HLAT: High-quality Large Language Model Pre-trained on AWS Trainium

Haozheng Fan*[1], Hao Zhou*[2], Guangtai Huang[1], Parameswaran Raman[1], Xinwei Fu[1], Gaurav Gupta[2], Dhananjay Ram[3], Yida Wang[1], Jun Huan[2]

[1]Amazon Web Services, [2]AWS AI Labs, [3]AGI Foundations, Amazon

{fanhaozh, zhuha, guangtai, prraman, fuxinwe, gauravaz, radhna, wangyida, lukehuan}@amazon.com

ABSTRACT

Getting large language models (LLMs) to perform well on the downstream tasks requires pre-training over trillions of tokens. This typically demands a large number of powerful computational devices in addition to a stable distributed training framework to accelerate the training. The growing number of applications leveraging AI/ML had led to a scarcity of the expensive conventional accelerators (such as GPUs), which begs the need for the alternative specialized-accelerators that are *scalable and cost-efficient*. AWS Trainium is the second-generation machine learning accelerator that has been purposely built for training large deep learning models. Its corresponding instance, Amazon EC2 *trn1*, is an alternative to GPU instances for LLM training. However, training LLMs with billions of parameters on *trn1* is challenging due to its relatively nascent software ecosystem. In this paper, we showcase HLAT: a 7 billion parameter decoder-only LLM pre-trained using *trn1* instances over 1.8 trillion tokens. The performance of HLAT is benchmarked against popular open source baseline models including LLaMA and OpenLLaMA, which have been trained on NVIDIA GPUs and Google TPUs, respectively. On various evaluation tasks, we show that HLAT achieves model quality on par with the baselines. We also share the best practice of using the Neuron Distributed Training Library (NDTL), a customized distributed training library for AWS Trainium to achieve efficient training. Our work demonstrates that AWS Trainium powered by the NDTL is able to successfully pre-train state-of-the-art LLM models with high performance and cost-effectiveness.

KEYWORDS

LLMs, pre-training, AWS Trainium, distributed training

1 INTRODUCTION

Large language models (LLMs), based on transformer architecture [53] and trained on massive text data, is the most recent breakthrough in artificial intelligence. They not only show remarkable capabilities in understanding and generating text [31], but offer immense potential across diverse downstream tasks, such as machine translation [24], information retrieval [59], code generation [44] and so on [57].

Pre-training is the crucial first step in building LLMs because it lays the foundation for their impressive capabilities. It initializes the model with random weights, and trains the model to convergence using tokens from a large text corpus. The training process is designed to be self-supervised. For decoder-only model, such as GPT [7] and LLaMA [51], the model is trained to predict the next token given a sequence of previous tokens. Eventually, the model learns everything ranging from syntax and semantics to world knowledge and commonsense reasoning with a large amount of training data. Pre-training provides the raw material - the language skills and understanding, which facilitate the subsequent fine tuning from different downstream tasks.

Since pre-training requires a large amount of training data (trillions of tokens), it demands highly on computational resources. Advanced AI accelerators, such as AWS Trainium [1], Google TPU [2], and NVIDIA A100/H100 GPUs [3], have been specifically designed for this kind of workloads. These AI accelerators are often integrated with dedicated tensor processing units which offer fast matrix operations and high training throughput. They also have much larger on-chip memory (tens of GBs per accelerator) and high communication bandwidth (hundreds of Gbps) between accelerators across different machines, which allows pre-training of larger models with efficient hardware utilization.

Even with the powerful AI accelerators, due to the sheer size and complexity of LLMs, it's impractical to train them on a single device. In other words, a single accelerator simply doesn't have enough memory or processing power to handle the massive datasets, model parameters, and intricate calculations involved in LLM training. Practitioners rely on distributed training libraries [57] to orchestrate a number of accelerators to conduct the training together. Distributed libraries can shard the model parameters and optimizer states across multiple accelerators with different kinds of parallelism strategy. They also spread the workload across multiple machines, effectively tapping into a combined pool of resources, allowing to train the models at the scale of multi-billions of parameters. Additionally, by splitting the workload, distributed libraries significantly reduce the training time.

Although there have been many successful demonstrations of pre-training LLMs on conventional accelerators (GPUs and TPUs) using state-of-the-art distributed training libraries [16, 43, 47, 58], training LLMs with billions of parameters on AWS Trainium is still challenging. First, Trainium uses a relatively nascent software ecosystem ranging from runtime, compiler, to distributed training library. The training script developed for other accelerators needs to be adjusted to comply with the low-level APIs and operators supported by Trainium. Second, the optimal training configurations that ensure stable convergence and optimal training throughput may also differ from other accelerators, e.g., level of precision, dimensions of 3D parallelism, compiler flags to mention

*Both authors contributed equally to this research.

[1]https://aws.amazon.com/machine-learning/trainium

[2]https://cloud.google.com/tpu

[3]https://www.nvidia.com/en-us/data-center/a100;https://www.nvidia.com/en-us/data-center/h100

a few. On the other hand, Amazon EC2 *trn1* instance, equipped with AWS Trainium accelerators, provides the comparable computation power to Amazon EC2 *p4d* instance, equipped with Nvidia A100 40GB GPUs, but comes with only 60% of the price (details in Section 2). This makes it appealing to fully utilize the compute power of AWS Trainium for LLM pre-training.

In this paper, we for the first time provide an end-to-end LLM pre-training practice on top of AWS Trainium, demonstrating the effectiveness and efficiency of this accelerator. Specifically, we make the following contributions:

- We pre-train HLAT (**H**igh-quality **L**LM pre-trained on **A**WS **T**rainium), a 7B model following the architecture described in [51, 52] from scratch using a total training budget of 1.8 trillion tokens. The pre-training is performed up to 64 Amazon EC2 *trn1.32xlarge* instances with totalling up to 1024 AWS Trainium accelerators.
- We evaluate the pre-trained models on various evaluation tasks including commonsense reasoning, world knowledge, math, coding, etc. The results show that our pre-trained model provides comparable performance with 7B models trained on other AI accelerators and distributed training frameworks, including LLaMA-1, LLaMA-2, OpenLLaMA-1, and OpenLLaMA-2. We also evaluate the intermediate checkpoints during the training process for additional insights.
- We provide some best practices of pre-training process, e.g., different sharding strategies, training precisions, and fault tolerance mechanism, on AWS Trainium and Neuron Distributed Training Library (NDTL) [4]. In addition, we design a novel dataloader for online sample packing which performs both tokenization and example packing during training. For pretraining on large datasets, our dataloader saves significant developer time and compute resources.

2 BACKGROUND - DISTRIBUTED TRAINING ON TRAINIUM

AWS Trainium is the second-generation machine learning accelerator that AWS purposely built for deep learning training. Each Trainium accelerator includes two NeuronCores. Each NeuronCore has 16 GB of high-bandwidth memory, and delivers up to 95 TFLOPS of FP16/BF16 compute power. In this study, we trained our model on Amazon EC2 *trn1.32xlarge* instances: each instance is equipped with 16 Trainium accelerators, and supports 800 Gbps intra-instance network bandwith through NeuronLink. The aggregating compute power of Amazon EC2 *trn1.32xlarge* is 3040 TFLOPS in FP16/BF16, slightly higher to its GPU instance counterpart Amazon EC2 *p4d.24xlarge* at 2496 TFLOPS, but at a much lower price (on demand hourly rate: *trn1.32xlarge* $21.50 vs. *p4d.24xlarge* $32.77).

AWS Neuron is a software development kit (SDK)[5] with a compiler, runtime, and profiling tools that unlocks high-performance and cost-effective deep learning acceleration on AWS Trainium. Neuron is natively integrated with PyTorch [58] and TensorFlow [1], and offers features such as FP32 autocasting, stochastic rounding, collective communication, custom operators, and so on.

[4]https://awsdocs-neuron.readthedocs-hosted.com/en/latest/libraries/neuronx-distributed/index.html
[5]https://github.com/aws-neuron/aws-neuron-sdk

Neuron Distributed Training Library (NDTL), as part of Neuron SDK, is developed to support high-efficiency distributed training on Trainium:

- NDTL supports a variety of existing distributed training techniques, such as 3D parallelism [47] , i.e., Tensor Parallelism (TP), Pipeline Parallelism (PP) and Data Parallelism (DP). To reduce the activation memory during training, activation checkpointing [10] and sequence parallelism [28] are naturally supported with the 3D parallelism. NDTL also supports Zero Redundancy Optimizer Stage 1 (ZeRO-1) [42] to shard optimizer states. 3D parallelism and ZeRO-1 can be applied simultaneously during training.
- NDTL provides unified interfaces to port users' models and training scripts on Trainium accelerators. The NDTL interfaces are friendly to Huggingface `transformers` library [54]. To train models from Huggingface `transformers` on Trainium accelerators with NDTL, it only requires simple code changes in certain layers of the model.
- NDTL supports TP with mixed degrees, i.e., users can use more than one TP degrees to shard different model parameters. TP with mixed degrees is used to handle the case when the LLM has some model parts are not compatible with a unified large TP degree, e.g., when the Grouped Query Attention (GQA) [26] is applied, which prevents for using a unified TP degree to the whole model.
- NDTL supports automatic fault recovery and checkpointing. Checkpointing is optimized to save/load on different machines quickly at the same time, and even able to save asynchronously. In case of hardware failures or communication timeouts, NDTL can automatically restart training from latest saved checkpoints without manual intervention, which is critical for maintaining system uptime and stability.

3 METHOD

3.1 Model Architecture

HLAT adopts the decoder-only transformer architecture and applies same modifications used in LLaMA 7B [51, 52], including pre-normalization with RMSNorm, SwiGLU activation function, and Rotary Embeddings. The model is trained with a maximum sequence length of 4096.

3.2 Training Hyperparameters

We adopt same training hyperparameters as LLaMA 7B [51, 52] models. Specifically, we use a cosine learning rate scheduler with maximum learning rate of $3e^{-4}$ and minimum learning rate of $3e^{-5}$. We use a linear warmup of 2000 steps. The overall learning rate scheduler is plotted in Figure 1d. We use AdamW optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.95$. We use weight decay value of 0.1 for all parameters, including normalization weights. Gradient-norm clipping of 1.0 is applied for training stability.

We use 64 nodes with tensor parallel degree of 8, pipeline parallel degree of 1, and data parallel degree of 256. We use global batch size of 1024 sequences with maximum sequence length of 4096 tokens, so each step covers about 4 million tokens. We train for a total of 1.8 trillion tokens.

3.3 Training Dataset and Dataloader

Our pre-training dataset includes a mix of data from various publicly available sources. It does not contain any data from Amazon or AWS customers, products, or services. We designed a novel dataloader for online example packing to process our datasets for training. It performs both tokenization and packing online during training. In contrast to the offline packing approach, this saves a lot of developer time and compute resources specially for large datasets, and multiple experiments with different datasets and tokenizers. Note that the online tokenization has no impact on training throughput as the tokenization for future samples/batch happens during forward-backward pass of current samples/batch - we use CPU for tokenization and Trainiumdevice for training, so the computations are in parallel. The online sample packing dataloader requires one or more dataset files in Apache Arrow format [4]. At first, all samples in the dataset are shuffled randomly and split into several subsets according to the total data parallel ranks. Each data split is treated as an independent data stream. For training efficiency, we use sample concatenation i.e. if a sample is shorter than the maximum sequence length of the model, we concatenate it with the following sample(s) to curate a sequence with total length equal or more than maximum sequence length. Any left over tokens from current sample is used in the following sample of the batch. The samples within a sequence is concatenated with a special end of sentence token. This gives the model necessary information to infer that the text separated by end of sentence token are unrelated [7]. Note that the concatenated samples may be from very different sources or can be of different formats (e.g. natural language and codes). Finally, each batch of samples are tokenized on the fly. This online dataloader also has the functionality to save the internal states to checkpoint which enables us to resume the states from loaded checkpoint and continue training, in case of unexpected failures.

3.4 Orchestration

For model training, we utilize a cluster with 64 *trn1.32xlarge* instances (nodes) with totalling to 1024 AWS Trainium accelerators. Accelerators within same instance are connected with NeuronLink. The nodes within the cluster are interconnected through Elastic Fabric Adapter (EFA)[6]. EFA is a network interface with uniquely designed operating system that bypasses traditional hardware interfaces, significantly enhancing performance for inter-node communications, a critical factor for collectives operations in distributed training.

We manage the cluster orchestration using Amazon EKS[7], a fully-managed Kubernetes service. Amazon EKS simplifies the deployment of Kubernetes both on AWS and on-premises environments. It autonomously manages the availability and scalability of Kubernetes control plane nodes, which are essential for tasks such as container scheduling, application availability management, cluster data storage, and other fundamental operations.

[6]https://aws.amazon.com/hpc/efa/
[7]https://aws.amazon.com/eks/

3.5 Training Efficiency

LLaMA [51] model uses the efficient implementation features for pre-training on GPUs, that include `xformer` library, activation checkpointing, model parallelism, and computation/communication overlapping, etc. Similar features are also supported by Trainium and Neuron SDK, as well as some unique enhancement such as BF16 with stochastic rounding. Below, we list the key features and configurations used in our model pretraining to improve the efficiency.

Model Parallelism: We adopt tensor parallelism (TP) to shard the model into 8 TP degrees, and sequence parallelism (SP) to shard the 4096 sequence length into 8 SP degrees. This sharding configuration is observed with the highest throughput.

Selective Activation Checkpointing: We use selective activation checkpointing [28] to improve the training efficiency. It has slightly higher memory cost as full activation checkpointing, but increases the overall training throughput.

BF16 with Stochastic Rounding: Pre-training with full precision (FP32) is inefficient for large LLMs, but generic half-precision training (BF16 or FP16) often has numerical stability issues [36]. AWS Trainium features BF16 with stochastic rounding (SR) [20] which is used in our training. Stochastic rounding, which theoretically provides an unbiased estimate of the input, prevents the computation precision-loss in BF16 by performing the rounding operations in a probabilistic manner. The chances of rounding-up/down are determined according to the relative distance from the two nearest representable values. This allows for small increments to accumulate over time, even when added to numbers of significantly higher magnitude, which leads to preciser results in gradient update. Empirically, we found that BF16 with SR shows the same convergence behavior as mixed precision training [36] for HLAT, with higher training throughput and lower memory footprint.

Coalescing Layers with Same Inputs: We coalesced linear layers with the same inputs to reduce the communication in tensor and sequence parallelism, and increase efficiency of matrix operations. Specifically, the `Q,K,V` layers in an attention block are coalesced, and the two linear projections layers in SwiGLU [46] are also coalesced.

Constant Attention Mask: As a decoder-only model, HLAT pre-training uses a constant attention mask (lower-triangular) matrix. Instead of passing attention mask as an input tensor in model training, AWS Trainium supports creating attention masks on Neuron Cores directly before use. This saves host memory usage, avoids redundant computation, and increases training throughput. To enable this feature in training script, the attention mask tensor is directly defined in attention block using `torch.triu` function and mapped to `device='xla'`. The Neuron compiler will therefore enable constant attention mask optimization during compilation.

Compiler Optimization: we use compiling flag `--distribution-strategy=llm-training` to enable the compiler to perform optimizations applicable to LLM training runs that shard parameters, gradients, and optimizer states across data-parallel workers. We also use `--model-type=transformer` that performs optimizations specific to transformer models. We set Neuron environment variable `NEURON_FUSE_SOFTMAX=1` to enable compiler optimizations on custom lowering forSoftmax operation. Finally, we used `NEURON_RT_ASYNC_EXEC_MAX_INFLIGHT_REQUESTS=3` to

reduce training latency with asynchronous execution. This overlaps some executions of accelerators and host (CPU).

4 TRAINING PROCESS

4.1 Training Curves

During the training process, we monitor the training loss, as well as l_2 norm of gradients and l_2 norm of parameters for debugging training stability. Figure 1a shows the training loss over global batches, reduced over all data parallel ranks. The training loss decreases fast for the initial ~250B tokens, and enters a log-linear decrease afterwards. Similar trends are observed in other LLM training [18, 51, 52].

In Figure 1b, we show the gradient l_2 norm during the training. Overall, we see that the gradient norm is stable across the training journey without divergence. Note that gradient spikes are imminent in LLM pre-training when using layer-normalization, or even RMSNorm [50], and some times due to overflow in low-precision, such as 16-bit floats. We show an assuring trend in Figure 1b even with using 16-bit floats such as BF16. Furthermore, we clip the gradient-norm to 1.0 (see Section 3.2) which also allows for magnitude stabilization. Note that sustained spikes in the gradient norm leads to training divergence due to improper weight updates, even after gradient normalization through clipping. In Figure 2, we show that the gradient spikes often last for a single step, and did not lead to training divergence. Specifically, we first track a running average (r) of gradient norm over a window of 20 steps to smooth out the natural fluctuations due to batching. We define occurrence of a gradient spike when the current gradient norm is higher than $r + 0.1$. Next, we track the number of steps for gradient norm returning to less than $r + 0.1$. Over 86%, the spike deviates from running average for only a single step.

Finally, we show the parameter l_2 norm in Figure 1c. During first ~250B tokens, the parameter norm increases consistently. This phase also coincides with the fast decreasing phase of training loss where model parameters converge from random initialization to a structured distribution. After that, the parameter norm consistently decrease since AdamW applies weight decay for regularization [35].

4.2 Hardware and System Failures

The pre-training process can be interrupted due to hardware failures, communication timeouts, etc [6]. We monitor the node active time and frequency of training interruptions. As listed in Table 1, the overall system uptime of HLAT pre-training is 98.81%. With automatic fault recovery mechanism in NDTL (see Section 2), the training quickly recovers from failures automatically without wasting much time. For comparison, we also listed another experimental training run (over 600 billion tokens) without automatic fault recovery, and we observed an average of 20% lower system up time.

Table 1: Percentage of system uptime with vs without automatic fault recovery.

With Fault Recovery	**Without** Fault Recovery
98.81%	77.83%

Table 2: Comparison of model architectures and number of training tokens.

Model Name	Size	Sequence length	Tokens
HLAT	7B	4096	1.8T
LLaMA-1	7B	2048	1T
LLaMA-2	7B	4096	2T
OpenLLaMA-1	7B	2048	1T
OpenLLaMA-2	7B	2048	1T

4.3 Training Instability

We describe a few changes we made before and during the training process for convergence and training stability.

Initialization: We use a scaled initialization strategy for initializing model parameters. Specifically, the initial standard deviation of output layers in attention blocks and MLP layers are scaled by $1/\sqrt{2l}$ where l is the layer index. Similar as discussed in [50], we found better numerical stability and convergence with smaller initial variance on deeper layers. In addition, all parameters are initialized on CPU and offloaded to Trainium.

Normalization: We used tensor parallelism to shard the model parameter matrices except normalization layers. The normalization layer weights, however, are slightly different across TP ranks due to stochastic rounding. Empirically, we found the differences are small, and RMSNorm weights values are all close to 1. OLMo [19] used non-parametric layer norm, which is equivalent as all weights equals 1. Both HLAT and OLMo show similar quality as LLaMA, despite of differences in normalization weights.

Gradient Synchronization: Since Neuron SDK uses a different underlying collective library as other accelerators. It needs careful attention while writing distributed training library, such as NDTL. Mis-referencing APIs may cause training instability.

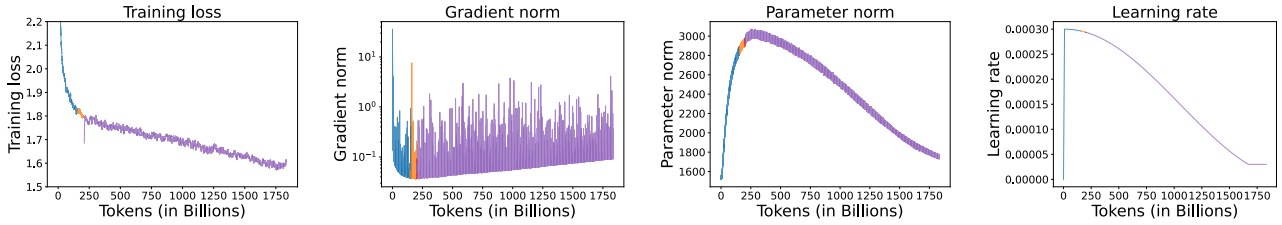
Neuron Persistent Cache on Local Worker: In HLAT training, all instances share a same file system using Amazon FSx[8] for storing data, checkpoints, logs, etc. However, we found that storing Neuron Persistent Cache[9] on FSx may cause communication bottleneck because those cached graphs are frequently accessed by all Trainium devices in the cluster. Such bottleneck may lead to communication timeout and affects training stability. Therefore, we instead store Neuron Persistent Caches in file system of each local worker.

5 EVALUATION

Baselines: We evaluate HLAT against several open-source benchmark models. Since HLAT structure is similar as LLaMA model, we include LLaMA-1 7B [51], LLaMA-2 7B [52], OpenLLaMA-1 7B and OpenLLaMA-2 7B [18]. The model architecture and composition of the training data of the models being compared are listed in Table 2. OpenLLaMA-1 model is trained on RedPajama [14] dataset. OpenLLaMA-2 model shares same structure as OpenLLaMA-1, but is trained on a different data mixture which includes data from Falcon-RefinedWeb [40], StarCoder [32], and RedPajama [14].

[8] https://aws.amazon.com/fsx/

[9] https://awsdocs-neuron.readthedocs-hosted.com/en/latest/general/arch/neuron-features/neuron-caching.html



(a) (b) (c) (d)

Figure 1: HLAT training progress. (a) The training loss vs number of tokens (in billions) seen by the model during training. Gradient/Parameter norm vs number of tokens in (b)/(c), respectively. (d) The learning rate schedule vs number of tokens. The warm-up steps are 2000 iterations (about 8 billion tokens, see Section 3.2).

Table 3: Evaluation of HLAT against 4 open-source models on 7 groups of tasks describted in Section 5. For HLAT, the results are reported using the final (1.8T token) checkpoint.

Task	Shots	Metric	OpenLLaMA-1	OpenLLaMA-2	LLaMA-1	LLaMA-2	HLAT
MMLU	5	accuracy	30.552 (3.432)	41.075 (3.611)	35.1	45.3	41.318 (3.602)
BBH	3	multiple choice grade	35.535 (1.864)	35.502 (1.861)	30.3	32.6	36.565 (1.845)
Commonsense Reasoning	0	accuracy	55.587 (1.203)	56.893 (1.195)	-	-	56.152 (1.194)
	0	accuracy (norm)	58.411 (1.201)	61.262 (1.19)	67.3*	67.5*	59.455 (1.206)
World Knowledge	5	exact match	38.942 (0.532)	37.023 (0.52)	46.2*	48.9*	38.846 (0.534)
Reading Comprehension	0	accuracy	70.459 (0.798)	72.416 (0.782)	76.5	77.4	72.508 (0.781)
Math	8	accuracy	5.08 (0.605)	5.231 (0.613)	11.0	14.6	9.401 (0.804)
Code	0	pass@1	4.77	9.06	10.5	12.8	7.62
	0	pass@10	12.83	23.58	-	-	19.83
	0	pass@100	23.78	40.24	36.5	45.6	34.15

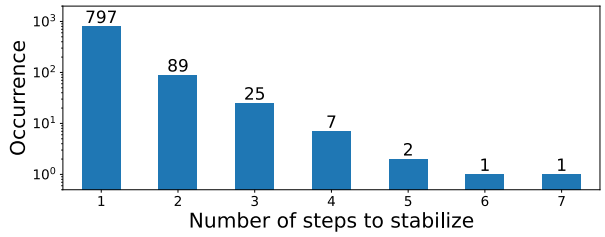


Figure 2: Number of occurrence of sustained gradient spikes vs contiguous length of appearance. Over 86%, the spike lasts for only a single step.

Evaluation Tasks: We evaluate the pre-trained model on 7 groups of tasks including both zero-shot and few-shot evaluations [8]. We use Language Model Evaluation Harness [17] for natural language tasks, and use HumanEval [9] for coding tasks.

- **MMLU (Massive Multitask Language Understanding)** [22, 23] contains 57 test tasks, spanning STEM, humanities, social sciences, and other subjects. The difficulty ranges from elementary to professional levels. The breadth of the dataset is suitable to test model's overall problem solving and knowledge ability.
- **BIG-Bench Hard (BBH)** [49] is a suite of 23 challenging BIG-Bench tasks [48], for which prior language models did not outperform the average human-rater. It can evaluate the model's abilty on challenging tasks.
- **Commonsense Reasoning** consists of 6 commonly-used datasets: PIQA [5], HellaSwag [55], WinoGrande [45], ARC easy and challenge [12], and OpenBookQA [18]. Those multi-choice tasks include carefully crafted riddles, puzzles, and scenarios designed to probe a model's ability to leverage implicit knowledge, make logical inferences, and navigate the unsaid rules of our physical and social worlds.
- **World Knowledge** includes NaturalQuestions [29] and TriviaQA [27]. Both tasks are designed to test model's question-answering ability in *closed book* setting. The models are not provided documents that may contain information about the question, and it has to rely on information learnt or memorized in pre-training data.
- **Reading Comprehension** uses BoolQ [11] to test model's *open book* comprehension ability. BoolQ is a question answering dataset for yes/no questions. Each example is a triplet of `(question, passage, answer)`, with the title of the page as optional additional context. The model is required to answer the question based on the given context in `passage`.

- **Math** ability is evaluated with GSM8K (Grade School Math 8K) [13]. GSM8K contains 8,500 grade school math problems. Both problems and answers are provided in natural language. These problems take between 2 and 8 steps to solve, which is ideal for testing basic multi-step reasoning ability of the model.
- **Code** evaluation uses HumanEval [9] dataset includes 164 programming problems with a function signature, docstring, body, and several unit tests. They were handwritten to ensure not to be included in the training set of code generation models.

5.1 Performance against Opensource models

We compare the performance of HLAT using the final (1.8T tokens) checkpoint with other opensource benchmarks in Table 3. For OpenLLaMA-1 and OpenLLaMA-2, we use the available pre-trained weights and evaluate using the same evaluation pipeline as our pre-trained model. For LLaMA-1 and LLaMA-2, we directly use results from corresponding papers [51, 52]. For all evaluation tasks, we adopt same hyper-parameters (number of shots) as used in [52]. We include both average performance and variance (in the parentheses, if available). All numbers are reported in percentage.

On MMLU tasks, HLAT performs better than OpenLLaMA-1 and LLaMA-1, and is worse then LLaMA-2. As discussed in [51], this may due to the difference in training datasets. We also compare the detailed performance of models on each MMLU subjects, including STEM, humanities, social science, and others (see Table 5 in Appendix). The performance is slightly worse in STEM, but similar trends is observed in OpenLLaMA-2. Comparing with LLaMA-2, HLAT is comparable on STEM and humanities. The gaps are mainly on social science and others.

On BBH, Commonsense Reasoning, and World Knowledge tasks, HLAT performs similar as OpenLLaMA-1 and OpenLLaMA-2 models. By diving deep into performance on each individual task (see Table 5 in Appendix), HLAT excels in 19/29 tasks as compared with OpenLLaMA-1, and 15/29 tasks compared with OpenLLaMA-2. Both HLAT and OpenLLaMA models have some gaps with LLaMA-1 and LLaMA-2 models. This might be due to using different evaluation libraries and slightly difference in datasets. Specifically, the gaps in Commonsense Reasoning and World Knowledge are mainly from OpenBookQA and TriviaQA, respectively. On TriviaQA, The results of LLaMA models [52] are reported on wikipedia subset; whereas HLAT and OpenLLaMA are evaluated on entire dataset.

On Math problems (GSM8K), HLAT performs significantly better than OpenLLaMA-1 and OpenLLaMA-2. As will be discussed in the next section, HLAT has a big improvement of Math ability in later training stage.

On Coding problems, HLAT performs better than OpenLLaMA-1, comparable with LLaMA-1, and worse than OpenLLaMA-2 and LLaMA-2. First, for OpenLLaMA-1, the tokenizer merges consecutive spaces which negatively affects the coding performance, as it eliminates important information such as indentation and line breaks. This issue is subsequently fixed in OpenLLaMA-2, which explains its better performance. Besides, OpenLLaMA-2 is trained with additional code data from StarCoder which also contributes to performance improvement.

5.2 Intermediate Model Performance

During the model training, we also evaluate the intermediate checkpoints every 200 billion tokens. On most benchmarks, the performance improves steadily, and correlates with the training loss of the model (see Figure 1a). Detailed evaluation results are shown in Appendix B (Table 6 and Figure 5).

We found that for different tasks, the model converges at different rates. Figure 3 plots the model performance on three sets of evaluation tasks with respective to number of seen training tokens (in billions).

For Commonsense Reasoning, the model accuracy improves quickly at beginning of training, and starts to saturate at later training stages. This is similar as the trends observed in other LLM model trainings [19, 51].

However, for Math task (GSM8K) shown in Figure 3b, the learning curve shows an exponentially increasing trend. It increase very gradually for the initial ~1 trillion tokens and begins to improve significantly during the later stages of training. Intuitively, this seems to indicate that the model is able to grasp more logical abilities after entering a relatively stable training plateau. We defer further research into this behavior as a future work.

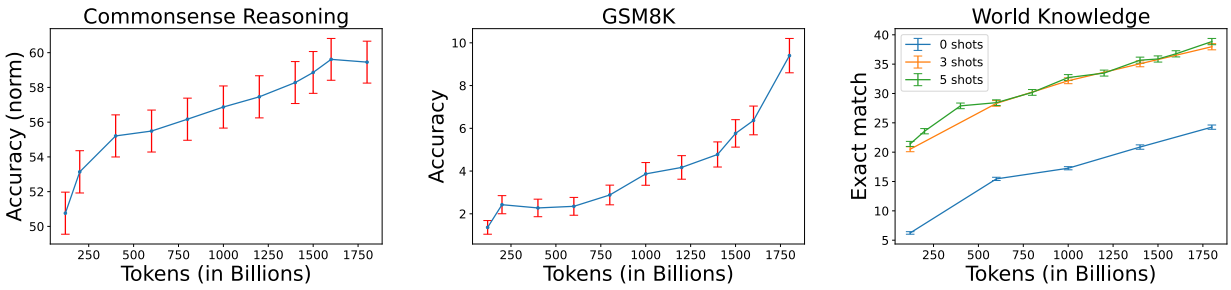
For World Knowledge task shown in Figure 3c, the performance increases almost linearly with number of training tokens. Since this is a *closed book* test and mainly evaluates the model's ability of memorizing facts in pre-training data, the model seems to consistently improve its ability on this domain with more training steps and epochs. In addition, we also tested if the trending is related to number of shots used in evaluation. The trends are very similar for zero-shot, 3-shot, and 5-shot tests.

Those observations indicate the necessity of a set of evaluation tasks covering a wide range of domains for LLM pre-training. A single validation set or evaluation tasks from narrow domains may not comprehensively reflect the actual over- or under-fitting of the model for general downstream tasks.

5.3 Truthfulness and Bias

We report the model's truthfulness and bias using TruthfulQA [33] and CrowS-pairs [38]. TruthfulQA presents a collection of meticulously crafted questions spanning diverse domains such as health, law, finance, and even politics. These queries deliberately target areas where human intuition and personal biases can lead to incorrect responses, and measure an LLM's resistance to misinformed or erroneous knowledge. CrowS-Pairs is a benchmark designed to probe LLMs for social biases across nine categories, including gender, religion, race/color, sexual orientation, age, nationality, disability, physical appearance and socioeconomic status. Each example is composed of a stereotype and an anti-stereotype.

We present the results in Table 4 with 0 shot inference. For TruthfulQA, we measure the multiple-choice score, and higher score shows better truthfulness. For CrowS-Pairs, it measures the percentage of models choosing answers of stereotypes, so lower scores indicates smaller bias. Overall, HLAT performs similar as other opensource models.



(a) (b) (c)

Figure 3: Intermediate model performance with number of seen tokens. (a) Commonsense Reasoning, (b) Math, (c) World Knowledge. We observe different trends of learning curves for different tasks. Detailed results of other tasks are in Figure 5.

Table 4: Model Truthfulness and Bias evaluation.

Dataset	Domains	Metric	OpenLLaMA-1	OpenLLaMA-2	LLaMA-1	HLAT
CrowS-Pairs	age	pct_stereotype	62.93 (4.997)	57.399 (5.1)	70.1	61.819 (5.002)
	physical appearance	pct_stereotype	66.667 (5.495)	73.611 (5.177)	77.8	62.5 (5.62)
	race/color	pct_stereotype	47.022 (2.226)	50.282 (2.256)	57.0	51.907 (2.271)
	disability	pct_stereotype	69.499 (5.702)	66.48 (5.757)	66.7	65.676 (5.893)
	gender	pct_stereotype	54.925 (2.756)	57.887 (2.742)	70.6	61.788 (2.696)
	nationality	pct_stereotype	48.969 (3.174)	45.502 (3.174)	64.2	51.273 (3.22)
	sexual orientation	pct_stereotype	77.207 (4.38)	80.946 (4.1)	81.0	82.607 (3.974)
	religion	pct_stereotype	72.652 (4.179)	70.043 (4.257)	79.0	68.723 (4.305)
	socioeconomic	pct_stereotype	68.649 (3.348)	72.309 (3.225)	71.5	66.88 (3.391)
TruthfulQA	-	multiple choice 1	23.133 (1.476)	22.644 (1.465)	-	23.623 (1.487)
		multiple choice 2	35.141 (1.355)	34.576 (1.348)	-	37.188 (1.349)

5.4 Efficiency and Scalability

We describe the training efficiency in terms of Cost per 4-million tokens (CPT) and scalability reported in [37]. For comparison, the GPU baseline is established using *p4d.24xlarge* instances and NeMo 23.08 [21] (available inside NeMo docker container with tag `23.08`) software stack. Figure 4 plots the normalized CPT of training on Trainium and scaling. For reference, we include both a 7B model and a 70B model (similar architecture as LLaMA-70B [52]). The Trainium CPT is normalized, such that the CPT of the GPU baseline is 100%. On 64 nodes, training HLAT 7B model costs about 54% of the GPU baseline.

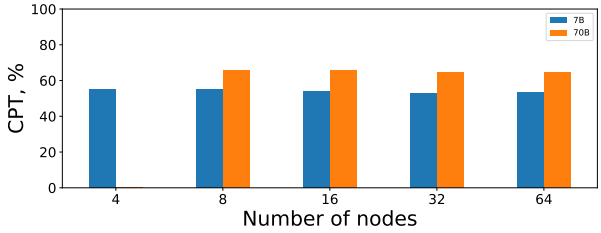


Figure 4: Normalized cost per 4 million tokens (CPT) for 7B and 70B models on AWS Trainium with various number of nodes. CPT of GPU baseline is normalized to 100%. 70B models ran into out-of-memory on 4 nodes.

5.5 Model Limitation

We note some limitations of HLAT in this section. Similar as other LLMs, HLAT suffers a set of limitations such as hallucinations, potential non-factual generations, biases, and toxicity [56]. For example, although comparable with other open-source pre-trained models, the bias of HLAT is still relative high on some subjects such as sexual orientation, physical appearance, religion, and socioeconomic (see Table 4). This is partially due to the usage of publicly available datasets. More importantly, as a pre-trained model HLAT has not gone through a supervised finetuning and human preference alignment. Those fine-tuning methods have been shown to be able to alleviate some limitations of pre-trained LLMs [52].

Another limitation is our training is stopped after 1.8 trillion tokens. As is suggested by Figure 3, HLAT may be able to further improve on certain tasks, such as math and world knowledge, with more training tokens.

6 BEST PRACTICES & FUTURE DIRECTIONS

In this section, we share some best practices we observed for training on AWS Trainium, and raise open questions for future research.

Parallelism: NDTL supports TP up to 32 degrees and pipeline parallelism. For a 7B model, we found that the combination of TP=8 and PP=1 provides the highest training throughput. However, for models of other sizes and architectures, the optimal parallelism configuration could vary. To achieve the highest training throughput, parallelism configuration needs to be jointly optimized with choice of activation checkpointing method, gradient accumulation steps, and training precision, to avoid out-of-memory on Trainium.

Training Precision: NDTL supports various training precision settings, including full precision (FP32), AMP BF16 mixed precision, BF16 with and without SR, etc. Full precision training is often memory-wise infeasible for multi-billion LLMs. We compared three training strategies for HLAT: pure BF16, BF16 with SR, and mixed precision training. Empirically, we found that training loss of pure BF16 diverges. BF16 with SR and mixed precision get similar training loss. We finally chose BF16 with SR for higher throughput. However, for models of other sizes and architecture, BF16 with SR may not guarantee the same convergence as mixed precision. Usually, the divergence can be observed in first few thousands of steps.

Choice of β_2: We observed that using $\beta_2 = 0.99$ causes training instability and slower convergence. This is related to the choice of BF16 with SR training precision. A large β_2 fails to capture the gradient explosion at current and recent steps, and hence does not effectively reduce the gradients in occurrence of gradient explosion. Switching to $\beta_2 = 0.95$ addresses the above-mentioned problem.

Weight decay: We applied weight decay to all layers. Empirically, weight decay is not applied to normalization and bias layers [15]. In our experiment, we did not found much performance-wise difference of those two methods.

Dataloader: Dataloader is another key factor in model performance. We compared our dataloader described in Section 3.3 with Nemo dataloader [21] to demonstrate better performance on our dataloader. The Nemo dataloader tends to overfit due to repetition within sequence.

Pre-compilation: Trainium requires pre-compiling the scripts to graphs. The compilation takes some time, especially for large models. Debugging on training scripts (e.g., printing out intermediate tensors) may require re-compilation. Instead of directly developing on a large model, we found it more efficient to develop and test on a smaller model and scale up afterwards.

7 RELATED WORK

LLM pre-training: After the Transformer architecture [53] was introduced, BERT [15] was proposed to pre-train a language model on a large corpus of unlabeled data. Following the success of BERT model on various NLP tasks, many pre-trained language models are later introduced with different architectures and training methods, such as GPT-2 [41], RoBERTa [34], BART [30], and so on [57]. Studies later observed significant performance improvement of language models by increasing model size and training data [25]. Such abilities are further demonstrated in LLMs such as GPT-3 [7], PaLM [3], LLaMA [51], Falcon [2], etc. Pre-trained on trillions of tokens, LLMs with tens or hundreds of billions parameters show remarkable ability in generating creative text contents, as well as a variety of downstream tasks, such as question answering, summarization, machine translation, programming, etc. [57].

AI accelerators: Most models are trained on NVIDIA GPU accelerators, such as GPT [7, 39] and LLaMA [51, 52]. Falcon-180B [2] was trained on AWS SageMaker, with up to 4,096 A100 40GB GPUs using *p4d* instances. However, the landscape of hardware accelerators for deep learning training has blossomed in recent years, with established players like NVIDIA GPUs facing fierce competition from custom offerings like Google's TPU and AWS Trainium. PaLM-2 [3] and OpenLLaMA [18] have demonstreated successful LLM pre-training on Google TPU. Recently, OLMo [19] is an opensource model developed by AI2. It has two models trained on AMD and Nvidia GPUs, separately. The two models have nearly identical performance on their evaluation suite by 2T tokens. AWS Trainium is a machine learning accelerator developed for deep learning training with high performance and cost-competitiveness. Our work is the first demonstration of end-to-end multi-billion LLM pre-trained on AWS Trainium. Ultimately, the optimal choice depends on the specific needs of the training task, with further research required to fully explore the potential of each accelerator and their possible convergence in future architectures.

8 CONCLUSION

In this paper, we pre-train HLAT, a 7 billion parameter large language model, using AWS Trainium over 1.8 trillion tokens. HLAT follows the decoder-only architecture and is trained with 64 AWS *trn1.32xlarge* instances. We evaluate the performance of HLAT against popular open-source baseline models including LLaMA and OpenLLaMA on a variety of popular benchmarking tasks. We find that HLAT achieves model quality on par with these baseline models. This work for the first time demonstrates that AWS Trainium with NDTL is able to successfully pre-train high-quality LLMs with high efficiency and low cost.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] 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. 2023. The Falcon Series of Open Language Models. arXiv:2311.16867 [cs.CL]

[3] Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric

Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussalem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. 2023. PaLM 2 Technical Report. arXiv:2305.10403 [cs.CL]

[4] Apache Arrow. 2020. Apache Arrow, a crosslanguage development platform for in-memory analytics. https://arrow.apache.org/.

[5] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 7432–7439.

[6] Alexander Borzunov, Max Ryabinin, Artem Chumachenko, Dmitry Baranchuk, Tim Dettmers, Younes Belkada, Pavel Samygin, and Colin A Raffel. 2024. Distributed Inference and Fine-tuning of Large Language Models Over The Internet. *Advances in Neural Information Processing Systems* 36 (2024).

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[8] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2023. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology* (2023).

[9] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. (2021). arXiv:2107.03374 [cs.LG]

[10] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training Deep Nets with Sublinear Memory Cost. arXiv:1604.06174 [cs.LG]

[11] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. In *NAACL*.

[12] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *ArXiv* abs/1803.05457 (2018). https://api.semanticscholar.org/CorpusID:3922816

[13] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168* (2021).

[14] Together Computer. 2023. *RedPajama-Data: An Open Source Recipe to Reproduce LLaMA training dataset.* https://github.com/togethercomputer/RedPajama-Data

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[16] FairScale authors. 2021. FairScale: A general purpose modular PyTorch library for high performance and large scale training. https://github.com/facebookresearch/fairscale.

[17] Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. 2021. A framework for few-shot language model evaluation. *Version v0. 0.1. Sept* (2021).

[18] Xinyang Geng and Hao Liu. 2023. *OpenLLaMA: An Open Reproduction of LLaMA.* https://github.com/openlm-research/open_llama

[19] Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Raghavi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, Tushar Khot, William Merrill, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Valentina Pyatkin, Abhilasha Ravichander, Dustin Schwenk, Saurabh Shah, Will Smith, Emma Strubell, Nishant Subramani, Mitchell Wortsman, Pradeep Dasigi, Nathan Lambert, Kyle Richardson, Luke Zettlemoyer, Jesse Dodge, Kyle Lo, Luca Soldaini, Noah A. Smith, and Hannaneh Hajishirzi. 2024. OLMo: Accelerating the Science of Language Models. arXiv:2402.00838 [cs.CL]

[20] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37* (Lille, France) *(ICML'15)*. JMLR.org, 1737–1746.

[21] Eric Harper, Somshubra Majumdar, Oleksii Kuchaiev, Li Jason, Yang Zhang, Evelina Bakhturina, Vahid Noroozi, Sandeep Subramanian, Koluguri Nithin, Huang Jocelyn, Fei Jia, Jagadeesh Balam, Xuesong Yang, Micha Livne, Yi Dong, Sean Naren, and Boris Ginsburg. [n. d.]. *NeMo: a toolkit for Conversational AI and Large Language Models.* https://github.com/NVIDIA/NeMo

[22] Dan Hendrycks, Collin Burns, Steven Basart, Andrew Critch, Jerry Li, Dawn Song, and Jacob Steinhardt. 2021. Aligning AI With Shared Human Values. *Proceedings of the International Conference on Learning Representations (ICLR)* (2021).

[23] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring Massive Multitask Language Understanding. *Proceedings of the International Conference on Learning Representations (ICLR)* (2021).

[24] Amr Hendy, Mohamed Abdelrehim, Amr Sharaf, Vikas Raunak, Mohamed Gabr, Hitokazu Matsushita, Young Jin Kim, Mohamed Afify, and Hany Hassan Awadalla. 2023. How good are gpt models at machine translation? a comprehensive evaluation. *arXiv preprint arXiv:2302.09210* (2023).

[25] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. 2022. Training Compute-Optimal Large Language Models. arXiv:2203.15556 [cs.CL]

[26] Drew A. Hudson and Christopher D. Manning. 2019. GQA: A New Dataset for Real-World Visual Reasoning and Compositional Question Answering. arXiv:1902.09506 [cs.CL]

[27] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551* (2017).

[28] Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. 2022. Reducing Activation Recomputation in Large Transformer Models. arXiv:2205.05198 [cs.LG]

[29] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural Questions: a Benchmark for Question Answering Research. *Transactions of the Association of Computational Linguistics* (2019).

[30] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 7871–7880. https://doi.org/10.18653/v1/2020.acl-main.703

[31] Junyi Li, Tianyi Tang, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2022. Pretrained language models for text generation: A survey. *arXiv preprint arXiv:2201.05273* (2022).

[32] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. StarCoder: may the

source be with you! (2023). arXiv:2305.06161 [cs.CL]
[33] Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring How Models Mimic Human Falsehoods. arXiv:2109.07958 [cs.CL]
[34] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692 [cs.CL]
[35] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Bkg6RiCqY7
[36] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. 2017. Mixed precision training. *arXiv preprint arXiv:1710.03740* (2017).
[37] N.A. [n. d.]. Paper under review.
[38] Nikita Nangia, Clara Vania, Rasika Bhalerao, and Samuel R. Bowman. 2020. CrowS-Pairs: A Challenge Dataset for Measuring Social Biases in Masked Language Models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Online.
[39] OpenAI. 2023. GPT-4 Technical Report. *ArXiv* abs/2303.08774 (2023). https://arxiv.org/abs/2303.08774
[40] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The RefinedWeb dataset for Falcon LLM: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116* (2023). arXiv:2306.01116 https://arxiv.org/abs/2306.01116
[41] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
[42] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. ArXiv. https://www.microsoft.com/en-us/research/publication/zero-memory-optimizations-toward-training-trillion-parameter-models/
[43] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3505–3506.
[44] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).
[45] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Commun. ACM* 64, 9 (2021), 99–106.
[46] Noam Shazeer. 2020. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202* (2020).
[47] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:1909.08053 [cs.CL]
[48] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2022. Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615* (2022).
[49] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. 2022. Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them. *arXiv preprint arXiv:2210.09261* (2022).
[50] Sho Takase, Shun Kiyono, Sosuke Kobayashi, and Jun Suzuki. 2023. Spike No More: Stabilizing the Pre-training of Large Language Models. arXiv:2312.16903 [cs.CL]
[51] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).
[52] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).
[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. 5998–6008.
[54] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, 38–45. https://www.aclweb.org/anthology/2020.emnlp-demos.6
[55] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830* (2019).
[56] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, and Shuming Shi. 2023. Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models. *arXiv preprint arXiv:2309.01219* (2023).
[57] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A Survey of Large Language Models. *arXiv preprint arXiv:2303.18223* (2023). http://arxiv.org/abs/2303.18223
[58] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. 2023. Pytorch FSDP: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277* (2023).
[59] Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Zhicheng Dou, and Ji-Rong Wen. 2023. Large language models for information retrieval: A survey. *arXiv preprint arXiv:2308.07107* (2023).

A DETAILED EVALUATION RESULTS

In Table 5 we show the evaluation results on all individual tasks described in Section 5. The number of shots are same as those listed in Table 3.

B EVALUATION RESULTS ON INTERMEDIATE CHECKPOINTS

In Table 6 we list the evaluation results of HLAT on intermediate checkpoints for every 200 billions of training tokens. For Commonsense Reasoning, we report the normalized accuracy if exists. For Code, we report pass@100. We also plot the trends in Figure 5.

C NEURON NEMO MEGATORN PACKAGE

We also experimented on AWS Neuron Nemo Megatron package[10], which is adapted from Nemo [21] for running on AWS Trainium accelerators. In Figure 6, we plot both training loss and gradient norm of two runs with same configuration on NDTL and Neuron Nemo Megatron, respectively. Neuron Nemo Megatron provides same convergence as NDTL.

[10]https://awsdocs-neuron.readthedocs-hosted.com/en/latest/libraries/nemo-megatron/index.html