Train Models on Cheap Clusters with Low Economic Cost using Block Coordinate Descent

Anonymous ACL submission

Abstract

LLMs training process has high memory demands and high economic cost, making it challenging for many organizations to adopt and scale effectively. In this paper, we train the model using block coordinate descent(BCD) on cheap RTX 4090 clusters, combining with engineering improvements to train LLM with lower economic cost and lower memory demands. In BCD training process, only a subset of parameters is updated, significantly reducing the memory requirements. Through experiments, we show that 1. for a wide range of models and datasets, BCD is capable of training models with the same level of accuracy as traditional method. 2. Averagely, BCD outperforms the OffLoad's 42.0% in training time cost with the same computation resources. BCD matches distributed training speed using just half of resources. 3. BCD training economic cost has been reduced more than 53.6% compared to traditional methods on the 4090 cluster, and more than 74.9% compared to traditional methods on the A100 cluster averagely.

1 Introduction

As models grow in size currently, model training increasingly becomes a bottleneck for performance: as the model expands, the required GPU memory space and economic cost gradually increases. From the view of memory, when using the Adam optimizer for full-parameter training, training a LLM model with parameters W using 1024 tokens requires gradients of size W, optimizer states of size $3W$, and activations of size around $0.7W$. Consequently, approximately $5.7W$ of GPU memory is needed for training. When using re-compute, $5W$ of GPU memory is needed for training for system does not save the activations. From the view of economic cost, training a 20B LLM needs around one million dollers.

The traditional approach to training LLMs involves 3D distributed parallel training. This parallelism method distributes parameters and optimizer states across different GPUs' memories, necessitating constant communication among different GPUs during computation. This practice directly addresses the issue of insufficient memory on a single GPU, enabling the training of large models. However, as the scale of the models increases, the communication overhead also grows. Meanwhile, due to the concurrent consumption of substantial GPU resources, the training economic costs surge accordingly.

Expensive A100/A800 clusters are traditional clusters for LLMs Training. However, a number of cheap GPU clusters with high computational performance, low economic cost but high communication costs and have emerged currently, such as clusters composed of RTX 4090 GPUs or DCU7000s. The comparsion of RTX 4090 GPUs and NVIDIA A100 are shown in table 1. The users want to reduce the economic cost by using cheap clusters, like RTX 4090 based clusters, with limited resources.

Feature	RTX 4090	A100
CUDA Cores	16,384	6,912
Tensor Cores	4th Gen	3rd Gen
Memory	24 GB	40/80 GB
Bandwidth	1,008 GB/s	1,555 GB/s
Memory Bus	384-bit	5,120-bit
FP32 Perf	82.6 TFLOPS	19.5 TFLOPS
Tensorcore FP16	330 TFLOPS	312 TFLOPS
Economic Cost	0.29$/hour	1.20$/hour

Table 1: Comparison of RTX 4090 and NVIDIA A100. RTX 4090's computational performance and economic cost are better then A100, but not suitable for LLM training. LLMs are trained in A100 for its large memory and high bandwidth but the economic cost is high.

However, there is no mature algorithm for how to use these cheap clusters to train large models in a low economic cost and limited resources manner. The traditional method for training models with limited resources is mainly the OffLoad method

in DeepSpeed. Although the OffLoad method can complete model training process with fewer computational resources, the constant migration of parameters, optimizer states, activations, and gradients between GPU memory and system memory during iterations results in a significant drop in training performance which significantly increases the time consumption and does not guarantee a lower economic cost compared to distributed methods.

In this paper, we will demonstrate how to use the block coordinate descent (BCD), combined with engineering improvements, to train large models on cheap clusters at a lower economic cost. BCD updates only a portion of the parameters in each iteration. The remaining parameters do not need to record optimizer states or gradient information, and some parameters do not need to store activation information. As a result, the memory required for training the model is significantly reduced. Specifically, if we update only 1/3 of the parameters of a full model with a volume of W in each iteration, we will use less than 50% of the memory required by the full model with re-computing setting. Compared with distributed training, the variation in training time required by BCD is relatively small, and thus the training economic costs have also been reduced significantly.

In our experiments, we will show that 1. BCD can train models that perform as well as traditional methods. 2. The time cost of BCD is close to that of distributed methods but with less computation resources, and outperforms the OffLoad method when using the same resources. 3. The economic cost of BCD on the RTX 4090 platform is lower than that of distributed methods on the 4090, A100, and A800 platforms.

2 Related Work

2.1 Block Coordinate Descent

BCD is a highly mature non-gradient descent method and has been extensively studied and analyzed(Tseng, 2001; Beck and Tetruashvili, 2013; Wright, 2015; Richtárik and Takáč, 2014; Cai et al., 2023; Nutini et al., 2022; Tu et al., 2016). Recently, some researches discuss how to adopt BCD in to training DNN(Zisselman et al., 2019; Zhao et al., 2014; Blondel et al., 2013; Wu et al., 2021; Damaskinos et al., 2021). The convergence of BCD on DNN is proved by the work (Zeng et al., 2019; Zhang and Brand, 2017). In some works(Zeng et al., 2019; Lau et al., 2018), they claim that in some cases, BCD can produce a better performance model compared with traditional optimizers. Compared to training with all parameters, BCD takes more rounds, but it uses less memory and computing power. There is no work discuss how to use BCD on LLM training and its advantages on economic cost.

2.2 LLM Training

2.2.1 Distributed Parallel Training

Large-scale parallel training has become the mainstream approach for training large models(Narayanan et al., 2021; Lai et al., 2023). The distributed training of large models primarily relies on 3D parallelism. Large model 3D parallelism is an efficient model training method, with each parameter, optimizer state, and so on, mapped to individual GPUs. 3D parallelism technology is widely used in various training frameworks, such as DeepSpeed(Holmes et al., 2024; Aminabadi et al., 2022; Rajbhandari et al., 2022) and Megatron.

2.2.2 OffLoad Training

The OffLoad(Rajbhandari et al., 2021; Narayanan et al., 2021; Aminabadi et al., 2022) mode is a technique that dynamically transfers model parameters, gradients, and optimizer states from the GPU to the CPU or other storage devices to reduce GPU memory usage. It enables training large models on GPUs with limited memory. While this approach significantly improves the feasibility of training large models, but OffLoad leads to slower training speeds. The OffLoad mode is widely used in different framework, especially the the limited resources cases.(Gao et al., 2024; Zhang et al., 2024; Athlur et al., 2022; Lv et al., 2023).

3 Block Coordinate Descent

3.1 Adopt BCD into LLM Training

The block coordinate descent method advances the training process by reducing memory requirements during training.

Take single-GPU training as an example: when a single GPU can fully store the model, the amount of parameters participating in training per iteration can be adjusted based on the remaining available memory of the GPU. In extreme cases, as long as there is enough memory to train a single parameter, training can proceed. For instance, with an RTX 4090 (24GB), it is theoretically possible to train

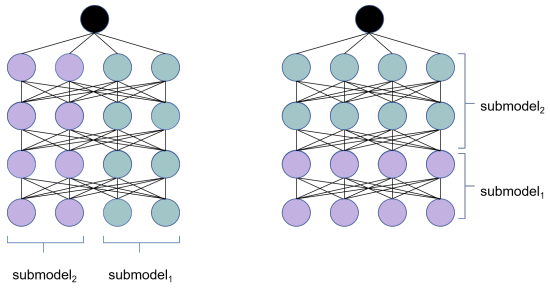

Figure 1: When we partition the model, slicing it by layers allows for better utilization of optimized computation kernels (as shown in the right figure). If a portion of the parameters in each layer are updated simultaneously (as shown in the left figure), computational performance decreases, and pre-inference cannot be used to accelerate the forward process.

a 12B Large Language Model. However, training model under such extreme conditions incurs very high time costs. The standard BCD method imposes no mathematical requirements on parameter partitioning, freezing any parameters satisfies the algorithm's needs. In practical, the choice of which parameters to freeze must consider the computational performance benefits provided by the architecture and how to reduce memory usage during computation. Computation processes can be viewed as combinations of multiple optimized computational kernel to complete tasks. Thus, partitioning parameters based on operators is the most reasonable approach to freezing parameters.

In neural networks, most operators are contained entirely within the "layer" data structure. Therefore, freezing and unfreezing parameters by layers or by data structures based on layers, such as blocks in ResNet networks, is the most appropriate approach, which shown in figure 1. This maximizes the performance benefits of optimized computation kernel. Consequently, we have adapted the general BCD method to the specific needs of deep learning, resulting in the following algorithm 1.

As illustrated in the algorithm, each iteration trains the current parameters to a local convergence. During this process, traditional optimization algorithms, such as SGD and AdamW, are used to iteratively minimize along the given direction. Regarding the choice of optimization algorithm, there is no fundamental difference, as all algorithms can drive the process forward. However, in practice, SGD often achieves better model results compared to Adam or AdamW.

Algorithm 1 Using BCD Training LLM

Input: DeepLearning Model $model$, Traning Dataset $\mathcal{D}$,and convergence criterion.
Initialize: Set $k = 0$ and Split $model$ into $\{submodel_1, submodel_2, ..., submodel_M\}$.
repeat
  Select a submodel $submodel_{i_k}, i_k \in \{1, 2, \ldots, M\}$ cyclically.
  Freezing the parameters in $submodel_j, j \neq i_k$
  Unfreezing the parameters in $submodel_{i_k}$
  Building optimizer of $submodel_{i_k}$. The opitmizer can be SGD or Adam.
  Training the $model$ on $\mathcal{D}$ until converged. Only the parameters in $submodel_{i_k}$ is updated.
  Increment $k \leftarrow k + 1$.
until Convergence criterion is satisfied.
Output: Converged model $model_k$.

3.2 Engineering Improvements

3.2.1 Parallel Training

When training with BCD in a multi-GPU environment, it is important to ensure that different computing hardware can fully utilize their computational resources. This means that the computing hardware should be able to maintain the busy state. To achieve above target, both frozen and unfrozen parameter parts should be stored simultaneously on the same hardware. Common multi-GPU parallelism methods include data parallelism, tensor parallelism, and pipeline parallelism. Among them, tensor parallelism and pipeline parallelism are model parallelism approaches, which require further adaptation for BCD.

Typically, freezing and unfreezing parameters within a matrix multiplication operation can significantly impact the performance of the computation kernel. When performing tensor parallelism, only part of the parameters are activated in the matrix multiplication, which results in a loss of spatial locality in the computation. Current framework computation kernel optimizations are based on layer-based neural network structures for computational performance. What is more, the tenor parallel manner would incur the need of communication, which would increase the system burden.

On the other hand, pipeline parallelism is designed based on layers, which shown in figure 2. So after freezing, it has a performance advantage at

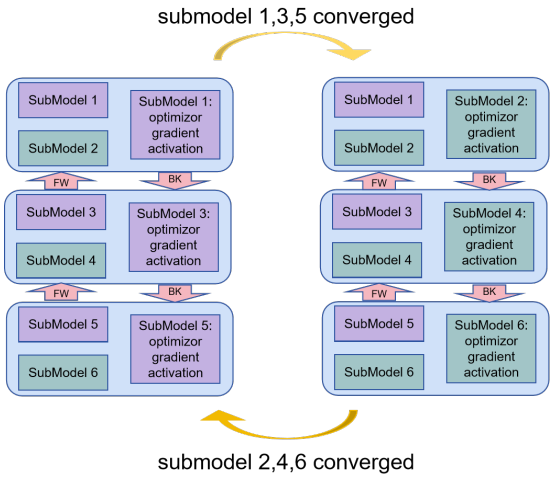

Figure 2: Training a large model one three GPUs. The model is divided into six submodels. Two submodels are addressed into one GPU. Different GPUs are paralleled via pipeline parallel manner. The light blue part is GPU memory.

the computation kernel level. Therefore, pipeline parallelism is a more suitable multi-GPU parallel approach for training models with BCD.

3.2.2 Pre-inference

When the model is sufficiently split, most of the model parameters are only involved in the inference process and in backward process, they are freezed. Meanwhile, due to the increased number of model iteration rounds, although the computational cost of backward process is reduced, the cost of forward process remains unchanged. Therefore, in BCD methods, reducing the time required for inference is necessary to make it comparable in performance to traditional full-parameter training methods.

When performing model training on clusters with high computational performance but low communication performance (e.g., clusters composed of RTX 4090 GPUs or DCUs), we observe that these clusters exhibit excessive computational capacity, leaving a significant portion of the computational resources idle. However, large-scale utilization of these resources can quickly lead to increased costs related to communication, data migration, and cluster stability. How to effectively utilize such types of cluster resources is a problem that requires further consideration.

To address above problems, we propose a pre-inference approach in the context of model training using BCD methods to fully utilize these resources. Pre-inference refers to the situation where, when the training of submodels closer to the label, the parameters in the submodels near the data input remain unchanged. Therefore, these submodels near the data input can be treated as a complete model. In this model, all data is inferred to produce a new dataset, which is then used to train and update the parameters of the submodels, shown in figure 3.

Pre-inference is primarily aimed at accelerating computation in high-performance, high-communication-cost cluster environments. In such environments, all computational resources can be utilized for submodel inference tasks to construct new datasets, significantly reducing the overall time consumed by inference. When computational resources with high performance and high communication costs are unlimited, the inference cost of fixed submodel parts becomes a fixed time cost. This means that the time to infer a single sample is equal to the sum of the inference time and the cost of a single all-reduce operation.

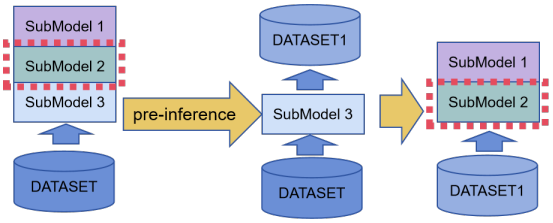

Figure 3: The process of pre-inference. We Train submodel 2 in this case. The parameters in submodel 3 is fixed. We can inference the $Dataset$ on submodel 3 and gain $Dataset1$ on high communication cost cluster on AllReduce manner. Then Using $Dataset1$ to train the model composed by submode 1 and submodel 2. The parameters in submodel 2 is trainable.

3.2.3 Reducing Training Dataset

According to the scaling law, there exists a certain multiplier relationship among model size, data volume, and computational load. Therefore, when the model size is reduced, the amount of training data required can be moderately decreased, to achieve a balance between computational efficiency and model performance.

The process of training using the BCD method can be regarded as training a smaller model. Scaling law suggests that the amount of data needed to train this smaller model is not the same as that required for the full model. Furthermore, based on our needs, we can reduce the total amount of data when training submodels, thus reducing inference costs and improving computational speed.

To utilize the entire dataset as much as possible, the BCD method can adopt a sampling approach

for training. Sufficient subsets of data are sampled from the total training dataset, and the submodel is trained on these sampled subsets. When training a new submodel, a new subset of data is used. This approach ensures that the entire training dataset is fully utilized while minimizing the computational burden on model training caused by large datasets.

4 Experiments

We conducted three experiments to demonstrate the superiority of BCD: 1. In the Validity Experiment, BCD was able to train models with performance equivalent to that of full-parameter training. Additionally, in this experiment, we also estimated the number of iterations required by BCD as a multiple of those needed for full-parameter training, and we name this multiple as B-F multiplier. 2. In the Performance Experiment, we compared BCD's time performance per training iteration with that of OffLoad and distributed training methods. Furthermore, we estimated and compared the total time consumption based on B-F multiplier. 3. Based on the rental costs, Validity Experiment and Performance Experiment results obtained on the GTX 4090, A100, and A800 platforms, we analyzed the training economic costs.

4.1 Experiment Setting

The experiments were conducted cluster which consist of a node with 8 A100, a node with 8 A800 and 4 nodes with 8 RTX 4090. All servers were interconnected using a 2×25Gbps high-speed network.

In Validity Experiments, we use ResNet 8, ResNet 14 on cifar 10, cifar 100 datasets and 0.15B GPT 2 on wikitext, Webtext2 dataset.

We use 1.6B GPT2 (abbr. G-1.6B), 5.4B GPT2 (abbr. G-5.4B), 10B GPT2 (abbr. G-10B), 20B-GPT2 (abbr. G-20B), 1.5B LLama (abbr. L-1.5B), 6B LLama (abbr. L-6B), 11B LLama (abbr. L-11B), 23B LLama (abbr. L-23B) on WebText2 in Performance and Economic Cost Experiments.

In our experiments, to ensure clarity and consistency, we divided the model into three submodels for all BCD experiments. This implies that during each iteration, we only update one-third of the full parameter set. Additional experiments detailing various BCD settings are presented in the Appendix for further reference.

4.2 Validity Experiments

4.2.1 Algorithms Setting

The experiment includes ResNet series models and a GPT-2 model with 0.15B parameters. The ResNet models were trained on the CIFAR-10 dataset, while the GPT-2 model was trained on the Wikitext and WebText2 datasets. For the optimizer settings, the SGD optimizer is set as learning rate of 0.1, a momentum of 0.9, and a weight decay of 1e-5. The Adam optimizer was set with a learning rate of 1e-4, a weight decay of 1e-5, and default values for the first and second moment estimates.

4.2.2 Experimental Results

In this experiment, we employed two optimizers, Adam and SGD, and trained the models using both the BCD method and full-parameter updating method. The results are shown in table 2. Overall, we observe that BCD-based training demonstrates certain advantages. From the perspective of the loss function, the performance of models trained with BCD is superior to those trained with full-parameter updating in most cases. This phenomenon has been explained by the work (Zeng et al., 2019). The difference in loss function values across different models and datasets is less than 0.1 and the performance gap between the model of BCD and model of full parameters is samll.

4.2.3 B-F multiplier

Based on the experimental results, to gain the same performance modes, in Adam experiments, we found that the average number of training epochs required by the BCD method is 1.39 times that of full parameters Adam methods, with the worst-case scenario being 2.77 times. We set the Adam's average B-F (BCD-Full multiplier) multiplier as 1.39 and Adam's the worst B-F multiplier as 2.77.

4.3 Performance Experiments

In this experiment, we will compare the performance of the BCD method, the OffLoad method, and distributed training methods in one iteration. Additionally, we will provide the time cost required to fully train the models using these different methods, based on the B-F multiplier. In this part, we only use Adam optimizer to train LLM models for Adam is the main stream LLM optimizer and OffLoad only provides optimized Adam optimizer.

Model	SGD			BCD-SGD			Adam			BCD-Adam		
	epoch	Acc	loss	epoch	Acc	loss	epoch	Acc	loss	epoch	Acc	loss
ResNet8	639	86.9	0.1671	**395**	**87.2**	0.1783	389	83.1	0.3955	**309**	**86.0**	**0.2279**
ResNet14	323	88.9	0.0708	391	**89.0**	**0.0613**	491	85.7	0.0969	**372**	**87.6**	**0.0527**
ResNet20	191	88.7	0.0574	322	88.7	**0.0263**	768	86.3	0.0269	**423**	**88.3**	**0.0222**
ResNet50	273	90.4	0.0260	**251**	88.9	**0.0120**	/	/	/	/	/	/
ResNet101	286	70.9	0.0691	376	70.1	**0.0411**	/	/	/	/	/	/
	iter	PP	loss	iter	PP	loss	iter	PP	loss	iter	PP	loss
GPT2-wiki	111587	6.9639	1.9407	204575	**6.9189**	**1.9343**	83690	6.5988	1.8869	232471	6.6263	1.8910
GPT2-web	1045000	28.733	3.3580	2770000	31.285	3.4430	499000	28.735	3.3580	1045000	30.712	3.4240

Table 2: The experimental results BCD method and full parameters training with SGD and Adam optimizer.

4.3.1 Algorithm Setting

We endeavor to compare different methods using identical hardware and model configurations to the extent possible. Specifically, for the BCD and OffLoad methods, we employ the same hardware setup for computation and comparison. On the other hand, for the distributed training method and parallel BCD algorithm, we utilize the minimum configuration capable of supporting the training of the model for training and comparison purposes. The code of OffLoad and distributed experiments are from Megatron-LM. The BCD code is also modified from Megatron-LM.

OffLoad Setting In the experiments, the offload configuration was set to enable stage 1, offloading the optimizer state to the CPU with the device set to "cpu" and pin memory enabled as true. Additionally, communication and computation overlap was enabled, along with contiguous gradient storage.

Distributed Setting In the multi-node environment, the experiments were extended to multiple nodes. The multi-node configurations used in the experiments included two nodes with 8 GPUs, two nodes with 16 GPUs, four nodes with 16 GPUs, and four nodes with 32 GPUs. The code configuration utilized the pipeline-model-parallel-size parameter, with the PP-SIZE setting to specify the scale of pipeline parallelism, allowing for flexible adjustment of the parallel granularity to accommodate various hardware configurations.

BCD Setting Due to the setting of our BCD experiments, where only one-third of the full parameter set is updated at a time, the memory cost and the number of GPUs required for BCD are half of those needed for distributed methods. In a parallel BCD setting, we utilize the same configuration as the distributed setting for a fair assessment.

Model	#GPU	Offload-Adam	BCD-Adam
G-1.6B	1	2403 ms	**378 ms**
G-5.4B	4	4373 ms	**1211 ms**
G-10B	8	7076 ms	**2414 ms**
L-1.5B	1	4824 ms	**415 ms**
L-6.0B	4	6367 ms	**941 ms**
L-11B	8	7292 ms	**1807 ms**

Table 3: The time comparison between OffLoad and BCD in one iteration.

Model	Distributed			BCD-Adam		
	N/G	Mem	time	N/G	Mem	time
G-1.6B	1/2	24.077	576 ms	**1/1**	**12.038**	**378 ms**
G-5.4B	2/4	81.881	1934 ms	**1/4**	**40.942**	**1211 ms**
G-10B	4/4	160.738	3178 ms	**1/8**	**80.370**	**2414 ms**
G-10B	2/8	160.738	3252 ms	**1/8**	**80.370**	**2414 ms**
G-20B	4/8	312.805	6378 ms	**2/8**	**156.402**	**3606 ms**
L-1.5B	1/2	24.632	423 ms	**1/1**	**11.662**	**415 ms**
L-6.0B	2/4	89.815	1287 ms	**1/4**	**44.908**	**941 ms**
L-8.0B	2/6	120.327	1651 ms	**1/6**	**60.163**	**1197 ms**
L-11B	4/4	175.714	2511 ms	**1/8**	**87.858**	**1807 ms**
L-23B	4/8	347.539	4908 ms	**2/8**	**173.769**	**3479 ms**

Table 4: The time comparison between distributed method and BCD in one iteration.N/G means the number of node and the number of GPU per nodes. Mem is measured in GB (gigabytes).

4.3.2 Experimental Result

The comparsion between OffLoad and BCD Based on the experimental results, table 3, there is a great performance difference between the OffLoad and BCD methods on the GPT-2 and LLaMA models, particularly in terms of time efficiency and scalability, where each method exhibits distinct characteristics. Our method outperforms the OffLoad method in both performance and scalability.

Compared with OffLoad, the BCD method avoids frequent communication between devices by freezing some parameters in GPU memory and only updating the parts that need to be computed.

This design not only significantly reduces communication overhead but also allows for more computational resources to be retained in memory for the actual training tasks. So, the BCD method exhibits better scalability in multi-GPU environments.

It is noteworthy that, even with the offload mode enabled, the current OffLoad is unable to fully leverage GPU resources for training models larger than 10B in a pipeline manner. While using the offload stage3 mode could theoretically enable data parallelism for >10B larger-scale models, officially, such an interface is not provided. Additionally, due to the involvement of cross-node training, this mode on >10B model would be slower.

The comparsion between Distributed method and BCD Based on the experimental results, for the G-1.6B model, the iteration time of the BCD method on a single GPU is 378.49 ms, while the iteration time for traditional distributed single-node 2-GPU training is 576 ms, resulting in a performance improvement of approximately 1.52 times. For larger models, such as G-10B, the iteration time of the BCD method on eight GPUs is 2414.32 ms, while the iteration times for traditional distributed four-node 16-GPU and two-node 16-GPU configurations are 3178 ms and 3252 ms, respectively, showing improvements of 1.32 times and 1.35 times. For even larger models, such as G-20B, the iteration time of the BCD method on a two-node 16-GPU configuration is approximately 1.76 times faster than that of the traditional distributed four-node 32-GPU configuration.

From all the experimental results, the performance improvement in iteration time for the BCD method ranges from 1.52 times to 1.76 times, with an average improvement of approximately 1.41 times. This result indicates that the BCD method not only performs exceptionally well on smaller models but also shows significant performance advantages in the distributed training of larger models, requiring fewer GPU hours per iteration.

4.3.3 Full Training Analysis

The total training time can be calculated by multiplying the number of iterations by the iteration time per round. Considering that full-parameter training for 10B and 20B models requires an economic cost close to $100,000, which is difficult to bear, we estimate the overall training time using the Adam's average B-F multiplier and the worst B-F multiplier. As described in the B-F multiplier section, the Adam's average B-F multiplier is 1.39.

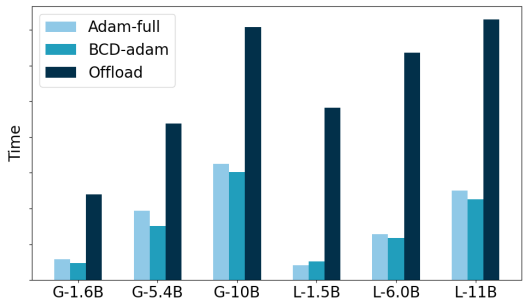

Figure 4: Training time of GPT and LLaMA models using different methods at various model scales.

Therefore, for BCD-Adam, the ratio of the one iteration training time multiplied by 1.39 to the one iteration training time of the full-parameter Adam training method (including OffLoad and distributed methods) can be considered as the ratio of the training time for BCD-based models to that for full-parameter models. The results are shown in the figure 4, showing that averagely, BCD outperforms the OffLoad's 42.0% in training time cost with the same computation resources. BCD matches distributed training speed using just half of resources. For the worst-case scenario, we replace the average B-F multiplier in the above calculation with the the worst B-F multiplier, which is 2.77. This leads to the conclusion that, in the worst-case scenario, the training speed of BCD-Adam, using half computational resources, is 48.9% slower than distributed methods but more than 112.5% faster than OffLoad methods using equivalent resources averagely.

4.4 Economic Cost Experiments

In this section, experiments were conducted using GPT-2 models of various parameter scales. Through training economic cost analysis, we demonstrated that our proposed method offers significant economic cost advantages compared to traditional methods.

4.4.1 The Rental Costs of GPUs

Currently, the market price ratio of 4090, A800, and A100 is approximately 1:2.5:4, with more detailed pricing information provided in the appendix. In this section, we refer to the rental costs from Wuwen Xingqiong Intelligent Technology Co., Ltd., which are as follows: $0.29/hour for a single RTX 4090 GPU, $1.20/hour for a single A100 GPU, and $0.69/hour for a single A800 GPU.

The total training cost is the product of the number of training rounds, the training time per round,

the number of GPUs used, and the rental cost. The GPU hour details on different platform can be found in the appendix.

4.4.2 Economic Cost Analysis

	G-1.6B		G-5.4B		G-10B		G-20B	
	N/G	cost ↓	N/G	cost ↓	N/G	cost ↓	N/G	cost ↓
Full Update	1/2	0%	1/8	0%	2/8	0%	4/8	0%
BCD(The worst)	1/1	**56.5%**	1/4	**30.1%**	1/8	-2.8%	2/8	**29.4%**
BCD(Average)	1/1	**80.4%**	1/4	**68.5%**	1/8	**53.6%**	2/8	**68.3%**

Table 5: The training cost reduction of BCD compared to full-parameter updates across different models on RTX 4090 cluster. N/G means the number of node and the number of GPU per nodes.

	G-1.6B		G-5.4B		G-10B		G-20B	
	N/G	cost ↓	N/G	cost ↓	N/G	cost ↓	N/G	cost ↓
Full Update	**1/1**	0%	**1/2**	0%	**1/4**	0%	**1/8**	0%
BCD(The worst)	1/1	**75.1%**	1/4	**45.6%**	1/8	**44.4%**	2/8	**57.3%**
BCD(Average)	1/1	**88.8%**	1/4	**75.5%**	1/8	**74.9%**	2/8	**80.8%**

Table 6: The training cost reduction of BCD on 4090 clusters compared to full-parameter updates on A100 cluster across different models. N/G means the number of node and the number of GPU per nodes.

	G-1.6B		G-5.4B		G-10B		G-20B	
	N/G	cost ↓	N/G	cost ↓	N/G	cost ↓	N/G	cost ↓
Full Update	**1/1**	0%	**1/2**	0%	**1/4**	0%	**1/8**	0%
BCD(The worst)	1/1	**61.1%**	1/4	**9.9%**	1/8	**7.0%**	2/8	**28.5%**
BCD(Average)	1/1	**82.5%**	1/4	**59.4%**	1/8	**58.1%**	2/8	**67.7%**

Table 7: The training cost reduction of BCD on 4090 clusters compared to full-parameter updates on A800 cluster across different models. N/G means the number of node and the number of GPU per nodes.

In tabel 5, we can observe that, on average, the BCD algorithm takes a time close to (or even slightly better than) that required for full-parameter Adam training. However, considering that the BCD method utilizes only half of the number of GPUs, the training cost of BCD is less than half of that of full-parameter training. In the worst-case scenario for BCD, the time spent by BCD is less than double that of full-parameter training, but the economic cost is reduced by half. Therefore, the BCD algorithm has a significant advantage in economic cost compared to full-parameter training on a 4090 cluster. Additionally, the GPU Hours of the BCD method are also reduced, showing a decreasing trend across all configurations compared to full-parameter training.

On the A100 GPU, the per-iteration training time for the GPT-2 model with parameter sizes of 1.6B, 5.4B, 10B, and 20B is 1064 ms, 2952 ms, 5772 ms, and 11222 ms, respectively. On the A800 GPU, the per-iteration training time for the GPT-2 model with parameter sizes of 1.6B, 5.4B, 10B, and 20B is 1122 ms, 3126 ms, 6042 ms, and 11722 ms, respectively.

Table 6 and table 7 show the economic cost reductions between full parameters updating on A100 and A800 cluster. They show that both on average and in the worst-case scenarios, BCD training on 4090 GPUs results in substantial economic cost reductions compared to full-parameter training on A100/800 GPUs.

Since RTX 4090 and A800 GPUs are interconnected using PCIe, which has limited bandwidth, the communication cost increases rapidly as the model size grows. Therefore, the BCD method, which requires less bandwidth, results in a larger economic cost reduction in larger model training cases. In the 1.6B cases, half of the GPU memory of A100 and A800 remains unused, leading to a waste of resources. As a result, the BCD method achieves a significant economic cost reduction.

5 Discussion

BCD is capable of training with fewer resources while keeping the training time controllable, resulting in significant cost reductions. This enables BCD to utilize older-generation GPU platforms that have been phased out for training scenarios (such as V100) and non-top-tier GPU platforms of the current era (like 4090) for training purposes. Additionally, it lowers the economic cost barrier for the development of large-scale models. Furthermore, BCD enables platforms of current scale to challenge larger models.

6 Conclusion

In this paper, we demonstrate using block coordinate descent (BCD) with engineering improvements to train large models cheaply on high performance, high communication cost and low economic cost clusters. BCD updates a portion of parameters per iteration, reducing memory needs. Experiments show: 1) BCD trains models as well as traditional methods. 2) BCD's time cost nears distributed methods with fewer resources, outperforming OffLoad with same resources. 3) BCD uses less money on RTX 4090 than distributed training on 4090, A100, and A800 platforms.

Limitations

Despite the significant advantages of our method in training large-scale models on low-cost clusters, several limitations remain:1)Although BCD reduces training costs and memory usage, its sequential parameter updates may introduce additional computational overhead. Future work could explore optimizing its parallelization strategy to improve training efficiency. 2)Due to financial constraints, we have not conducted full-scale training on ultra-large models (e.g., 100B-parameter models), and its applicability at this scale remains to be verified. 3)The experimental setup for BCD was limited to a two-node, 16-GPU (RTX 4090) configuration, without extension to larger-scale clusters. Therefore, its scalability in settings beyond four-node, 32-GPU configurations still requires further investigation, and its adaptability to more powerful computing resources, such as A100 or H100 GPUs, warrants further exploration.

References

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. 2022. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE.

Sanjith Athlur, Nitika Saran, Muthian Sivathanu, Ramachandran Ramjee, and Nipun Kwatra. 2022. Varuna: scalable, low-cost training of massive deep learning models. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 472–487.

Amir Beck and Luba Tetruashvili. 2013. On the convergence of block coordinate descent type methods. *SIAM journal on Optimization*, 23(4):2037–2060.

Mathieu Blondel, Kazuhiro Seki, and Kuniaki Uehara. 2013. Block coordinate descent algorithms for large-scale sparse multiclass classification. *Machine learning*, 93:31–52.

Xufeng Cai, Chaobing Song, Stephen Wright, and Jelena Diakonikolas. 2023. Cyclic block coordinate descent with variance reduction for composite nonconvex optimization. In *International Conference on Machine Learning*, pages 3469–3494. PMLR.

Georgios Damaskinos, Celestine Mendler-Dünner, Rachid Guerraoui, Nikolaos Papandreou, and Thomas Parnell. 2021. Differentially private stochastic coordinate descent. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7176–7184.

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. 2024. {Cost-Efficient} large language model serving for multi-turn conversations with {CachedAttention}. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 111–126.

Connor Holmes, Masahiro Tanaka, Michael Wyatt, Ammar Ahmad Awan, Jeff Rasley, Samyam Rajbhandari, Reza Yazdani Aminabadi, Heyang Qin, Arash Bakhtiari, Lev Kurilenko, et al. 2024. Deepspeed-fastgen: High-throughput text generation for llms via mii and deepspeed-inference. *arXiv preprint arXiv:2401.08671*.

Zhiquan Lai, Shengwei Li, Xudong Tang, Keshi Ge, Weijie Liu, Yabo Duan, Linbo Qiao, and Dongsheng Li. 2023. Merak: An efficient distributed dnn training framework with automated 3d parallelism for giant foundation models. *IEEE Transactions on Parallel and Distributed Systems*, 34(5):1466–1478.

Tim Tsz-Kit Lau, Jinshan Zeng, Baoyuan Wu, and Yuan Yao. 2018. A proximal block coordinate descent algorithm for deep neural network training. *arXiv preprint arXiv:1803.09082*.

Kai Lv, Yuqing Yang, Tengxiao Liu, Qinghui Gao, Qipeng Guo, and Xipeng Qiu. 2023. Full parameter fine-tuning for large language models with limited resources. *arXiv preprint arXiv:2306.09782*.

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

Julie Nutini, Issam Laradji, and Mark Schmidt. 2022. Let's make block coordinate descent converge faster: faster greedy rules, message-passing, active-set complexity, and superlinear convergence. *Journal of Machine Learning Research*, 23(131):1–74.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. 2022. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International conference on machine learning*, pages 18332–18346. PMLR.

Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. 2021. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–14.

Peter Richtárik and Martin Takáč. 2014. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1):1–38.

Paul Tseng. 2001. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications*, 109:475–494.

Stephen Tu, Rebecca Roelofs, Shivaram Venkataraman, and Benjamin Recht. 2016. Large scale kernel learning using block coordinate descent. *arXiv preprint arXiv:1602.05310*.

Stephen J Wright. 2015. Coordinate descent algorithms. *Mathematical programming*, 151(1):3–34.

Ruiyuan Wu, Anna Scaglione, HoiTo Wai, Nurullah Karakoc, Kari Hreinsson, and Wing-Kin Ma. 2021. Federated block coordinate descent scheme for learning global and personalized models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10355–10362.

Jinshan Zeng, Tim Tsz-Kit Lau, Shaobo Lin, and Yuan Yao. 2019. Global convergence of block coordinate descent in deep learning. In *International conference on machine learning*, pages 7313–7323. PMLR.

Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun. 2024. Adam-mini: Use fewer learning rates to gain more. *arXiv preprint arXiv:2406.16793*.

Ziming Zhang and Matthew Brand. 2017. Convergent block coordinate descent for training tikhonov regularized deep neural networks. *Advances in Neural Information Processing Systems*, 30.

Tuo Zhao, Mo Yu, Yiming Wang, Raman Arora, and Han Liu. 2014. Accelerated mini-batch randomized block coordinate descent method. *Advances in neural information processing systems*, 27.

Ev Zisselman, Jeremias Sulam, and Michael Elad. 2019. A local block coordinate descent algorithm for the csc model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8208–8217.

A Experiments

A.1 Model Setting

The parameter scales, configurations, and memory usage of the two models are shown in table 8. To ensure the fairness and comparability of the results, we strictly kept the model hyperparameters consistent across different experimental setups, including the learning rate, batch size, and optimizer (Adam). In terms of computational precision, all experiments were conducted using FP32 training to avoid the introduction of additional variables from mixed-precision or lower-precision computations that could affect the results.

Model	#GPU	L	H	A	Mem
G-1.6B	1	30	2048	16	24.077
G-5.4B	4	28	3968	32	81.881
G-10B	8	56	3968	32	160.738
G-20B	16	110	3968	32	312.805
L-1.5B	1	30	2048	16	24.632
L-6.0B	4	32	4096	32	89.815
L-11B	8	64	4096	32	175.714
L-23B	16	128	4096	32	347.539

Table 8: In the table, L represents the number of layers in the Transformer model, H denotes the dimension of the hidden layers, and A stands for the number of attention heads. Mem is measured in GB (gigabytes). Additionally, the L-1.5B model has a feed-forward network (FFN) size of 5050, while other LLaMA models have an FFN size of 9200.

A.2 Dataset compress Experiments

This part will show the impact of reducing the training dataset size to the reduced training scale per epoch in BCD. According to the Scaling Law, to achieve optimal model performance, there should be a proportional relationship among computational resources, training dataset size, and model size. Based on the Scaling Law, when training each submodel, as the scale of the submodel decreases, the corresponding training dataset resources should also be reduced accordingly. This experiment is conducted to test this hypothesis.

A.2.1 Model Setting

We conducted this experiments on ResNet 14 and GPT 2 model for its faster convergence speed. All model settings are the same with the setting with ResNets experiments and LLM experiments.

A.2.2 Algorithm Setting

We conducted this experiments on BCD with SGD and Adam optimizer and full parameters SGD and Adam optimizer. All algorithms setting are the same with the ResNets experiments and LLM experiments.

A.2.3 Dataset Setting

We use cifar 10 and Wikitext dataset as benchmark. In each epoch, we re-sampling 90% dataset in all training dataset. We also use all training dataset results as the benchmarks.

A.2.4 Results

The results are shown in table 9 and 10. The difference in tables is the dataset sampling rate, optimizer.

#SR	#UFP	Opt	#epoch	Acc	Loss	Mem
0.9	1/3	SGD	40	68.9%	1.9312	1013.58MB
1	1/3	SGD	44	69.1%	1.9343	1013.58MB
1	1	SGD	24	69.6%	1.9407	1824.45MB
0.9	1/3	Adam	60	67.5%	1.9110	1216.30MB
1	1/3	Adam	50	66.2%	1.8910	1216.30MB
1	1	Adam	18	65.9%	1.8869	2432.60MB

Table 9: The GPT 2 model performance with different training dataset size and different training methods.

#SR	#UFP	#epoch	Acc	Loss	Mem
0.9	1/3	427	88.7%	0.0616	1.12MB
1	1/3	391	89.0%	0.0613	1.12MB
1	1	323	88.9%	0.0708	2.01MB

Table 10: The ResNet 14 model performance with different training dataset size and different training methods.

A.2.5 Analysis

From the experimental results presented in table 9 and table 10, it can be observed that reducing the training dataset size by 10% has a minimal impact on model performance under the BCD framework. Compared to training with the full dataset, sampling 90% of the data leads to only a slight decrease in final accuracy. For example, in the GPT-2 training experiments, when using the SGD optimizer, the accuracy of the model trained with the full dataset is 69.1%, whereas the accuracy of the model trained with 90% of the sampled data is 68.9%, showing only a 0.2% reduction. Under the Adam optimizer, the model trained with 90% of the sampled data even achieves a slightly higher accuracy than the one trained with the full dataset (67.5% vs. 66.2%), suggesting that in certain cases, reducing the training dataset size may contribute to improved optimizer stability during convergence.

For the ResNet-14 model, reducing the training dataset by 10% results in only a 0.3% accuracy drop, from 89.0% to 88.7%, while the loss value remains nearly the same (0.0613 vs. 0.0616). This indicates that the data sampling strategy has a similarly negligible effect on convolutional neural networks. Additionally, the results show that models trained on the full dataset tend to converge in fewer epochs. This suggests that reducing the training dataset may slightly affect the convergence speed, but its overall impact on final model performance remains limited.

A.3 Unfrozen Parts Experiments

#UFP	Opt	#epoch	Acc	Loss	Mem
1	SGD	24	69.6%	1.9407	1824.45MB
1/2	SGD	40	67.1%	1.9033	1216.30MB
1/3	SGD	44	69.1%	1.9343	1013.58MB
1/4	SGD	50	67.5%	1.9109	912.23MB
1	Adam	18	65.9%	1.8869	2432.60MB
1/2	Adam	40	66.3%	1.8924	1520.38MB
1/3	Adam	50	66.2%	1.8910	1216.30MB
1/4	Adam	60	66.8%	1.8995	1064.26MB

Table 11: The performance of GPT2 models with different scales of frozen parameters under various optimizers.

Table 11 shows that reducing the number of UFP significantly decreases memory consumption. For instance, in SGD training, memory usage drops from 1824.45MB to 912.23MB, while in Adam training, it decreases from 2432.60MB to 1064.26MB. Meanwhile, the model accuracy experiences only a slight reduction, with a maximum drop of 2.1% under SGD, and even a slight improvement under Adam. In terms of training time, models with more frozen parameters generally require more epochs to converge. Therefore, in large-scale training, it is essential to balance computational resources, convergence speed, and final performance to determine an appropriate freezing strategy.

A.4 Market Price Statistics for GPUs

According to publicly available information from various computing rental service providers, AutoDL Cloud Computing offers RTX 4090 at $0.31/hour and A800 80GB at $0.88/hour. Parallel Technology provides A100 80GB at $2.25/hour and RTX 4090 at $0.50/hour. AnyGPU offers RTX 4090 at $0.40/hour. AI Galaxy Cloud lists RTX 4090 at $0.19/hour and A100 80GB at $1.12/hour. Hengyuan Cloud provides A100 80GB at $1.39/hour, A800 80GB at $1.25/hour, and RTX 4090 at $0.22/hour. It should be noted that rental prices may vary based on market supply and demand, rental duration, and other factors, with the official platform information serving as the definitive reference.

	G-1.6B	G-5.4B	G-10B	G-20B
	GPUh ↓	GPUh ↓	GPUh ↓	GPUh ↓
Full Update	0%	0%	0%	0%
BCD(The worst)	56.1%	13.3%	-2.8%	29.4%
BCD(Average)	80.2%	60.9%	53.6%	68.2%

Table 12: Reduction in GPU hours using BCD compared to full parameter updates across different models on an RTX 4090 cluster.

	G-1.6B	G-5.4B	G-10B	G-20B
	GPUh ↓	GPUh ↓	GPUh ↓	GPUh ↓
Full Update	0%	0%	0%	0%
BCD(The worst)	1.5%	-127.0%	-131.8%	-78.0%
BCD(Average)	55.6%	-2.4%	-4.5%	19.7%

Table 13: Reduction in GPU hours using BCD on an RTX 4090 cluster compared to full-parameter updates on an A100 cluster across different models.

	G-1.6B	G-5.4B	G-10B	G-20B
	GPUh ↓	GPUh ↓	GPUh ↓	GPUh ↓
Full Update	0%	0%	0%	0%
BCD(The worst)	6.4%	-114.7%	-121.6%	-70.5%
BCD(Average)	57.8%	3.2%	0.08%	23.1%

Table 14: Reduction in GPU hours using BCD on 4090 clusters compared to full-parameter updates on an A800 cluster across different models.

A.5 GPU Hour Analysis

Based on the data presented in tables 12, 13, and 14, the performance of the BCD method varies across different model scales, demonstrating both significant advantages and certain limitations. During the training of G-1.6B and G-20B models, the BCD method effectively reduces GPU hours. However, for G-5.4B and G-10B models, its GPU computation time is comparable to that of full-parameter training, and in some cases, even slightly higher. This indicates that the optimization of computational overhead in the BCD method still has room for improvement.

B UFP experiments

In this experiment, UFP (Unfrozen Parts) refers to the proportion of model parameters that remain trainable during training. By freezing a certain proportion of parameters in different experimental settings (e.g., 1/2, 1/3, 1/4, 1/5), we analyze the impact of parameter freezing on memory usage and training efficiency. Freezing part of the parameters reduces gradient computations and optimizer state storage, thereby lowering memory consumption, while potentially affecting training time and final model performance. This section focuses on investigating the memory usage and training time variations of GPT-2 and LLaMA models under different UFP configurations.

B.0.1 UFP experiments

UFP setting

In this section of the experiment, we froze 1/2, 1/3, 1/4, and 1/5 of the parameters, and the memory usage is as follows.

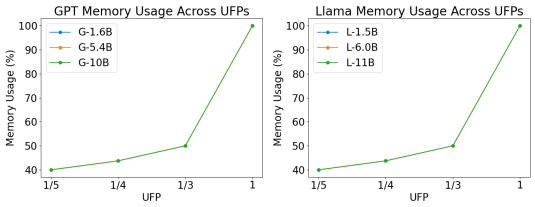

Figure 5: This figure presents the approximate memory usage percentage of GPT-2 and LLaMA models with different parameter scales under various freezing ratios, considering only the parameters, gradients, and optimizer states.

Experimental Result

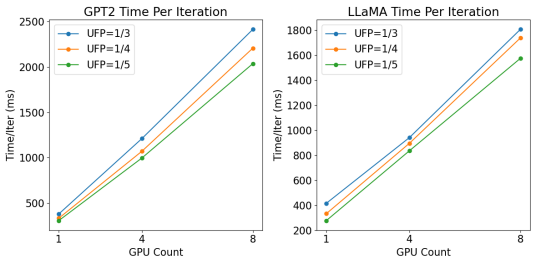

Figure 6: This figure presents the average training time per iteration for GPT-2 and LLaMA models when unfreezing different scales of parameters using the BCD method. The time is measured in milliseconds (ms).

Model	#GPU	t-1/3	t-1/4	t-1/5
G-1.6B	1	378.49	331.7375	305.52
G-5.4B	4	1210.61	1070.81	996.47
G-10B	8	2414.32	2207.405	2036.528

Table 15: This table presents the average training time per iteration for the GPT-2 model when unfreezing different scales of parameters using the BCD method. For example, t-1/3 represents the average training time per iteration when unfreezing one-third of the parameters. The time is measured in milliseconds (ms).

Analysis The experimental results indicate that as the proportion of frozen parameters increases,

Model	#GPU	t-1/3	t-1/4	t-1/5
L-1.5B	1	414.81	331.81	276.266
L-6.0B	4	941.44	895.7075	835.798
L-11B	8	1807.11	1738.2525	1574.92

Table 16: This table presents the average training time per iteration for the LLaMA model when unfreezing different scales of parameters using the BCD method. For example, t-1/3 represents the average training time per iteration when unfreezing one-third of the parameters. The time is measured in milliseconds (ms).

the time per iteration shows a decreasing trend. This trend has been validated across different model scales and GPU configurations.

From the data, it can be observed that for both GPT-2 and LLaMA models, freezing more parameters leads to a reduction in computation time per iteration, with the effect being more pronounced in smaller models. This suggests that in resource-constrained environments, freezing a portion of the parameters can effectively reduce memory consumption and computational overhead per iteration. However, it may require more iterations to achieve convergence.

Furthermore, as the model size increases, the absolute training time also rises, but the reduction in iteration time due to parameter freezing remains significant. This further demonstrates that in large-scale model training, adopting an appropriate parameter freezing strategy can alleviate the computational burden per iteration under resource constraints. However, the total training time and final model performance must be carefully considered to determine the optimal freezing ratio.