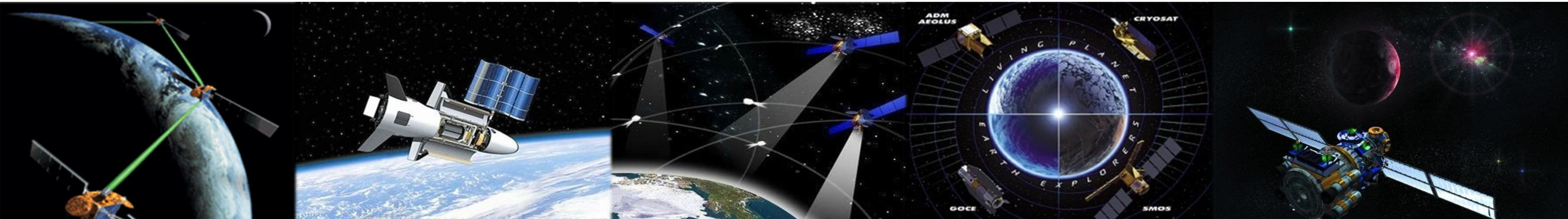
北京航空航天大學
BEIHANG UNIVERSITY

HEIR: A Unified Representation for Cross-Scheme Compilation of Fully Homomorphic Computation

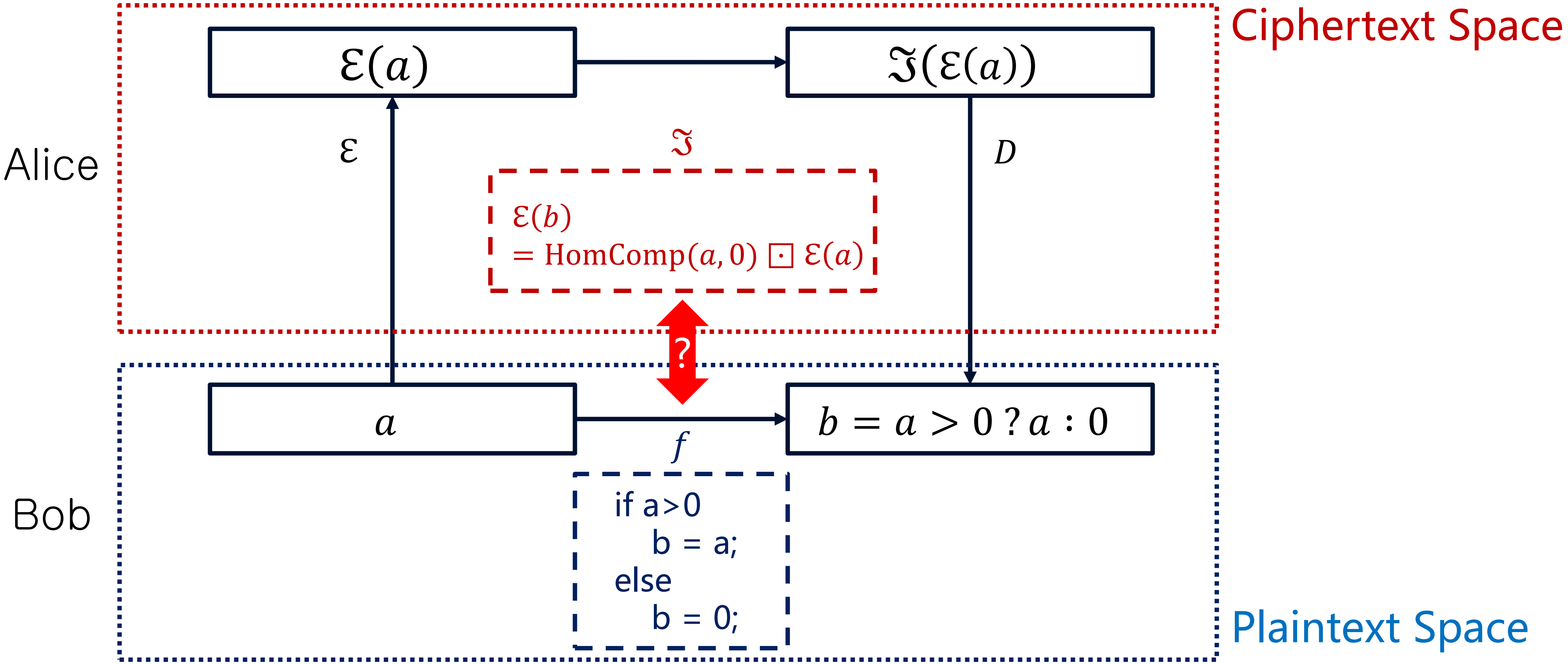
Song Bian, **Zian Zhao**, Zhou Zhang, Ran Mao,
Kohei Suenaga, Yier Jin, Zhenyu Guan, Jianwei Liu

Beihang University, Kyoto University, University of Science and Technology of China



Fully Homomorphic Encryption

Q: How to automate such design?



FHE Compiler Overview

User Program | **FHE Compiler** | FHE Library

Standard C

```
int foo(int [] x, int[] y){
  int[] r;
  for(i = 0;i < 4; ++i){
    r[i] = x[i] * y[i]
  }
  return r;
}
```

Compiler's IR

```
 func foo(%arg0:RLWECipher, %arg1:RLWECipher){
  RLWECipher v0 = heir.define() : RLWECipher
  RLWECipher v1 = heir.rlwemult(%arg0, %arg1) :
RLWECipher
  return v1;
}
```

SEAL Library

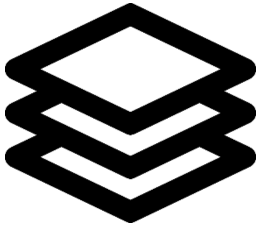
```
Ciphertext foo(Ciphertext x,
               Ciphertext y){
  Ciphertext r;
  evaluator.multiply(x, y, r);
  return r;
}
```

Why do we need FHE compilers?

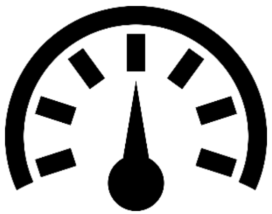
FHE programming Paradigm

No If/Else

No Loops

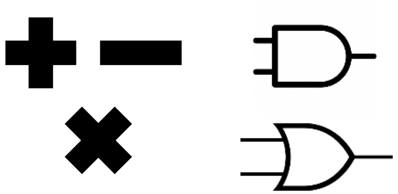


SIMD Batching



Parameter Selection

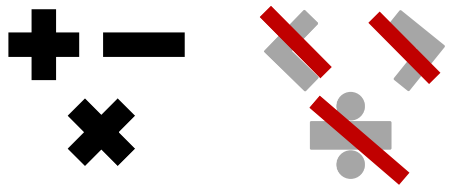
Noise Management



Limited Operators

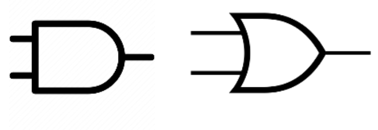
Why do we need FHE compilers?

FHE schemes

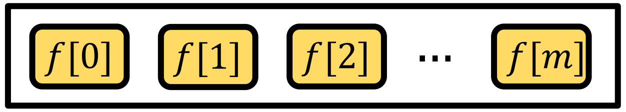


Arithmetic FHE

- Schemes: BFV/BGV/CKKS
- RLWE-based ciphertext
- Support SIMD parallelism
- Only support arithmetic operations



Boolean Gates



Look-up Table

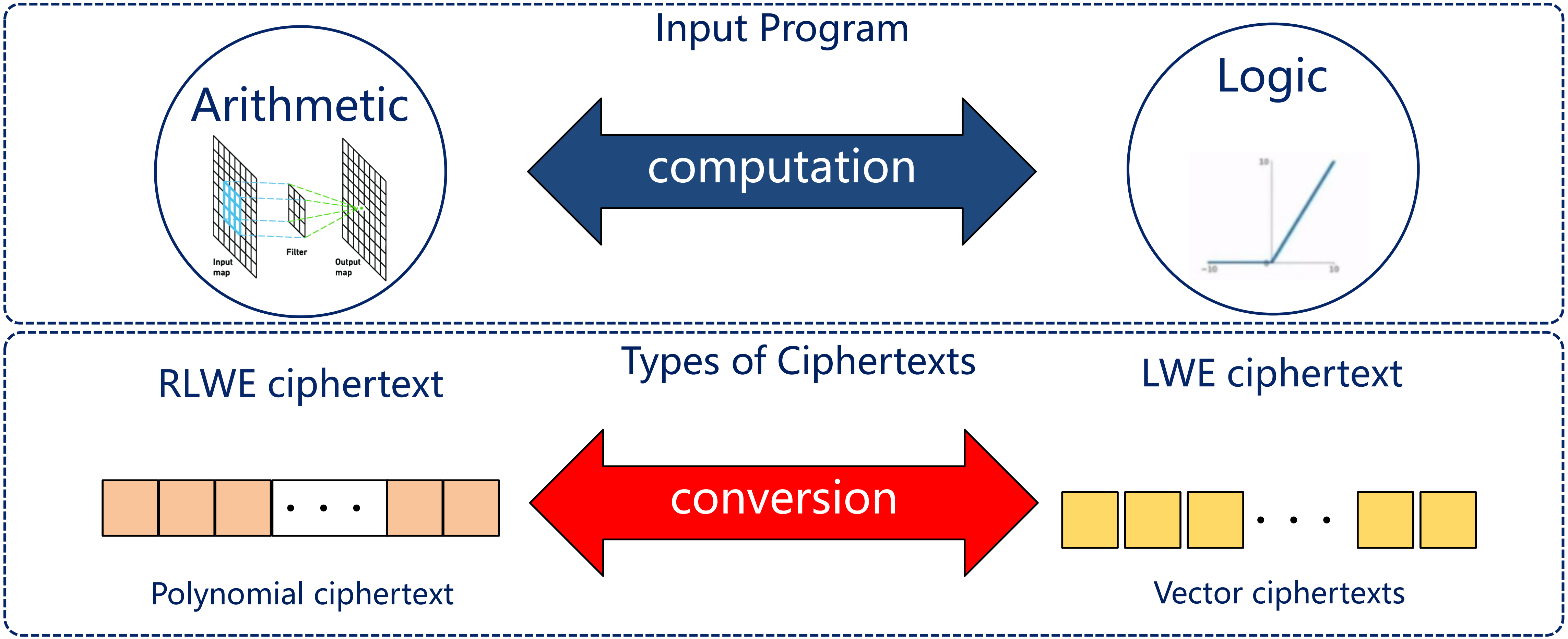
Logic FHE

- Schemes: FHEW/TFHE
- LWE-based ciphertext
- Support logic operations through LUT and Boolean gates (Bootstrapping)
- Slow multiplication operations

The existing FHE compilers only support compilation for single FHE scheme

Why do we need FHE compilers?

Automatic scheduling for different FHE schemes

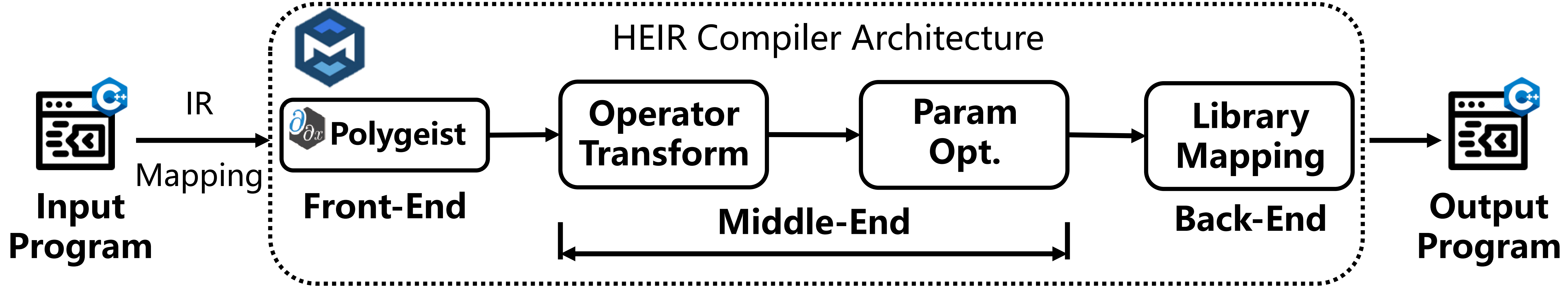


Motivation

- Build a DSL-free functional complete FHE compiler that does not solely rely on Boolean-circuit representation
- Propose an unified intermediate representation for FHE to support cross-scheme FHE program scheduling and optimization
- Support compilation for large-scale computation tasks that contain both arithmetic and logic operations

HEIR Compiler Pipeline

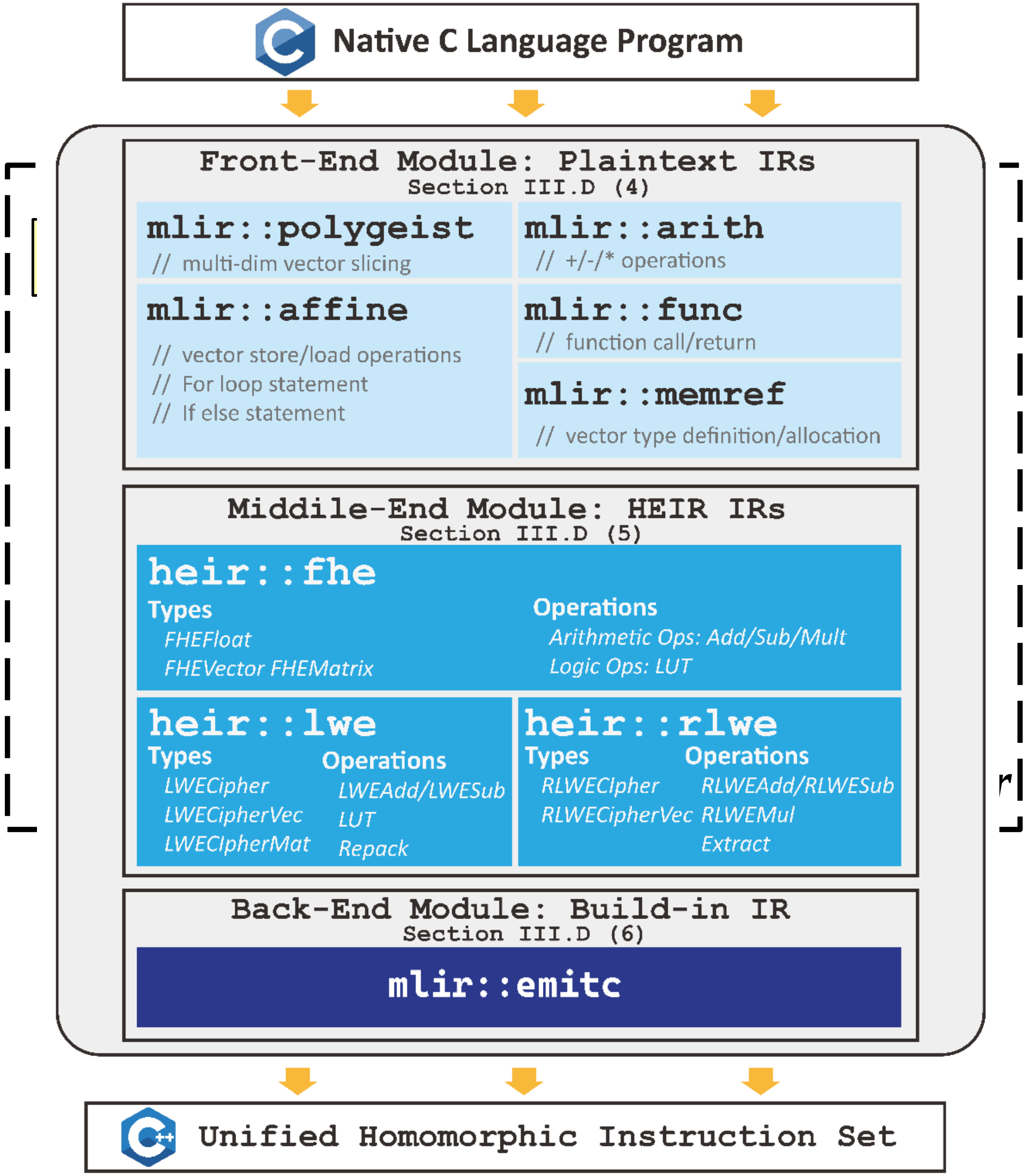
HEIR: Compiler for FHE



Key components:

- Program Segmentation: cut operators into regions according to arithmetic or logic circuit
- Encoding Optimization: cut data into types according to plaintext encoding
- Parameter Management: Setting and optimizing encryption parameters

IR stack definition in HEIR



- Front-End tool（Polygeist）converts C source code to MLIR **Standard dialects**
- Lowering MLIR Standard dialects to the unified **FHE Dialect** representation
- Segmenting the FHE IR program and lowering to **LWE** and **RLWE Dialects**
- Lowering to **EmitC Dialect** to dock with APIs/instructions in FHE library

Example Program

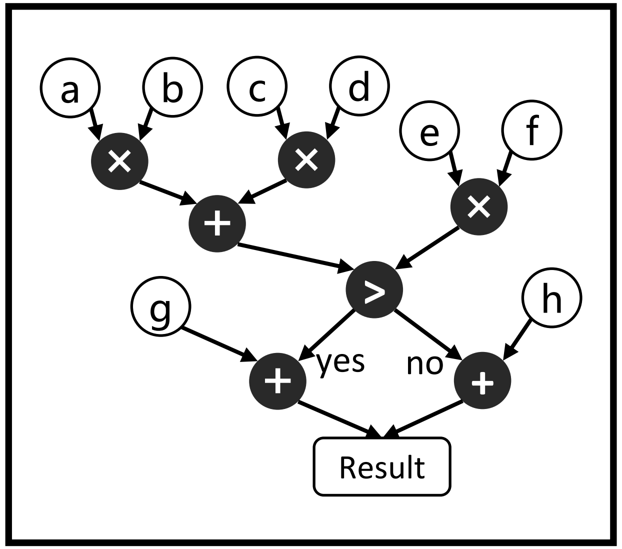
Program Transformation

```
#include<stdio.h>
int min_dist(int data[5][4], cent[4]) {
  int min;
  int dist[5];
  for (int i=0; i < 5; i++) {
    for (int j = 0; j < 4; j++)
      dist[i]+=(data[i][j]-cent[j])^2
  }
  min = dist[0];
  for (int i = 1; i < 5; i++) {
    if (min > dist[i]) min = dist[i];
  }
  return min;}
```

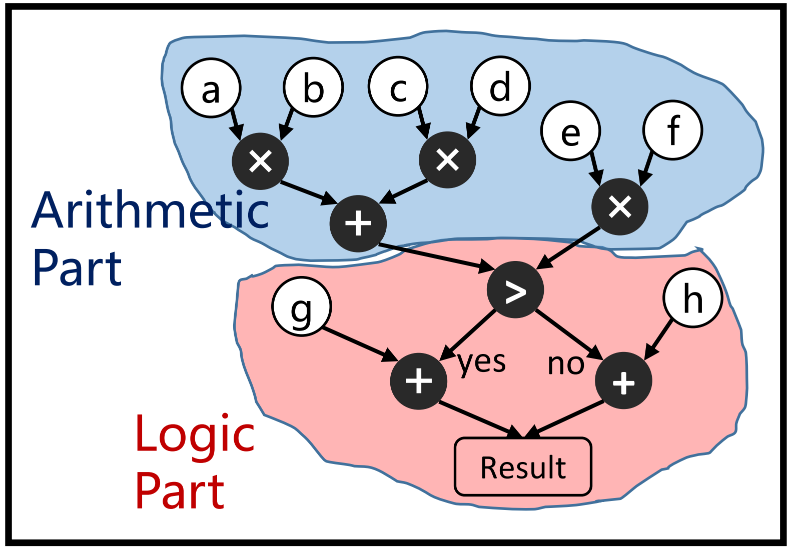
```
LWECipher min_dist(vector<RLWECipher> data[5][4],
RLWECipher cent) {
  LWECipher min;
  vector<LWECipher> dist;
  for (int i=0; i < 5; i++)
    dist[i] = inner_prod(data[i], data[i]) +
    inner_prod(cent, cent)-2*inner_prod(data[i], cent);
  min = dist[0]
  for (int i = 1; i < 5; i++) {
    LWECipher msb = dist[i] – min;
    msb = LUT(msb, x < 0 ? 1 : 0);
    min = msb * dist[i] + (1 – msb) * min
  } return min;}
```

Program Segmentation

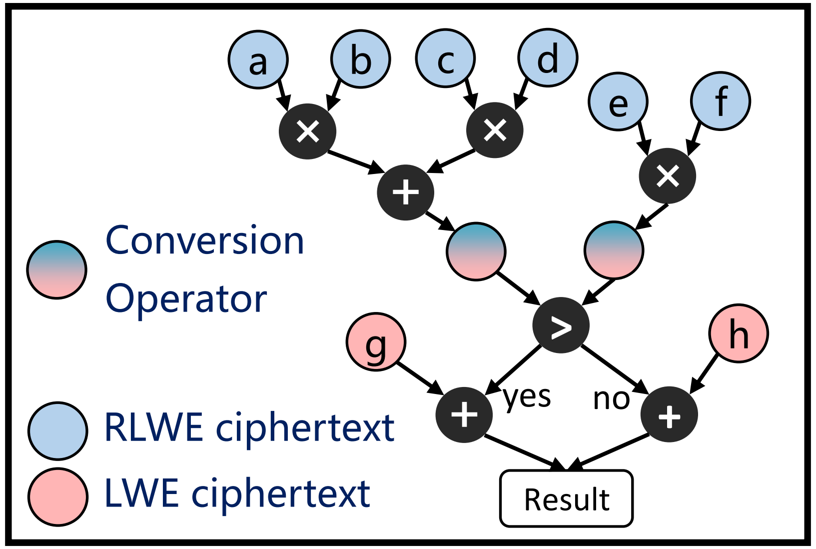
Operator-based Type Inference



Data flow graph of input program



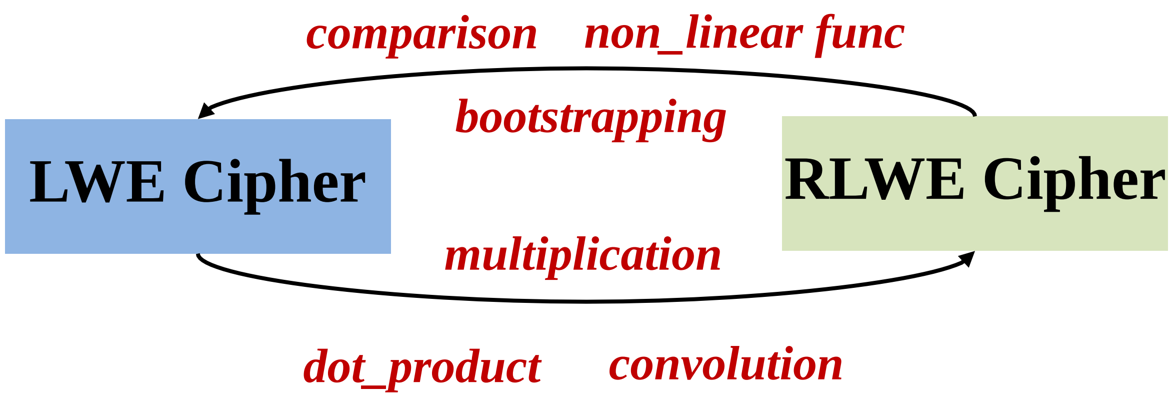
Program Segmentation



Data type Alignment

Program Segmentation

Ciphertext Type Conversion in HEIR



Specific Operator	Constraint Type	Operator Insertion
fhe.CompareOp	LWE	fhe.ExtractOp
fhe.LUTOp	LWE	fhe.ExtractOp
fhe.MultOp	RLWE	fhe.RepackOp
fhe.DotProductOp	RLWE	fhe.RepackOp

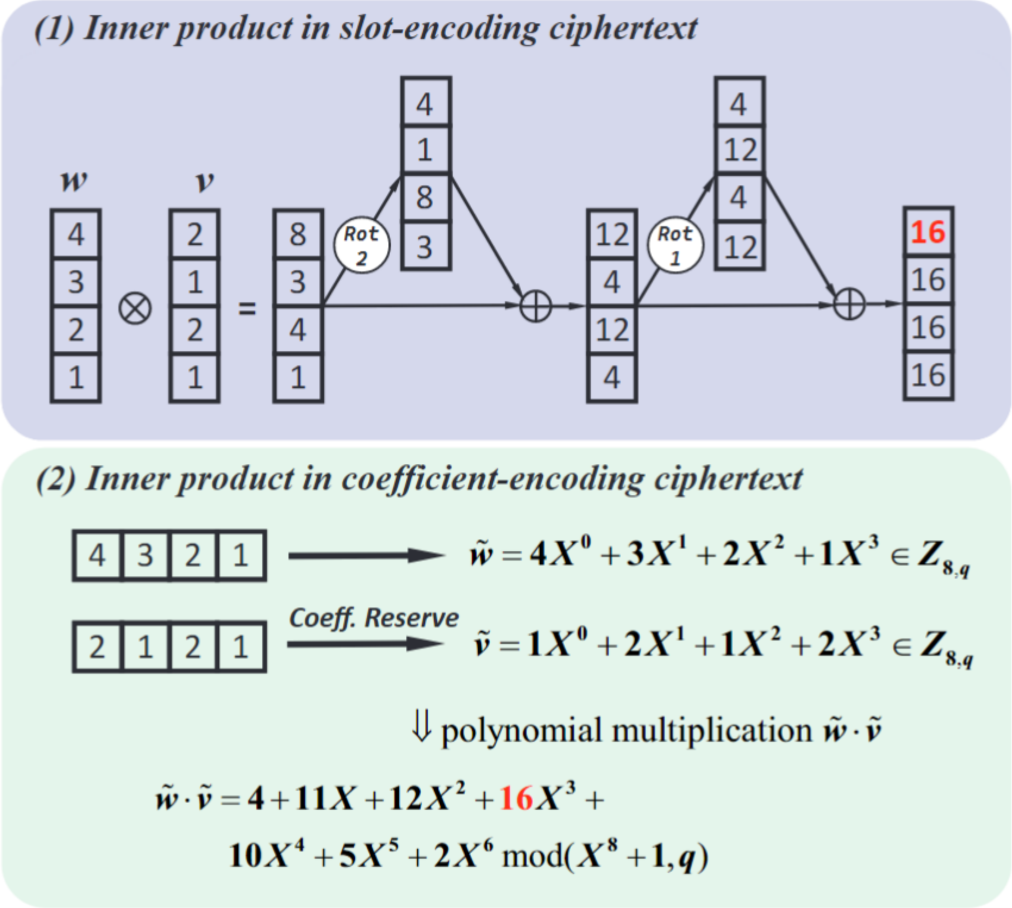
$$\text{FHE} - \text{LWE}\ \frac{u \in \{Inputs, Variables\} \quad u.type \in \{\texttt{FHEVector},\ \texttt{FHEMatrix}\}}{u.type \leftarrow \{\texttt{LWECipherVec},\ \texttt{LWECipherMat}\}}$$

$$\text{LWE} - \text{RLWE}\ \frac{u \in \{Inputs, Variables\} \quad u.type \in \{\texttt{LWECipherVec},\ \texttt{LWECipherMat}\} \quad u.op \in \{\textsc{Multiply}, \textsc{Slice}\}}{u.type \leftarrow \{\texttt{RLWECipher},\ \texttt{RLWECipherVec}\} \quad u \leftarrow \textsc{Repack}(u)}$$

$$\text{RLWE} - \text{LWE}\ \frac{u \in \{Inputs, Variables\} \quad u.type \in \{\texttt{RLWECipher},\ \texttt{RLWECipherVec}\} \quad u.op \in \{\textsc{Lut}, \textsc{Load}, \textsc{Store}\}}{u.type \leftarrow \{\texttt{LWECipherVec},\ \texttt{LWECipherMat}\} \quad u \leftarrow \textsc{SampleExtract}(u)}$$

Encoding Optimization

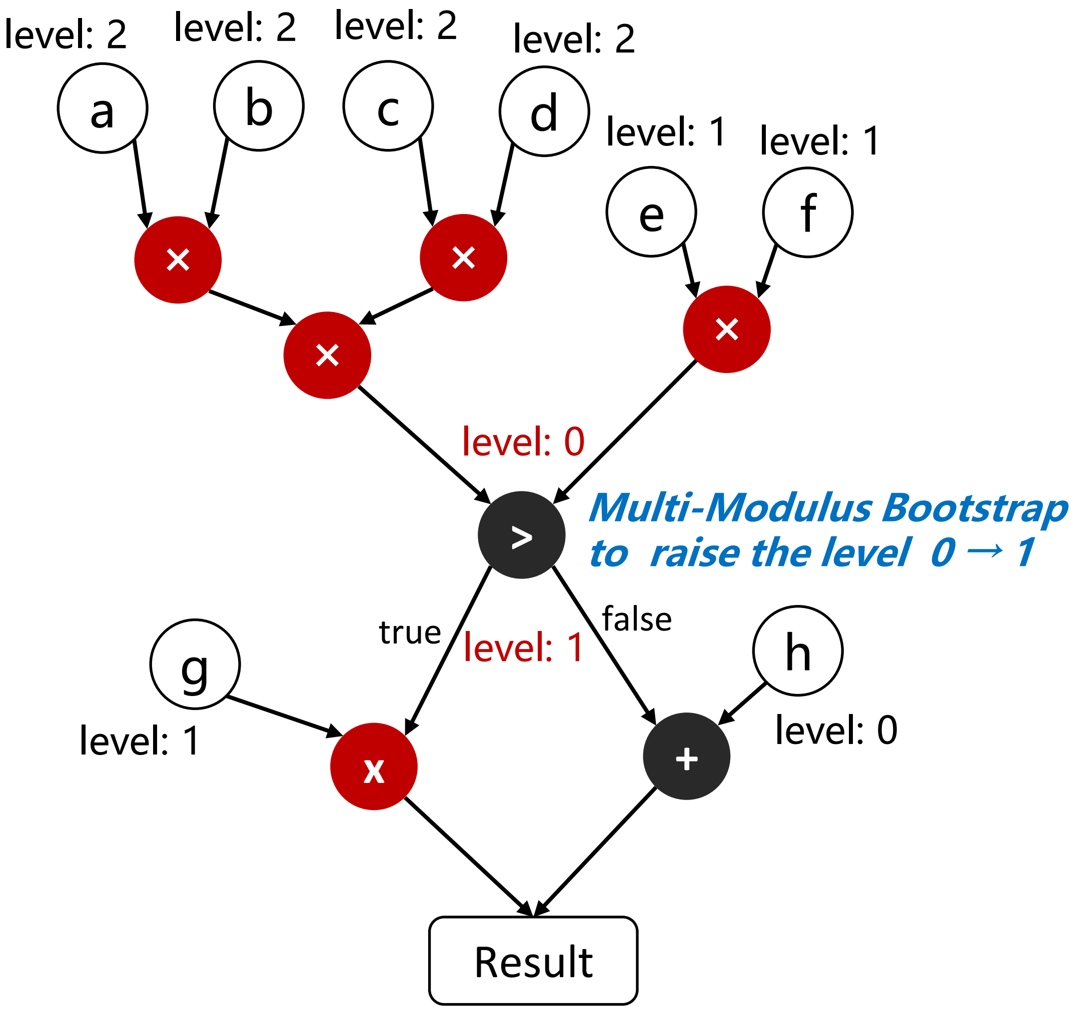
```
LWECipher min_dist(vector<RLWECipher> data[5][4],
RLWECipher cent) {
  LWECipher min;
  vector<LWECipher> dist;
  for (int i=0; i < 5; i++)
    dist[i] = inner_prod(data[i], data[i]) +
    inner_prod(cent, cent)-2*inner_prod(data[i], cent);
  min = dist[0]
  for (int i = 1; i < 5; i++) {
    LWECipher msb = dist[i] – min;
    msb = LUT(msb, x < 0 ? 1 : 0);
    min = msb * dist[i] + (1 – msb) * min
  } return min;}
```



- Rule-based method (Pattern Matching) to accelerate computations, e.g., accumulation/inner product/convolution

Parameter Selection

Level Management

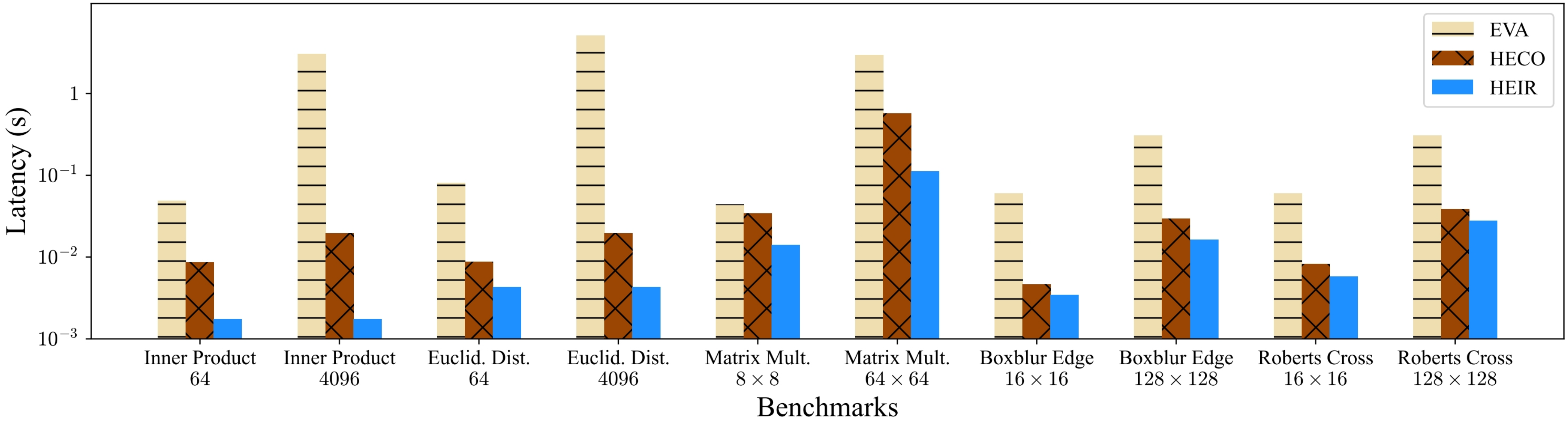


- The level of a ciphertext is associated with its "modulus", indicating the multiplicative depth that can be supported.

- Multi-Modulus Bootstrapping is proposed to enhance the level of a ciphertext while enabling non-polynomial operations.

Experiments

1) Arithmetic Circuit Evaluation

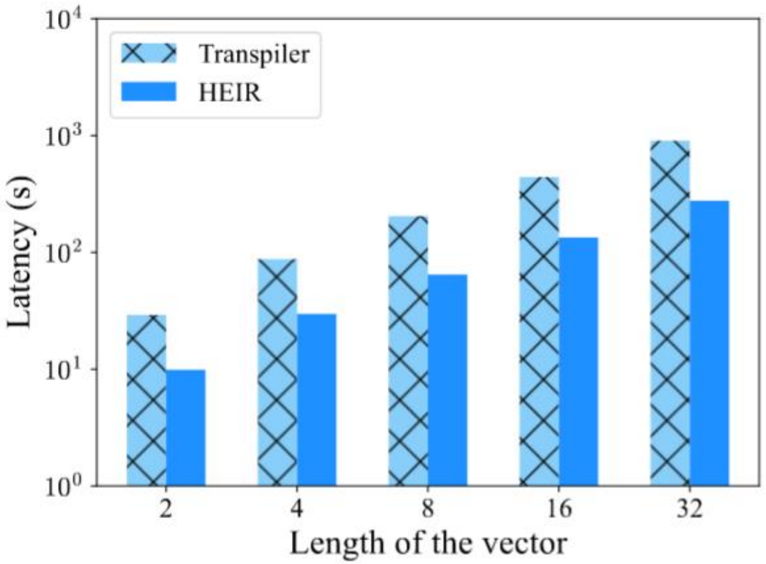


- **Compared with EVA and HECO, HEIR introduces encoding optimization**

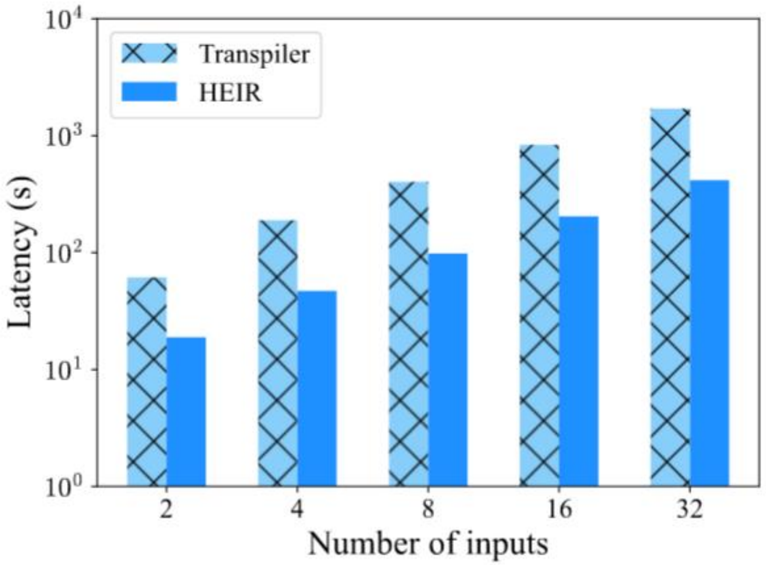
Dathathri, R. et al. EVA: An Encrypted Vector Arithmetic Language and Compiler for Efficient Homomorphic Computation – PLDI 2020.
Viand, A. et al. HECO: Fully Homomorphic Encryption Compiler – USENIX Security 2023.

Experiments

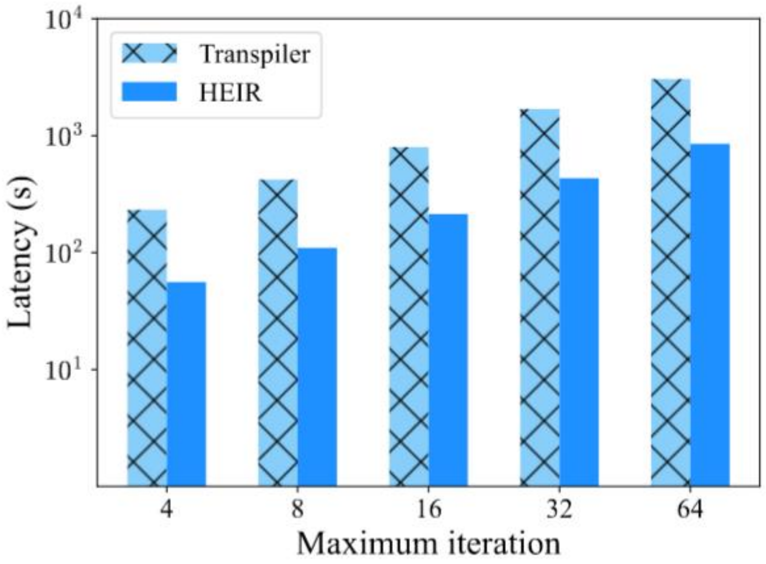
2) Logic Circuit Evaluation



(a) Minimum value in the the vector



(b) Minimum index in the the vector



(c) Result of a Fibonacci sequence

- **Transpiler compiles the program to a logic circuit while HEIR evaluates non-polynomial functions with Look-Up Tables**

Gorantala, S. et al. A general purpose transpiler for fully homomorphic encryption – IACR EPRINT 2021.

Experiments

Compiling a K-Means Program

(1) 5 data points, 3 features, 2 centroids

Implemententaions	Latency (s)
Transpiler	**24321.191**
HEIR	**135.634**
Hand-tuned	**59.338**

(2) 10 data points, 3 features, 2 centroids

Implementations	Euclid Dist	Euclid Dist + Min Value	Centroids Update	Latency (s)
Transpiler	2842.126	5406.806	2385.829	**Compilation fail**
HEIR	--	--	--	**248.823**
Hand-tuned	--	--	--	**105.621**

Generated program runs ~180x faster than Google Transpiler
Supports larger-scale program compilation

Gorantala, S. et al. A general purpose transpiler for fully homomorphic encryption – IACR EPRINT 2021.

github.com/heir-compiler/HEIR

hub.docker.com/repository/docker/zhaozian/heir

HEIR: A Unified Representation for Cross-Scheme Compilation of Fully Homomorphic Computation