Solution of iDASH 2024 Track 1

SCB @ Chongqing*†

September 14, 2024

1 Workflow

DASHFormer follows a workflow similar to Transformer, consisting of an input layer, encoding layer, attention layer, feedforward layer, pooling layer, and output layer. In this document, all vectors are in row.

1.1 Input Layer

- For the Track 1 of iDash 2024, the input contains $1,000$ protein sequences, each of length 50, with the alphabet taken from a set of 25 letters {A, B, ..., Y}. Namely, each sequence is a vector in {A, B, ..., Y}^{50}.
- But for DASHFormer, each input is a vector $\boldsymbol{a} \in$ {A, B, ..., Y}^{50}. However, before starting DASHFormer, we should first tokenize each sequences.
- So the real input for DASHFormer is $\boldsymbol{x} \in \mathbb{Z}^{50}$, which is tokenized by the file named 'dashformer_tokenizer.json'.
- Using one-hot encoding, one can get $\boldsymbol{X} \in \{0,1\}^{50\times 25}$.

1.2 Encodign Layer

1.2.1 Embedding

- Embedding each token into a 128-dimensional vector. Since the alphabet contains only 25 letters, the size of this embedding matrix is 25×128. This matrix can be found in the file named 'embedding_Embedding_weights.txt'.
- After embedding, each $\boldsymbol{X} \in \{0,1\}^{50\times 25}$ will be converted into a matrix $\boldsymbol{X} = \boldsymbol{X}\cdot\boldsymbol{W}_e \in \mathbb{R}^{50\times 128}$, where $\boldsymbol{W}_e \in \mathbb{R}^{25\times 128}$ is the matrix of embedding weights.

*CIGIT, CAS: Jingwei Chen (chenjingwei@cigit.ac.cn), Linhan Yang, Chen Yang, Shuai Wang, Rui Li, Weijie Miao, and Wenyuan Wu (wuwenyuan@cigit.ac.cn).
†Sansure Biotech Inc.: Li Yang and Jia Liu.

1.2.2 Positional Encoding

- Each token in each sequence has its own position, which can be encoded as a vector of dimension 128. The encoding method is decided by the file named 'positional_encoding_Lookup.txt'.
- The positional encoding is $\boldsymbol{P} \in \mathbb{R}^{50\times 128}$.

1.2.3 Output of the Encoding Layer

- The output of the encoding layer is $\boldsymbol{X} := \boldsymbol{X} + \boldsymbol{P} \in \mathbb{R}^{50\times 128}$.
- Save $\boldsymbol{Y} := \boldsymbol{X}$ for future use (Add and LayerNorm).

1.3 Attention Layer

1.3.1 Attention

- The number of heads: $H = 4$. Hence, in what follows, $h = 0, 1, 2, 3$.
- Head weights and bias: $\boldsymbol{W}_Q^{(h)}, \boldsymbol{W}_K^{(h)}, \boldsymbol{W}_V^{(h)} \in \mathbb{R}^{128\times 32}$ and $\boldsymbol{b}_Q^{(h)}, \boldsymbol{b}_K^{(h)}, \boldsymbol{b}_V^{(h)} \in \mathbb{R}^{32}$.
- Queries: $\boldsymbol{Q}^{(h)} = \boldsymbol{X}\boldsymbol{W}_Q^{(h)} + \boldsymbol{1}^{\mathrm{T}} \cdot \boldsymbol{b}_Q^{(h)} \in \mathbb{R}^{50\times 32}$. Alternatively, $\boldsymbol{Q}^{'(h)} = \boldsymbol{X}' \cdot \boldsymbol{W}_Q^{'(h)}$ with $\boldsymbol{X}' = (\boldsymbol{X}, \boldsymbol{1}^{\mathrm{T}})\binom{\boldsymbol{W}}{\boldsymbol{b}_Q^{(h)}}$.
- Keys: $\boldsymbol{K}^{(h)} = \boldsymbol{X}\boldsymbol{W}_K^{(h)} + \boldsymbol{1}^{\mathrm{T}} \cdot \boldsymbol{b}_K^{(h)} \in \mathbb{R}^{50\times 32}$.
- Values: $\boldsymbol{V}^{(h)} = \boldsymbol{X}\boldsymbol{W}_V^{(h)} + \boldsymbol{1}^{\mathrm{T}} \cdot \boldsymbol{b}_V^{(h)} \in \mathbb{R}^{50\times 32}$.
- Attention: $\boldsymbol{X}^{(h)} = \mathrm{softmax}(\boldsymbol{Q}^{(h)} \cdot \boldsymbol{K}^{(h)\mathrm{T}}/\sqrt{d})\boldsymbol{V}^{(h)}$, where $d = 32$, i.e., the number of columns of $\boldsymbol{Q}$, $\boldsymbol{K}$ and $\boldsymbol{V}$.
- Concatenate: $\boldsymbol{X} = (\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \boldsymbol{X}^{(3)}, \boldsymbol{X}^{(4)}) \in \mathbb{R}^{50\times 128}$.

1.3.2 Combining Head (A Linear Layer)

- Head combining weight and bias: $\boldsymbol{W}_c \in \mathbb{R}^{128\times 128}$ and $\boldsymbol{b}_c \in \mathbb{R}^{128}$.
- Combining: $\boldsymbol{X} = \boldsymbol{X}\boldsymbol{W}_c + \boldsymbol{1}^{\mathrm{T}} \cdot \boldsymbol{b}_c \in \mathbb{R}^{50\times 128}$.

1.3.3 Add and LayerNorm-1

- Add: $\boldsymbol{X} := \boldsymbol{X} + \boldsymbol{Y}$.
- LayerNorm-1: $\boldsymbol{X} := \mathrm{LayerNorm}_1(\boldsymbol{X})$, where the parameters can be found in the file named 'transformer_block_LayerNorm1_weights.txt'.
- Update $\boldsymbol{Y} := \boldsymbol{X}$.

1.4 Feed Forward with ReLU

- $\boldsymbol{X} := \boldsymbol{X}\boldsymbol{W_1} + \mathbf{1}^{\mathrm{T}} \cdot \boldsymbol{b}_1$ with $\boldsymbol{W}_1 \in \mathbb{R}^{128\times256}$ and $\boldsymbol{b}_1 \in \mathbb{R}^{256}$.
- $\boldsymbol{X} := \mathrm{ReLU}(\boldsymbol{X})$.
- $\boldsymbol{X} := \boldsymbol{X}\boldsymbol{W}_2 + \mathbf{1}^{\mathrm{T}} \cdot \boldsymbol{b}_2$ with $\boldsymbol{W}_1 \in \mathbb{R}^{256\times128}$ and $\boldsymbol{b}_1 \in \mathbb{R}^{128}$, where the parameters can be found in the file named 'transformer_block_FFN_weights.txt'.

1.4.1 Add and LayerNorm-2

- Add: $\boldsymbol{X} := \boldsymbol{X} + \boldsymbol{Y}$.
- LayerNorm-2: $\boldsymbol{X} := \mathrm{LayerNorm}_2(\boldsymbol{X})$, where the parameters can be found in the file named 'transformer_block_LayerNorm2_weights.txt'.

1.5 Average Pooling

- Given $\boldsymbol{X} \in \mathbb{R}^{50\times128}$, output $\boldsymbol{x} = \mathbf{1} \cdot \boldsymbol{X} \in \mathbb{R}^{128}$.

1.6 Dense Classification

- Taking $\boldsymbol{x} \in \mathbb{R}^{128}$ as input, output $\boldsymbol{y} = \boldsymbol{x}\boldsymbol{W}_d + \boldsymbol{b}_d$, where $\boldsymbol{W}_d \in \mathbb{R}^{128\times25}$ and $\boldsymbol{b}_d \in \mathbb{R}^{25}$.

2 Implementation

We strictly follow the workflow introduced in the previous section for ciphertext computation. Specifically, non-polynomial functions are handled using approximation methods. At the same time, we fully leverage the SIMD support provided by the CKKS scheme [2] to adopt flexible data packing methods. This allows us to use tensor-matrix computations to complete the prediction of 100 samples simultaneously, and successfully applies the Baby-Step Giant-Step (BSGS) method [3] to accelerate the matrix-matrix multiplications.

2.1 Approximate Non-polynomial Functions

2.1.1 Softmax

The Softmax of a vector $\boldsymbol{x} = (x_i)_i$ is defined as

$$\mathrm{softmax}(\boldsymbol{x}) = \left(\frac{\exp(x_i)}{\sum_i \exp(x_i)}\right)_i.$$

Instead of high-degree polynomial approximations of the exponential function in softmax, we use the square softmax (sqmax) in the following form:

$$\mathrm{sqmax}(\boldsymbol{x}) = \left(\frac{(x_i + c)^2}{\sum_i (x_i + c)^2}\right)_i.$$

Furthermore, we use a constant as the denominator to avoid the cipheretext division, i.e.,

$$\mathrm{sqmax}(\boldsymbol{x}) = \left(\frac{(x_i + c)^2}{\delta}\right)_i.$$

Since there are four heads in DASHformer, we choose different c and δ for each head. These coefficients are stored in "`config/config.go`".

2.1.2 ReLU

For $x \in \mathbb{R}$, $\mathrm{ReLU}(x) = \max(0, x)$. Currently, we use a polynomial of degree 6 to approximate the ReLU function:

$$f(x) = a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0,$$

where the coefficients are stored in "`config/config.go`".

2.1.3 LayerNorm

Given a vector $\boldsymbol{x} = (x_i)_i$,

$$\mathrm{LayerNorm}(\boldsymbol{x}) = \left(\gamma \cdot \frac{x_i - \mu}{\sqrt{\sigma^2}} + \beta\right)_i$$

with μ and σ^2 the mean and variance of $\boldsymbol{x}$. The LayerNorm function applies to all the rows of the matrix $\boldsymbol{X} \in \mathbb{R}^{50\times 128}$.

Instead of computing $\frac{1}{\sigma^2}$ in ciphertext, we compute it in plaintext with all 1000 samples to get a 50-dimensional vector for the inverse of the standard deviation so that

$$\mathrm{LN}(\boldsymbol{x}) = (\gamma \cdot \tilde{\sigma} \cdot (x_i - \mu) + \beta)_i,$$

where $\tilde{\sigma}$ is the learned standard deviation from the given 1000 samples. If we use the matrix form, we have

$$\begin{aligned}
\mathrm{LN}(\boldsymbol{X}) &= \mathrm{diag}(\frac{1}{\sqrt{\sigma_i^2}})(\boldsymbol{X} - \boldsymbol{U})\mathrm{diag}(\gamma_i) + \mathbf{1}^T \cdot \boldsymbol{\beta} \\
&= \mathrm{diag}(\frac{1}{d\sqrt{\sigma_i^2}})(d\boldsymbol{X} - \boldsymbol{X} \cdot \boldsymbol{E})\mathrm{diag}(\gamma_i) + \mathbf{1}^T \cdot \boldsymbol{\beta} \\
&=: \boldsymbol{\Sigma}^{-1} \cdot \boldsymbol{X} \cdot \boldsymbol{\Gamma} + \mathbf{1}^T \cdot \boldsymbol{\beta}.
\end{aligned}$$

Here d is the number of columns of $\boldsymbol{X}$, $\boldsymbol{\Sigma}^{-1} = \mathrm{diag}(\frac{1}{d\sqrt{\sigma_i^2}})$ and

$$\boldsymbol{\Gamma} = (d\boldsymbol{I} - \boldsymbol{E}) \cdot \mathrm{diag}(\gamma_i),$$

where $\boldsymbol{I}$ is the identity matrix and all entries of $\boldsymbol{E}$ are 1. The parameters $\boldsymbol{\Sigma}^{-1}$, $\boldsymbol{\Gamma}$ and $\boldsymbol{\beta}$ are stored in "`data/layerNorm1_Reciprocal_SqrtVariance.txt`" and "`data/layerNorm2_Reciprocal_SqrtVariance.txt`".

2.2 Multiplicative Depth

Let $\boldsymbol{X}_0 \in \{0,1\}^{50\times 25}$ be the one-hot encoding for an input $\boldsymbol{x} \in \mathbb{Z}^{50}$. Then

$$\begin{aligned}
&\text{DASHFormer}(\boldsymbol{X}) \\
=&\mathbf{1}\cdot\boldsymbol{X}\cdot\boldsymbol{W}_d+\boldsymbol{b}_d \\
=&\mathbf{1}\cdot\text{LN}_2(\boldsymbol{X}+\boldsymbol{Y}_2)\cdot\boldsymbol{W_d}+\boldsymbol{b}_d \\
=&\mathbf{1}\cdot(\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{X}+\boldsymbol{Y}_2)\Gamma_2+\mathbf{1}^T\boldsymbol{\beta}_2)\cdot\boldsymbol{W_d}+\boldsymbol{b}_d \\
=&\mathbf{1}\cdot\boldsymbol{\Sigma}_2^{-1}\boldsymbol{X}\Gamma_2\cdot\boldsymbol{W_d}+\mathbf{1}\cdot\boldsymbol{\Sigma}_2^{-1}\boldsymbol{Y}_2\Gamma_2\cdot\boldsymbol{W_d}+\mathbf{1}\cdot\mathbf{1}^T\boldsymbol{\beta}_2\cdot\boldsymbol{W_d}+\boldsymbol{b}_d. \qquad (1)
\end{aligned}$$

In Eq. (1), we have

$$\begin{aligned}
\boldsymbol{Y}_2=&\text{LN}_1(\boldsymbol{X}+\boldsymbol{Y}_1) \\
=&\boldsymbol{\Sigma}_1^{-1}\boldsymbol{X}\boldsymbol{\Gamma_1}+\boldsymbol{\Sigma}_1^{-1}\boldsymbol{Y}_1\boldsymbol{\Gamma_1}+\mathbf{1}^T\cdot\boldsymbol{\beta}_1 \\
=&\boldsymbol{\Sigma}_1^{-1}\boldsymbol{X}\boldsymbol{\Gamma_1}+\boldsymbol{\Sigma}_1^{-1}\boldsymbol{X}_0\boldsymbol{W}_e\boldsymbol{\Gamma_1}+\boldsymbol{\Sigma}_1^{-1}\boldsymbol{P}\boldsymbol{\Gamma_1}+\mathbf{1}^T\cdot\boldsymbol{\beta}_1 \qquad (2)
\end{aligned}$$

where $\boldsymbol{Y}_1=\boldsymbol{X}_0\boldsymbol{W}_e+\boldsymbol{P}$. In Eq. (2),

$$\begin{aligned}
\boldsymbol{\Sigma}_1^{-1}\boldsymbol{X}\boldsymbol{\Gamma_1}=&\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{X}\boldsymbol{W}_c+\mathbf{1}^T\cdot\boldsymbol{b}_c)\boldsymbol{\Gamma_1} \\
=&\boldsymbol{\Sigma}_1^{-1}\boldsymbol{X}\boldsymbol{W}_c\boldsymbol{\Gamma_1}+\boldsymbol{\Sigma}_1^{-1}\cdot\mathbf{1}^T\cdot\boldsymbol{b}_c\boldsymbol{\Gamma_1} \\
=&\boldsymbol{\Sigma}_1^{-1}\cdot\text{sqmax}(\boldsymbol{Q}\boldsymbol{K}^T/\sqrt{d})\boldsymbol{V}\boldsymbol{W}_c\boldsymbol{\Gamma_1}+\boldsymbol{\Sigma}_1^{-1}\cdot\mathbf{1}^T\cdot\boldsymbol{b}_c\boldsymbol{\Gamma_1},
\end{aligned}$$

where

$$\boldsymbol{Q}=\boldsymbol{X}\boldsymbol{W}_Q+\mathbf{1}^T\cdot\boldsymbol{b}_Q=\boldsymbol{X}_0\boldsymbol{W}_e\boldsymbol{W}_Q+\boldsymbol{P}\boldsymbol{W}_Q+\mathbf{1}^T\cdot\boldsymbol{b}_Q,$$

and similarly for $\boldsymbol{K}$ and $\boldsymbol{V}$.

In Eq. (1),

$$\begin{aligned}
&\mathbf{1}\cdot\boldsymbol{\Sigma}_2^{-1}\boldsymbol{X}\Gamma_2\cdot\boldsymbol{W_d} \\
=&\mathbf{1}\cdot\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{X}\boldsymbol{W}_2+\mathbf{1}^T\cdot\boldsymbol{b}_2)\Gamma_2\cdot\boldsymbol{W_d} \\
=&\mathbf{1}\cdot\boldsymbol{\Sigma}_2^{-1}\boldsymbol{X}\boldsymbol{W}_2\cdot\Gamma_2\cdot\boldsymbol{W_d}+\mathbf{1}\cdot\boldsymbol{\Sigma}_2^{-1}\mathbf{1}^T\cdot\boldsymbol{b}_2\cdot\Gamma_2\cdot\boldsymbol{W_d} \qquad (3)
\end{aligned}$$

In Eq. (3),

$$\mathbf{1}\cdot\boldsymbol{\Sigma}_2^{-1}\boldsymbol{X}\boldsymbol{W}_2\cdot\Gamma_2\cdot\boldsymbol{W_d}=\mathbf{1}\cdot\boldsymbol{\Sigma}_2^{-1}\text{ReLU}(\boldsymbol{X}\boldsymbol{W}_1+\mathbf{1}^T\cdot\boldsymbol{b}_1)\boldsymbol{W}_2\cdot\Gamma_2\cdot\boldsymbol{W_d} \qquad (4)$$

In Eq. (4),

$$\text{ReLU}(\boldsymbol{X}\boldsymbol{W}_1+\mathbf{1}^T\cdot\boldsymbol{b}_1)=\text{ReLU}(\boldsymbol{Y}_2\boldsymbol{W}_1+\mathbf{1}^T\cdot\boldsymbol{b}_1)$$

with $\boldsymbol{Y}_2$ as computed in Eq. (2). So

$$\boldsymbol{Y}_2\boldsymbol{W}_1=\boldsymbol{\Sigma}_1^{-1}\boldsymbol{X}\boldsymbol{\Gamma_1}\boldsymbol{W}_1+\boldsymbol{\Sigma}_1^{-1}\boldsymbol{X}_0\boldsymbol{W}_e\boldsymbol{\Gamma_1}\boldsymbol{W}_1+\boldsymbol{\Sigma}_1^{-1}\boldsymbol{P}\boldsymbol{\Gamma_1}\boldsymbol{W}_1+\mathbf{1}^T\cdot\boldsymbol{\beta}_1\boldsymbol{W}_1.$$

We now can give the multiplicative depth cost of our homomorphic evaluation of the DASHFormer as in Table 1. It shows that one can evaluate the whole circuit with at most 10 multiplicative depths (including both ciphertext-ciphertext and plaintext-ciphertext multiplication), which allows us to choose $\log N = 14$ to achieve 128-bit security [1].

Table 1: Multiplicative depth for the whole DASHFormer

Task	Before ReLU	ReLU	After ReLU	Total
Depth	6	3	1	10

2.3 Numeric Error-Control

To ensure that the ciphertext computation results (at the lowest level) still can be decrypted correctly, we scale the coefficients used during the last step in Section 1.6. Namely, we multiply a constant $c = 1/2649.372705$ with $\boldsymbol{W}_d$ and $\boldsymbol{b}_d$, respectively. Correspondingly, to make the decrypted results consistent with the plaintext results, we also multiplied the decrypted results by $1/c$.

2.4 Encrypted Tensor Product

For each sample $\boldsymbol{x}$ to be classified, the inference of DASHFormer is as stated in the previous subsection. However, we have to deal with 100 samples during the evaluation phase of Track 1. Instead of a 50×25 matrix $\boldsymbol{X}_0$ as input, we have to compute with a tensor $\boldsymbol{X}_0 \in \mathbb{R}^{100\times 50\times 25}$. When packing and encrypting data for the CKKS scheme, we treat each 100×50 slice of the tensor as a ciphertext. This makes us choose $\log N = 14$ for CKKS without using the conjugate invariant, i.e., the number of slots is $8192 > 5000$. In this setting, our solution can evaluate at most 163 samples.

2.5 Encrypted Matrix Multiplication

The most time-consuming part of the entire process is to compute $\mathrm{sqmax}(\boldsymbol{Q}\boldsymbol{K}^T)\boldsymbol{V}$. Since sqmax is a second-order approximation of softmax, this step essentially involves the sequential multiplication of three ciphertext matrices. We encode the matrices column by column. Thus, after $\boldsymbol{C} = \boldsymbol{Q}\boldsymbol{K}^T$ is computed, the diagonal encoding of $\boldsymbol{C}$ is obtained. Then, when computing $\boldsymbol{C}\cdot\boldsymbol{V}$, the computation is performed by $\boldsymbol{C}\boldsymbol{v}_i$ using the Halevi-Shoup algorithm [3], where $\boldsymbol{v}_i$ is the ith column of $\boldsymbol{V}$. Ultimately, the ciphertexts of the columns of the resulting matrix $\boldsymbol{R} = \mathrm{sqmax}(\boldsymbol{Q}\boldsymbol{K}^T)\boldsymbol{V}$ are obtained, allowing for subsequent computations.

Double-BSGS. Through some non-trivial packing techniques, we can fully utilize the so-called baby-step-giant-step (BSGS) technique *twice* during computation of $\boldsymbol{R}$ to reduce the number of required rotations. We call this technique *double-BSGS*. Assuming $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ are all $n \times m$ matrices, our method reduces the required number of ciphertext rotations from $m^2 \cdot n$ to $m(m+1)(\ell + k)$ with $n \le \ell \cdot k$. In our implementation of DASHFormer, $(n, m, p, k, \ell) = (50, 128, 25, 8, 7)$. Particularly, we reduce the required number of rotations from $819,200$ to $247,680$, saving $571,520$ (about 70%) rotations.

3 Performance

We use the CKKS scheme in Lattigo (v5.0.2) [4] to implement our solution. The parameters are chosen as the following:

- $\log N = 14$, where N is the polynomial degree of the ring,
- $\log Q = 38 + 33 \cdot 10 = 368$, where Q is the largest ciphertext modulus,
- $\log P = 36 \cdot 2 = 72$, where P is the special prime for key-switching,
- $\log \Delta = 33$, where Δ is the scaling factor.

According to the latest security guidelines for homomorphic encryption [1], the above setup can achieve 128-bit security.

In addition, we use *four* threads to parallelize our implementation. On a node (Intel Xeon Gold 6248R 3.00 GHz and 128 GB memory) of a cluster with Centos 8 system (Platform 1) and a PC (i9-12900K 3.2 GHz and 32 GB memory) with Ubuntu 22.04 (Platform 2), we tested our implementation. The maximum memory used during our execution is about 29 GB. The timings are obtained with limited 4 cores and listed in Table 2.

Table 2: Timings in second

Task	Before ReLU	ReLU	After ReLU	Total (with i/o)
Platform 1	314.4	10.5	5.8	333.9
Platform 2	209.4	6.2	3.9	222.9

We execute our implementation several times for randomly chosen 100 samples; the micro-AUC ranges from 0.91 to 0.97, as shown in Figure 1.

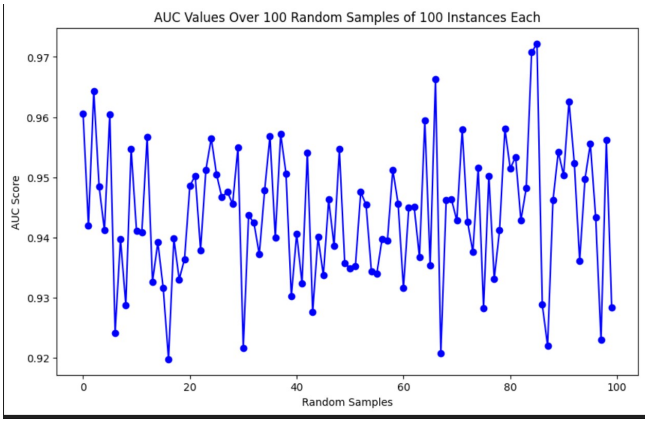

Figure 1: Micro-AUC for randomly chosen Samples

References

[1] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018. `https://eprint.iacr.org/2019/939.pdf`.

[2] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In T. Takagi and T. Peyrin, editors, *Proceedings of ASIACRYPT 2017 – 23rd International Conference on the Theory and Applications of Cryptology and Information Security (December 3-7, 2017, Hong Kong, China), Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, Heidelberg, 2017. `https://doi.org/10.1007/978-3-319-70694-8_15`.

[3] S. Halevi and V. Shoup. Algorithms in HElib. In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology – CRYPTO 2014 (Santa Barbara, USA, August 17-21, 2014)*, volume 8616 of *Lecture Notes in Computer Science*, pages 554–571. Springer, Heidelberg, 2014. `https://doi.org/10.1007/978-3-662-44371-2_31`.

[4] Tune-Insight. Lattigo (v5.0.2), Accessed in July, 2024. `https://github.com/tuneinsight/lattigo/releases/tag/v5.0.2`.