Accelerating Secure Collaborative Machine Learning with Protocol-Aware RDMA

Zhenghang Ren, Mingxuan Fan, Zilong Wang, Junxue Zhang, and Chaoliang Zeng, *iSING Lab@The Hong Kong University of Science and Technology;* Zhicong Huang and Cheng Hong, *Ant Group;* Kai Chen, *iSING Lab@The Hong Kong University of Science and Technology and University of Science and Technology of China*

https://www.usenix.org/conference/usenixsecurity24/presentation/ren

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

Accelerating Secure Collaborative Machine Learning with Protocol-Aware RDMA

Zhenghang Ren[1], Mingxuan Fan[1], Zilong Wang[1], Junxue Zhang[1], Chaoliang Zeng[1], Zhicong Huang[2], Cheng Hong[2], and Kai Chen[1,3]

[1]iSING Lab@The Hong Kong University of Science and Technology [2]Ant Group
[3]University of Science and Technology of China

Abstract

Secure Collaborative Machine Learning (SCML) suffers from high communication cost caused by secure computation protocols. While modern datacenters offer high-bandwidth and low-latency networks with Remote Direct Memory Access (RDMA) capability, existing SCML implementation remains to use TCP sockets, leading to inefficiency. We present CORA[1] to implement SCML over RDMA. By using a *protocol-aware* design, CORA identifies the protocol used by the SCML program and sends messages directly to the remote party's protocol buffer, improving the efficiency of message exchange. CORA exploits the chance that the SCML task is determined before execution and the pattern is largely input-irrelevant, so that CORA can plan message destinations on remote hosts at compile time. CORA can be readily deployed with existing SCML frameworks such as Piranha with its socket-like interface. We evaluate CORA in SCML training tasks, and our results show that CORA can reduce communication cost by up to 11× and achieve 1.2× −4.2× end-to-end speedup over TCP in SCML training.

1 Introduction

Secure Collaborative Machine Learning (SCML) provides solutions to protect privacy in machine learning tasks [21, 30, 44, 55]. SCML enables multiple individuals or organizations to perform model training or prediction collaboratively with their private data and get higher-quality results without compromising privacy. However, SCML faces high communication cost [41, 44, 58, 70] because it adopts communication-intensive protocols such as Secret Sharing (SS) and Oblivious Transfer (OT) as basic building blocks.

The widely deployed datacenters have brought opportunities to accelerate SCML. On one hand, it has been a common practice to store datasets and deploy machine learning tasks in a datacenter. The tenants within a datacenter will get better models if they collaboratively train on their datasets. On the other hand, datacenters provides high-performance computing and network facilities essential to train models. However, when performing SCML tasks, the datacenter network is not fully utilized because the implementation of SCML protocols is mainly based on TCP sockets, which is inefficient due to cost in host network stack [16]. Remote Direct Memory Access (RDMA), as a more efficient networking technology, is generally available in datacenters [26], yet remains to be integrated with SCML frameworks.

The communication cost consists of the time spent in the network protocol stack and the time for data propagation on the wire. While existing works accelerate SCML communication by configuring the transport layer protocol [12, 50] or reducing communication complexities [13], RDMA can speed up communication by offloading network stack to RDMA Network Interface Controller (RNIC), bringing better latency and bandwidth than kernel TCP.

However, accelerating SCML with RDMA still faces the following challenges:

- Current SCML frameworks usually use hybrid protocols [15, 31, 41–43, 48, 69]. But RDMA is unaware of the protocols, so messages are mixed in the staging buffer on receiving side which requires extra costs of demultiplexing the messages to multiple protocols' buffers.
- For the same protocol conducted by multiple parties, the communication patterns may differ across parties because the parties are different entities and have different runtime configurations, such as segmentation sizes, which require the extra cost of partitioning or concatenating messages.
- New SCML frameworks are still emerging with different protocols and communication patterns. Manually demultiplexing messages of multiple protocols is not scalable in new SCML frameworks.

We present CORA to address the above challenges with the following observations: 1) SCML tasks are data-parallel with each element in the input matrix processed by the same

[1]CORA: secure Collaborative machine learning Over RdmA

protocol. 2) The communication pattern of protocols will not change with the input, i.e., for each protocol used in SCML, it yields the same communication pattern [35] for different elements in the input matrix. 3) SCML tasks have high-level descriptions such as Domain Specific Languages (DSL) and APIs [33, 69], which provide the input's length and adopted protocols (§2.3). These opportunities enable us to demultiplex messages belonging to multiple protocols at compile time.

With the above observations, CORA tackles the challenges with the following designs:

- For the challenge of demultiplexing messages of hybrid protocols, CORA sets up a dedicated buffer for each protocol used in SCML. With the observations 2 and 3, CORA can identify the remote receiving buffer with the current task and adopted protocol. For example, when performing a convolution with SS protocol, the sender writes messages to an agreed buffer that is used to store SS messages specifically.

- For the challenge of heterogeneous communication patterns, CORA assigns offset addresses for the messages in the protocol buffers. With the observations 1 and 2, we know that the communication pattern is determined, so the address of the message will not change with input. The receiver can get the message directly from the corresponding offset without extra cost.

- For the challenge of applying CORA in new SCML frameworks, CORA relies on the observation 3 that current SCML frameworks have high-level descriptions of SCML tasks, including model structure and input length. CORA only requires the backend protocol to implement the layers in the SCML model. These features are commonly available in recent SCML frameworks.

We implement CORA as an independent module. Developers can use CORA in existing SCML frameworks without much effort because the interfaces are similar to the socket. It consists of three parts: 1) A Parser that extracts the SCML layers and adopted protocols as part of the protocol contexts to identify the current protocol. 2) A Planner that assigns starting addresses for the protocols to store messages. 3) Basic communication primitives that encode the current protocol in RDMA operations so that the receiver side knows the protocol context and addresses of arrived messages.

We thoroughly evaluate CORA in machine learning tasks, and the key results are as follows:

- In SCML building blocks, such as machine learning layers, CORA achieves $1.8\times - 7.2\times$ speedup over TCP.

- In SCML training tasks including SecureML [44], LeNet [36], VGG16 [51], and AlexNet [34], CORA achieves $1.2\times - 4.2\times$ speedup over TCP.

- In SCML inference tasks, CORA achieves $1.2\times - 2.0\times$ speedup over TCP.

- CORA reduces the communication cost in the model training tasks by up to $11\times$. The communication cost is no longer the bottleneck after using CORA.

To summarize, this paper makes two main contributions: First, we are among the first to use RDMA to accelerate SCML tasks, and we improve the efficiency of using RDMA for SCML tasks with CORA, a protocol-aware design of RDMA. Second, we integrate CORA into existing SCML frameworks and show the performance benefit of CORA.

2 Background

2.1 Secure Collaborative Machine Learning

SCML is mainly built on top of Secure Multi-party Computing (MPC) protocols, which enables a group of entities such as individuals or companies to jointly evaluate a function without revealing sensitive input to other parties. SCML combines various of different protocols, such as Oblivious Transfer (OT), Secret Sharing (SS), Garbled Circuits (GC), and Homomorphic Encryption (HE) to perform machine learning tasks. Recent efforts have achieved success in optimizing SCML in terms of computational efficiency [58], communication [45], versatility [33], and ease of use [69].

However, communication is still one of the major overheads in SCML due to the usage of MPC protocols [41, 44, 58, 70] and large amounts of training data. Studies have shown that communication is the bottleneck in both training and inference tasks of SCML [41, 44, 45].

Recent works such as ABY3 [43], Fantastic Four [19], Sharemind [11], and PrivPy [37] have enabled deploying SCML in a datacenter. For example, ABY3 enables three non-colluding servers to collaboratively perform machine learning tasks in a datacenter. An arbitrary number of parties can generate secret sharings of their data and distribute them to the servers. The privacy of data is guaranteed as long as at most one of the three servers is compromised.

Datacenters have high-performance interconnects with hundreds of Gbps bandwidth and a few microseconds latency. However, existing SCML frameworks fail to fully utilize the network facilities because of the inefficient TCP network [16]. During sending and receiving, it consumes lots of CPU cycles to process TCP packets, which bounds the maximum bandwidth [27, 39, 63] and harms computing performance. These problems motivate us to deploy SCML over RDMA network which bypasses CPU during data transmission.

2.2 Remote Direct Memory Access

RDMA enables direct access to a remote computer's memory without involving either side's operating system. It achieves

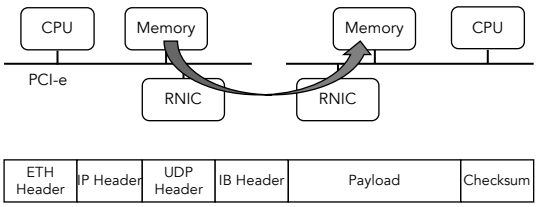


Figure 1: Overview of data transmission in RDMA network and RDMA packet format (RoCE v2).

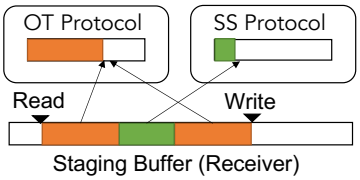


Figure 2: Illustration of the cost brought by the staging buffer because of multiple protocols and fractioned messages.

high bandwidth and low latency with zero-copy data transfer [64] and hardware-based network protocols in RNICs, which bypasses operating systems and CPUs during data transmission.

An overview of RDMA network is shown in Figure 1. A memory region on the remote host is registered and authorized to be directly accessed by RNICs without extra copy. When sending data, the RNIC possesses the full network stack to transmit the packets without involving CPUs. Moreover, RDMA packets have negligible overhead in headers compared to existing network protocols. For example, RoCE v2 [4], as one of the standard RDMA packet formats, is built on top of UDP protocol with an extra IB header (12 Bytes).

With offloaded network stack and zero-copy networking, RDMA outperforms traditional networks on both latency and bandwidth and has been widely deployed in datacenters. Moreover, RDMA brings chances to accelerate SCML for the following reasons:

- SCML frameworks intensively use sockets to send/receive data which triggers a context switch between user space and kernel space and interrupts the CPU when handling the received data. RDMA offloads the API to user space, and the program gets received data with a notification generated by RDMA hardware, which improves the utilization of CPU cycles.
- TCP requires copying the data to kernel buffers for sending, while RDMA supports zero-copy data transmission.
- TCP constantly consumes CPU when sending data, while RDMA implements network protocols in hardware and does not consume CPU during sending/receiving.

Despite performance benefits, RDMA presents a distinct interface compared to Linux socket and requires applications to change the way of sending/receiving data. RDMA defines *verbs* as basic primitives to access RNICs, including *read*, *write*, *send*, and *recv*. The hosts first register a memory region and allow direct access from remote hosts. When sending data, the sender needs to specify both the address of source data on local host and the destination address on remote host, and then sends messages by posting a *write* verb to the RNIC. The sender can also rely on the receiver to decide the destination address. In that case, the receiver first specifies receiving address by posting a *recv* verb to the RNIC, and then the sender posts a *send* verb with the address of source data.

2.3 Challenges and Opportunities

It is still a challenging task to use RDMA efficiently in SCML. We elaborate on the challenges and opportunities as follows:

Demultiplexing Cost A straightforward way of using RDMA is to write all messages sequentially into the remote memory, which is common in existing applications [22, 24, 25, 32, 40]. However, using a unified remote buffer is inefficient in SCML due to heterogeneous configurations of parties and hybrid protocols. SCML combines multiple secure protocols with their messages mixed in the same buffer, which requires demultiplexing to different protocols' buffers, as shown in Figure 2. With hybrid protocols, RDMA is unaware of the correspondence between protocols and the received messages. The demultiplexing cost cannot be ignored, considering the large amount of data traffic between parties in SCML.

Heterogeneous Communication Patterns The parties in SCML are different entities and may have different runtime configurations (e.g. batch size) resulting in undefined orders of messages arrival. As shown in Figure 2, the orange messages represent two messages of OT. The receiver needs to concatenate them into its buffer. Moreover, the address of messages is unknown to the receiver, so they needs extra interactions to locate the fractioned messages.

Generality in SCML frameworks Although we can set up multiple dedicated buffers for different protocols used in SCML, it is not scalable to new SCML protocols. Developers will have to manually configure the remote memory to send the messages of corresponding protocols.

Opportunities State-of-the-art SCML frameworks develop Domain Specific Languages (DSL) and programming APIs that have high-level description of machine learning tasks to ease the development of SCML programs. They define basic building blocks such as matrix multiplication, convolution, ReLU activation, max pooling, etc. Every building block is determined by several parameters. For example, the convolution layer is determined by the length (or shape) of the input matrix, kernel size, strides, etc. These parameters are determined before execution and will not change throughout the

task.

With these highly structured task descriptions and the secure protocols used to execute the task, we gain the knowledge of the messages that will be sent during task execution at compile time. For example, if the matrix multiplication is implemented using additive secret sharing, and the parties decompose into vector products, we know that the parties will exchange messages that have the same lengths with the decomposed vectors, according to the multiplication in secret sharing protocol [10].

The prior knowledge enables CORA to assign addresses for messages at compile time, so that applications can write messages directly to the corresponding protocol's buffer, saving the cost of demultiplexing.

3 Design

3.1 Overview

A naive incorporation of RDMA into SCML with a unified buffer leads to suboptimal performance due to its extra overhead of copying and reordering messages. The root cause exists in its inability to recognize upper-layer protocols when sending/receiving data. CORA distinguishes itself with a protocol-aware design of RDMA library that has a static demultiplexing scheme generated at compile time, which improves the efficiency of message exchanging. At its core, CORA identifies the message's protocol and writes it to a *preset* memory address on the receiver side using RDMA. The receiver passes the received message's address to upper-layer protocols without an extra copy.

There are mainly two technical challenges encountered when communicating via RDMA. First, it lacks mechanisms to notify receivers when messages arrive because RDMA bypasses receivers' OS. Although receivers can be notified by posting RDMA *recv* verbs, it requires the same communication patterns between the sender and receiver, which is not always satisfied in SCML. Second, heterogeneous communication patterns lead to unpredictable messages' addresses on receivers' memories, leaving receivers unable to locate received messages. It is impractical to preset remote addresses manually, given various protocols' implementations.

CORA solves the technical challenges by leveraging both characteristics of SCML tasks and RDMA operations. First, SCML tasks are naturally data-parallel and input-irrelevant. Each element in an input array is processed with the same function, such as matrix multiplication and activation, and yields the same communication pattern. Second, RDMA allows verbs to carry a customizable immediate number, which serves as a notification to receivers.

CORA assigns ids encoded in immediate numbers to the messages, which notify the receiver when messages arrive. The receiver locates messages from remote parties by interpreting message ids associated with their addresses in the

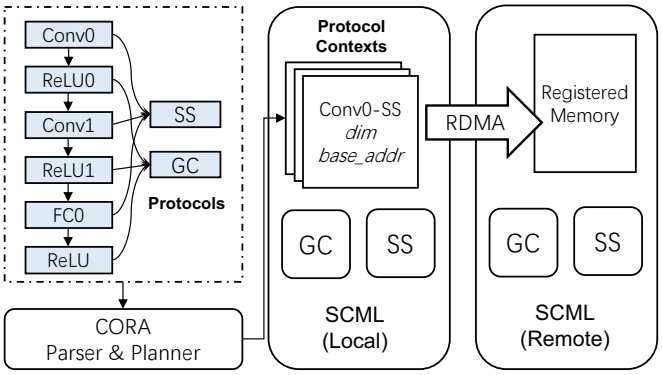


Figure 3: Example of training a model collaboratively with CORA. RDMA is used to send messages to protocols' receiving buffers.

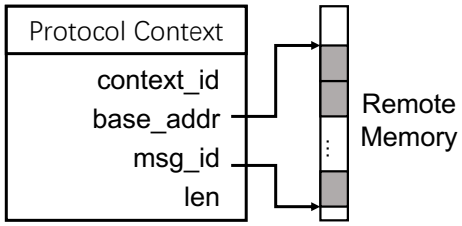


Figure 4: How the protocol context points to a message memory.

Specifically, the input array is applied to the same function, producing message arrays with identical message sizes. We group the messages produced by the input array and access them using a base address and an index. SCML tasks typically require multiple rounds of interactions between parties, leading to multiple message arrays. CORA manages base addresses and indexes with a data structure named *protocol context*. When sending a message, CORA loads the protocol context and writes to the remote address using RDMA. The receiver converts the message id as an index to locate the received message.

We design a *Parser* and a *Planner* to manage protocol contexts. We illustrate their functions with a SCML training task shown in Figure 3. The user defines SCML tasks and provides the backend protocols for secure computation for each layer: Convolution (Conv) layers and Fully Connected (FC) layers are performed with SS protocol, and ReLU activations are performed with GC.

The Parser first processes the model structure, protocol, and input data's length. This information helps to get the memory space needed for each message array. When starting the task, the parties initialize RDMA connections. Each connection registers a memory region on the party's server so that remote parties can write messages directly.

The parties then collaboratively train the model using secure protocols. For each layer, CORA writes messages to the addresses indicated by the protocol context. As shown in

Figure 4, the protocol context assigns a remote address for the message by providing a base address (base_addr) and a message index (msg_id). The remote buffer is filled with a list of messages with the same sizes indicated by grey blocks, as they are produced by the same protocol on different elements in the input array.

The receiver has the same protocol context when receiving messages. When a message arrives, the receiver checks its protocol context for the base address and accesses the message using the index. Upper-layer protocols get the message by reference. Details are shown in §4.

When calculating the next layer, the Planner initializes a new protocol context and allocates a buffer as the base address. The Planner recycles the buffers of existing protocol contexts whose messages have been processed and are no longer needed in the future. Details are shown in §3.3.

3.2 Parser

The Parser is the frontend of CORA, which extracts model structures, backend secure protocols, and input lengths from users' descriptions. It indicates the memory space needed to store messages for each protocol context. The output of Parser reduces the complexity of planning remote addresses when facing all kinds of implementations of secure protocols.

The Parser works without running SCML tasks. With input irrelevance, we know the protocol will yield the same amount of messages on each element in the input array. The Parser extracts the length of input array, which is interpreted as the number of messages. Then CORA gets the memory needed for the layer. For example, in an array with length l where each element corresponds to m messages in the protocol, we have $l \times m$ messages to send to the remote buffer.

Note that some layers, such as convolution, are not performed in a per-element way. For example, convolutions decompose input as subarrays. We classify the layers into arithmetic and boolean layers and apply different parsing methods.

Besides machine learning layers, SCML involves protocols that are not included in model structures. The Parser supports including these protocols in model structures by adding extra layers to models.

- Protocol conversion: SCML may need to convert protocols for encrypting secret data, as no current protocols can support all operations efficiently. Protocol conversions may also involve communication, and CORA builds protocol conversion contexts to accelerate protocol conversion with RDMA (§3.2.3).

- Offline protocols: SCML tasks require generating correlated random data such as Beaver Triples [10] and edaBits [23], which also brings lots of communication (§3.2.4).

3.2.1 Arithmetic Layers

The Parser decomposes arithmetic layers into basic operations and extracts the lengths of messages. Arithmetic layers typically perform matrix multiplications such as convolution layers and fully connected layers. CORA parses the layers' parameters to get the lengths and the adopted secure protocols at compile time as part of the context.

Matrix multiplications are decomposed into inner products of row vectors and column vectors. Assuming the input matrix with lengths $M \times N$ and $N \times P$, the total number of basic operations is $M \times N \times P$. For the messages generated in matrix multiplication, the message ids are $0 \ldots MNP-1$, and the message gets its index with the input elements' positions.

Matrix multiplication has optimized protocol with lower communication overhead [44]. The optimized secure matrix multiplication regards input matrices as a whole. When performing secure matrix multiplications, the optimized implementation yields messages with the same lengths as the input matrices, namely $M \times N$ and $N \times P$. In this case, the message ids are $0 \ldots MN+NP-1$.

Note that the sizes of input matrices are usually specified in the SCML program, which provide us with the sizes of messages at compile time. In rare cases where the SCML program does not specify the lengths of vectors, the parties would have to agree on the input length before execution so that Parser could still get the length and assign indexes for the messages.

3.2.2 Boolean Layers

Boolean layers such as ReLU activation, Softmax activation, and Max Pooling are mostly implemented with protocols such as boolean secret sharing and garbled circuits. Although it is feasible to implement all layers with boolean protocols, current SCML frameworks avoid doing so because boolean protocols are generally less efficient in arithmetic tasks [20].

The Parser assigns message lists with the indexes of data elements. Boolean layers are mainly implemented by feeding the input array's elements iteratively into the boolean protocol. For single-input activations such as ReLU and Softmax, each element yields the same communication pattern; for multi-input protocols such as Max Pooling implemented by comparison protocol, the Parser assigns message lists with the element with minimal index in the input matrix.

Note that boolean layers may have multiple rounds of interaction. For example, Max Pooling layers may be implemented by boolean secret sharing with multiple AND gates, which requires multiple rounds of communication. In this case, the Parser aggregates the total number of gates as the message indexes list.

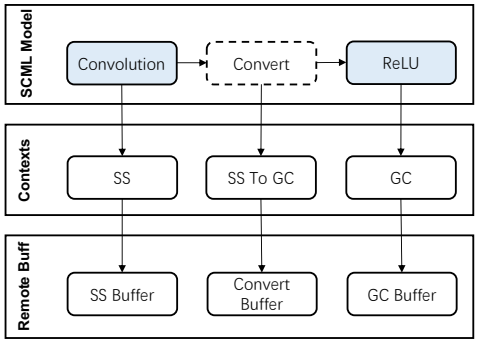


Figure 5: CORA detects protocol conversion and sets up protocol contexts to handle communication during converting.

3.2.3 Protocol Conversion

Existing SCML frameworks usually use hybrid protocols for different operations [43, 45]. For example, arithmetic secret sharing is used for matrix multiplication, while GC is used for Softmax. Hybrid protocols require converting the form of secret sharing to other protocols when needed.

Protocol conversions change the representations of secret data using different conversion functions, which may involve communication [20]. For example, when converting secret data from arithmetic secret sharing to boolean secret sharing, the parties jointly compute a bit extraction circuit; when converting from boolean sharing to arithmetic sharing, the parties evaluate a bit-adder circuit. We regard protocol conversions as extra layers in SCML tasks, as shown in Figure 5.

3.2.4 Offline Protocols

SCML needs protocols that generate Beaver triples and other correlated random data consumed in the online task. Since the generation of triples is not related to the input of online tasks, existing works often schedule these protocols offline, which is done before the execution of online task [43, 44, 58].

Considering that offline protocols may bring large amounts of communication in SCML, especially when the model is large, CORA builds extra protocol context to accelerate communication in offline protocols. Compared to SCML layers, offline protocols do not have associations with the input data. CORA creates a protocol context for offline protocols specifically.

There are mainly two methods for generating Beaver triples: OT-based protocols and HE-based protocols [20]. CORA sends the messages during triple generation to a specific buffer on the remote party. The messages are passed to upper-layer protocols without extra copies.

3.3 Planner

The Planner manages the protocol contexts by assigning their base addresses and message ids before sending/receiving. It realizes memory space needed with the input lengths extracted by the Parser and the message sizes.

When sending a message, the Planner checks if the protocol context exists. If so, it assigns the message id with the data index in the array. Otherwise, it first allocates memory and initializes a new protocol context. The Planner frees the remote memory when the remote party has processed the messages.

Specifically, the remote address is decided by the following equation:

$$\text{dest\_addr} = \text{base\_addr} + \text{msg\_size} \times \text{index} \quad (1)$$

In the equation, dest_addr represents the destination address on remote memory; base_addr represents the base address in the protocol context; msg_size represents the size of message; index is the position of element in the array.

On the sender side, the Planner first gets the message's size, the index in the input array, and the layer's input length. It allocates the memory space for the current layer and initializes the base address. The Planner allocates memory with a simple searching algorithm: it finds the first memory region that satisfies the need. Note that the allocation algorithms are the same for all parties. So, the protocol contexts on different parties have the same base address, guaranteeing that senders write to the same location from which receivers read.

Algorithm 1 How Planner allocates memory space and prepare protocol context for a message

Input: *msg_size*, *input_len*, *p_context*, *index*
G: dependency graph of the model, l: current layer
Output: Modified *p_context*

1: $M \leftarrow msg\_size \times input\_len$ ▷ The total memory needed
2: **if** *p_context* is not initialized **then**
3: Search Remote Buffer to place M
4: **if** Success with *addr* **then**
5: $p\_context.base\_addr \leftarrow addr$
6: **else**
7: Report Failure
8: **end if**
9: **end if**
10: $p\_context.msg\_id \leftarrow index$
11: $p\_context.len \leftarrow msg\_size$
12: **if** l finished **then** ▷ free memory
13: $L \leftarrow l.prev$ ▷ The layers that l_i depends on.
14: $G.L.next -= 1$ ▷ All layers have one less child layer.
15: **if** any layer l_i in $G.L$ has no child layer **then**
16: Free the memory used by l_i
17: **end if**
18: **end if**

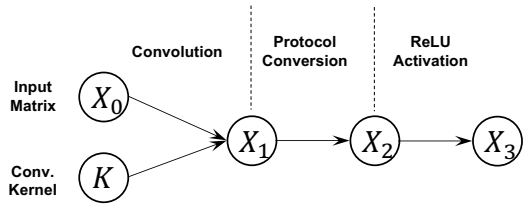


Figure 6: Dependency graph generated from a convolution followed by ReLU activation.

On the receiver side, the Planner modifies the protocol context in a symmetric way to the sender's so that the protocol contexts at both sizes point to the same message on the receiver's buffer. We summarize the algorithm of Planner, as shown in 1.

The Planner releases the memory region of a protocol context when its messages are no longer needed by the remote party. It manages the remote memory usage with a dependency graph generated from the model structure. As shown in Figure 6, each node represents data that correspond to messages in the protocols. The convolution is followed by the activation. For every layer, the Planner associates the protocol context with the layer's data. When the layer's data is no longer used, the Planner releases the memory occupied by this layer. For example, in Figure 6, if X_2 has finished, CORA knows the messages produced during the convolution will no longer be used and releases the memory region used by the protocol context. We list the algorithm of releasing memory in List 1 line 12-17.

Besides machine learning operations, it also brings communication to generate offline data and convert protocols as mentioned in §3.2.4 and §3.2.3. Messages for offline data generation do not depend on messages in machine learning layers. Therefore, the Planner allocates a fixed memory region for offline protocol's context. For protocol conversion, the Planner regards it as an extra layer in the model.

4 Implementation

The core component of CORA is the protocol context. The Parser creates protocol contexts based on the model structure. The Planner initializes the protocol context with the current layer and message sizes. When sending a message, the message address indicated by the protocol context is encoded as a notification to the receiver.

Send Sending messages is implemented using *write* verb with immediate data, which writes data directly to the specified address on the remote host with an additional 32-bit immediate number carried by the verb. The immediate number not only notifies task completion but also carries 32-bit extra data besides the payload. CORA configures the four most significant bits to indicate the protocol, and other bits to indicate the offset. For example, CORA sets 0000 as the most significant bits of the immediate data to indicate OT protocol.

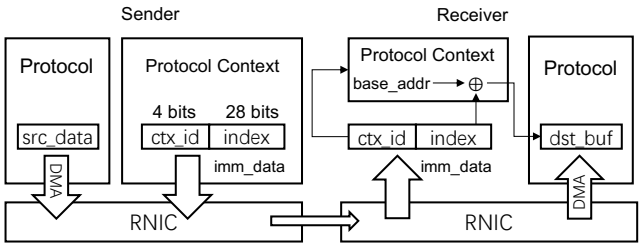


Figure 7: Implementation of sending a message with protocol context.

The remaining bits indicate the offset, which is interpreted as the address on receiving buffer.

Receive On the receiver side, the RNIC will generate a new element in the completion queue when a message arrives. CORA sets a dedicated thread to handle completion elements. It interprets the 32-bit immediate data as the protocol and offset. Next, a completion flag is set to inform upper-layer protocols of a new message arrival.

Note that the immediate number has four bits to encode the protocol contexts, allowing 16 protocol contexts, which are often less than the total number of layers needed. However, the four-bit encoding is enough in runtime because the number of concurrent protocols is usually less than 16. For instance, the parties may only need four protocol contexts for the current layer, the last layer, the next layer, and the offline protocol that generates Beaver triples.

CORA hides the details of sending/receiving from secure protocols by providing socket-like APIs. When CORA executes a layer in a machine learning task, it loads the corresponding protocol context, which contains a base address that points to the remote buffer. The messages are written to the remote address with the offsets associated with the indexes in the array. When a protocol needs to receive a message, it polls for the arrival of that message by checking the finish state with the message id.

Multi-Threading Some frameworks use multiple threads to accelerate computation. There are mainly two cases when using CORA in multithreaded frameworks: The case where worker threads synchronize with only one thread handles communication, and the case where multiple worker threads handle communication.

For the first case, CORA is set up in the communicating thread. The usage of CORA stays the same as the single-thread case; for the second case, we initialize multiple CORA objects in each thread. Each thread binds a unique port, resulting in multiple RDMA connections. Unlike single-threading, multi-threading CORA needs the index of the corresponding element in the global input array when sending or receiving a message because the indexes of the threads' input may overlap. The global index can be converted from the local index and the thread index.

CORA provides a high-level interface for integrating into

Framework	#Existing Lines	#Lines Modified
EMP-OT [3]	1873	54
ABY [20]	20209	141
Piranha [58]	17976	47

Table 1: Number of lines in frameworks and number of lines need to modify in order to utilize CORA

existing SCML frameworks easily. We revised the implementations of Piranha, ABY, and EMP-OT. The numbers of codes changed are shown in Table 1. Although ABY and EMP-OT are not designed for machine learning tasks, they can also use CORA to accelerate communication when using multiple protocols.

```
1
2  void SCML_Layer(Matrix *inputs, Matrix *outputs,
       protocol *p) {
3    // high-level description of the SCML layer
4    // first execute some codes to perform local
       computation
5    Matrix *tmp = local_preprocessing(inputs);
6    // then prepare the messages
7    Matrix *source_data = prepare_message(tmp);
8    // get the protocol context
9    CORA.select_protocol_context(p);
10   // perform interaction
11   CORA.send(source_data);
12   Matrix *recv_data = CORA.recv_msg();
13   // there may be other rounds of interaction
14   // ...
15   // post processing, write output
16   outputs = post_processing(inputs, recv_data);
17 }
```

Listing 1: A general description of how CORA is used in existing frameworks.

Code 1 shows the general procedure of using CORA in existing frameworks. Note that the Matrix data structure and APIs are for demonstration and may differ with frameworks. Existing SCML frameworks typically use Linux sockets to send/receive messages. To send a message, they use send API or write to the socket's file descriptor. CORA replaces this routine with protocol-aware RDMA write. It first loads the protocol context to get the protocol id and remote address for the next write operation. Then we call send API of CORA, which posts a work request to the RNIC and returns. The message should not be modified until the transmission finishes.

For message receiving, CORA replaces this routine with its receive API. The protocol context stores the address of the next message and a flag variable indicating the message's arrival. CORA returns the message's address to the caller when the flag is set. The address is configured at compile time by Planner so that other incoming RDMA operations will not overwrite the message (§3.3).

5 Security of CORA

CORA relies on existing security mechanisms of RDMA to provide isolation of parties. First, a party needs the remote server's key generated specifically for the registered memory region before accessing. Second, a party cannot access other parties' remote memory regions because it lacks other regions' keys. The isolation guarantees that the parties cannot steal other parties' messages.

It is possible to have security issues when using RDMA. For example, the parties may mistakenly store sensitive data in the memory regions accessible by other parties so that other parties can read the sensitive data without notifying the owners. The unintended vulnerabilities may require extra examination in runtime.

Compared to existing SCML frameworks, CORA exposes the memory access pattern to other parties for efficiency. The memory access pattern will not cause security issues because MPC protocols are input-irrelevant. The memory access pattern does not expose information about the secret input.

Moreover, the security of RDMA hardware has been improving. ReDMArk [49] gave mitigation mechanisms to fix RDMA vulnerabilities. Later, Bedrock [59] proposed a hardware-based defense system to secure RDMA without performance penalty. It is expected that the security of RDMA system will continue to be enhanced in the future.

Besides hardware security, the parties may need secure channels to prevent overhearing by adversaries. Recent works have proposed secure channels over RDMA. For example, sRDMA [53] provided efficient authentication and encryption for RDMA to prevent information leakage and message tampering; PANIC [38] implemented multi-tenant isolation in a public cloud. Moreover, the latest commercial RNICs have supported autonomous TLS offload [46], which achieves better performance than TLS on CPU. The TLS offloading has shown the RNIC's efficiency in packet encryption and decryption. Thus, RDMA can bring both performance improvement and security guarantee to applications.

6 Evaluation

Evaluation Overview We summarize the test cases and the results as follows:

- Performance on SCML building blocks, including offline protocols, machine learning layers including matrix multiplication, convolution, and ReLU. The result shows that CORA brings $1.8\times - 7.2\times$ speedup over TCP and $1.6\times - 6.6\times$ speedup over rsocket on basic building blocks (§6.3).
- Performance on model training and inference, including SecureML, LeNet, VGG16, and AlexNet. The result shows that CORA brings $1.2\times - 4.2\times$ speedup over TCP and $1.1\times - 3.9\times$ speedup over rsocket (§6.4).

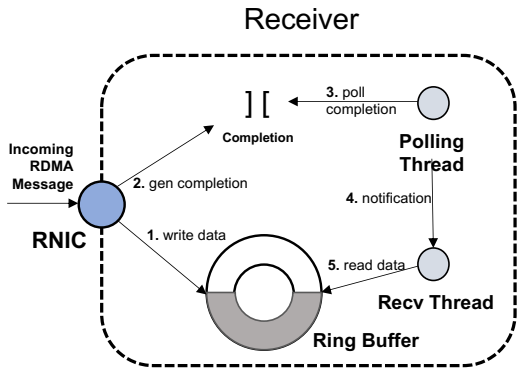


Figure 8: Illustration of rsocket that uses unified ring buffer to send/receive data

- We compare CORA's communication cost to existing transport optimization, including Transputation [12] and MOTION [13] (§6.5).
- Performance Decomposition. We recorded the communication and computation costs during training. We show that CORA reduces communication cost by $2.9\times - 11.0\times$. Communication is no longer the bottleneck of SCML (§6.6).

CORA focuses on deploying SCML in a single datacenter, where servers are interconnected with RDMA network (§2.1). Some SCML frameworks deploy SCML in multiple datacenters. We show that the performance of CORA in inter-datacenter networks is bounded by the propagation delay between datacenters (§6.6).

6.1 Experiment Settings

We run our experiments in a computing cluster with 100 Gbps Mellanox ConnectX-5 RDMA NICs. Each server has an Intel Xeon 5218R CPU (40 cores), 128 GB memory, and a Nvidia Tesla V100 GPU. The ping between servers is about 0.1 ms.

We assume that each party in SCML has one server. Both TCP-based and RDMA-based experiments use the RNIC. RNIC supports both TCP and RDMA traffic by automatically switching between TCP and RDMA modes according to the way of accessing the network interface.

Baseline We compare CORA against the default TCP network in SCML tasks. In addition, to show the benefits of protocol-aware RDMA, we show the performance of CORA over rsocket [7], a general-purpose RDMA library that uses a unified ring buffer to send/receive data, as shown in Figure 8. rsocket sets a unified buffer on the receiver side. When a new RDMA message arrives, RNIC writes the message to the buffer's tail and generates completion. The receiver has a polling thread that notifies the receiving thread when getting a new completion. Then, the receiving thread reads new messages from the buffer's head.

Note that Piranha utilizes GPU and stores data in GPU memory. However, RDMA only supports accessing host memory by default. We use GPUDirect RDMA [2], a built-in feature from CUDA 11.4, so that RDMA can access the GPU's memory just like accessing the host memory.

Test Cases We evaluate the speedup on the following tasks:

Offline Protocols Piranha ignores the cost of the offline phase, which involves generating multiplication triples, and only implements the online phase. However, the offline phase takes a large portion of the total cost when taken into account. Existing SCML frameworks generate multiplication triples based on OT (§3.2.4). We integrate CORA in EMP-OT to test the performance of OT and triple generation.

Basic Building Blocks For basic building blocks, we tested matrix multiplication, convolution, and ReLU activation. We tested the speedup under different sizes of input and different numbers of parties. The matrix multiplications and convolutions are implemented with arithmetic secret sharing. The ReLU activation is based on GC.

Model Training For the whole training process, we tested models in SecureML, LeNet, VGG16, and AlexNet. SecureML and LeNet are trained on MNIST [5] dataset; VGG16 and AlexNet are trained on CIFAR-10 [1] dataset. Each case is implemented with two-party semi-honest (2PC) [44], three-party replicated-ring (3PC) [43], and four-party replicated-ring (4PC) [19] protocols, respectively. These protocols are most widely studied in existing SCML frameworks. All protocols had their field set as 32 bits. The computation on float-point arithmetic is the same as the implementation of SCML frameworks.

Inference When performing an inference task, we pay more attention to the time spent to infer a single data sample because it represents the end-to-end latency of performing an inference task. We tested inference performance with the same model as in the training experiment. We set the input batch size as one to test the inference performance.

6.2 Offline Protocols

One of the most significant costs in the offline phase is to generate Beaver triples [10], which are used when multiplying two secret sharings. Existing works have implemented triple generation based on OT. We test the triple generation based on IKNP OT [9] implemented by EMP-OT [3] library.

For the performance of CORA in OT, we integrate CORA in EMP-OT to test the speedup on different OT protocols, including IKNP and Ferret [61]. The implementations of OT protocols assume semi-honest security settings. We tested TCP, rsocket, and CORA in OT, respectively, and the result is shown in Figure 9. CORA achieves $1.3\times - 4.5\times$ speedup over TCP and $1.1\times - 4.2\times$ speedup over rsocket. The performance of rsocket is bounded by the message copy on both

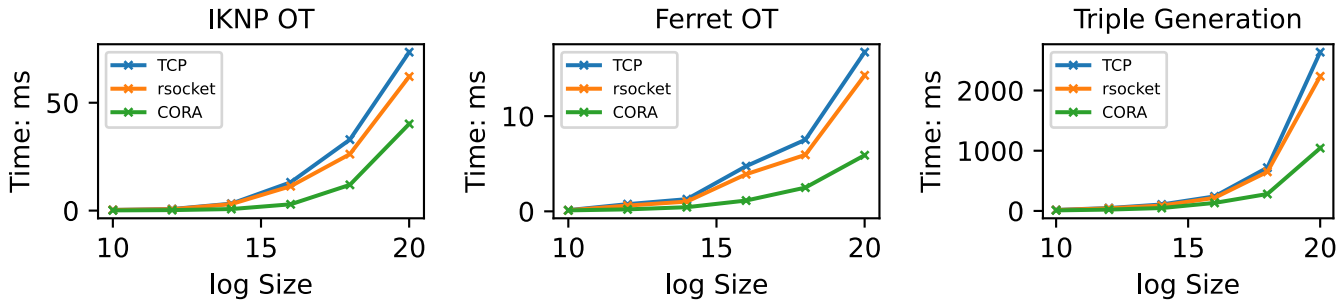


Figure 9: Time elapsed for conducting OT and triple generation.

sides. In the OT task, the sender generates and sends an array of ciphertext to the receiver. On the sender side, rsocket implements the sending task by copying the message to a unified buffer on the local host and posts RDMA *write*. On the receiver side, rsocket copies the received data to the OT buffer. Compared to rsocket, CORA reads source data from the sender's buffer and writes to the receiver's buffer directly without an extra copy.

OT-based Beaver triple generation is performed with a batch of OT tasks [20]. We integrate CORA into ABY, a two-party computation framework that implements the triple generation based on semi-honest IKNP OT with two parties. The width of the triple element is set as 32 bits. For each triple, the parties need 32 OTs. Figure 9 shows the performance improvement. In the figure, N stands for the number of triples to generate. CORA achieves $1.8\times-2.6\times$ speedup over TCP and $1.6\times-2.3\times$ speedup over rsocket.

6.3 Basic Building Blocks

We compare CORA with the default TCP network and rsocket by integrating into Piranha, a GPU-based SCML framework. We run the basic building blocks, including matrix multiplications, convolutions, and activations. In general, CORA achieves $1.8\times-7.2\times$ speedup over TCP and $1.6\times-6.6\times$ speedup over rsocket. Figure 10 summarizes the result for each basic building block.

Matrix multiplication is tested by multiplying two $N\times N$ matrices using arithmetic secret sharing under a semi-honest security setting. The offline phase generates triples with IKNP OT. The matrix multiplication is decomposed as a series of vector dot products. CORA achieves $2.1\times-4.9\times$ speedup over TCP and $1.8\times-4.3\times$ speedup over rsocket.

Essentially, convolution is also decomposed as matrix multiplications. We chose the size of convolution layers that are commonly used in model training and calculated the number of total basic multiplications. In model training on the MNIST dataset and the CIFAR-10 dataset, the convolution cost is from 1.6×10^6 to 5.7×10^8. CORA achieves $1.8\times-4.7\times$ speedup over TCP and $1.6\times-3.5\times$ speedup over rsocket.

For ReLU activation, the operation is implemented by multiplying with a flag, which is generated by comparing with zero. The test is conducted on the implementation of Piranha based on edaBits. The communication task involves one round of reconstruction between parties. CORA achieves $3.6\times-7.2\times$ speedup over TCP and $2.6\times-6.6\times$ speedup over rsocket.

The implementation of basic building blocks usually involves only one round of interaction: matrix multiplication protocol reconstructs the secrets with offset among parties, and ReLU with edaBits reconstructs the masked input with one round of interaction. CORA mainly benefits from the high bandwidth and the zero-copy characteristic compared to TCP and rsocket.

6.4 Model Training and Inference

The training process relies on forward propagations and backward propagations to update model parameters. We set other task parameters the same as the Piranha's, such as the bits reserved for float arithmetic and sizes of ciphertexts. We trained for ten epochs on each model and calculated the average cost for training each batch of data.

The performance of CORA on the whole training process is shown in Table 2. CORA achieves $1.2\times-4.2\times$ speedup over TCP and $1.1\times-3.9\times$ speedup over rsocket. Besides the basic building blocks, the speedup of CORA also comes from the protocol conversion when performing activation on linearly secret sharings (§3.2.3) and the backward propagation process.

6.5 Comparison to TCP Optimizations

To show how CORA improves communication performance compared to other transport optimizations, we tested machine learning tasks using Transputation [12] and MOTION [13]. Transputation, as an independent communication module, is integrated into Piranha for testing. MOTION, however, is a full-stack framework. We only compare their time spent in communication, as shown in Table 3.

Compared to the other two frameworks, CORA has far less communication cost. The speedup over the other frameworks can be explained from two aspects: First, Transputation and MOTION configures the TCP sockets for high-latency

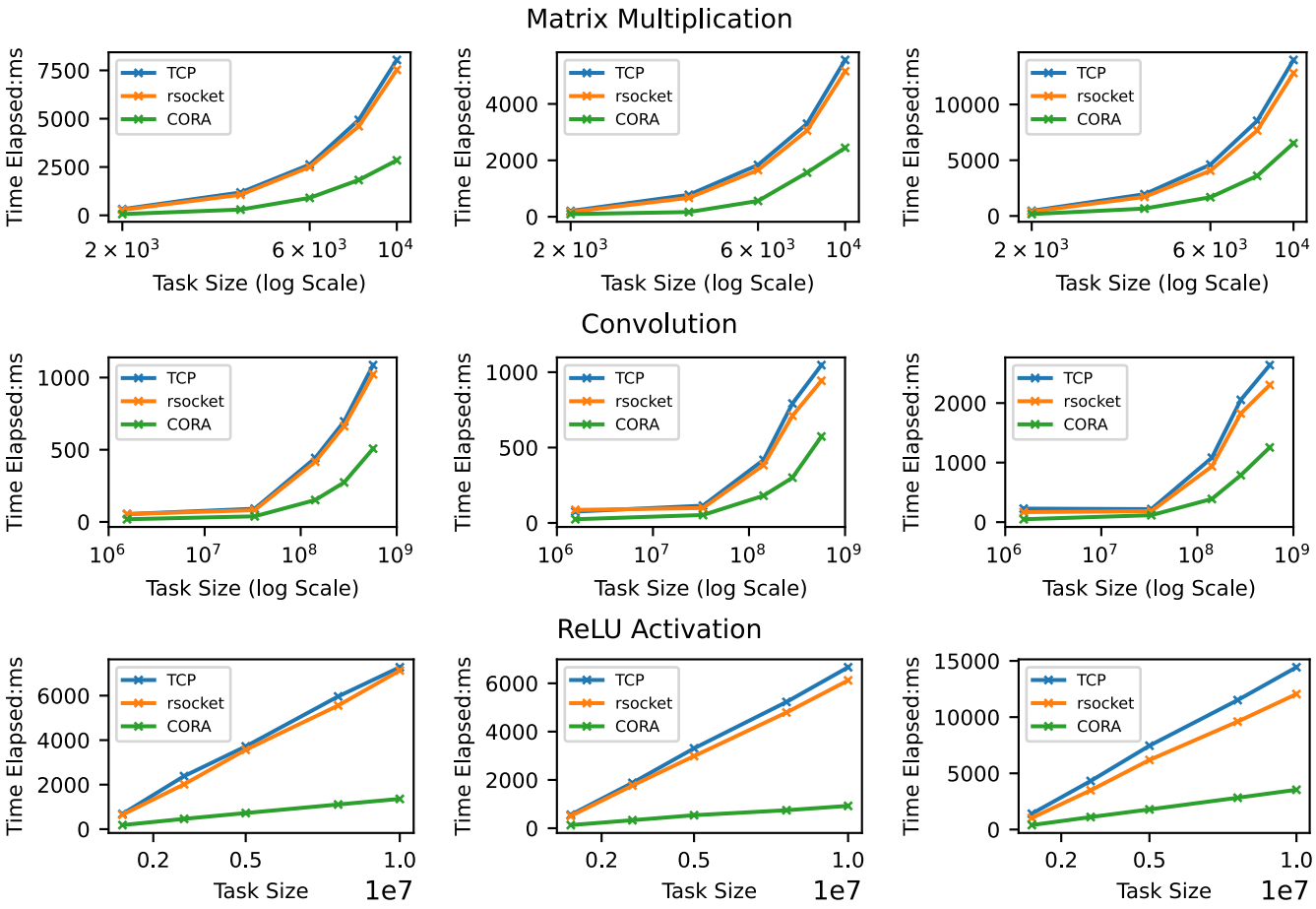


Figure 10: Performance of basic building blocks after applying CORA.

network environments, such as non-blocking operations. However, datacenter provides a low-latency interconnect, and it is more important to optimize data transmission at the end host. Second, TCP-based optimization requires CPU cycles for packet processing. When CPU cores are busy with computing, the communication is delayed.

6.6 Performance Decomposition

To inspect how CORA achieves better performance over rsocket and TCP, we profile the communication and computation costs of the training task. We show that CORA has a much smaller communication cost compared to rsocket and TCP.

Figure 11 illustrates the decomposed cost for model training. Compared to CORA, rsocket and TCP have much higher communication cost, which dominates the total cost. In contrast, CORA has much lower communication cost, and the main cost shifts to computation overhead.

When sending a message, rsocket first copies the message into the buffer on the local host and sends the message using RDMA *write*. On the receiver side, rsocket monitors the arrival of new messages by checking the head pointer of the buffer, which can only detect the arrival of messages adjacent to the head pointer. CORA is more efficient because it does not involve copies during sending, and the receiver can get the addresses of arrived messages by checking the immediate data.

Training in Inter-Datacenter Network CORA relies on the high bandwidth and low latency of RDMA network in the same datacenter. For the case where SCML parties reside in multiple datacenters, the network performance is bounded by the bandwidth and propagation delay between datacenters [65]. For example, the Round Trip Time (RTT) between two datacenters across continents could be over 60 ms, and the bandwidth is less than 10 Gbps, compared to an intra-datacenter network that has 100 Gbps bandwidth and $< 1ms$ latency.

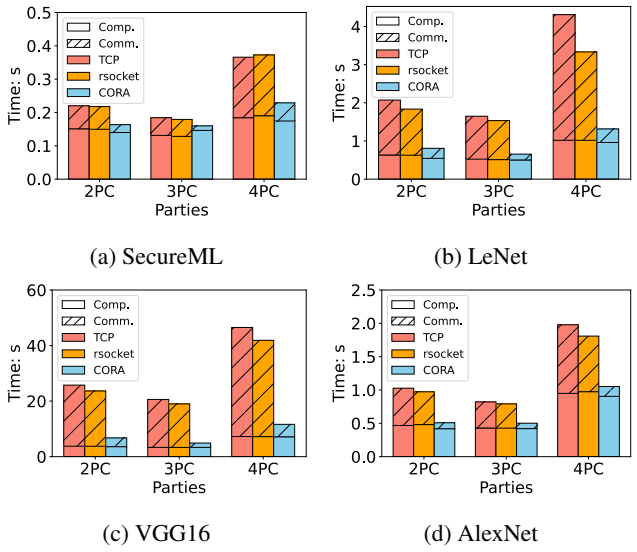
We show that the performance of SCML in the inter-datacenter is bounded by the propagation delay after using CORA. We simulate the inter-datacenter network by limiting the bandwidth to 10 Gbps and adding 10 ms RTT between servers. We decompose the communication cost into transmission cost at the end host, including data copy and transmission by RNIC, and the propagation cost. The result is shown in

	Model	SecureML			LeNet			VGG16			AlexNet		
	# Parties	2	3	4	2	3	4	2	3	4	2	3	4
Train	TCP	220	184	366	2072	1649	4309	25760	20609	46513	1027	821	1978
	rsocket	218	179	373	1836	1534	3332	23648	18992	41886	973	791	1809
	CORA	163	160	229	808	656	1315	6753	4904	11643	509	502	1052
Speedup	/ TCP	1.3	1.2	1.6	2.6	2.5	3.3	3.8	4.2	4.0	2.0	1.6	1.9
	/ rsocket	1.3	1.1	1.6	2.3	2.3	2.5	3.5	3.9	3.6	1.9	1.6	1.7
Inference	TCP	50	29	102	140	87	326	864	491	1422	266	156	470
	rsocket	44	25	97	135	83	281	832	455	1352	245	147	422
	CORA	40	23	56	111	61	162	500	359	885	192	136	326
Speedup	/ TCP	1.2	1.2	1.8	1.3	1.4	2.0	1.7	1.4	1.6	1.4	1.2	1.4
	/ rsocket	1.1	1.1	1.7	1.2	1.4	1.7	1.7	1.3	1.5	1.3	1.1	1.3

Table 2: Average time spent (ms) for training (batch = 128) and inference on different models and different numbers of parties.

	SecureML			LeNet			VGG16			AlexNet		
#Parties	2	3	4	2	3	4	2	3	4	2	3	4
CORA	24	14	54	263	153	354	3141	1566	4481	89	81	148
Transputation [12]	52	44	143	1225	988	2885	19223	14471	30422	325	223	667
MOTION [13]	119	127	154	2206	2427	2935	24261	25442	28334	922	1077	1129

Table 3: Communication Time (ms) in the machine learning training (batch = 128) task.



(a) SecureML (b) LeNet

(c) VGG16 (d) AlexNet

Figure 11: Cost decomposition of training different models on different numbers of parties.

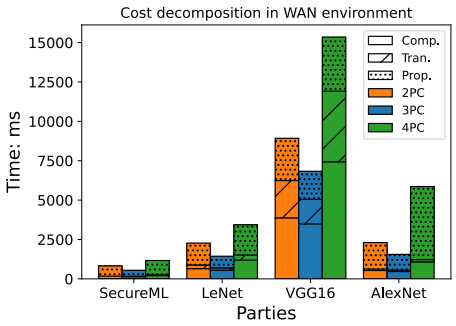


Figure 12: Decomposition of cost in the inter-datacenter environment, where the bandwidth is limited to 10Gbps and RTT=10ms.

Figure 12. It shows that in a network condition where the RTT is 10ms, the propagation cost takes at least nearly half of the total communication cost.

7 Discussion

Performance in Inter-Datacenter Network CORA focuses on accelerating SCML in a intra-datacenter network, where communication bottleneck exists at end hosts. If the parties of SCML reside in multiple datacenters, the network latency between parties could be hundreds of milliseconds and the bandwidth is usually less than 10 Gbps. In this case, the performance of SCML is bounded by the inter-datacenter network, which is out of the scope of this work.

Even with the marginal improvement in the inter-datacenter setting, RDMA is better than TCP in SCML. First, intra-datacenter environment is preferred for machine learning tasks, considering that companies and individuals have already stored their data in datacenters. Recent works such as ABY3 [43], Sharemind [6, 11], PrivPy [37] have deployed SCML among multiple servers in intra-datacenter environment. Second, SCML applications have higher utilization of CPU resources with RDMA, because RDMA implements the data transmission in hardware and does not need consume extra CPU cycles for packet processing.

Applying to General Secure Computation CORA can be extended to general MPC frameworks that use graph-based description of tasks, so that CORA parser can identify the protocol contexts. For example, ABY describes the MPC tasks with a graph, in which the nodes represent gates and the edges represent wires. CORA can be applied to ABY with a new frontend that parses circuit files and produces protocol contexts that match the execution of circuits. Moreover, ABY circuit files can be analyzed by CORA Planner to find the dependency of secret sharings, so that the memory can be reused efficiently. We have applied CORA to ABY, as listed

in Table table 1.

For secure computation frameworks that do not have a high-level description, CORA can be used by managing the receiving buffer as a ring. The developer defines the protocols used in the secure computation program, and CORA creates the buffers that are used specifically for the protocols. When sending a message, CORA writes to the end of the corresponding buffer of the protocol. On the receiving side, CORA passes the addresses of messages to upper-layer protocols. When the receiver consumes messages, it notifies the sender so that the buffer can be reused by other messages.

That said, it might be challenging to apply CORA to general secure computation frameworks. First, some frameworks have optimizations that may change the execution order. For example, MP-SPDZ sorts the computation graph in topology order which results in a different execution order as the task file's, which requires more efforts in order to embed CORA to MP-SPDZ runtime. Second, some frameworks, such as Opaque [68], relies on hardware enclaves that encrypt the memory which can be decrypted only inside the processor. A co-design of RDMA and hardware enclave is needed in order to perform RDMA operation to encrypted remote memory.

Limitations CORA requires that all parties in SCML have RDMA hardwares. If some parties lack RDMA support, they can use soft-ROCE [8], a software implementation of RDMA that does not require dedicated hardware to run RDMA programs. However, soft-ROCE is slower than hardware RDMA, as the packet processing in the software is less efficient than RDMA hardware. Moreover, RDMA performance suffers from bursty and lossy network environments, as there might be head-of-line blocking [71] or deadlock [28], and the packet retransmission in current hardwares is not efficient either. Researchers have been working on new RDMA solutions to address these issues. For example, Tagger [29] proposed efficient deadlock handling, and SRNIC [57] implemented efficient retransmission for RDMA.

8 Related Work

There have been many efforts that could accelerate SCML. We classify existing works with their methods and compare them against CORA.

Protocol Optimizations focus on designing more efficient protocols for secure computation tasks. For example, GAZELLE [31] and MiniONN [41] focused on efficient secure inference on trained models using OT protocols; ABY3 [43] proposed an efficient honest-majority protocol for machine learning among three parties; Falcon [56] proposed an honest-majority and maliciously secure protocol to train neural networks among three parties; Cerebro [69] built a framework for general SCML applications and automatically optimize the circuit layout; Sphinx [54] proposed new protocols for efficient secure online learning; FedSVD [17] accelerated secure singular vector decomposition protocol on large datasets. SOLAR [47] proposed new secret sharing protocol that automatically balance between computation and communication. CORA could bring extra benefit to these works by accelerating communication between parties.

Computation Optimizations accelerate SCML by leveraging hardware such as multi-core CPU, GPU, and FPGA [67]. CPU-based accelerations, such as [14], implemented protocols in parallel to fully utilize CPU cores; GPU-based accelerations, such as HAFLO [18], CryptGPU [52], and Piranha [58], accelerated SCML computation with massive parallel GPU threads. FPGA-based accelerations such as [62] and FLASH [66] accelerated computation with efficient and dedicated FPGA circuits. Computation optimizations are orthogonal to CORA, which could be combined together.

Network Optimizations use other transport protocols to accelerate communication. For example, Shrishak et al. [50] used UDP protocol in lossless network environments, and they chose UDT protocol in Wide Area Network (WAN) environments. Brandt et al. [12] proposed an optimal transport protocol that can adjust to different network environments for secure computation. MOTION [13] adopted full communication serialization that enabled MPC over arbitrary messaging interfaces and removed the need to own network sockets. These works identified TCP sockets as the bottleneck of communication and tried to replace the TCP socket with a more efficient socket system. CORA distinguishes itself with an offloaded transport on hardware, which is more efficient than software transport in network applications. Moreover, CORA combines the SCML communication pattern and RDMA interface to process messages efficiently.

9 Summary

This paper presented CORA to implement SCML over the RDMA network. CORA improves the efficiency of using RDMA in SCML workload by using a protocol-aware design. It maintains socket-like APIs that can be easily integrated into existing SCML frameworks. We compare CORA with TCP and rsocket, a common practice of RDMA without a protocol-aware mechanism, under the SCML building blocks, model training, and inference. Our results show that CORA achieves $1.2\times - 4.2\times$ speedup over TCP and rsocket. CORA's source code is publicly available (https://github.com/renzh1998/CORA).

Acknowledgements

We thank the reviewers and shepherd for their valuable feedback to the paper. This work is supported in part by Hong Kong RGC TRS T41-603/20-R, GRF-16213621, ITF-ACCESS, NSFC Grant 62062005, Key-Area R&D Program of Guangdong (2021B0101400001), an Alibaba Research Grant, and the Turing AI Computing Cloud (TACC) [60]. Kai Chen is the corresponding author.

References

[1] The cifar-10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html.

[2] Developing a linux kernel module using gpudirect rdma. https://docs.nvidia.com/cuda/gpudirect-rdma/.

[3] emp-ot,oblivious transfer, oblivious transfer extension and variations. https://github.com/emp-toolkit/emp-ot.

[4] Infiniband trade association. rocev2. https://cw.infinibandta.org/document/dl/7781.

[5] Mnist handwritten digit database, yann lecun, corinna. https://yann.lecun.com/exdb/mnist/.

[6] Privacy enhancing technologies for data driven business | sharemind. https://sharemind.cyber.ee/.

[7] rsocket(7) - linux man page. https://github.com/linux-rdma/rdma-core/blob/master/librdmacm/docs/rsocket.

[8] Software rdma over converged ethernet. https://github.com/SoftRoCE.

[9] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 535–548, 2013.

[10] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in cryptology — CRYPTO '91*, pages 420–432, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[11] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: a framework for fast privacy-preserving computations. Cryptology ePrint Archive, Paper 2008/289, 2008. https://eprint.iacr.org/2008/289.

[12] Markus Brandt, Claudio Orlandi, Kris Shrishak, and Haya Shulman. Optimal transport layer for secure computation. Cryptology ePrint Archive, Paper 2019/836, 2019. https://eprint.iacr.org/2019/836.

[13] Lennart Braun, Daniel Demmler, Thomas Schneider, and Oleksandr Tkachenko. Motion – a framework for mixed-protocol multi-party computation. *ACM Trans. Priv. Secur.*, 25(2), mar 2022.

[14] Niklas Buescher and Stefan Katzenbeisser. Faster secure computation through automatic parallelization. In *24th USENIX security symposium (USENIX security 15)*, pages 531–546, Washington, D.C., August 2015. USENIX Association.

[15] Niklas Büscher, Daniel Demmler, Stefan Katzenbeisser, David Kretzmer, and Thomas Schneider. Hycc: Compilation of hybrid protocols for practical secure computation. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 847–861, New York, NY, USA, 2018. Association for Computing Machinery.

[16] Qizhe Cai, Shubham Chaudhary, Midhul Vuppalapati, Jaehyun Hwang, and Rachit Agarwal. Understanding host network stack overheads. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 65–77, New York, NY, USA, 2021. Association for Computing Machinery.

[17] Di Chai, Leye Wang, Junxue Zhang, Liu Yang, Shuowei Cai, Kai Chen, and Qiang Yang. Practical lossless federated singular vector decomposition over billion-scale data. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, August 2022.

[18] Xiaodian Cheng, Wanhang Lu, Xinyang Huang, Shuihai Hu, and Kai Chen. Haflo: Gpu-based acceleration for federated logistic regression, 2021.

[19] Anders Dalskov, Daniel Escudero, and Marcel Keller. Fantastic four: Honest-majority four-party secure computation with malicious security. Cryptology ePrint Archive, Paper 2020/1330, 2020. https://eprint.iacr.org/2020/1330.

[20] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - a framework for efficient mixed-protocol secure two-party computation. In *Proceedings 2015 Network and Distributed System Security Symposium*. Internet Society, 2015.

[21] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. *33rd International Conference on Machine Learning, ICML 2016*, 1:342–351, 2016. ISBN: 9781510829008.

[22] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. FaRM: Fast Remote Memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 401–414, Seattle, WA, April 2014. USENIX Association.

[23] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for mpc over mixed arithmetic-binary circuits. Cryptology ePrint Archive, Paper 2020/338, 2020. https://eprint.iacr.org/2020/338.

[24] Philipp Fent, Alexander van Renen, Andreas Kipf, Viktor Leis, Thomas Neumann, and Alfons Kemper. Low-Latency Communication for Fast DBMS Using RDMA and Shared Memory. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1477–1488, 2020.

[25] William Gropp. MPICH2: A New Start for MPI Implementations. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 7–7. Springer Berlin Heidelberg, 2002.

[26] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 202–215, New York, NY, USA, 2016. Association for Computing Machinery.

[27] Sangjin Han, Scott Marshall, Byung-Gon Chun, and Sylvia Ratnasamy. MegaPipe: A New Programming Interface for Scalable Network I/O. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 135–148, USA, 2012. USENIX Association. event-place: Hollywood, CA, USA.

[28] Shuihai Hu, Yibo Zhu, Peng Cheng, Chuanxiong Guo, Kun Tan, Jitendra Padhye, and Kai Chen. Deadlocks in datacenter networks: Why do they form, and how to avoid them. In *HotNets 2016*.

[29] Shuihai Hu, Yibo Zhu, Peng Cheng, Chuanxiong Guo, Kun Tan, Jitendra Padhye, and Kai Chen. Tagger: Practical pfc deadlock prevention in data center networks. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 451–463, 2017.

[30] Zhicong Huang, Wen jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure Two-Party deep neural network inference. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 809–826, Boston, MA, August 2022. USENIX Association.

[31] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX security symposium (USENIX security 18)*, pages 1651–1669, Baltimore, MD, August 2018. USENIX Association.

[32] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Using RDMA Efficiently for Key-Value Services. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 295–306, New York, NY, USA, 2014. Association for Computing Machinery. event-place: Chicago, Illinois, USA.

[33] Marcel Keller. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. ACM, October 2020.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[35] Sam Kumar, David E Culler, and Raluca Ada Popa. Mage: Nearly zero-cost virtual memory for secure computation. In *OSDI*, pages 367–385, 2021.

[36] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[37] Yi Li and Wei Xu. PrivPy. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, July 2019.

[38] Jiaxin Lin, Kiran Patel, Brent E. Stephens, Anirudh Sivaraman, and Aditya Akella. PANIC: A High-Performance programmable NIC for multi-tenant networks. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 243–259. USENIX Association, November 2020.

[39] Xiaofeng Lin, Yu Chen, Xiaodong Li, Junjie Mao, Jiaquan He, Wei Xu, and Yuanchun Shi. Scalable kernel tcp design and implementation for short-lived connections. *SIGPLAN Not.*, 51(4):339–352, mar 2016.

[40] Feilong Liu, Lingyan Yin, and Spyros Blanas. Design and evaluation of an rdma-aware data shuffling operator for parallel database systems. *ACM Trans. Database Syst.*, 44(4), dec 2019.

[41] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious Neural Network Predictions via MiniONN Transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, October 2017.

[42] Wen-jie Lu, Zhicong Huang, Qizhi Zhang, Yuchen Wang, and Cheng Hong. Squirrel: A scalable secure two-party computation framework for training gradient boosting decision tree. In *32st USENIX Security Symposium (USENIX Security 23)*. USENIX Association, 2023.

[43] Payman Mohassel and Peter Rindal. ABY3: A mixed protocol framework for machine learning. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 35–52, 2018. ISBN: 9781450356930.

[44] Payman Mohassel and Yupeng Zhang. SecureML: A System for Scalable Privacy-Preserving Machine Learning. *Proceedings - IEEE Symposium on Security and Privacy*, pages 19–38, 2017. ISBN: 9781509055326.

[45] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: Improved mixed-protocol secure two-party computation, 2020. tex.howpublished: Cryptology ePrint Archive, Report 2020/1225.

[46] Boris Pismenny, Haggai Eran, Aviad Yehezkel, Liran Liss, Adam Morrison, and Dan Tsafrir. Autonomous nic offloads. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, page 18–35, New York, NY, USA, 2021. Association for Computing Machinery.

[47] Zhenghang Ren, Xiaodian Cheng, Mingxuan Fan, Junxue Zhang, and Cheng Hong. Communication efficient secret sharing with dynamic communication-computation conversion. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2023.

[48] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, pages 707–721, New York, NY, USA, 2018. Association for Computing Machinery. event-place: Incheon, Republic of Korea.

[49] Benjamin Rothenberger, Konstantin Taranov, Adrian Perrig, and Torsten Hoefler. Redmark: Bypassing rdma security mechanisms. In *USENIX Security Symposium*, pages 4277–4292, 2021.

[50] Kris Shrishak, Haya Shulman, and Michael Waidner. Removing the bottleneck for practical 2pc. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 2300–2302, New York, NY, USA, 2018. Association for Computing Machinery.

[51] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[52] Sijun Tan, Brian Knott, Yuan Tian, and David J. Wu. Cryptgpu: Fast privacy-preserving machine learning on the gpu. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1021–1038, 2021.

[53] Konstantin Taranov, Benjamin Rothenberger, Adrian Perrig, and Torsten Hoefler. sRDMA – efficient NIC-based authentication and encryption for remote direct memory access. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 691–704. USENIX Association, July 2020.

[54] Han Tian, Chaoliang Zeng, Zhenghang Ren, Di Chai, Junxue Zhang, Kai Chen, and Qiang Yang. Sphinx: Enabling privacy-preserving online learning over the cloud. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2487–2501, 2022.

[55] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019(3):26–49, July 2019.

[56] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. Falcon: Honest-majority maliciously secure framework for private deep learning. *arXiv preprint arXiv:2004.02229*, 2020.

[57] Zilong Wang, Layong Luo, Qingsong Ning, Chaoliang Zeng, Wenxue Li, Xinchen Wan, Peng Xie, Tao Feng, Ke Cheng, Xiongfei Geng, Tianhao Wang, Weicheng Ling, Kejia Huo, Pingbo An, Kui Ji, Shideng Zhang, Bin Xu, Ruiqing Feng, Tao Ding, Kai Chen, and Chuanxiong Guo. SRNIC: A scalable architecture for RDMA NICs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1–14, Boston, MA, April 2023. USENIX Association.

[58] Jean-Luc Watson, Sameer Wagh, and Raluca Ada Popa. Piranha: A GPU Platform for Secure Computation, 2022. Published: Cryptology ePrint Archive, Paper 2022/892.

[59] Jiarong Xing, Kuo-Feng Hsu, Yiming Qiu, Ziyang Yang, Hongyi Liu, and Ang Chen. Bedrock: Programmable network support for secure RDMA systems. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2585–2600, Boston, MA, August 2022. USENIX Association.

[60] Kaiqiang Xu, Xinchen Wan, Hao Wang, Zhenghang Ren, Xudong Liao, Decang Sun, Chaoliang Zeng, and Kai Chen. Tacc: A full-stack cloud computing infrastructure for machine learning tasks. *arXiv preprint arXiv:2110.01556*, 2021.

[61] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast Extension for coRRElated oT with small communication, 2020. Published: Cryptology ePrint Archive, Paper 2020/924.

[62] Zhaoxiong Yang, Shuihai Hu, and Kai Chen. Fpga-based hardware accelerator of homomorphic encryption for efficient federated learning. *CoRR*, abs/2007.10560, 2020.

[63] Kenichi Yasukata, Michio Honda, Douglas Santry, and Lars Eggert. Stackmap:low-latency networking with the os stack and dedicated nics. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 43–56, 2016.

[64] Bairen Yi, Jiacheng Xia, Li Chen, and Kai Chen. Towards zero copy dataflows using rdma. In *Proceedings of the SIGCOMM Posters and Demos*, pages 28–30. 2017.

[65] Gaoxiong Zeng, Wei Bai, Ge Chen, Kai Chen, Dongsu Han, Yibo Zhu, and Lei Cui. Congestion control for cross-datacenter networks. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–12, 2019.

[66] Junxue Zhang, Xiaodian Cheng, Wei Wang, Liu Yang, Jinbin Hu, and Kai Chen. FLASH: Towards a High-performance Hardware Acceleration Architecture for Cross-silo Federated Learning. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1057–1079, Boston, MA, April 2023. USENIX Association.

[67] Junxue Zhang, Xiaodian Cheng, Liu Yang, Jinbin Hu, Ximeng Liu, and Kai Chen. Sok: Fully homomorphic encryption accelerators, 2023.

[68] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 283–298, Boston, MA, March 2017. USENIX Association.

[69] Wenting Zheng, Ryan Deng, Weikeng Chen, Raluca Ada Popa, Aurojit Panda, and Ion Stoica. Cerebro: A Platform for Multi-Party Cryptographic Collaborative Learning. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, August 2021.

[70] Wenting Zheng, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Helen: Maliciously Secure Coopetitive Learning for Linear Models. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2019.

[71] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale rdma deployments. In *SIGCOMM 2015*.