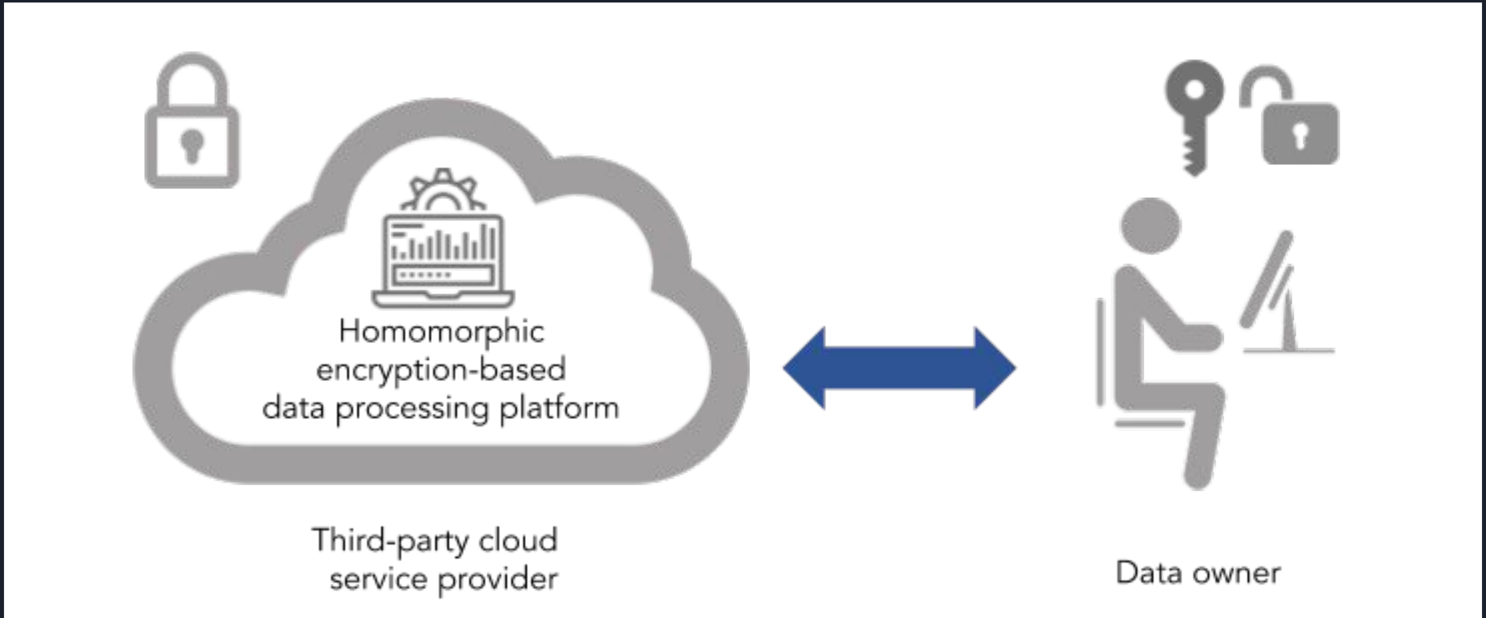
Logistic Regression over Encrypted Data

Marwan Nour
Advisor: Prof. Mohamed El Baker Nassar
American University of Beirut
Final Year Project - Spring 2020

Motivation

- Machine Learning as a Service (MLaaS)
- Protecting the tech consumer's privacy



Outline

I. Homomorphic Encryption
   A. BFV Encryption Scheme
   B. CKKS Encryption Scheme

II. Implemented Functionalities using CKKS
   A. Linear Transformation
   B. Matrix Matrix Multiplication

III. Logistic Regression
   A. Polynomial approximation of the Sigmoid function
   B. Polynomial Evaluation

IV. Benchmark Tests

V. Possible Optimizations and Improvements

VI. Challenges

Homomorphic Encryption

BFV (Brakerski-Fan-Vercauteren)

- Computes integers using Modular arithmetic
- Yields exact results
- Number of multiplications limited by “noise budget” of Ciphertexts
- Risk of integer overflow from modular arithmetic (can reduce risk with encoding):
  - Integer Encoding -> Base 2 Polynomials: $26 = 2^4 + 2^3 + 2^1 = x^4 + x^3 + x^1$
  - Batch Encoding -> 2 by N/2 Matrix where N is the “Polynomial Modulus Degree parameter”

CKKS (Cheon-Kim-Kim-Song)

- Uses Additions and Multiplications on encrypted real or complex numbers
- Yields only approximate results
- Number of multiplications limited by the maximum “scale” of Ciphertexts
- No overflow risk since it doesn’t use modular arithmetic
- Input Vector is encoded into N/2 vector where N is the “Polynomial Modulus Degree” parameter

Encryption Parameters

- Polynomial Modulus Degree
  - Positive power of 2 (i.e 2048, 4096, 8192, ...)
  - High Polynomial Modulus Degree = Slower Computations
- Ciphertext Coefficient Modulus
  - Product of distinct prime numbers each up to 60 bits in size
  - Represented by a vector of prime numbers forming a Modulus Chain

    ```
    params.set_coeff_modulus(CoeffModulus::Create(poly_modulus_degree,
    {60, 40, 40, 60}));
    ```
- Plaintext Modulus (BFV Only)
  - Determines the size of the Plaintext data type and the consumption of “noise budget” in multiplications

Ciphertext Size

- The Size of a Ciphertext refers to the number of polynomials and starts at 2
  - Increases with Ciphertext multiplication
  - If M and N are the sizes of the two inputs then homomorphic multiplication of those inputs results in a Ciphertext of size: MxN -1
  - The larger the Size of a Ciphertext the more noise budget it consumes (for BFV only) and the slower our next computations will be.

- SEAL allows us to Relinearlize a ciphertext and lower its size from 3 to 2.

Ciphertext Scale (CKKS Only)

- The Scale of a Ciphertext determines the bit-precision of CKKS encoding
  - Increases with Ciphertext multiplication
  - Can corrupt the encoding if it exceeds the size of the Coefficient Modulus

- SEAL allows us to Rescale Ciphertexts
  - The number of times we are able to rescale is determined by the ‘level’: Level = Modulus Chain length - 1
  - Cannot add/sub ciphertexts with different scales
  - Cannot also add/sub/mult ciphertexts with different levels
  - We can bring down the level of a ciphertext by Modulus Switching

Implemented Functionalities using CKKS

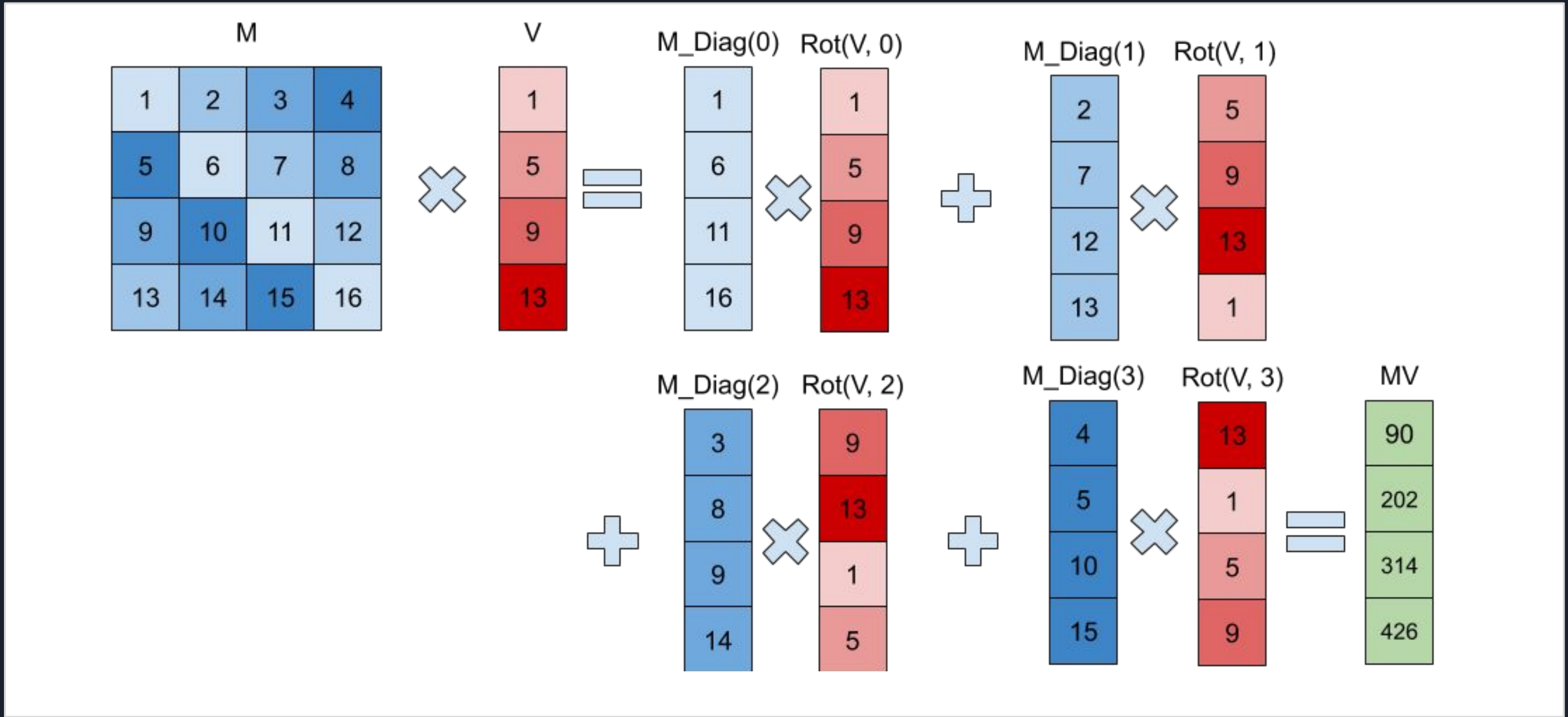
Linear Transformation (or MV Multiplication)

- Cannot select a specific element in the vector since it’s a Ciphertext
- Need to use available operations: Addition, Multiplication and Rotation

- Sum of the products of the diagonals with the rotations of the vector

$$U \bullet m = \sum_{0 \leq \ell < n} (u_\ell \odot \rho(m, \ell))$$

Linear Transformation (or MV Multiplication)



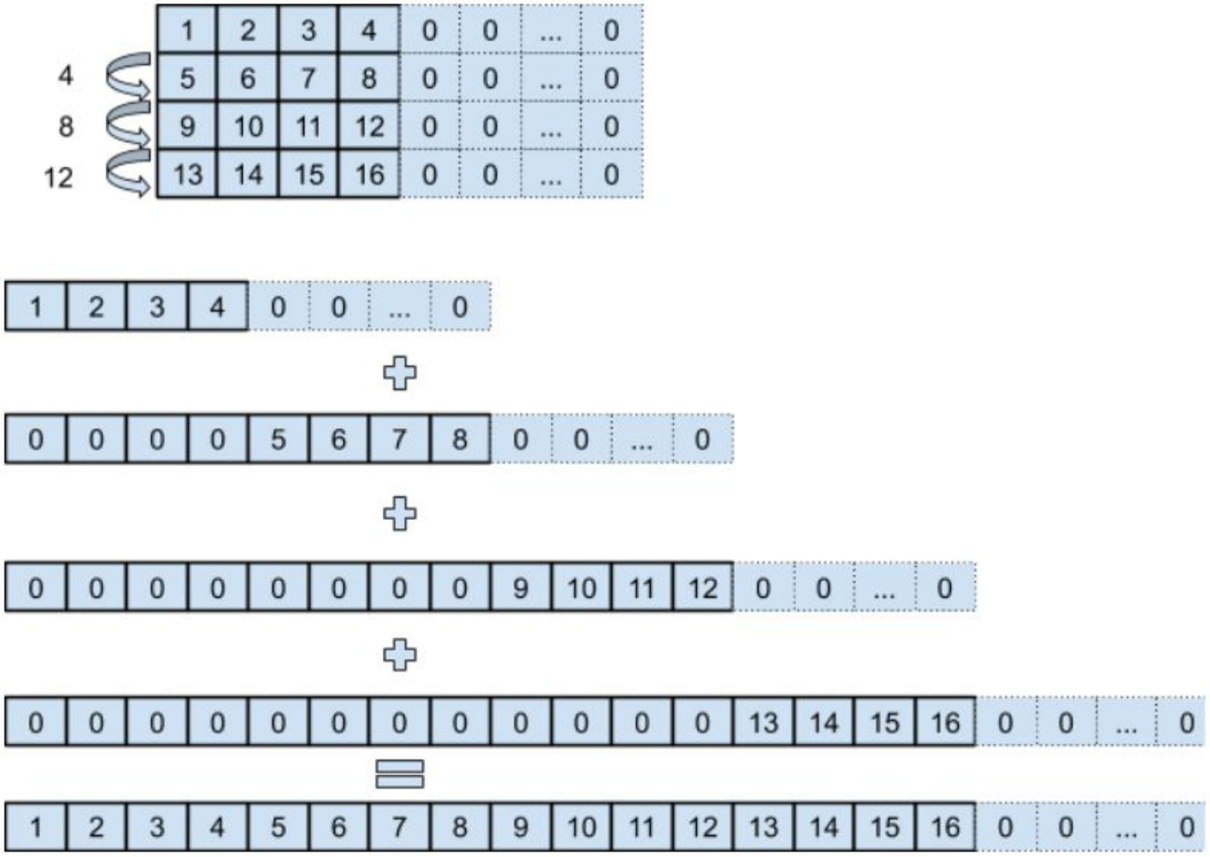
Matrix Matrix Multiplication

- Uses Linear Transformation to compute permutations of the matrices:
  - $\sigma(A)_{i,j} = A_{i,\, i+j}$
  - $\boldsymbol{\tau}(A)_{i,j} = A_{i+j,\, j}$
  - $\phi^k(A)_{i,j} = A_{i,\, j+k}$
  - $\Psi^k(A)_{i,j} = A_{i+k,\, j}$

- Requires Matrix Encoding

$$A \bullet B = \sum_{k=0}^{d-1} (\phi^k \circ \sigma(A)) \odot (\psi^k \circ \tau(B))$$

Matrix Encoding Example with 4x4 Matrix



Matrix Matrix Multiplication Example with 3x3 matrices

ctA Matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

ctB Matrix:

$$\begin{bmatrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{bmatrix}$$

STEP 1

U_sigma Matrix × ctA Matrix = ctA_result[0] Matrix

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
\times
\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{bmatrix}
=
\begin{bmatrix} 1 \\ 2 \\ 3 \\ 5 \\ 6 \\ 4 \\ 9 \\ 7 \\ 8 \end{bmatrix}$$

U_tau Matrix × ctB Matrix = ctB_result[0] Matrix

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}
\times
\begin{bmatrix} 7 \\ 8 \\ 9 \\ 4 \\ 5 \\ 6 \\ 1 \\ 2 \\ 3 \end{bmatrix}
=
\begin{bmatrix} 7 \\ 5 \\ 3 \\ 4 \\ 2 \\ 9 \\ 1 \\ 8 \\ 6 \end{bmatrix}$$

STEP 2

k = 1

V_1 Matrix × ctA_result[0] Matrix = ctA_result[1] Matrix

$$\begin{bmatrix} 0&1&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0&0&0\\ 0&0&0&1&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1&0&0 \end{bmatrix} \times \begin{bmatrix} 1\\2\\3\\5\\6\\4\\9\\7\\8 \end{bmatrix} = \begin{bmatrix} 2\\3\\1\\6\\4\\5\\7\\8\\9 \end{bmatrix}$$

W_1 Matrix × ctB_result[0] Matrix = ctB_result[1] Matrix

$$\begin{bmatrix} 0&0&0&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1\\ 1&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0 \end{bmatrix} \times \begin{bmatrix} 7\\5\\3\\4\\2\\9\\1\\8\\6 \end{bmatrix} = \begin{bmatrix} 4\\2\\9\\1\\8\\6\\7\\5\\3 \end{bmatrix}$$

k = 2

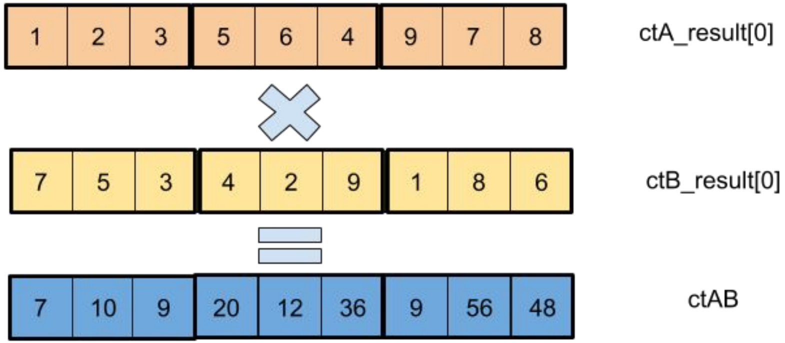
V_2 Matrix × ctA_result[0] Matrix = ctA_result[2] Matrix

$$\begin{bmatrix} 0&0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0&0\\ 0&0&0&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1&0 \end{bmatrix} \times \begin{bmatrix} 1\\2\\3\\5\\6\\4\\9\\7\\8 \end{bmatrix} = \begin{bmatrix} 3\\1\\2\\4\\5\\6\\8\\9\\7 \end{bmatrix}$$

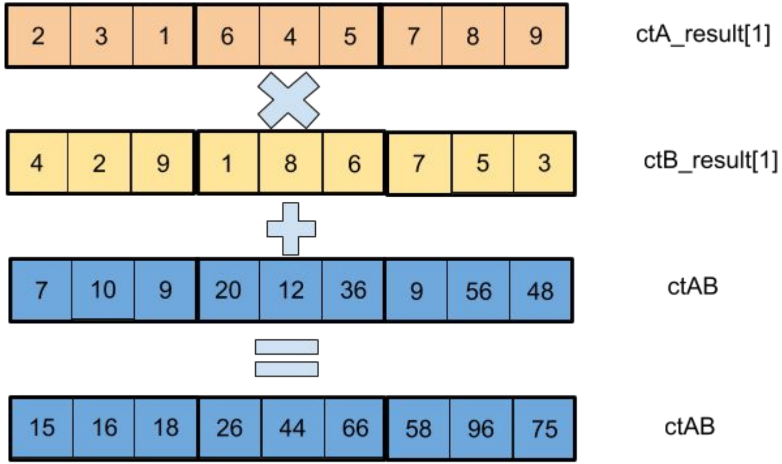
W_2 Matrix × ctB_result[0] Matrix = ctB_result[2] Matrix

$$\begin{bmatrix} 0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1\\ 1&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0&0&0 \end{bmatrix} \times \begin{bmatrix} 7\\5\\3\\4\\2\\9\\1\\8\\6 \end{bmatrix} = \begin{bmatrix} 1\\8\\6\\7\\5\\3\\4\\2\\9 \end{bmatrix}$$

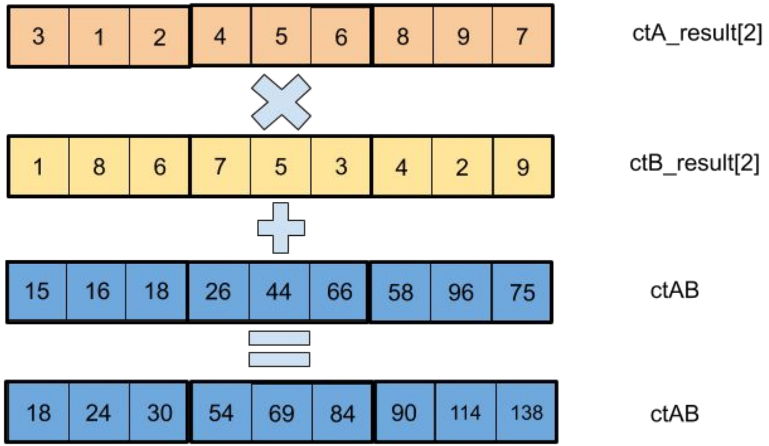
STEP 3



k = 1



k = 2

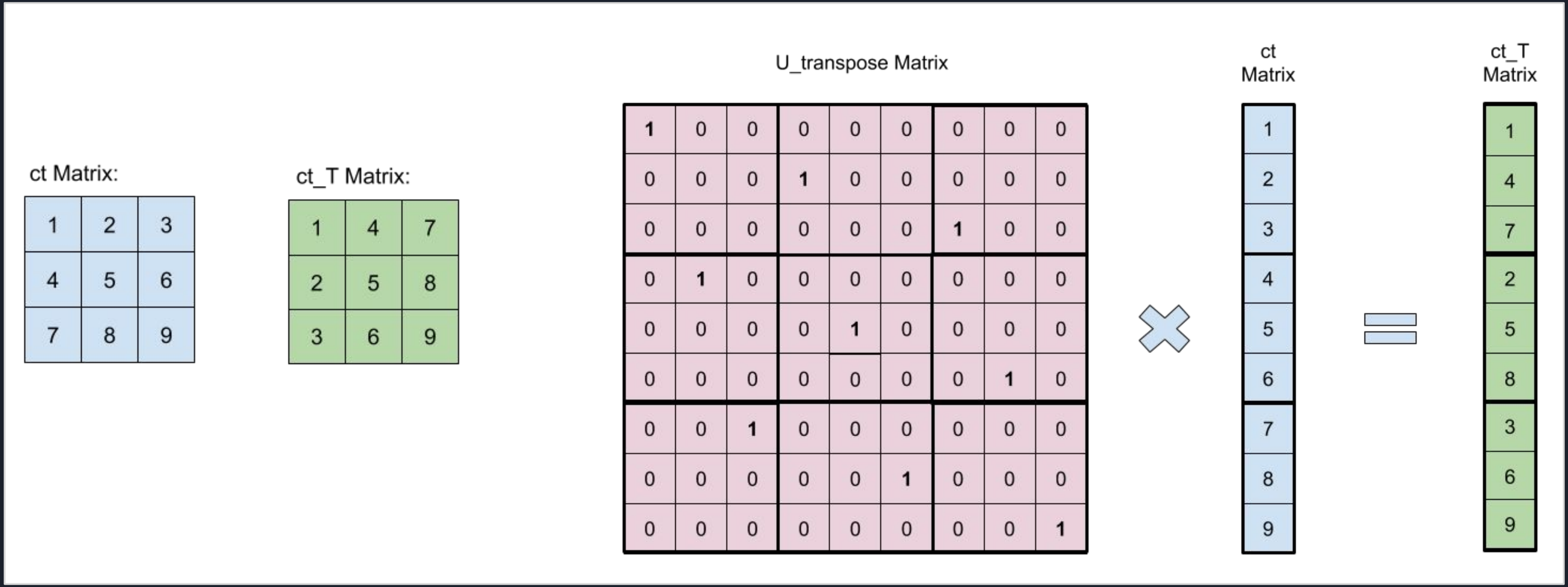


ctAB Matrix:

18	24	30
54	69	84
90	114	138

Matrix Transpose

- It's a permutation of the matrix -> Use linear transformation



Logistic Regression

Polynomial approximation of the Sigmoid function

- Cannot Divide in CKKS -> Division of real numbers is finding a certain multiple of the divisor that is either equal or rounded to the dividend, which involves multiplication and comparison.
- Sigmoid function:

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$

- Polynomial Approximations of Sigmoid:

$$\begin{aligned} g_3(x) &= 0.5 + 1.20096 \cdot (x/8) - 0.81562 \cdot (x/8)^3, \\ g_5(x) &= 0.5 + 1.53048 \cdot (x/8) - 2.3533056 \cdot (x/8)^3 \\ &\quad +1.3511295 \cdot (x/8)^5, \\ g_7(x) &= 0.5 + 1.73496 \cdot (x/8) - 4.19407 \cdot (x/8)^3 \\ &\quad +5.43402 \cdot (x/8)^5 - 2.50739 \cdot (x/8)^7. \end{aligned}$$

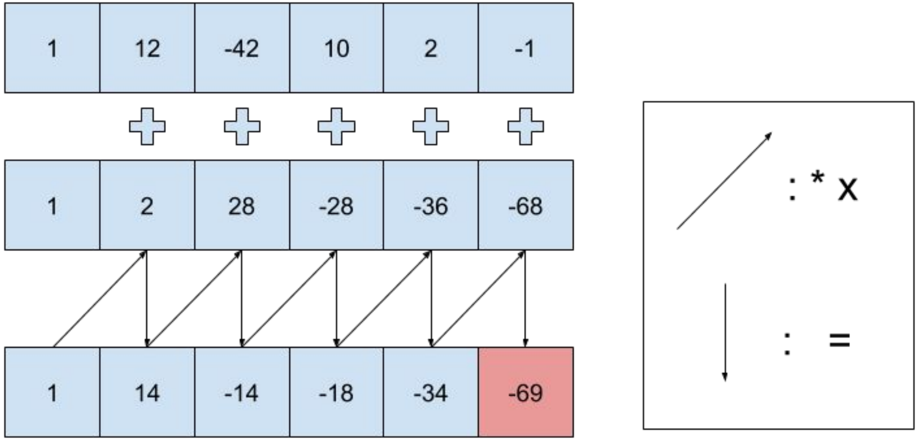
Polynomial Evaluation - Horner’s Method

- Uses a sequence of multiplication and addition to compute a polynomial.
- O(D) circuit depth: Need to rescale and relinearize D times

Consider the polynomial:

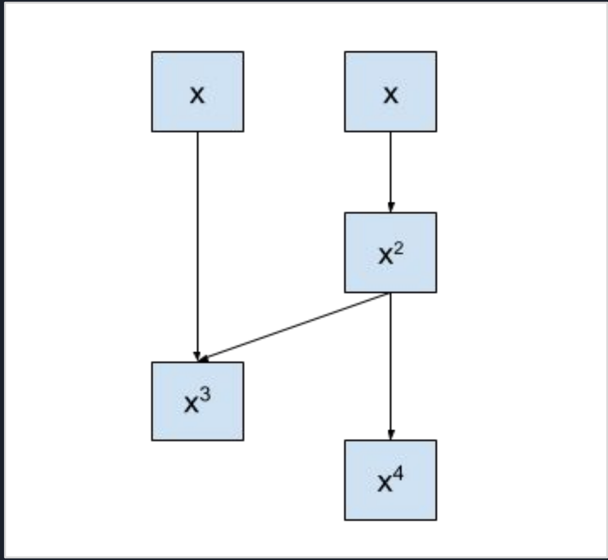
$$f(x) = x^5 + 12x^4 - 42x^3 + 10x^2 + 2x - 1$$

For x = 2:



Polynomial Evaluation - Tree Method

- O(log D) circuit depth
- Computes powers of x in a tree (figure on the right)
- Performs a dot product between the variables (1, x,... xn-1) and the coefficients (a0, a1,... an-1)
- Faster than Horner's with large polynomials

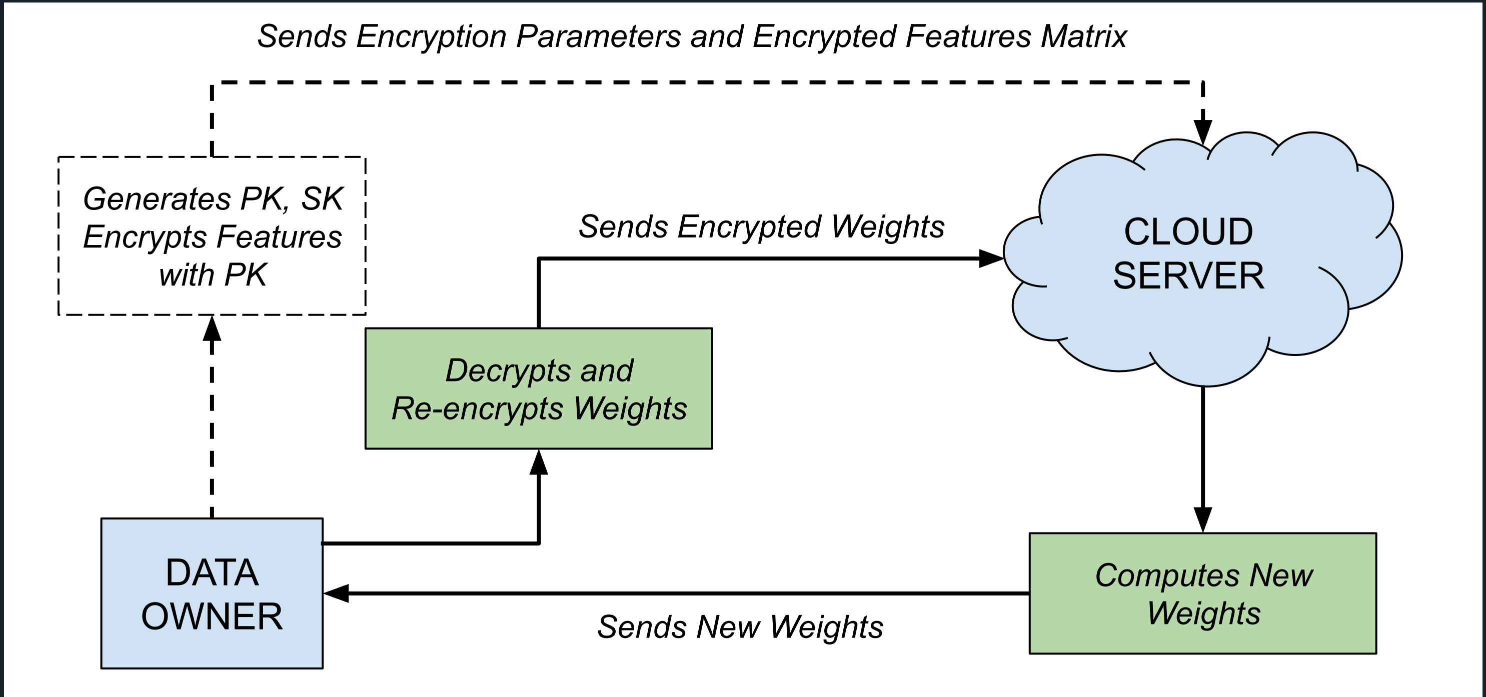


Higher Polynomial Degree = Better Approx. ?

- In theory, the higher the degree of polynomial approximation of the sigmoid function, the better the approximation
- Evaluating large polynomials harms bit-precision
- More error the larger the polynomial

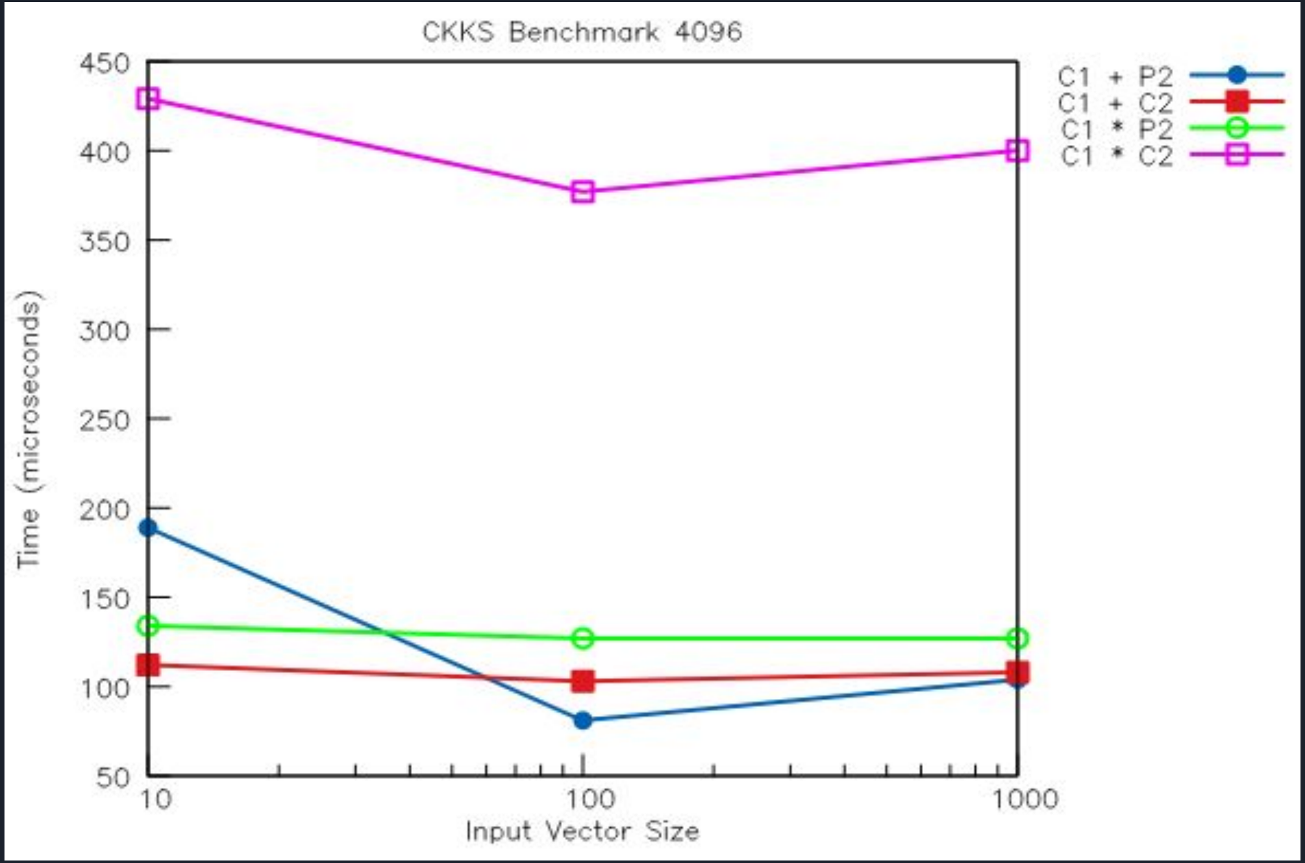
To get the best Performance and Precision -> use degree 3 polynomial with Horner's method

Training Protocol

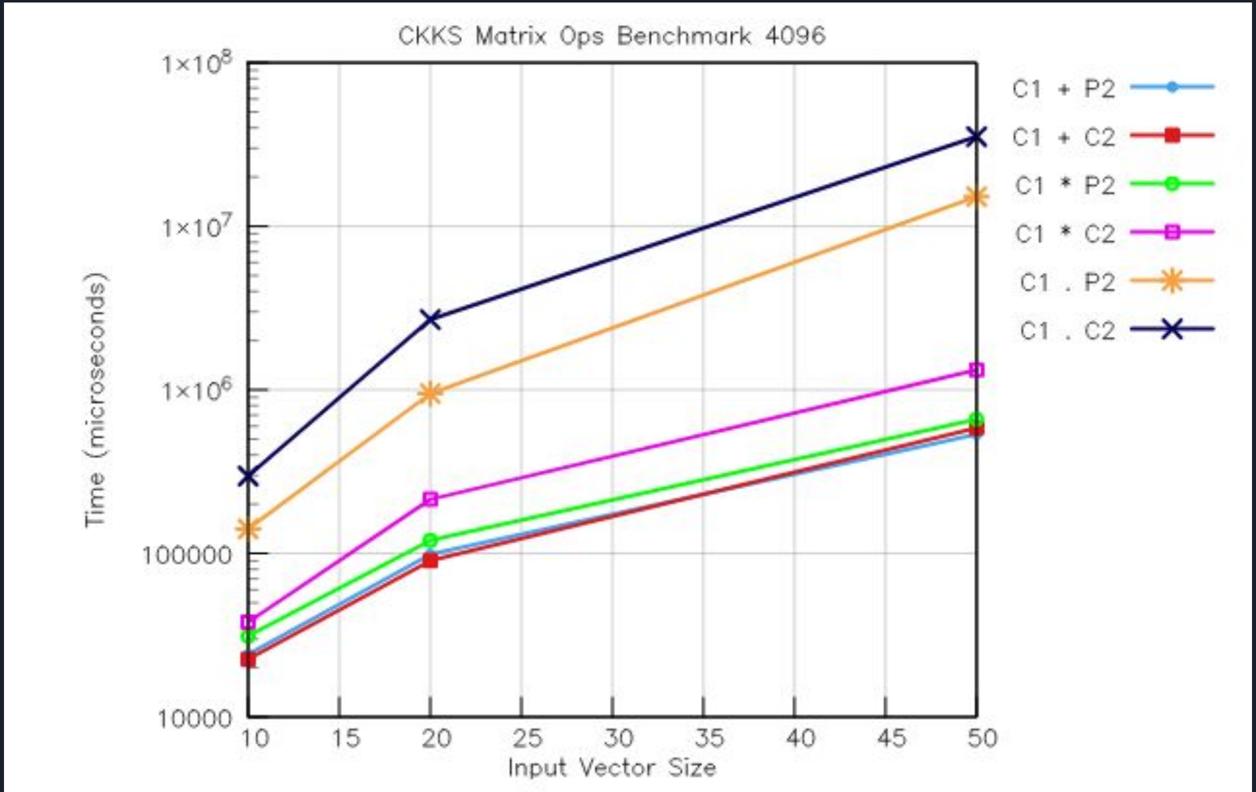


Benchmark Tests

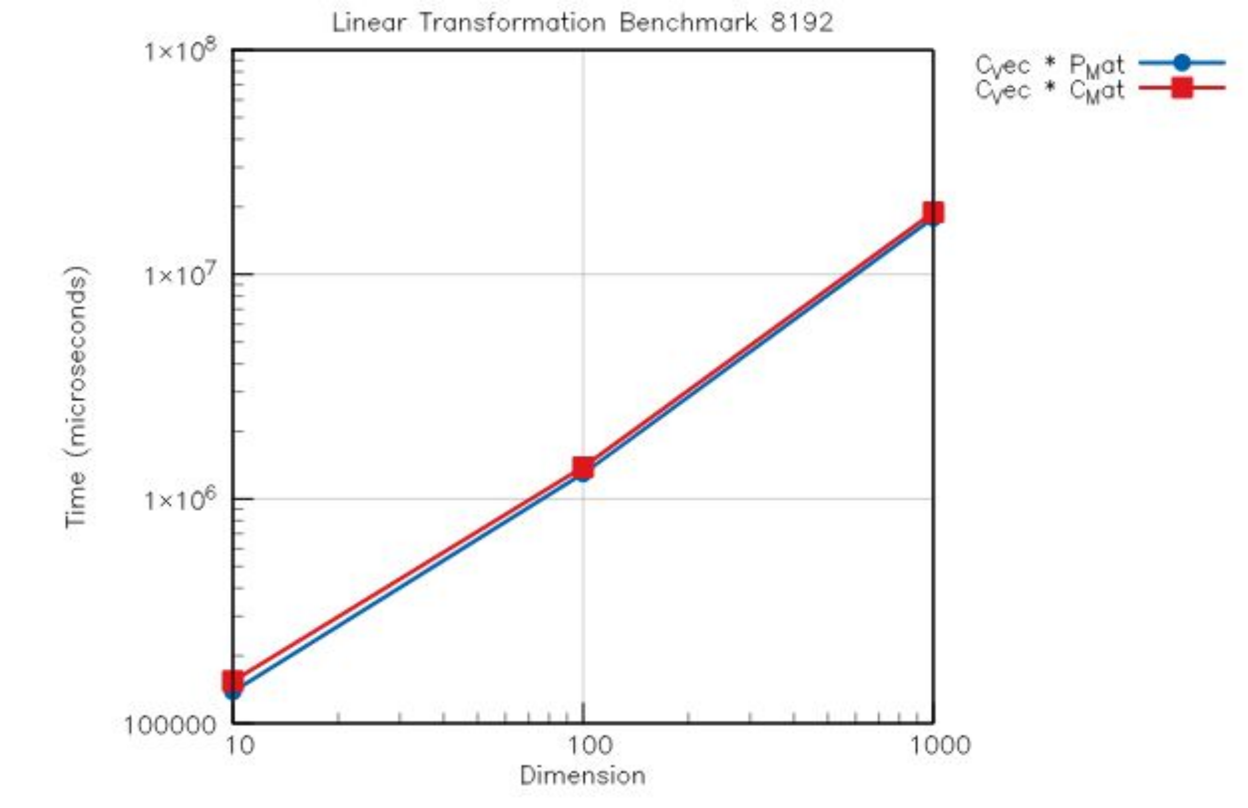
Vector Operations



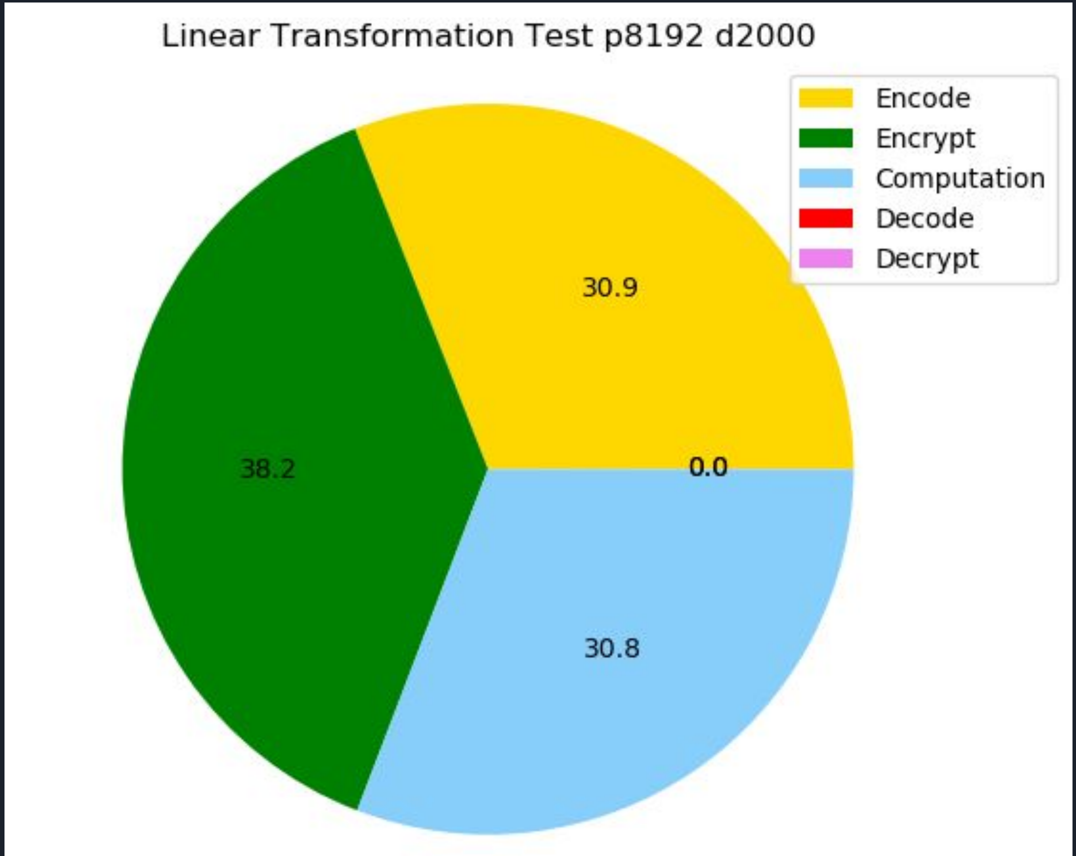
Matrix Operations (with Naive Matrix-Matrix Multiplication)



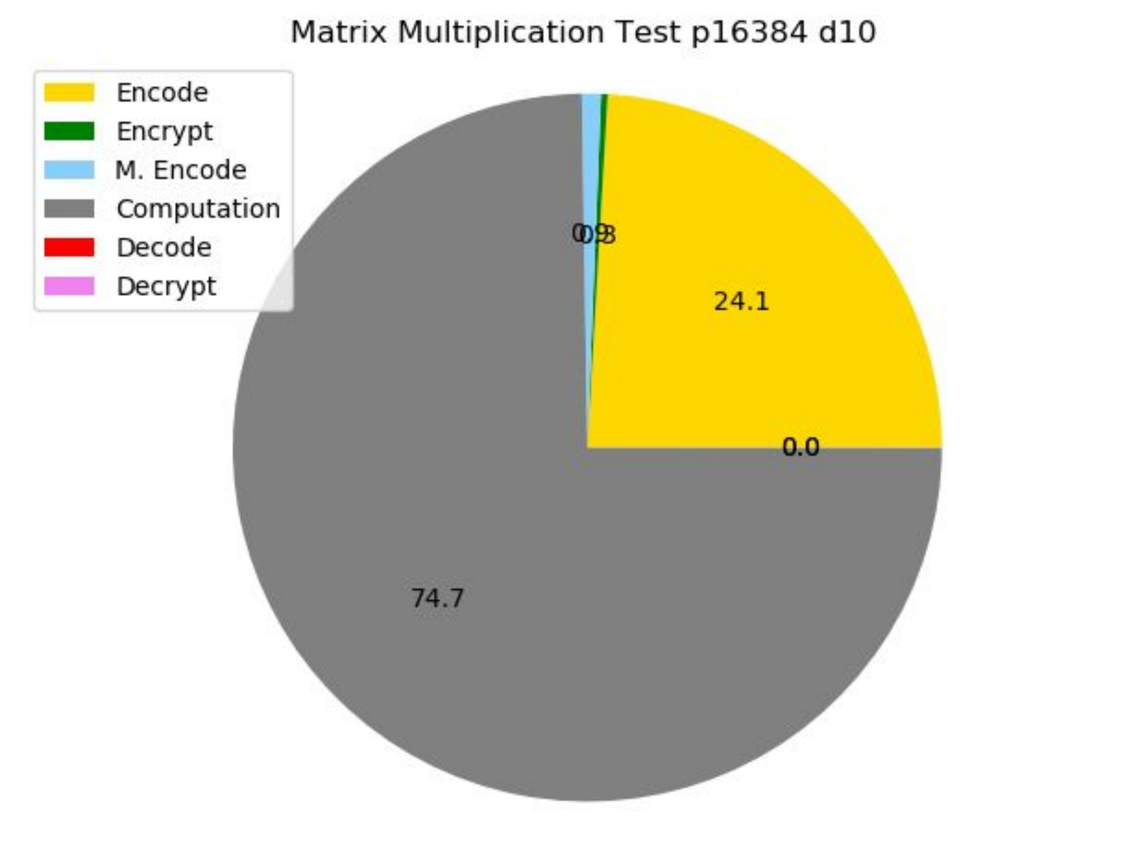
Linear Transformation



Linear Transformation Pi Chart



Matrix Multiplication Pi Chart



Possible Optimizations and Improvements

- Parallelizing on a GPU
  - Vectors are encoded in batches -> Great for SIMD
  - Data Reuse in Linear Transformation and Matrix Matrix Multiplication
  - Requires CUDA implementation for SEAL

- Fragmented Encoding and Encrypting
- Security Evaluation
  - Test for Data leakage from ciphertext

- Linear Transformation and Matrix Matrix Multiplication with Rectangular Matrices

Challenges

- Circuit Optimization
  - Reducing Multiplication Depth and allowing further computations

- Scale, Level and Bit-Precision
  - Need to keep track of ciphertext level and scale
  - Bad re-scaling can significantly harm bit-precision

- Minimal Documentation and Steep Learning Curve
  - Implementing algorithms from scratch (including logistic regression)
  - Coming up with workarounds (i.e matrix encoding, duplicate vector...)

References

- “Secure Outsourced Matrix Computation and Application to Neural Networks”

  https://eprint.iacr.org/2018/1041.pdf
- “Logistic regression model training based on the approximate homomorphic encryption”

  https://eprint.iacr.org/2018/254.pdf
- “Algorithms in HElib”

  https://shoup.net/papers/helib.pdf
- Hao Chen’s Repository for Polynomial Evaluation

  https://github.com/haochenuw/algorithms-in-SEAL/
- “Logistic Regression — ML Glossary documentation”

  https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html
- “Secure Logistic Regression Based on Homomorphic Encryption: Design and Evaluation”

  https://eprint.iacr.org/2018/074.pdf

Thank You

Email: marwan.s.nour@gmail.com

Project Repository:
https://github.com/MarwanNour/SEAL-FYP-Logistic-Regression