SimiGrad: Fine-Grained Adaptive Batching for Large Scale Training using Gradient Similarity Measurement

Anonymous Author(s)
Affiliation
Address
email

Abstract

Large scale training requires massive parallelism to finish the training within a reasonable amount of time. To support massive parallelism, large batch training is the key enabler but often at the cost of generalization performance. Existing works explore adaptive batching or hand-tuned static large batching, in order to strike a balance between the computational efficiency and the performance. However, these methods can provide only coarse-grained adaption (e.g., at a epoch level) due to the intrinsic expensive calculation or hand tuning requirements. In this paper, we propose a fully automated and lightweight adaptive batching methodology to enable fine-grained batch size adaption (e.g., at a mini-batch level) that can achieve state-of-the-art performance with record breaking batch sizes. The core component of our method is a lightweight yet efficient representation of the critical gradient noise information. We open-source the proposed methodology by providing a plugin tool that supports mainstream machine learning frameworks. Extensive evaluations on popular benchmarks (e.g., CIFAR10, ImageNet, and BERT-Large) demonstrate that the proposed methodology outperforms state-of-the-art methodologies using adaptive batching approaches or hand-tuned static strategies in both performance and batch size. Particularly, we achieve a new state-of-the-art batch size of 78k in BERT-Large pretraining with SQuAD score 90.69 compared to 90.58 reported in previous state-of-the-art with 59k batch size.

1 Introduction

Recent years have witnessed the success of deep learning in a variety of applications, such as computer vision [1] and natural language processing [2, 3]. To push the performance boundaries, deep learning models in many applications have become increasingly complicated, and training such models requires large amounts of data as well as high cost. Taking GPT3 [3] (an autoregressive language model that uses deep learning to produce human-like text) as an example, the state-of-the-art model has 175 billion parameters and is trained on 300 billion data, which costs $12 million for a single training run [4]. Therefore, it is of critical importance to study scalable and efficient training for large scale machine learning [5, 6].

The core idea of large scale machine learning is to parallelize model training across vast amounts of computing resources to speed up the training process. However, the overhead of the model synchronization due to massive parallelism in training can be the bottleneck for large scale training. To reduce the model synchronization overhead, large-batch training is a commonly employed strategy that can reduce the synchronization frequency. Taking BERT training as an example, increasing batch size from 256 to 4k can increase the training speed 2.5x faster on a 16-GPU cluster. OpenAI uses a batch size of 3.2 *million* to train GPT3 over a supercomputing cluster [3], which accelerates the training speed by approximately over 500x compared to sample-by-sample run on a 16-GPU cluster.

Despite the benefits of large batch size, large batch size may degrade the model generalization performance [7, 8, 9]. To address this issue, existing works [10, 11, 12] demonstrate that a carefully tuned learning rate scheduling can help to mitigate such performance loss caused by large batch

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

size. However, manually tuning learning rate requires significant amounts of efforts, which limits its potential use for wide adoption. [13, 14] develop automated learning rate tuning methods for large batch size, which, however, rely on users to set appropriate batch sizes. [15] develops adaptive batching using second order information, aiming to reduce the generalization impact due to large batch sizes. [16] uses the gradient variance information to guide the batch size and learning rate configurations. Due to the high cost of analyzing the gradient variance and the second order information, existing methods can adapt the batch size only at a coarse-grained level (i.e., at the end of each epoch). It is worth noting that within each epoch, the variance of gradient may change swiftly and significantly [17, 18] (also see Section 4.1). Therefore, the course-grained methods cannot find a suitable batch size to reduce the performance loss, and thus miss the opportunity to achieve larger batch sizes with less performance impact.

To address the above challenges, we propose a fine-grained batch size adaption approach SimiGrad to optimize the batch size at the iteration level. SimiGrad is driven by a novel gradient similarity measurement that can accurately capture the gradient variance information in a lightweight manner. We use the proposed gradient similarity measurement to derive the optimized batch size and learning rate with an automated algorithm. We integrate SimiGrad into mainstream machine learning frameworks by providing an open-sourced plugin tool.

We evaluate SimiGrad via extensive experiments using popular benchmarks, such as CIFAR10, ImageNet, and BERT-Large. The experimental results demonstrate that SimiGrad can consistently outperform state-of-the-art methods in both the average batch size and the model generalization performance. Particularly, we achieve a new state-of-the-art batch size of 78k in BERT-Large pre-training with the SQuAD score 90.69, compared to 90.58 reported in the state-of-the-art LAMB[19] with 59k batch size. We also achieve better Pareto frontier for the trade-off between the batch size and the model performance, compared with baselines and other state-of-the-art results.

We summarize our main contributions as follows:

- We develop an accurate gradient similarity measurement for capturing the gradient variance at the iteration level. This measurement can be directly inferred from the gradients of replicas with low computation overhead.
- We develop a fully automated fine-grained adaptive batching algorithm that can identify the optimal batch size and learning rate using the gradient similarity measurement.
- We integrate SimiGrad into mainstream machine learning frameworks and open-source it to benefit the community[1].
- SimiGrad enables a record breaking large batch size of 77k for BERT-Large pretraining. Notably, with very large batch sizes, SimiGrad is able to achieve better performance than state-of-the-art methods.

2 Background and Related Work

2.1 Mini-batching Training

The objective of training a neural network is to optimize the neural network parameters θ by minimizing the loss function $L(\theta)$. One typical method of training a neural network is the gradient descent method [20], which computes the gradient of $L(\theta)$ with respect to θ over the entire dataset; however, it could be too expensive for training deep neural networks with large training datasets. A typical solution is based on mini-batch stochastic gradient descent (SGD) [21], which samples a small subset, namely mini-batch, from the dataset and approximates the gradient using this batch instead. Such approximation introduces inevitable noise to the gradients. By increasing the batch size (i.e., the number of samples), we can reduce the noise level, but it is a diminishing return, meaning that after a sweet point we can gain almost no more information by further increasing the batch size [22]. In other words, the gradient computed on a too small batch is biased with noise, whereas a too large batch leads to a waste of time and insufficient updates to the model [23]. This makes the batch size a key control knob, and the choice of the batch size is of critical importance. In practice, a large batch may be too large to fit into GPU memory. The solution [24, 25] is to break it into many micro batches and accumulates the gradients of all micro batches. This method is called gradient accumulation, which is widely used in large scale training.

2.2 Large Batch Size Training

For large scale neural network training, it is desirable to have a large batch size, as it improves hardware utilization and system scalability as well as reduce wall clock training time. However, there is a dispute as to how exact large batch size would impact the model performance. In [7, 8], it is

[1] Anonymous repo URL: https://github.com/SimiGrad/SimiGrad

reported that training neural network model with a large batch size leads to a non-trivial generalization gap compared with a small batch size. [10] suggests that this gap could be closed by adjusting the training regime. [11] observes from a large number of training runs that this gap could be mitigated if the learning rate is well-tuned for a large batch size. [12] further observes that the range of "good" learning rates is shrinking as the batch size grows. [9] further argues that with carefully tuned hyper parameters, the smaller batch sizes still outperforms large ones in terms of test accuracy.

Despite the disagreement over large batch size's generalization performance, much effort has been made on how to choose an appropriate batch size and the corresponding learning rate for neural network training. [26] formulates SGD as stochastic differential equation and suggests that there exists an optimal noise scale related to the batch size, the learning rate and the dataset. Along this line, [16] formulates the noise scale in a different way and tests it in a variety of applications. Another line of research explores the potential of large batch size training from a practical perspective by fine tuning the training setting, especially the learning rate. [24] suggests that the learning rate should be increased proportionally to the square root of the increase ratio of the batch size. [13] proposes the warmup learning rate scheduling that can scale up the batch size to 8k for ResNet-50 on ImageNet.

The most closely related works to this paper are AdaScale SGD [14] and gradient noise scale [16]. AdaScale SGD [14] proposes a novel learning rate scaling rule based on the gradient variance for training with a large batch size; however, its performance hinges on the choice of a suitable batch size, which is challenging to determine in practice. Gradient noise scale [16] proposes to predict the batch size by comparing the gradients before and after the allreduce call in the data parallelism as an alternative implementation; however, the gradients collected in this way depends on the degree of parallelism in the system and introduces inevitable noise in the batch size estimation, which is addressed by manually tuning extra parameters.

2.3 Adaptive Batching

Using a fixed batch size throughout the training may not be the best practice. A desirable approach is to use adaptive batch sizes during training, which is suggested by research from both theoretical [22] and practical [27, 28] perspectives. [27, 28] use fixed multi-step batch size schedulers that closely resemble the multi-step learning rate scheduler. Recent works focus on utilizing the gradient noise (variance) information to adjust batch sizes by various rules and measurements. [29] proposes a coupled criterion for adjusting both the batch size and the learning rate. [15] adjusts the batch size using the second order information and adversarial training. [16] proposes a gradient noise scale to adjust the batch size.

3 Motivation and Challenges

Due to the high cost of analyzing the gradient variance and the second order information, existing methods can adapt the batch size only at a coarse-grained level (i.e., at the end of each epoch). Our key observation is that the gradient variance may change swiftly across iterations within one epoch (see Fig. 2 in [17] and Fig. 3 in [18]), which cannot be captured by the existing methods. Taking [16] as an example, to get the gradient variance information, it needs to run at least 20 batches with different batch sizes. To perform such analysis at each iteration, the extra overhead due to gradient variance analysis would be more than 20 times of the original training, which is infeasible in practice. Due to these issues, the batch size usually needs to be compromised and large scale training methods that achieve state-of-the-art performance still heavily rely on manual tuning efforts [5, 6].

To address these issues, one promising solution is by improving the granularity of gradient analysis, which, however, is challenging due to the following reasons.

- To compute the gradient variance, we need to first collect the gradient information from batches at different sizes and then take large amounts of computing time and resources. For example, computing the gradient variance for a single 8k batch takes 441ms and the time grows significantly with the batch size.
- Even if we can obtain the gradient variance, it is not straightforward to derive the optimal batch size as the gradient variance is a big matrix.
- Existing batch adaption methods usually require a fine-tuned learning rate scheduler to achieve a good performance. Fine tuning the learning rate at the iteration level requires significant amounts of manual efforts, which may not be feasible in practice.
- Due to the frequent gradient analysis, the system deployment is no longer a trivial engineering problem as the analysis and tuning overhead can disrupt the well optimized training pipeline in existing machine learning frameworks. For example in [16], executing multiple batches with different batch sizes to compute the gradient variance is not supported by the training pipeline and may lead to inefficient memory use.

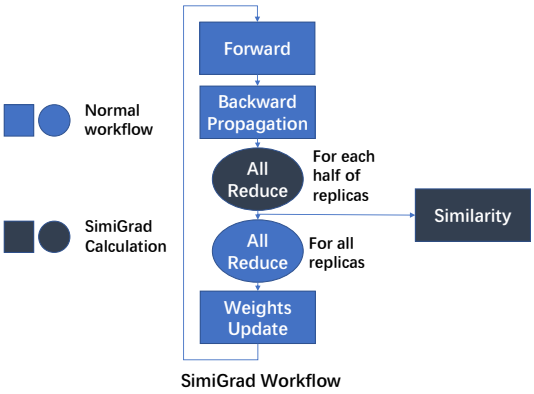

Figure 1: Workflow of how SimiGrad is built into normal data parallelism pipeline.

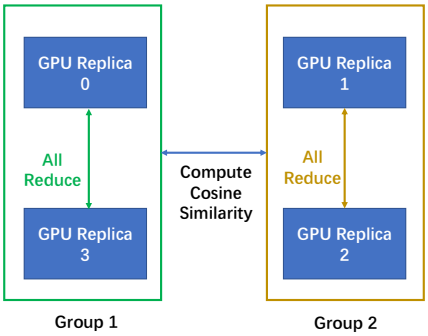

Figure 2: Cosine similarity computation from two groups of replicas.

4 SimiGrad: Fine-Grained Adaptive Batching Methodology

To address the above challenges, we propose a fine-grained adaptive batching methodology SimiGrad that supports swift batch size adaption at the iteration level. The core component of SimiGrad is a novel lightweight gradient variance measurement that can capture the gradient variance change per iteration without expensive analysis. Our adaptive batching algorithm uses the gradient variance measurement to guide the batch size configuration so that the batch size is maximized while the gradient variance is maintained at a steady level. SimiGrad also supports automated learning rate scheduling and requires little manual effort. Finally, we integrate SimiGrad into mainstream machine learning frameworks with optimized real system performance and open-source it for easy adoption.

4.1 Measure Gradient Variance via Cosine Similarity of Batching

Existing works typically estimate gradient variance via collecting gradients with different batch sizes [16], which, however, is quite expensive due to the calculation of multiple batches as well as the process of collecting gradients from different batch sizes. In this paper, we propose a new way to measure the gradient variance by computing the cosine similarity of the gradient of a mini-batch.

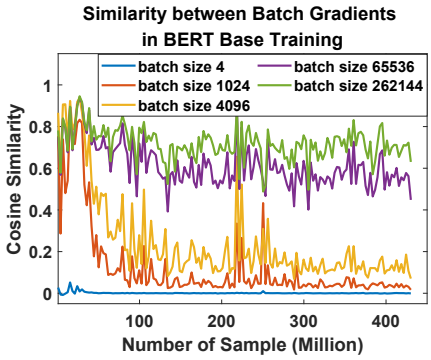

Figure 3: Cosine similarity in BERT pretrain.

Cosine similarity can efficiently measure the angle (i.e., direction difference) between gradients obtained from different batches, based on which we can estimate the "noise" level and optimize the batch size. This provides an indirect way to infer the gradient variance. As cosine similarity does not require collecting additional results using different batch sizes, we can evaluate the gradient variance at each iteration.

Specifically, the cosine similarity is computed at each iteration using 2 aggregated gradients. To this end, we utilize the replicas in distributed training systems. A replica is a local copy of model and optimizer on certain hardware (typically a GPU) [30, 31]. As shown in Fig.1, in each training step, all replicas $r_1, r_2, ..., r_n$ start with the same weights θ but with different mini-batches. After the feed forward and backward propagation of loss, each replica has its own gradient information $g_1, g_2, ..., g_n$. Then allreduce, a collective operation [32], is called to aggregate gradients from replicas, thanks to its excellent capability of utilizing network bandwidth to achieve fast communication.

Before the gradients are aggregated by allreduce, we split replicas equally into two groups, denoted as $\{g_1, g_3, ...\}$ and $\{g_2, g_4, ...\}$. Then we perform aggregation using allreduce in each group and divide the gradients by the number of replicas in each group. This process is visualized in Fig. 2. The aggregated gradients in each group are denoted as $G_1 = \frac{2}{n}\sum_{i=1}^{\frac{n}{2}} r_{2i-1}$ and $G_2 = \frac{2}{n}\sum_{i=1}^{\frac{n}{2}} r_{2i}$, respectively, which are equivalent to the gradients of two mini-batches with half of the batch size. Then we compute the cosine similarity of the two group gradients as the measurement of gradient variance $\Phi = \cos(G_1, G_2)$. Once the gradient variance is computed, we aggregate the two group gradients again using allreduce to compute the final aggregated gradients. At this point, we reach the step as the normal workflow.

When computing the cosine similarity, the training pipeline is not interrupted and the main overhead is an additional allreduce call and a cosine similarity computation. This is a significant cost reduction, compared with the gradient variance evaluation in previous works that need to collect multiple additional batches with different batch sizes or use heuristic algorithms. Taking BERT-Large pretrain on a 32 GPU cluster as an example, the overhead of an extra allreduce call and cosine similarity computation is only around 10-15 ms, which is irrelevant to the batch sizes. In comparison, the gradient computation of a single 8k batch of BERT takes about 441ms and the time grows with the batch size and the need of computing multiple batch sizes [16].

To demonstrate the proposed cosine similarity is an effective measure for tracking the gradient variance, we plot the cosine similarity over the BERT pertraining for different batch sizes in Fig. 3. It is observed that the similarity of the gradients increases with the batch size. For the small batch size of 4, due to the high "noise" in the gradients, the cosine similarity is almost always 0. For the batch sizes of 1024 and 4096, the gradients are stable initially. However, the gradient quickly becomes unstable and thus the cosine similarity drops. For the large batch sizes of 65536 and 262144, the cosine similarity is relatively stable over the time, which corresponds to the stable gradients we get.

4.2 Automated and Fine-grained Adaptive Batching Algorithm

Using the cosine similarity, we propose an automated and fine-grained adaptive batching algorithm to derive optimized large batch sizes. The total batch size is denoted as $B = m * s * n$, where m is the micro batch size on each replica, s is the gradient accumulation steps, and n is the number of replicas. In practice, the number of replicas n depends on the number of available GPUs and the micro batch size m is limited by GPU memory. Thus the factor n is given and m has an upper bound $m_{\max}$.

Algorithm 1 provides an overview of our adaptive batching algorithm. For every n steps, we compare the measured gradient similarity Φ with the user defined similarity threshold γ and update the target batch size B accordingly using a predefined incremental step (e.g., 10%) to make the batch gradient similarity as close to the threshold as possible. Then, Algorithm 2 is invoked to update the micro batch size m and the gradient accumulation steps s to make the effective batch size as close to the target batch size as possible. Based on the updated batch size, we update the learning rate accordingly.

Note that in Algorithm 1, the only required parameter is the noise threshold γ and the other parameters are optional to improve the flexibility of the algorithm. Specifically, the adjust interval n is used when training on small batch sizes or slow networks to reduce overhead. The batch size lower bound $B_{\min}$ is introduced for special scenarios where a well-tuned batch size and a learning rate scheduling are provided. The batch size upper bound $B_{\max}$ is used when training with only limited epochs to ensure a reasonable number of updates to the model. In Sec. 5.3.2, we perform an ablation study to analyze the impact of the parameters in Algorithm 1. The results indicate that Algorithm 1 performs well with a broad range of γ even without any optional parameters.

Algorithm 1: Adaptive Batching Algorithm Overview

User specifications: noise threshold γ, adjust interval n (optional), batch size lower bound $B_{\min}$ (optional), batch size upper bound $B_{\max}$ (optional), micro batch size upper bound $m_{\max}$ (optional).
Initialization: learning rate modifier $\alpha = 1$. If $m_{\max}$ not set, $m_{\max} = m$. If n not set, $n = 1$.
For every n steps in training
1) Retrieve noise measurement Φ.
2) If $n < \gamma$, target batch size $B = 0.9B$, else, $B = 1.1B$.
3) Update the batch size to approximate B based on Algorithm 2.
4) Update the learning rate modifier α according to $\frac{\alpha_{new}}{\alpha_{old}} = \sqrt{\frac{B_{new}}{B_{old}}}$.

Algorithm 2: Adaptive Batch Size

Input: Target batch size b, micro batch size upper bound $m_{\max}$, batch size lower bound $B_{\min}$ (optional), batch size upper bound $B_{\max}$ (optional). (Here, '$\backslash$' means integer division)
1) If $b_{\max}$ set, bound the batch size $b = \min(b, b_{\max})$.
2) If $b_{\min}$ set, bound the batch size $b = \max(b, b_{\min})$.
3) If $m_{\max}$ set, update micro batch size $m = \min(b\backslash n, m_{\max})$.
4) Update gradient accumulation steps $s = b/m\backslash n$.

4.3 Learning Rate Scheduling Support

Existing works [10, 11, 12, 26, 24, 13] suggest that the learning rate should be adjusted while adapting batch sizes in order to achieve good performance. We employ a learning rate scaling rule from [10, 24, 19] to adjust the learning rate automatically using a square root factor of the batch size change (see Step 4 of Algorithm 1). It is worth nothing that SimiGrad works with or without warmup [13]. If warmup is not used (i.e., a random initial learning rate is used), SimiGrad can still achieve good performance. It is interesting to observe that the learning rate automatically adjusted by the learning rate scaling rule resembles a similar pattern as the fine-tuned warmup scheduling (see Fig. 5). This indicates SimiGrad is robust and provides good support for the learning rate adaption.

4.4 Machine Learning System Integration

We can further optimize SimiGrad runtime overhead by instrumenting the allreduce operation. As we have two groups of replicas, we decompose an allreduce process into a reduce-scatter process inside each group, an allreduce among groups, and an allgather process inside each group. After the reduce-scatter process, each replica holds one portion of the reduced gradients in each group. Thus we can directly compute the cosine similarity from the gradients. Then the allreduce is executed among groups but only to exchange the portion of the reduced gradients. Here the transferred data volume size is inversely proportional to the degree of parallelism and thus the overhead is small. Finally, the allgather is executed inside each group, the same as the normal workflow. In this way, we can avoid the overhead of an extra allreduce to further reduce the overhead of SimiGrad to only the cosine computation.

Modern machine learning frameworks are built with fixed micro batch sizes. Even if we can instrument the the framework to make it support flexible batch sizes, it may disrupt the optimized existing training pipelines, such as the widely used NVIDIA DALI[33]. Therefore, we opt to design a batch dispatch queue that accumulates the data samples until the micro batch size is reached before aggregating them into a micro batch to be executed. This general design enables our approach to be easily integrated into almost any machine learning framework.

Furthermore, we do not directly change the learning rate as the client scripts may operate or depend on it (e.g., a custom learning rate scheduler). Instead, we use a learning rate modifier factor to indicate the scaling of learning rate due to the adaptive batch size, which will be applied when model weights update.

5 Evaluation

We evaluate SimiGrad via extensive experiments using popular benchmarks and demonstrate the superiority of SimiGrad over these benchmarks using the following criteria:

- *Performance*: We compare the performance of SimiGrad with state-of-the-art adaptive batching and large batch training techniques, and demonstrate superior results on ResNet18, ResNet50, and BERT models.
- *Generalization*: We demonstrate that SimiGrad establishes a better Pareto frontier of average batch size under different evaluation metrics, and sustains good generalization at very large batch sizes.
- *Robustness*: We show that SimiGrad does not need a manual learning rate warm-up schedule required by most large batch training techniques or careful tuning of hyperparameters, which makes SimiGrad robust against hyper-parameter variations.
- *Scalability*: We demonstrate that SimiGrad allows deep learning training to scale more efficiently on a wide range of GPU clusters with both slow and fast interconnect.

5.1 Evaluation Setup

Evaluations are performed on Ubuntu 18.04 on NVIDIA DGX-2 nodes. The applications are evaluated with DeepSpeed 0.2, Pytorch 1.6+, and CUDA 10.1 and 11.1. To ensure fair comparison, the hyperparameters, including learning rate and seeds, are kept the same between the baseline and our approach evaluations. Gradient accumulation is used when the batch size exceeds the graphic memory limit. All the training codes are publicly available together with our framework implementation at anonymous repo URL: https://github.com/SimiGrad/SimiGrad.

- *ResNet18 on CIFAR10*: The training lasts 200 epochs with SGD optimizer. The initial learning rate is 0.1, which decays at epoch 80 and 120 by factor 0.1, respectively. For SimiGrad, we set the similarity threshold 0.1 and the maximum batch size 2048.
- *ResNet50 on ImageNet*: The training lasts 90 epochs with SGD optimizer. The initial learning rate is 0.1, which decays at epoch 30, 60, and 80 by factor 0.1, respectively. For SimiGrad, we set the similarity threshold 0.1 and the maximum batch size 32k.

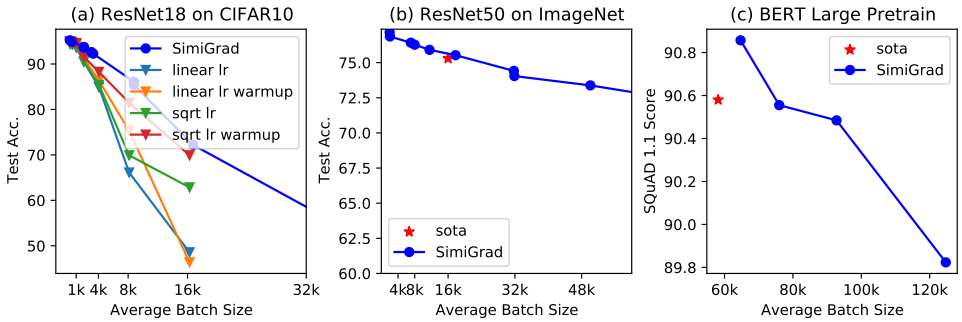

Figure 4: Pareto frontier of the average batch size under different performance metrics. For the same average batch size, SimiGrad always performs better than the other benchmarks.

- *BERT Pretrain on WiKi and BookCorpus Datasets*: We pretrain BERT Large [2] from scratch for 170 epochs on WiKi and BookCorpus datasets with fixed sequence length 128 and 512 using LAMB optimizer. The training procedure follows the recommendation from the BERT paper [2], based on which we train epoch 1-150 using the sequence length 128 and epoch 151-170 using the sequence length 512. We evaluate it for two cases: (i) a well-tuned warmup learning rate scheduling is used and (ii) a simple learning rate decay is used. For the warmup case, the learning rate follows an exponential increase to 0.011 at epoch 9 and then decays with a factor of 0.9 for every 250 steps until epoch 150. For the simple case, a fixed learning rate 0.0006 with a linear decay is used until epoch 150. For both cases, the sequence length 512 training follows a warmup scheduling until learning rate 0.02 and then decay with a factor of 0.9 for every 150 steps. The batch size for sequence length 512 is fixed at 32k. For the implementation details, please refer to our open-sourced codes.

State-of-the-Art Adaptive Batching Methods. We cross-compare the performance of SimiGrad with other approaches that can adaptively adjust the batch size during training as well as the hand-tuned baselines. The evaluation metrics are the average batch size and the final generalization performance. The approaches we compared with are:

- *AdaBatch[28]*: a hard tuned strategy that gradually increases the batch size b and the learning rate α along the training to keep b/α constant.
- *AdaptDL*: a library that adaptively sets the batch size based on the gradient noise scale [16] and sets the learning rate according to AdaScale [14].
- *ABSA[15]*: an adaptive batching scheme using second order methods and adversarial training.

5.2 Performance Analysis

For CIFAR-10, Table 1 indicates that SimiGrad achieves a higher test accuracy than the baseline, while keeping the largest average batch size among the other approaches that have similar test accuracies. Compared with AdaBatch that uses 15k model updates, SimiGrad uses only 5k model updates, which indicates almost 3x speedup.

For ImageNet, Table 1 shows that SimiGrad achieves a better test accuracy with twice as large as the baseline batch size. Note that we use the vanilla SGD without tricks (such as label-smoothing and mixup data augmentation) and sophisticated learning rate scaling (such as LARS). Under this setup, the state-of-the-art approach [34] achieves 75.3% test accuracy with 16k batch size under well-tuned learning rate scaling, warmup, and decay. To show the performance of SimiGrad with larger batch sizes, we test more aggressive settings with similarity threshold 0.4, minimum batch size 2k, and maximum batch size 32k. Under this setting, the average batch size of SimiGrad reaches 16k, while keeping the test accuracy at 75.71% which is 0.4% higher than the hand-tuned results in [34].

For BERT Large pretraining, the model has 340 million parameters. The results in Table 1 shows the advantage of SimiGrad in large scale training over other approaches (the original BERT paper and the state-of-the-art LAMB). Specifically, even with an ultra large batch size 128k (using similarity threshold 0.8), SimiGrad can achieve comparable SQuAD scores. With less extreme batch size 66k (using similarity threshold 0.5), SimiGrad can achieve a better SQuAD score 90.85 than the state-of-the-art LAMB that uses an average batch size 59k and achieves SQuAD score 90.58.

5.3 Generalization and Robustness

To demonstrate the generalization and robustness of SimiGrad with large batch sizes, we generate the Pareto frontier of the average batch size under different performance metrics for different applications. As illustrated in Fig. 4, it can be observed that the model performance degrades as the batch size

Table 1
ADAPTIVE BATCH RESULTS

ResNet18 on CIFAR10 (SGD)				
	Number of Updates	Average Batch Size	Number of Samples	Test Accuracy (%)
Baseline	78125	128	10M	94.28
AdaBatch	15137	330	5M	94.38
AdaptDL	7489	1335	10M	93.07
SimiGrad	**5094**	**1963**	**10M**	**94.33**
ABSA	4327	2311	10M	85.85
ResNet50 on ImageNet (SGD)				
	Number of Updates	Average Batch Size	Number of Samples	Test Accuracy (%)
Baseline	56250	2048	115M	76.56
Hand-tuned Baseline	7038	16384	115M	75.3
SimiGrad	**23284**	**4952**	**115M**	**76.85**
	7007	**16456**	**115M**	**75.71**
BERT Large Pre-Training				
	Number of Updates	Average Batch Size	Number of Samples	SQuAD F1 score
BERT Paper Baseline	1000000	256	256M	90.9
LAMB Paper Baseline	8599	59542	512M	90.58
NoWarmup Baseline	106793	4096	437M	86.9
Ours	**5620**	**77831**	**437M**	**90.69**
	3425	**127711**	**437M**	**89.82**
Ours (no warmup)	**6604**	**66236**	**437M**	**90.87**
	4779	**91530**	**437M**	**90.39**
Ours (no warmup, no optional parameters)	**7207**	**60694**	**437M**	**90.52**

increases, which agrees with the observations in related work. It is worth noting that SimiGrad is able to keep the model performance reasonably good for a wide range of batch sizes and constantly outperforms the state-of-the-art approaches.

5.3.1 Robustness under Learning Rate Schedules

Learning rate warmup and learning rate scaling are two common techniques used to improve the model generalization for large batch training. Fig. 4(a) shows the impact of two common learning rate scaling rules (the linear scaling and squared root scaling) with large batch size; for both scenarios, we also show the impact of learning rate warm up. As illustrated in Fig. 4(a), the results with warmup have better test accuracies than the counterpart without warmup, and the squared root scaling is better than the linear scaling. However, despite the effectiveness of the learning rate warmup and the learning rate scaling, identifying an optimal schedule that works well for a certain batch size requires a non-trivial tuning effort.

Notably, SimiGrad can mitigate this effort and outperform even well tuned schedules with minimal tuning. To show the robustness of SimiGrad with untuned learning rate, we evaluate SimiGrad's performance for BERT pretrain without using any warmup but with a *untuned* learning rate with linear decay (see the details in Sec. 5.1). The results are presented in Table 1. Compared with the scenario of well-tuned learning rate, SimiGrad with untuned learning rate chooses more conservative batch sizes (still larger than state-of-the-art batch sizes) but has better SQuAD scores. These results also outperform the state-of-the-art results. Compared with SimiGrad, NoWarmup Baseline in Table 1 (the same untuned learning rate with LAMB without any warmup schedule using a fixed batch size 4k) shows a non-trivial performance degradation.

The reason why SimiGrad is robust against the learning rate warm-up is because it uses the gradient similarity metric to automatically adjust the effective learning rate at run time. From Fig. 5(a), it can be observed that even when no learning rate warmup schedule is provided, SimiGrad is able to

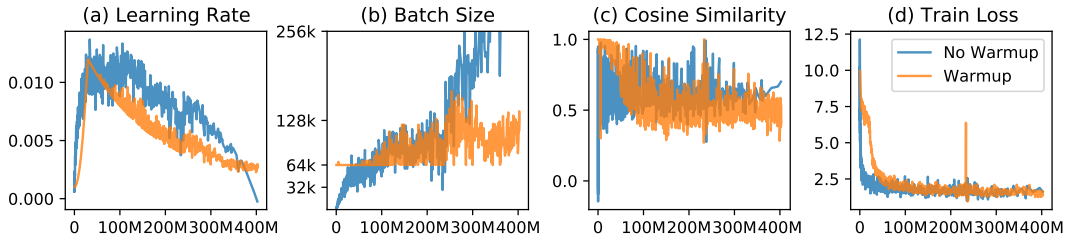

Figure 5: Details of BERT Large pretrain with adaptive batch size in sequence length 128.

come up with a warmup-like learning rate scaling based on the adaptive batch size, which not only saves considerable tuning efforts but also leads to a better model performance than the hand-tuned warmup schedules. When highly tuned warmup schedule is available for a large batch size, SimiGrad can leverage it to start the training from a pre-tuned large batch size, and increase it further over the training, which ultimately reaches a larger average batch size.

5.3.2 Robustness Ablation Study

We perform the ablation study to evaluate SimiGrad's robustness against different parameters. First, we get rid of all optional parameters (such as batch size bounds) and extra learning rate scheduling (warmup), and run the BERT pretrain with the gradient similarity $\gamma = 0.6$. The results are shown in the bottom of Table 1, which shows comparable performance as the other experiments and outperforms the baselines. This indicates that SimiGrad can work well without any optional parameters nor extra learning rate scheduling. Then, in the same setting we evaluate the impact of gradient similarity values $\gamma = [0.6, 0.7, 0.8]$ on the performance of SimiGrad. The corresponding SQuAD scores are 90.52, 90.22, and 90.38 with the average batch size 60k, 75k, and 91k, respectively. This further indicates that SimiGrad can perform well under different gradient similarity thresholds.

5.4 Real-world Scalability and Speed Up

Large batch size enables training the model on a large number of GPUs without getting bottle-necked by the all-reduce communication. If we assume that a GPU can compute X samples per second, then a batch size of B can be processed by N gpus in

$$iteration_time = \frac{B}{NX} + allreduce_cost = \frac{B}{NX} + 2 \times \frac{N-1}{N} \times \frac{M}{bw} \tag{1}$$

where, the absolute allreduce cost can be written as function of N, network bandwidth (bw) between GPUs and the model size (M). But relatively, more time is spent in allreduce as N gets larger or B gets smaller. Therefore, a large batch size is necessary to achieve good efficiency.

Using (1), we can evaluate the impact of batch sizes on end-to-end training time as a function of GPU cluster size and GPU-GPU bandwidth within the cluster. We measure X using an NVIDIA V100 GPU, and simulate the end-to-end training time for BERT-Pretraining on a small, medium and large cluster with slow (1GB/s), moderate (5GB/s) and fast interconnect (60 GB/s). Our simulation in Fig. 6 shows that SimiGrad achieves a 23.67%, 40.44%, and 32.97% reduction in training time compared to baseline on 32, 512 and 2048 GPUs on the small, medium and large clusters with slow, moderate and fast interconnect, respectively. This demonstrates the real world impact of our approach in improving training scalability across a wide range of actual GPU clusters with varying network bandwidth.

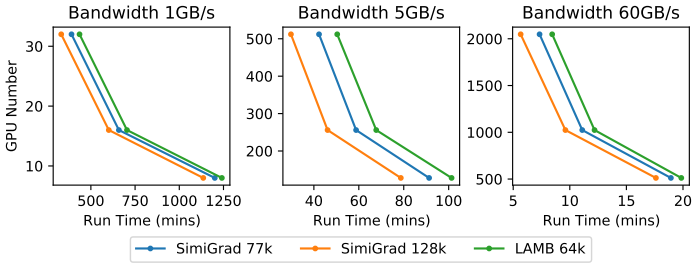

Figure 6: Cosine similarity of different batch sizes in BERT pretrain.

6 Conclusion

In this paper, we propose SimiGrad, a fine-grained adaptive batching methodology for enabling swift batch size adaption at the iteration level. The core component of SimiGrad is a lightweight gradient similarity measure based on cosine similarity and an enhanced synchronization approach. Our optimized machine learning system integration allows SimiGrad to be executed with negligible overhead in practice. SimiGrad achieves record breaking large batch size of 78k for BERT-Large pretraining with SQuAD score of 90.69.

References

[1] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 2019.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[4] Kyle Wiggers, 2020. https://venturebeat.com/2020/06/01/ai-machine-learning-openai-gpt-3-size-isnt-everything/; [Online].

[5] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, Tiegang Chen, Guangxiao Hu, Shaohuai Shi, and Xiaowen Chu. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *CoRR*, abs/1807.11205, 2018.

[6] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch SGD: training resnet-50 on imagenet in 15 minutes. *CoRR*, abs/1711.04325, 2017.

[7] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[8] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade - Second Edition*, volume 7700 of *Lecture Notes in Computer Science*, pages 9–48. Springer, 2012.

[9] Samuel L. Smith, Erich Elsen, and Soham De. On the generalization benefit of noise in stochastic gradient descent. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 9058–9067. PMLR, 2020.

[10] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 1731–1741, 2017.

[11] Christopher J. Shallue, Jaehoon Lee, Joseph M. Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. Measuring the effects of data parallelism on neural network training. *J. Mach. Learn. Res.*, 20:112:1–112:49, 2019.

[12] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *CoRR*, abs/1804.07612, 2018.

[13] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017.

[14] Tyler B. Johnson, Pulkit Agrawal, Haijie Gu, and Carlos Guestrin. Adascale SGD: A user-friendly algorithm for distributed training. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 4911–4920. PMLR, 2020.

[15] Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W. Mahoney. Large batch size training of neural networks with adversarial training and second-order information. *CoRR*, abs/1810.01021, 2018.

[16] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *CoRR*, abs/1812.06162, 2018.

[17] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[18] Fartash Faghri, David Duvenaud, David J. Fleet, and Jimmy Ba. A study of gradient variance in deep learning. *CoRR*, abs/2007.04532, 2020.

[19] Yang You, Jing Li, Sashank J. Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training BERT in 76 minutes. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[20] Haskell B Curry. The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics*, 2(3):258–261, 1944.

[21] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *19th International Conference on Computational Statistics, COMPSTAT 2010, Paris, France, August 22-27, 2010 - Keynote, Invited and Contributed Papers*, pages 177–186. Physica-Verlag, 2010.

[22] Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. Don't decay the learning rate, increase the batch size. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[23] Fengxiang He, Tongliang Liu, and Dacheng Tao. Control batch size and learning rate to generalize well: Theoretical and empirical evidence. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 1141–1150, 2019.

[24] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997, 2014.

[25] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, 2011.

[26] Samuel L. Smith and Quoc V. Le. A bayesian perspective on generalization and stochastic gradient descent. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[27] Soham De, Abhay Kumar Yadav, David W. Jacobs, and Tom Goldstein. Big batch SGD: automated inference using adaptive batch sizes. *CoRR*, abs/1610.05792, 2016.

[28] Aditya Devarakonda, Maxim Naumov, and Michael Garland. Adabatch: Adaptive batch sizes for training deep neural networks. *CoRR*, abs/1712.02029, 2017.

[29] Lukas Balles, Javier Romero, and Philipp Hennig. Coupling adaptive batch sizes with learning rates. In Gal Elidan, Kristian Kersting, and Alexander T. Ihler, editors, *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017*. AUAI Press, 2017.

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019.

[31] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In Kimberly Keeton and Timothy Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, pages 265–283. USENIX Association, 2016.

[32] NVIDIA. The nvidia collective communications library (nccl). `https://github.com/NVIDIA/nccl`.

[33] NVIDIA. The nvidia data loading library (dali). `https://github.com/NVIDIA/DALI`.

[34] Chris Ying, Sameer Kumar, Dehao Chen, Tao Wang, and Youlong Cheng. Image classification at supercomputer scale. *CoRR*, abs/1811.06992, 2018.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section .
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We provide ablation study in section 5.3.2
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Our framework and all training code can be found at the anonymous Github repo https://github.com/SimiGrad/SimiGrad
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Please see section 5.1
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] All our results (except the GLUE finetuning as suggested not to by BERT paper) are using fixed seed
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Please see section 5.1
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]