GROUP-WISE VERIFIABLE DISTRIBUTED COMPUTING FOR MACHINE LEARNING UNDER ADVERSARIAL ATTACKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Distributed computing has been a promising solution in machine learning to accelerate the training procedure on large-scale dataset by utilizing multiple workers in parallel. However, there remain two major issues that still need to be addressed: i) adversarial attacks from malicious workers, and ii) the effect of slow workers known as stragglers. In this paper, we tackle both problems simultaneously by proposing Group-wise Verifiable Coded Computing (GVCC), which leverages coding techniques and group-wise verification to provide robustness to adversarial attacks and resiliency to straggler effects in distributed computing. The key idea of GVCC is to verify a group of computation results from workers at a time, while providing resilience to stragglers through encoding tasks assigned to workers with Group-wise Verifiable Codes. Experimental results show that GVCC outperforms the existing methods in terms of overall processing time and verification time for executing matrix multiplication, which is a key computational component in machine learning and deep learning.

1 INTRODUCTION

Recently, machine learning and big data analysis have achieved huge success in various areas such as computer vision, natural language processing, and reinforcement learning, etc. Since they usually demand a massive amount of computation on a large dataset, there has been increasing interest in distributed systems, where one node is used as a master and the others are used as workers. Depending on whether each worker has an entire model or not, machine learning in distributed settings can be categorized into two groups.

The first category is distributed learning where each worker possess an entire model and a dataset. In this setting, workers compute gradients on their models with respect to own datasets (vectors that minimize empirical losses), and send them to a master. A master updates a main model by aggregating these gradients, then distributes an updated model back to workers (Dean et al., 2012; Li et al., 2014; Anderson et al., 2017; Li et al., 2020a). By iteratively communicating gradients and updated models, a single neural network can be trained in parallel utilizing multiple workers. It is also a possible option for workers to directly send their local models instead of gradients to a master. More often than not, each worker may have to preserve the privacy of own data or it is burdensome to allow the system to communicate every iteration. Federated learning (McMahan et al., 2017; Bonawitz et al., 2019; Li et al., 2020b) was proposed to handle communication load or privacy issue. However, Zhao et al. (2018) argues that the performance of distributed and federated learning is severely degraded when distributions of local datasets are not i.i.d. Furthermore, it is hard to exploit memory-restricted machines as workers when data or model size is immense.

Distributed computing is the second category that can to deal memory concerns (Dalcín et al., 2005; 2011). In distributed computing, each worker does not have its own model and data. Instead, a master divides and distributes tasks (which require far small memory than the model) to workers, and they compute the assigned tasks and send results back to a master.

ratio of attacks	0	1/32	2/32	4/32	8/32
CIFAR10	164	224	235	262	287
CIFAR100	589	785	809	845	958
MNIST	16	18	21	23	25
FMNIST	213	264	281	298	322

Table 1: Number of epochs till convergence. (Trained with 32 workers under random value attacks)

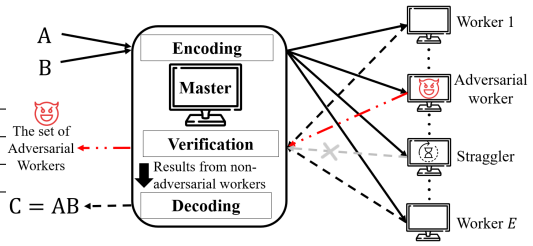


Figure 1: Distributed computing for a matrix multiplication $\mathbf{C} = \mathbf{AB}$ under adversarial attacks

Distributed computing can be utilized to compute matrix multiplication in machine learning, the most important and frequent computation block. However, synchronized communication is the foremost consideration to embed distributed computing in machine learning applications. It has been reported that some workers which return their computation results much slower than others, called stragglers, can be a serious bottleneck to perform large-scale computation tasks (Dean & Barroso, 2013; Huang et al., 2017; Tandon et al., 2017). To handle straggler effects, coded computing was first suggested in Lee et al. (2018). A master can encode a computation task while retaining redundancy in task allocation with a coding technique. A master does not need all results of tasks to achieve final output and can ignore stragglers. This approach has been applied to various computation tasks, especially on matrix multiplication (Dutta et al., 2016; Yu et al., 2017; Park et al., 2018; Reisizadeh et al., 2019; Dutta et al., 2019; Yu et al., 2020). Therefore, even if each worker has a restricted memory and computational resource, a large-scale neural network can be trained in parallel with multiple workers.

In a distributed setting, however, some of the workers could be adversarial workers, which send perturbed results to the master to contaminate or degrade the performance of neural network. Many studies (Biggio et al., 2012; Blanchard et al., 2017; El Mhamdi et al., 2018; Sohn et al., 2020; Bagdasaryan et al., 2020; Wang et al., 2020) demonstrate that adversarial workers slow down overall training process and severely degrade performance of a neural network. In this paper, we mainly consider attacks from adversarial workers (i.e. Byzantine workers) in distributed computing, where they try to contaminate the final computation product by returning wrong computation results for the assigned task. Table 1 shows the convergence speed of neural network trained with distributed computing, under a random value attack. Neural network requires about $\times 1.3$ more epochs for convergence with only 3.125% of adversarial workers.[1] To be robust against adversarial attacks, a master can typically exploit two approaches in designing the computation tasks. The first one is to encode the computation tasks and decode the final output from received results. By using coding approach, the final computation product can be correctly recovered at a master even if a part of computation results of the workers are wrong. The second one is that a master checks whether the computation results from workers are correct or not to figure out adversarial workers. Based on this idea, there have been some recent studies that suggest an encoding scheme for assigning computation tasks to workers to tolerate wrong computation results when obtaining the final computation product, and to identify adversarial workers from the returned computation results (Yu et al., 2019; Soleymani et al., 2021; Hong et al., 2022). Another line of research, which we call verifiable computing, has focused on identifying adversarial workers with a fast verification procedure. In verifiable computing, a master verifies the correctness of computation results by using a verification key (Freivalds, 1977; Tang et al., 2022).

In this paper, we focus on designing a fast and robust distributed computing system for matrix multiplication. We use both a coding approach and verifiable computing to combat adversarial attacks from malicious workers and straggler effects. Our system model is depicted in Fig 1, where a master encodes the computation tasks, distributes them to the workers, and decodes the final product after receiving and verifying the computation results from the workers.

To be specific, we suggest a novel encoding scheme that enables verifying a large number of computation results at a time, and decoding the final computation product from a part of computation results from the workers to mitigate straggler effects as well. We experiment with various settings and demonstrate that our proposed scheme can identify adversarial workers and obtain the final

[1] We provide more explanation and detailed experiment settings in Appendix A.

computation product much faster than existing methods. The main contributions of this paper are as follows.

- We propose **Group-wise Verifiable Coded Computing (GVCC)**, handling both straggler effects and adversarial attacks in distributed computing for matrix multiplication tasks. To be more specific, we suggest group-wise verifiable codes (GVC) for encoding the computation tasks of workers and provide a suitable group-wise verification algorithm. We first provide a process of group-wise verification trial (GVT), then propose a two-stage verification algorithm based on GVT. We also provide modified verification algorithm that can be utilized under straggler effects.
- We demonstrate the performance of GVCC via experiments in terms of overall processing time and verification time for executing matrix multiplication tasks in distributed computing under adversarial attacks and straggler effects. According to experimental results, GVCC speeds up overall processing time and verification time up to $\times 1.11$ and $\times 3.34$ compare to that of the existing verifiable coded computing scheme in Tang et al. (2022), and speeds up up to $\times 2.46$ and $\times 174$ than group testing based adversarial attack identification scheme for coded matrix multiplication in Hong et al. (2022), respectively.

Notations. For $a, b \in \mathbb{Z}$, $[a : b]$ denotes $\{a, a+1, \ldots, b\}$.

2 Problem Setting

We consider a distributed computing scenario for matrix multiplication task with one master and E workers $\{W_i\}_{i=1}^{E}$. A master wants to perform a matrix multiplication $\mathbf{C} = \mathbf{AB}$ on input matrices $\mathbf{A} \in \mathbb{F}_q^{a \times b}$ and $\mathbf{B} \in \mathbb{F}_q^{b \times c}$ for a sufficiently large finite field $\mathbb{F}_q$. The master generates encoded matrices using $\mathbf{A}$ and $\mathbf{B}$, then distributes them to workers to allocate tasks. To be specific, the master encodes $\tilde{\mathbf{A}}_i$, $\tilde{\mathbf{B}}_i$ and sends them to W_i for $i \in [1 : E]$. After receiving $\tilde{\mathbf{A}}_i$ and $\tilde{\mathbf{B}}_i$, each worker W_i computes the allocated task

We assume that some of the workers could be adversarial workers or stragglers. We list our assumptions on adversarial workers and stragglers as follows:

- There exist l adversarial workers $W_{\mathcal{L}}$, where $\mathcal{L}$ denotes a set of adversarial workers for $\mathcal{L} \subset [1 : E]$ and $|\mathcal{L}| = l$.
- Stragglers occur randomly depending on the network condition and currently available computing resource at each worker.
- Adversarial workers have knowledge about input matrices $\mathbf{A}$ and $\mathbf{B}$, and all the protocol of distributed computing including encoding, task allocation, and decoding, only except the random key $\mathbf{r}$ that is used for verification.[2]
- A master cannot be aware of the number of adversarial workers in advance.
- Adversarial workers cannot collude with each other, which implies that they cannot access the information about assigned tasks of other workers.

To improve the performance of distributed computing systems under adversarial attacks and stragglers, we aim to reduce verification time and overall processing time of distributed matrix multiplication.

3 Background: Verifiable Computing and Coded Computing

In this section, we first introduce two methods that are leveraged in GVCC: i) verifiable computing that enables identifying adversarial workers and ii) coded computing that mitigates the effects of stragglers.

[2]Protecting the random key from the adversarial workers can be realized by randomly generating $\mathbf{r}$ at the master, after receiving results from the workers.

3.1 VERIFIABLE COMPUTING FOR ADVERSARIAL ATTACK

The authors of Freivalds (1977) proposed a verification method (Freivalds' method) that uses a random key to verify the computation results. Freivalds' method for matrix multiplication can be summarized in three steps as follows.

Computation assignment: A master assigns a matrix multiplication task $\mathbf{C} = \mathbf{AB}$ for $\mathbf{A} \in \mathbb{F}_q^{a \times b}$ and $\mathbf{B} \in \mathbb{F}_q^{b \times c}$ to an available worker. The worker executes the assigned task, and returns its result $\mathbf{C}'$ to the master.

Verification key generation: The master generates a random key $\mathbf{r} \in \mathbb{F}_q^{1 \times a}$ and computes a verification key $\mathbf{s}_1 = \mathbf{rA}$.

Correctness check: The master computes $\mathbf{s}_2 = \mathbf{s}_1\mathbf{B} = \mathbf{rAB} = \mathbf{rC}$ and $\mathbf{s}_2{}' = \mathbf{rC}'$ from the received result. By comparing $\mathbf{s}_2$ and $\mathbf{s}_2'$, the master can verify whether $\mathbf{C}'$ is correct or not.

It has been known that the master can identify adversarial workers using this verification process with probability $1 - \frac{1}{q}$ in the finite field $\mathbb{F}_q$ (Sahraei & Avestimehr, 2019). In this verification procedure, the master computes $ab + bc + ca$ scalar products while the original task $\mathbf{C} = \mathbf{AB}$ requires abc scalar products.

3.2 CODED COMPUTING FOR STRAGGLERS

In distributed computing, several methods have been proposed to mitigate straggler effects. One traditional method is to replicate a computational task and allocate it to multiple workers (Wang et al., 2014). However, this replication-based method is inefficient as it requires K additional workers to tolerate a single straggler, when the computational task is divided into K smaller tasks. To handle straggler effects more efficiently, coded computing was suggested in Lee et al. (2018). By using a coding theoretic approach for allocation of tasks to workers, the master can retrieve the final product after receiving enough number of workers. We now define an important metric that has been used in many coded computing researches.

Recovery threshold R: The minimum number of computation results of tasks from non-adversarial workers that the master requires to obtain the final product for the worst-case scenario.

In coded computing, the master can decode the final output from R out of E results and it is able to tolerate up to $E - R$ stragglers. Therefore, coded computing requires S additional workers for handling S stragglers, while the replication-based method requires KS additional workers for S stragglers.

4 GROUP-WISE VERIFIABLE CODED COMPUTING

In this section, we provide GVC and propose an algorithm for group-wise verification. We also suggest a modified algorithm for GVCC under straggler effects.

4.1 ENCODING OF GROUP-WISE VERIFIABLE CODES (GVC)

We now explain the encoding procedure of GVC that facilitates group-wise verification at the master. In GVC, the master divides input matrices $\mathbf{A}$ and $\mathbf{B}$ into sub-matrices of equal size $\mathbf{A}_w \in \mathbb{F}_q^{\frac{a}{m} \times b}$ for $w \in [1:m]$, and $\mathbf{B}_z \in \mathbb{F}_q^{b \times \frac{c}{n}}$ for $z \in [1:n]$. Then input matrices and their product $\mathbf{C} = \mathbf{AB}$ are given by

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 \\ \vdots \\ \mathbf{A}_m \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_1 & \cdots & \mathbf{B}_n \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} \mathbf{A}_1\mathbf{B}_1 & \cdots & \mathbf{A}_1\mathbf{B}_n \\ \vdots & \ddots & \vdots \\ \mathbf{A}_m\mathbf{B}_1 & \cdots & \mathbf{A}_m\mathbf{B}_n \end{bmatrix}. \tag{1}$$

To generate the encoded matrices, the master uses encoding functions $\mathbf{p_A}$ and $\mathbf{p_B}$ constructed by using sub-blocks of $\mathbf{A}$ and $\mathbf{B}$ as coefficients, which are given by

$$\mathbf{p_A}(x) = \Sigma_{i=1}^{m} \mathbf{A}_i x^{i-1}, \quad \mathbf{p_B}(x) = \Sigma_{j=1}^{n} \mathbf{B}_j f^{j-1}(x), \tag{2}$$

where x represents the variable of encoding functions $\mathbf{p_A}$ and $\mathbf{p_B}$. $f(x)$ is the basic building component for constructing $\mathbf{p_B}$ and we now introduce the conditions on $f(x)$ that enable group-wise verification of GVC.[3]

Condition I : The polynomial function $f(x)$ used in the encoding function $\mathbf{p_B}(x)$ should satisfy the following conditions.

i) $f(x)$ is an mth order polynomial function.

ii) $f(x)$ has at least $\frac{E}{m}$ sets of m evaluation points $\alpha_{1,1}, \alpha_{1,2}, ..., \alpha_{\frac{E}{m},m}$ that satisfy $f(\alpha_{t,1}) = f(\alpha_{t,2}) = \cdots = f(\alpha_{t,m}) = w_t$ for $t \in \left[1 : \frac{E}{m}\right]$.

Using the encoding functions and evaluation points satisfying Condition I, the master generates the encoded matrices $\tilde{\mathbf{A}}_{t,u}$ and $\tilde{\mathbf{B}}_{t,u}$ for $t \in \left[1 : \frac{E}{m}\right]$ and $u \in [1 : m]$ as

$$\begin{aligned} \tilde{\mathbf{A}}_{t,u} &= \mathbf{p_A}(\alpha_{t,u}) = \Sigma_{i=1}^{m} \mathbf{A}_i \alpha_{t,u}^{i-1}, \\ \tilde{\mathbf{B}}_{t,u} &= \mathbf{p_B}(\alpha_{t,u}) = \Sigma_{j=1}^{n} \mathbf{B}_j f^{j-1}(\alpha_{t,u}) = \Sigma_{j=1}^{n} \mathbf{B}_j w_t^{j-1} = \tilde{\mathbf{B}}_{t,*}, \end{aligned} \tag{3}$$

where $w_t = f(\alpha_{t,u})$ for $u \in [1 : m]$, which are the values of encoding functions $\mathbf{p_A}$ and $\mathbf{p_B}$ at the evaluation points $x = \alpha_{t,u}$.

Remark 1. (Quantization and embedding into the finite field) In this paper, we assume that input matrices are defined on a finite field $\mathbb{F}_q$, while the dataset for machine learning and polynomial function $f(x)$ used for encoding are based on the real field $\mathbb{R}$. To express real field data and function in the finite field, we quantize matrices $\mathbf{A}, \mathbf{B}$, and $f(x)$ using v bits, and embed them into a finite field $\mathbb{F}_q$ of integers at modulo a prime q. Quantization of real numbers and embedding into the finite field have been used in several previous works (Yu et al., 2017; Ji et al., 2021; Tang et al., 2022). To analyze the effect of quantization in training of the deep neural network, we provide an experiment in Appendix B.

Remark 2. (Polynomials for GVC) GVC can be constructed using any polynomial satisfying Condition I. For instance, Chebyshev polynomial, which has been used in several coded computing studies (Fahim & Cadambe, 2021; Hong et al., 2021), satisfies Condition I and thus can be used as $f(x)$. We provide the proof in Appendix C.

Remark 3. (Reduced Computational overhead for encoding) In GVC, since it is guaranteed $\tilde{\mathbf{B}}_{t,u} = \tilde{\mathbf{B}}_{t,*}, \forall u \in [1 : m]$ for $t \in \left[1 : \frac{E}{m}\right]$, the master encodes $(1 + \frac{1}{m})E$ matrices to assign E tasks, instead of encoding $2E$ matrices as in the conventional methods. Thus, the computational overhead for encoding can be lowered by GVC.

4.2 TASK ASSIGNMENT AND COMPUTING AT WORKERS

To assign the tasks to workers, the master randomly rearranges workers into $\frac{E}{m}$ groups of the same size, which can be denoted as $W_{t,u}$ for $t \in \left[1 : \frac{E}{m}\right]$ and $u \in [1 : m]$, and distributes $\tilde{\mathbf{A}}_{t,u}$ and $\tilde{\mathbf{B}}_{t,u}$ to $W_{t,u}$.

Each worker $W_{t,u}$ computes the assigned task $\tilde{\mathbf{C}}_{t,u} = \tilde{\mathbf{A}}_{t,u}\tilde{\mathbf{B}}_{t,u}$, which corresponds to the value of the evaluation function $\mathbf{p_C}(x)$ at the evaluation point $x = \alpha_{t,u}$, where $\mathbf{p_C}(x)$ is given by

$$\mathbf{p_C}(x) = \mathbf{p_A}(x) \times \mathbf{p_B}(x) = \Sigma_{i=1}^{m} \Sigma_{j=1}^{n} \mathbf{A}_i \mathbf{B}_j x^{i-1} f^{j-1}(x). \tag{4}$$

After each worker finishes the assigned task, it returns its computation result to the master.

4.3 GROUP-WISE VERIFICATION FOR GVC

After receiving the results from workers, the master begins a verification process. We first explain a group-wise verification trial process and suggest two-stage verification algorithm based on this process.

[3]The condition for $f(x)$ was also used in squeezed polynomial codes (Hong et al., 2020), in order to reduce communication load in distributed computing.

4.3.1 A GROUP-WISE VERIFICATION TRIAL (GVT)

Let us denote workers with finished computation among $\{W_{t,u}\}_{u=1}^{m}$ by $\mathcal{T}_t$ for $t \in [1 : \frac{E}{m}]$. By GVC, a master can verify results from group $\mathcal{T}_t$ in a single trial. In conventional verifiable computing approach, a master verifies the computation result $\tilde{\mathbf{C}}_i^{'}$ for $i \in [1 : E]$ individually. However, if the tasks are encoded by GVC, it is ensured that $\Sigma_{\mathcal{T}_t}\tilde{\mathbf{C}}_{t,u} = \Sigma_{\mathcal{T}_t}\tilde{\mathbf{A}}_{t,u}\tilde{\mathbf{B}}_{t,u} = \Sigma_{\mathcal{T}_t}\tilde{\mathbf{A}}_{t,u}\tilde{\mathbf{B}}_{t,*} = \Sigma_{\mathcal{T}_t}\tilde{\mathbf{A}}_{t,u} \times \tilde{\mathbf{B}}_{t,*}$ for a group of workers $\mathcal{T}_t$ among $\{W_{t,u}\}_{u=1}^{m}$ where $|\mathcal{T}_t| = k$ ($k \leq m$). Therefore, the master can verify k computation results $\tilde{\mathbf{A}}_{t,u}\tilde{\mathbf{B}}_{t,u}$ in a group by verifying $\Sigma_{\mathcal{T}_t}\tilde{\mathbf{A}}_{t,u} \times \tilde{\mathbf{B}}_{t,*}$ instead. On the basis of this fact, we suggest a process for group-wise verification trial (GVT).

Verification key generation: For each trial of group-wise verification, a master generates a random key $\bar{\mathbf{r}}_t \in \mathbb{F}^{1\times\frac{a}{m}}$ and computes a verification key $\bar{\mathbf{s}}_{1,t} = \bar{\mathbf{r}}_t\Sigma_{\mathcal{T}_t}\tilde{\mathbf{A}}_{t,u}$.

Group-wise Correctness check: The master computes $\bar{\mathbf{s}}_{2,t} = \frac{1}{k}\bar{\mathbf{s}}_{1,t}\Sigma_{\mathcal{T}_t}\tilde{\mathbf{B}}_{t,u} = \bar{\mathbf{s}}_{1,t} \times \tilde{\mathbf{B}}_{t,*} = \bar{\mathbf{r}}_t\Sigma_{\mathcal{T}_t}\tilde{\mathbf{A}}_{t,u} \times \tilde{\mathbf{B}}_{t,*} = \bar{\mathbf{r}}_t\Sigma_{\mathcal{T}_t}(\tilde{\mathbf{A}}_{t,u}\tilde{\mathbf{B}}_{t,u}) = \bar{\mathbf{r}}_t\Sigma_{\mathcal{T}_t}\tilde{\mathbf{C}}_{t,u}$ and $\bar{\mathbf{s}}'_{2,t} = \bar{\mathbf{r}}_t\Sigma_{\mathcal{T}_t}\tilde{\mathbf{C}}'_{t,u}$ from the received computation results. If $\bar{\mathbf{s}}_{2,t} \neq \bar{\mathbf{s}}'_{2,t}$, then the master marks the output of verification trial as **Positive**, which implies there exist more than one adversarial worker in the tested group $\mathcal{T}_t$, and marks as **Negative** otherwise.[4]

Remark 4. (Computational complexity for verification) Each group-wise verification trial requires $\left(\frac{ab}{m} + \frac{bc}{n} + \frac{ac}{mn}\right)$ scalar-product calculations. This is the same computational complexity for verifying a single computation result in AVCC (Tang et al., 2022), while up to m computation results can be verified at once in a group-wise verification for GVC.

4.3.2 TWO-STAGE VERIFICATION ALGORITHM

We now provide a two-stage verification algorithm of GVC, given by Algorithm 1.[5]

Algorithm 1: Group-wise Verifiable Coded Computing

Two-stage group-wise verification: After receiving all computation results from workers, the master begins group-wise verification.
- **for** $t \in [1 : \frac{E}{m}]$ **do**
  - The master verifies m workers using GVT in Section 4.3.1, where $\mathcal{T}_t = \{W_{t,u}\}_{u=1}^{m}$. `// Stage 1`
  - **if** *Group-wise verification result for $\mathcal{T}_t$ is negative* **then**
    - Put all workers in $\mathcal{T}_t$ to a set of non-adversarial workers $\mathcal{T}^n$.
  - **else**
    - Put all workers in $\mathcal{T}_t$ to a set of possible adversarial workers $\mathcal{T}^p$.
  - **if** $|\mathcal{T}^n| \geq R$ **then**
    - break;
- **if** $|\mathcal{T}^n| < R$ **then**
  - The master verifies each worker in $\mathcal{T}^p$ using GVT for $k = 1$, and put non-adversarial workers whose verification result is negative to $\mathcal{T}^n$ until $|\mathcal{T}^n| = R$. `// Stage 2`

Two-stage group-wise verification: After receiving all computation results of tasks from workers, the master starts group-wise verification based on GVT. However, one thing we should take into account is that GVC can verify up to m results at once. Thus, we propose a two-stage group-wise verification algorithm adjusted to the parameter m.

- **Stage 1:** The master groups $\{W_{t,u}\}_{u=1}^{m}$ as $\mathcal{T}_t$ for $t \in [1 : \frac{E}{m}]$. The master performs a group-wise verification on $\mathcal{T}_t$ using GVT.

[4]It should be noted that group-wise verification trial has the same accuracy with the individual verification in Freivalds (1977). We demonstrate this by experiment under various adversarial attacks in Appendix D.

[5]We do not consider straggler effects here for simplicity, but provide a modified verification algorithm of GVC that considers straggler effects in Section 4.5.

- **Stage 2:** The master verifies each computation result in $\mathcal{T}_t$ if group-wise verification result for $\mathcal{T}_t$ is positive, until it achieves R non-adversarial workers to obtain $\mathbf{C}$.

4.4 Decoding at the master

After verifying R non-adversarial computation results, the master starts decoding. As we can see in equation 1 and equation 4, $\mathbf{A}_i\mathbf{B}_j$ for $i \in [1:m]$, $j \in [1:n]$ can be retrieved from $\mathbf{p_C}(x)$ to achieve $\mathbf{C}$. Since $\tilde{\mathbf{C}}_{t,u} = \mathbf{p_C}(\alpha_{t,u})$ and $\mathbf{p_C}(x)$ is a polynomial of degree $mn-1$, it can be interpolated from mn distinct value of $\tilde{\mathbf{C}}_{t,u}$. Hence, the master needs any fastest mn non-adversarial computation results to decode the final output $\mathbf{C}$. Thus, the recovery threshold of GVC is given by

$$R = mn. \tag{5}$$

Decoding can be done by the inversion of the coefficient matrix in

$$\begin{bmatrix} \tilde{\mathbf{C}}_1 \\ \vdots \\ \tilde{\mathbf{C}}_{mn} \end{bmatrix} = \left(\begin{bmatrix} {\beta_1}^0 & \dots & {\beta_1}^{m-1} & f(\beta_1) & \dots & {\beta_1}^{m-1} f(\beta_1)^{n-1} \\ \vdots & & & \ddots & & \vdots \\ {\beta_{mn}}^0 & \dots & {\beta_{mn}}^{m-1} & f(\beta_{mn}) & \dots & {\beta_{mn}}^{m-1} f(\beta_{mn})^{n-1} \end{bmatrix} \otimes I_{\frac{a}{m} \times \frac{a}{m}} \right) \times \begin{bmatrix} \mathbf{A}_1\mathbf{B}_1 \\ \vdots \\ \mathbf{A}_m\mathbf{B}_n \end{bmatrix}, \tag{6}$$

where $\tilde{\mathbf{C}}_k$ and β_k for $k \in [1:mn]$ denote the fastest mn computation results and the evaluation points of them, respectively and $\otimes$ denotes Kronecker product. It can be also achieved by interpolation (Kedlaya & Umans, 2011) of $\mathbf{p_C}(x)$ and extracting $\mathbf{A}_i\mathbf{B}_j$ from $\mathbf{p_C}(x)$ by repeated division of $f(x)$. We provide more detailed decoding process in Appendix E.

Remark 5. (Optimal Recovery threshold) It has been known in Yu et al. (2017) that the optimal recovery threshold for coded matrix multiplication is mn when $\mathbf{A}$ and $\mathbf{B}$ are divided into m and n partitions in a row-wise and a column-wise manner, respectively. Thus *GVC achieves the optimal recovery threshold*.

Remark 6. (Upper bound of tolerating adversarial attacks) Since GVCC requires mn computation results to obtain the final product, GVCC can tolerate up to $l_{upper} = E - S - mn$ adversarial workers, where S denotes the number of stragglers.

4.5 Modified verification algorithm of GVCC under Straggler Effects

In Section 4.3.2, the master starts a group-wise verification when all the computation results are received from workers. We now provide a modified algorithm that can be used under straggler effects, where the master performs a group-wise verification although part of the computation results are not received yet.

Two-stage group-wise verification in modified GVCC:

- **Stage 1**: The master groups $\{W_{t,u}\}_{u=1}^{m}$ as $\mathcal{T}_t$ for $t \in [1 : \frac{E}{m}]$ as in GVCC. The master establishes a certain threshold z. The master starts a group-wise verification for the computation results of all groups received up to that time when the master receives z computation results from workers.
- **Stage 2**: The master first individually verifies the computation results that arrived late, and verifies each computation result in $\mathcal{T}_t$ if group-wise verification result for $\mathcal{T}_t$ is positive, until it finds R non-adversarial workers to obtain $\mathbf{C}$.

By setting a proper threshold z, the modified verification algorithm can efficiently deal with straggler effects. However, it may lose some of the benefits of group-wise verification as a trade-off because the master verifies a smaller group of workers in a single trial. The impact on overall processing time will be analyzed in Section 6 via experiment.

5 Related Works

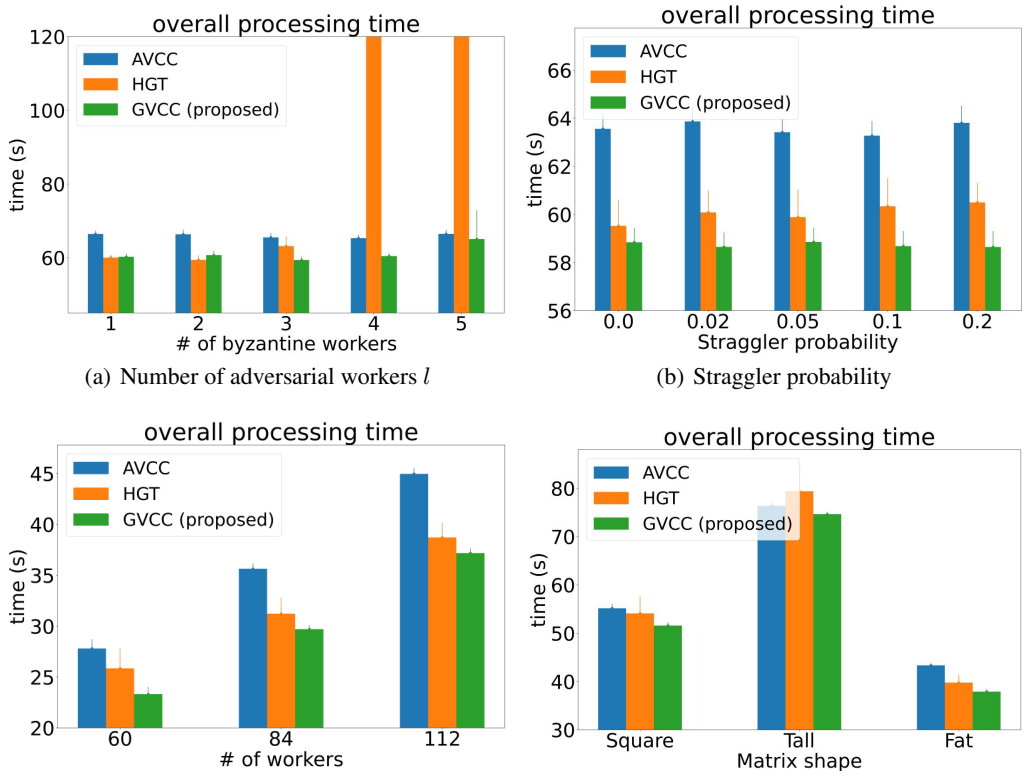
In distributed computing, there have been several studies on handling adversarial attacks and straggler effects simultaneously. First, to have tolerance over adversarial attacks and provide

resiliency to straggler effects, Lagrange coded computing (LCC) has been suggested in Yu et al. (2019). In LCC, a master requires $2K$ additional non-adversarial workers to be robust against K adversarial workers. In addition, a list-decoding approach has been applied to distributed matrix multiplication in Subramaniam et al. (2019), and it reduces the additional cost of tolerating adversarial attacks by a factor of two asymptotically compared to LCC. However, the computational complexity to perform list-decoding increases quadratically with the number of workers.

Recently, verifiable computing has been introduced to coded computing schemes. The authors of Tang et al. (2022) have suggested AVCC to counter straggler and adversarial workers. In AVCC, a master only requires K additional computation results from workers if there exist K adversarial workers, because it can verify each computation result using a verification key. However, since the master can verify only a single computation result at a time, the verification process can be a bottleneck for overall processing in distributed computing if the master utilizes numerous workers in the system.

In addition, there have been several studies that are based on a group-wise identification to find adversarial workers, while mitigating straggler effects using coded matrix multiplication. First, group testing algorithm has been applied to coded computing in Solanki et al. (2019). However, since it has a constraint on the minimum testable size for group testing, the number of test trials can be significantly large. To mitigate this limitation, hierarchical group testing (HGT) has been proposed in Hong et al. (2022). However, HGT imposes high computational complexity on a master since it requires matrix inversion or Reed-Solomon decoding to identify adversarial workers, whereas verifiable computing requires matrix-vector multiplications to verify them.

6 Experimental Results



(a) Number of adversarial workers l

(b) Straggler probability

(c) Number of workers

(d) Matrix shape

Figure 2: Overall processing time of AVCC, HGT, and GVCC (proposed).

In this section, we provide the performance of GVCC and modified GVCC in terms of overall processing time, and compare with the two existing schemes, AVCC (Tang et al., 2022) and HGT

(Hong et al., 2022). We run experiments on distributed matrix multiplication in a cluster of Amazon EC2 cloud, and use one t2.2xlarge node as a master and use t2.micro nodes as workers. Implementation is done by MPI4py (Dalcín et al., 2005), which is a well-known tool for distributed computing and matrices are randomly generated by NumPy. We set the verification time limit as 100 seconds and if the verification process takes more than 100 seconds, we terminate the experiment and mark it as a failure. We run four experiments and evaluate the overall processing time. Each experiment is repeated 20 times and the result is depicted in Fig. 2. We also list stage-wise processing time (encoding (Enc), task assignment (SA), computation & return (CR), verification (Ver), decoding (Dec), and overall processing times) of experiments in Appendix F.1.

In Fig. 2(a), we compare overall processing time with different number of adversarial workers l, when $\mathbf{A} \in \mathbb{F}_q^{3500\times 6300}, \mathbf{B} \in \mathbb{F}_q^{6300\times 3600}, m = 5, n = 9$, and $E = 60$. GVCC speeds up overall processing time compared to AVCC for all cases, and also shows faster overall processing than HGT when $l \geq 3$. As we have claimed in Remark 3, the computational overhead for encoding the tasks can lowered by GVC, which results in reduced encoding and overall processing time compared to AVCC. GVCC can also reduce verification time than AVCC with the help of two-stage group-wise verification. Furthermore, for the verification time of HGT increases significantly with the number of workers, the verification and overall processing time of GVCC is much lower than HGT when $l \geq 3$. To be specific, GVCC speeds up verification time up to $\times 3.34$ than AVCC, and $\times 174$ than HGT, and HGT even fails to proceed verification process in the given time limit (100s) when $l = 5$.

In Fig. 2(b), we show the overall processing time with stragglers in distributed computing systems, where $\mathbf{A} \in \mathbb{F}_q^{3500\times 6300}, \mathbf{B} \in \mathbb{F}_q^{6300\times 3600}, m = 5, n = 9, E = 60$, and $l = 3$. To simulate straggler effects in distributed computing, we randomly pick stragglers with probabilities $p = 0, 0.02, 0.05, 0.1$, and 0.2. Stragglers are forced to run background thread which slows down the execution of assigned tasks. In this experiment, we use the modified algorithm in Section 4.5 for GVCC ($z = R = 63$) and compare with AVCC and HGT. Since verification is done asynchronously for the received computation results in GVCC, verification time is added to computation & return time. It is observed that GVCC outperforms AVCC and HGT for all cases. Thus, it is claimed that GVCC can efficiently handle straggler effects while leveraging group-wise verification approach. We also provide additional experiment with modified algorithm on another parameter setting in Appendix F.2.

The performance of GVCC with the different number of workers is depicted in Fig. 2(c). We run experiments to multiply two input matrices $\mathbf{A} \in \mathbb{F}_q^{2100\times 4000}$ and $\mathbf{B} \in \mathbb{F}_q^{4000\times 3861}$, with three parameter setting (E, m, n): i) $(60, 5, 9)$, ii) $(84, 6, 11)$, iii) $(112, 7, 13)$, where $l = 3$. GVCC achieves the lowest overall processing time in every setting. To be specific, GVCC speeds up verification time up to $\times 3.5$ than AVCC and $\times 57.29$ than HGT. Furthermore, GVCC speeds up overall processing time up to $\times 1.20$ than AVCC and $\times 1.10$ than HGT. It should be noted that, verification time of GVCC remains almost the same in all three settings, while the verification time of AVCC increases linearly with the number of workers, which demonstrates the effectiveness of GVC in large-scale distributed computing systems.

In Fig. 2(d), we compare overall processing time with different shape of input matrices with three parameter settings: i) Square ($\mathbf{A} \in \mathbb{F}_q^{3600\times 3600}$, $\mathbf{B} \in \mathbb{F}_q^{3600\times 3600}$), ii) Tall ($\mathbf{A} \in \mathbb{F}_q^{4500\times 1500}$, $\mathbf{B} \in \mathbb{F}_q^{1500\times 4500}$), iii) Fat ($\mathbf{A} \in \mathbb{F}_q^{2700\times 7000}$, $\mathbf{B} \in \mathbb{F}_q^{7000\times 2700}$), where $m = 5, n = 9, l = 3$, and $E = 60$. GVC speeds up overall processing time compared to AVCC and HGT, regardless of matrix shapes.

7 Conclusion

In this paper, we propose GVCC, a robust and fast distributed computing scheme for matrix multiplication that can deal with two important problems: adversarial attacks and straggler effects. By combining the verifiable computing with coded computing, GVCC allows a master to verify a group of computation results at a time, reducing verification time to find adversarial workers and provide resiliency to straggler effects simultaneously. We also suggest a two-stage verification algorithm for GVCC and a modified algorithm that can be used under straggler effects. Consequently, GVCC is shown to have significant reduction in verification and overall processing time, and experimental results demonstrate the effectiveness of GVCC in distributed computing systems.

REFERENCES

Dustin Anderson, Jean-Roch Vlimant, and Maria Spiropulu. An mpi-based python framework for distributed training with keras. *arXiv preprint arXiv:1712.05878*, 2017.

Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108, pp. 2938–2948. PMLR, 26–28 Aug 2020.

Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *ICML*, 2012.

Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. In *Proceedings of Machine Learning and Systems*, volume 1, pp. 374–388, 2019.

Lisandro Dalcín, Rodrigo Paz, and Mario Storti. Mpi for python. *Journal of Parallel and Distributed Computing*, 65(9):1108–1115, 2005.

Lisandro Dalcín, Rodrigo Paz, Pablo Kler, and Alejandro Cosimo. Parallel distributed computing using python. *Advances in Water Resources*, 34(9):1124–1139, 2011. ISSN 0309-1708.

Jeffrey Dean and Luiz André Barroso. The tail at scale. *Commun. ACM*, 56(2):74–80, 2013.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc' aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc Le, and Andrew Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

Sanghamitra Dutta, Viveck Cadambe, and Pulkit Grover. Short-dot: Computing large linear transforms distributedly using coded short dot products. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

Sanghamitra Dutta, Mohammad Fahim, Farzin Haddadpour, Haewon Jeong, Viveck Cadambe, and Pulkit Grover. On the optimal recovery threshold of coded matrix multiplication. *IEEE Transactions on Information Theory*, 66(1):278–301, 2019.

El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. The hidden vulnerability of distributed learning in Byzantium. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 3521–3530. PMLR, 10–15 Jul 2018.

Mohammad Fahim and Viveck R. Cadambe. Numerically stable polynomially coded computing. *IEEE Transactions on Information Theory*, 67(5):2758–2785, 2021.

Rusins Freivalds. Probabilistic machines can use less running time. In *IFIP congress*, volume 839, 1977.

Sangwoo Hong, Heecheol Yang, and Jungwoo Lee. Squeezed polynomial codes: Communication-efficient coded computation in straggler-exploiting distributed matrix multiplication. *IEEE Access*, 8:190516–190528, 2020.

Sangwoo Hong, Heecheol Yang, Youngseok Yoon, Taehyun Cho, and Jungwoo Lee. Chebyshev polynomial codes: Task entanglement-based coding for distributed matrix multiplication. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pp. 4319–4327. PMLR, 18–24 Jul 2021.

Sangwoo Hong, Heecheol Yang, and Jungwoo Lee. Hierarchical group testing for byzantine attack identification in distributed matrix multiplication. *IEEE Journal on Selected Areas in Communications*, 40(3):1013–1029, 2022.

Peng Huang, Chuanxiong Guo, Lidong Zhou, Jacob R. Lorch, Yingnong Dang, Murali Chintalapati, and Randolph Yao. Gray failure: The achilles' heel of cloud-scale systems. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems, HotOS 2017, Whistler, BC, Canada, May 8-10, 2017*, pp. 150–155. ACM, 2017.

Ruowan Ji, Asit Kumar Pradhan, Anoosheh Heidarzadeh, and Krishna R. Narayanan. Squeezed random khatri-rao product codes. In *2021 IEEE Information Theory Workshop (ITW)*, pp. 1–6, 2021.

Kiran S Kedlaya and Christopher Umans. Fast polynomial factorization and modular composition. *SIAM Journal on Computing*, 40(6):1767–1802, 2011.

Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris S. Papailiopoulos, and Kannan Ramchandran. Speeding up distributed machine learning using codes. *IEEE Trans. Inf. Theory*, 64(3):1514–1529, 2018.

Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, volume 27, 2014.

Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020a.

Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020b.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54, pp. 1273–1282. PMLR, 20–22 Apr 2017.

Hyegyeong Park, Kangwook Lee, Jy-Yong Sohn, Changho Suh, and Jaekyun Moon. Hierarchical coding for distributed computing. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 1630–1634, 2018. doi: 10.1109/ISIT.2018.8437669.

Amirhossein Reisizadeh, Saurav Prakash, Ramtin Pedarsani, and Amir Salman Avestimehr. Coded computation over heterogeneous clusters. *IEEE Transactions on Information Theory*, 65(7):4227–4242, 2019.

Saeid Sahraei and A. Salman Avestimehr. Interpol: Information theoretically verifiable polynomial evaluation. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 1112–1116, 2019.

Jy-yong Sohn, Dong-Jun Han, Beongjun Choi, and Jaekyun Moon. Election coding for distributed learning: Protecting signsgd against byzantine attacks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 14615–14625, 2020.

Atnav Solanki, Martina Cardone, and Soheil Mohajer. Non-colluding attacks identification in distributed computing. In *2019 IEEE Information Theory Workshop (ITW)*, pp. 1–5, 2019.

Mahdi Soleymani, Ramy E. Ali, Hessam Mahdavifar, and A. Salman Avestimehr. List-decodable coded computing: Breaking the adversarial toleration barrier. *IEEE Journal on Selected Areas in Information Theory*, 2(3):867–878, 2021.

Adarsh M. Subramaniam, Anoosheh Heidarzadeh, and Krishna R. Narayanan. Collaborative decoding of polynomial codes for distributed computation. In *2019 IEEE Information Theory Workshop (ITW)*, pp. 1–5, 2019.

Rashish Tandon, Qi Lei, Alexandros G. Dimakis, and Nikos Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp. 3368–3376. PMLR, 06–11 Aug 2017.

Tingting Tang, Ramy E. Ali, Hanieh Hashemi, Tynan Gangwani, Salman Avestimehr, and Murali Annavaram. Adaptive verifiable coded computing: Towards fast, secure and private distributed machine learning. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 628–638, 2022.

Da Wang, Gauri Joshi, and Gregory Wornell. Efficient task replication for fast response times in parallel computation. In *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, pp. 599–600, 2014.

Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. Attack of the tails: Yes, you really can backdoor federated learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 16070–16084, 2020.

Qian Yu, Mohammad Ali Maddah-Ali, and A Salman Avestimehr. Polynomial codes: an optimal design for high-dimensional coded matrix multiplication. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 4406–4416, 2017.

Qian Yu, Songze Li, Netanel Raviv, Seyed Mohammadreza Mousavi Kalan, Mahdi Soltanolkotabi, and Salman A. Avestimehr. Lagrange coded computing: Optimal design for resiliency, security, and privacy. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89, pp. 1215–1225. PMLR, 16–18 Apr 2019.

Qian Yu, Mohammad Ali Maddah-Ali, and A Salman Avestimehr. Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding. *IEEE Transactions on Information Theory*, 66(3):1920–1933, 2020.

Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.