

OpenFHE

Fully Homomorphic Encryption for Privacy-Preserving Machine
Learning using the OpenFHE Library
February 20, 2024

Ian Quah, Yuriy Polyakov, Sukanya Mandal
[iquah,ypolyakov,smandal}@openfhe.org](mailto:{iquah,ypolyakov,smandal}@openfhe.org)

Agenda

- Installation Options
- Lab Discussion: Comparing and Contrasting PPML Methods
- Spinning Up in Homomorphic Encryption for ML
- Q&A on FHE for ML/Quick Break
- OpenFHE Library
- Quick Break
- Hands-on: Using an In-the-Clear Model in the OpenFHE Library
- Hands-on: Training an Encrypted Logistic Regression Model in the OpenFHE Framework
- ML Applications and FHE Challenges



Installation Options

Installation options

- Build from source
 - See instructions at <https://github.com/openfheorg/openfhe-python>
- Docker
 - Build the image from scratch, OR,
 - Pull the image from Dockerhub at <https://hub.docker.com/r/openfheorg/openfhe-docker>
 - OpenFHE repositories (openfhe-development and openfhe-python) are cloned to /
 - Repositories with Python examples can be cloned to /workspace



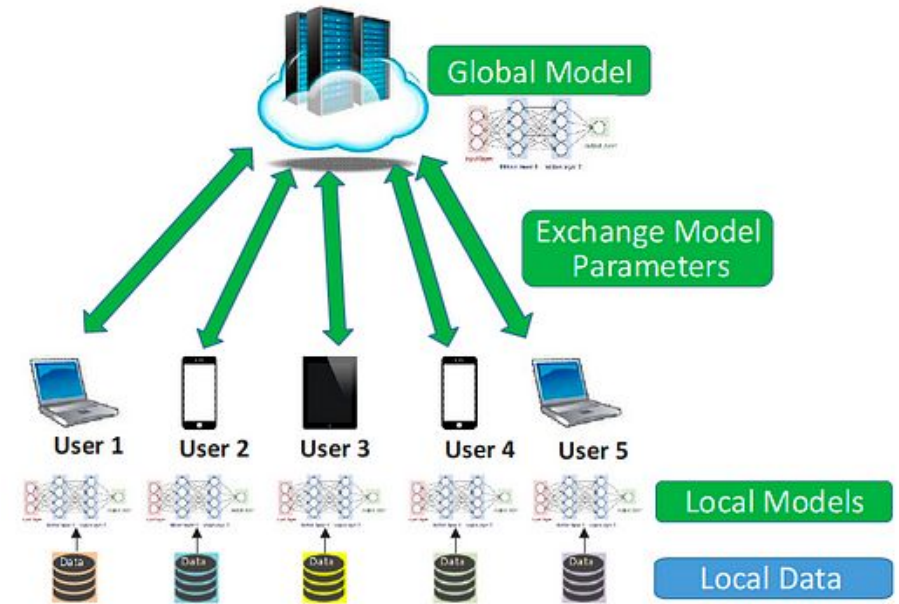
Comparing and Contrasting PPML Methods

DISTRIBUTED LEARNING

Definition: A system of interconnected learning models that train on distributed datasets.

Key Features: Scalability, Privacy, Decentralization.

Applications: Large-scale machine learning, geographically dispersed data, collaborative research.



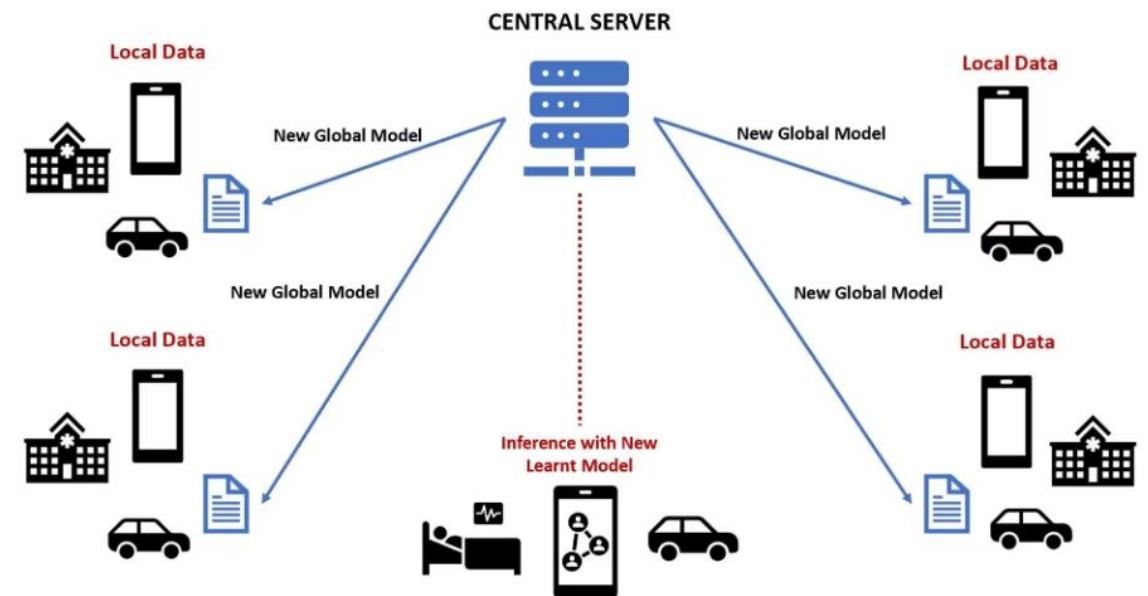
Source:
<https://digestize.medium.com/centralized-learning-vs-distributed-learning-c75ee9e94423>

FEDERATED LEARNING

Definition: A machine learning approach where a model is trained across multiple decentralized devices or servers holding local data samples, without exchanging them.

Key Features: Privacy preservation, Reduced communication costs, Collaborative learning without sharing raw data.

Applications: Smartphone keyboard prediction, Healthcare data analysis, Collaborative fraud detection.



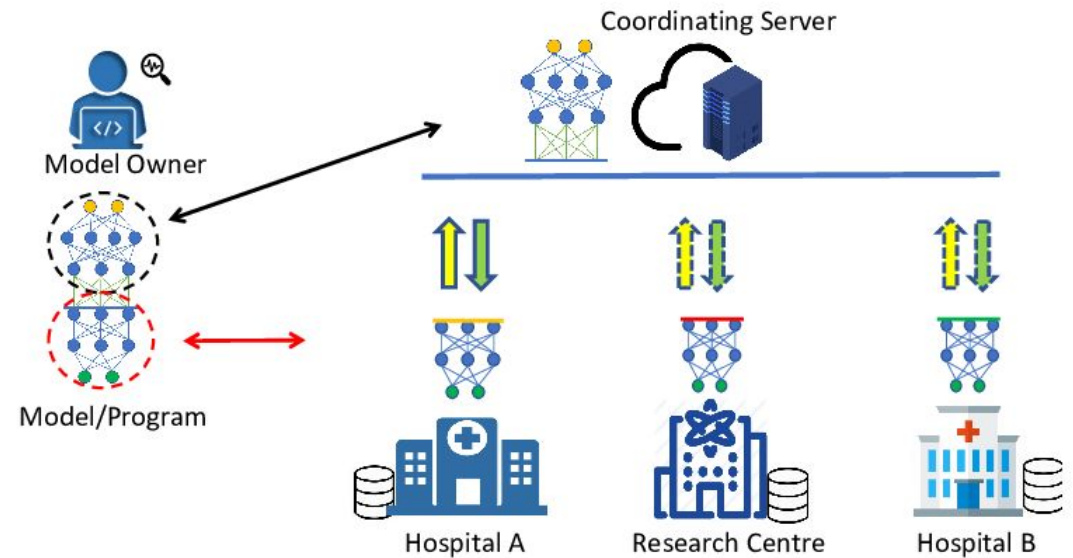
Source: <https://www.netapp.com/blog/future-of-AI-federated-learning/>

SPLIT LEARNING

Definition: A collaborative machine learning technique where the model training is split between the client device and a central server.

Key Features: Reduced data transfer, Privacy protection, Efficient use of bandwidth.

Applications: Edge computing, IoT devices, Privacy-sensitive industries.



Source:

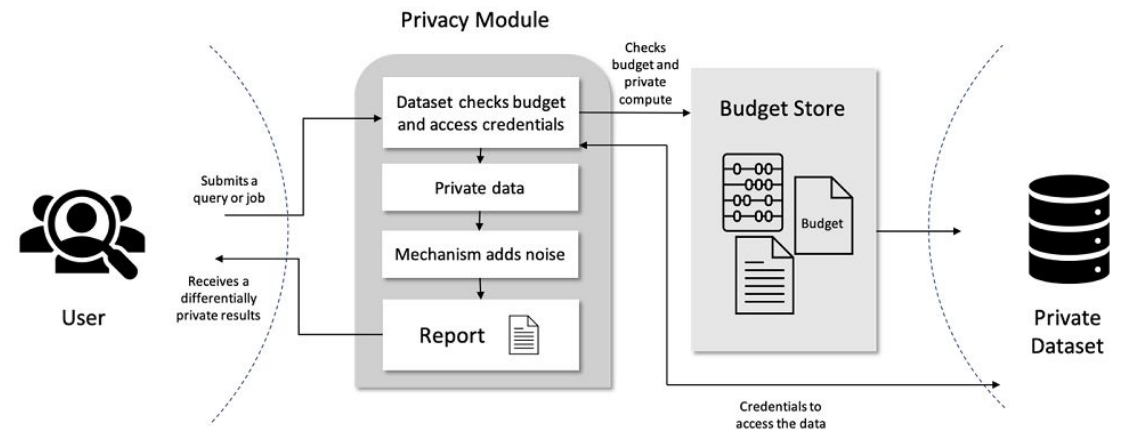
https://www.researchgate.net/figure/An-application-of-split-learning-in-a-cross-siloed-health-environment_fig5_344754274

DIFFERENTIAL PRIVACY

Definition: A technique that adds 'noise' to data to prevent the disclosure of individual information while still allowing for accurate aggregate analysis.

Key Features: Privacy guarantee, Robust to post-processing, Quantifiable privacy loss.

Applications: Census data, User behavior analytics, Public data releases.



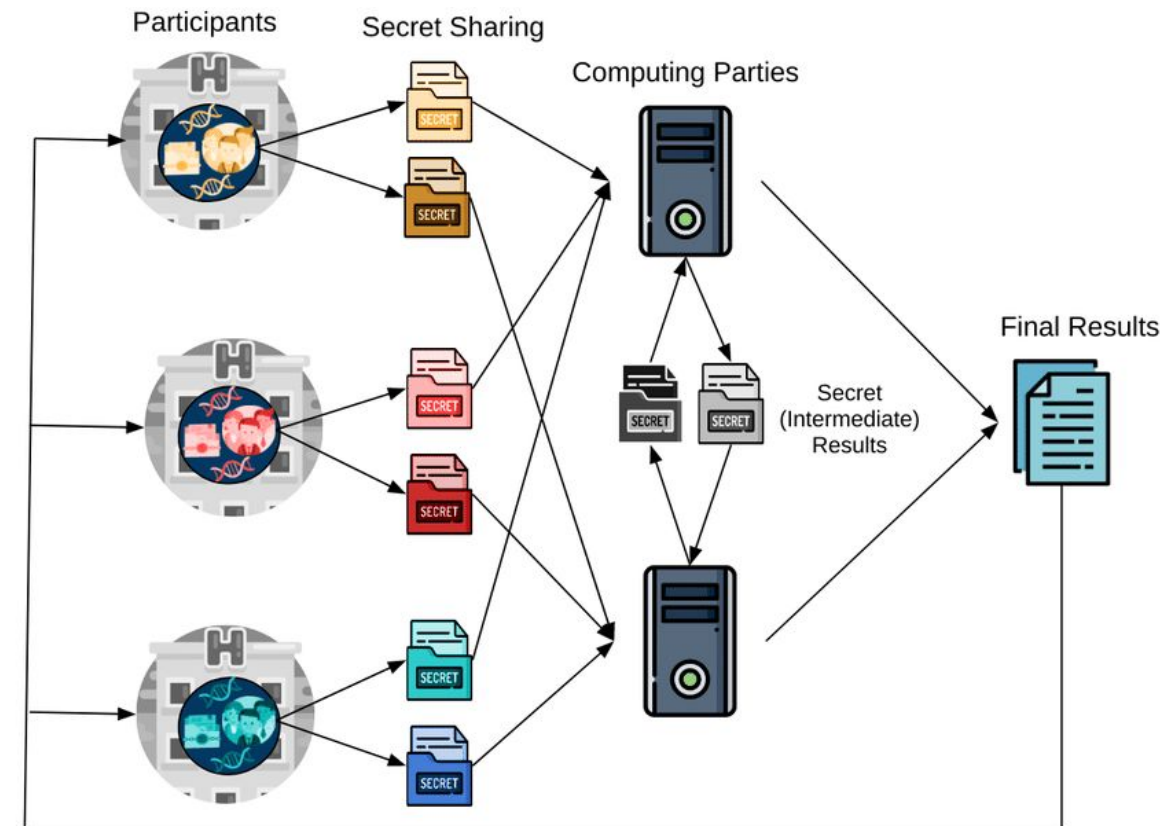
Source:
<https://cloudblogs.microsoft.com/opensource/2020/05/19/new-differential-privacy-platform-microsoft-harvard-opensdp/>

SECURE MULTI-PARTY COMPUTATION

Definition: A cryptographic protocol that enables parties to jointly compute a function over their inputs while keeping those inputs private.

Key Features: Data confidentiality, Computation integrity, Collaborative computation without mutual trust.

Applications: Joint financial analysis, Privacy-preserving scientific research, Secure voting systems.



Source:

https://www.researchgate.net/figure/Secure-multi-party-computation-Each-participant-shares-a-separate-different-secret-with_fig2_343179246

SECURE ENCLAVES/TRUSTED EXECUTION ENVIRONMENT


Definition: Secure Enclaves, also known as Trusted Execution Environments (TEE), are protected areas within a processor or a larger computing system. They ensure that sensitive code and data are stored, processed, and protected in a secure environment.

Key Features:

- **Isolated Execution:** Code within the TEE runs in an isolated environment to prevent unauthorized access or tampering.
- **Data Encryption:** Data is encrypted in the TEE, ensuring confidentiality and integrity even if the system is compromised.
- **Remote Attestation:** Enables a third party to verify the enclave's integrity and the authenticity of the software running inside.
- **Access Control:** Only authorized code can run inside the TEE, preventing malicious operations.

Applications:

- **Secure Mobile Payment Processing:** Protects payment information from being exposed to the rest of the system.
- **Confidential Computing:** Allows sensitive operations like personal data processing or proprietary algorithms to run securely.
- **DRM and Content Protection:** Ensures that digital rights are enforced by only allowing access to content under secure conditions.

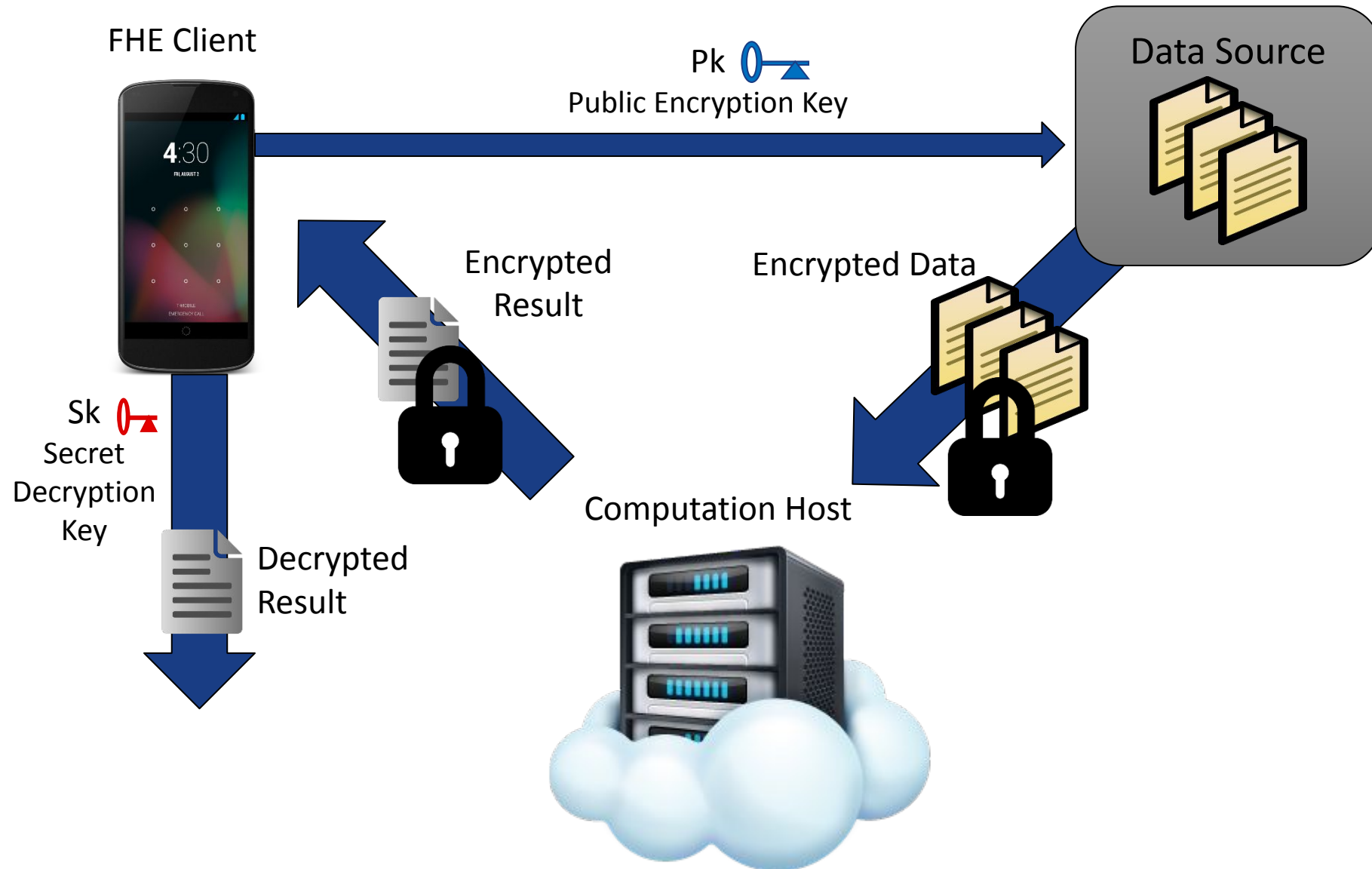


Spinning Up in Homomorphic Encryption for ML

WHAT IS HOMOMORPHIC ENCRYPTION?

- Encryption protocol with one extra operation: Evaluation
 - Allows for computation on encrypted data
 - Enables outsourcing of data storage/processing
- How is FHE related to symmetric and public key encryption?
 - FHE schemes provide efficient instantiations of post-quantum public-key and symmetric-key encryption schemes
 - Homomorphic encryption can be viewed as a generalization of public key encryption
- Key milestones in the history of homomorphic encryption
 - Rivest, Adleman, Dertouzos (1978) -- “On Data Banks and Privacy Homomorphisms”
 - Gentry (2009) -- “A Fully Homomorphic Encryption Scheme”
 - Multiple HE schemes developed after 2009

EXAMPLE OF FHE WORKFLOW



FHE vs OTHER SECURE COMPUTING APPROACHES

	FHE	MPC	Secure Enclaves/SGX
Performance	Compute-bound	Network-bound	Close to plaintext
Privacy	Encryption	Encryption / Non-collusion	Trusted Hardware
Non-interactive	✓	✗	✓
Cryptographic security	✓	✓	✗ (known attacks)

Hybrid approaches are also possible, e.g., MPC + FHE

TYPICAL FHE OPERATIONS

- Encrypt bits and perform logical AND, OR, XOR operations on the ciphertexts.
 - $0 \text{ AND } 1 \rightarrow 0$, $0 \text{ OR } 1 \rightarrow 1$, $1 \text{ XOR } 1 \rightarrow 0$
- Encrypt small integers and perform addition and multiplication, as long as the result does not exceed some fixed bound, for instance, if the bound is 10000
 - $123 + 456 \rightarrow 579$, $12 * 432 \rightarrow 5184$, $35 * 537 \rightarrow \text{overflow}$
- Encrypt 8-bit unsigned integers (between 0 and 255) and perform addition and multiplication modulo 256
 - $128 + 128 \rightarrow 0$, $2 * 129 \rightarrow 2$
- Encrypt fixed-point numbers and perform addition and multiplication with the result rounded to a fixed precision, for instance, two digits after the decimal point
 - $12.42 + 1.34 \rightarrow 13.76$, $2.23 + 5.19 \rightarrow 11.57$
- Different homomorphic encryption schemes support different plaintext types and different operations on them.

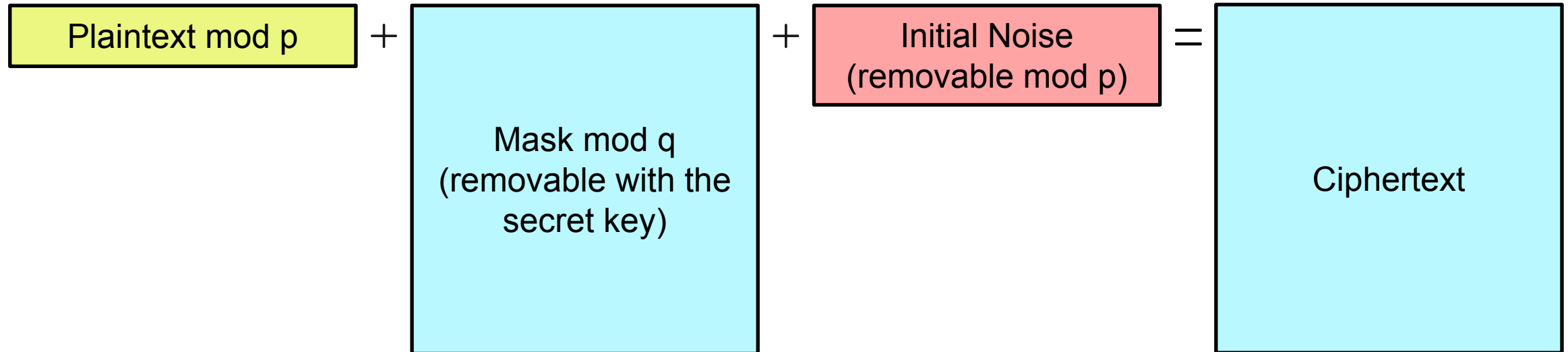
SOME EXAMPLES OF REAL-SCALE FHE APPLICATIONS

- Private information retrieval
 - <https://eprint.iacr.org/2017/1142>, IEEE S&P 2018
- Private set intersection
 - <https://eprint.iacr.org/2017/299>, ACM CCS 2017
 - <https://eprint.iacr.org/2018/787>, ACM CCS 2018
 - <https://eprint.iacr.org/2021/1116>, ACM CCS 2021
- Genome-wide association studies based on chi-square test and logistic regression training
 - <https://eprint.iacr.org/2020/563>, PNAS 2020
- Logistic regression training
 - <https://eprint.iacr.org/2018/662>, AAAI Conference on AI 2019
- Neural network inference (ResNet-20 to ResNet-110)
 - <https://eprint.iacr.org/2021/1688>, ICML 2022

MAIN CONCEPTS

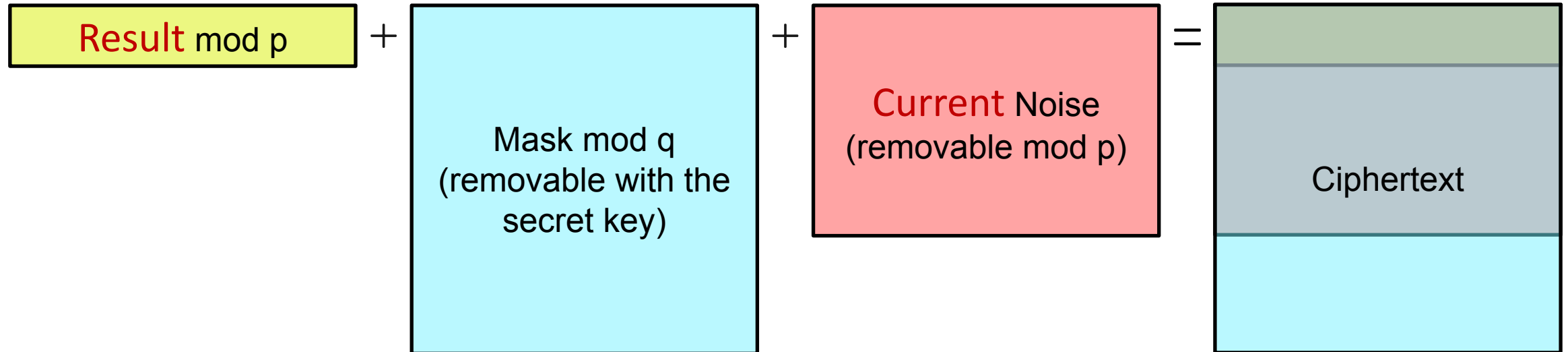
- *Homomorphic*: a (secret) mapping from plaintext space to ciphertext space that preserves arithmetic operations.
- *Mathematical Hardness: (Ring) Learning with Errors Assumption*
 - Every image (ciphertext) of this mapping looks uniformly random in range (ciphertext space).
- *Security level*: the hardness of inverting this mapping without the secret key
 - Often estimated as a work factor.
 - Example: 128 bits $\rightarrow 2^{128}$ operations to break using best known lattice attack
- *Plaintext*: Elements and operations of a polynomial ring $(\text{mod } x^n + 1, \text{mod } p)$.
 - Example: $3x^5 + x^4 + 2x^3 + \dots$
 - For all practical purposes, you can think of it as a vector of (small) finite integers
- *Ciphertext*: elements and operations of a polynomial ring $(\text{mod } x^n + 1, \text{mod } q)$.
 - Example: $7862x^5 + 5652x^4 + \dots$
 - For all practical purposes, you can think of it as a vector of (larger) finite integers
- *Noise*: random integers with Gaussian distribution, which are “added” to the plaintext to achieve the desired security level based on Ring Learning With Errors

FRESH ENCRYPTION



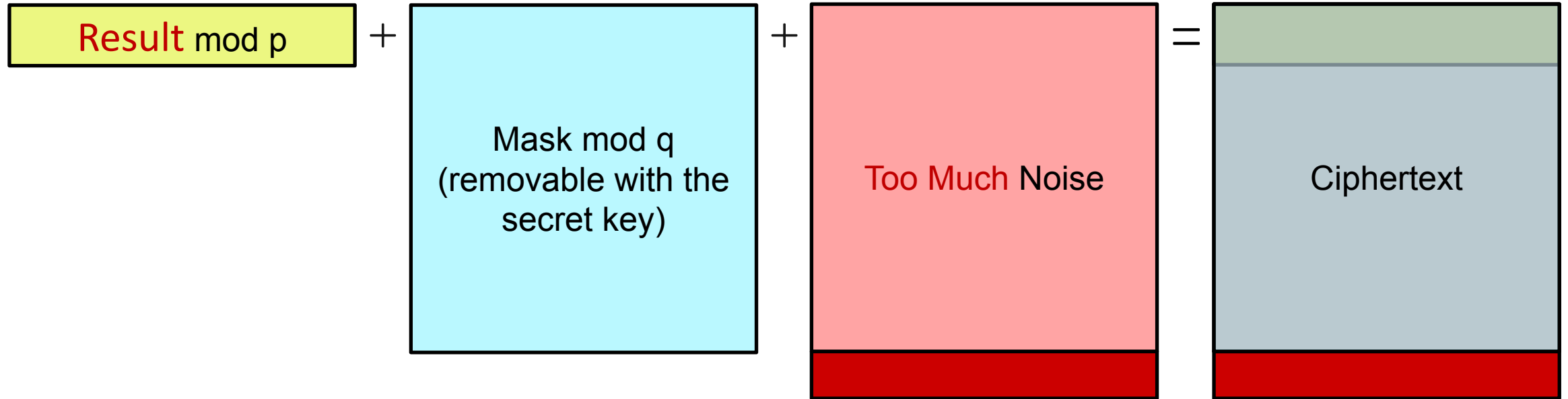
- Horizontal: each coefficient in a polynomial or in a vector.
- Vertical: size of coefficients.
- Initial noise is small in terms of coefficients' size.

AFTER SOME COMPUTATIONS



- Horizontal: each coefficient in a polynomial or in a vector.
- Vertical: size of coefficients.
- Initial noise is small in terms of coefficients' size.

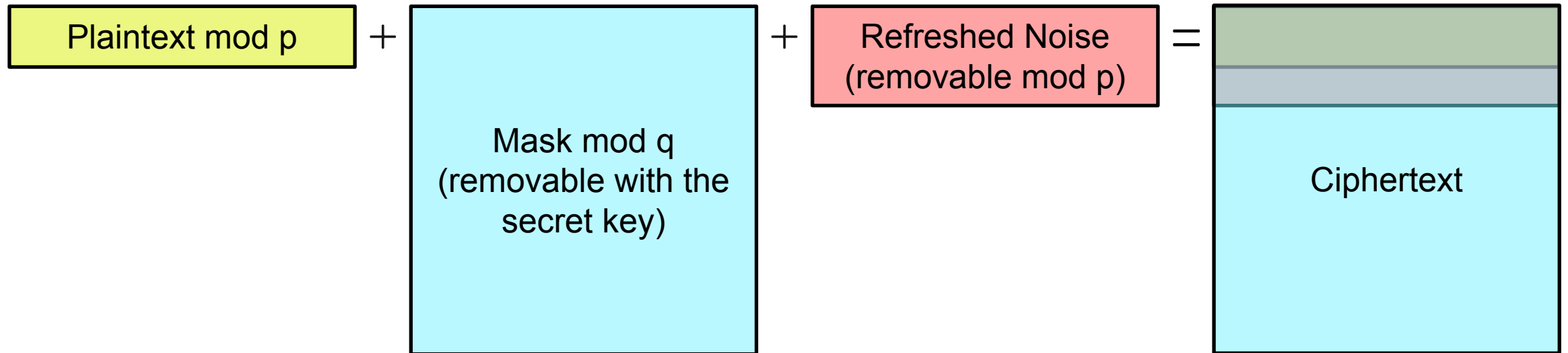
NOISE OVERFLOW (RESULTS IN DECRYPTION FAILURE)



- Horizontal: each coefficient in a polynomial or in a vector.
- Vertical: size of coefficients.
- Initial noise is small in terms of coefficients' size.

BOOTSTRAPPING (NOISE REFRESHING PROCEDURE)

Evaluates the decryption circuit homomorphically and resets the noise.



- Horizontal: each coefficient in a polynomial or in a vector.
- Vertical: size of coefficients.
- Initial noise is small in terms of coefficients' size.

TYPES OF HOMOMORPHIC ENCRYPTION

- Partially homomorphic encryption (weakest notion)
 - supports only one type of operation, e.g. addition or multiplication.
- Somewhat homomorphic encryption schemes
 - can evaluate two types of gates/operations, but only for a subset of circuits.
- **Leveled fully homomorphic encryption**
 - supports more than one operation but only computations of a predetermined size (typically multiplicative depth); supports much deeper circuits than somewhat homomorphic encryption
- **Fully homomorphic encryption**
 - supports arbitrary computation on encrypted data; it is the strongest notion of homomorphic encryption.



Classes of FHE Schemes

CLASSES OF HOMOMORPHIC SCHEMES

1. Modular (Exact) Integer Arithmetic: BGV / BFV

- Plaintext data represented as **vectors modulo a plaintext modulus “ t ”** (or their vectors)
- Computations expressed as **vectors arithmetic mod t**

2. Functional (Programmable) Bootstrapping: DM (FHEW) / CGGI (TFHE)

- Plaintext represented as **integers/Boolean values**
- Supports evaluation of arbitrary functions using Look-Up Tables (LUTs)
- Computation for each integer is evaluated separately

3. Approximate Number Arithmetic: CKKS

- Plaintext data represented as **vectors of real numbers** (or complex numbers)
- Compute model similar to **floating-point arithmetic** but dealing with fixed-point numbers

MODULAR (EXACT) INTEGER ARITHMETIC APPROACH

- Features:
 - Efficient SIMD computations over vectors of integers (using batching, also called CRT packing)
 - Fast high-precision integer arithmetic
 - Fast private information retrieval/private set intersection/secure database query
 - Leveled design (often used without bootstrapping)
- Main schemes:
 - Brakerski-Vaikuntanathan (BV) [BV11] - foundation for other schemes
 - Brakerski-Gentry-Vaikuntanathan (BGV) [BGV12]
 - Brakerski/Fan-Vercauteren (BFV) [Bra12, FV12]

FUNCTIONAL (PROGRAMMABLE) BOOTSTRAPPING APPROACH

- Features:
 - Fast number comparison
 - Initially proposed for Boolean circuits
 - Supports arbitrary functions by evaluating LUTs
 - Fast bootstrapping (noise refreshing procedure)
 - Does not support batching/CRT packing
- Related schemes:
 - Gentry-Sahai-Waters (GSW) [GSW13] – used in bootstrapping
 - Fastest Homomorphic Encryption in the West (DM/FHEW) [DM15]
 - Fast Fully Homomorphic Encryption over the Torus (CGGI/TFHE) [CGGI16,CGGI17]
 - Efficient FHEW Bootstrapping with Small Evaluation Keys [LMCKDEY23]

APPROXIMATE NUMBER ARITHMETIC APPROACH

- Features:
 - Efficient SIMD computations over vectors of real numbers (using batching)
 - Fast polynomial approximation
 - Relatively fast multiplicative inverse and Discrete Fourier Transform
 - Deep approximate computations, such as logistic regression learning
 - Leveled design, but also used with approximate bootstrapping in many ML applications
 - Best amortized bootstrapping time
- Selected schemes:
 - Cheon-Kim-Kim-Song (CKKS) [CKKS17]

SELECTING SECURITY PARAMETERS

The ciphertext dimension (degree of polynomial) should be chosen according to the security tables published at [HomomorphicEncryption.org](https://homomorphicencryption.org)

distribution	n	security level	logq	uSVP	dec	dual
(-1, 1)	1024	128	27	131.6	160.2	138.7
		192	19	193.0	259.5	207.7
		256	14	265.6	406.4	293.8
	2048	128	54	129.7	144.4	134.2
		192	37	197.5	233.0	207.8
		256	29	259.1	321.7	273.5
	4096	128	109	128.1	134.9	129.9
		192	75	194.7	212.2	198.5
		256	58	260.4	292.6	270.1
8192		128	218	128.5	131.5	129.2
		192	152	192.2	200.4	194.6
		256	118	256.7	273.0	260.6



FHE Approaches for ML

THREE MOST COMMON WAYS TO DO ENCRYPTED ML USING FHE

Here we focus on (supervised) learning using the models based on logistic regression, decision trees, and neural networks, i.e., relatively deep computations requiring bootstrapping.

1. **Approximate approach based on CKKS**
 - Fastest for most ML applications, especially with larger problem sizes.
 - Polynomial approximations should be used with care.
2. **Hybrid approximate/LUT approach based on CKKS and DM (FHEW) /CGGI (TFHE)**
 - Significantly slower than approach 1 for most ML applications.
 - Evaluates “tricky” non-linear functions, such as comparison, using functional bootstrapping in DM/CGGI.
3. **LUT approach based on functional bootstrapping in DM/CGGI**
 - Slowest option (typically by orders of magnitude compared to option 1) and does not scale well with the problem size.
 - Easiest to use in most cases.

APPROXIMATE APPROACH BASED ON CKKS

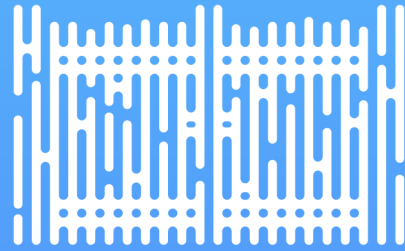
- Complicated functions are approximated using polynomials
 - Typically Chebyshev interpolations or similar minimax methods are used
 - Evaluates functions over vectors of real numbers, e.g., a function is evaluated for 32K real numbers at once
 - When building the polynomial interpolation, the input range has to be specified
- Approximate CKKS bootstrapping is used to support deep computations, such as logistic regression training
 - Approximate bootstrapping significantly increases the approximation error after first bootstrapping, but further bootstrapping operations typically have a small effect for many ML applications (the ML computations are often approximate in nature)
 - Multiprecision CKKS (META-BTS) bootstrapping was recently proposed in <https://eprint.iacr.org/2022/1167> (CCS'22)
- Amortized cost of CKKS bootstrapping gets as low as 1ms per real number [BMT+21]
- Inference is fast for many models, e.g., logreg inference for hundreds of samples and dozens of features takes less than 1 second

HYBRID APPROXIMATE/LUT APPROACH BASED ON CKKS+DM/CGGI

- CKKS is used for all polynomial/matrix arithmetic computations
- For “tricky” non-linear functions, LUT evaluation using functional bootstrapping with DM/CGGI is used
 - Useful for comparisons
 - Can be used for piecewise polynomial evaluation, addressing the input range issue in polynomial approximations
- Requires scheme switching from CKKS to/from DM/CGGI
- LUT evaluation is often the main bottleneck

LUT APPROACH BASED ON FUNCTIONAL BOOTSTRAPPING IN DM/CGGI

- Any deep learning algorithm can be evaluated using LUTs for non-linear functions
- The main drawback is performance
 - Functional bootstrapping does not support the evaluation of a LUT over a vector of integers in a SIMD manner
 - LUT evaluation even for a small plaintext modulus (few bits of precision) requires 100ms or so
 - To evaluate an arbitrary function for a larger plaintext space, many small LUT evaluations are needed
 - Typically orders of magnitude slower than the approximate approach based on CKKS
 - Even inference takes substantial time, e.g., logreg inference for hundreds of samples and dozens of features takes more than 10 minutes



Selected Studies in PPML using FHE

RECENT DARPA PROGRAMS

- Cooperative Secure Learning (CSL) [August 2020 – January 2022]
 - Develop methods to protect data, models, and model outputs among a community of entities desiring to securely share their information to better inform ML model development
 - Enable multiple parties to cooperate for the purpose of improving each other's ML models while assuring that each entity's individual, pre-existing datasets and models will remain private
- Data PRotection in Virtual Environments (DPRIVE) [January 2021 – present]
 - Develop a hardware accelerator for FHE computations that will dramatically reduce the compute runtime overhead compared to software-based FHE approaches
 - Motivating applications are logistic regression training, CNN inference, and CNN training

SELECTED PAPERS FOR APPROXIMATE METHOD BASED ON CKKS

- Logistic regression training
 - <https://eprint.iacr.org/2018/662>, AAAI Conference on AI 2019
 - 422108 samples over 200 features in 17 hours on a single machine
- Genome-wide association studies based on chi-square test and logistic regression training
 - <https://eprint.iacr.org/2020/563>, PNAS 2020
 - 500000 SNPs and 100000 individuals in 5.6 hours on a single machine
- ResNet-20 deep neural network evaluation
 - <https://eprint.iacr.org/2021/783>; CIFAR-10 in 3 hours on a single server with 64 threads
 - <https://eprint.iacr.org/2021/1688>; CIFAR-10 in 40 minutes on a single server with 1 thread; Resnet-110 for the same setup took about 3.7 hrs

SELECTED PAPERS FOR HYBRID METHOD

- Semi-parallel logistic regression training (GWAS)
 - <https://eprint.iacr.org/2019/101>, BMC Medical Genomics 2020
 - 10643 SNPs, 245 patients, and 3 covariates: 4 min. to 3 hrs on a single machine
 - The best approaches based on the CKKS method took few minutes for the problem sizes that required few hours for the hybrid approach
- Decision tree evaluation and K-means clustering
 - <https://eprint.iacr.org/2020/1606>, IEEE S&P 2021
 - Decision tree evaluation: for 60 internal nodes, 57 features, and 2 classification labels the runtime was about 7 seconds
 - K-means clustering: for 4096 data points and 8 clusters, the runtime was 52 minutes



Introduction to Approximate FHE

MOTIVATION FOR APPROXIMATE FHE

- Good for any application where we work with real numbers, e.g., where we use floating-point numbers
 - Encrypt real numbers and perform addition and multiplication with the result rounded to a fixed precision, for instance, two digits after the decimal point
 - $12.42 + 1.34 = 13.76$, $2.23 * 5.19 = 11.57$
- The main limitation of integer homomorphic encryption (BGV/BFV) is that it requires very large plaintext moduli to support operations over real numbers
 - All computations in integer HE are performed exactly
 - Integer HE rapidly becomes inefficient as further multiplications are performed
- Approximate homomorphic encryption allows dropping least significant bits by *rescaling* the encrypted data, similar to how it is done for floating-point numbers in practice
- Typical applications of approximate HE
 - Statistical computations
 - Polynomial evaluation
 - Matrix arithmetic
 - Regression inference and training
 - Evaluation of non-linear/non-smooth functions using their polynomial approximations

FIXED-POINT ARITHMETIC WITH RESCALING

- - Suppose we have two numbers: $a = 0.125654$ and $b = 3.534365$
 - We choose a scaling factor Δ that converts these real numbers into integers (like in fixed-point arithmetic)
 - In this case, $\Delta = 10^6$ is a good choice to maintain the same precision
 - We multiply a and b by the scaling factor Δ , yielding $a' = a * \Delta = 125654$ and $b' = b * \Delta = 3534365$
 - If we want to compute $a + b$, we do the following
 - $a' + b' = 3660019$, which corresponds to $a + b = \frac{a' + b'}{\Delta} = 3.660019$
 - If we want to compute $a * b$, we do the following
 - $a' * b' = 444107099710$, which corresponds to $a * b = \frac{a' * b'}{\Delta^2} = 0.444107099710$
 - We convert $a * b$ to the original scale Δ by dropping the 6 least significant digits (dividing by Δ), i.e.,
 - Compute $\frac{a' * b'}{\Delta} = 444107$ and use it for future computations

FIXED-POINT ARITHMETIC WITH RESCALING IN CKKS

- - The Cheon-Kim-Kim-Song (CKKS) scheme is the approximate homomorphic encryption implemented in OpenFHE
 - Multiple variants of the CKKS are implemented in OpenFHE, but they all share common properties and vary only in the approximation error, performance, and usability
 - The fixed-point arithmetic with rescaling in CKKS has additional two features
 - Binary fixed-point arithmetic, i.e., $\Delta = 2^p$ is used instead of decimal fixed-point arithmetic
 - CKKS operations introduce extra approximation error, which “erases” typically 12-25 least significant bits
 - The actual scaling factor in CKKS should be set to a higher value to compensate for the CKKS approximation error
 - By default, the CKKS implementation in OpenFHE automatically performs rescaling, similar to how normalization and other internal adjustments are done in double-precision floating-point arithmetic

MAIN DATA STRUCTURE

- The main data structure is a vector (array) of real numbers
- Many real numbers (typically between 2K and 64K) are “packed” in one vector (ciphertext)
 - Let us denote the vector size as n (a power of two)
- **Addition** and **multiplication** of n real numbers can be done using a single addition/multiplication
 - Similar to Single Instruction Multiple Data (SIMD) instruction sets available on many modern processors
 - The SIMD capability should be used as much as possible to achieve best efficiency
- **Rotation** operation is added to allow accessing the value at a specific index of the array
- Addition, multiplication, and rotation are three primitive operations in approximate FHE

COMPLETE LIST OF PRIMITIVE OPERATIONS

- Two-argument operations (the plaintext can represent a vector of real numbers or a single real number)
 - Ciphertext-Ciphertext addition: **EvalAdd**
 - Ciphertext-Plaintext addition: **EvalAdd**
 - Ciphertext-Ciphertext multiplication: **EvalMult**
 - Ciphertext-Plaintext multiplication: **EvalMult**
 - Ciphertext-Ciphertext subtraction: **EvalSub**
 - Ciphertext-Plaintext subtraction: **EvalSub**
- Unary operations
 - Negation: **EvalNegate**
 - Vector rotation: **EvalRotate**
- The result of all these operations is a ciphertext, i.e., an encrypted vector
 - The benefit of this in practice is that mixed model-data modes can be supported, e.g.,
 - Encrypted model, data in the clear
 - Model in the clear, encrypted data

DATA ENCODING

- Packing technique: **CKKSPackedEncoding**

- Packs real numbers into a vector of size n

- Supports component-wise addition (**EvalAdd**) and multiplication (**EvalMult**)

$$\begin{array}{cccccc} [1.1] & [4.4] & [5.5] & [1.5] & [4.5] & [6.75] \\ [2.2] + [5.5] = [7.7], & [2.5] * [5.0] = [12.50] \\ [3.3] & [6.6] & [9.9] & [3.5] & [6.1] & [21.35] \end{array}$$

- Adds a new rotation operation (**EvalRotate**)

- Left shift: positive index
- Right shift: negative index
- Rotations work cyclically over a vector of size n

MAIN PARAMETERS

- Scaling factor $\Delta = 2^p$
 - Determines the precision of computations
- Ciphertext modulus q
 - Functional parameter that determines how many computations are allowed (how much noise can be tolerated)
 - Often set implicitly using the value of multiplicative depth specified by the user
- Ciphertext dimension N
 - Minimum value is computed based on the desired security level and ciphertext modulus q
 - It is also double the size of the vector of encrypted real numbers, i.e., $N = 2n$
- Batch size b
 - By default, $b = N / 2$ (full packing)
 - A smaller number can be used to support sparse packing (smaller vectors) to reduce computational complexity and noise if the application operates on smaller vectors

GUIDELINES FOR SETTING SCALING FACTOR

- - The scaling factor $\Delta = 2^p$ determines the precision after the radix point
 - Think of this as precision in the fixed-point sense
 - CKKS “erases” 12-25 least significant bits (depending on the multiplicative depth)
 - Each addition and multiplication may consume up to 1 bit (typically significantly less)
 - Hence the value of p for desired precision v is something like $p \approx v + 20$
 - It can be adjusted as needed, depending on the multiplicative depth of the computation
 - Just like floating-point arithmetic, approximate homomorphic encryption introduces an error, and errors from prior computations get accumulated
 - OpenFHE outputs estimated precision for a decrypted CKKS result

GUIDELINES FOR SETTING CIPHERTEXT MODULUS

- Ciphertext modulus q is the main functional parameter that is determined by the computation
 - Each arithmetic operation increases the noise, and q should be large enough to accommodate the noise from all arithmetic operations
 - From the noise perspective, multiplication is much costlier than addition
 - In OpenFHE, q is automatically computed based on the multiplicative depth and scaling factor Δ
- Multiplicative depth is not necessarily the number of multiplications
 - For example, if we need to compute $\mathbf{a*b*c*d}$, we can compute $\mathbf{e=a*b}$ and $\mathbf{f=c*d}$ using one level, and then compute $\mathbf{e*f}$ using the second level. Hence we use 2 levels (depth of 2) rather 3 if we were to do the multiplication sequentially.
 - This technique is called **binary tree multiplication**, and it should be used to minimize the multiplicative depth wherever possible.

GUIDELINES FOR SETTING CIPHERTEXT DIMENSION

- Ciphertexts are represented as two arrays of size N
- This size N , called ciphertext dimension, should have a certain minimum value to comply with the chosen security level and desired ciphertext modulus
- Main options for security levels in OpenFHE (we implemented the recommendations from the HE standard published at [HomomorphicEncryption.org](https://homomorphicencryption.org)):
 - *HEStd_128_[classic/quantum]* – 128-bit security against [classical/quantum] computers
 - *HEStd_192_[classic/quantum]* – 192-bit security against [classical/quantum] computers
 - *HEStd_256_[classic/quantum]* – 256-bit security against [classical/quantum] computers
 - *HEStd_NotSet* – toy settings (for debugging and prototype development)
- The ciphertext dimension N also determines the maximum size of the vector of encrypted real numbers ($n = N/2$).
 - It may sometimes be useful to use a larger ring dimension than the minimum one needed for security.
 - In this case, the user can specify the ring dimension explicitly.

CKKS SECURITY FOR SCENARIO OF SHARED DECRYPTIONS

- Li and Micciancio recently showed (<https://eprint.iacr.org/2020/1533>, EUROCRYPT'21) that the IND-CPA security may not be strong enough when decryption results need to be published or shared with untrusted parties; they introduced IND-CPA^D security to account for this.
- They also described mitigation strategies. Adding enough Gaussian noise during decryption is the most common option.
- In a later paper, Li et al. (<https://eprint.iacr.org/2022/816>, CRYPTO'22) quantified how much noise should be added for 128 bits of security (about extra 45 bits on top of the existing approximation error) for the scenario with an unbounded number of related queries.
- OpenFHE implements a practical mitigation for this scenario, and supports adding 45 bits of precision (using 128-bit CKKS) when decryptions are allowed to be shared without any restrictions

HIGH-PRECISION CKKS

- OpenFHE includes a high-precision CKKS implementation
 - Scaling factor in this case can be as large as 2^{119} (compared to 2^{59} for the regular CKKS implementation in OpenFHE)
 - The high-precision CKKS implementation provides support for double-precision arithmetic, i.e., 52 bits if the scaling factor is set to 70 or more bits
 - Another benefit is the support for 128-bit security for the scenario where decryption results are shared
 - High-precision CKKS is recommended for ML applications that require IND-CPA^D security
 - Runtime is about 4-6x slower than for regular CKKS implementation for the same ring dimension (additional 2x when the ring dimension needs to be doubled for LWE security)



OpenFHE library

OPENFHE DESIGN PRINCIPLES

- OpenFHE is an open-source C++17 FHE software library launched in July 2022 that incorporates selected design ideas from prior FHE projects, including **PALISADE**, **HElib**, **HEAAN**, and **FHEW**, and includes several new design concepts and ideas.
- The main new design features can be summarized as follows:
 - From the **cryptology** perspective, we assume from the very beginning that all implemented FHE schemes will support **bootstrapping** and **scheme switching**
 - From the **performance** perspective, OpenFHE supports multiple hardware acceleration backends using a standard **Hardware Abstraction Layer (HAL)**
 - From the **usability** perspective, OpenFHE includes both
 - **user-friendly modes**, where all maintenance operations, such as modulus switching, key switching, and bootstrapping, are automatically invoked by the library, and
 - **compiler-friendly modes**, where an external compiler makes these decisions

CRYPTOGRAPHIC CAPABILITIES

- OpenFHE includes efficient implementations of all common FHE schemes:
 - Brakerski/Fan-Vercauteren (BFV) scheme for integer arithmetic
 - Brakerski-Gentry-Vaikuntanathan (BGV) scheme for integer arithmetic
 - Cheon-Kim-Kim-Song (CKKS) scheme for real-number arithmetic (includes approximate bootstrapping)
 - Ducas-Micciancio (DM/FHEW), Chillotti-Gama-Georgieva-Izabachene (CGGI/TFHE), and Lee-Micciancio-Kim-Choi-Deryabin-Eom-Yoo (LMKCDEY) schemes for evaluating Boolean circuits and arbitrary functions over larger plaintext spaces using lookup tables
- OpenFHE includes the following multiparty extensions of FHE:
 - Threshold FHE for BGV, BFV, and CKKS schemes
 - Interactive bootstrapping for Threshold CKKS
 - Proxy Re-Encryption for BGV, BFV, and CKKS schemes
- OpenFHE also supports switching between CKKS and FHEW/TFHE to evaluate non-smooth functions, e.g., comparison, using FHEW/TFHE functional bootstrapping.

SCHEME SUPPORT MATRIX

Library/ Scheme or Extension	BGV	BGV Bootstr.	BFV	CKKS	CKKS Bootstr.	DM	CGGI	Threshold FHE (MP)	PRE (MP)
FHEW						✓			
HEAAN				✓	✓				
HELib	✓	✓		✓					
Lattigo	✓		✓	✓	✓			✓	
OpenFHE	✓	*	✓	✓	✓	✓	✓	✓	✓
PALISADE	✓		✓	✓		✓	✓	✓	✓
SEAL	✓		✓	✓					
TFHE-rs							✓		
TFHE-lib							✓		

* - prototype exists, but not part of release

KEY FACTS ABOUT OPENFHE

- Current version is 1.1.2 (released on December 16, 2023)
- Designed by (some of) authors of PALISADE, HElib, HEAAN, and FHEW libraries
- Official successor of PALISADE
- Complies with the HomomorphicEncryption.org post-quantum security standards for homomorphic encryption
- We offer OpenFHE under the 2-clause BSD open-source license, making it easier to wrap and redistribute OpenFHE in products
- Generously supported by DARPA
- A community-driven open-source project developed by a diverse group of contributors from both industry and academia, including Duality, Samsung, Intel, MIT, UCSD, and others
- Google Transpiler uses the CGGI (TFHE) implementation as the FHE backend
- OpenFHE is formally affiliated with the NumFocus stable of open-source software projects

DESIGN PAPER [<https://eprint.iacr.org/2022/915>]

Paper 2022/915

OpenFHE: Open-Source Fully Homomorphic Encryption Library

Ahmad Al Badawi, Duality Technologies

Jack Bates, Duality Technologies

Flavio Bergamaschi, Intel Corporation

David Bruce Cousins, Duality Technologies

Saroja Erabelli, Duality Technologies

Nicholas Genise, Duality Technologies

Shai Halevi, Algorand Foundation

Hamish Hunt, Intel Corporation

Andrey Kim, Samsung Advanced Institute of Technology

Yongwoo Lee, Samsung Advanced Institute of Technology

Zeyu Liu, Duality Technologies

Daniele Micciancio, University of California, San Diego, Duality Technologies

Ian Quah, Duality Technologies

Yuriy Polyakov, Duality Technologies

Saraswathy R.V., Duality Technologies

Kurt Rohloff, Duality Technologies

Jonathan Saylor, Duality Technologies

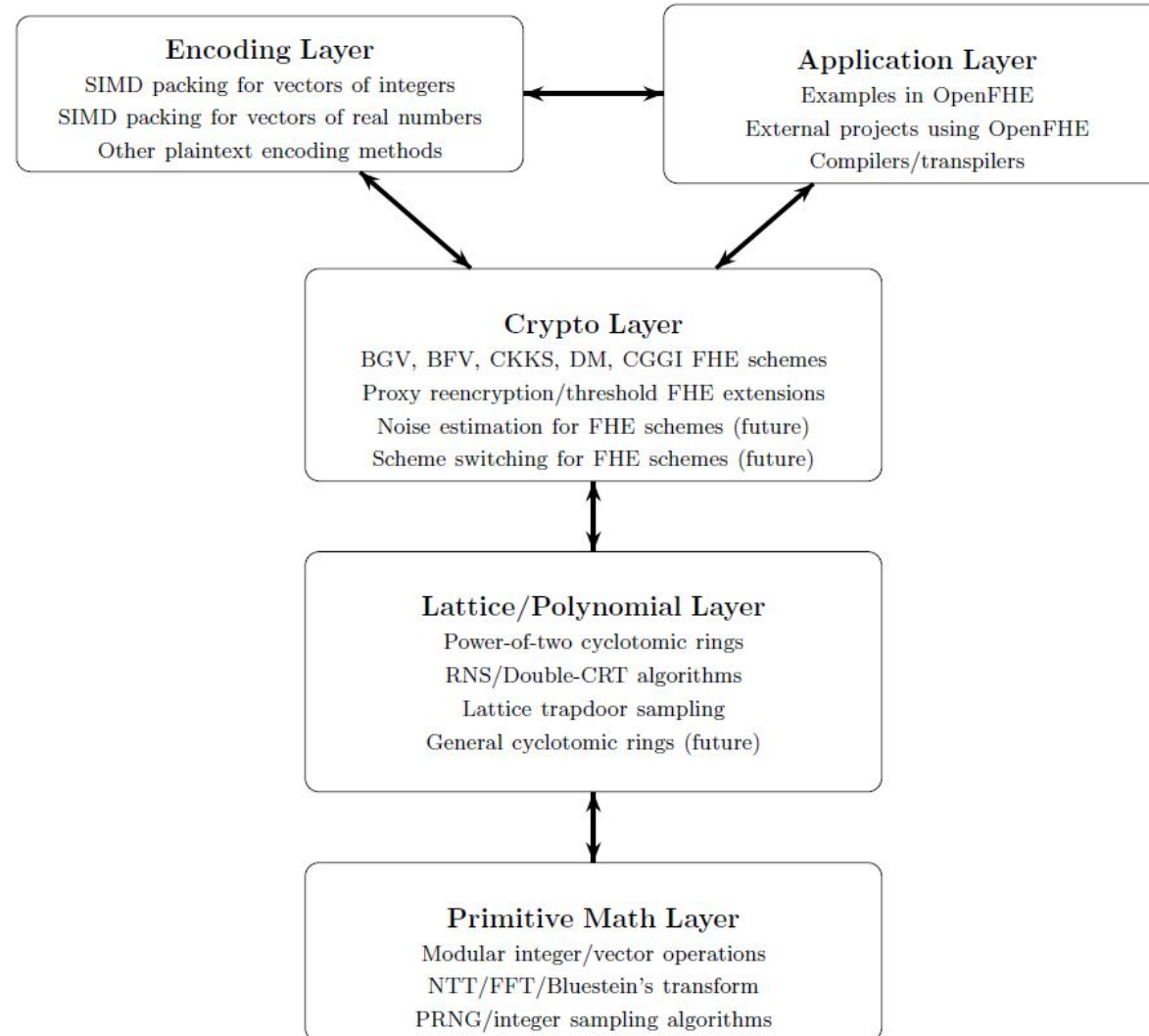
Dmitriy Suponitsky, Duality Technologies

Matthew Triplett, Duality Technologies

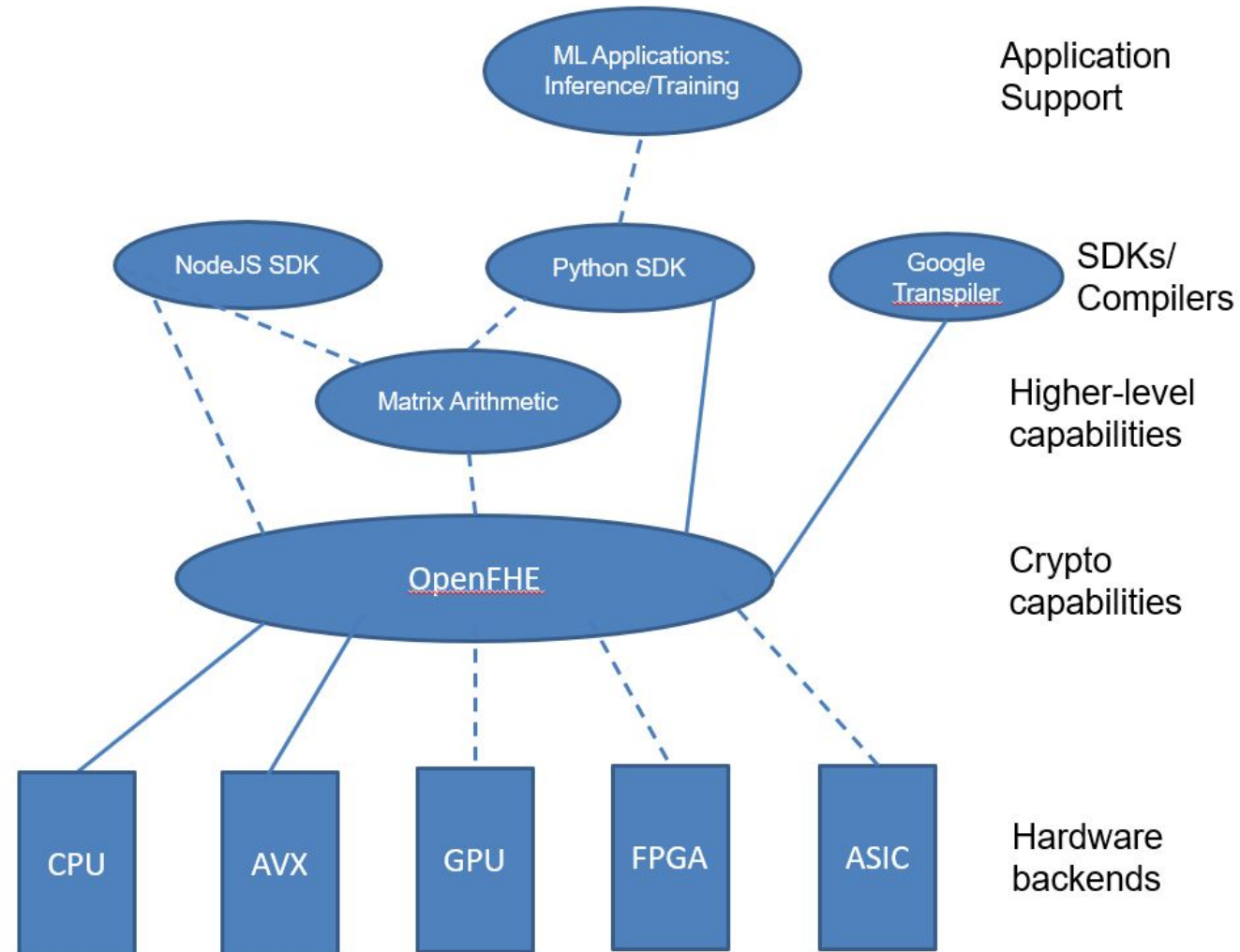
Vinod Vaikuntanathan, Massachusetts Institute of Technology, Duality Technologies

Vincent Zucca, DALI, Universite de Perpignan Via Domitia, LIRMM, University of Montpellier

LAYERS IN OPENFHE (CONTRIBUTOR VIEW)



BROADER OPENFHE COMMUNITY (USER VIEW)



OPENFHE VISION FOR MACHINE LEARNING (ML) USING FHE

- The main ML focus is on
 - Approximate method based on CKKS
 - Hybrid approximate/LUT approach based on CKKS and DM (FHEW) /CGGI (TFHE)
- Features that are already available in OpenFHE
 - CKKS bootstrapping to support deep learning
 - Large-precision comparison and small-precision LUT evaluation
 - Scheme switching between CKKS and DM/CGGI to evaluate comparisons and (arg)min
 - Python SDK
- Features under development
 - NodeJS SDK
 - Matrix arithmetic library

MAIN RESOURCES AND LINKS FOR OPENFHE

- OpenFHE design paper: <https://eprint.iacr.org/2022/915>
- OpenFHE website: <https://openfhe.org>
- ReadTheDocs documentation for OpenFHE:
<https://openfhe-development.readthedocs.io/en/latest/>
- OpenFHE development repository: <https://github.com/openfheorg/openfhe-development>
- OpenFHE github organization where various OpenFHE-dependent projects are housed:
<https://github.com/openfheorg>
- Community Forum for OpenFHE: <https://openfhe.discourse.group/>



CKKS Examples in Python

STEP 1 – SET CRYPTOCONTEXT

```
from openfhe import *

mult_depth = 1
scale_mod_size = 50
batch_size = 8

parameters = CCParamsCKKSRNS()
parameters.SetMultiplicativeDepth(mult_depth)
parameters.SetScalingModSize(scale_mod_size)
parameters.SetBatchSize(batch_size)

# Enable the features that you wish to use
cc = GenCryptoContext(parameters)
cc.Enable(PKESchemeFeature.PKE)
cc.Enable(PKESchemeFeature.KEYSWITCH)
cc.Enable(PKESchemeFeature.LEVELED_SHE)
```

STEP 2 – KEY GENERATION

```
# Generate a public/private key pair  
keys = cc.KeyGen()
```

```
# Generate the relinearization key  
cc.EvalMultKeyGen(keys.secretKey)
```

```
# Generate the rotation evaluation keys  
cc.EvalRotateKeyGen(keys.secretKey, [1, -2])
```


STEP 3 – ENCRYPTION

Inputs

```
x1 = [0.25, 0.5, 0.75, 1.0, 2.0, 3.0, 4.0, 5.0]
```

```
x2 = [5.0, 4.0, 3.0, 2.0, 1.0, 0.75, 0.5, 0.25]
```

Encoding as plaintexts

```
ptx1 = cc.MakeCKKSPackedPlaintext(x1)
```

```
ptx2 = cc.MakeCKKSPackedPlaintext(x2)
```

Encrypt the encoded vectors

```
c1 = cc.Encrypt(keys.publicKey, ptx1)
```

```
c2 = cc.Encrypt(keys.publicKey, ptx2)
```

STEP 4 – EVALUATION

Homomorphic additions

```
c_add = cc.EvalAdd(c1, c2)
```

Homomorphic subtraction

```
c_sub = cc.EvalSub(c1, c2)
```

Homomorphic scalar multiplication

```
c_scalar = cc.EvalMult(c1,4)
```

Homomorphic multiplication

```
c_mult = cc.EvalMult(c1, c2)
```

Homomorphic rotations

```
c_rot1 = cc.EvalRotate(c1, 1)
```

```
c_rot2 = cc.EvalRotate(c1, -2)
```

STEP 5 – DECRYPTION

```
# Decrypt the result of addition
precision = 8
print("Results of homomorphic computations:")
result = cc.Decrypt(c1, keys.secretKey)
result.SetLength(batch_size)
print("x1 = " + str(result))
print("Estimated precision in bits: " + str(result.GetLogPrecision()))

# Decrypt the result of scalar multiplication
result = cc.Decrypt(c_scalar, keys.secretKey)

# Decrypt the result of multiplication
result = cc.Decrypt(c_mult, keys.secretKey)

# Decrypt the result of rotations
result = cc.Decrypt(c_rot1, keys.secretKey)
result = cc.Decrypt(c_rot2, keys.secretKey)
```

OTHER PYTHON EXAMPLES FOR CKKS

Examples listed in README.md of <https://github.com/openfheorg/openfhe-python>

- FHE for arithmetic over real numbers (CKKS):
 - [Simple Code Example](#)
 - [Advanced Code Example](#)
 - [Advanced Code Example for High-Precision CKKS](#)
 - [Arbitrary Smooth Function Evaluation](#)
 - [Simple CKKS Bootstrapping Example](#)
 - [Advanced CKKS Bootstrapping Example](#)
 - [Double-Precision \(Iterative\) Bootstrapping Example](#)
- SVM examples: <https://github.com/openfheorg/python-svm-examples>
 - Linear and polynomial kernel SVM models
- Regression inference/training examples: <https://github.com/openfheorg/python-log-reg-examples>
 - Will be discussed in the next Hands-On sessions



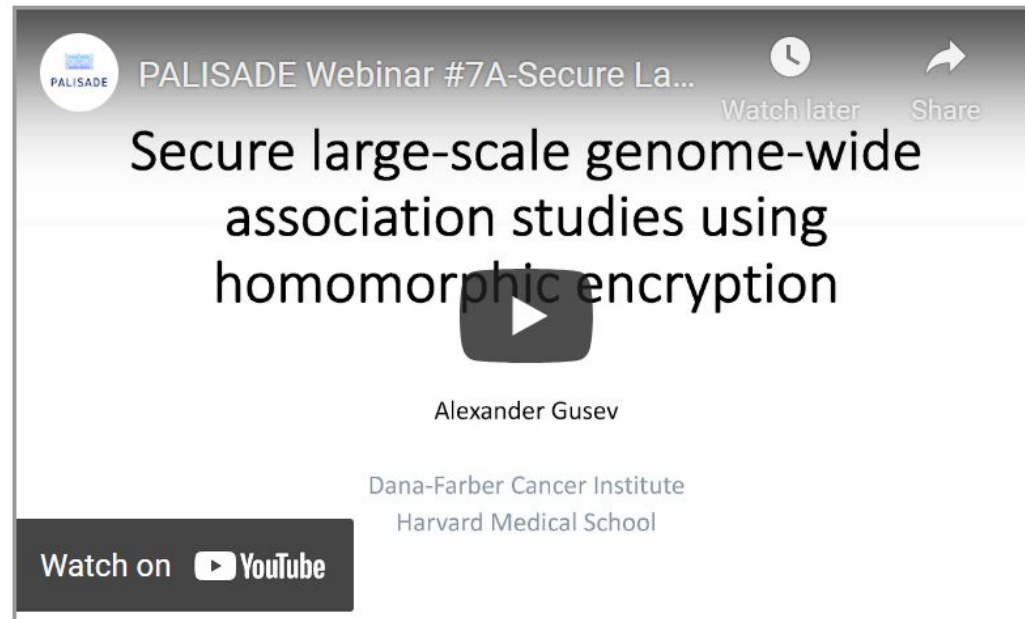
ML Applications and FHE Challenges



ML Applications

SECURE GENOME-WIDE ASSOCIATION STUDIES

Webinar #7A – Secure Large-Scale Genome-Wide Association Studies using Homomorphic Encryption, by Alexander Gusev, April 30, 2021 (Slides)



The image shows a YouTube video player thumbnail. At the top left is the PALISADE logo. The title of the video is "Secure large-scale genome-wide association studies using homomorphic encryption". Below the title is a play button icon. The presenter's name, "Alexander Gusev", is listed below the play button. Underneath the name are the affiliations "Dana-Farber Cancer Institute" and "Harvard Medical School". At the bottom left, there is a "Watch on YouTube" button. In the top right corner of the video frame, there are icons for "Watch later" and "Share".

Secure large-scale genome-wide association studies using homomorphic encryption

Marcelo Blatt^{a,1}, Alexander Gusev^{a,b,1}, Yuriy Polyakov^{a,1,2}, and Shafi Goldwasser^{a,c,1,2}

^aDuality Technologies, Inc., Newark, NJ 07103; ^bDana-Farber Cancer Institute, Harvard Medical School, Boston, MA 02215; and ^cSimons Institute for the Theory of Computing, University of California, Berkeley, CA 94720

Contributed by Shafi Goldwasser, February 15, 2020 (sent for review October 18, 2019; reviewed by Jung Hee Cheon and David J. Wu)

Genome-wide association studies (GWASs) seek to identify genetic variants associated with a trait, and have been a powerful approach for understanding complex diseases. A critical challenge for GWASs has been the dependence on individual-level data that typically have strict privacy requirements, creating an urgent need for methods that preserve the individual-level privacy of participants. Here, we present a privacy-preserving framework based on several advances in homomorphic encryption and demonstrate that it can perform an accurate GWAS analysis for a real dataset of more than 25,000 individuals, keeping all individual data encrypted and requiring no user interactions. Our extrapolations show that it can evaluate GWASs of 100,000 individuals and 500,000 single-nucleotide polymorphisms (SNPs) in 5.6 h on a single server node (or in 11 min on 31 server nodes running in parallel). Our performance results are more than one order of magnitude faster than prior state-of-the-art results using secure multiparty computation, which requires continuous user interactions, with the accuracy of both solutions being similar. Our homomorphic encryption advances can also be applied to other domains where large-scale statistical analyses over encrypted data are needed.

communication and computationally intensive nature of the garbled circuit solution, GWASs beyond monogenic diseases were not addressed, and the patient cohort was small. Jagadeesh et al. estimated that, even for the monogenic example, garbled circuits would be at least 5,000 times faster than FHE. Cho et al. (6) followed by successfully computing a GWAS by dividing data among multiple servers and computing the GWAS via multiparty secure protocol among the servers, subsets of which are trusted not to collaborate against other servers, else privacy is lost. Here, we no longer need to resort to this trust assumption. We are successfully using HE to encrypt the genomic sequences of study participants while enabling GWAS computations without the ability to decrypt, and scaling to hundreds of thousands of samples (Fig. 1).

We implement two common GWAS techniques—the allelic chi-square test for case control differences and a logistic regression approximation (LRA) with covariates—within our HE framework. The LRA algorithm utilizes a previously proposed semiparallel approach to efficiently iterate over each genetic variant without requiring repeated likelihood maximizations (7). Our HE LRA implementation of this approach works

Previous work: secure multi-party GWAS

Encrypted computing approach: secure multi-party computation^[1]

- Statistical test: Cochran Armitage trend test
- Benchmark GWAS: 26k samples x 260k SNPs

Results:

- Runtime on 100k samples x 500k SNPs: **193 hours**
- Requires live, interactive communication
- Logistic regression “does not yield a practical runtime”
- *Expect that HE would be 5,000-10,000x slower and infeasible^[2]*

[1] Cho et al. 2018 Nat Biotechnol; [2] Jagadeesh et al. 2017 Science

Results

	Prior MPC work	Our HE work
Algorithm	Multi-party computation	Homomorphic encryption
Statistical test	Cochran Armitage Trend (CAT)	Allelic χ^2 (CAT equivalent) Logistic regression
Dataset	26k samples x 260k SNPs + extrapolation	
Accuracy of test	Nearly perfect	
Runtime on 100k samples x 500k SNPs	193 hours Practically impossible	5.6 hours 234 hours (log reg)

MORE INFORMATION

- Source code: <https://github.com/openfheorg/openfhe-genomic-examples>
- PNAS Paper: <https://www.pnas.org/content/117/21/11608>
- PALISADE/OpenFHE Webinars
 - <https://www.openfhe.org/portfolio-item/secure-large-scale-genome/>
 - <https://www.openfhe.org/portfolio-item/best-practices-for-building-efficient-homomorphic-encryption-solutions/>

LOGISTIC REGRESSION TRAINING EXAMPLE

- Source code: <https://github.com/openfheorg/openfhe-logreg-training-examples>
- The examples were developed as part of the DARPA DPRIVE program
- The repository provides an implementation of logistic-regression model training and model inference on the 2014 US Infant Mortality Dataset
 - Logistic Regression Training is performed using Nesterov Accelerated Gradient Descent
 - CKKS bootstrapping is performed after each iteration of logistic regression training

CNN INFERENCE

- We recently developed a CNN prototype for a model with 7 convolution layers and one fully connected layer
- CNN inference of a CIFAR-10 image in OpenFHE takes several minutes
- The estimated runtime for an ASIC-accelerated implementation is about 3 orders of magnitude faster

COMMERCIAL ML CAPABILITIES BASED ON OPENFHE

- Duality Platform includes several ML/statistical capabilities based on CKKS in OpenFHE
 - Logistic regression training
 - Linear/ridge regression training
 - Inference for GLM and gradient boosted trees
 - Kaplan-Meier survival analysis
 - Descriptive statistics



FHERMA Project

FHERMA: PLATFORM FOR FHE CHALLENGES

- FHERMA is an open platform for Fully Homomorphic Encryption (FHE) challenges, jointly developed by Fair Math (formerly Yasha Lab) and the OpenFHE teams.
- The main goal of the project is to develop an open-source library of FHE components.
 - Such a library can significantly simplify application development and accelerate the adoption of FHE.
 - The initial focus is on components for Machine Learning and Blockchain applications.
- Launched on November 7, 2023
- URL: <https://fherma.io/>
- The winning solutions are published under the Apache 2.0 license
 - No IP restrictions
 - More details on the terms are available at <https://fherma.io/terms>

CHALLENGE TYPES

Black Box:

- Participants develop solutions according to the challenge requirements, process encrypted test data, and submit to the platform only the serialized final ciphertext.
- Does not require submitting source code or any other data that reveals the details of the solution itself.
- The main criterion for evaluating and ranking solutions in this type of challenge is accuracy.

White Box:

- It is not possible to evaluate the performance of the solution based solely on the ciphertext. While the Black Box type is suitable for many challenges, for others, it is crucial to obtain the most efficient solution from a performance perspective. For these, the White Box type is available.
- Participants are required to submit the source code of their projects to the platform.
- The platform will compile the project and run tests to measure performance and accuracy.
- The main criteria for evaluating and ranking solutions are performance and accuracy.
- Solutions uploaded to the platform are confidential and are not available to other participants.

GOVERNANCE

Transparency

- One of the main priorities is a transparent and equal environment for all participants.
- We eliminated the human factor when assessing the results of the participants.
- At the end of each challenge, the winner's solution is published in open form.

Committee

- Gurgen Arakelov, Fair Math
- Elvira Kharisova, Fair Math
- Yuriy Polyakov, Duality
- Kurt Rohloff, Duality

INITIAL FHE CHALLENGES

- Matrix Multiplication
 - Performing efficient multiplication of encrypted matrices
 - Award: \$3,000
- Sign Evaluation
 - Efficient comparison using CKKS
 - Award: \$3,000
- Logistic Function
 - One of main functions in machine learning
 - Award: \$5,000

Logistic Function Challenge

1. **Challenge type:** Black Box.
2. **Encryption Scheme:** CKKS.
3. **Input Data:**
 - Encrypted vector \mathbf{X}
 - Cryptocontext
 - Public key
 - Multiplication key
 - Rotation key for indexes [1, -1, 2, -2]

Two testcases:

Testcase 1:

1. **Batch size (Vector length):** 2048
2. **Maximum multiplicative depth:** 7
3. **Ring dimension:** 32768
4. **Scale Mod Size:** 50
5. **Element range:** The range for each element within the input vector is defined to be from -25 to 25.

Testcase 2:

1. **Batch size (Vector length):** 2048
2. **Maximum multiplicative depth:** 4
3. **Ring dimension:** 16384
4. **Scale Mod Size:** 50
5. **Element range:** The range for each element within the input vector is defined to be from -25 to 25.

Logistic Function Challenge

Two testcases: Testcase 2 is more important because standard methods do not provide sufficient accuracy for it.

Testcase 1:

1. **Batch size (Vector length):** 2048
2. **Maximum multiplicative depth:** 7
3. **Ring dimension:** 32768
4. **Scale Mod Size:** 50
5. **Element range:** The range for each element within the input vector is defined to be from -25 to 25.

EvalLogistic provides ~ **99.9%** accuracy

Winner's Solution provides ~ **99.9%** accuracy

Testcase 2:

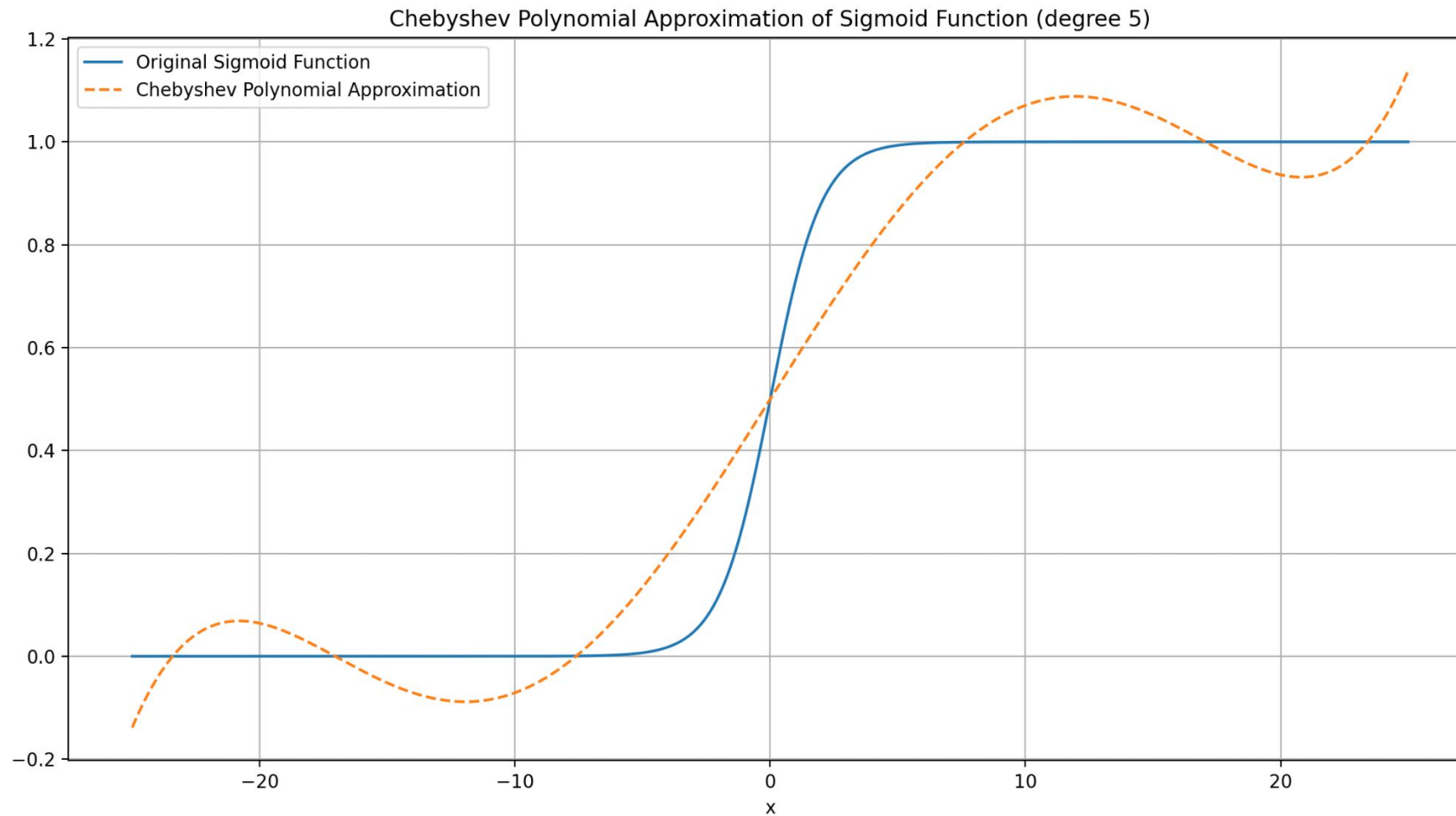
1. **Batch size (Vector length):** 2048
2. **Maximum multiplicative depth:** 4
3. **Ring dimension:** 16384
4. **Scale Mod Size:** 50
5. **Element range:** The range for each element within the input vector is defined to be from -25 to 25.

EvalLogistic provides ~ **88.12%** accuracy

Winner's Solution provides > **96.5%** accuracy

Logistic Regression Function Challenge

Testcase 2



NEW FHE CHALLENGES (TO BE ANNOUNCED EARLY NEXT WEEK)

- Encrypted image classification from the CIFAR-10 imageset (CKKS)
- RELU on encrypted data (CKKS)



THANK YOU

contact@openfhe.org

References

- [BGV14] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.
- [Bra12] Z. Brakerski. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *CRYPTO 2012*. Pages 868 – 886.
- [BV11] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Annual cryptology conference*, pages 505–524. Springer, 2011.
- [CGGI16]: I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Asiacrypt 2016 (Best Paper)*, pages 3-33.
- [CGGI17]: I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE. *ASIACRYPT (1) 2017*: 377-408.
- [CKKS17] J. H. Cheon, A. Kim, M. Kim, Y. Song, Homomorphic Encryption for Arithmetic of Approximate Numbers. In *ASIACRYPT 2017*. Pages 409–437.
- [DM15]: L. Ducas and D. Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. *EUROCRYPT 2015*.
- [FV12] J. Fan and F. Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *Cryptology ePrint Archive*. Report 2012/144, 2012. <http://eprint.iacr.org/2012/144>.
- [GSW13]: C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. *CRYPTO 2013*.
- [LMKCDEY13]: Y. Lee, A. Kim, D. Micciancio, R. Choi, Deryabin M., J. Eom, D. Yoo. Efficient FHEW Bootstrapping with Small Evaluation Keys, and Applications to Threshold Homomorphic Encryption, *EUROCRYPT 2023*.