the figures by sharing directly on the LLM360 Weights & Biases page[2]. A few example metrics include downstream evaluation results, training loss, gradient norm, etc.

In §4.3, we introduce how one can make use of the metrics, and illustrate an experiment tracking the memorization behavior of a model throughout training. The metrics are released in coordination with the data chunks and checkpoints for researchers to easily find their correspondence. Furthermore, we provide open access to the analysis and evaluation code used to foster reproducibility. The code and all the metrics can be found at an LLM360 repository: Analysis360.

4 Initial Model Release

4.1 Amber

In this section, we introduce AMBER, the first model in the LLM360 family, as well as the finetuned models AMBERCHAT and AMBERSAFE.

4.1.1 Details on Data Preparation and Model Architectures

Below we review the details of our pre-training dataset, including data preprocessing, format, data mixing ratios, along with architectural details of our LLM model and specific pre-training hyperparameters. The exact setup of AMBER can be found in the LLM360 code base.

Figure 2: AMBER is a 7B parameter English open-source LLM.

Details on our pre-training dataset We conduct the data preparation process similar to OpenLLaMA[3]. Specifically, our pretraining data is a mixture of RefinedWeb, StarCoder, and RedPajama-v1. A slight difference with OpenLLaMA-v2 is our inclusion of C4, since we do not intend to introduce dupliciated documents after the deduplication process conducted by RefinedWeb. We simply put together all the original aforementioned datasets (without any further cleaning, filtering, or sub-sampling), conduct a global permutation, and partition them evenly into 360 data chunks. In total, we have 1.26 Trillion tokens. Table 2 presents the combination.

The LLM architecture We used the exact same model architecture as LLaMA 7B[4]. Detailed LLM architectural configurations are summarized in Table 3, incorporating rotary positional embeddings (RoPE) at each layer of the network [30].

Pre-training procedure and hyperparameters We followed the pre-training hyperparameters from LLaMA as closely as possible [3]. AMBER is trained using the AdamW optimizer with the following hyperparameters: $\beta_1 = 0.9, \beta_2 = 0.95$. The initial learning rate is set to $\eta = 3e^{-4}$, following a cosine learning rate schedule that decreases to a final rate of $\eta = 3e^{-5}$. We apply a

[2] `https://wandb.ai/llm360/projects`

[3] `https://github.com/openlm-research/open_llama#dataset-and-training`

[4] The architectural details are directly fetched from `https://huggingface.co/huggyllama/llama-7b`

weight decay of 0.1 and use gradient clipping at 1.0. The model is warmed up over $2,000$ steps. Differing from the LLaMA setup, based on our hardware setting with 224 GPUs, we use a pre-training batch size of $2,240$ (224×10) instead of $2,048$.

Subset	Tokens (Billion)
Arxiv	30.00
Book	28.86
C4	197.67
Refined-Web	665.01
StarCoder	291.92
StackExchange	21.75
Wikipedia	23.90
Total	1259.13

Table 2: Data mix in Amber pre-training.

Hyperparameter	Value
Number Parameters	$6.7B$
Hidden Size	4096
Intermediate Size (in MLPs)	11008
Number of Attention Heads	32
Number of Hidden Layers	32
RMSNorm ϵ	$1e^{-6}$
Max Seq Length	2048
Vocab Size	32000

Table 3: LLM architecture & hyperparameters.

4.1.2 Details on the Pre-training Infrastructure

Amber is trained on an in-house GPU cluster.

The GPU cluster The GPU cluster consists of 56 DGX A100 nodes, each equipped with 4× 80GB A100 GPUs. Each GPU is connected with 4 links NVLink. Cross node connection setting is 2 port 200 Gb/sec (4× HDR) InfiniBand. The throughput we manage to achieve with our distributed training framework is around $582.4k$ tokens per second.

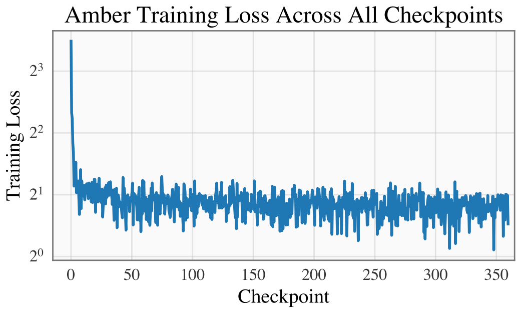


Figure 3: The training loss of Amber over all model checkpoints.

The pretraining framework Our pretraining framework is lit-llama[5] developed based on PyTorch Lightning. We used mixed-precision during pre-training with BF16 for activations and gradients and FP32 for model weights [35].

4.1.3 Finetuned Amber models

We also release a few finetuned versions of Amber, namely AmberChat and AmberSafe. AmberChat is trained on the evolved instruct training data as used by WizardLM [36]. We use FastChat [37] to finetune the model for 3 epochs on 8 A100s (80G) distributed by FSDP [38], the learning rate is 2×10^{-5}, gradient accumulation steps is 16, warmup ratio is 0.04. We also finetune an aligned version of the model: AmberSafe, by conducting Direct Parameter Optimization (DPO) [39]. AmberSafe is trained on ShareGPT 90K[6], and further optimized on the SafeRLHF dataset [40]. We set β to 0.1, gradient accumulation steps to 4, and the learning rate to 5×10^{-7}.

4.1.4 Results and Analysis

Benchmark Results We use four benchmark datasets in the Open LLM Leaderboard[7] as our evaluation on different aspects, *i.e.,* ARC, HellaSwag, MMLU, and TruthfulQA, following the leaderboard settings. We run the evaluation on all 360 checkpoints, to observe the model ability across the pretraining process. As shown in Figure 4, we can see that the HellaSwag and ARC

[5] `https://github.com/Lightning-AI/lit-llama`

[6] The base model for this is checkpoint 355 instead of the last checkpoint

[7] `https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard`

evaluation scores monotonically increase during pre-training, while the TruthfulQA score seems to decrease as the training proceeds. Another interesting trend is observed in the MMLU progress, where the score decreases in the initial stage of pretraining and then starts to increase.

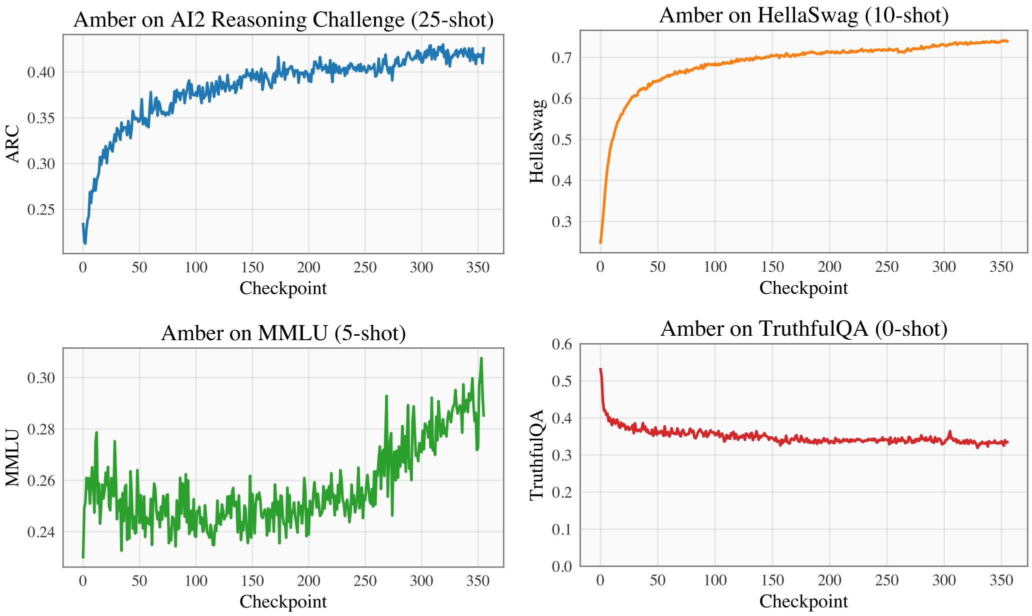


Figure 4: Results for Amber on the Open LLM leaderboard metrics.

In Table 4, we compare the final model performance of Amber to a set of models trained around similar time, namely OpenLLaMA, RedPajama-INCITE, Falcon, MPT. Many are inspired by the design of LLaMA. We found that Amber is relatively competitive in scores such as MMLU, but its performance on ARC is behind the curve. We also find that our finetuned Amber models are relatively strong, even compared with other similar models. In our early study, we note that AmberChat simply trained on ShareGPT 90K also demonstrates much higher performance than our base model, which is slightly different from the trends shown on other models in the table. We leave further investigation of this to future work.

The LLMs	ARC	HellaSwag	MMLU	TruthfulQA	Avg.
LLaMA2-7B-chat	52.9	78.55	48.32	45.57	56.34
LLaMA2-7B	53.07	77.74	43.8	38.98	53.39
AmberSafe	45.22	74.14	37.78	55.44	53.15
LLaMA-7B	50.94	77.8	35.67	34.34	49.69
AmberChat	42.83	74.03	38.88	40.72	49.12
OpenLLaMA-v2-7B	43.69	72.2	41.29	35.54	48.18
MPT	47.7	77.57	30.8	33.44	47.38
Falcon-7B	47.87	78.13	27.79	34.26	47.01
RedPajama-INCITE-7B-Instruct	44.11	72.02	37.61	33.96	46.93
Falcon-7B-instruct	46.16	70.85	25.66	44.07	46.69
OpenLLaMA-v1-7B	47.01	71.98	30.49	34.85	46.08
Amber	41.89	74.14	30.76	34.00	45.20
RedPajama-INCITE-7B-Base	46.25	71.63	27.68	33.03	44.65
RedPajama-INCITE-7B-Chat	42.06	70.82	26.94	36.09	43.98

Table 4: Open LLM leaderboard comparisons for a few LLMs developed around the same time.

4.1.5 Issues Encountered During Pre-training

In this section, we discuss several major issues encountered during the pre-training process of AMBER. These issues could potentially impact our final model performance. We have addressed most of these issues in subsequent LLM pre-training efforts.

NaN loss on a few data chunks During the pre-training procedure, we encountered NaN loss in four out of 360 data chunks. Whenever we faced this issue, we tentatively skipped the entire data chunk. Initially our plan was to train on these four data chunks in later stage of the training, however, we found that these data chunks tend to cause NaN loss regardless of the position of training. We end up finishing our training by taking the first four chunks from the training sequence to complete our learning rate schedule.

Missing optimizer states In our pre-training framework, we did not manage to save the optimizer states; we only saved model checkpoints for each data chunk. This oversight might be the cause of the NaN loss issue observed in the four data chunks, as mentioned earlier. Each time we resumed pre-training from a previous model checkpoint, the optimizer state in the AdamW optimizer was re-initialized. This re-initialization could potentially affect model training stability.

Discrepancies on the precision of checkpoints In the initial phase of pre-training, our codebase had an issue where model checkpoints were saved with BF16 precision, despite our mixed precision training process maintaining model weights at FP32. This issue was later identified and rectified by our team, ensuring that all subsequent model checkpoints were saved with FP32 precision. We anticipate that the initial BF16 model checkpoints may have contributed to some degree of accuracy drop in the model.

4.2 CRYSTALCODER

This section provides a summary of the dataset and the model architecture utilized in CRYSTALCODER. For a detailed evaluation of results on benchmarks and a comparison with previous works on specific benchmarks, we refer readers to our future reports.

Figure 5: CRYSTALCODER is a 7B parameter English and code open-source LLM.

3-Stage Pre-training Dataset The pre-training dataset employed in CRYSTALCODER is a blend of SlimPajama [41] and StarCoder data [42] with around 1382B tokens in total. Diverging from previous approaches such as Code Llama [43], which strictly sequentially trains on English and coding data, we adopt a more gradual approach by seamlessly combining and training on both types of data, to provide a balance between code and general ability. The training process is divided into three stages. In the first stage, we train on half of the SlimPajama data, totaling around 345 billion tokens. Moving to the second stage, the remaining half of the SlimPajama data is utilized, along with two epochs of StarCoder data, resulting in approximately 927 billion tokens. In the third stage, we train on Python and web-related data, encompassing HTML, JavaScript, and CSS subsets from StarCoder, totaling 100 billion tokens. Additionally, we sample 10 billion tokens from the SlimPajama dataset in this stage. The preprocessed data and data mixing scripts are released in the Huggingface and Github repository of LLM360.

Model Architecture CRYSTALCODER employs a model architecture closely resembling LLaMA 7B, with the incorporation of maximal update parameterization (muP) [44]. In addition to this specific parameterization, we have made several slight modifications, the application of RoPE restricted to the first 25% of hidden dimensions (similar to the implementation of GPT-NeoX [27]), and the use of a sequence length of 2048 with an embedding dimension of 32032. In addition, we simply use LayerNorm instead of RMSNorm since the CG-1 architecture supports efficient computation for vanilla LayerNorm.

Compute Infrastructure CRYSTALCODER is trained on the Cerebras Condor Galaxy 1 (CG-1), a 4 exaFLOPS, 54 million core, 64-node cloud AI supercomputer[8].

[8] https://www.cerebras.net/condor-galaxy-1

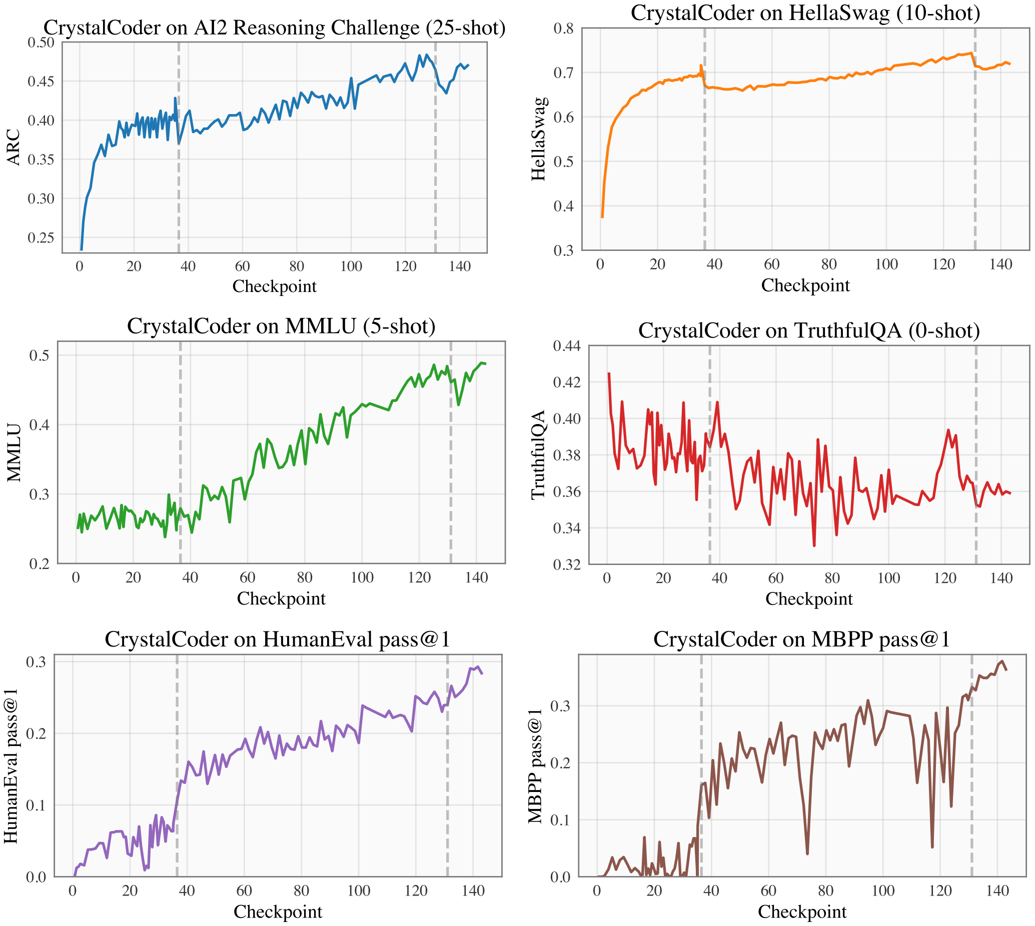


Figure 6: Results for CRYSTALCODER on the Open LLM leaderboard metrics. Grey vertical dashed lines denote the transition between the three stages of training.

Open LLM Leaderboard and Code Evaluations We also benchmark this model on the four benchmark datasets in the Open LLM Leaderboard (similar to AMBER), as well as coding benchmark datasets, including HumanEval pass@1, and MBPP pass@1. We show results in Figure 6.

The LLMs	Language Tasks					Code Tasks			Avg.
	ARC	HellaSwag	MMLU	TruthfulQA	Avg.	HumanEval	MBPP	Avg.	
Mistral-7B	59.98	83.31	64.16	42.15	63.40	29.12	38.78	33.95	48.68
CRYSTALCODER (7B)	47.01	71.97	48.78	35.91	50.92	28.38	36.38	32.38	41.65
CodeLlama-7B	39.93	60.80	31.12	37.82	42.42	33.50	41.40	37.45	39.94
OpenLLaMA-v2-7B	43.69	72.20	41.29	35.54	48.18	15.32	12.69	28.01	38.10
LLaMA2-7B	53.07	77.74	43.80	38.98	53.39	13.05	20.09	16.57	34.98
LLaMA-7B	50.94	77.80	35.67	34.34	49.69	10.61	17.04	13.83	31.76
Falcon-7B	47.87	78.13	27.79	34.26	47.01	9.42	13.39	11.41	29.21
StarCoder-15B	–	–	–	–	–	33.63	43.28	38.46	–

Table 5: Evaluation comparisons among a few notable code and language models. The last column is the average of the language task average and the code task average. CRYSTALCODER strikes a good balance between both language and code tasks.

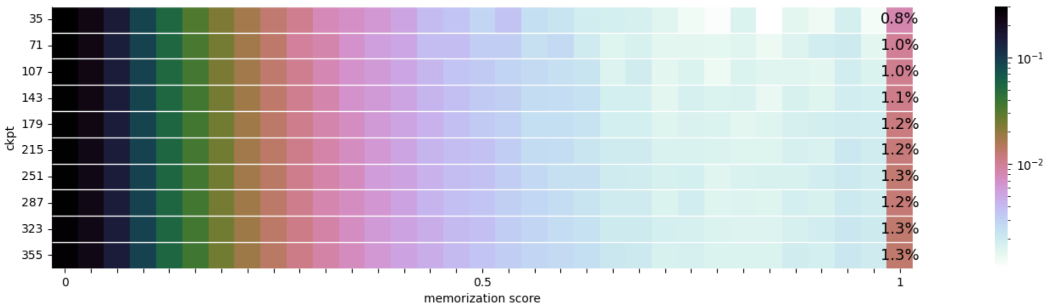


Figure 7: Each row corresponds to the distribution of memorization scores of a checkpoint. We annotate the percentage of $\text{score} = 1$ (k-extractible) for clearer demonstration.

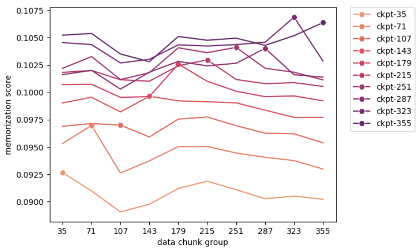
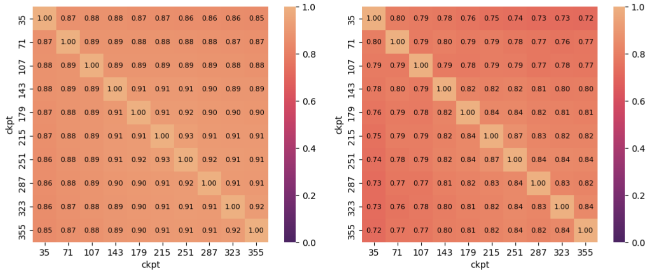


Figure 8: Memorization score on data chunk for each checkpoint. The marked spots indicate the latest chunk seen by that checkpoint. The part on right of each mark indicates unseen data.



(a) Memorization score

(b) k-extractible

Figure 9: The correlation of sequences in terms of memorization score and k-extractible between each checkpoints

4.3 Analysis360

Prior work such as Pythia [16] has shown that an insightful study can be done by analyzing the intermediate checkpoints of a model. We hope LLM360 can also provide the community useful resources for both reference and research purposes. To this end, we release the initial version of the Analysis360 project, an organized repositories that analyze the model behavior on various aspects, including model characteristics and downstream evaluation results.

As an example of the analysis that can be performed over the set of model checkpoints, we conduct an initial study on memorization in LLMs. Recent work [45, 46] shows that LLMs may memorize a significant part of their training data, which can be extracted with appropriate prompting. Such memorization not only raises privacy concerns in leaking private training data, but also downgrades the performance of LLMs if the training data contains unintended duplicates or peculiarities. As we release all checkpoints and data, we can conduct a comprehensive analysis of memorization across the whole stage of training.

We adopt the *memorization score* introduced in [12], indicating the accuracy of tokens in the continuation of length l with a prompt of length k,

$$\text{score}(k, l) = \frac{1}{l} \sum_{i}^{l} \mathbf{1}[S_{k+i} = G_{k+i}],$$

where $S_{0:k+l}$ is the sequence from training data, while $G_{k:k+l}$ is the generated sequence with prompt $S_{0:k}$. A *memorized* or k-extractible [45] sequence has a memorization score of 1. Following [12, 16], we conduct our experiments with $k = l = 32$. We sampled 1000 sequence from each of the 360 data chunks, and use the first 64 tokens of each sequence to conduct the following experiments.

We show the distribution of memorization scores for 10 selected checkpoints in Figure 7, and additionally annotate the percentage of $\text{score} = 1$. For every checkpoint, we only include the data chunks it has already been trained on. From the result, we learn that 1) More than 1% of the sequences are 32-extractible from AMBER; 2) AMBER can memorize more sequences with the training going; 3) The spike at $\text{score} = 1$ indicates that AMBER can memorize a much larger number of tokens than our preset threshold 32 (consistent with prior work [46, 12]).

We group the data chunks according to the selected checkpoints, and plot the memorization score on each data chunk group for each checkpoint in Figure 8. We find that 1) AMBER checkpoints memorize the latest seen data much more than previous data; 2) For each data chunk, the memorization score drops a bit with additional training, but keeps increasing afterwards.

We show the correlation between sequences in terms of memorization score or k-extractible in Figure 9. We witness a strong correlation between the checkpoints.

5 Summary and Take-home Messages

In this section, we summarize the observations and a few take-home messages from our pre-training of AMBER and CRYSTALCODER, our initial modeling efforts in the LLM360 series. We understand that pre-training is a computationally daunting task that many academic labs or small organizations cannot afford to conduct. We hope that LLM360 can provide comprehensive knowledge, allowing users to understand what happens during LLM pre-training (*e.g.,* loss curve behaviors, how the evaluation metrics emerge, etc.) without the need to do so themselves. We also provide some potential use cases showing how researchers and developers can use LLM360 for their own projects.

Take-home Messages Below we list a few of the lessons learned during our initial model training.

- In the pre-training of AMBER, NaN losses were periodically observed, which may have been caused by certain random states, the training precision, or data quality issues. Some solutions include switching to a different random seed or skipping those data chunks. We notice some “misbehaved” data chunks can cause NaN loss regardless of when they are trained. In a preliminary experiment, we move the “misbehaved” data chunks to the end of the training but still observe NaN losses.
- In the pre-training of CRYSTALCODER and our subsequent LLM pre-training efforts, we observed that a hybrid and carefully tuned parallelism strategy—combining data, tensor-model, and pipeline (also referred to as 3D) parallelism strategies [29]—achieves better system throughput than FSDP, especially in distributed clusters with limited intra-node bandwidth.
- Data cleaning (and/or data quality filtering), along with data mixing ratios, are crucial aspects of LLM pre-training, as is the scheduling for various pre-training data categories (*e.g.,* CommonCrawl, Books, StarCoder, etc.). In AMBER pre-training, we attempted to adhere as closely as possible to the hyperparameters used in LLaMA; however, our performance still lags significantly behind LLaMA’s. A key omission in LLaMA’s technical report is a detailed description of their exact pre-training dataset. Our carefully crafted CRYSTALCODER pre-training dataset, which mixes English and coding data, achieves competitive performance with LLaMA on both the Open LLM Leaderboard and Code Evaluation benchmarks. We, along with the entire LLM open-source community, are diligently exploring the best approaches for data cleaning, data quality filtering, and determining optimal data mixing ratios, a pioneering effort exemplified by the DoReMi method [15].

Potential Use Cases of LLM360 We describe a few potential use cases of LLM360 below.

- One can conduct experimental studies at any stage of model training. As previously mentioned, the optimal data mixing ratio remains a significant open problem in LLM pre-training. However, it is nearly impossible to verify a specific mixing ratio by conducting full LLM pre-training. A more feasible approach is to adjust the data mixing ratios on the fly, *i.e.,* starting from an intermediate checkpoint, and either increasing or decreasing a specific data ratio from a particular category, *e.g.,* increasing the data weight in Wikipedia.
- For building domain-specific LLMs (*e.g.,* medical, finance, law, etc.), one may not necessarily want to start from the last pre-trained LLM checkpoint (which would make it more akin to fine-tuning). Instead, one can always pick one of the LLM360 checkpoints (*e.g.,* from 50% of the pre-training stage) and resume the pre-training to obtain a domain-specific LLM.

- A lot of algorithmic approximation frameworks for efficient training require partially trained model weights [47, 48]. LLM 360 provides perfect model initializations for those methods.

LLM360 and Responsible Usage Given the wide-ranging applicability and high performance of LLMs, applications powered by them have the potential to deeply influence various aspects of life. Consequently, it becomes essential for all parties involved in the chain of production of LLMs to carefully manage the potential impact and risks associated with them. All stakeholders need to be informed of these implications and take necessary actions accordingly.

We believe the transparent nature of the LLM360 initiative can help make the potential risks known to stakeholders. As one example, many risks associated with LLMs are related to certain forms of biases [49], such as the risk of social stereotypes, discrimination and exclusion, and the risk of under-representing certain languages or domains. By inspecting the exact training data and bias analysis (e.g. BOLD [50]) in ANALYSIS360, stakeholders can have a thorough review of these risks before deploying the models. LLM360 can also help with risk mitigation. The project shares reproducible traces and exact data during LLM training, providing a reusable environment for researchers to conduct experiments to design better guardrails to contain potential risks.

We understand the importance of controlling the risk of LLMs and we are committed to further developing the LLM360 framework to foster responsible usage of LLMs. We would like invite the community to work with us, by sharing research results or by simply providing feedback.

6 Conclusion and Future Work

In this paper, we introduce LLM360, an initiative for comprehensive and fully open-sourced LLMs. Along with the first release of LLM360, we released two 7B LLMs: AMBER (an English general-purpose LLM) and CRYSTALCODER (an LLM pre-trained specifically for code generation). In terms of artifacts, we released pre-training code, configurations, hyperparameters, intermediate model checkpoints, optimizer states, as well as the data sequence and data processing code. Our vision is to significantly advance and promote transparency within the open-source LLM pre-training community.

For future work, we are conducting a more detailed analysis on AMBER and CRYSTALCODER's base models as well as their fine-tuned models. Detailed results will be released and discussed in their respective technical reports. Our team is also pre-training a much larger LLM, which will be fully released as soon as the pre-training is complete. Additionally, we will explore the optimal ratios for mixing different subsets in the pre-training datasets.

Acknowledgements

We would like to thank Natalia Vassilieva, Joel Hestness, William Marshall, and Bhargav Kanakiya for their contribution to CRYSTALCODER and support on the LLM360 project. We would also like to thank the MBZUAI and Cerebras team for providing and managing the computing infrastructure.

References

[1] OpenAI. Gpt-4 technical report, 2023.

[2] Claude. Claude 2.1 model card. Technical report, Claude Inc., 2023.

[3] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[4] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[5] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The refinedweb dataset for falcon llm: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*, 2023.

[6] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[7] Tianwen Wei, Liang Zhao, Lichang Zhang, Bo Zhu, Lijie Wang, Haihua Yang, Biye Li, Cheng Cheng, Weiwei Lü, Rui Hu, Chenxia Li, Liu Yang, Xilin Luo, Xuejie Wu, Lunan Liu, Wenjun Cheng, Peng Cheng, Jianhao Zhang, Xiaoyu Zhang, Lei Lin, Xiaokun Wang, Yutuan Ma, Chuanhai Dong, Yanqi Sun, Yifu Chen, Yongyi Peng, Xiaojuan Liang, Shuicheng Yan, Han Fang, and Yahui Zhou. Skywork: A more open bilingual foundation model, 2023.

[8] Kun Zhou, Yutao Zhu, Zhipeng Chen, Wentong Chen, Wayne Xin Zhao, Xu Chen, Yankai Lin, Ji-Rong Wen, and Jiawei Han. Don't make your llm an evaluation benchmark cheater, 2023.

[9] Together Computer. Redpajama: an open dataset for training large language models, 2023.

[10] Together Computer. Redpajama-incite-7b-base, 2023.

[11] Xinyang Geng and Hao Liu. Openllama: An open reproduction of llama, May 2023.

[12] Stella Biderman, USVSN Sai Prashanth, Lintang Sutawika, Hailey Schoelkopf, Quentin Anthony, Shivanshu Purohit, and Edward Raf. Emergent and predictable memorization in large language models. *arXiv preprint arXiv:2304.11158*, 2023.

[13] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models, 2022.

[14] Yue Yu, Yuchen Zhuang, Jieyu Zhang, Yu Meng, Alexander Ratner, Ranjay Krishna, Jiaming Shen, and Chao Zhang. Large language model as attributed training data generator: A tale of diversity and bias, 2023.

[15] Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy Liang, Quoc V Le, Tengyu Ma, and Adams Wei Yu. Doremi: Optimizing data mixtures speeds up language model pretraining. *arXiv preprint arXiv:2305.10429*, 2023.

[16] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR, 2023.

[17] Edward Beeching, Clémentine Fourrier, Nathan Habib, Sheon Han, Nathan Lambert, Nazneen Rajani, Omar Sanseviero, Lewis Tunstall, and Thomas Wolf. Open llm leaderboard. `https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard`, 2023.

[18] Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. `https://github.com/kingoflolz/mesh-transformer-jax`, May 2021.

[19] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. Gpt-neox-20b: An open-source autoregressive language model, 2022.

[20] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[21] BigScience Workshop, Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

[22] MosaicML NLP Team. Introducing mpt-7b: A new standard for open-source, commercially usable llms, 2023. Accessed: 2023-05-05.

[23] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, Daniele Mazzotta, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. The falcon series of open language models, 2023.

[24] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[25] 01.ai. 01-ai/yi: A series of large language models trained from scratch by developers @01-ai, 2023.

[26] Zhiqiang Shen, Tianhua Tao, Liqun Ma, Willie Neiswanger, Zhengzhong Liu, Hongyi Wang, Bowen Tan, Joel Hestness, Natalia Vassilieva, Daria Soboleva, and Eric Xing. Slimpajama-dc: Understanding data combinations for llm training, 2023.

[27] Alex Andonian, Quentin Anthony, Stella Biderman, Sid Black, Preetham Gali, Leo Gao, Eric Hallahan, Josh Levy-Kramer, Connor Leahy, Lucas Nestler, Kip Parker, Michael Pieler, Jason Phang, Shivanshu Purohit, Hailey Schoelkopf, Dashiell Stander, Tri Songz, Curt Tigges, Benjamin Thérien, Phil Wang, and Samuel Weinbach. GPT-NeoX: Large Scale Autoregressive Language Modeling in PyTorch, 9 2023.

[28] Danny Hernandez, Tom Brown, Tom Conerly, Nova DasSarma, Dawn Drain, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Tom Henighan, Tristan Hume, Scott Johnston, Ben Mann, Chris Olah, Catherine Olsson, Dario Amodei, Nicholas Joseph, Jared Kaplan, and Sam McCandlish. Scaling laws and interpretability of learning from repeated data, 2022.

[29] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

[30] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

[31] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022.

[32] Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. Llm-qat: Data-free quantization aware training for large language models, 2023.

[33] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[34] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Peng Zhang, Yuxiao Dong, and Jie Tang. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*, 2023.

[35] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[36] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*, 2023.

[37] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023.

[38] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. Pytorch fsdp: Experiences on scaling fully sharded data parallel, 2023.

[39] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2023.

[40] Jiaming Ji, Mickel Liu, Juntao Dai, Xuehai Pan, Chi Zhang, Ce Bian, Chi Zhang, Ruiyang Sun, Yizhou Wang, and Yaodong Yang. Beavertails: Towards improved safety alignment of llm via a human-preference dataset, 2023.

[41] Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. `https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama`, 2023.

[42] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

[43] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[44] Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466*, 2022.

[45] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650, 2021.

[46] Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying memorization across neural language models. *arXiv preprint arXiv:2202.07646*, 2022.

[47] Hongyi Wang, Saurabh Agarwal, and Dimitris Papailiopoulos. Pufferfish: Communication-efficient models at no extra cost. *Proceedings of Machine Learning and Systems*, 3:365–386, 2021.

[48] Hongyi Wang, Saurabh Agarwal, Yoshiki Tanaka, Eric Xing, Dimitris Papailiopoulos, et al. Cuttlefish: Low-rank model training without all the tuning. *Proceedings of Machine Learning and Systems*, 5, 2023.

[49] Laura Weidinger, Jonathan Uesato, Maribeth Rauh, Conor Griffin, Po-Sen Huang, John Mellor, Amelia Glaese, Myra Cheng, Borja Balle, Atoosa Kasirzadeh, Courtney Biles, Sasha Brown, Zac Kenton, Will Hawkins, Tom Stepleton, Abeba Birhane, Lisa Anne Hendricks, Laura Rimell, William Isaac, Julia Haas, Sean Legassick, Geoffrey Irving, and Iason Gabriel. Taxonomy of risks posed by language models. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '22, page 214–229, New York, NY, USA, 2022. Association for Computing Machinery.

[50] Jwala Dhamala, Tony Sun, Varun Kumar, Satyapriya Krishna, Yada Pruksachatkun, Kai-Wei Chang, and Rahul Gupta. Bold: Dataset and metrics for measuring biases in open-ended language generation. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21. ACM, March 2021.