Comparison of FHE and GC approaches in PPML

Presenter: **Kalyan Cheerla**

Committee:

Dr. Lotfi Ben Othmane

Dr. Kirill Morozov

Dr. Song Fu

Outline

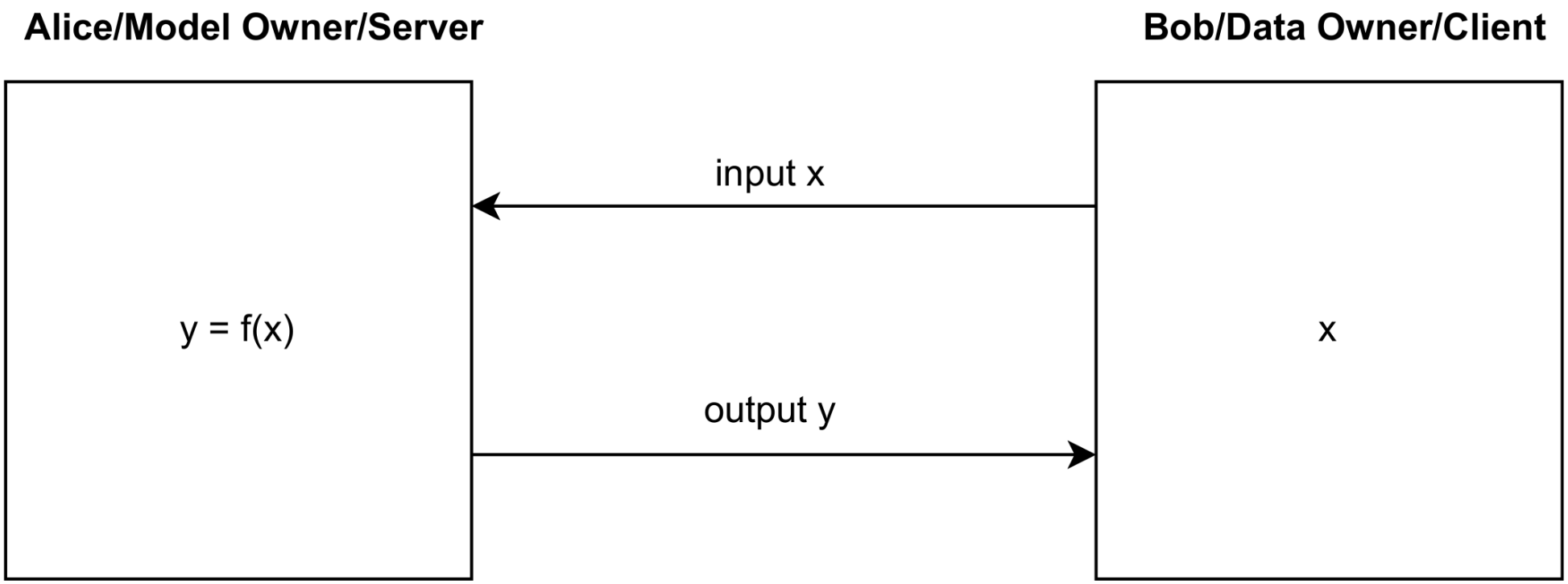
- Introduction
- Background and Preliminaries
- Current Literature
- Design and Implementation
- Comparative Analysis
- Conclusion and Future Work

What is PPML?

- PPML stands for Privacy-preserving Machine Learning
- ML is increasingly deployed in sensitive domains like healthcare, finance, law, etc.,
- PPML seeks to address these privacy and security issues.
- FHE and GC are promising cryptographic solutions

- Another term, SNNI stands for Secure Neural Network Inference.

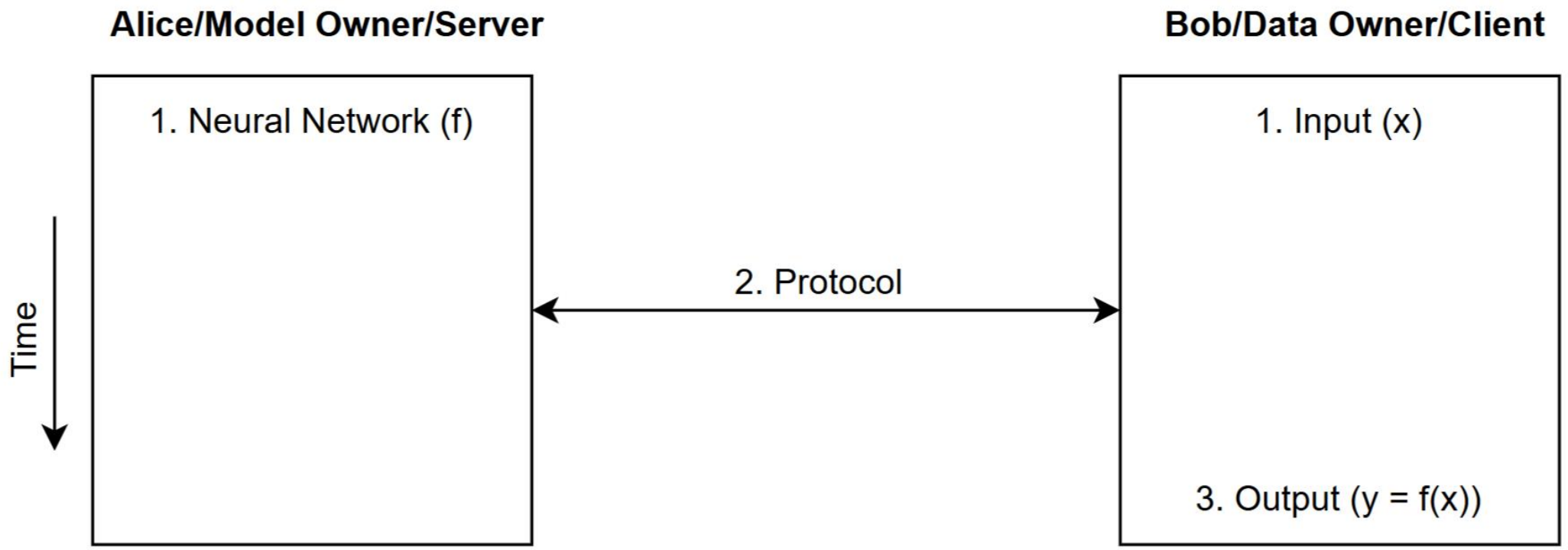
Typical ML inference…



Terminology

- Two parties:
  - Client (Bob/Data Owner) holds private input ***x***.
  - Server (Alice/Model Owner) holds private model ***f(x)***.
- Goal: Securely compute ***y=f(x)*** without leaking input or model.
- Address conflicting privacy goals.

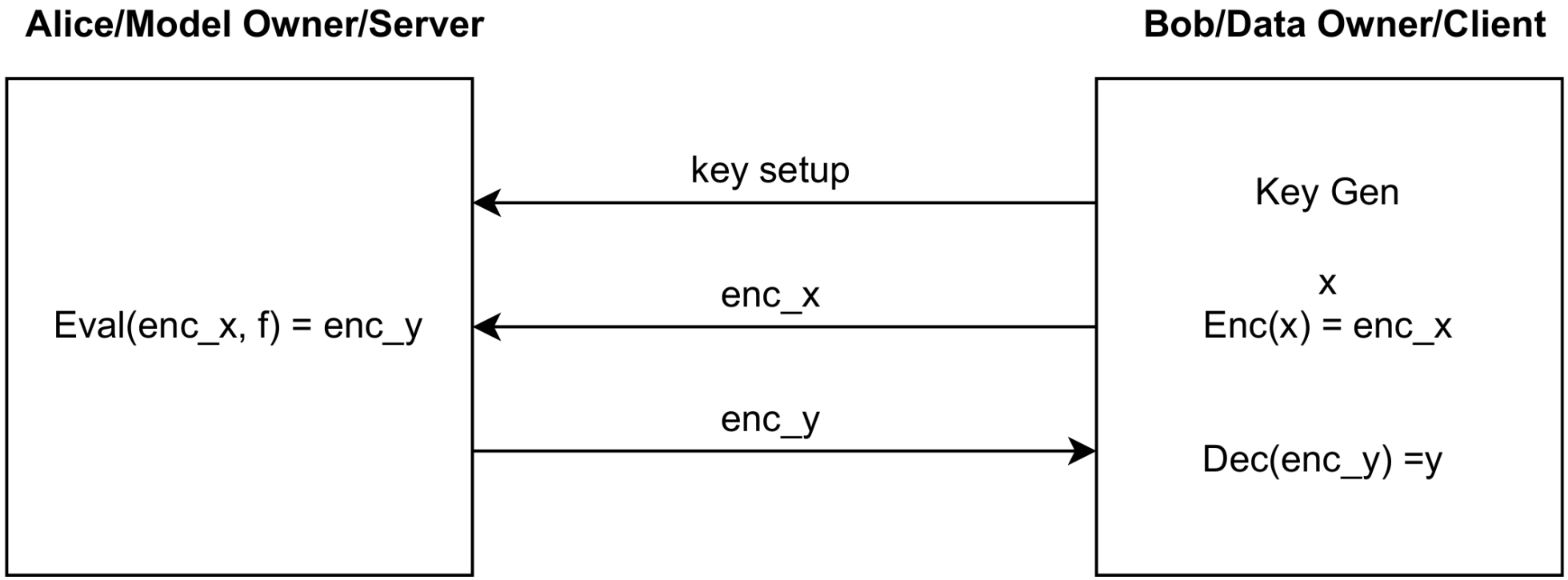
Typical PPML/SNNI Inference



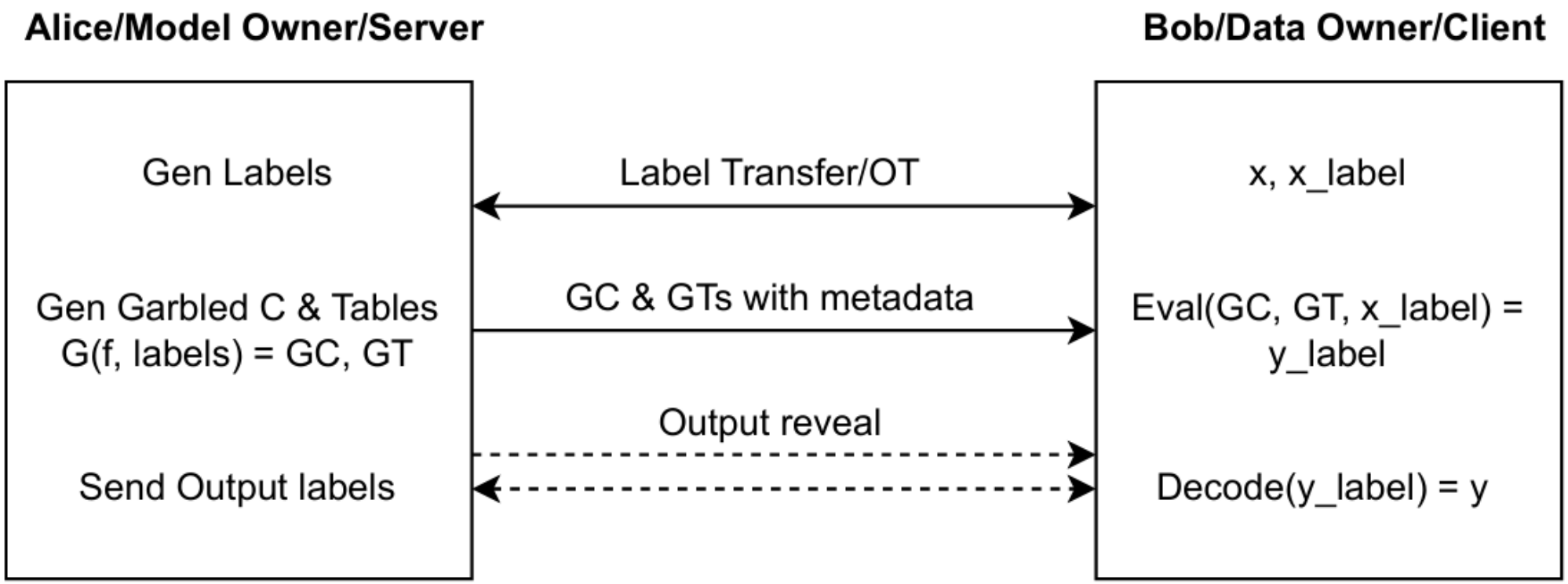
Our problem setup

- Compare and Contrast between these two Secure Computation Techniques for PPML.
- Understand how these techniques fare in real-world applications.
- Implement these two setups from the ground up to test their performances.

How a FHE based secure inference looks like?



GC-based Inference



Comparative Metrics

- Efficiency: Runtime, memory, communication.
- Accuracy: Approximation effects.
- Privacy: Input and model confidentiality.
- Scalability: Scaling to bigger networks.

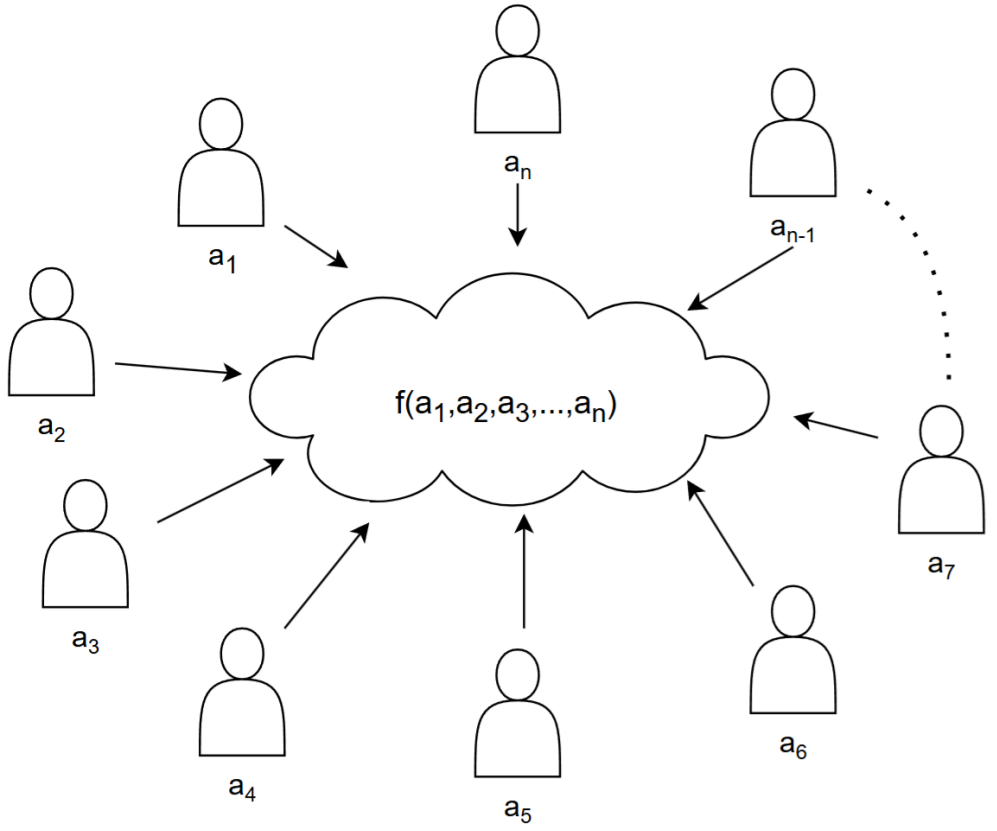
- Assumptions:
  - No trusted hardware.
  - Semi-honest adversarial model.

Threat Model

- Parties:
  - Client (Bob): Holds private input x.
  - Server (Alice): Holds private model parameters $f(x)$.
- Security Goals:
  - Client learns only the output $y=f(x)$, not the model internals.
  - Server learns nothing about the Client's input.
- Assumptions:
  - Semi-honest behavior: **Both parties follow protocol but try to infer extra info.**
  - No side-channel attacks considered.
  - Secure communication channels (e.g., TLS).

Background – Secure Computation

- Secure Computation enables public function evaluation over private inputs.
- Garbled Circuits (GC): Using logic gates for Encryption and Evaluation.
- Fully Homomorphic Encryption (FHE): Direct computation on ciphertexts.
- Different trade-offs in interaction, overhead, expressiveness.



Background – FHE

- FHE allows operations directly on encrypted data.
- CKKS scheme: Approximate real-number computation.
- Homomorphic addition, multiplication, rotation supported.
- Requires careful scale and modulus management.

Background - GC

- GC encrypts circuit gates; only correct output is revealed.
- Evaluator learns no intermediate information.
- TinyGarble2.0 used for efficient sequential circuit execution.
- Free-XOR and Half-Gates optimizations improve efficiency.

Related Work Overview

- Secure inference using Secure Function Evaluation (SFE) and Private Function Evaluation (PFE).
- Garbled Circuits (GC) and Fully Homomorphic Encryption (FHE) widely studied.
- Recent works: SecureML [1], CHET [2], Sigma [3], Iron [4], etc.
- Gap Identified:
  - Limited real-world side-by-side evaluations of FHE vs GC.
- Our Contribution:
  - Practical implementation and empirical benchmarking under consistent settings.

Design Goals

Privacy Preservation: Hide inputs and model parameters.

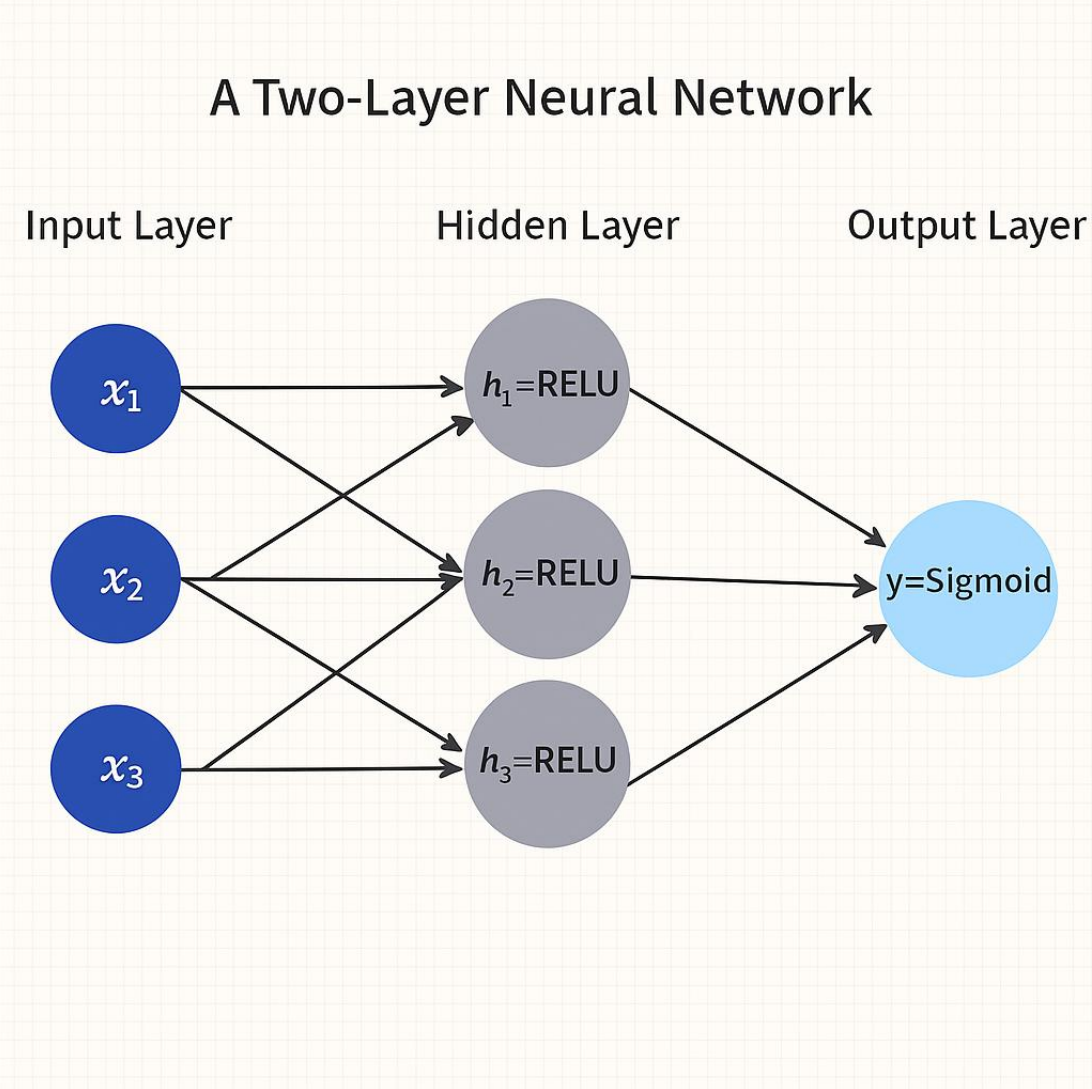
Inference Deviation: Measure output deviation w.r.t plaintext baseline. (Functional Deviation)

Reproducibility: Use open-source tools and detailed benchmarking.

NN Inference Function

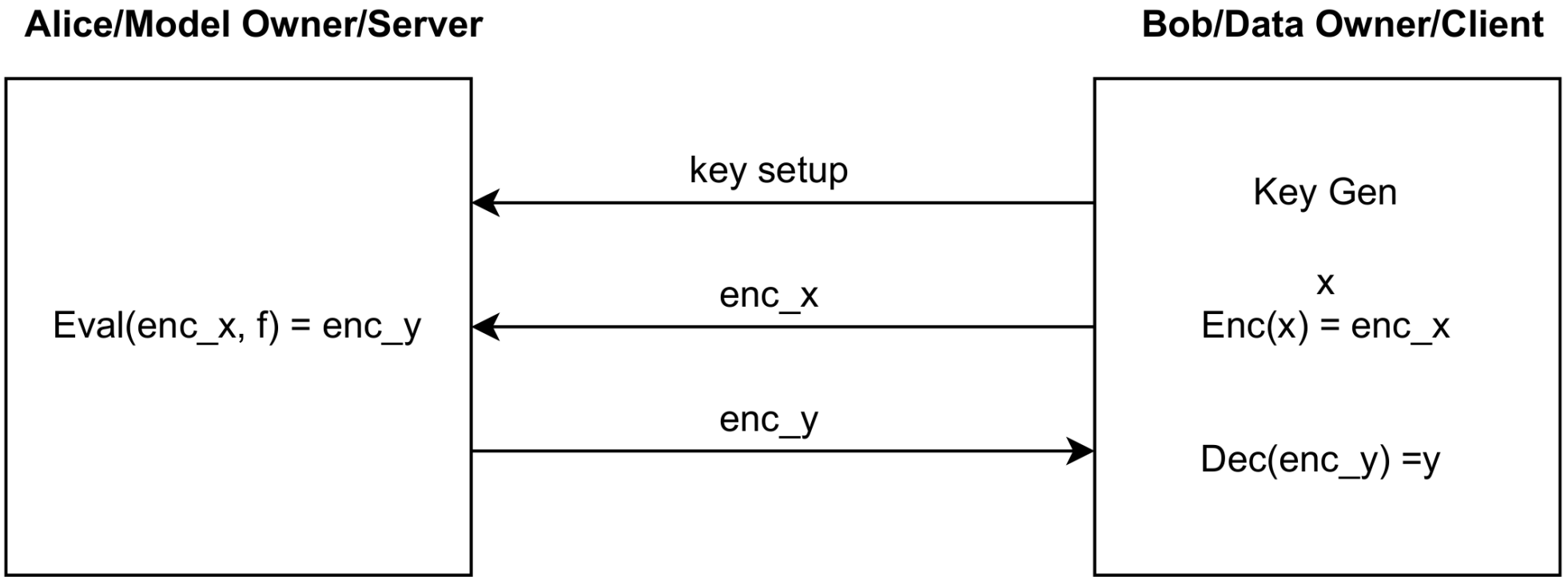
- $\mathbf{y = \sigma(W_2\ ReLU(W_1.x + b_1) + b_2)}$
- **3 input → 4 hidden → 1 output**
- ReLU = max(0, x)
- σ = sigmoid = $\frac{1}{(1+e^{-x})}$

- ReLU ≈ x^2
- σ ≈ $0.5+0.197x -0.004x^2$



FHE-based Implementation

- Microsoft SEAL library (CKKS scheme).
- Operations over encrypted real numbers.
- ReLU approximated as square, sigmoid as 2nd-degree polynomial.
- Non-interactive inference: Client sends encrypted input, receives encrypted output.



Algorithm 1 FHE-based Secure Inference Protocol

Client (Bob):

Generate CKKS keys (public, secret, relin, galois)

Encode and encrypt input vector x

Send encrypted input and evaluation keys to Server

Server (Alice):

for each row w_i in W_1 **do**

    Multiply enc_x with w_i

    Rotate and sum to simulate dot product

    Add bias b_i and perform square for ReLU approximation

end for

Multiply ReLU outputs with W_2, add b_2

Apply sigmoid approximation: $0.5 + 0.197z - 0.004z^2$

Send encrypted output to Client

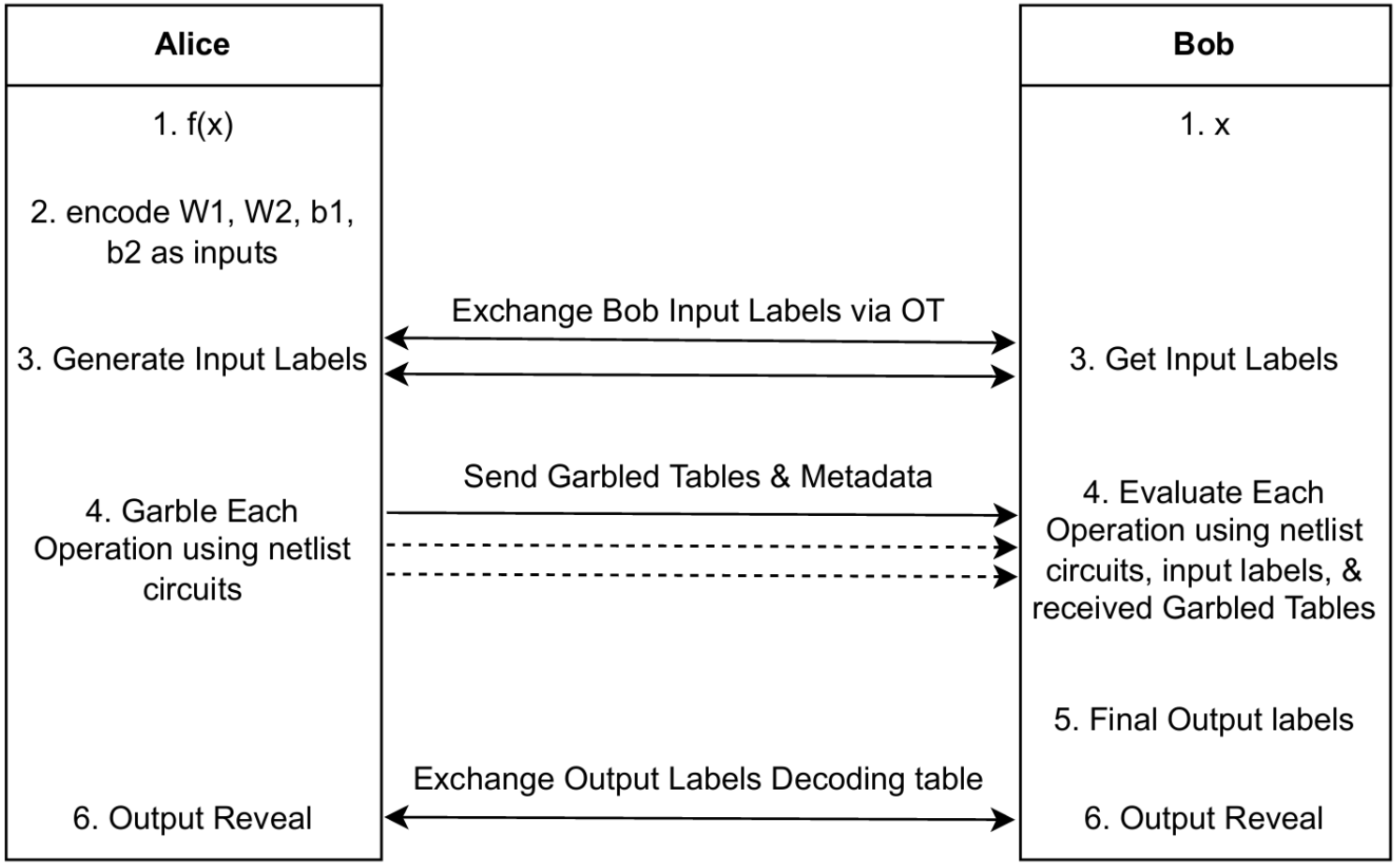
Client (Bob):

Decrypt and decode final result y

Pseudo Code for FHE-based Inference

GC-based Implementation

- TinyGarble2.0 framework (sequential GC execution).
- Fixed-point representation with scaling factor.
- Exact ReLU, polynomial sigmoid approximation.
- Requires interactive, multi-round evaluation with oblivious transfer (OT).



Algorithm 2 GC-based Secure Inference Protocol using TinyGarble2.0

Server (Alice):

Define model function $f(x)$ with weights W_1, W_2 and biases b_1, b_2

Initialize W_1, b_1, W_2, b_2 as fixed-point integers

Encode model values as ALICE inputs using `TG_int_init()`

Generate input labels for all inputs (client + server)

for each operation (add, mult, `divscale`[§], matmul, ReLU, etc.) **do**

    Load the corresponding precompiled circuit netlist

    Garble the circuit and generate garbled tables

    Send garbled tables and metadata to the client

end for

Client (Bob):

Encode input vector x as fixed-point integers

Encode x as BOB inputs using `TG_int_init()`

Retrieve input labels via Oblivious Transfer (OT)

for each operation in the network (add, mult, `divscale`[§], matmul, ReLU, etc.) **do**

    Receive garbled tables and metadata

    Evaluate corresponding garbled circuit using `sequential_2pc_exec_sh()`

    Pass evaluated intermediate output labels to the next operation

end for

Reveal final output using `reveal()` and convert to floating point

[§]The `divscale` operation refers to a custom helper that performs division followed by bitwidth reduction. This step is essential in fixed-point arithmetic to reverse scale inflation after multiplications and maintain compatibility with existing precompiled circuit bitwidths.

Pseudo Code for GC-based Inference

Experimental Setup

Hardware: 8 vCPU, 32 GB RAM, Ubuntu 24.04 VM.

Neural Network:

$$\mathbf{y = Sigmoid(W_2\ ReLU(W_1.x + b_1) + b_2)}$$

3 input → 4 hidden → 1 output

Fixed weights and biases.

Same NN input across all modes for fair comparison.

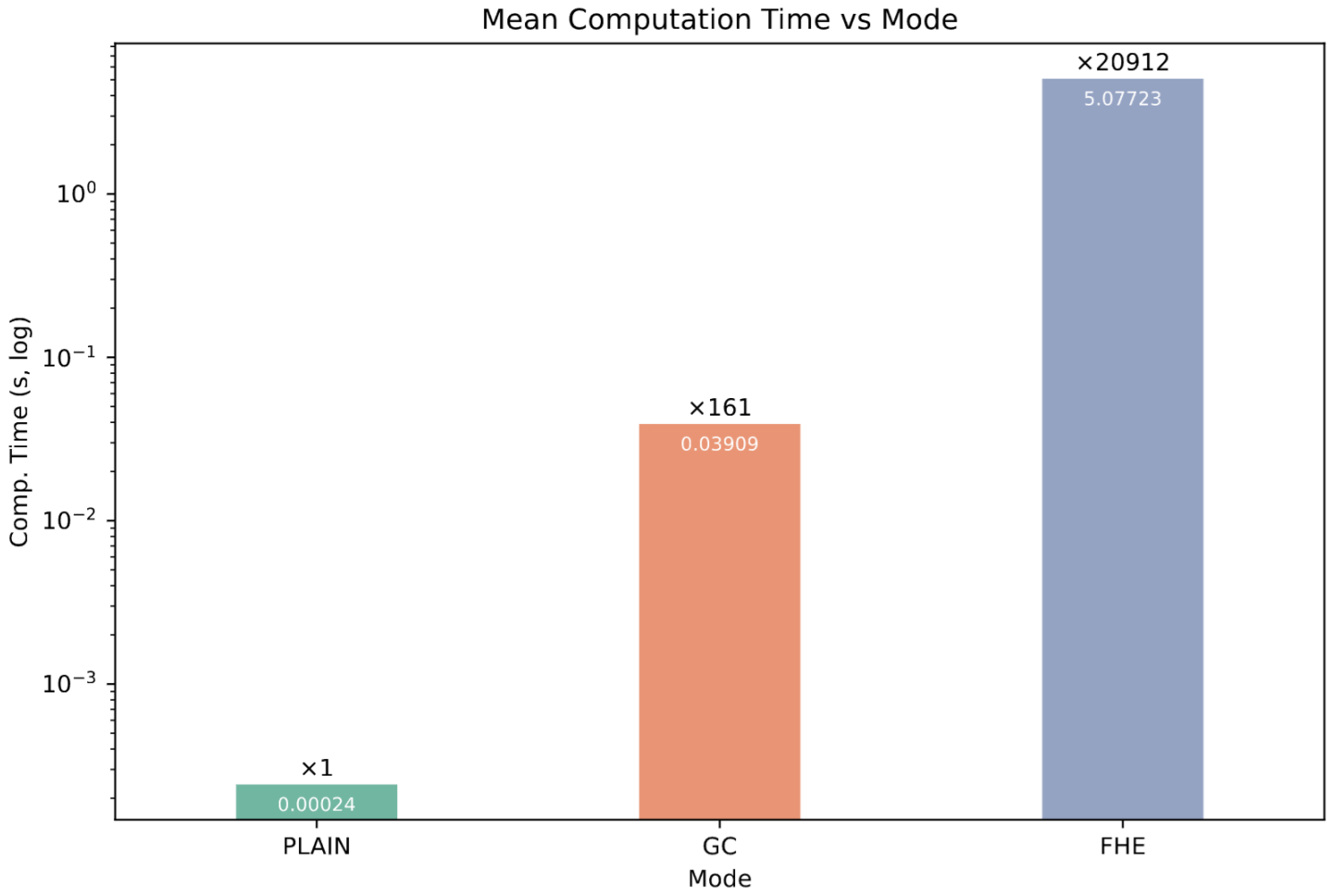
Protocol Parameters

- **CKKS Scheme**
- Polynomial Modulus Degree = 16384 = 2^{14}
- Coeff. Modulus Chain: [60,40,40,40,30,30]
- Initial Scale: 2^{30}

- **TinyGarble2.0**
- Fixed-point scaling factor = 1000 (3-digit precision)
- Bit-widths
  - Inputs, Outputs = 20 bits
  - Intermediate = 64 bits

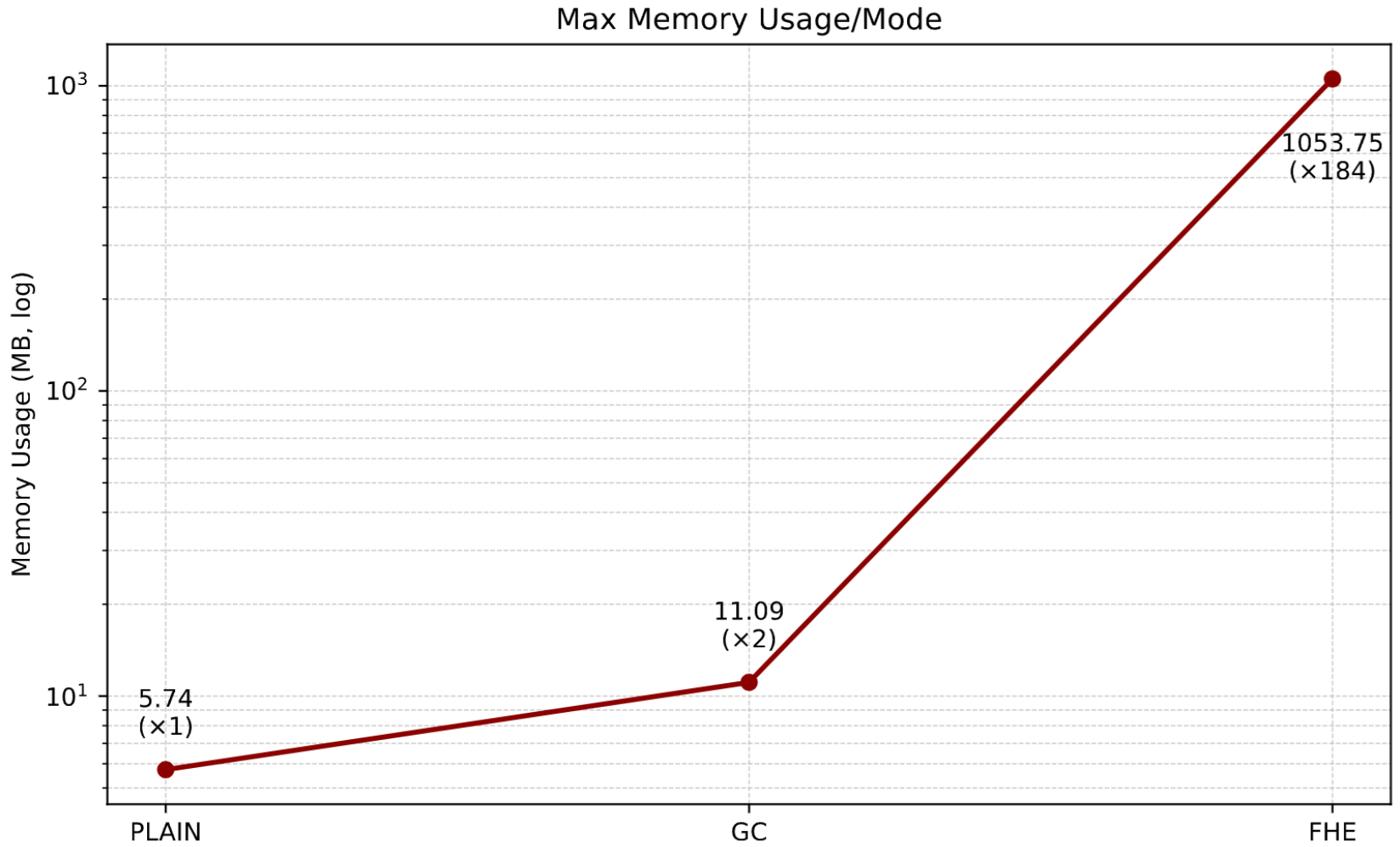
Computation Time Analysis

- Plaintext: 0.24 ms (baseline)
- GC: ~39 ms (×161 slowdown)
- FHE: ~5.1 sec (×20,912 slowdown)



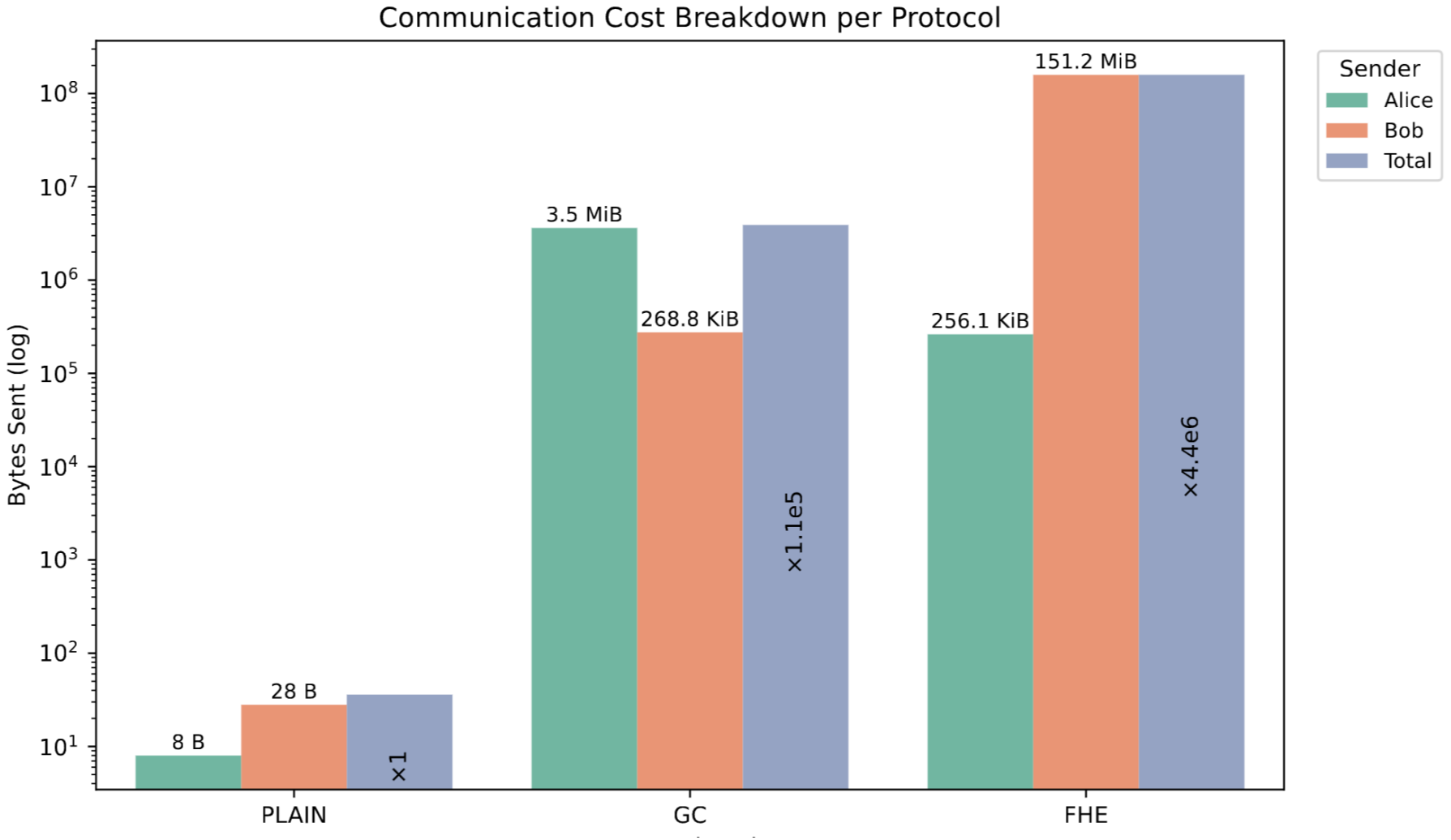
Memory Usage Results

- Plaintext: ~5.7 MB
- GC: ~11.1 MB (lightweight)
- FHE: ~1053 MB (very heavy)



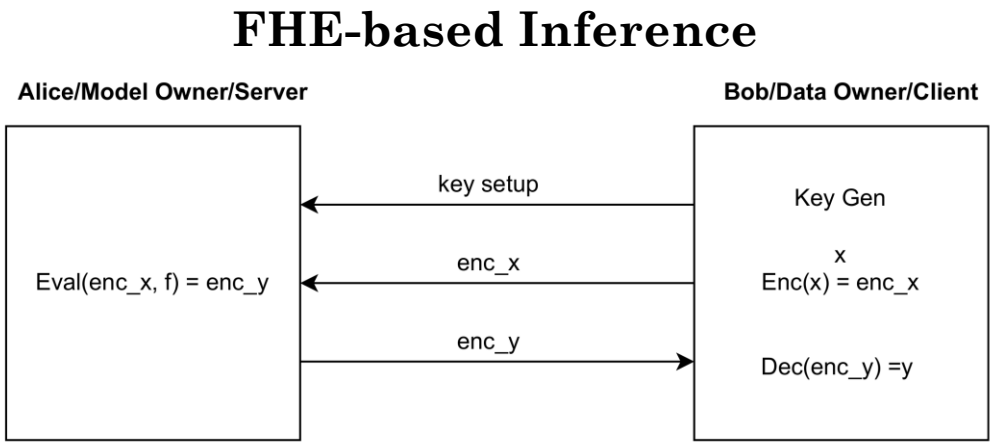
Communication Overhead

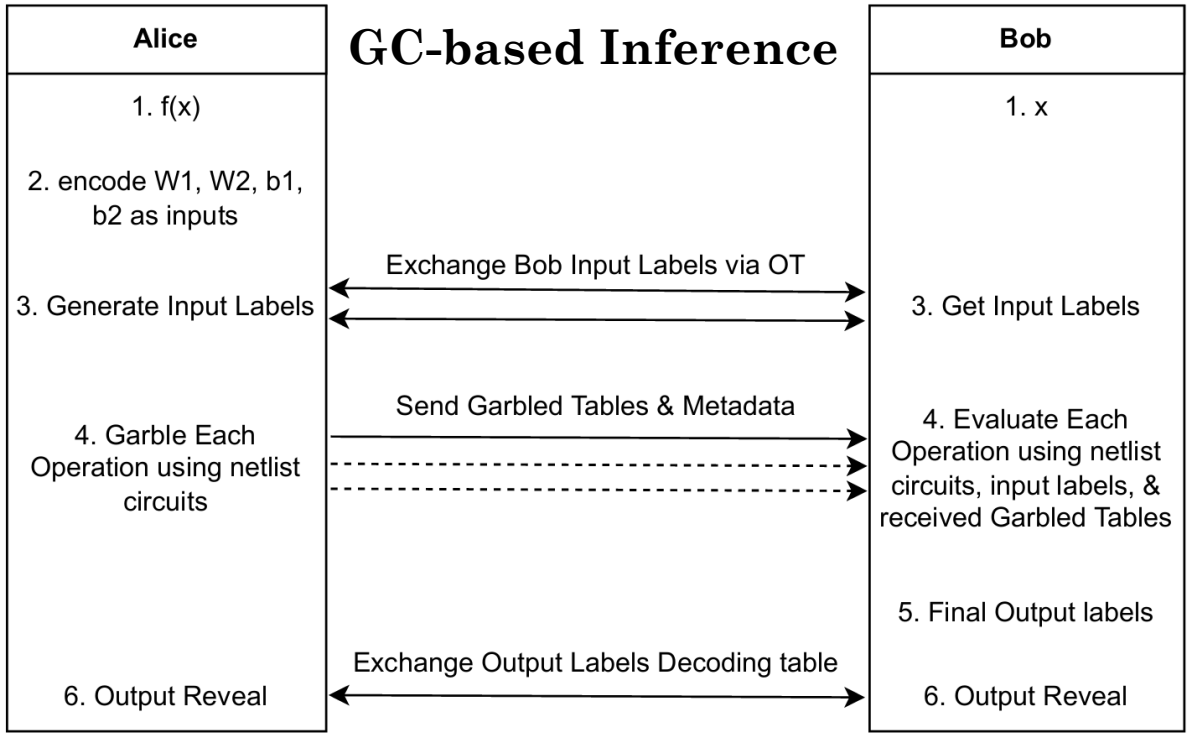
- GC: ~268 KB – 3.5 MB data exchanged
- FHE: ~151.5 MiB data exchanged



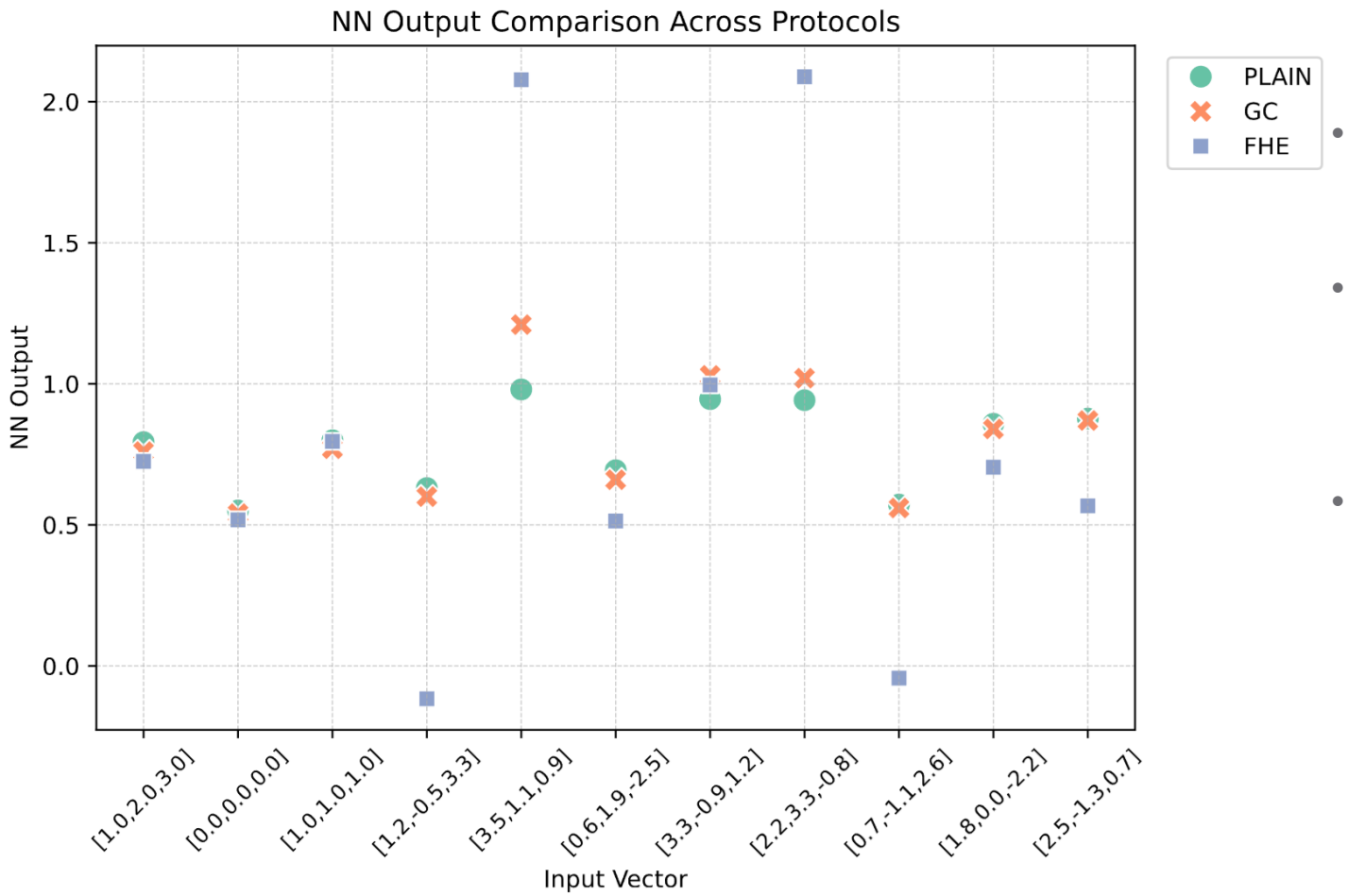
Communication Rounds

- (3 inputs x 2 OTs + 1) = 7 rounds for GC.
- 1 round for FHE.





Inference Output Deviation



- Outputs compared to plaintext reference.
- 23.46% and 121.69% - max deviation observed for GC & FHE.
- Deviations due to:
  - GC: rounding errors.
  - FHE: rescaling and approximation artifacts.

Privacy & Scalability

- FHE – provided FULL model confidentiality and input privacy.
- GC – preserved input and parameter privacy but **not model structure privacy**.

- FHE – Depth Constraints – Limited by polynomial modulus degree and modulus chain configuration. (16384 – 438 bits – 9 Max Modulus Count)
- GC – Circuit size grows linearly with model depth and bitwidth.
  - Each additional layer adds extra matmuls, activations, and associated scaling.
  - Modularity of TinyGarble2 helps manage memory overhead.
  - Communication costs vary based on added circuit complexity.

Summary of Observations

Metric	Garbled Circuits (GC)	Fully Homomorphic Encryption (FHE)
Round-trip Time	~ 0.04 seconds	~ 5.0 seconds
Peak Memory Usage	~ 11.09 MB	~ 1053.75 MB
Total Data Sent	~ 3.94 MB	~ 158.81 MB
Communication Rounds	7 (interactive)	1 (non-interactive)
Maximum Output Deviation	~23.46%	~121.69%
Model Structure Privacy	Leaked (unless using UCs)	Preserved
Input Privacy	Preserved	Preserved
Scalability Constraints	Scales in circuit depth, but linearly increases communication	Limited by modulus depth (multiplicative level budget)

Contd.

Feature	**GC**	**FHE**
Numerical Precision	Fixed-point (scaled floating-point numbers)	Approximate floating-point (CKKS)
Activation Function Handling	Native ReLU, polynomial sigmoid approximation	Square-approximated ReLU, polynomial sigmoid approximation
Offline Client Support	No (continuous interaction required)	Yes (after initial key setup)
Deployment Complexity	Requires precompiled netlists and interactive setup (OT, label management)	Requires parameter tuning (scale, moduli), high memory footprint
Strengths	Fast, lightweight, low memory usage	Non-interactive inference, expressive approximate arithmetic
Limitations	Interactive; sensitive to network latency	High computation cost; large memory and bandwidth cost

Conclusion

- Garbled Circuits:
  - Best for low-latency, online interactive scenarios.
- Fully Homomorphic Encryption:
  - Best for privacy-focused, offline inference settings.
- Both faithfully preserve neural network functionality.

Future Work

- Scale to deeper networks (e.g., DNNs, Transformers).
- Explore malicious adversarial models.
- Hybrid protocols combining GC and FHE, also exploring SS (mainly FSS) techniques.
- Real-world deployment optimizations (e.g., mobile devices).
- Achieving **Semi-Honest Secure LLM Deployment**.

References

1. Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In 2017 IEEE symposium on security and privacy (SP),pages 19–38. IEEE, 2017.

2. Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. Chet: compiler and runtime for homomorphic evaluation of tensor programs. arXiv preprint arXiv:1810.00845, 2018.

3. Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. Sigma: Secure gpt inference with function secret sharing. Cryptology ePrint Archive, 2023.

4. Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. Advances in neural information processing systems, 35:15718–15731, 2022.

5. Other..

Thank You!

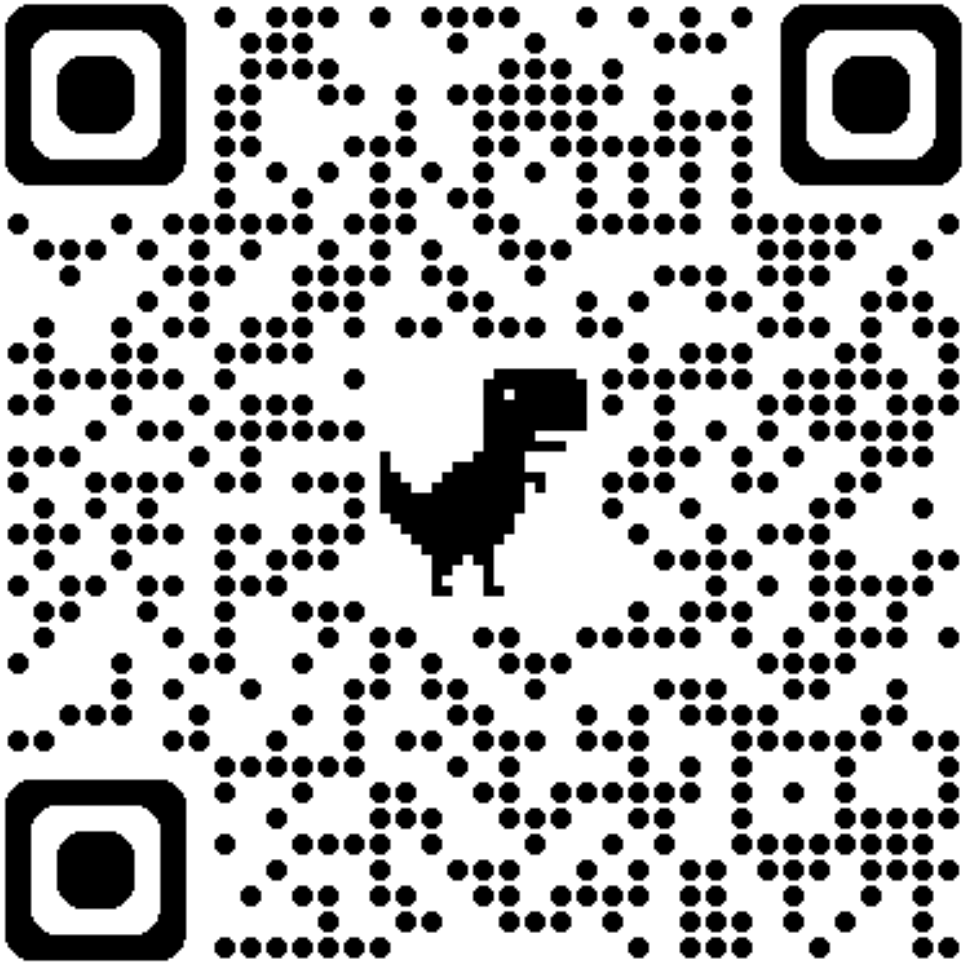
Questions?

Detailed Contributions

- Benchmarked implementations using a common communication structure extended on the NetIO protocol from the EMP-toolkit, allowing unified performance metrics for round-trip time, peak memory usage, communication overhead, and communication rounds.
- Addressed a gap in the PPML literature through a system-level evaluation of FHE-based (SEAL-CKKS) and GC-based (TinyGarble2.0) implementations using the same neural network architecture, input vectors, and threat model.
- Analyzed the practical trade-offs of FHE and GC protocols in terms of interactivity, approximation effects on model outputs, and potential for scaling to deeper neural networks.

GitHub Implementation

https://github.com/kalyancheerla/snni-fhe-gc



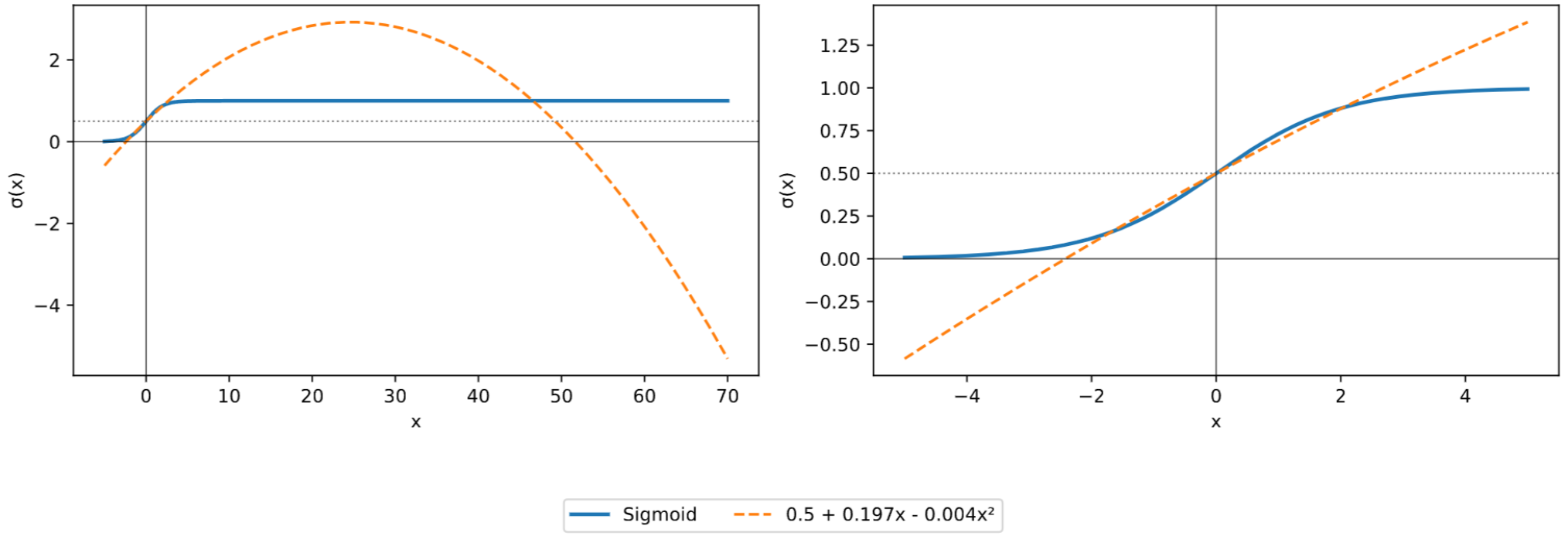
Find me at…

- https://kalyanch.com
- https://github.com/kalyancheerla/
- https://www.linkedin.com/in/kalyancheerla/

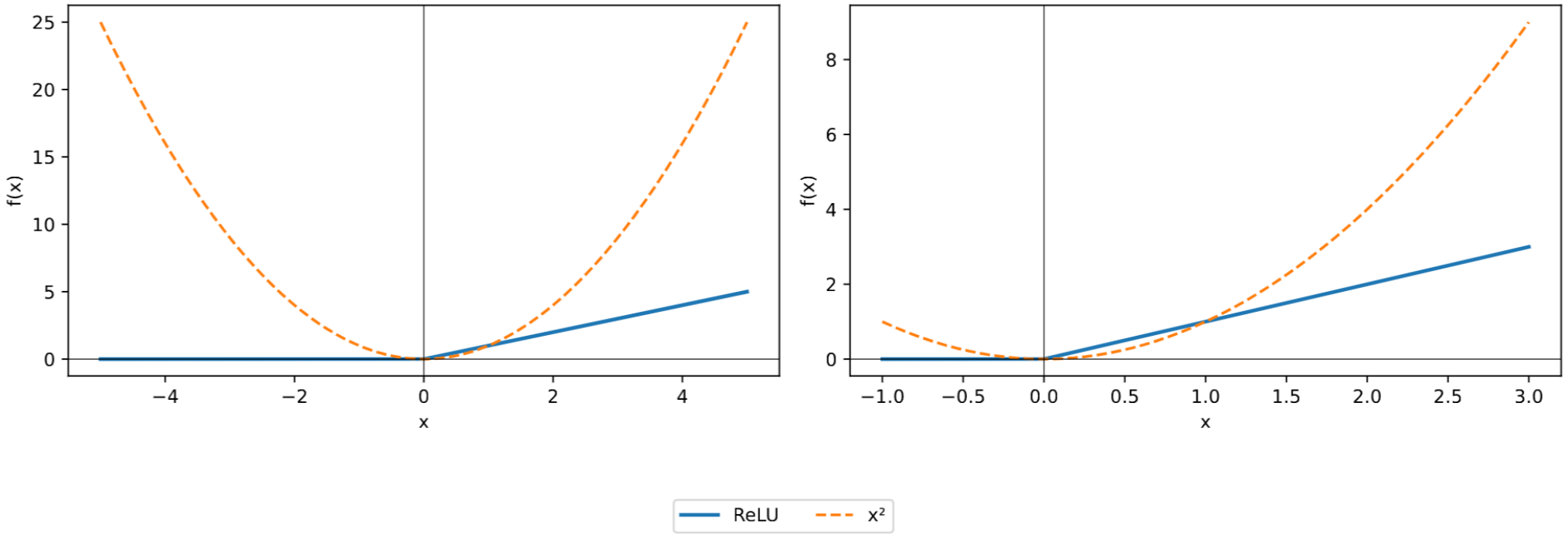
Oblivious Transfer

- 1-out-of-2 OT or 1-out-of-n OT box diagram

Approximations - Sigmoid



Approximations - ReLU



Polynomial Modulus Degree vs. Modulus Chain Limits (128-bit security)

Polynomial Modulus Degree	Total Bit-Length of Coefficient Modulus Chain	Max Modulus Count
1024	27 bits (not usable for CKKS)	1
2048	54 bits	1
4096	109 bits	3
8192	218 bits	5
16384	438 bits	9
32768	881 bits	16