# 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 led to a scarcity of the expensive conventional accelerators (such as GPUs), which emphasizes the need for the alternative specialized-accelerators that are *scalable and cost-efficient*. AWS TRAINIUM is the second-generation machine learning accelerator purposely built for training large deep learning models. However, training LLMs with billions of parameters on AWS TRAINIUM is challenging due to its relatively nascent software ecosystem. In this paper, we showcase HLAT: a family of 7B and 70B decoder-only LLMs pre-trained using 4096 AWS TRAINIUM accelerators over 1.8 trillion tokens. The performance of HLAT is benchmarked against popular open source 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 of similar model size. We also open-source all the training scripts and configurations of HLAT[1] and share the best practice of using the NeuronX Distributed Training (NxDT), a customized distributed training library for AWS TRAINIUM. Our work demonstrates that AWS TRAINIUM powered by NxDT is able to successfully pre-train state-of-the-art LLM models with high performance and cost-effectiveness.**

## I. INTRODUCTION

Large language models (LLMs), based on transformer architecture [1] and trained on massive text data, are the most recent breakthrough in artificial intelligence. They not only show remarkable capabilities in understanding and generating text [2], but offer immense potential across diverse downstream tasks, such as machine translation [3], information retrieval [4], code generation [5] and so on [6].

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 models, such as GPT [7] and LLaMA [8]–[10], the model is trained to predict the next token in a given sequence. 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 facilitates subsequent fine-tuning for various 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[2], Google TPU, and NVIDIA A100/H100 GPUs, have been specifically designed for such 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 which has limited memory and processing power to handle the massive datasets, model parameters, and intricate calculations involved in LLM training. Practitioners rely on distributed training libraries [11]–[14] 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, allowing training of models with multi-billion parameters. They also spread the workload across multiple machines, effectively tapping into a combined pool of resources, to 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 [11]–[14], 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, such as level of precision, dimensions of 3D parallelism, compiler flags and so on. On the other hand, Amazon EC2 *trn1* instance, equipped

*Both authors contributed equally to this research.
[1]https://github.com/awslabs/HLAT
[2]https://aws.amazon.com/machine-learning/trainium

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. This makes it appealing to fully utilize the compute power of AWS TRAINIUM for LLM pre-training.

In this paper, we make the following contributions:

- Our work is the first to reproduce a SOTA LLM model on a completely new hardware - AWS TRAINIUM. We end-to-end pre-train HLAT-7B and HLAT-70B (**H**igh-quality **L**LM pre-trained on **A**WS **T**RAINIUM), following the architecture described in [9] from scratch. The pre-training covers 1.8 trillion tokens and is performed on up to 256 Amazon EC2 *trn1.32xlarge* instances with totalling up to 4096 AWS TRAINIUM accelerators.
- We evaluate and show that both HLAT-7B and HLAT-70B perform comparable to models of similar size trained on other AI accelerators including LLaMA and OpenLLaMA. The evaluation is performed over a variety of tasks including commonsense reasoning, world knowledge, MMLU [15], math, coding, etc.
- We propose multiple techniques to improve training efficiency on AWS TRAINIUM such as a novel online dataloader, layer coalesing, selective activation checkpointing, precision strategy, and fault recovery mechanisms. These techniques save significant time and compute resources for pretraining on large datasets, and can be applied to other accelerators (GPU, TPU, etc.) as well.
- We open-source all the training scripts and configurations of HLAT-7B and HLAT-70B models[3], including the Pytorch training script, model definition script, and training configuration files (including all hyper-parameters). We also share best practices of pre-training on AWS TRAINIUM and NeuronX Distributed Training (NxDT), such as sharding strategies, training precisions, compiler settings, etc. With those artifacts, practitioners can easily reproduce HLAT and pre-train their own custom LLMs on AWS TRAINIUM.

## II. BACKGROUND - DISTRIBUTED TRAINING ON AWS 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 bandwidth 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 (*trn1.32xlarge* \$21.50 vs. *p4d.24xlarge* \$32.77).

[3]https://github.com/awslabs/HLAT

**AWS Neuron** is a software development kit (SDK)[4] 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 [14] and TensorFlow [16], and offers features such as FP32 autocasting, stochastic rounding, collective communication, custom operators, and so on.

**NeuronX Distributed Training** (NxDT)[5], as part of Neuron SDK, is developed to enable high-efficiency distributed training on TRAINIUM: NxDT supports a variety of distributed training techniques, such as 3D parallelism [12], i.e., Tensor Parallelism (TP), Pipeline Parallelism (PP) and Data Parallelism (DP). To reduce the activation memory during training, activation checkpointing [17] and sequence parallelism [18] are naturally supported with the 3D parallelism. NxDT also supports Zero Redundancy Optimizer Stage 1 (ZeRO-1) [19] to shard optimizer states, which can be applied simultaneously with 3D parallelism. NxDT provides unified interfaces to port custom models and run training scripts on AWS TRAINIUM. To train models from Huggingface `transformers` library on TRAINIUM accelerators with NxDT, it only requires simple code changes in certain layers of the model. NxDT supports TP with mixed degrees, i.e., users can use more than one TP degrees to shard different model parameters. This is helpful when some model parts of LLMs are not compatible with a unified large TP degree, e.g., with Grouped Query Attention (GQA) [20]. Finally, NxDT supports automatic fault recovery and checkpointing. In case of hardware failures or communication timeouts, NxDT can automatically restart training from latest auto-saved checkpoints without manual intervention, which is critical for maintaining system uptime and training efficiency.

## III. METHOD

### A. Model Architecture and Hyperparameters

HLAT models adopt the decoder-only transformer architecture and apply same modifications used in LLaMA [8]–[10], including pre-normalization with RMSNorm, SwiGLU activation function, and Rotary Embeddings. HLAT-70B in addition applies GQA [20] with group size of 8. The models are trained with a maximum sequence length of 4096.

We adopt training hyperparameters used in LLaMA2 [9]. Specifically, the global batch size is 1024 sequences, so each step covers about 4 million tokens. We use a cosine learning rate scheduler. The maximum learning rate is $3e^{-4}$ for HLAT-7B, and $1.5e^{-4}$ for HLAT-70B. The minimum learning rate decays to 10% of maximum learning rate. 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.

[4]https://github.com/aws-neuron/aws-neuron-sdk

[5]https://awsdocs-neuron.readthedocs-hosted.com/en/latest/libraries/nxd-training/overview.html

## B. Training Dataset and Dataloader

Our pre-training dataset includes RedPajama-1T [21], peS2o [22], and OpenWebMath [23]. HLAT-7B is purely trained on RedPajama-1T with 1.8 trillion tokens. HLAT-70B is initially trained on RedPajama-1T over 1.4 trillion tokens and is then continually trained on an up-sampled dataset with RedPajama-1T, peS2o and OpenWebMath for 400B tokens (see Section V-C for details).

We designed a novel dataloader which performs both tokenization and packing *online* during training. The dataloader takes one or more dataset files in Apache Arrow format [24]. All samples are randomly shuffled and split into several subsets according to the total DP 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 concatenated sequence is used in the following sequence. The samples within a concatenated sequence is concatenated with a special end of sentence (EOS) token. This gives the model necessary information to infer that the text separated by EOS token are unrelated [7]. Note that samples in the same concatenated sequence may be from very different sources or can be of different formats (e.g. natural language and codes). Finally, each batch of sequences are tokenized on the fly during the training.

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 TRAINIUM devices for training, so the computations are in parallel. In comparison, the offline dataloader requires pre-tokenization of entire datasets which costs a lot of developer time and compute resources for large datasets. The offline Nemo dataloader [25], for example, takes about 154 hours to pre-tokenize a dataset with 2 trillion tokens.

## C. Orchestration

HLAT pre-training is performed on clusters with 256 *trn1.32xlarge* instances (nodes), totalling to 4096 AWS TRAINIUM accelerators. Both Amazon EKS and AWS ParallelCluster can effectively manage the training cluster. Accelerators within same node are connected with NeuronLink[6]. The nodes within the cluster are interconnected through Elastic Fabric Adapter (EFA)[7]. 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.

## D. Training Efficiency

LLaMA [8] 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**: HLAT-7B is pre-trained over 64 nodes with TP=8, PP=1, and DP=256. HLAT-70B is pre-trained over 256 nodes with TP=32, PP=8, and DP=32. Both use sequence parallel (SP). This sharding configuration is observed to have the highest throughput.

**Selective Activation Checkpointing**: We use selective activation checkpointing [18] to improve the training efficiency. It has slightly higher memory cost as full activation checkpointing, but increases the overall training throughput.

**Training precision**: Pre-training with full precision (FP32) is in-efficient for large LLMs, but generic half-precision training (BF16 or FP16) often has numerical stability issues [26]. On GPU, mixed precision training [26] is widely used to achieve similar precision as full FP32 with better efficiency. For HLAT-7B, we used BF16 with stochastic rounding (SR) [27], featured by AWS TRAINIUM. 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. Empirically, we found that BF16 with SR shows the same convergence behavior as mixed precision training for HLAT-7B, with higher training throughput and lower memory footprint. However, on HLAT-70B, BF16 with SR shows worse convergence than mixed precision training. We found that mainly due to the nondeterministic normalization error introduced by stochastic rounding (see Section IV-C), and low-precision computations errors in gradient and attention operations. Such errors accumulate and signify on large models with more parameters and deeper layers. Therefore, we developed a standard mixed precision training strategy. Specifically, unless specified, all computation and storage use BF16 without stochastic rounding. This strategy uses FP32 in precision sensitive operators, local gradient accumulation, and global gradient synchronization. It also uses master weights and FP32 optimizer states. Finally, it uses ZeRO-1 [11] for memory efficiency.

**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 accelerators directly before use. The masked-out tensors will not be computed in the first place, which avoids redundant computation, saves host memory usage, and increases training throughput. To enable this feature in training script, the attention mask tensor is directly defined using `torch.triu` function and mapped to `device='xla'`. The Neuron compiler will therefore enable the optimization during compilation.

**Coalescing Layers with Same Inputs**: We coalesced linear layers with the same inputs to reduce the communication

[6]https://awsdocs-neuron.readthedocs-hosted.com/en/latest/general/arch/neuron-hardware/trainium.html

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

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 [28] are also coalesced. Listing 1 shows a code snippet for implementation. Note that this technique also applies to other accelerators such as GPU and TPU.

```
# Without layer coalescing
query_states = q_proj(hidden_states) # hidden_size
key_states = k_proj(hidden_states)   # hidden_size
value_states = v_proj(hidden_states) # hidden_size

# With layer coalescing
qkv_states = qkv_proj(hidden_states) # 3*hidden_size
query_states, key_states, value_states = qkv_states.
    split(3, dim=2)
```

Listing 1: Example of layer coalescing.

**Compiler Optimization**: we use compiling flag `--distribution-strategy=llm-training` and `--model-type=transformer` to enable the compiler to perform optimizations applicable to LLM (transformer model) training runs that shard parameters, gradients, and optimizer states across data-parallel workers. We set Neuron environment variable `NEURON_FUSE_SOFTMAX=1` to enable compiler optimizations on custom lowering for Softmax 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).

## IV. Training Process

### A. Training Curves

During the training process, we monitor the cross entropy 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 rapidly over the initial ∼250 billion tokens, and enters a log-linear decrease afterwards. Similar trends are observed in other LLM pre-training [8], [9], [29].

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 common in LLM pre-training when using layer-normalization, or even RMSNorm [30], and sometimes 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.

Note that sustained spikes in the gradient norm leads to training divergence due to improper weight updates, even after gradient normalization through clipping (see Section III-A). 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 $r+0.1$. We then 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 [31].

### B. Hardware and System Failures

Pre-training on a cluster with thousands of accelerators often faces hardware-level errors that interrupts the training process. Those errors could be due to malfunction of AWS Trainium chips or host machines, network communication timeouts, and so on [32]. Manually restarting the pre-training from last-saved checkpoints demands significant developer time and causes low system uptime. For example, we performed an experimental training run (over 600 billion tokens) without automatic fault recovery, and we observed an average system uptime of 77.83%. In HLAT pre-training, we enabled automatic fault recovery mechanism in NxDT (see Section II), which automatically replaces the faulty nodes and restart training from latest checkpoints. The overall system uptime of HLAT pre-training is then improved 20% to 98.81%.

### C. Training Convergence and 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 [30], we found better numerical stability and convergence with smaller initial variance on deeper layers.

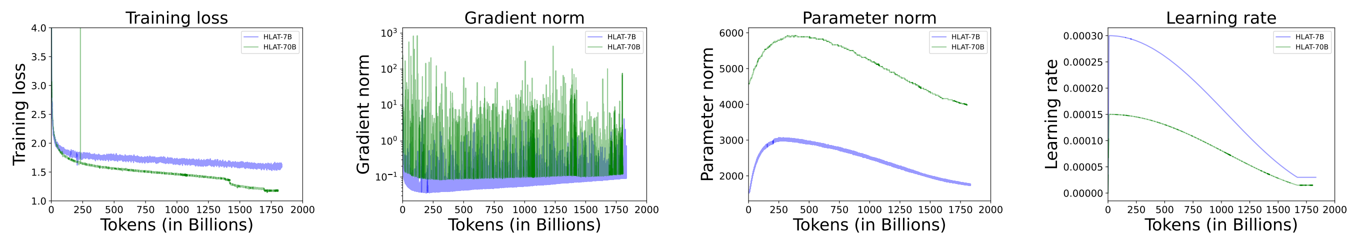
**Normalization**: We used tensor parallelism to shard the model parameter matrices except normalization layers. For HLAT-7B, the normalization layer weights, however, are slightly different across TP ranks due to stochastic rounding. Empirically, we found the differences are small, and RMSNorm weights values are all close to 1. Note that with standard mixed precision strategy, we do not see such weight difference for HLAT-70B.

**Checkpoint Averaging**: For HLAT-70B, the final checkpoint used for evaluation is an average of last two checkpoints in the training process. Similar as [10], [33], we found averaging the last two checkpoints provides better performance than each single checkpoint. We provide more detailed discussion in Section V-D.

**Neuron Persistent Cache on Local Worker**: In HLAT pre-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

[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)

Fig. 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 III-A).

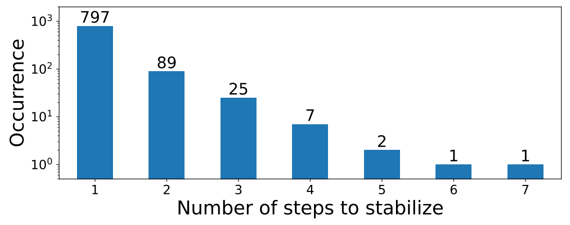


Fig. 2: Number of occurrence of sustained gradient spikes vs contiguous length of appearance. Over 86%, the spike lasts for only a single step.

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 caches in file system of each local worker.

## V. EVALUATION

**Baselines:** We evaluate HLAT against several open-source benchmark models. Since HLAT structure is similar as LLaMA model, we include LLaMA-1 (7B, 13B, 65B) [8], LLaMA-2 (7B, 70B) [9], OpenLLaMA-1 (7B, 13B) and OpenLLaMA-2 (7B) [29]. The model architecture and composition of the training data of the models being compared are listed in Table I. OpenLLaMA-1 model is trained on RedPajama [21] dataset. OpenLLaMA-2 model shares same structure as OpenLLaMA-1, but is trained on a different data mixture which includes data from Falcon-RefinedWeb [34], StarCoder [35], and RedPajama [21].

TABLE I: Model architectures comparison.

| Model | Sizes | Sequence length |
|---|---|---|
| OpenLLaMA1 | 7B, 13B | 2048 |
| OpenLLaMA2 | 7B | 2048 |
| LLaMA1 | 7B, 13B, 33B, 65B | 2048 |
| LLaMA2 | 7B, 13B, 70B | 4096 |
| HLAT | 7B, 70B | 4096 |

**Evaluation Tasks:** We evaluate HLAT against baselines on 7 groups of tasks including both zero-shot and few-shot tasks [36]. We use HumanEval [37] for coding tasks, and Language Model Evaluation Harness [38] for others.

**Massive Multitask Language Understanding (MMLU)** [15], [39] contains 57 tasks, spanning STEM, social sciences, humanities, and other subjects. The difficulty ranges from elementary to professional levels. The breadth of the dataset tests model's overall problem solving and knowledge ability.

**Commonsense Reasoning (CR)** consists of 6 datasets: PIQA [40], HellaSwag [41], WinoGrande [42], ARC easy and challenge [43], and OpenBookQA [29]. These 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 rules of physical and social worlds.

**World Knowledge (WK)** includes NaturalQuestions [44] and TriviaQA [45]. 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 (RC)** uses BoolQ [46] 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) [47]. 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.

**Code** evaluation uses HumanEval [37] dataset including 164 programming problems with a function signature, docstring, body, and several unit tests. They were handwritten to ensure not present in the training set of the models.

### A. *Performance against open-source Models*

We compare the performance of HLAT with other open-source benchmarks in Table II. The numbers are reported in percentage and for HLAT results, we include both mean and

TABLE II: Evaluation of HLAT against 4 open-source models on 6 groups of tasks described in Section V. Numbers in the parentheses represent standard deviation, if available.

| Model | Size | MMLU | CR | WK | RC | Math | Code | | Average |
|---|---|---|---|---|---|---|---|---|---|
| - | - | accuracy | accuracy | exact match | accuracy | accuracy | pass@1 | pass@10 | - |
| OpenLLaMA-1 | 7B | 30.5 | 58.4 | 40.6 | 70.5 | 5.2 | 4.5 | 13.4 | 41.2 |
| OpenLLaMA-2 | 7B | 41.1 | 61.3 | 37.9 | 72.4 | 6.8 | 9.7 | 25 | 44.9 |
| LLaMA-1 | 7B | 35.1 | 63.5 | 43.6 | 76.5 | 11 | 10.5 | 21.3 | 47.4 |
| LLaMA-2 | 7B | 45.3 | 64 | 45.2 | 77.4 | 14.6 | 12.2 | 25.2 | 49.2 |
| **HLAT-7B** | 7B | 41.3 (3.6) | 59.5 (1.2) | 38.8 (0.5) | 72.5 (0.8) | 9.4 (0.8) | 7.6 | 19.8 | 44.6 |
| OpenLLaMA-1 | 13B | 43.5 | 62 | 45.9 | 72.3 | 8.3 | 7 | 17 | 47.1 |
| LLaMA-1 | 13B | 46.9 | 65.3 | 49.7 | 78.1 | 17.8 | 15.8 | 22.6 | 53.1 |
| LLaMA-2 | 13B | 45.3 | 66.3 | 50.5 | 81.7 | 28.7 | 18.3 | 30.5 | 54.6 |
| LLaMA-1 | 33B | 57.8 | 68.9 | 54.6 | 83.1 | 35.6 | 21.7 | 38.4 | 59.2 |
| LLaMA-1 | 65B | 63.4 | 69.8 | 57 | 85.3 | 50.9 | 23.7 | - | 62.1 |
| LLaMA-2 | 70B | 68.9 | 70.7 | 59 | 85 | 56.8 | 30.5 | 59.4 | 64.7 |
| **HLAT-70B** | 70B | 65.1 (3.4) | 67.3 (1.2) | 54.5 (0.6) | 82.6 (0.7) | 48.5 (1.4) | 21.4 | 57.9 | 60.8 |

standard deviation (in the parentheses, if available). We also report an average score over all tasks in the last column.

HLAT-7B performs better than OpenLLaMA-1 and is on-par with OpenLLaMA-2. Both HLAT-7B and OpenLLaMA models have some gap with LLaMA-1 and LLaMA-2, which is likely due to the training data quality. Even though the data composition of RedPajama-1T is similar as those used in LLaMA-1, the data cleaning pipeline and final data quality are different, which therefore affects the model performance [48]. For HLAT-70B, we use the same training dataset as the 7B model for consistency. Although there is no OpenLLaMA baseline for a fair comparison, HLAT-70B performs better than LLaMA-1 and LLaMA-2 models of smaller sizes. The model performance gap with LLaMA-1 (65B) and LLaMA-2 (70B) is also smaller than those on 7B models. We acknowledge the lack of effort on data quality improvement, but our main goal is to showcase the effectiveness and efficiency of AWS Trainium.

On MMLU (5-shot), both HLAT models perform better than OpenLLaMA-1 and LLaMA-1 models of similar size. The performance is slightly worse than LLaMA-2 family of models, likely due to the difference in training dataset size and composition [8].

On Commonsense Reasoning (0-shot) and World Knowledge (5-shot), HLAT-7B performs similar to OpenLLaMA-1 and OpenLLaMA-2 models. By diving deep into performance on each individual task, HLAT-7B 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, which may be due to the training set quality. Nevertheless, the gap ($\sim 3\%$) is consistent on 7B and 70B models.

On Math problems (GSM8K, 8-shot), HLAT-7B performs significantly better than OpenLLaMA-1 and OpenLLaMA-2. As will be discussed in the next section, HLAT-70B has a big improvement of Math ability in later training stage. HLAT-70B performs similar as LLaMA1-65B, and we observed significant improvement in upsampling training stage.

On Coding problems, both HLAT-7B and HLAT-70B perform comparable with LLaMA-1. HLAT-7B performs better than OpenLLaMA-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.

## B. Intermediate Model Performance

During the model training, we also evaluate the intermediate checkpoints about every 200 billion tokens. Figure 3 and Figure 4 show the model performance of HLAT-7B and HLAT-70B with respect to number of seen training tokens (in billions), respectively. On most benchmarks, the performance improves steadily, and correlates with the training loss.

We found that for different tasks, the model converges at different rates. 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 [8], [49].

However, for Math task (GSM8K) shown in Figure 3e, the learning curve shows an exponentially increasing trend. It increase very gradually for the initial $\sim$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 tested if the trending is related to number of shots used in evaluation. It turns out that the trends are very similar for zero-shot, 3-shot, and 5-shot tests.

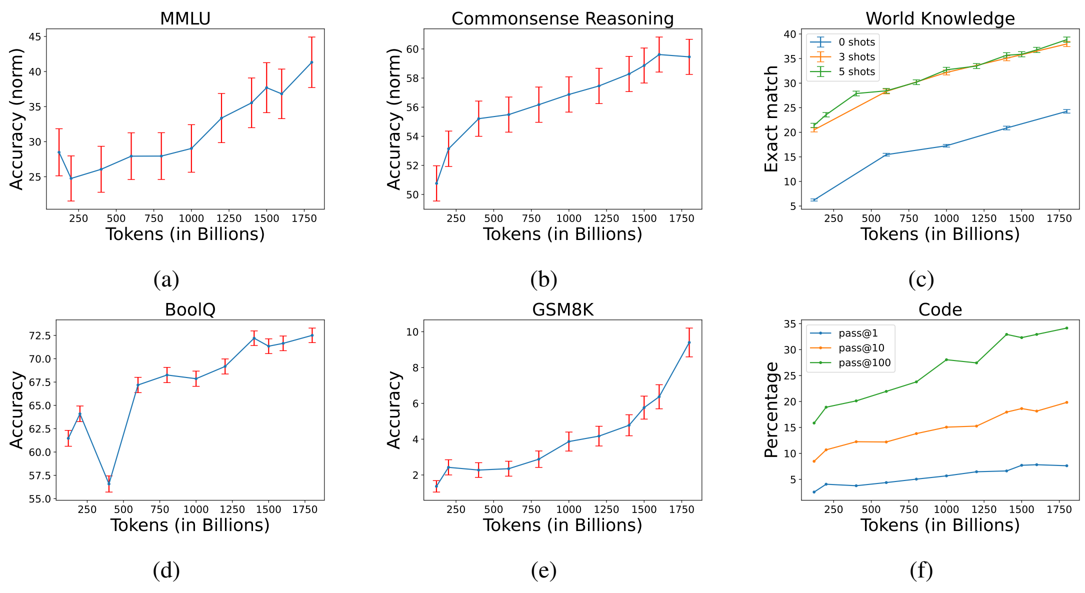


Fig. 3: Intermediate model performance with number of seen tokens for HLAT-7B.

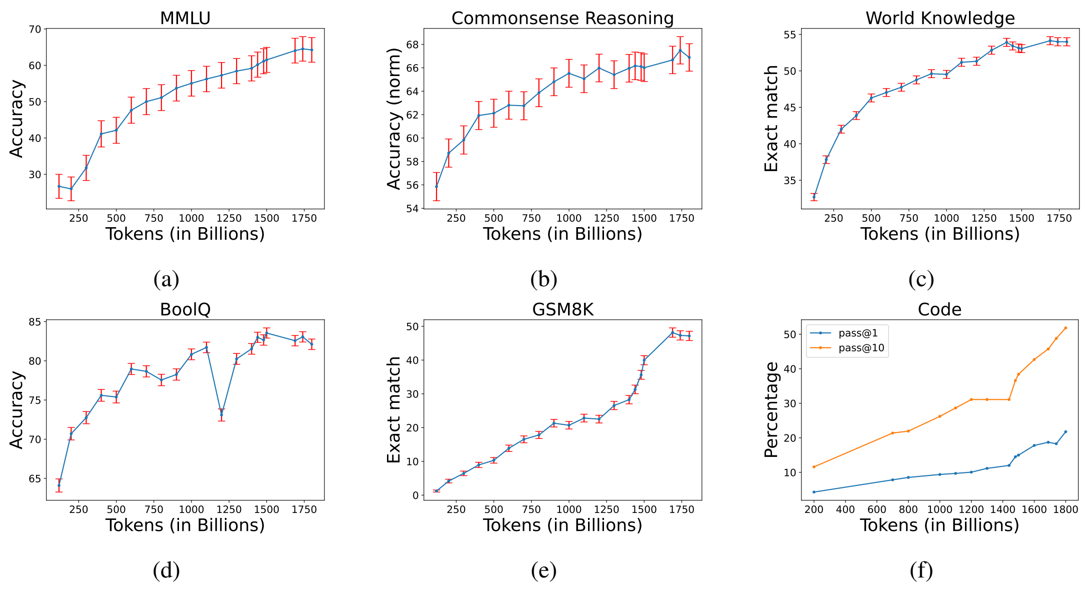


Fig. 4: Intermediate model performance with number of seen tokens for HLAT-70B.

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 fully reflect the actual over- or under-fitting of the model for general downstream performance.

## C. Upsampling

During HLAT-70B training, we upsampled the training dataset in last 400B tokens. Specifically, we use 35.47% web data, 41.27% math data, and 23.26% coding data with more details listed in Table III. In Figure 4, we plot the evaluation performance of HLAT-70B with seen training tokens. In upsampling training stage, that is, after 1400B tokens,

TABLE III: Upsampling dataset composition for HLAT-70B.

| | Datasets | Size (billions of tokens) | Percentage |
|---|---|---|---|
| Web Data | Wikipedia [21]<br>C4 [21] | 90 | 35.47% |
| Domain Specific | StackExchange<br>Arxiv [21]<br>Open-Web-Math [23]<br>PeS2o [22] | 104.7 | 41.27% |
| Code | Github [21] | 59 | 23.26% |
| Total | - | 253.7 | 15.16% |

we observe significant model performance improvement over math, coding, and MMLU performance. It improved math by 10% and coding by 5%. This is consistent with the findings in LLaMA-3 [10], where the researchers found significant improvement of LLaMA-3 8B model on math problems. However, they mentioned such method did not help much for 405B models. Our experiment fills the model size gap, and shows that upsampling still helps for a 70B model.

## D. *Checkpoint Averaging*

For HLAT-70B, we average the last two checkpoints used in pre-training to generate a checkpoint for final evaluation [10], [33]. Table IV compares the individual checkpoints with 1740B training tokens and 1800B training tokens, as well as the averaged checkpoints. The averaged checkpoints outperforms individual checkpoints on average performance, as well as some individual tasks.

TABLE IV: Evaluation of HLAT-70B with individual and averaged checkpoints.

| Checkpoint | MMLU | RC | WK | CR | Math | Code | Avg. |
|---|---|---|---|---|---|---|---|
| 1740B | 64.5 | 67.5 | 54 | 83.1 | 47.3 | 18.3 | 60.3 |
| 1800B | 64.2 | 66.9 | 54 | 82.1 | 47.2 | 21.8 | 60.2 |
| Average | 65.1 | 67.3 | 54.5 | 82.6 | 48.5 | 21.4 | 60.8 |

## E. *Truthfulness and Bias*

We report the model's truthfulness and bias using TruthfulQA [50] and CrowS-pairs [51]. 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.

TABLE V: Model Truthfulness and Bias evaluation. CrowS-pairs (CSP) uses *percentage of stereotypes* as metric and TruthfulQA (TQA) uses *multiple choice accuracy* as metric.

| Dataset | Size | CSP (↓) | CSP | TQA (↑) | TQA |
|---|---|---|---|---|---|
| Tasks | - | english | french | mc1 | mc2 |
| OpenLLaMA-1 | 7B | 64.6 | 50.1 | 23.1 | 35.1 |
| OpenLLaMA-2 | 7B | 65.6 | 51.7 | 22.6 | 34.6 |
| LLaMA-1 | 7B | 53.7 | 47.5 | 22.0 | 34.1 |
| LLaMA-2 | 7B | 66.9 | 54.9 | 25.2 | 39.0 |
| **HLAT-7B** | 7B | 65.2 | 54.5 | 23.6 | 37.2 |
| LLaMA-1 | 65B | 69.3 | 58.3 | 27.9 | 42.6 |
| LLaMA-2 | 70B | 69.8 | 63.5 | 30.6 | 44.8 |
| **HLAT-70B** | 70B | 68.1 | 59.1 | 32.3 | 45.9 |

We present the results in Table V 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 to other open-source models.

## F. *Efficiency and Scalability*

We describe the training efficiency in terms of Cost per 4-million tokens (CPT) and scalability reported in [52]. The CPT is defined as CPT $= \frac{C}{T \times 3600} \times 4e^6$, where $C$ is instance cost per hour ($21.50 for Trainium, and $32.77 for GPU), $T$ is training throughput (tokens per second). CPT quantifies both the training speed and also hardware cost. We use this metric to compare training efficiency of Trainium and GPU.

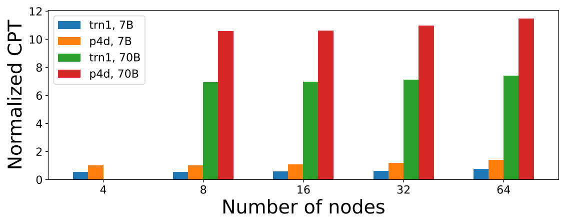


Fig. 5: Normalized cost per 4 million tokens (CPT) for 7B and 70B models on AWS TRAINIUM with various number of nodes. CPT of GPU baseline (*p4d*, 7B) with 4 nodes is normalized to 100%. 70B models ran into out-of-memory on 4 nodes.

For comparison, the GPU baseline is established using *p4d.24xlarge* instances and NeMo 23.08 [25] (available inside NeMo docker container with tag `23.08`) software stack. Figure 5 plots the normalized CPT of training on TRAINIUM and scaling. The TRAINIUM CPT is normalized, such that the CPT of the GPU baseline (*p4d*, 7B) on 4 nodes is 100%. Overall, the training cost on *trn1* is approximately 60% of GPU, and is consistent with the number of nodes. In addition, the CPTs on 70B models are roughly 10 times of those on 7B models.

## G. *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 [53]. 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 V). 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 [9]. Another limitation is that our training is stopped after 1.8 trillion tokens. As is suggested by LLaMA-3 [10], HLAT may be able to further improve on certain tasks, such as math, world knowledge, MMLU, and coding, with more training tokens.

## VI. 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**: NxDT 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, but not for HLAT-70B. So the optimal parallelism configuration varies with model sizes and architectures. 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 balance memory and communication costs.

**Training Precision**: NxDT supports various training precision configurations, including full precision (FP32), BF16 with and without SR, standard mixed precision training, etc. Full precision training is often memory-wise infeasible for multi-billion LLMs. We compared multiple training strategies for HLAT: pure BF16, BF16 with SR and standard mixed precision training. Empirically, we found that training loss of pure BF16 diverges. BF16 with SR shows similar training loss as mixed precision on HLAT-7B model, but converges slower on HLAT-70B. We finally chose BF16 with SR for higher throughput on HLAT-7B, but standard mixed precision on HLAT-70B. For models of other sizes and architecture, warm-up study may be needed to decide the optimal training precision. Usually, the divergence can be observed in first few thousands of steps.

**Choice of $\beta_2$**: 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 $\beta_2$ 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 [54]. In our experiment, we did not found much performance-wise difference of those two methods.

**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.

## VII. Related Work

**LLM pre-training:** After the Transformer architecture [1] was introduced, BERT [54] 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 [55], RoBERTa [56], BART [57], and so on [6]. Studies later observed significant performance improvement of language models by increasing model size and training data [58]. Such abilities are further demonstrated in LLMs such as GPT-3 [7], PaLM [59], LLaMA [8]–[10], Falcon [60], Gemini [61], Phi [48], 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. [6].

**AI accelerators:** Most models are trained on NVIDIA GPU accelerators, such as GPT [7], [62] and LLaMA [8], [9]. Falcon-180B [60] 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 [59] and OpenLLaMA [29] have demonstrated successful LLM pre-training on Google TPU. Recently, OLMo [49] is an open-source 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.

## VIII. Conclusion

In this paper, we pre-train HLAT, a family of 7 billion and 70 billion parameter large language models, using AWS Trainium over $\sim$1.8 trillion tokens. HLAT follows the decoder-only architecture and is trained with up to 256 Amazon EC2 *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 of similar sizes. This work demonstrates, for the first time, that AWS Trainium with NxDT is able to successfully pre-train high-quality LLMs with high efficiency and low cost.

## References

[1] A. Vaswani, N. Shazeer *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[2] J. Li, T. Tang *et al.*, "Pretrained language models for text generation: A survey," *arXiv preprint arXiv:2201.05273*, 2022.

[3] A. Hendy, M. Abdelrehim *et al.*, "How good are gpt models at machine translation? a comprehensive evaluation," *arXiv preprint arXiv:2302.09210*, 2023.

[4] Y. Zhu, H. Yuan *et al.*, "Large language models for information retrieval: A survey," *arXiv preprint arXiv:2308.07107*, 2023.

[5] B. Roziere, J. Gehring *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[6] W. X. Zhao, K. Zhou *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[7] T. Brown, B. Mann *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato *et al.*, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[8] H. Touvron, T. Lavril *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[9] H. Touvron, L. Martin *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[10] A. Dubey, A. Jauhri *et al.*, "The llama 3 herd of models," 2024.

[11] J. Rasley, S. Rajbhandari *et al.*, "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*, 2020, pp. 3505–3506.

[12] M. Shoeybi, M. Patwary *et al.*, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[13] FairScale authors, "Fairscale: A general purpose modular pytorch library for high performance and large scale training," https://github.com/facebookresearch/fairscale, 2021.

[14] Y. Zhao, A. Gu *et al.*, "Pytorch fsdp: experiences on scaling fully sharded data parallel," *arXiv preprint arXiv:2304.11277*, 2023.

[15] D. Hendrycks, C. Burns *et al.*, "Aligning ai with shared human values," *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[16] M. Abadi, A. Agarwal *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.

[17] T. Chen, B. Xu *et al.*, "Training deep nets with sublinear memory cost," *arXiv preprint arXiv:1604.06174*, 2016.

[18] V. A. Korthikanti, J. Casper *et al.*, "Reducing activation recomputation in large transformer models," *Proceedings of Machine Learning and Systems*, vol. 5, pp. 341–353, 2023.

[19] S. Rajbhandari, J. Rasley *et al.*, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16.

[20] D. A. Hudson and C. D. Manning, "Gqa: A new dataset for real-world visual reasoning and compositional question answering," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 6700–6709.

[21] T. Computer, "Redpajama: an open dataset for training large language models," 2023. [Online]. Available: https://github.com/togethercomputer/RedPajama-Data

[22] L. Soldaini and K. Lo, "peS2o (Pretraining Efficiently on S2ORC) Dataset," Allen Institute for AI, Tech. Rep., 2023.

[23] K. Paster, M. D. Santos *et al.*, "Openwebmath: An open dataset of high-quality mathematical web text," *arXiv preprint arXiv:2310.06786*, 2023.

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

[25] E. Harper, S. Majumdar *et al.*, "NeMo: a toolkit for Conversational AI and Large Language Models." [Online]. Available: https://github.com/NVIDIA/NeMo

[26] P. Micikevicius, S. Narang *et al.*, "Mixed precision training," *arXiv preprint arXiv:1710.03740*, 2017.

[27] S. Gupta, A. Agrawal *et al.*, "Deep learning with limited numerical precision," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 1737–1746.

[28] N. Shazeer, "Glu variants improve transformer," *arXiv preprint arXiv:2002.05202*, 2020.

[29] X. Geng and H. Liu, "Openllama: An open reproduction of llama," May 2023. [Online]. Available: https://github.com/openlm-research/open_llama

[30] S. Takase, S. Kiyono *et al.*, "Spike no more: Stabilizing the pre-training of large language models," *arXiv preprint arXiv:2312.16903*, 2023.

[31] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2019.

[32] A. Borzunov, M. Ryabinin *et al.*, "Distributed inference and fine-tuning of large language models over the internet," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[33] P. Izmailov, D. Podoprikhin *et al.*, "Averaging weights leads to wider optima and better generalization," *arXiv preprint arXiv:1803.05407*, 2018.

[34] G. Penedo, Q. Malartic *et al.*, "The RefinedWeb dataset for Falcon LLM: outperforming curated corpora with web data, and web data only," *arXiv preprint arXiv:2306.01116*, 2023.

[35] R. Li, L. B. Allal *et al.*, "Starcoder: may the source be with you!" 2023.

[36] Y. Chang, X. Wang *et al.*, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, 2023.

[37] M. Chen, J. Tworek *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[38] L. Gao, J. Tow *et al.*, "A framework for few-shot language model evaluation," *Version v0. 0.1. Sept*, 2021.

[39] D. Hendrycks, C. Burns *et al.*, "Measuring massive multitask language understanding," *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[40] Y. Bisk, R. Zellers *et al.*, "Piqa: Reasoning about physical commonsense in natural language," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 05, 2020, pp. 7432–7439.

[41] R. Zellers, A. Holtzman *et al.*, "Hellaswag: Can a machine really finish your sentence?" *arXiv preprint arXiv:1905.07830*, 2019.

[42] K. Sakaguchi, R. L. Bras *et al.*, "Winogrande: An adversarial winograd schema challenge at scale," *Communications of the ACM*, vol. 64, no. 9, pp. 99–106, 2021.

[43] P. Clark, I. Cowhey *et al.*, "Think you have solved question answering? try arc, the ai2 reasoning challenge," *ArXiv*, vol. abs/1803.05457, 2018.

[44] T. Kwiatkowski, J. Palomaki *et al.*, "Natural questions: a benchmark for question answering research," *Transactions of the Association of Computational Linguistics*, 2019.

[45] M. Joshi, E. Choi *et al.*, "Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension," *arXiv preprint arXiv:1705.03551*, 2017.

[46] C. Clark, K. Lee *et al.*, "Boolq: Exploring the surprising difficulty of natural yes/no questions," in *NAACL*, 2019.

[47] K. Cobbe, V. Kosaraju *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[48] S. Gunasekar, Y. Zhang *et al.*, "Textbooks are all you need," *arXiv preprint arXiv:2306.11644*, 2023.

[49] D. Groeneveld, I. Beltagy *et al.*, "Olmo: Accelerating the science of language models," *arXiv preprint arXiv:2402.00838*, 2024.

[50] S. Lin, J. Hilton, and O. Evans, "Truthfulqa: Measuring how models mimic human falsehoods," *arXiv preprint arXiv:2109.07958*, 2021.

[51] N. Nangia, C. Vania *et al.*, "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*, Nov. 2020.

[52] X. Fu, Z. Zhang *et al.*, "Distributed training of large language models on aws trainium," in *Proceedings of the 2024 ACM Symposium on Cloud Computing*, 2024.

[53] Y. Zhang, Y. Li *et al.*, "Siren's song in the ai ocean: A survey on hallucination in large language models," *arXiv preprint arXiv:2309.01219*, 2023.

[54] J. Devlin, M.-W. Chang *et al.*, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguistics*, 2019.

[55] A. Radford, J. Wu *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[56] Y. Liu, M. Ott *et al.*, "Roberta: A robustly optimized bert pretraining approach," *ArXiv*, vol. abs/1907.11692, 2019.

[57] M. Lewis, Y. Liu *et al.*, "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*, D. Jurafsky, J. Chai *et al.*, Eds. Association for Computational Linguistics, Jul. 2020, pp. 7871–7880.

[58] J. Hoffmann, S. Borgeaud *et al.*, "Training compute-optimal large language models," *arXiv preprint arXiv:2203.15556*, 2022.

[59] R. Anil, A. M. Dai *et al.*, "Palm 2 technical report," *arXiv preprint arXiv:2305.10403*, 2023.

[60] E. Almazrouei, H. Alobeidli *et al.*, "The falcon series of open language models," *arXiv preprint arXiv:2311.16867*, 2023.

[61] G. Team, R. Anil *et al.*, "Gemini: a family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2023.

[62] OpenAI, "Gpt-4 technical report," *ArXiv*, vol. abs/2303.08774, 2023.