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.

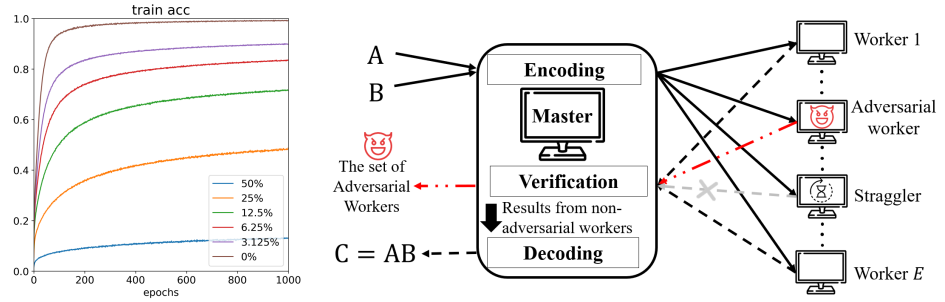
1 INTRODUCTION

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.

One possible option is distributed computing (Dalcín et al., 2005; 2011), where the workers compute partial computation task received from the master. In distributed computing, the master divides and distributes tasks (which require far small memory than the original task) to workers, and they compute the assigned tasks and send results back to a master. Distributed computing can be utilized to compute matrix multiplication in machine learning, the most important and frequent computation block. In a distributed setting, however, there exist two foremost considerations to embed distributed computing in machine learning applications, i) stragglers and ii) adversarial workers.

Stragglers are workers that return their computation results much slower than others. It has been reported that stragglers can be a serious bottleneck to performing 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). In coded computing, a master encodes a computation task with a coding technique while retaining redundancy in task allocation. Due to the redundancy arisen from the coding technique, a master does not need all results of tasks to achieve the 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).

Moreover, some of the workers could be adversarial workers, which send perturbed results to the master to contaminate or degrade the performance of neural networks. 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 the overall training process and



(a) Training curve of neural networks under attacks.

(b) Distributed computing for a matrix multiplication $\mathbf{C} = \mathbf{AB}$ adversarial under adversarial workers and stragglers.

Figure 1: Training curve of neural networks under adversarial workers and system model for distributed computing

severely degrade the 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.

To demonstrate the effect of adversarial workers in distributed computing, we trained a neural network under adversarial attack. Fig 1(a) shows the training curve of neural networks on CIFAR-100. 32 workers were used for training and adversarial workers send random values with the same magnitude as the original results during training. As we can see in the figure, the training accuracy of a neural network degraded severely when there exist adversarial workers. The training accuracy of a neural network converged to 0.99 after 560 epochs when there is no adversarial worker. However, with only 3.125% of adversarial workers, the training accuracy of the neural network was only 0.89 after 1000 epochs.[1]

To be robust against adversarial attacks, a master can check 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(b), 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 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.

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

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.

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.

[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.

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.[3]

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.[4]

[3] It should be noted that previous works have included stragglers or adversarial workers in the recovery threshold and in this case, the recovery threshold of GVC can be expressed as $R' = R + S + l$.

[4] 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.

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 $t \in [1 : \frac{E}{m}]$.

cUsing 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 $t \in [1 : \frac{E}{m}]$ 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), can be used as $f(x)$ by using quantization as stated in **Remark 1**. We also 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 $t \in [1 : \frac{E}{m}]$, 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 same size, which can be denoted as $W_{t,u}$ for $t \in [1 : \frac{E}{m}]$ and $u \in [1 : m]$, and distributes $\tilde{\mathbf{A}}_{t,u}$, $\tilde{\mathbf{B}}_{t,u}$ 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 process and suggest two-stage verification algorithm based on this process.

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.[5]

4.3.2 TWO-STAGE VERIFICATION ALGORITHM

We now provide a two-stage verification algorithm of GVC, given by Algorithm 1.[6]

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 $\mathcal{T}_t = \{W_{t,u}\}_{u=1}^m$.
    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 $\mathcal{T}^n$. `// Stage 1`
    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.
- **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 $\mathbf{p}_{\mathbf{C}}(x)$ to achieve $\mathbf{C}$. Since $\tilde{\mathbf{C}}_{t,u} = \mathbf{p}_{\mathbf{C}}(\alpha_{t,u})$ and $\mathbf{p}_{\mathbf{C}}(x)$ 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 $R = mn$.

[5]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.

[6]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.

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

$$\begin{bmatrix} \tilde{\mathbf{C}}_1 \\ \vdots \\ \tilde{\mathbf{C}}_{mn} \end{bmatrix} = \left(\begin{bmatrix} {\beta_1}^0 & \dots & {\beta_1}^{m-1} & f(\beta_1) & \dots & {\beta_1}^{m-1} f(\beta_1)^{n-1} \\ \vdots & & & \ddots & & \vdots \\ {\beta_{mn}}^0 & \dots & {\beta_{mn}}^{m-1} & f(\beta_{mn}) & \dots & {\beta_{mn}}^{m-1} f(\beta_{mn})^{n-1} \end{bmatrix} \otimes I_{\frac{a}{m} \times \frac{a}{m}} \right) \times \begin{bmatrix} \mathbf{A}_1\mathbf{B}_1 \\ \vdots \\ \mathbf{A}_m\mathbf{B}_n \end{bmatrix}, \tag{5}$$

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 $\mathbf{p}_{\mathbf{C}}(x)$ and extracting $\mathbf{A}_i\mathbf{B}_j$ from $\mathbf{p}_{\mathbf{C}}(x)$ by repeated division of $f(x)$. We provide more detailed decoding process in Appendix E.

Remark 4. (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 also achieves the optimal recovery threshold*.

Remark 5. (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

We now provide a modified algorithm that can be used under straggler effects.

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

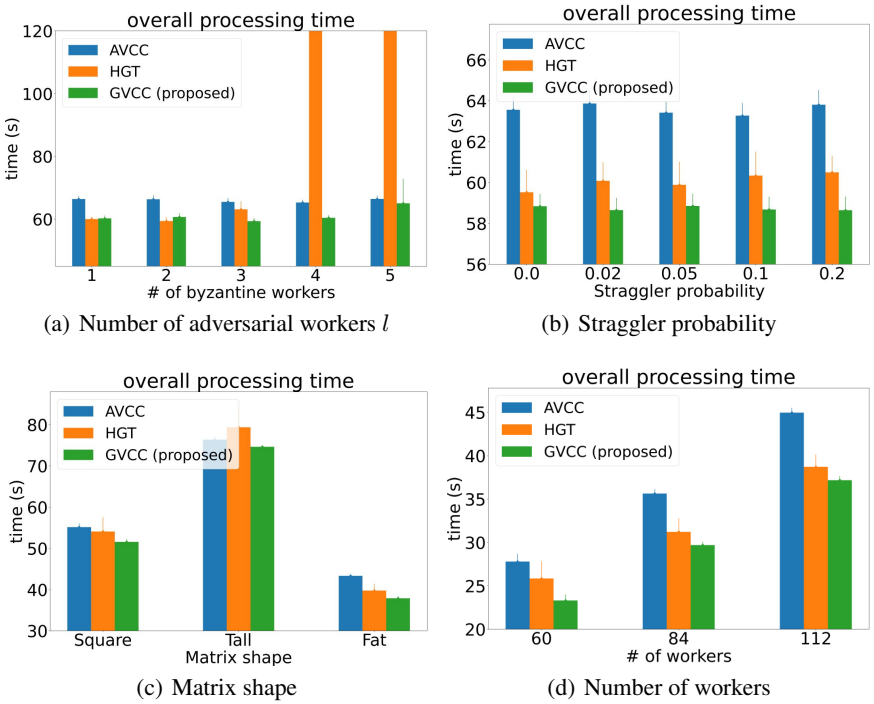
In distributed computing, there have been several studies on handling adversarial attacks and straggler effects simultaneously. First, to have tolerance over adversarial attacks and provide resiliency to straggler effects, Lagrange coded computing (LCC) has been suggested in Yu et al. (2019). In LCC, 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 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) Matrix shape

(d) Number of workers

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 proceed 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) 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. 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. As we can see in the Fig 2(b), GVCC outperforms AVCC and HGT for all cases. Thus, it is claimed that GVCC can efficiently handle straggler effects while leveraging a group-wise verification approach. We also provide additional experiments with a modified algorithm on another parameter setting in Appendix F.2.

In Fig. 2(c), we compare overall processing time with different shape of input matrices with three parameter settings: i) Square ($\mathbf{A} \in \mathbb{F}_q^{3600\times3600}$, $\mathbf{B} \in \mathbb{F}_q^{3600\times3600}$), ii) Tall ($\mathbf{A} \in \mathbb{F}_q^{4500\times1500}$, $\mathbf{B} \in \mathbb{F}_q^{1500\times4500}$), iii) Fat ($\mathbf{A} \in \mathbb{F}_q^{2700\times7000}$, $\mathbf{B} \in \mathbb{F}_q^{7000\times2700}$), 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.

The performance of GVCC with the different numbers of workers is depicted in Fig. 2(d). We run experiments to multiply two input matrices $\mathbf{A} \in \mathbb{F}_q^{2100\times4000}$ and $\mathbf{B} \in \mathbb{F}_q^{4000\times3861}$, 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.

Verification time (ms)	**Group-wise verifiable size m**			**Matrix Size N**		
	5	**6**	**7**	**900**	**1800**	**3600**
HGT	2750	1920	2440	2348	9554	42163
AVCC	140	170	210	16	45	163
GVC	48	48	46	5.6	17	62

Table 1: Verification time of GVC, AVCC, and HGT with different matrix size and m.

Furthermore, it should be noted that GVC shows more benefit than AVCC as N or m gets bigger. In specific, our methods have a verification complexity of $\mathbf{O}(\frac{R}{m}N^2)$ while AVCC has $\mathbf{O}(RN^2)$. To show the performance of GVC with different m and N, we indicate the verification time of GVC and the other schemes in table 1. We used parameter setting ((E, m, n): i) $(60, 5, 9)$, ii) $(84, 6, 11)$, iii) $(112, 7, 13)$, where $l = 3$ (which are same setting with experiment in Fig 2(d)). We indicate the verification time of HGT, AVCC, and GVC in Table 1. We can see that the verification time of GVC remains the same value, while other methods show an increase in the verification time. This is because other methods require the verification of more results as the recovery threshold increases, but GVC has the same number of groups that needs to be verified. Moreover, to show the benefits of GVC, we conduct additional experiments. We experimented with the same parameter setting in Fig 2(a) when $l = 3$ and only changed the size of the matrix ($m, n = 5, 9$, $E = 60$, $l = 3$ and $a = b = c = N$). We have changed the size of input matrix N from 900 to 3600 to show the benefit of GVC as N increases, and indicate the verification time in Table 1. As we can see in the table, verification gap between GVCC and other scheme becomes larger as N increases. In specific, HGT shows far larger verification time than GVC. Moreover, gap between AVCC and GVC increases $\times$2.66 when N becomes $\times$2, and $\times$9.36 when N becomes $\times$4.

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. 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

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.

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.

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.

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.

APPENDIX

We provide detailed experimental settings and results that are contained in the paper. We also provide proof of Remark 2, verification performance anlaysis of a group-wise verification trial (GVT) and detailed explanations about decoding procedure for GVC in this supplementary material. Then we provide additional experiments that shows the performance of GVCC under straggler effects.

A ADVERSARIAL ATTACK EXPERIMENT SETTING IN SECTION I.

To show the effect of adversarial workers in ML training, we experiment ML training on four famous datasets (MNIST, FMNIST, CIFAR-10, and CIFAR-100). We performed an experiment with four settings, where there exist 16 (50%), 8 (25%), 4 (12.5%), 2 (6.25%), and 1 (3.125%) adversarial workers among 32 workers. We use fully-connected neural networks with 2 hidden layers of size 32 for MNIST and FMNIST, and 3 hidden layers of size 200 for CIFAR-10 and CIFAR-100. The batch normalization was used for CIFAR-10 and CIFAR-100, and not for MNIST and FMNIST. ReLU function was used for non-linear activation. We also used a learning rate 0.001 with adam optimizer.

Since we have used batch of size 256, matrix multiplication for MNIST and FMNIST in the forward path is i) 256 by $(28 \times 28) \times (28 \times 28)$ by 32, ii) 256 by 32×32 by 32, and iii) 256 by 32×32 by 10 and forward path for CIFAR-10 and CIFAR-100 is i) 256 by $(3 \times 32 \times 32) \times (3 \times 32 \times 32)$ by 200, ii) 256 by 200×200 by 200, iii) 256 by 200×200 by 200 and iv) 256 by 200×200 by 10 or 100. These matrix multiplication tasks are distributed to workers. For matrix division, we used an uncoded setting for the experiments, where matrices are split row-wise and column-wise respectively. In each matrix multiplication, adversarial workers send a random noise matrix to the master, instead of the computation result. To be specific, random l blocks out of 32 blocks are random values from adversarial workers. The random noise follows a uniform distribution and has the same average power as the computation results, and more various attacks will be discussed in Appendix D.

The experiment was repeated 10 times and we indicate the average results in Table 2 and Fig 3. It was observed that only one adversarial worker (3.125%) during training degrades the performance of neural networks upto 10%. This performance degradation is more severe in complex datasets such as CIFAR100.

Therefore, it is essential to verify and exclude the adversarial workers during the ML training in a distributed computing system. ML training could be completed without adversarial workers by using GVCC (proposed), since GVCC verifies and excludes adversarial workers before decoding each matrix multiplication task. Therefore, GVCC can perfectly eliminate the effect of adversarial workers on ML training at a cost of additional verification time. (The verification performance analysis of GVCC is also given in Appendix D.)

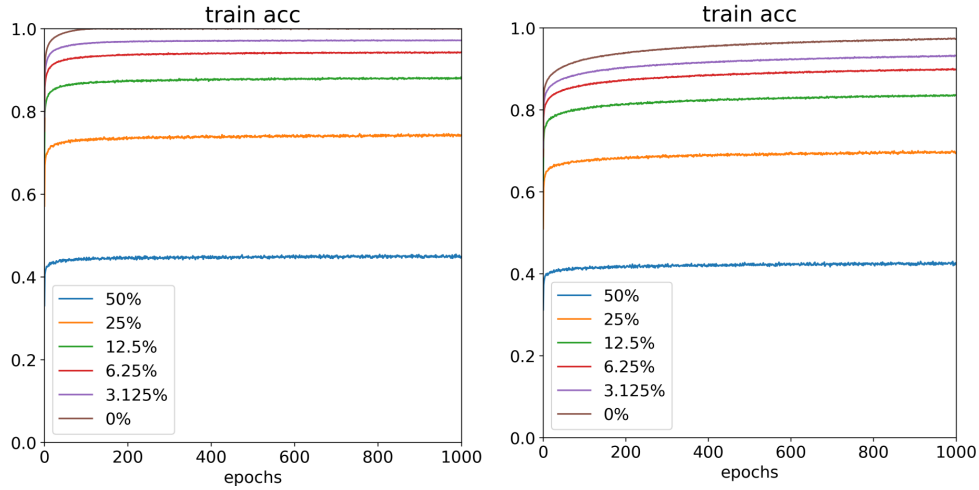
ratio of attacks (%)	**0**	**3.125**	**6.25**	**12.5**	**25**	**50**
CIFAR-10	0.999	0.959	0.923	0.849	0.651	0.28
CIFAR-100	0.994	0.899	0.836	0.714	0.489	0.131
MNIST	1	0.972	0.943	0.882	0.742	0.448
FMNIST	0.97	0.937	0.899	0.838	0.695	0.432

Table 2: Training accuracy after 1000 epochs under adversarial attacks

B MACHINE LEARNING WITH QUANTIZED VALUES. (REMARK 1)

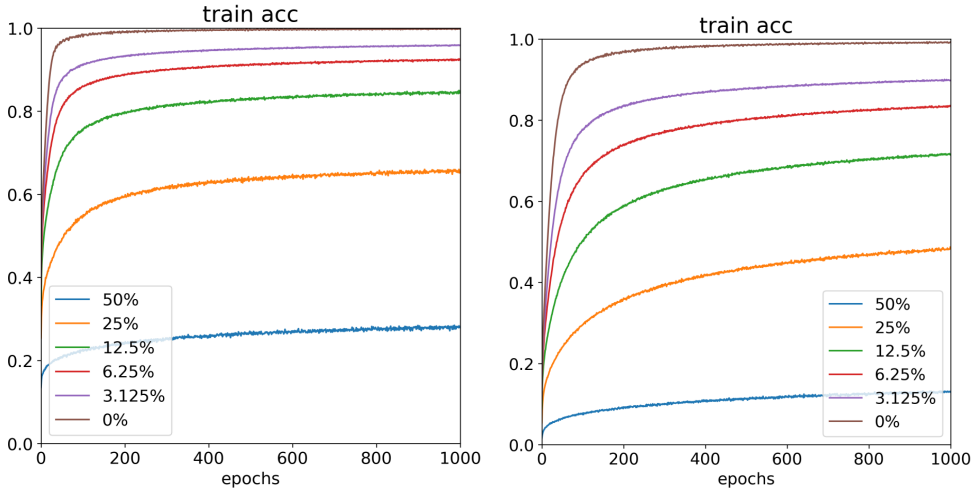
To use GVCC for machine learning, the input matrices and all the arithmetic operations should be in the same field. Therefore, We proceed an experiment that analyzes the accuracy of matrix multiplication after quantization and mapping the real matrices to the finite field.

To express real field data and function in the finite field, we quantize matrix $\mathbf{A}, \mathbf{B}$, and $f(x)$ using v bits as $\bar{\mathbf{A}} = \text{round}(\mathbf{A} \times 2^v)$, $\bar{\mathbf{B}} = \text{round}(\mathbf{B} \times 2^v)$, and $f_r(x) = \text{round}(f(x) \times 2^v)$, and embed into



(a) Training curve of neural networks under adversarial attacks on MNIST

(b) Training curve of neural networks under adversarial attacks on FMNIST.



(c) Training curve of neural networks under adversarial attacks on CIFAR-10.

(d) Training curve of neural networks under adversarial attacks on CIFAR-100.

Figure 3: Training curve of neural networks under adversarial workers and system model for distributed computing

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 previous works (Yu et al., 2017; Ji et al., 2021; Tang et al., 2022).

The final product $\mathbf{C}$ can be achieved using $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ by computing $(\bar{\mathbf{A}} \times \bar{\mathbf{B}})/2^{2v} = \bar{\mathbf{C}}$ ($\bar{\mathbf{C}} \in \mathbb{R}$). To analyze the relationship between error and the number of bits used for quantization, we derive mean square error (MSE) and normalized root mean square error (NRMSE) for 1000 by 1000 matrices $\mathbf{A}$ and $\mathbf{B}$ whose elements are in the real fields and follow normal and uniform distributions. The experiment results are indicated in the Table 3.

	# of bits used for quantization	0	1	2	4	8	16	32
Uniform distribution	MSE	79.831	15.388	3.558	0.218	0.000851	1.30E-08	3.01E-18
	NRMSE	0.00127	2.45E-04	5.69E-05	3.49E-06	1.36E-08	2.07E-13	4.79E-23
Normal distribution	MSE	173.619	42.097	10.437	0.651	0.00254	3.88E-08	9.03E-18
	NRMSE	0.173	0.0421	0.0104	0.000651	2.54E-06	3.88E-11	9.04E-21

Table 3: MSE and NRMSE of matrix multiplication after quantization

As can be seen in the table, the matrix multiplication can be done with very small errors by using more than 8 bits for quantization. Considering that computers use 16-bit or 32-bit precision for multiplication, it is guaranteed that GVC can proceed matrix multiplication with very high precision.

To further analyze the effect of quantization in ML use cases, we performed experiments on ML training and inference with quantization. We use fully-connected neural network with 2 hidden layers of size 64 for MNIST and FMNIST, and 3 hidden layers of size 200 for CIFAR10 and CIFAR100. And Relu function was used for non-linear activation. We experiment 10 times and indicate the average training accuracy and the number of epochs in the Table 4. As can be seen in the Table 4, the networks using 0, 1, and 2 bits did not trained at all regardless of the datasets. In MNIST and FMNIST, neural networks trained using 4 bits show more than 90% of performance compared to the original neural network (no quantization), and neural networks that are trained using more than 8 bits show almost the same convergence speed as the original neural network for all the datasets. On the other hand, CIFAR10 and CIFAR100 require more than 8 bits to show the same convergence speed as the original neural network. Therefore, we can conclude that GVC can be used in real ML cases by using more than 8 bits for quantization.

	# of bits used for quantization	No quantization	0	1	2	4	8	16
CIFAR-10	Train ACC	0.996	0.1	0.1	0.1	0.523	0.989	0.994
	# of epochs	79	-	-	-	182	92	82
CIFAR-100	Train ACC	0.978	0.01	0.01	0.01	0.182	0.969	0.976
	# of epochs	119	-	-	-	199	139	123
MNIST	Train ACC	0.999	0.1	0.1	0.1	0.998	0.999	0.999
	# of epochs	42	-	-	-	46	43	42
FMNIST	Train ACC	0.982	0.1	0.1	0.1	0.967	0.979	0.979
	# of epochs	190	-	-	-	199	198	198

Table 4: Train accuracy and number of epochs till convergence with quantized machine learning training

C Proof of Remark 2. (Using Chebyshev polynomial as $f(x)$)

In this subsection, we provide proof of Remark 2.

There exist two conditions for $f(x)$ and these conditions can be satisfied using Chebyshev polynomial for $f(x)$.

The first condition "(i) $f(x)$ is a mth order polynomial function" can be easily satisfied by using mth order Chebyshev polynomial for $f(x)$.

The second condition "(ii) $f(x)$ has at least $\frac{E}{m}$ sets of m 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 $t \in [1 : \frac{E}{m}]$." is satisfied using property of Chebyshev polynomials. A mth order Chebyshev polynomial has $m + 1$ extrema points for $x \in [-1 : 1]$ and all of the extrema have values that are either -1 or 1. We can find evaluation points for condition (ii) by setting $\frac{E}{m}$ distinct value for w_t for $t \in [1 : \frac{E}{m}]$, between the local minimum value (-1) and local maximum value (1). If we can always find m distinct roots of $f(x) = w_t$ for $t \in [1 : \frac{E}{m}]$, we can guarantee that there exist $\frac{E}{m}$ sets of m points that satisfies the condition (ii). And we can guarantee that there exist m distinct roots of $f(x) = w_t$ for $t \in [1 : \frac{E}{m}]$ by using the Intermediate Value Theorem. By applying the Intermediate Value Theorem for consecutive extrema points, we can guarantee that there exist a root for $f(x) = w_t$. Since there exist $m + 1$ consecutive extrema points, we can apply the Intermediate Value Theorem m times for different domain of x and find m distinct roots for $f(x) = w_t$.

This completes the proof.

D Verification performance analysis under various adversarial attacks.

In this section, we show the verification performance of group-wise verification under various adversarial attacks. The verification performance can be measured by the difference between $\bar{\mathbf{s}}_{2,t}$

and $\bar{\mathbf{s}}'_{2,t}$. We consider four types of attacks i) Random value attack (Normal), ii) Random value attack (Uniform), iii) Constant attack, and iv) Inverse value attack on two types of data distribution i) Normal, and ii) Uniform. We experiment by two matrices with size 3000 by 3000. The matrices and the contaminated results from adversarial attacks are randomly generated by NumPy. The contaminated result is generated with equal power to un-contaminated result. We verify matrix multiplication by group-wise verification trial in Section 4.3.1. To be specific, we first compute $\bar{\mathbf{s}}_{2,t} = \frac{1}{k}\bar{\mathbf{s}}_{1,t}\Sigma_{\mathcal{T}_t}\tilde{\mathbf{B}}_{t,u} = \bar{\mathbf{s}}_{1,t} \times \tilde{\mathbf{B}}_{t,*} = \bar{\mathbf{r}}_t\Sigma_{\mathcal{T}_t}\tilde{\mathbf{A}}_{t,u} \times \tilde{\mathbf{B}}_{t,*} = \bar{\mathbf{r}}_t\Sigma_{\mathcal{T}_t}(\tilde{\mathbf{A}}_{t,u}\tilde{\mathbf{B}}_{t,u}) = \bar{\mathbf{r}}_t\Sigma_{\mathcal{T}_t}\tilde{\mathbf{C}}_{t,u}$ and $\bar{\mathbf{s}}'_{2,t} = \bar{\mathbf{r}}_t\Sigma_{\mathcal{T}_t}\tilde{\mathbf{C}}'_{t,u}$. Then we compute the normalized mean square error (NRMSE) of $\bar{\mathbf{s}}_{2,t} - \bar{\mathbf{s}}'_{2,t}$. ($|\mathcal{T}_t| = k$) The results are indicated in the Table 5. As we can see in the table, NRMSE is not effected by k, which implies group-wise verification process has the same accuracy with individual verification ($k = 1$) in (Freivalds, 1977). Furthermore, for NRMSE tends to have a very high value for all cases, the master can verify the results with extremely high probability by setting a threshold for NRMSE as 0.01.

k (# of verified results at once)		1	3	5	7	10
Adversarial attack types	Distribution of the matrix					
Random value attack (Normal)	Normal	6.10E+03	6.00E+03	6.00E+03	6.00E+03	6.00E+03
	Uniform	2.23E+06	2.24E+06	2.26E+06	2.25E+06	2.21E+06
Random value attack (Uniform)	Normal	6.31E+03	5.77E+03	6.07E+03	6.13E+03	5.79E+03
	Uniform	4.11E+04	4.06E+04	4.08E+04	4.12E+04	4.12E+04
Constant attack	Normal	6.20E+03	7.09E+03	6.29E+03	5.71E+03	4.92E+03
	Uniform	2.49E+02	2.48E+02	2.50E+02	2.50E+02	2.48E+02
Inverse value attack	Normal	1.18E+04	1.19E+04	1.19E+04	1.20E+04	1.23E+04
	Uniform	9.01E+06	9.04E+06	9.01E+06	8.93E+06	8.91E+06

Table 5: Normalized Root Mean Square Error (NRMSE) of correctness check in Section 4.3.1 under adversarial attacks.

E Decoding procedure for GVC

Decoding of GVC can be done by i) the inversion of the coefficient matrix in (6) in the manuscript or ii) interpolation and repeated divisions. In this section, we explain the decoding procedure by interpolation and repeated divisions.

The evaluation function $\mathbf{p}_{\mathbf{C}}(x)$ can be expressed as

$$\begin{aligned}\mathbf{p}_{\mathbf{C}}(x) = \mathbf{A}_1\mathbf{B}_1 + \cdots + \mathbf{A}_m\mathbf{B}_1x^{m-1} + \mathbf{A}_1\mathbf{B}_2f(x) + \cdots \\ + \mathbf{A}_{m-1}\mathbf{B}_nx^{m-2}(x)f^{n-1}(x) + \mathbf{A}_m\mathbf{B}_nx^{m-1}f^{n-1}(x).\end{aligned} \tag{6}$$

Since $\mathbf{p}_{\mathbf{C}}(x)$ is $mn-1$th order polynomial, it can be interpolated from the mn computation results. After interpolation, the master can get $\mathbf{A}_m\mathbf{B}_n$ as the quotient by dividing $\mathbf{p}_{\mathbf{C}}(x)$ with the highest degree term $x^{m-1}f^{n-1}(x)$. Similarly, $\mathbf{A}_{m-1}\mathbf{B}_n$ can be achieved as quotient by dividing the remainder of former division with the highest degree term $x^{m-2}f^{n-1}(x)$. All the required results $\mathbf{A}_i\mathbf{B}_j$ for $i \in [1:m]$ $j \in [1:n]$ can be achieved by using repeated divisions by the highest degree term in the same way.

F Additional experimental results

F.1 Stage-wise processing time of experimnts in Section 6.

We first provide the stage-wise processing time of experiment in Section 6 in Table 6-Table 7. In the table, bold indicates the best results among the codes.

From Table 6, we can see that HGT achieves the shortest encoding time for all cases. However, the verification time of HGT increases exponentially with the number of adversarial workers, even fails to verify in the given time limit 100 seconds when $l = 5$. On the other hand, GVCC requires much smaller verification time, thus speeding up overall processing time than AVCC for all values of l and HGT when $l \geq 3$. The results in Table 7 show that the GVCC outperforms other schemes under regardless of straggler probability, reducing overall processing time. From Table 8, we can see that the verification time of GVCC does not increase although the number of workers increases with

Codes	# of adversarial workers	Enc	SA	CR	Ver	Dec	Overall
AVCC	1	17.74	12.02	1.59	0.27	35.06	67.22
HGT		**11.15**	12.05	**1.56**	**0.044**	35.16	**60.45**
GVCC		12.07	**11.48**	1.70	0.081	**35.00**	60.65
AVCC	2	17.81	11.12	1.66	0.29	**35.07**	66.38
HGT		**11.22**	**10.79**	1.69	**0.047**	35.42	**59.58**
GVCC		12.03	11.49	**1.65**	0.083	35.20	60.88
AVCC	3	17.55	10.33	2.25	0.29	34.69	65.52
HGT		**10.97**	10.84	**1.57**	3.61	34.72	62.13
GVCC		11.90	**10.03**	1.65	**0.091**	**34.68**	**58.78**
AVCC	4	17.69	**10.67**	1.61	0.29	**34.65**	65.33
HGT		**10.84**	11.09	1.66	50.44	34.89	109.33
GVCC		11.86	11.08	**1.59**	**0.10**	34.75	**59.82**
AVCC	5	17.97	**11.44**	1.67	0.30	**34.92**	66.43
HGT		**11.01**	11.78	**1.66**	100.11 (Failure)	34.99	159.97
GVCC		13.14	11.83	1.78	**0.12**	40.87	**65.04**

Table 6: Processing time with different number of adversarial workers

Codes	Straggler probability	Enc	SA	CR	Dec	Overall
AVCC	0	17.15	10.18	**1.44**	34.36	63.55
HGT		**10.63**	**10.11**	4.03	34.34	59.52
GVCC		10.84	10.33	2.91	**34.33**	**58.84**
AVCC	0.02	17.19	10.41	**1.51**	34.33	63.86
HGT		**10.57**	**10.10**	4.66	34.33	60.07
GVCC		10.80	10.21	2.92	**34.29**	**58.64**
AVCC	0.05	17.12	**10.08**	**1.52**	37.26	63.41
HGT		**10.61**	10.47	4.12	**34.28**	59.88
GVCC		10.94	10.20	2.87	34.42	**58.85**
AVCC	0.1	17.03	**10.11**	**1.43**	34.27	63.27
HGT		**10.67**	10.44	4.41	34.41	60.33
GVCC		10.80	10.25	2.82	**34.39**	**58.68**
AVCC	0.2	17.15	10.38	**1.46**	34.39	63.80
HGT		**10.60**	**10.27**	4.89	**34.32**	60.49
GVCC		10.79	10.27	2.79	34.36	**58.64**

Table 7: Processing time with different straggler probability

the help of the group-wise verification approach. Whereas, the verification time of AVCC increases linearly with the number of workers. In addition, HGT requires more time for verification than AVCC and GVCC due to its high computational complexity for identifying adversarial workers. To be specific, GVCC speeds up verification time up to $\times 3.5$ than AVCC and $\times 57.29$ than HGT. Consequently, GVCC speeds up overall processing time up to $\times 1.20$ than AVCC and $\times 1.10$ than HGT. The results of experiment on the matrix shape is given in Table 9, also demonstrating the performance of GVCC.

F.2 Additional experiment.

To show the performance of GVCC in more various settings, we experiment in another parameter setting. To be specific, we use $m = 9, n = 7, E = 90$, and $l = 5$, with matrices $\mathbf{A} \in \mathbb{F}_q^{3600\times 6300}$, $\mathbf{B} \in \mathbb{F}_q^{6300\times 3500}$, and experiment for five straggler probabilities $0, 0.02, 0.5, 0.1$, and 0.2. Each experiment is repeated 20 times and we indicate average results. We used a modified algorithm in Section 4.5 and the overall processing time is depicted in Figure 4. We can see that GVCC still outperforms other schemes in overall processing time for all straggler probabilities. Furthermore, we indicate the stage-wise processing time in Table 10.

As we can see in the table, GVCC effectively handles straggler effects by using modified algorithm, while providing fast verification to adversarial attacks via group-wise verification.

Codes	# of workers	Matrix division parameters	Enc	SA	CR	Ver	Dec	Overall
AVCC	60	$(m,n)=(5,9)$	8.47	**4.85**	0.58	0.14	13.18	27.80
HGT			**3.03**	4.93	0.69	2.75	**13.05**	25.85
GVCC (proposed)			4.36	5.02	**0.57**	**0.048**	13.08	**23.31**
AVCC	84	$(m,n)=(6,11)$	11.43	**5.77**	0.31	0.17	17.48	35.63
HGT			**5.21**	5.81	**0.27**	1.92	**17.38**	31.21
GVCC (proposed)			5.54	5.83	**0.27**	**0.048**	17.46	**29.70**
AVCC	112	$(m,n)=(7,13)$	14.64	6.68	**0.28**	0.21	22.34	44.96
HGT			**6.49**	6.64	0.29	2.44	**22.05**	38.71
GVCC (proposed)			7.01	**6.60**	0.31	**0.046**	22.42	**37.17**

Table 8: Processing time with different number of workers

Matrix shape	Codes	Enc	SA	CR	Ver	Dec	Overall
Square ($\mathbf{A}\in\mathbb{F}_q^{3600\times3600}$, $\mathbf{B}\in\mathbb{F}_q^{3600\times3600}$)	AVCC	9.44	6.84	1.32	0.16	36.95	55.14
	HGT	**5.66**	**6.60**	**1.08**	3.28	37.05	54.08
	GVCC	6.06	6.78	1.33	**0.06**	**36.91**	**51.56**
Tall ($\mathbf{A}\in\mathbb{F}_q^{4500\times1500}$, $\mathbf{B}\in\mathbb{F}_q^{1500\times4500}$)	AVCC	4.83	3.63	1.17	0.098	66.18	76.34
	HGT	**2.95**	3.64	**1.08**	5.16	**66.09**	79.34
	GVCC	3.12	**3.62**	1.14	**0.056**	66.23	**74.61**
Fat ($\mathbf{A}\in\mathbb{F}_q^{2700\times7000}$, $\mathbf{B}\in\mathbb{F}_q^{7000\times2700}$)	AVCC	14.10	9.64	1.19	0.23	17.74	43.32
	HGT	**8.74**	9.21	1.19	2.53	**17.66**	39.75
	GVCC	9.40	**9.09**	**1.17**	**0.073**	17.75	**37.89**

Table 9: Processing time with different matrix shapes

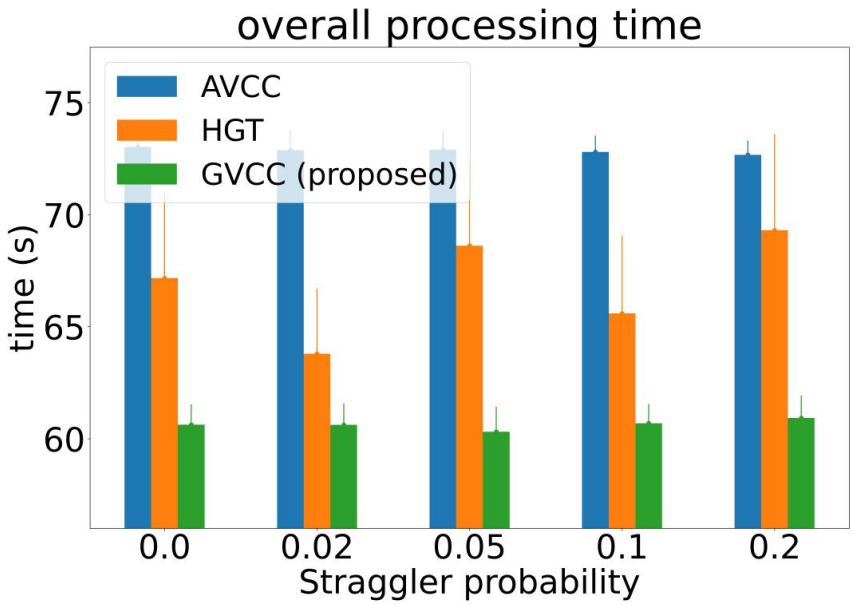


Figure 4: Overall processing time of AVCC, HGT, and GVCC (proposed) with different straggler probability

Codes	Straggler probability	Enc	SA	CR	Dec	Overall
AVCC	0	29.10	13.72	**1.44**	28.02	73.01
HGT		**15.72**	**13.40**	9.23	**27.97**	67.16
GVCC		16.10	13.59	2.23	**27.97**	**60.62**
AVCC	0.02	29.13	13.61	**1.39**	27.99	72.86
HGT		**15.77**	**13.43**	5.71	28.04	63.78
GVCC		16.08	13.49	2.34	**27.96**	**60.61**
AVCC	0.05	29.00	13.68	**1.50**	27.98	72.89
HGT		**15.81**	**13.55**	10.43	27.96	68.60
GVCC		16.16	13.59	1.89	**27.92**	**60.30**
AVCC	0.1	29.00	13.54	**1.47**	28.00	72.79
HGT		**15.79**	13.58	7.37	28.00	65.58
GVCC		16.10	**13.48**	2.44	**27.93**	**60.68**
AVCC	0.2	28.94	13.54	**1.46**	27.98	72.66
HGT		**15.88**	**13.49**	11.10	**27.79**	69.30
GVCC		16.07	13.75	2.39	27.97	**60.92**

Table 10: Processing time with different straggler probability