Silentflow: Leveraging Trusted Execution for Resource-Limited MPC via Hardware-Algorithm Co-design

Zhuoran Li*, Hanieh Totonchi Asl*, Ebrahim Nouri*
Yifei Cai†, Danella Zhao*
**Electrical and Computer Engineering*, *University of Arizona*, Tucson, United States
{zli1122, haniehta, ebinouri, danellazhao}@arizona.edu
†*Electrical and Computer Engineering*, *Old Dominion University*, Norfolk, United States
ycai001@odu.edu

***Abstract*—Secure Multi-Party Computation (MPC) offers a practical foundation for privacy-preserving machine learning at the edge, with MPC commonly employed to support non-linear operations. These MPC protocols fundamentally rely on Oblivious Transfer (OT), particularly Correlated OT (COT), to generate correlated randomness essential for secure computation. Although COT generation is efficient in conventional two-party settings with resource-rich participants, it becomes a critical bottleneck in real-world inference on resource-constrained devices (e.g., IoT sensors and wearables), due to both communication latency and limited computational capacity. To enable real-time secure inference, we introduce Silentflow, a highly efficient Trusted Execution Environment (TEE)-assisted protocol that eliminates communication in COT generation. We tackle the core performance bottleneck—low computational intensity—through structured algorithmic decomposition: kernel fusion for parallelism, Blocked On-chip eXpansion (BOX) to improve memory access patterns, and vectorized batch operations to maximize memory bandwidth utilization. Through design space exploration, we balance end-to-end latency and resource demands, achieving up to 39.51× speedup over state-of-the-art protocols. By offloading COT computations to a Zynq-7000 SoC, SilentFlow accelerates PPMLaaS inference on the ImageNet dataset under resource constraints, achieving a 4.62× and 3.95× speedup over Cryptflow2 and Cheetah, respectively.**

***Index Terms*—Security & Privacy, Multiparty Computation, Trusted Execution Environment, FPGA acceleration**

I. Introduction

To address privacy concerns in Machine Learning as a Service (MLaaS), Privacy-Preserving MLaaS (PPMLaaS) incorporates cryptographic primitives to safeguard sensitive data [1–5]. Among these primitives, secure Multi-Party Computation (MPC) has been widely adopted for its effectiveness in handling nonlinear operations [6–10]. A core component of MPC protocols is Oblivious Transfer (OT), which plays a critical role in ensuring data privacy between parties.

Over time, OT protocols have evolved significantly to improve efficiency and scalability. Recent advancements, such as Silent OT [11] and the Ferret protocol [12], demonstrate superior performance compared to classic OT protocols like IKNP [13], particularly in throughput and communication overhead. While the cost of generating usable OTs is often negligible [11, 12, 14–18] in two-party computation (2PC) settings—where both client and server run on resource-rich platforms such as workstations—it becomes a major bottleneck in real-world inference scenarios involving resource-constrained client devices, as shown in Fig. 1.

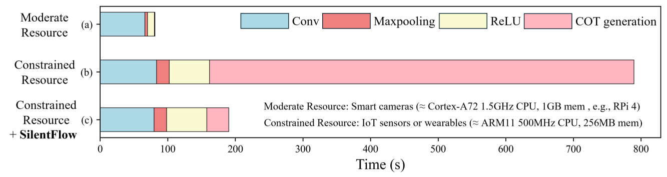

Fig. 1. Resnet-50 inference performance under 2-Party Cheetah [7] framework with a network condition of 200Mbps bandwidth, 50ms latency

Motivations and Challenges. In real-world PPMLaaS deployments—particularly at the edge—clients often run on resource-constrained hardware such as IoT sensors, wearables, or portable medical devices that have limited memory and computational power [19, 20], which introduces two critical challenges. First, limited memory necessitates splitting inference into small batches, as there is insufficient storage to retain large intermediate values—particularly correlated oblivious transfers (COTs), which are essential for secure computation of nonlinear operations like ReLU. As a result, even a single nonlinear layer must be executed batch by batch, with fresh COTs generated for each batch. This repeated COT generation intensifies interactive rounds and results in substantial communication overhead. Second, since COT generation is dominated by low arithmetic intensity and high global memory traffic, state-of-the-art methods [11, 12, 18] suffer performance degradation in resource-limited environments—not due to computational limitations, but primarily due to frequent cache misses when working sets exceed the limited on-chip memory. These frequent cache misses during the generation process lead to inefficient memory access and increased client-side latency. This inefficiency not only increases end-to-end inference time but also wastes server-side resources, as servers often idle while awaiting responses from slow clients.

To address these dual bottlenecks, we propose a set of targeted design optimizations as **contributions**:

- First, we introduce a novel TEE-assisted protocol that eliminates the communication overhead inherent in OT extension. SilentFlow leverages synchronized seeds within each party's TEE to generate a minimal amount of correlated randomness, effectively bridging the gap between prior interactive approaches [11, 12, 15, 18]. This allows both parties to locally derive identical pseudorandom values and establish correlated randomness—traditionally requiring interaction—without any communication, achieving up to a 39.51× speedup over prior approaches §IV-C. In contrast to previous methods that delegate extensive computation to the TEE—potentially exposing client inputs—SilentFlow confines the TEE's role to input-independent COT generation [21–25]. As a result, even if the TEE is compromised, client data and model parameters remain secure throughout the end-to-end PPMLaaS inference.

- Second, to address the memory-bound bottleneck in COT generation, we propose a hardware–algorithm co-design that employs structured algorithmic decomposition to optimize memory access patterns and reduce intermediate data movement. By fusing parallel kernels, introducing Blocked On-chip eXpansion (BOX), applying vectorized batch operations, and exploring the design space to balance performance and resource utilization, SilentFlow achieves scalable, low-latency execution with 10.21× speedup, while using only 160KB of local memory.
- Finally, we conduct extensive experiments by offloading COT generation to a Zynq-7000 SoC, accelerating secure ResNet-50 inference with ImageNet dataset by 4.62× and 3.95× over Cryptflow2 [6] and Cheetah [7], respectively.

II. Preliminaries

A. Oblivious Transfer

OT enables a sender to transmit multiple messages such that the receiver learns only the one matching their selection bit, while the sender learns nothing about the receiver's choice. A complete OT protocol consists of an *input-independent* correlated randomness generation phase and an *input-dependent* oblivious data transfer phase, during which the client's secret message exchange occurs. The former dominates communication cost, while the latter incurs only linear, information-theoretic cost with respect to the number of messages [14].

As illustrated in Fig. 2, the most efficient method for generating the required correlated randomness is COT—a cryptographic primitive in which the sender's messages follow a fixed correlation Δ. Specifically, the sender holds two values v and $v \oplus \Delta$, and the receiver holds a choice bit $u \in \{0, 1\}$ and the corresponding correlated value r_u. In the input-dependent phase, the receiver masks their actual selection bit b by computing $c = b \oplus u$ and sends c to the sender. The sender then returns the masked messages $m_0 \oplus r_c$ and $m_1 \oplus r_{c\oplus 1}$. Using r_u, the receiver recovers m_b while learning nothing about $m_{1\oplus b}$.

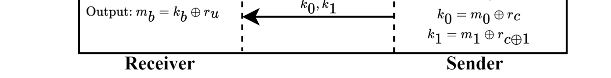

Fig. 2. Oblivious Transfer process

B. Related work and Key Motivations

The constitution of COT is crucial but not trivial. If COTs are constructed purely using public-key cryptographic techniques, each COT instance require expensive public-key operations, leading to $O(\lambda)$ communication and computation per transfer, where λ is the security parameter (typically 128 or 256 bits). State-of-the-art COT generation protocols typically follow a two-stage template: (1) a setup phase that generates sparse correlations using primitives such as Homomorphic Secret Sharing (HSS), Distributed Point Functions (DPF) [11], or OTs [12]; and (2) a non-interactive linear expansion phase based on the Learning Parity with Noise (LPN) assumption [11, 12, 15, 16, 18, 26, 27]. While these designs are efficient on powerful machines, they are ill-suited to resource-constrained environments such as IoT sensors and wearables.

In practice, secure operations in PPMLaaS—such as ReLU and Maxpooling—rely on large volumes of COTs; however, on resource-constrained devices, these operations must be split into small batches, requiring repeated COT generation with fresh correlations for each batch. In such scenarios, although SOTA protocols claim non-interactive extension, the sparse correlation setup still incurs significant communication overhead. For example, recent silent OT-extension protocols [11] require nontrivial key exchanges—typically via DPFs—before parties can expand to a large number of OTs. Ferret [12] generates the required correlations using multi-point COT, which introduces considerable communication costs. Similarly, Boyle et al. [15] use two rounds of OT-based key exchange, relying on Puncturable Pseudorandom Functions (PRFs) with position-specific constraints to construct sparse correlations. SoftSpokenOT [26] applies a related strategy over larger fields, where each correlation block is instantiated using a specialized OT that also demands substantial setup. Despite being labeled "silent," these protocols still require communication in the correlation setup, becoming a performance bottleneck in resource-constrained, batch-based settings.

III. Design of SilentFlow

This section presents the core design of SilentFlow, starting with a high-level architectural overview. §III-B describes how TEEs are leveraged to eliminate interaction during COT generation. Finally, we introduce a hardware accelerator tailored to overcome the computational bottleneck in COT generation. SilentFlow assumes at most one malicious party, achieving malicious security for COT generation via TEE-synchronized randomness, while inheriting semi-honest security for online inference from the underlying MPC framework.

A. System Architecture

As illustrated in Fig. 3, SilentFlow tackles excessive communication overhead in COT generation through four integrated modules: *shared seed generation*, *initial correlation setup*, *sparse correlation constitution*, and *LPN-based local computation*. The first two modules are one-time setup steps; regardless of how many future batches are needed, execution can resume directly from the third module. Together,

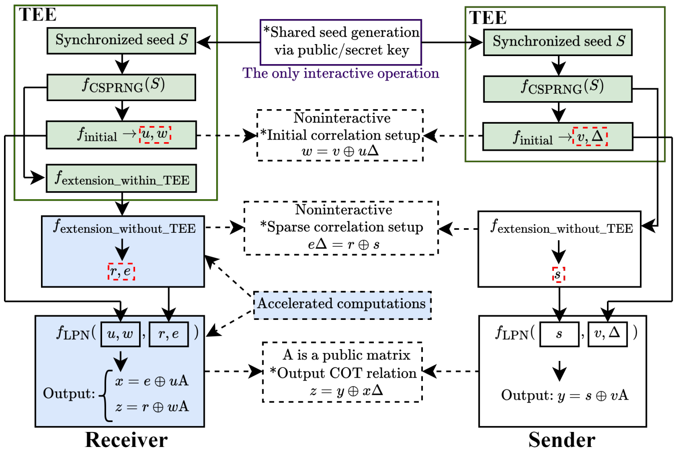
TEE
Synchronized seed S
$f_{\text{CSPRNG}}(S)$
$f_{\text{initial}} \rightarrow u, w$
$f_{\text{extension\_within\_TEE}}$
*Shared seed generation via public/secret key
The only interactive operation
Noninteractive *Initial correlation setup $w = v \oplus u\Delta$
$f_{\text{initial}} \rightarrow v, \Delta$
$f_{\text{extension\_without\_TEE}}$
Noninteractive *Sparse correlation setup $e\Delta = r \oplus s$
r, e
s
Accelerated computations
$f_{\text{LPN}}(u, w, r, e)$
$f_{\text{LPN}}(s, v, \Delta)$
A is a public matrix *Output COT relation $z = y \oplus x\Delta$
Output: $x = e \oplus u\text{A}$, $z = r \oplus w\text{A}$
Output: $y = s \oplus v\text{A}$
Receiver
Sender

Fig. 3. System Overview of SilentFlow

these components eliminate client-server communication during COT generation, supporting efficient small-batch execution without repeated network interaction.

SilentFlow initially establishes correlations of the form $\mathbf{w} = \mathbf{v} \oplus \mathbf{u}\Delta$ non-interactively within the TEE, thereby removing costly base OTs and interaction overhead typical in state-of-the-art protocols such as Ferret [12]. Subsequently, the COT extension module expands these initial correlations locally in the untrusted domain, avoiding the frequent and expensive cross-boundary operations required by prior TEE-based approaches [28]. Crucially, this non-interactive extension generates local correlations in the form of $\mathbf{r} \oplus \mathbf{s} = \mathbf{e}\Delta$, where $\mathbf{e}$ is a sparse binary vector (often Hamming weight). In the final step, the sparse correlation is added to the product of the initial correlation and a public matrix $\mathbf{A}$, ensuring compliance with the LPN assumption and thereby satisfying the security requirements for COT generation.

B. TEE-Assisted COT Generation

Shared Seed Generation. We adopt a synchronized seed generation protocol, similar to the secure initialization in [24], where both parties contribute random values and compute the shared seed as their sum. This ensures mutual unpredictability while allowing both parties to derive consistent CSPRNG outputs. To securely execute this protocol, secret keys and local randomness must remain confidential. Therefore, all cryptographic operations are performed within TEEs, which provide isolated execution and prevent leakage even in the presence of an untrusted client or server.

Initial Correlation Setup. As a synchronized seed is generated, both the sender's and receiver's TEEs deterministically sample the same vectors $\mathbf{u} \in \mathbb{F}_2^k$, $\mathbf{v} \in \mathbb{F}_{2^k}^k$, and the global key $\Delta \in \mathbb{F}_{2^k}$ via CSPRNG (Fig. 3). The receiver's TEE locally computes $\mathbf{w} = \mathbf{v} \oplus \mathbf{u}\Delta$. The sender then sends $\mathbf{v}$, Δ out of TEE, while the receiver sends $\mathbf{u}$, $\mathbf{w}$ out of TEE.

This setup phase avoids the interactive communication required in traditional base OT protocols, reducing initialization cost and enabling fast, communication-free correlation setup. Although CSPRNGs in TEEs incur higher computational latency than lightweight PRNGs, the overhead is minimal, as only a small number of base COTs are needed to bootstrap the later extension phase. Moreover, this one-time cost is amortized over large batches of extended COTs. Compared to prior TEE-based frameworks [28] that generate all COTs inside the enclave, SilentFlow minimizes trusted-side computation by restricting TEE usage to the initial seed and base COTs.

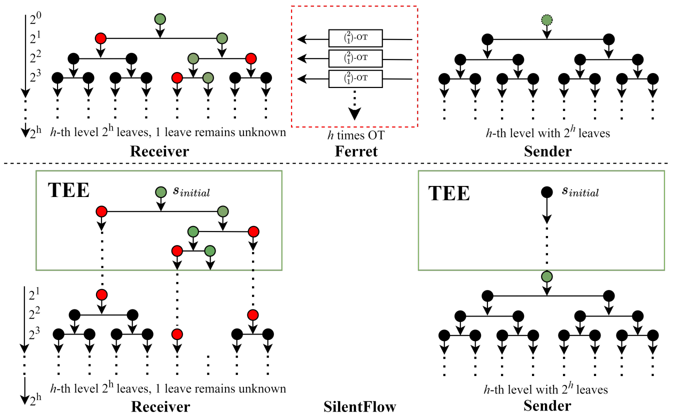

Fig. 4. Design of non-interactive sparse correlation constitution (GGM extension)

Non-interactive Sparse Correlation Constitution. As shown in Fig. 4, we adopt the standard approach of expanding a Goldreich-Goldwasser-Micali (GGM) binary tree to construct the sparse correlation, but with a key distinction—our TEE-assisted protocol enables this process to be entirely non-interactive, a feature not achieved by state-of-the-art works [11, 14, 15, 18]. The constitution involves the receiver obtaining a set of seeds from the sender, where the seed corresponding to the secret index is hidden, and the remaining *red* seeds are extended to the leaves of the tree. To ensure that the receiver obtains only one of two seeds at each tree level, protocols like Ferret [12] perform a 1-out-of-2 OT at every level of the GGM tree. As the receiver traverses this single secret path using the OT-derived seeds, it reaches the h-th level and reconstructs all leaves except the one corresponding to the nonzero entry in $\mathbf{e}$. This process guarantees that the sender and receiver generate correlated pseudorandom vectors $\mathbf{s}$ and $\mathbf{r}$ satisfying $\mathbf{r} + \mathbf{s} = \mathbf{e} \cdot \Delta$. However, this method incurs high communication overhead under resource-limited devices due to low-batch execution, requiring h OT operations per extension, which becomes a major performance bottleneck.

Our key idea is to replace the costly OT-based selection with a synchronized, shared seed within the TEE. Specifically, S_{TEE} and R_{TEE} generate the same initial seed s_{initial} on each side by following the steps outlined in the shared seed generation. S_{TEE} send s_{initial} directly to the untrusted domain for tree extension, while R_{TEE} retains s_{initial} to keep it hidden from the untrusted environment. On the sender's side, the extension is systematic, involving the expansion of s_{initial} to perform 2^{h+1} computations up to level h.

The main challenge lies in extending only the selected seed without revealing its identity to the receiver. To address this, R_{TEE} generates a single value $b \in \{0, 1\}^h$ that encodes the selection bits across the tree levels, where the path defined by b corresponds to either the red or green seed. For each extension at level $i \in \{1, \ldots, h\}$, R_{TEE} retains the green seed along the path determined by b, while the complementary red seed is released to the untrusted domain to enable faster seed expansion. As a key optimization of the TEE-assisted extension compared to prior work [28], only h seed expansion operations are performed inside the TEE, while the remaining $2^{(h+1)} - h$ seed expansions are offloaded to the untrusted domain, inducing only $h - 1$ trust boundary crossings. This avoids costly computation inside the TEE. At each level, the green seed on the secret path remains hidden from the receiver. At the final level, a masked value is computed inside R_{TEE} by adding the global key to the corresponding green leaf. This value is then released to the receiver. As a result, the receiver learns only the masked value and the unmasked red values, while the secret leaf remains hidden. This achieves the same effect as the communication-intensive OT-based approach used in Ferret [12], but with only $h-1$ trust boundary crossings and no interaction between the client and server. For clarity, the figure illustrates a single GGM tree extension, though multiple trees are processed in parallel in practice.

C. Hardware Acceleration of Silentflow

This section introduces a novel hardware–algorithm co-design aimed at accelerating COT generation (particularly, sparse correlation constitution and LPN-based local computation) in Silentflow. Using the Roofline model [29], which relates computational throughput to memory bandwidth and arithmetic intensity, we identify that our baseline algorithm is memory-bound, indicating that performance is limited primarily by memory access rather than computation. To address this, we propose a unified optimization strategy, termed **structured algorithmic decomposition**, which targets latency bottlenecks at both the system and module levels. Our approach improves computational throughput while significantly reducing slow, energy-intensive off-chip data transfers [30].

Latency Optimization via Kernel Fusion. The non-interactive extension phase consists of two serialized phases: GGM tree expansion (for sparse correlation) and LPN computation, typically serialized due to presumed data dependencies. However, our key insight is that the only true dependency lies in the final XOR stage. We thus apply kernel fusion to reorganize the pipeline by decoupling the LPN into two independent sub-stages: a vector-matrix multiplication (VM) and a final XOR. This decoupling transforms the original latency equation from a strictly sequential model (1) to a latency-balanced parallelizable form (2).

$$\text{Latency} = L_{\text{GGM}} + L_{\text{LPN}} \tag{1}$$

$$\text{Latency} = \max(L_{\text{GGM}}, L_{\text{VM}}) + L_{\text{XOR}} \tag{2}$$

To minimize latency, our design balances the computational resources between the GGM and VM modules. While GGM is inherently more compute-intensive, the runtime of both modules depends heavily on resource allocation. By employing structured algorithmic decomposition, we explore an optimal resource allocation strategy to balance latency and resource

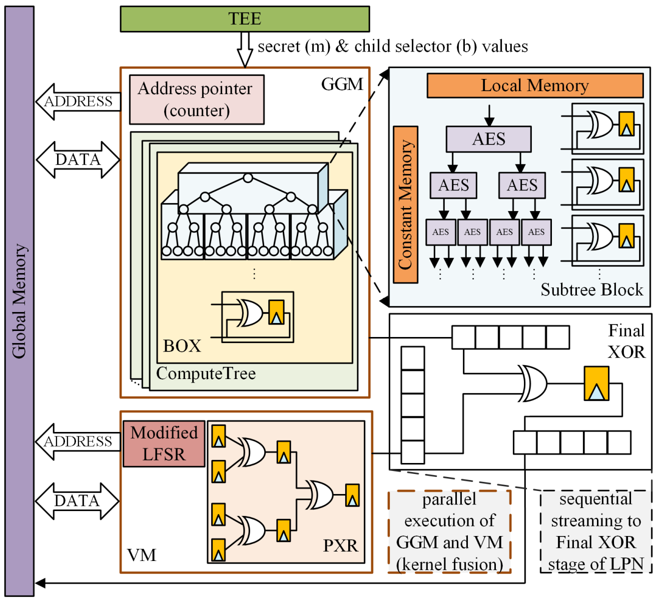
TEE
secret (m) & child selector (b) values
Global Memory
ADDRESS
DATA
Address pointer (counter)
GGM
Local Memory
Constant Memory
AES
AES
AES
AES
AES
AES
AES
Subtree Block
BOX
ComputeTree
Final XOR
ADDRESS
DATA
Modified LFSR
VM
PXR
parallel execution of GGM and VM (kernel fusion)
sequential streaming to Final XOR stage of LPN

Fig. 5. System Decomposition of SilentFlow

usage. This decomposition is applied not only at the system level but also recursively within each module, such as the blocked expansion used for GGM discussed in Section III-C. Fig. 5 illustrates our hardware architecture, highlighting how structured decomposition improves bandwidth utilization, parallelism, and overall execution latency.

GGM Acceleration via Blocked On-chip eXpansion (BOX). Each node expansion in the GGM tree uses a pseudorandom generator (PRG), typically AES in ECB mode [12]. To generate the required n pseudorandom leaves for LPN noise inputs, the baseline method expands multiple binary trees of height h. This expansion is memory-intensive due to frequent global accesses to temporary intermediate nodes, resulting in poor locality, low computational intensity, and exponential memory growth with tree depth h (see Table I).

TABLE I
MEMORY ACCESS COST COMPARISON: NAIVE VS. PROPOSED. C_g/C_l DENOTE AVERAGE GLOBAL/LOCAL ACCESS CYCLES.

Version	Global	Local	Cost
Naive	$6 \times 2^{h-1}$	0	$6 \times 2^{h-1} \cdot C_g$
Proposed	$2 \times 2^{h-1}$	$4 \times 2^{h-1}$	$2 \times 2^{h-1} \cdot (C_g + 2C_l)$

To mitigate this performance bottleneck, we propose a second-level decomposition strategy, **Blocked On-chip eXpansion (BOX)**, which decomposes the GGM expansion into independent subtree blocks of depth s. Each subtree reads its root from global memory, expands locally to generate 2^s leaves, and keeps all intermediate states on-chip. This data localization enables parallel execution of subtrees at the same level, significantly improving reuse and reducing intermediate memory transfers as shown in Table I. Since average global memory access cost (C_g) significantly exceeds the local memory access cost (C_l), our approach reduces memory access overhead by approximately a factor of three.

As shown in Fig. 6, our hardware unrolls inner loops

```
Algorithm: COT Generation
1. for i=0 to t:  //------------------------------------------------------------ GGM
2.     for p=1 to (h-1) step s:
3.         for z=1 to s: //Initialize sublayer XORs
4.         for e=(1<<p)-1 to 0:
5.             for j=0 to s: //---- Subtree block with local computation
6.                 for q=(1<<(j+1))-2 to (1<<j)-1: //AES exp. & sublayer XORs
7.             for f=0 to (1<<s) step 4: //Vectorized write-back with masking
8.  for j=0 to $size_n$ step $size_{batch}$ : //--------------------- VM (parallel to GGM)
9.  for j=0 to $size_n$: //----------------- XOR (combining GGM and VM results)
```

Fig. 6. Pseudocode of the proposed decomposition algorithm

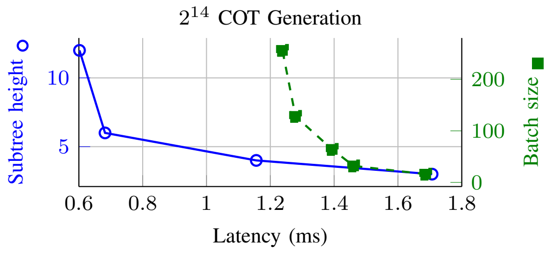

Fig. 7. GGM and VM Latency across varying subtree heights and batch sizes

to exploit fine-grained parallelism and pipelines outer loops of sub-tree block calls to achieve high-throughput dataflow. To avoid redundant global memory accesses at the final depth, masking is applied during leaf node write-back. This optimization leverages the lower latency of TEE-based data access compared to OT, which incurs higher cryptographic and network overhead. Consequently, subtrees can proceed without stalling on OT data. To identify the optimal subtree depth, we explore the design space to balance LPN and GGM latencies, aiming to minimize both absolute latency and latency gap. As shown in Fig. 7, while $batch\_size$=16, s=3 achieves the smallest latency gap, $batch\_size$=256, s=4 offers lower absolute latency with a small gap, providing the best trade-off used in our evaluation.

To further accelerate BOX execution, we optimize AES at the microarchitectural level, as its frequent invocation during node expansion significantly affects performance. As illustrated in Fig. 8, precomputed round keys and S-box values are stored in local RAM to reduce memory overhead and eliminate redundant key expansions. The round key addition is deferred to the final stage to minimize data dependencies and enhance pipelining. Moreover, the deeply nested loop is restructured into a pipeline-friendly format, shortening critical paths and enabling efficient memory transfers across iterations.

```
Algorithm: AES

(a) Alg#1: Conventional
1.  for i=0 to 1:   //Initial round
2.      K=aes_key[i*11+0]
3.      g[i]=g[i]⊕K
4.  for r=1 to 9:   //Main rounds
5.      for i=0 to 1:
6.          K=aes_key[i*11+r]
7.          AESRound(g[i],K,sbox)
8.  for i=0 to 1:   //Final round
9.      K=aes_key[i*11+10]
10.     AESFinalRound(g[i],K,sbox)

(b) Alg#2: Optimized
1.  for i=0 to 1:
2.      offset=i+(i<<1)+(i<<3)
    //Split addRoundKey for better pipelining
3.      l_g[i]=l_g[i]⊕aes_key[offset+0]
    //Change loop structure of rounds comp.
4.      for r=1 to 9:
5.          AESRound(l_g[i])
6.          l_g[i]=l_g[i]⊕aes_key[offset+r]
7.      AESFinalRound(l_g[i])
8.      l_g[i]=l_g[i]⊕aes_key[offset+10]
```

(a) Alg#1: Conventional (b) Alg#2: Optimized

Fig. 8. AES expansion algorithm comparison

VM Acceleration via Vectorized Batch XOR Reduction. The LPN computation involves a vector-matrix multiplication (VM), e.g., $(\mathbf{u}, \mathbf{v}, \mathbf{w}) \cdot \mathbf{A}$ in Fig. 3, step #3, where $\mathbf{A} \in \mathbb{F}_2^{k \times n}$ is a sparse binary matrix with d non-zero entries per column. Each VM computes the XOR of d selected input indices, as detailed in [12, 14]. The resulting low spatial locality—e.g., limited reuse of nearby memory addresses—leads to poor cache utilization, as only small portions of cache lines are accessed [31]. This increases memory pressure in large-scale COT generation, where frequent global memory accesses exceed cache capacity. Prior works use AES with SIMD extensions to generate random indices, but XORs are computed sequentially, incurring two global memory $reads$ and one $write$ per operation. This read-modify-write pattern introduces loop-carried dependencies, preventing full pipelining. Linear Feedback Shift Registers (LFSRs) provide lightweight pseudorandom number generation but are also sequential, limiting parallelism in high-throughput designs.

To address memory constraints on resource-limited hardware, we introduce a pipeline- and parallelism-friendly approach based on fine-grained decomposition of the VM computation—decoupling index generation, memory access, and XOR reduction into independently optimized stages. Let $\mathbf{k}$ denote the input vector from the TEE-assisted Initial Correlation Setup, and $\mathbf{i} \in \mathbb{F}_k^d$ represent a set of randomly generated indices used to select entries from $\mathbf{k}$. The XOR of selected elements, $\mathbf{k}[i_1], \ldots, \mathbf{k}[i_d]$, is computed in local memory, thereby reducing on-chip usage by avoiding storage of unused portions of $\mathbf{k}$. We introduce two variants of a *Pipelined/Parallel XOR Reducer* (PXR) that replace sequential XOR operations. The pipelined reducer (Alg#1) avoids intermediate reads for each update, while the parallel reducer (Alg#2) eliminates iteration dependencies, maximizing performance when additional resources are available, as shown in Fig. 9.

```
Algorithm: VM computation
Input: k; Output: n; Parameters: l_k/n, k/n in local mem; init_s, initial seed.

(a) Alg#1: Constrained resource
1.  for i=0 to $size_n$ step $size_{batch}$:
2.      for l=0 to $size_{batch}$:
3.          init_s=k[i]+(i+1)(l+1)
4.          m_LFSR(s, init_s, 1)
            //XOR in local memory
5.          for j=0 to (d-1):
6.              m_LFSR(r, $s+2^j+i+j$, j)
7.              l_n[l]=PXR(l_n[l],k[r])
8.          n[i+l]=l_n[l]

(b) Alg#2: Moderate resource or higher
//Allocate local memory for vector k
1.  for i=0 to $size_n$ step $size_{batch}$:
2.      init_s=l_k[i]+(i+1)
3.      m_LFSR(s, init_s, i)
4.      for l=0 to $size_{batch}$:
5.          for j=0 to (d/2-1):
                //Generate locations for XOR
6.              m_LFSR($r_1$, $s+2^j+i$, j)
7.              m_LFSR($r_2$, $s+2^j+i+1$, j+d/2)
8.              lt[j]=l_k[$r_1$]⊕l_k[$r_2$]
9.          l_n[l]= PXR(lt[0:(d/2-1)])
10.         n[i+l]=l_n[l]
```

(a) Alg#1: Constrained resource (b) Alg#2: Moderate resource or higher

Fig. 9. Algorithms for VM computation with different resources

Unlike prior methods, our algorithm supports *batched memory transactions* by removing LFSR feedback dependencies. We achieve this via a multiplexer that injects an auxiliary bit into the XOR path, allowing independent iterations with unique seeds derived from the iteration index. The system uses a 512-bit data bus to fetch four 128-bit elements per transaction, aligning with DDR4/AXI burst sizes for improved bandwidth and latency. Consequently, our VM module supports dataflow parallelism through pipelining, local memory reuse, and batched partitioning. Structured decomposition across GGM and VM modules further reduces memory traffic and latency, setting a new benchmark for COT acceleration. Detailed performance results are provided in Section IV-B.

IV. Experiments

A. Experiment Setup

We evaluate our benchmarks under three network configurations: (1) a regular LAN environment with 3Gbps bandwidth and 0.3ms latency, (2) a constrained WAN setting with 200Mbps bandwidth and 50ms latency, and (3) a mobile-like setting with 100Mbps bandwidth and 80ms latency. For secure computation primitives, we synthesize and implement COT generation on a low-end Zynq-7000 SoC FPGA using Vitis High-Level Synthesis [32]. We define the client profile as a constrained configuration equivalent to typical IoT sensors (500MHz single-core CPU, 256MB mem), reflecting real-world platforms like the BeagleBone embedded system used in cryptographic benchmarking [33]. The server resource profile reflects an edge-server setup with equivalent resources as Intel NUC 12 Pro. Intel SGX [34] is used as the TEE testbed, Silentflow relies on a limited set of CSPRNG and PRNG operations, thereby functioning with ultra-lightweight resource requirements and avoiding computational overhead.

B. FPGA Acceleration of SilentFlow

Table II shows hardware utilization and timing results for the GGM and VM units, as well as overall COT generation, after place-and-route. The number of BRAMs reflects on-chip memory use, similar to CPU cache. We use Ferret as the baseline for comparison. Our BOX approach achieves a 6.20× to 23.13× speedup, depending on the subtree height configuration. For VM, our vectorized batch PXR method achieves 3.18× and 5.06× lower latency using Alg#1 and #2, respectively, with $batch\_size$=256—the best-performing setting in Fig. 7. Finally, the integrated end-to-end COT generation improves latency by 10.21× and 11.78×, with the chosen configuration balancing performance and resource trade-offs between VM and GGM, as discussed in Section III-C. However, with sufficient BRAM, we observe up to a 53.29× improvement in VM computation, highlighting BRAM's advantage in supporting highly parallel operations. This result suggests that with adequate resources—such as high-performance embedded processors or smartphones (e.g., Raspberry Pi 4, or iPhone X and later as the client)—significant speedups are achievable. Our design thus provides two options: one for resource-limited devices and another for more capable platforms.

C. COT generation

We compare SilentFlow with representative state-of-the-art OT protocols, as summarized in Table III. SilentFlow achieves substantial speedups ranging from 5.14×∼39.51× across different protocols. Notably, the speedup increases as network latency worsens, while SilentFlow remains unaffected due to its fully non-interactive design—even during the sparse correlation generation phase, enabled by our TEE-based approach. Consequently, the total COT generation time is dominated solely by local computation, which is further accelerated by our FPGA-based hardware architecture.

TABLE II
Hardware resource cost of VM, GGM, and COT computation, with $k = 32771, n = 2^{20}$, using different optimization methods on Zynq-7000 SoC Z-7045. *Alg#2 exceeds the FPGA storage limits and is viable when sufficient resources are available.

Unit	Method	BRAM	DSP	FF	LUT	Latency(ms)	Speedup
GGM	Ferret	48	0	14564	19206	281.40	-
	BOX(s=3)	67	0	31446	20193	109.37	6.20×
	BOX(s=4)	67	0	42346	28269	73.99	9.16×
	BOX(s=6)	67	0	82378	55921	43.82	15.47×
	BOX(s=12)	227	0	73189	60490	29.31	23.13×
VM	Ferret	4	0	8520	8265	281.40	-
	Alg#1	4	1	8748	3885	88.46	3.18×
	Alg#2	276	0	14272	13275	55.66	5.06×
	Alg#2*	2072	0	25920	23005	5.28	53.29×
COT	Ferret	84	0	22881	23986	985.79	-
	Alg#1(s=4)	71	1	50044	31870	96.54	10.21×
	Alg#2(s=6)	295	0	101376	62341	83.67	11.78×

TABLE III
Performance comparison with various state-of-the-art COT generation protocols. Following the standard in prior works, the generation of 10^7COTs is used as the benchmark, including both the sparse correlation constitution and the extension phases, while excluding the negligible one-time setup.

Protocol	10^7 **COTs generation (second)**					
	LAN	Speedup	WAN	Speedup	Mobile	Speedup
QuietOT [27]	36.24	29.46×	43.89	35.65×	48.48	39.51×
SilentOT [11]	17.32	14.08×	24.97	20.28×	29.56	24.09×
Ferret [12]	9.703	7.89×	17.02	13.83×	21.64	17.64×
SSOT [26]	6.33	5.14×	13.98	11.36×	18.57	15.13×
SilentFlow	1.230	-	1.231	-	1.227	-

D. End-to-end Framework

In Table IV, we compare SilentFlow against the Ferret and IKNP protocols integrated into state-of-the-art PPMLaaS frameworks. We evaluate two deep learning models: ResNet-50, representing a large-scale network, and SqueezeNet, representing a lightweight architecture. The results show that with SilentFlow, the Cheetah and CrypTFlow2 frameworks can complete inference under a mobile network environment in 60s for ResNet-50 and 152s for SqueezeNet, achieving a speedup of 4.75× to 4.78× over previous protocols. As network conditions improve, the relative speedup from communication reduction diminishes; however, SilentFlow still achieves at least a 3.17× acceleration on ResNet-50, primarily due to FPGA-based computation, as communication is typically not a bottleneck under LAN settings. These results demonstrate that although SilentFlow primarily targets COT generation, it delivers a 2.88×∼4.78× improvement in end-to-end inference performance—validating the effectiveness of our design.

V. Conclusion

In this work, we demonstrate that secure MPC-based deep learning inference is becoming practical in resource-constrained environments by addressing the primary bottleneck of COT generation. SilentFlow introduces a hardware–algorithm co-design that enables SqueezeNet inference in under 1 minute over mobile networks on hardware equivalent to IoT sensors or wearables, achieving a 4.78× speedup.

TABLE IV
PERFORMANCE COMPARISON WITH FERRET AND IKNP PROTOCOLS BY IMPLEMENTATIONS IN REAL-WORLD PPMLAAS FRAMEWORKS.

Framework	**ResNet50 (second)**			**SqueezeNet (second)**		
	LAN	WAN	Mobile	LAN	WAN	Mobile
SCI [6]	581	1058	1424	357	584	728
SCI+**SilentFlow**	183	266	308	124	139	152
Speedup	3.17×	3.97×	4.62×	2.88×	4.20×	4.78×
Cheetah [7]	500	795	1013	129	209	285
Cheetah+**SilentFlow**	145	216	256	38	52	60
Speedup	3.448×	3.68×	3.95×	3.39×	4.02×	4.75×

For more complex models like ResNet-50, SilentFlow completes inference in under 5 minutes with a 4.62× speedup.

REFERENCES

[1] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A low latency framework for secure neural network inference," in *27th USENIX security symposium (USENIX security 18)*, 2018, pp. 1651–1669.

[2] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "Xonn: oblivious deep neural network inference," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1501–1518.

[3] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: Scalable provably-secure deep learning," in *Proceedings of the 55th annual Design Automation Conference (DAC)*, 2018, pp. 1–6.

[4] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference system for neural networks," in *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*, 2020, pp. 27–30.

[5] Q. Zhang, C. Xin, and H. Wu, "Gala: Greedy computation for linear algebra in privacy-preserved neural networks," *Network and Distributed Systems Security (NDSS) Symposium*, 2021.

[6] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020, pp. 325–342.

[7] Z. Huang, W.-j. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two party deep neural network inference," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 809–826.

[8] Q. Pang, J. Zhu, H. Möllering, W. Zheng, and T. Schneider, "Bolt: Privacy-preserving, accurate and efficient inference for transformers," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 4753–4771.

[9] W.-j. Lu, Z. Huang, Z. Gu, J. Li, J. Liu, C. Hong, K. Ren, T. Wei, and W. Chen, "Bumblebee: Secure two-party inference framework for large transformers," *Network and Distributed Systems Security (NDSS) Symposium*, 2025.

[10] Q. Zhang, T. Xiang, C. Xin, and H. Wu, "From individual computation to allied optimization: Remodeling privacy-preserving neural inference with function input tuning," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 4810–4827.

[11] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl, "Efficient pseudorandom correlation generators: Silent ot extension and more," in *Annual International Cryptology Conference*. Springer, 2019, pp. 489–518.

[12] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated ot with small communication," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020, pp. 1607–1626.

[13] V. Kolesnikov and R. Kumaresan, "Improved ot extension for transferring short secrets," in *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*. Springer, 2013, pp. 54–70.

[14] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai, "Compressing vector ole," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018, pp. 896–912.

[15] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl, "Efficient two-round ot extension and silent non-interactive secure computation," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019, pp. 291–308.

[16] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, N. Resch, and P. Scholl, "Correlated pseudorandomness from expand-accumulate codes," in *Annual International Cryptology Conference*. Springer, 2022, pp. 603–633.

[17] ——, "Oblivious transfer with constant computational overhead," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2023, pp. 271–302.

[18] G. Couteau, P. Rindal, and S. Raghuraman, "Silver: silent vole and oblivious transfer from hardness of decoding structured ldpc codes," in *Annual International Cryptology Conference*. Springer, 2021, pp. 502–534.

[19] M. Zheng, D. Xu, L. Jiang, C. Gu, R. Tan, and P. Cheng, "Challenges of privacy-preserving machine learning in IoT," in *Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things (AIChallengeIoT)*. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–7.

[20] A. Aminifar, M. Shokri, and A. Aminifar, "Privacy-preserving edge federated learning for intelligent mobile-health systems," *Future Generation Computer Systems*, vol. 161, pp. 625–637, 2024.

[21] R. Bahmani, M. Barbosa, F. Brasser, B. Portela, A.-R. Sadeghi, G. Scerri, and B. Warinschi, "Secure multiparty computation from sgx," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 477–497.

[22] J. I. Choi, D. Tian, G. Hernandez, C. Patton, B. Mood, T. Shrimpton, K. R. Butler, and P. Traynor, "A hybrid approach to secure function evaluation using sgx," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, 2019, pp. 100–113.

[23] P. Wu, J. Ning, J. Shen, H. Wang, and E.-C. Chang, "Hybrid trust multi-party computation with trusted execution environment." in *Network and Distributed Systems Security (NDSS) Symposium*, 2022.

[24] X. Zhou, Z. Xu, C. Wang, and M. Gao, "Ppmlac: high performance chipset architecture for secure multi-party computation," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 87–101.

[25] Y. Zhou, Z. Wang, L. Wu *et al.*, "Efficient privacy-preserving image classification for resource-constrained edge devices," in *Proceedings of the 29th ACM Conference on Computer and Communications Security (CCS)*, 2022.

[26] L. Roy, "Softspokenot: Quieter ot extension from small-field silent vole in the minicrypt model," in *Annual international cryptology conference*. Springer, 2022, pp. 657–687.

[27] G. Couteau, L. Devadas, S. Devadas, A. Koch, and S. Servan-Schreiber, "Quietot: Lightweight oblivious transfer with a public-key setup," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2024. [Online]. Available: https://eprint.iacr.org/2024/1079

[28] W. Dong and C. Wang, "Poster: Towards lightweight tee-assisted mpc," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2023, pp. 3609–3611.

[29] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, p. 65–76, Apr. 2009.

[30] J. de Fine Licht, G. Kwasniewski, and T. Hoefler, "Flexible communication avoiding matrix multiplication on fpga with high-level synthesis," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, p. 244–254.

[31] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Processing data where it makes sense: Enabling in-memory computation," *Microprocessors and Microsystems*, vol. 67, pp. 28–41, 2019.

[32] AMD, *Vitis High-Level Synthesis User Guide (UG1399)*, 2024, https://docs.amd.com/r/en-US/ug1399-vitis-hls.

[33] M. Sabo, S. Wesner, C. Krauß, and H. Federrath, "ULCL: An ultra-lightweight cryptographic library for embedded systems," in *Proceedings of the 2nd Workshop on Cryptography and Security in Computing Systems (CS2)*. ACM, 2015, pp. 13–18. [Online]. Available: https://doi.org/10.1145/2694805.2694809

[34] V. Costan and S. Devadas, "Intel sgx explained," *Cryptology ePrint Archive*, 2016.