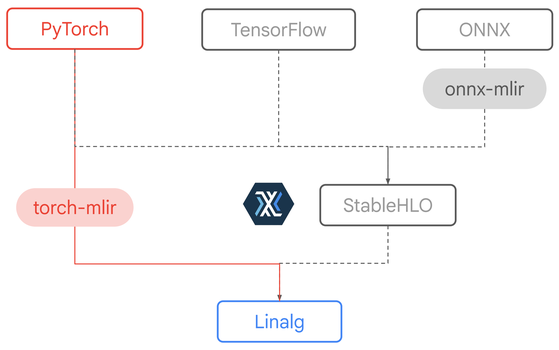
HEIR: Fully Homomorphic Machine Learning with a Universal Compiler

Asra Ali
asraa@google.com
https://arxiv.org/abs/2508.11095

An FHE compiler toolchain and development platform without sacrificing generality and extensibility

ML Frontend



Torch models are converted with *torch-mlir* to `linalg` on tensors (with `tensor` and `arith` dialects) is used as HEIR input

Linalg Entry Level

Linalg dialect is a funnel dialect for HEIR's MLIR frontend. Its abstraction level is required for matching on ML kernel operations for optimization. Canonicalization patterns simplify and reduce memory shuffling operations and reduce non-linear operations at this level.

```
module attributes {backend.openfhe, scheme.ckks} {
 func.func @mnist(
    %input: tensor<784> {secret.secret}) -> tensor<10> {
   %matrix1 = arith.constant dense<...>: tensor<512x784>
   %bias1 = arith.constant dense<...> : tensor<512>
   %matrix2 = arith.constant dense<...>: tensor<10x512>
   %bias2 = arith.constant dense<...> : tensor<10>
   %cst = arith.constant dense<0.0> : tensor<512>
   %0 = linalg.matvec ins(%matrix1, %input)
       {kernel = "diagonal"}
   %1 = arith.addf %0, %bias1
   %2 = arith.maximumf %1, %cst
       {degree = 6, lower = -15.0, upper = 12}
   %3 = linalg.matvec ins(%matrix2, %2)
   %4 = arith.addf %3, %bias2
   return %4
 }
}
```

Compilation Configuration

- **Backend** and **scheme** selection
- **Secret input data** selected with annotations
- Non-linear activation **approximation degree**
- **Range bounds** from model metadata
- User controlled **kernel** selection

ML Compilation Pipeline

Relation-based Ciphertext Layouts

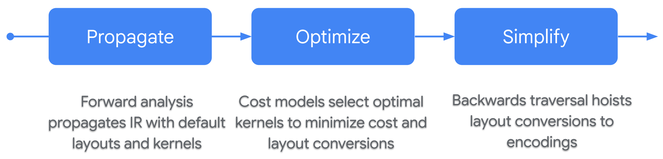
Layouts are a *partial function* mapping from the index set of a cleartext tensor to the index set of a list of ciphertext slots using *Presburger relations*, quasi-affine formula.

- Fully general layout annotations describing plaintext - ciphertext relation
- Polyhedral optimization with Integer Set Library analyzes and manipulates layouts, for e.g. to compute kernel simplifications or slot utilization for batching

$$\left\{ (i,j) \mapsto (ct, slot) \;\middle|\; \begin{array}{l} (i - j + ct) \mod 8 = 0 \\ (i - slot) \mod 1024 = 0 \\ 0 <= i, j, ct < 8 \\ 0 <= slot < 1024 \end{array} \right\}$$

Mapping (i, j) of an 8x8 tensor to 8 ciphertexts with 1024 slots

Layout Optimization Flow



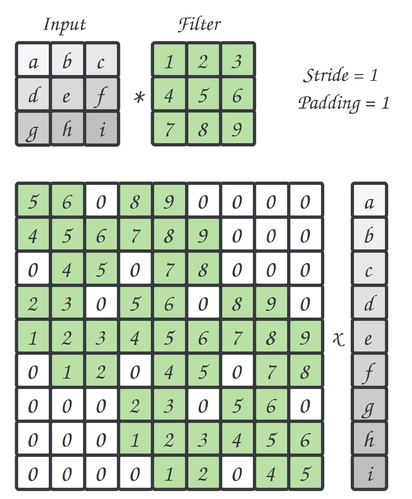


Diagram modified from Fig 3 of Orion: A Fully Homomorphic Encryption Framework for Deep Learning

$$m_r = (i_{dr} + P)F + i_{dc} + P$$
$$m_c = W_d i_{dr} + i_{dc} + W_d i_{fr} + i_{fc}$$

New Layout Integrations

HEIR integrates *bicyclic* [1] and *tricyclic* [2] layouts and kernels to compute batched matrix multiplication for parallelized multi-head self-attention with optimal multiplicative depth.

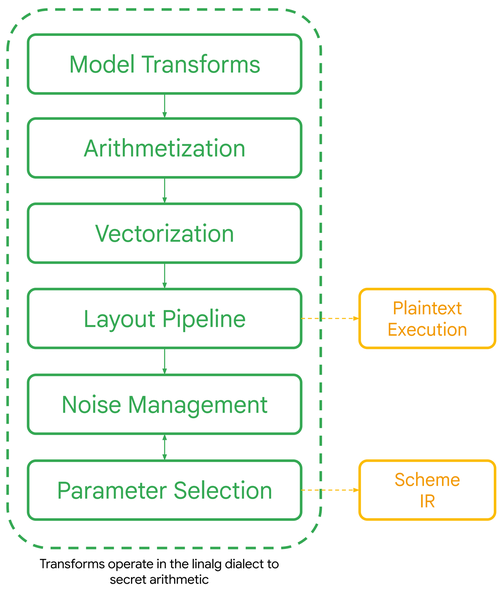
Supported layouts and kernels are easily extended with ISL utilities and a testable MLIR-agnostic kernel library.

[1] J. Chen, L. Yang, W. Wu, Y. Liu and Y. Feng, "Homomorphic Matrix Operations Under Bicyclic Encoding," in IEEE Transactions on Information Forensics and Security, vol. 20, pp. 1390-1404, 2025, doi: 10.1109/TIFS.2024.3490862.
[2] Lim, Lawrence et al. "Tricycle: Private Transformer Inference with Tricyclic Encodings." IACR Cryptol. ePrint Arch. 2025 (2025): 1200.

Optimization Variety Pack

HEIR's ML pipeline utilizes a number of generally applicable optimization patterns:

- Sparse matrix product simplification
- Baby-step giant-step for general reductions
- Minimal depth polynomials evaluation with Paterson-Stockmeyer
- Fast (hoisted) rotation rewrites
- Minimized extended key basis switching
- High level program vectorization
- Shift networks for layout conversions
- Loop support with HALO optimizations
- Multiplexed data packing for slot utilization



Make It Easy

HEIR simplifies the developer and debugging experience with

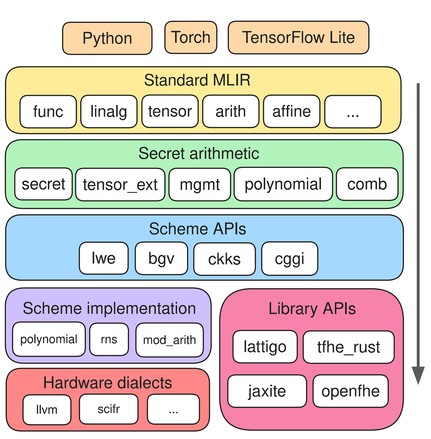
- Tracking and debugging utilities from MLIR
- Plaintext execution mode with custom debug handlers
- Client helpers for encoding and encryption/decryption
- Output code is human-readable code to support inspection and modification
- Cleartext computations are hoisted to separate functions for precomputation
- Scheme-specific parameter selection

```
// preprocessing functions
PlaintextT matvec__preprocessing(CryptoContextT cc) {
 ...
 const auto& pt2 = cc->MakeCKKSPackedPlaintext(c0);
 return pt2;
}

// main workload
CiphertextT matvec(CryptoContextT cc, CiphertextT ct) {
 ...
 const auto& ct5 = cc->EvalMult(ct4, pt2);
 const auto& ct6 = cc->EvalRotate(ct, 3);
 ...
 const auto& ct47 = cc->EvalAdd(ct38, ct46);
 const auto& ct48 = cc->EvalMultNoRelin(ct47, ct47);
 const auto& ct49 = cc->Relinearize(ct48);
 ...
}

// client functions
CiphertextT matvec__encrypt__arg0(
CryptoContextT cc, std::vector<float> v0, PublicKeyT pk);
std::vector<float> matvec__decrypt__result0(
CryptoContextT cc, CiphertextT ct, PrivateKeyT sk);
CryptoContextT matvec__generate_crypto_context();
CryptoContextT matvec__configure_crypto_context(
CryptoContextT cc, PrivateKeyT sk);
```

Hardware Integrations



Exit dialects

Support for multiple backends (CPU, GPU, FPGA, ASICs, and photonics) allows for comprehensive testing and benchmarking. After HEIR's high level program analysis and compilation, data layouts, kernels, schemes, and parameters are selected and the IR uses scheme level operations. Scheme level IR is lowered in two possible ways to exit HEIR:

1. ***Library dialects*** (e.g. Lattigo, OpenFHE, tfhe-rs) mirror APIs and are translated to code via HEIR's emitter. Allows fast prototyping and easy integration but limits the ability to perform fusion or other cross-operation optimizations.
2. ***Low level IRs***: scheme operations are implemented using polynomial and modular arithmetic dialects. Hardware specific toolchains handle further optimization, scheduling and assembly. (e.g. the LLVM toolchain compiles the MLIR for CPU). This path is suitable for longer term, robust integrations.

Belfort

Belfort's integrates their FPGA-based accelerator with HEIR through the CGGI boolean and shortint APIs. They utilize vectorization strategies in HEIR and software optimizations in their custom tfhe-rs library for performance.

Van Beirendonck, Michiel et al. "FPT: A Fixed-Point Accelerator for Torus Fully Homomorphic Encryption." In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23). Association for Computing Machinery, New York, NY, USA, 741–755. https://doi.org/10.1145/3576915.3623159

CROSS

TPU-native CKKS implementation with SoTA performance vs GPU (20ms bootstrap) using JAX. HEIR integrations utilizes the CKKS dialect to lower to the CROSS API exit dialect.

Tong, Jianming et al. "Leveraging ASIC AI Chips for Homomorphic Encryption." in IEEE International Symposium on High-Performance Computer Architecture, 2026.

Optalysys

Optalysys utilizes *photonic computing technology* to perform modular arithmetic operations over the Polynomial Modular Number System (PMNS). Integration with HEIR's generated low level NTT and mod arith code will allow running FHE workloads on Optalysys' optical processing chips.

https://optalysys.com/resource/optalysys-partners-with-google-heir/

CORNAMI Intelligent Computing

Cornami's MX2 systolic array is integrated as a backend to HEIR's MLIR pipeline for CGGI and CKKS schemes. HEIR exits to Cornami's Secure Computing Interface Framework (SCIFR) with custom optimizations.

Ovichinnikov, Denis et al. "Resource Estimation of CGGI and CKKS scheme workloads on FracTLcore Computing Fabric." ArXiv abs/2510.16025 (2025): n. pag.

Plus more backends in progress (e.g. FIDESlib [1] GPU backend) and under NDA.

C. Agullo-Domingo et al., "FIDESlib: A Fully-Fledged Open-Source FHE Library for Efficient CKKS on GPUs," in 2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Ghent, Belgium, 2025, pp. 1–3, doi: 10.1109/ISPASS64960.2025.00045.

FHETCH

The global FHE hardware consortium (www.fhetch.org)

HEIR tracks progress of the *polynomial intermediate representation* (IR) developed by FHE Technical Consortium for Hardware (FHETCH). The IR aims to provide an standardised set of hardware-level operations for interoperable platform integration. HEIR's *polynomial* dialect aligns with the evolving standard.

Community

HEIR's open-source framework supports major homomorphic encryption methods, enabling efficient research and benchmarking. Its architecture facilitates the integration of state of the art and emerging methodologies, as evidenced by various projects built with or incorporated into HEIR.

github.com/google/heir

discord.com/invite/fhe-org
channel: #heir

google.github.io/heir/community

FHelipe Layout Hoisting

HEIR uses a FHelipe's hoisting heuristic to minimize layout conversions between operations.

Average-Case Noise Analysis

HEIR was used to experientially demonstrate underestimations of average-case noise analysis.

HALO Compiler Loop Support

HEIR adopts transforms from the HALO compiler for loop-aware bootstrapping placement.

ROTOM: Autovectorizing HE

ROTOM's tensor vectorization strategy is integrated as an option for layout optimization.

Orion Compiler Kernels

HEIR incorporates Orion's convolution data layout and kernel with double-hoisting and BSGS.

KeyMemRT Memory Scalability

Key memory management minimization strategies are incorporated into HEIR.

Tricycle: Private Transformers

HEIR supports tricyclic layouts to enable ciphertext matrix multiplications for self-attention.

Vos-Vos-Erkin Shift Networks

Efficient shift network implementation of layout conversions using graph coloring

Chen, Edward et al. "Bridging Usability and Performance: A Tensor Compiler for Autovectorizing Homomorphic Encryption." IACR Cryptol. ePrint Arch. 2025 (2025): 1319.
Zhou, Zikai, et al. "Orbit: Optimizing Rescale and Bootstrap Placement with Integer Linear Programming Techniques for Secure Inference." no. 2026/213, 2026, Cryptology ePrint Archive. Cryptology ePrint Archive (eprint.iacr.org), https://eprint.iacr.org/2026/213.
Ünay, Eymen, et al. "KeyMemRT Compiler and Runtime: Unlocking Memory-Scalable FHE." arXiv:2601.18445, arXiv, 26 Jan. 2026. arXiv.org, https://doi.org/10.48550/arXiv.2601.18445.
Gao, Mingyu and Hongren Zheng. "A Critique on Average-Case Noise Analysis in RLWE-Based Homomorphic Encryption." Proceedings of the 13th Workshop on Encrypted Computing & Applied Homomorphic Computing (2025)
Krastev, Aleksandar, et al. "A Tensor Compiler with Automatic Data Packing for Simple and Efficient Fully Homomorphic Encryption." Proceedings of the ACM on Programming Languages, vol. 8, no. PLDI, June 2024, pp. 126–50. DOI.org (Crossref), https://doi.org/10.1145/3656382.
Cheon, Seonyoung, et al. "HALO: Loop-Aware Bootstrapping Management for Fully Homomorphic Encryption." Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 [Rotterdam Netherlands], 2025, pp. 572–85. DOI.org (Crossref), https://doi.org/10.1145/3669940.3707275.
Ebel, Austin, et al. "Orion: A Fully Homomorphic Encryption Framework for Deep Learning." Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 [Rotterdam Netherlands], 2025, pp. 734–49. DOI.org (Crossref), https://doi.org/10.1145/3676641.3716008.
Chen, Jingwei, et al. "Homomorphic Matrix Operations Under Bicyclic Encoding." IEEE Transactions on Information Forensics and Security, vol. 20, 2025, pp. 1390–404. IEEE Xplore, https://doi.org/10.1109/TIFS.2024.3490862.
Lim, Lawrence, et al. "Tricycle: Private Transformer Inference with Tricyclic Encodings." no. 2025/1200, 2025, Cryptology ePrint Archive. Cryptology ePrint Archive (eprint.iacr.org), https://eprint.iacr.org/2025/1200.
Vos, Jelle, et al. "Efficient Circuits for Permuting and Mapping Packed Values Across Leveled Homomorphic Ciphertexts." Computer Security – ESORICS 2022, edited by Vijayalakshmi Atluri et al., Springer International Publishing, 2022, pp. 408–23. Springer Link, https://doi.org/10.1007/978-3-031-17140-6_20.

Call for Contributions

Connect with us to explore potential research directions and integrations, including:

- Integrating Gentry-Lee FHE scheme "Fully Homomorphic Encryption for Matrix Arithmetic"
- Layout optimizer that uses the structure of Presburger relations, and/or the general joint layout+kernel selection problem
- New FHE scheme implementations (e.g. GBFV) and optimizations
- Incorporating memory constraints into cost models for kernel compilation
- Profile-guided optimizations for parameter selection & scale management