Physical-Layer Arithmetic for Federated Learning in Uplink MU-MIMO Enabled Wireless Networks

Tao Huang, Baoliu Ye, Zhihao Qu, Bin Tang, Lei Xie, Sanglu Lu

†National Key Laboratory for Novel Software Technology, Nanjing University, China
‡College of Computer and Information, Hohai University, China

Email: schrodinger.huang@gmail.com, yebl@nju.edu.cn, quzhihao@hhu.edu.cn, {tb, lxie, sanglu}@nju.edu.cn

Abstract—Federated learning is a very promising machine learning paradigm where a large number of clients cooperatively train a global model using their respective local data. In this paper, we consider the application of federated learning in wireless networks featuring uplink multiuser multiple-input and multiple-output (MU-MIMO), and aim at optimizing the communication efficiency during the aggregation of client-side updates by exploiting the inherent superposition of radio frequency (RF) signals. We propose a novel approach named Physical-Layer Arithmetic (PhyArith), where the clients encode their local updates into aligned digital sequences which are converted into RF signals for sending to the server simultaneously, and the server directly recovers the exact summation of these updates as required from the superimposed RF signal by employing a customized sum-product algorithm. PhyArith is compatible with commodity devices due to the use of full digital operation in both the client-side encoding and the server-side decoding processes, and can also be integrated with other updates compression based acceleration techniques. Simulation results show that PhyArith further improves the communication efficiency by 1.5 to 3 times for training LeNet-5, compared with solutions only applying updates compression.

I. INTRODUCTION

Federated learning and related decentralized learning are the kind of machine learning paradigm where the goal is to train a high quality centralized model, while training data remains distributed over a large number of clients [1]–[4]. In each round of learning algorithms for this paradigm, each client independently computes an update to the current model based on its local data, and sends this update to a central server, where the client-side updates are aggregated to compute a new global model. Communicating the model updates in each round has been observed to be a significant performance bottleneck [5] [6] for this paradigm. It is particularly serious for federated learning, since its typical clients are smart phones and IoT devices, which are with unreliable and relatively slow network connections.

To improve the communication efficiency, methods based on reducing the size of updates needed to transmit have been widely studied [5]–[9], by performing techniques like quantization, sparsification, etc. In addition to simply compressing client-side updates, there is also work focusing on improving the communication efficiency of federated learning by utilizing the characteristics of networks. Specifically, for wireless networks where uplink multiuser multiple-input and multiple-output (MU-MIMO) is enabled, i.e., clients can concurrently send their updates to the server in the same time-frequency resource, recent work like [10]–[13] study exploiting the superposition of radio frequency (RF) signals to effectively aggregate updates by averaging them, based on the technique called over-the-air computation (AirComp). These AirComp based work use uncoded analog transmission to send aligned client-side updates in the form of vector, each of which has the same number of elements placed in the same order. Clients in these work are assumed to be equipped with a special device featuring linear-analog-modulation and pre-channel-compensation (or simply assumed have no fading channel), such that the superimposed RF signal can be well utilized for fast update aggregation. However, these work are not dedicated to obtaining the exact average of updates. While the channel noise is not negligible and the pre-channel-compensation of each client is not perfect, their obtained average may contain significant aggregation error, which may seriously affect the convergence of the global model to train.

In this paper, we study utilizing the superposition of RF signals to efficiently aggregate client-side updates in federated learning by using commodity devices dedicated to modern wireless networks featuring uplink MU-MIMO, e.g., 802.11ax [14], where neither linear-analog-modulation nor pre-channel-compensation is necessarily available. We target at reliably obtaining the exact average of updates, such that the convergence of the global model to train is guaranteed for realistic channel conditions. Inspired by the fact that the amount of information of all updates is greater than that of their summation, our idea is that we can directly recover the exact summation based on the received superimposed RF signal, and then calculate their average. It should have lower outage probability compared with the conventional multiuser detection (MUD) based solution to deal with such mutual interference in uplink MU-MIMO, i.e., separating these colliding data streams based
on the superimposed RF signal, recovering each update, and averaging them, as shown in Fig. 1. We denote our solution as Physical-Layer Arithmetic (PhyArith), where the clients encode their local updates into aligned digital sequences which are converted into RF signals for sending to the server simultaneously, and the server directly recovers the exact summation of these updates from the superimposed RF signal through a customized decoder.

Fig. 1. Recovery of averaged update based on MUD and PhyArith

For this purpose, three main challenges should be well addressed. The first challenge is, how to encode each update at the client with commodity devices based on full digital operation, such that PhyArith can conveniently and reliably recover their exact summation, and effectively verify its correctness. To address this challenge, we encode updates into aligned digital sequences by taking two’s-complement representation \([15]\) as source codes, and systematic low-density parity check (LDPC) codes \([16]\) as channel codes, such that the addition and subtraction operations during aggregation can be unified, and error correction capability can be provided. We also append the cyclic-redundancy-checksum (CRC) \([17]\) with each update, such that its linear property can be exploited to verify the correctness of the directly recovered summation. The second challenge is, how to directly recover the exact summation of updates at the server, which is equipped with multiple antennas, based on the superimposed RF signal of these encoded updates emanating from a large number of different clients. To address this challenge, from the superimposed RF signal, we propose a sum-product algorithm \([18]\) derived polynomial time complexity decoder to directly obtain the marginal probability mass function (PMF) of the summation of updates. By maximizing the obtained PMF, we can recover the summation, verify its correctness, and update the global model. And the third challenge is, how to integrate PhyArith with those updates compression methods like sparsification, where the client-side updates are not necessarily aligned due to compression. To address this challenge, we adopt compressed-sensing here by using a shared underdetermined sensing matrix to compress the sparse update of each client, such that they are aligned and can be efficiently aggregated by PhyArith.

The main contributions are summarized as follows: (1) We show that even if not applying pre-channel-compensation, the superimposed RF signal can be exploited to efficiently aggregate updates, without incurring any aggregation error. (2) Based on full digital operation, we design the client-side updates encoding process of PhyArith, such that it can be applied on commodity 802.11 devices, and their summation can be reliably recovered and verified. We also design its server-side decoding process where a customized sum-product algorithm is provided to directly recover the summation of a large number of updates. (3) PhyArith can be integrated with other updates compression methods. Simulation results show that PhyArith further improves the communication efficiency by 1.5 to 3 times for training LeNet-5, compared with the solutions which apply quantization/sparsification to compress updates, and aggregate them via MUD.

II. RELATED WORK

A. Quantization and Sparsification

Updates quantization and sparsification are the two main approaches to overcome the communication bottleneck in decentralized learning. Updates quantization was first proposed in \([5]\), where elements not less than zero are encoded as bit 1, and those less than zero are encoded as bit 0. Following such 1-bit quantization, subsequent work like \([8]\) \([9]\) introduce multilevel updates quantization, so as to balance the communication efficiency and updates accuracy. Updates sparsification was first proposed in \([6]\), where only elements with large absolute value are encoded using 1-bit quantization, and sent out along with their associate index. Following this approach, various sparsification methods, like sending fixed proportion of large elements \([7]\), are proposed.

B. AirComp

Recent AirComp based work focus on leveraging the superposition of RF signal to aggregate client-side updates. Specifically, \([12]\) \([13]\) study performing power control and client selection to align the signal power received at the server and avoid incurring serious aggregation error. \([10]\) studies aggregating updates during wireless networks with power and bandwidth limitations, based on coded digital transmission in different time-frequency resources, and uncoded analog transmission in the same time-frequency resource, i.e., performing AirComp. It is also worth noting that although recent AirComp related work are all based on uncoded analog transmission, its original idea appeared in \([19]\), which relies on lattice codes to achieve reliable over-the-air computation. However, it also assumes no fading channel, which can not be directly applied on commodity devices at the client-side. And at the server-side, it lacks practical efficient decoding approach.

C. MUD

Different from AirComp, the conventional approach to deal with the superimposed RF signal is multiuser detection
(MUD), where updates are separately recovered and aggregated. The state-of-the-art polynomial time complexity MUD approach includes Gaussian message based interference cancellation (GM-IC) [20] [21], and linear minimum mean square error based interference cancellation (LMMSE-IC) [22], etc. Like our server-side decoder in PhyArith, these approaches are also derived from the sum-product algorithm, which operates on the factor-graph [18] in a message-passing manner.

III. MOTIVATION

In this section, through a theoretical analysis, we show that even if pre-channel-compensation of AirComp is not applied, the superimposed RF signal can still be exploited to efficiently aggregate updates, without incurring any aggregation error. Consider the following simple channel with two-input and one-output, denoted by

\[ y = h_0 s_0 + h_1 s_1 + w, \]

where \( y \in \mathbb{R} \) is the channel output symbol, \( h_0, h_1 \in \mathbb{R} \) are the channel gain, \( s_0, s_1 \) are independent channel input symbols uniformly chosen in \( \{-1,+1\} \), and noise \( w \sim \mathcal{N}(0, \sigma^2) \).

We study the average symbol time needed to accurately obtain function \( f(s_0, s_1) = s_0 + s_1 \) for MUD and PhyArith in theory. To calculate the symbol time, we study the entropy (i.e., the amount of information, in bits) [23] of \( s_0, s_1 \) and \( s_0 + s_1 \) first. Easy to find that entropy \( H(s_0) = H(s_1) = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1 \), and \( H(s_0 + s_1) = -0.5 \log_2 0.25 - 0.5 \log_2 0.5 = 1.5 \). And then we study the capacity of above channel, characterised by mutual information, i.e., the amount of information can be retrieved from the channel, in bits per symbol time [23]. For conventional successive-interference-cancellation (SIC) based MUD, we have the mutual information between \( y \) and \( s_0 \), i.e., \( I(y; s_0) = H(y) - H(y|s_0) \), and that between \( y \) and \( s_1 \) on condition of \( s_0 \), i.e., \( I(y|s_0); s_1) = H(y|s_0) - H(y|s_0, s_1) \). Therefore, the symbol time to obtain \( s_0 + s_1 \) for SIC based MUD is given by

\[
\max\{H(s_0)\} = \max\{H(y; s_0), H(s_1)\} = H(y)/I(y; s_0 | s_1). \\
(1)
\]

For PhyArith, we calculate the mutual information between \( y \) and \( s_0 + s_1 \) instead, which is given by \( I(y; s_0 + s_1) = H(y) - H(y|s_0 + s_1) \). Therefore, the symbol time to obtain \( s_0 + s_1 \) is given by

\[
H(s_0 + s_1)/I(y; s_0 + s_1). \\
(2)
\]

With (1) and (2), we plot Fig. 2 for \( h_0 = h_1, |h_0| = +1 \) and \( h_0 = -h_1, |h_0| = +1 \), where calculation processes of entropy \( H(y), H(y|s_0, s_1), H(y|s_0) \) and \( H(y|s_0 + s_1) \) are provided in Appendix-A. We can find that while \( h_0 = h_1, |h_0| = +1 \), which can be viewed as pre-channel-compensation is perfectly applied and function distortion (i.e., \( f(s_0, s_1) - y \) is small, symbol time of PhyArith is always less than that of MUD. And while \( h_0 = -h_1, |h_0| = +1 \), which can be viewed as no pre-channel-compensation applied and function distortion is extremely serious, symbol time of PhyArith is also less than that of MUD when signal-noise-ratio (SNR) is above some threshold. As a conclusion, we can use PhyArith to improve the communication efficiency of federated learning without incurring any effect on its convergence, even if pre-channel-compensation is not applied.

IV. SYSTEM MODEL

A. Model of Federated Learning

The goal of federated learning is to learn a model with parameters embodied in a vector \( M \) from data stored across a large number of clients. For simplicity, we consider synchronized algorithms for federated learning where quantization is taken to compress client-side updates and accelerate the communication. For cases where no compression methods are applied, i.e., directly sending floating-point updates, enabling PhyArith is similar to the quantization case, by treating the signed integer exponent of each floating-point number as the coefficient for de-quantization, and treating the significand as the quantized update element. And for cases where other updates compression methods like sparsification are applied, we show how to enable PhyArith for them in Section VIII.

A typical round \( t \) of such synchronized algorithms consists of the following steps: (1) A set of clients, denoted by \( V = \{v_0, \ldots, v_{|V| - 1}\} \), is selected. Each client \( v \in V \) downloads the current model \( M_t \) from the server. (2) Each \( v \in V \) computes an updated model \( M'_t \) based on their local data. (3) The quantized update vector \( A_t^v = Q_v(M'_t - M_t) \), \( v \in V \) (the quantization function), as well as the de-quantization function \( Q^{-1}_v(\cdot) \), are sent from each client to the server. (4) The server aggregates these updates by averaging and construct an improved global model \( M_{t+1} = M_t + \eta_t A_t \), where \( \eta_t \) is the learning rate, and \( A_t = \sum_{v \in V} A_t^v/[|V|] \). Here we assume the quantization function \( Q_v(\cdot) \) is like those proposed in [8] [9]. As a result, without loss of generality, we have \( A_t^v \) is a vector of signed integers, and \( Q^{-1}_v(\cdot) \) can be formed as

\[
Q^{-1}_v(A_t^v) = \beta_v A_t^v + \gamma_v 1, \\
\]

where \( \beta_v, \gamma_v \) are floating-point de-quantization coefficients, \( \beta_v > 0 \), and \( 1 \) is a vector with all entries being 1.

B. Model of Uplink MU-MIMO Channel

Here we present the model of wireless networks like 802.11ax, where uplink MU-MIMO is enabled. Let \( U = \{u_0, \ldots, u_{|U| - 1}\} \) denote the set of \( |U| \) antennas of the server. Clients in \( V \) are equipped with single antenna, which simultaneously transmit their local updates to the server, during
the same time-frequency resource, e.g., the same subcarriers within the same orthogonal frequency-division multiplexing (OFDM) symbols in 802.11ax. For the $j$-th time-frequency resource unit (e.g., 1 subcarrier within 1 OFDM symbol), the uplink MU-MIMO channel can be presented as follows

$$
\begin{bmatrix}
y_{u,j}^0 \\
y_{u,j}^1 \\
\vdots \\
y_{u,j}^{|V|-1}
\end{bmatrix} = H_j
\begin{bmatrix}
s_{v,j}^0 \\
s_{v,j}^1 \\
\vdots \\
s_{v,j}^{|V|-1}
\end{bmatrix} +
\begin{bmatrix}
w_{u,j}^0 \\
w_{u,j}^1 \\
\vdots \\
w_{u,j}^{|V|-1}
\end{bmatrix},
$$

where $w_{u,j}^i$ is i.i.d. random variable following circularly normal distribution $CN(0,2\sigma^2)$, and denotes the additive noise of antenna $v$ for the $j$-th time-frequency resource unit. $y_{u,j}^i \in \mathbb{C}$ denotes the observation of $u$, $s_{v,j}^i \in \mathcal{S}$ (where $\mathcal{S} \subset \mathbb{C}$) denotes the baseband symbol of $v$ obtained via modulation, and the channel gain matrix $H_j = [h_{v,j}^0, \cdots, h_{v,j}^{|V|-1}]$, where vector $h_v^i = [h_{v,u,j}^0, \cdots, h_{v,u,j}^{|V|-1}]^T$.

V. OVERVIEW OF PHYARITH

Here we provide an overview of PhyArith based on the model of federated learning and uplink MU-MIMO channel presented above. PhyArith consists of two phases, i.e., the client-side encoding phase and the server-side decoding phase. At the client-side, each quantized update vector $A_v^t$ is sliced into chunks, which are encoded into aligned digital sequence through the source codes and channel codes encoder by appending checksums and parity bits, and simultaneously transmitted through the uplink MU-MIMO channel. The source codes taken here are based on two’s-complement representation [15], and the channel codes are based on systematic low-density parity check (LDPC) codes [16], such that the addition and subtraction operations during aggregation can be unified, and error correction capability can be provided. The appended checksum is the cyclic-redundancy-checksum (CRC) [17], such that its linear property can be exploited to verify the correctness of the directly recovered summation. At the server-side, from the superimposed RF signal $[y_{u,j}^0, \cdots, y_{u,j}^{|V|-1}]^T$, the summations of a large number of updates chunks are directly recovered through a customized decoder derived from the sum-product algorithm. After verifying their correctness based on the linear property of CRC, these summations are averaged and assembled into $\sum_{v \in V} (\beta_v A_v^t + \gamma_v 1)/|V|$, which is then taken to update the global model to train.

VI. ENCODING PROCESS OF PHYARITH

In this section, we demonstrate the encoding process of PhyArith. As shown in Fig. 3, the quantized update vector $A_v^t$ of client $v$ is sliced into fixed size chunks first, i.e., $A_v^t = [\cdots, a_{v,j}, \cdots]$. For each chunk $a_v = [\cdots, a_{v,j}, a_{v,j+1}, \cdots]$, $a_{v,j} \in \mathbb{Z}$, it is encoded into a bit-sequence of the source codes. The obtained bit-sequence of source codes is then appended with its checksum (we use cyclic-redundancy-checksum, i.e., CRC, in PhyArith), and forms the $k$ bits sourceword of the channel codes, denoted by vector $b_v = [b_{v,0}, \cdots, b_{v,k-1}]$. Each such sourceword is further encoded into the $n$ bits codeword of the channel codes by appending a sequence of parity bits, denoted by vector $x_v = [x_{v,0}, \cdots, x_{v,n-1}]$. These codewords of channel codes are then mapped into $m$ baseband symbols via modulation, denoted by $s_v = [s_{v,0}, \cdots, s_{v,m-1}]$, and sent out.

A. Source Codes

Regarding the source codes, in the quantized update vector chunk $a_v = [\cdots, a_{v,j}, a_{v,j+1}, \cdots]$, each its element $a_{v,j}$ is a signed integer, and is encoded into $c$ bits $\{b_{v,j0}, \cdots, b_{v,j,c-1}\}$ within $b_v = [b_{v,0}, \cdots, b_{v,k-1}]$ using two’s-complement representation [15], according to the following rule

$$b_{v,jl} = \begin{cases} a_{v,j}/2^{c-l-1} \mod 2, & a_{v,j} \geq 0, \\ \lfloor (2^c + a_{v,j})/2^{c-l-1} \rfloor \mod 2, & \text{o.w.} \end{cases}$$

where $l = 0, \cdots, c-1$, $b_{v,j0}$ is the most significant bit and $b_{v,j,c-1}$ is the least significant bit. Here we should ensure $2^{c-1} \geq \max\{|a_{v,j}|, v \in V, j \in \mathbb{N}\}$. Compared to other systems for representing signed numbers (e.g., one’s complement), the advantage of two’s-complement is that the fundamental arithmetic operations of addition, subtraction, and multiplication are identical to those for unsigned binary numbers, such that it would be much convenient for our PhyArith to directly decode the summation of update vectors.

B. Checksum

The checksum appended to each bit-sequence of source codes is cyclic-redundancy-checksum (CRC) [17]. Together with the bit-sequence encoded from $a_v$, they form the sourceword $b_v$. We use CRC in PhyArith because it is a linear function with the following property

$$\text{CRC}(b_v \oplus b_v') = \text{CRC}(b_v) \oplus \text{CRC}(b_v') = 0 \oplus 0 = 0,$$

where $\oplus$ denotes bitwise-XOR, and $\text{CRC}(\cdot)$ returns the checksum of input in the form of sequence of bits. This property can help us verify the correctness of the decoded summation of update vectors. We present this verification process in detail in Section VII-C.

C. Channel Codes

In order to provide error correction capability in PhyArith, the $k$ bits sourceword $b_v$, consisting of the bit-sequence encoded from quantized update vector chunk $a_v$, and its CRC,
is encoded into \( n \) bits codeword \( x_v = [x_{v,0}, \ldots, x_{v,n-1}] \) of channel codes. The channel codes adopted here are systematic low-density parity check (LDPC) codes [16]. For each \( k \) bits sourceword \( b_v \), it is appended with \( n-k \) parity bits provided by LDPC codes encoder, and form the \( n \) bits codeword \( x_v \). To enable PhyArith, we require the quantized update vector chunks transmitted simultaneously share the same LDPC codes codebook, i.e., the mapping from \( b_v \) to \( x_v \) should be the same for different client \( v \). Besides, for modulation, the mapping from \( x_v \) to \( s_v = [s_{v,0}, \ldots, s_{v,m-1}] \) should be the same as well.

D. Compatibility with Commodity 802.11 Devices

In 802.11ax, similar with PhyArith, systematic LDPC codes are taken for error correction. For fixed coding rate, the codebook mapping from scrambled sourceword to LDPC codes codeword is fixed. Besides, for fixed modulation, the mapping from codeword to symbols transmitted in the same time-frequency resource is also fixed. As a conclusion, the encoding process of 802.11ax is fully compatible with PhyArith. That is, we can directly use commodity single antenna 802.11ax devices to send the sourceword \( b_v \) of PhyArith via a raw socket, where the channel codes related process is provided by 802.11ax devices.

VII. DECODING PROCESS OF PHYARITH

In this section, we demonstrate the decoding process of PhyArith. Limited by the decoding complexity, client set \( V \) in PhyArith is randomly partitioned into subsets \( V = \{V_0, \cdots, V_{|V|-1}\} \) first, where \( \forall V_i \in V, |V_i| \leq \Delta \) and \( \Delta \) is the subset size limit. Then, as shown in Fig. 4, based on the superimposed RF signal, PhyArith directly decodes the sourceword-sum of each subset \( V_i \), which is denoted by

\[
z_i = [z_{i,0}, \cdots, z_{i,k-1}] = \sum_{v \in V_i} \hat{\beta}_v b_v,
\]

where \( \hat{\beta}_v \) is an odd integer given by

\[
\hat{\beta}_v = \begin{cases} \beta_v \cdot 2^\zeta + 1 & \text{with probability } 1/2, \\ \beta_v \cdot 2^\zeta - 1 & \text{o.w.} \end{cases}
\]

and \( \zeta \) is a randomly chosen integer ensuring \( \forall v \in V, \beta_v \cdot 2^\zeta \geq 1 \). For these obtained sourceword-sum \( z_i \) verified to be correct, the summation of update vector chunks of \( V_i \), i.e., \( \sum_{v \in V_i} (\hat{\beta}_v a_v + \beta_v 1) \), is recovered. And for these \( z_i \) verified to be incorrect, retransmission is requested. Finally, all these recovered summations can be added up and averaged, then assembled into the averaged update vector \( A_t \), given by

\[
A_t = \left[ \cdots, \sum_{V_i \in V} \sum_{v \in V_i} (\beta_v a_v + \beta_v 1)/|V_i|, \cdots \right].
\]

Note that here we let each \( \hat{\beta}_v \) be a large odd integer to facilitate the verification of \( z_i \) in Section VII-C, and make sure the recovered \( \sum_{v \in V_i} (\beta_v a_v + \beta_v 1) \) is accurate in Section VII-D.

A. Building of the Factor-Graph

We first show how to directly decode each sourceword-sum \( z_i \) based on the superimposed RF signal. Let \( y_j = [y_{u_0,j}, \cdots, y_{u_{|V|-1},j}]^T \) be the \( j \)-th observation vector. The decoding problem here can be formulated as the following optimization problem

\[
z_0^*, \cdots, z_{|V|-1}^* = \arg \max_{z_0, \cdots, z_{|V|-1}} P(z_0, \cdots, z_{|V|-1}|y_0, \cdots, y_{m-1}).
\]

Since jointly optimize \( z_0, \cdots, z_{|V|-1} \) is too complicated to solve. We can relax (5) as follows

\[
z_{i,j}^* = \arg \max_{z_{i,j}} P(z_{i,j}|y_0, \cdots, y_{m-1}),
\]

where the marginal probability can be written as the following sum-product form via factorization

\[
P(z_{i,j}|y_0, \cdots, y_{m-1}) = \sum_{\sim z_{i,j}} P(y_0, \cdots, y_{m-1}, z_0, \cdots, z_{|V|-1})/P(y_0, \cdots, y_{m-1})
\]

\[
A_i \sum_{\sim z_{i,j}} [P(y_0, \cdots, y_{m-1}|s_{v_0}, \cdots, s_{v_{|V|-1}})]
\]

\[
|V|-1 \prod_{i'=0}^{\mid V \mid -1} [\prod_{v \in V_{i'}} I(s_{v_i}, x_v, b_v)] \delta(|| \sum_{v \in V_{i'}} (\beta_v a_v + \beta_v 1) || 1)].
\]

Here \( \sim z_{i,j} \) denotes all random variables except \( z_{i,j} \). \( I(s_{v_i}, x_v, b_v) \) is the indicator function to ensure \( x_v \) is a valid LDPC codes codeword encoded from \( b_v \), and \( s_{v_i} \) is a valid symbol sequence modulated from \( x_v \). And \( \delta(\cdot) \) denotes the Dirac delta function.

A general method to efficiently obtain the marginal probability of each \( z_{i,j} \) is the factor-graph based sum-product algorithm [18], which operates in a message-passing manner. Specifically, while \( |V| = 2 \), the factorization (6) can be visualized by a factor-graph, a kind of bipartite graph like Fig. 5, which consists of two kind of nodes. One is called variable node (denoted by circles), representing random variables, and the other is called check node (denoted by squares), representing the local functions of variables. By exchanging messages in the form of probability mass function
(PMF) between adjacent nodes, the marginal probability of random variables can be obtained. Generally, suppose client subset $V_i = \{v_{i,0}, \ldots, v_{i,|V|-1}\}$. For $V_i$, let the $j$-th baseband symbol vector $s_{i,j} = [s_{v_{i,0},j}, \ldots, s_{v_{i,|V|-1},j}]^T$, and codeword bit vector $x_{i,j} = [x_{v_{i,0},j}, \ldots, x_{v_{i,|V|-1},j}]^T$. Based on the factorization (6), we build the factor-graph $G$ for decoding the sourceword-sum in PhyArith. In $G$, as shown in Fig. 5, $Y_j$ denotes the $j$-th observation check node. And for client subset $V_i$, we let $C_{i,j}$ denote its $j$-th LDPC codes check node, $X_{i,j}$ its LDPC codes variable node (corresponding to variable $x_{i,j}$), $M_{i,j}$ its modulation check node, $S_{i,j}$ its modulation variable node (corresponding to variable $s_{i,j}$), $Z_{i,j}$ its sourceword-sum variable node (corresponding to variable $z_{i,j}$), and $\Sigma_{i,j}$ its sourceword-sum check node.

**Fig. 5. Example factor-graph for the sum-product algorithm derived decoder in PhyArith where $|V| = 2$**

**B. Decoding of the Sourceword-Sum**

The decoding process of the sourceword-sum on factor-graph $G$ is illustrated in Algorithm 1. Given factor-graph $G$, the channel gain vector $h_v^y$, and observation $y_j$, where $v \in V$, $j \in \{0, \ldots, m-1\}$. Algorithm 1 returns the decoding result $z_{i,j}^*$, where $i \in \{0, \ldots, |V|-1\}$, $j \in \{0, \ldots, k-1\}$. Algorithm 1 has MaxIterationNum iterations. In each iteration, on factor-graph $G$, it first parallelly propagates PMFs from all variable nodes to their adjacent check nodes, and then propagates PMFs from check nodes to their adjacent variable nodes. And after these iterations, it takes the following formula to obtain the final decoding result $z_{i,j}^*$

$$z_{i,j}^* = \arg \max_{z_{i,j}} P_{\Sigma_{i,j}} \rightarrow Z_{i,j} (z_{i,j}),$$

where $P_{\Sigma_{i,j}} \rightarrow Z_{i,j}(z_{i,j})$ denotes the PMF of random variable $z_{i,j}$, propagated from check node $\Sigma_{i,j}$ to variable node $Z_{i,j}$, given by (21) in Appendix-C.

The most challenging part in Algorithm 1 is how to efficiently and accurately obtain the PMF of $s_{i,j}$ propagated from $Y_j$ to $S_{i,j}$, given by

$$P_{Y_j \rightarrow S_{i,j}}(s_{i,j}) = A \cdot \sum_{s_{i,j}} [P(y_j|s_{0,j}, \ldots, s_{|V|-1,j}) \prod_{i \neq j} P_{S_{i,j} \rightarrow Y_j}(s_{i,j})].$$

Different from these MUD approaches like LMMSE-IC [22] where all random variables $s_{v,j}, v \in V$ are approximated by an independent circularly symmetric normal distribution, which lost the information of the relationship between these random variables. In order to efficiently utilize the superposition of RF signals to aggregate updates, here we treat each random variable pair $(s_{v,i}, s_{v,j}), v_i, v_j \in V_i$ is mutual dependent, and use the following function derived from the probability density function of multivariate normal distribution to approximate above PMF

$$P_{Y_j \rightarrow S_{i,j}}(s_{i,j}) = A \cdot \prod \left(- \text{Real}(\mu_{i,j} - H_j^t s_{i,j}) + \text{Real}(\mu_{i,j} - H_j^t s_{i,j})/2 \right),$$

where $H_j = [h_{v_0,j}, \ldots, h_{v_{|V|-1},j}]$ is the channel gain matrix of client subset $V_i$. Covariance matrix $\Gamma_{i,j}$ is obtained via (22) in Appendix-D. Vector $\mu_{i,j}$ is given by

$$\mu_{i,j} = y_j - \sum_{v \in V/V_i} h_{v,j}^t E(s_{v,j}),$$

where $E(s_{v,j})$ is the mean of random variable $s_{v,j}, v \in V_i$, obtained based on PMF $P_{S_{v,j} \rightarrow Y_j}(s_{v,j}).$ And operation $\text{Real}(\cdot): \mathbb{C}^{|V|} \rightarrow \mathbb{R}^{|V|}$ is defined as $\text{Real}(\alpha_0, \alpha_1, \ldots, \alpha_{|V|-1})^T \triangleq \langle \text{Re}(\alpha_0), \text{Im}(\alpha_0), \ldots, \text{Re}(\alpha_{|V|-1}), \text{Im}(\alpha_{|V|-1}) \rangle^T$.

**Algorithm 1 Decoding process of the sourceword-sum**

**Input:** $G$, $h_v^y$, $y_j$ where $v \in V$, $j \in \{0, \ldots, m-1\}$

**Output:** $z_{i,j}^*$ where $i \in \{0, \ldots, |V|-1\}$, $j \in \{0, \ldots, k-1\}$

1: for $i = 0$ do
2: for all variable nodes $S_{i,j}, X_{i,j}, Z_{i,j} \in G$ do
3: propagate their PMFs given by (11) (12) (13) (14) (15) (16) to their adjacent check nodes
4: end for
5: for all check nodes $Y_{i,j}, M_{i,j}, C_{i,j}, \Sigma_{i,j} \in G$ do
6: propagate their PMFs given by (8) (17) (18) (19) (20) (21) to their adjacent variable nodes
7: end for
8: end for
9: for $i = 0$ do
10: take (7) to obtain $z_{i,j}^*$
11: end for

**C. Verification of the Sourceword-Sum**

Here we show how to verify the correctness of the obtained sourceword-sum $z_{i,j}^*$. Note that for each correct $z_i = \beta_{v_0} b_{v_0} + \cdots + \beta_{v_{|V|-1}} b_{v_{|V|-1}}$, we have

$$z_i \mod 2 = (\beta_{v_0} \mod 2) b_{v_0} \oplus \cdots \oplus (\beta_{v_{|V|-1}} \mod 2) b_{v_{|V|-1}} = b_{v_0} \oplus \cdots \oplus b_{v_{|V|-1}}.$$
That is, we can verify the correctness of $\mathbf{z}^*_i$ via the following rule
\begin{equation}
\begin{cases}
\mathbf{z}^*_i \text{ is correct, } & \text{CRC}([\mathbf{z}^*_i \mod 2] = 0, \\
\mathbf{z}^*_i \text{ is incorrect, } & \text{o.w.}
\end{cases}
\end{equation}
(9)
Note that by verifying the checksum with (9), we can guarantee the correctness of $\mathbf{z}^*_i$ with high probability. However, some $\mathbf{z}^*_i$ can still be erroneous even if it passes the checksum verification.

D. Recovery of the Summation of Update Vector Chunks

Based on the sourceword-sum $\mathbf{z}_i$ that is verified to be correct, here we show how to recover $\sum_{v \in V_i} \beta_v a_v + \gamma_v 1$. Each element $\beta_v a_{v,j}$ in $a_v$ is encoded into $\{b_{v,j,0}, \ldots, b_{v,j,-1}\}$ in $b_v$ according to (3), we have
\[
\sum_{v \in V_i} \beta_v b_{v,j,0}, \ldots, \sum_{v \in V_i} \beta_v b_{v,j,-1} = \{z_{i,j,0}, \ldots, z_{i,j,-1}\}.
\]
That is, each element $\sum_{v \in V_i} (\beta_v a_{v,j} + \gamma_v)$ can be recovered via
\begin{equation}
\sum_{v \in V_i} (\beta_v a_{v,j} + \gamma_v) = \left\{ \begin{array}{ll}
\frac{\alpha}{2^c} + \sum_{v \in V_i} \gamma_v, & \text{if } \alpha = 2^c + c - 1 = 0, \\
\frac{\alpha - 2^c - c}{2^c} + \sum_{v \in V_i} \gamma_v, & \text{o.w.,}
\end{array} \right.
\end{equation}
(10)
where
\[
\alpha = (\sum_{i,j} 2^{c-l-1} + \sum_{i=0}^{c-1} z_{i,j} 2^{c+l}) \mod 2^{c+1},
\]
and $c = \lceil \log_2(|\sum_{v \in V_i} \beta_v|) \rceil$. The process presented in (10) can be viewed as transferring each $c$-bits signed integer in two's-complement representation (e.g., $\{b_{v,j,0}, \ldots, b_{v,j,-1}\}$) into a $(c+\epsilon)$-bits signed integer in two's-complement representation (e.g., $\{b_{v,j,0}, b_{v,j,1}, \ldots, b_{v,j,-1}\}$), by inserting $\epsilon$ extra most significant bit $b_{v,j,0}$ in its beginning, such that the addition of $|V|$ such signed integers won’t result in the overflowing problem.

VIII. PHYArith with Sparsification

In this section, we show how to enable PhyArith in federated learning where sparsification is taken to compress client-side updates and accelerate the communication.

A. Client-Side Process

In conventional sparsification based updates compression methods, each client simply sends sparse vector $\mathbf{A}_t^v = \text{SP}(\mathbf{M}_t - \mathbf{M})$ to the server, where $\text{SP}(\cdot)$ denotes the sparsification process. Different from these methods, in order to enable PhyArith, at the client-side, the update vector
\[
\mathbf{A}_t^v = Q_v(\mathbf{C} \cdot \text{SP}(\mathbf{M}_t - \mathbf{M})),
\]
is sent from each client to the server, where $\mathbf{C}$ is the random underdetermined sensing matrix shared across all clients and the server, used for compressing the sparse vector $\text{SP}(\mathbf{M}_t - \mathbf{M})$ based on compressed-sensing. For such $\mathbf{A}_t^v$, it can then be sliced into chunks, i.e., $\mathbf{A}_t^v = [\mathbf{a}_1, \ldots, \mathbf{a}_n]$, and encoded according to the routine provided in Section VI.

B. Server-Side Process

And at the server-side, similar to the routine provided in Section VII, all these recovered summations of update vector chunks, i.e., $\sum_{v \in V} (\beta_v a_v + \gamma_v 1)$, are assembled into the averaged compressed update vector $\mathbf{C} \cdot \mathbf{A}_t$, given by
\[
\mathbf{C} \cdot \mathbf{A}_t = \frac{\sum_{v \in V} \sum_{v \in V} (\beta_v a_v + \gamma_v 1)/|V|, \ldots}
\]
\[
= \sum_{v \in V} Q_v(\mathbf{A}_t^v)/|V| = \mathbf{C} \cdot \text{SP}(\mathbf{M}_t - \mathbf{M})/|V|.
\]
By recovering sparse vector $\mathbf{A}_t$ based on algorithms like convex programming [24] for compressed-sensing, the server can construct an improved global model $\mathbf{M}_{t+1} = \mathbf{M}_t + \eta_t \mathbf{A}_t$.

IX. PERFORMANCE EVALUATION

A. Experiment Setup

We conduct experiments using the federated learning algorithm FedSGD presented in [3] to train LeNet-5 [25] for the MNIST digit recognition task, where the training data is shuffled and equally distributed over all clients. During each round of FedSGD, all clients are selected, each of which computes the model update (i.e., the gradient) based on all its local data (i.e., batch size equals all its local data) and forward the update to the server. In the experiment, consistent with 802.11ax, we let $|U| = 8$ (i.e., the server is equipped with 8 antennas) and study different total client size $|V|$ in the following two subsections. Specifically, we study $|V| = 4, 8, 12, 16$, denoted by configuration $4 \times 8, 8 \times 8, 12 \times 8$, and $16 \times 8$, respectively. The channel gain matrix between clients and server is generated according to the multiple antenna channel model provided in [26]. The modulation scheme is BPSK. And we use the standard LDPC codes specified in 802.11 with coding rate 1/2, where $k = 324$ and $n = 648$, for error correction. The cyclic-redundancy-checksum adopted in the experiment is CRC24 where the polynomial is $0x800063$, the initial value of 24 bits register is 0x0, and the final XOR value is 0x0. Each quantized update vector chunk is encoded into a sequence of 300 bits. Such 300 bits appended with 24 bits CRC24 form every 324 bits LDPC codes sourceword.

B. Outage Ratio of Aggregating Client-Side Updates

We first show the outage ratio of aggregating client-side updates during training LeNet-5 using FedSGD. Specifically, we show the outage ratio of obtaining the exact summation of all update vector chunks transmitted simultaneously, based on PhyArith and the state-of-the-art polynomial time complexity MUD approach GM-IC and LMMSE-IC [20]–[22]. From the experiment results shown in Fig. 6, we can see 1) PhyArith always outperforms both MUD approaches; 2) PhyArith is especially better when $|V| \geq |U|$. This is because while $|V| \geq |U|$, the decoding process of MUD approaches like LMMSE-IC and GM-IC does not necessarily converge [21]. We also take the appended checksums to verify the correctness of all these erroneous summations directly recovered.
by PhyArith in Fig. 6 (55,479 erroneous summations in all configurations). Experiment results show that none of them passes the verification.

![Fig. 6. Outage ratio of obtaining the exact summation of all update vector chunks transmitted simultaneously in the first round of FedSGD based on PhyArith and MUD, where FedSGD is with configuration (a) 4 x 8; (b) 8 x 8; (c) 12 x 8; (d) 16 x 8, learning rate η = 0.01, updates are compressed using 10 bits quantization.](image)

**C. Test Accuracy of LeNet-5 Trained by FedSGD**

And then, we show the test accuracy of LeNet-5 trained by FedSGD with configuration 16 x 8, where the SNR of the uplink MU-MIMO channel is set as 23.33 dB, and the client-side updates are aggregated based on PhyArith and the MUD approach LMMSE-IC. Recall the results in Fig. 6 that the MUD approach can not handle the 16 x 8 configuration where updates from all 16 clients are transmitted to the server simultaneously. To be fair to the MUD approach, we let all its 16 updates be separately simultaneously transmitted in several slots, such that they can be efficiently aggregated. From the experiment results shown in Fig. 7 and Fig. 8, we can see that to achieve the same test accuracy, our proposal PhyArith can further improve the communication efficiency by 1.5 to 3 times, compared with the solutions which apply quantization/sparsification to compress updates, and aggregate them via MUD.

**X. Conclusion**

In this paper, we propose PhyArith that improves the communication efficiency of federated learning in wireless networks featuring uplink MU-MIMO, by directly aggregating client-side updates from the superimposed RF signal, without affecting the convergence of the training procedure. In the future, we would like to make other server-side aggregation methods be available in PhyArith, and further improve its decoding performance.

**APPENDIX**

**A. Calculation of Entropy**

While |h₀| = |h₁| = +1.0, entropy H(y) is given by

\[
H(y) = H\left\{ \frac{1}{4\sqrt{2}\pi\sigma} \right\} \left\{ 2\exp\left(-\frac{y^2}{2\sigma^2}\right) + \exp\left(-\frac{(y+2)^2}{2\sigma^2}\right) + \exp\left(-\frac{(y-2)^2}{2\sigma^2}\right) \right\},
\]

entropy H(y|s₀, s₁) is given by

\[
H(y|s₀, s₁) = H\left\{ \frac{1}{2\sqrt{2}\pi\sigma} \right\} \left\{ \exp\left(-\frac{y^2}{2\sigma^2}\right) \right\},
\]

and entropy H(y|s₀) is given by

\[
H(y|s₀) = H\left\{ \frac{1}{2\sqrt{2}\pi\sigma} \right\} \left\{ \exp\left(-\frac{(y+1)^2}{2\sigma^2}\right) + \exp\left(-\frac{(y-1)^2}{2\sigma^2}\right) \right\}.
\]
While \( h_0 = h_1, |h_0| = +1.0, \) entropy \( H(y|s_0 + s_1) = H(y|s_0, s_1). \) And while \( h_0 = -h_1, |h_0| = +1.0, \) entropy \( H(y|s_0 + s_1) \) is given by

\[
H(y|s_0 + s_1) = \frac{1}{2} \left( \frac{1}{\sqrt{2\pi}\sigma} \exp \left( -\frac{y^2}{2\sigma^2} \right) \right) + \frac{1}{2} \left( \frac{1}{\sqrt{2\pi}\sigma} \exp \left( -\frac{(y - 2\sigma)^2}{2\sigma^2} \right) \right).
\]

**B. PMF from Variable Node to Check Node in Algorithm 1**

The PMF propagated from \( S_{i,j} \) to \( Y_j \) is given by

\[
P_{S_{i,j} \rightarrow Y_j}(s_{i,j}) = \begin{cases} \frac{1}{|S|^{|V|}}, & \text{the 1st iteration,} \\ \frac{1}{|S|^{|V|}} \prod_{(X_{i,j},C_{i,j}) \in \mathcal{G}} P_{C_{i,j} \rightarrow X_{i,j}}(x_{i,j}), & \text{o.w.} \end{cases}
\]

The PMF propagated from \( S_{i,j} \) to \( M_{i,j} \) is given by

\[
P_{S_{i,j} \rightarrow M_{i,j}}(s_{i,j}) = \begin{cases} \frac{1}{|S|^{|V|}}, & \text{the 1st iteration,} \\ \frac{1}{|S|^{|V|}} \prod_{(X_{i,j},M_{i,j}) \in \mathcal{G}} P_{M_{i,j} \rightarrow X_{i,j}}(x_{i,j}), & \text{o.w.} \end{cases}
\]

The PMF propagated from \( X_{i,j} \) to \( M_{i,j} \) is given by

\[
P_{X_{i,j} \rightarrow M_{i,j}}(x_{i,j}) = \begin{cases} 1/2|V|, & \text{the 1st iteration,} \\ A \cdot \prod_{(X_{i,j},C_{i,j}) \in \mathcal{G}} P_{C_{i,j} \rightarrow X_{i,j}}(x_{i,j}), & \text{o.w.} \end{cases}
\]

The PMF propagated from \( X_{i,j} \) to \( C_{i,j} \) is given by

\[
P_{X_{i,j} \rightarrow C_{i,j}}(x_{i,j}) = \begin{cases} 1/2|V|, & \text{the 1st iteration,} \\ A \cdot \prod_{(X_{i,j},C_{i,j}) \in \mathcal{G}} P_{C_{i,j} \rightarrow X_{i,j}}(x_{i,j}), & \text{o.w.} \end{cases}
\]

The PMF propagated from \( X_{i,j} \) to \( \Sigma_{i,j} \) is given by

\[
P_{X_{i,j} \rightarrow \Sigma_{i,j}}(x_{i,j}) = A \cdot \prod_{(X_{i,j},\Sigma_{i,j}) \in \mathcal{G}} P_{\Sigma_{i,j} \rightarrow X_{i,j}}(x_{i,j}).
\]

The PMF propagated from \( Z_{i,j} \) to \( \Sigma_{i,j} \) is given by

\[
P_{Z_{i,j} \rightarrow \Sigma_{i,j}}(z_{i,j}) = 2^{-|V|}.\]

**C. PMF from Check Node to Variable Node in Algorithm 1**

The PMF propagated from \( M_{i,j} \) to \( S_{i,j} \) is given by

\[
P_{M_{i,j} \rightarrow S_{i,j}}(s_{i,j}) = A \cdot \prod_{j \neq t, (X_{i,j},M_{i,j}) \in \mathcal{G}} P_{C_{i,t} \rightarrow X_{i,t}}(x_{i,t}).
\]

where \( I(j', j, x_{i,j'}, s_{i,j}) \) is the indicator function to ensure the \( j' \)-th codeword bits vector \( x_{i,j'} \) is consistent with the \( j \)-th symbol vector \( s_{i,j} \).

The PMF propagated from \( M_{i,t} \) to \( X_{i,j} \) is given by

\[
P_{M_{i,t} \rightarrow X_{i,j}}(x_{i,j}) = \begin{cases} \prod_{j' \neq j, (X_{i,j'},M_{i,j}) \in \mathcal{G}} P_{X_{i,j'} \rightarrow M_{i,j}}(x_{i,j'}), & \text{for } i \neq t, \\ \prod_{j' \neq j, (X_{i,j'},M_{i,j}) \in \mathcal{G}} P_{X_{i,j'} \rightarrow M_{i,j}}(x_{i,j'}), & \text{for } i = t. \end{cases}
\]

For the subset \( V_i \), let \( \mathbf{B}_i = [\hat{\beta}_{v_i,0}, \cdots, \hat{\beta}_{v_{|V_i|-1},0}]^T \). The PMF propagated from \( \Sigma_{i,j} \) to \( X_{i,j} \) is given by

\[
P_{\Sigma_{i,j} \rightarrow X_{i,j}}(x_{i,j}) = A \cdot \prod_{j' \neq j, (X_{i,j'},C_{i,j}) \in \mathcal{G}} P_{X_{i,j'} \rightarrow C_{i,j}}(x_{i,j'}). \]

The PMF propagated from \( \Sigma_{i,j} \) to \( Z_{i,j} \) is given by

\[
P_{\Sigma_{i,j} \rightarrow Z_{i,j}}(z_{i,j}) = A \cdot \prod_{j' \neq j, (X_{i,j'},C_{i,j}) \in \mathcal{G}} P_{X_{i,j'} \rightarrow Z_{i,j}}(z_{i,j}).
\]

**D. Calculation of Covariance Matrix \( \Gamma_{i,j} \)**

Random variables in vector \( \text{Real}(y_j) \) follow multivariate normal distribution. On condition of random variable \( s_{i,j} \), we can calculate the covariance matrix \( \Gamma_{i,j} \) of random variables in vector \( \text{Real}(y_j) \) as follows

\[
\Gamma_{i,j} = \sum_{v_i \in V/\mathcal{V}_i} (\text{Var}(\text{Re}(s_{i,j}))) \text{Imag}(\mathbf{h}_{\mathbf{v}_i}) \text{Imag}(\mathbf{h}_{\mathbf{v}_i}^T) + \text{Var}(\text{Im}(s_{i,j})) \text{Re}(\mathbf{h}_{\mathbf{v}_i}) \text{Re}(\mathbf{h}_{\mathbf{v}_i}^T) + \text{Cov}(\text{Re}(s_{i,j}), \text{Im}(s_{i,j})).
\]

\[
(\text{Real}(\mathbf{h}_{\mathbf{v}_i}) \text{Imag}(\mathbf{h}_{\mathbf{v}_i})^T + \text{Imag}(\mathbf{h}_{\mathbf{v}_i}) \text{Re}(\mathbf{h}_{\mathbf{v}_i})^T) + \sum_{V_j \in V/\mathcal{V}_i} \sum_{v_j \in V_j} (\text{Cov}(\text{Re}(s_{i,j}), \text{Re}(s_{v_j})).
\]

\[
(\text{Imag}(\mathbf{h}_{\mathbf{v}_i}) \text{Imag}(\mathbf{h}_{\mathbf{v}_i})^T + \text{Re}(\mathbf{h}_{\mathbf{v}_i}) \text{Re}(\mathbf{h}_{\mathbf{v}_i})^T) + \text{Cov}(\text{Im}(s_{i,j}), \text{Im}(s_{v_j})).
\]

\[
(\text{Real}(\mathbf{h}_{\mathbf{v}_i}) \text{Imag}(\mathbf{h}_{\mathbf{v}_i})^T + \text{Imag}(\mathbf{h}_{\mathbf{v}_i}) \text{Re}(\mathbf{h}_{\mathbf{v}_i})^T) + \text{Cov}(\text{Re}(s_{i,j}), \text{Re}(s_{v_j})).
\]

\[
(\text{Imag}(\mathbf{h}_{\mathbf{v}_i}) \text{Imag}(\mathbf{h}_{\mathbf{v}_i})^T + \text{Re}(\mathbf{h}_{\mathbf{v}_i}) \text{Re}(\mathbf{h}_{\mathbf{v}_i})^T) + \text{Cov}(\text{Im}(s_{i,j}), \text{Re}(s_{v_j})).
\]

where \( \text{Var}(\cdot) \) denotes the variance of random variable, \( \text{Cov}(\cdot) \) denotes the covariance of random variables, both of which are obtained based on PMF \( P_{S_{i,j} \rightarrow Y_j}(s_{i,j}), \) \( \text{diag}(\cdot) \) denotes diagonal matrix. And operation \( \text{Imag}(\cdot) : \mathbb{C}^{|U|} \rightarrow \mathbb{R}^{2|U|} \) is defined as \( \text{Imag}(\omega_0, \cdots, \omega_{|U|-1})^T \neq [-\text{Im}(\omega_0), \text{Re}(\omega_0), \cdots, -\text{Im}(\omega_{|U|-1}), \text{Re}(\omega_{|U|-1})^T. \)
REFERENCES


