Orbit: Optimizing Rescale and Bootstrap placement with ILP Techniques for Secure Inference

Zikai Zhou*
IIIS, Tsinghua University

William Seo*
Amazon

Edward Chen
Carnegie Mellon University

Alex Ozdemir
Max Planck Institute

Fraser Brown
Carnegie Mellon University

Wenting Zheng
Carnegie Mellon University

Background

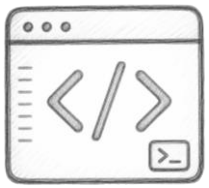
- AI is everywhere
- AI now operates in high-stakes domains



Healthcare



Finance



Code



Legal

Threat Model: Private inference in healthcare



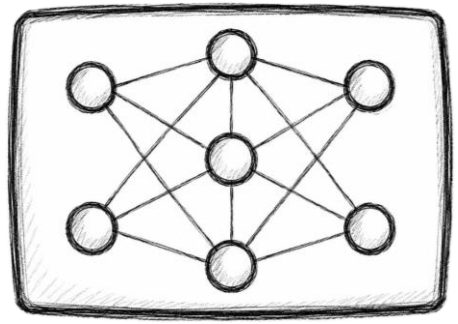
Doctor

Patient's record



Diagnosis





CNN model

Third-party Model Provider

Trust Surface 1: keeping client data private



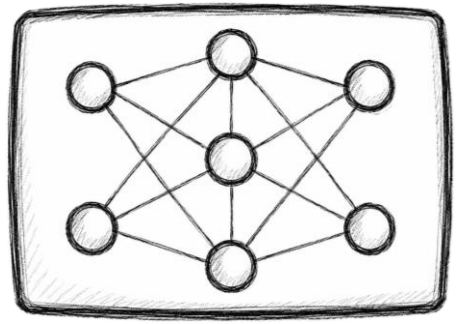
Doctor

Patient's record



Diagnosis

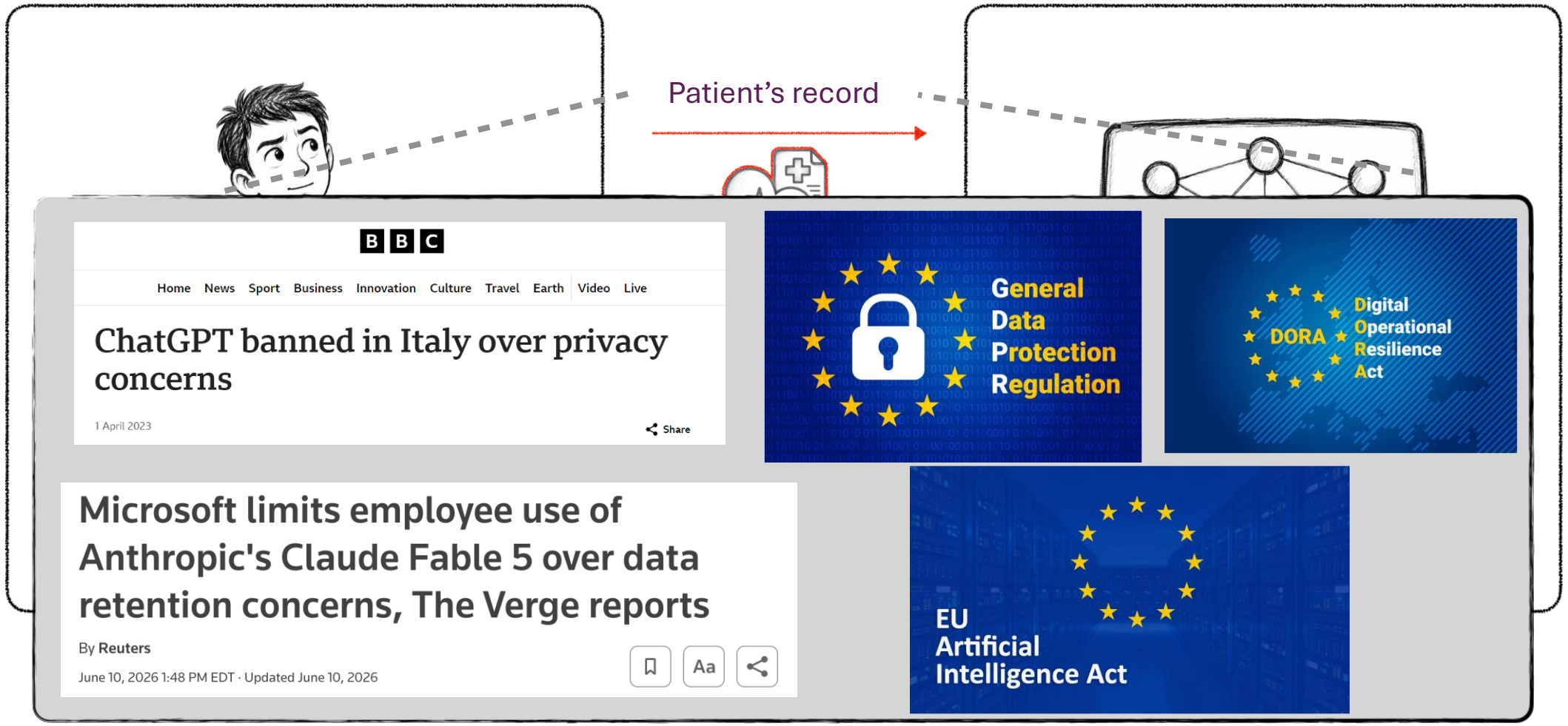




CNN model

Third-party Model Provider

Trust Surface 1: keeping client data private



Trust Surface 2: keeping model weights private



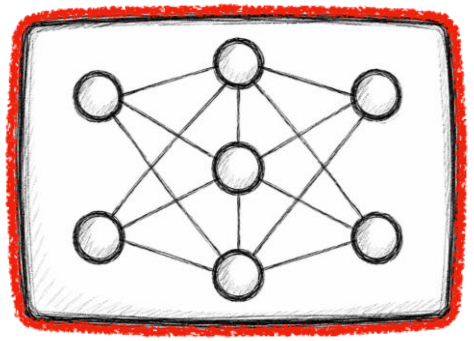
Doctor

Patient's record



Diagnosis

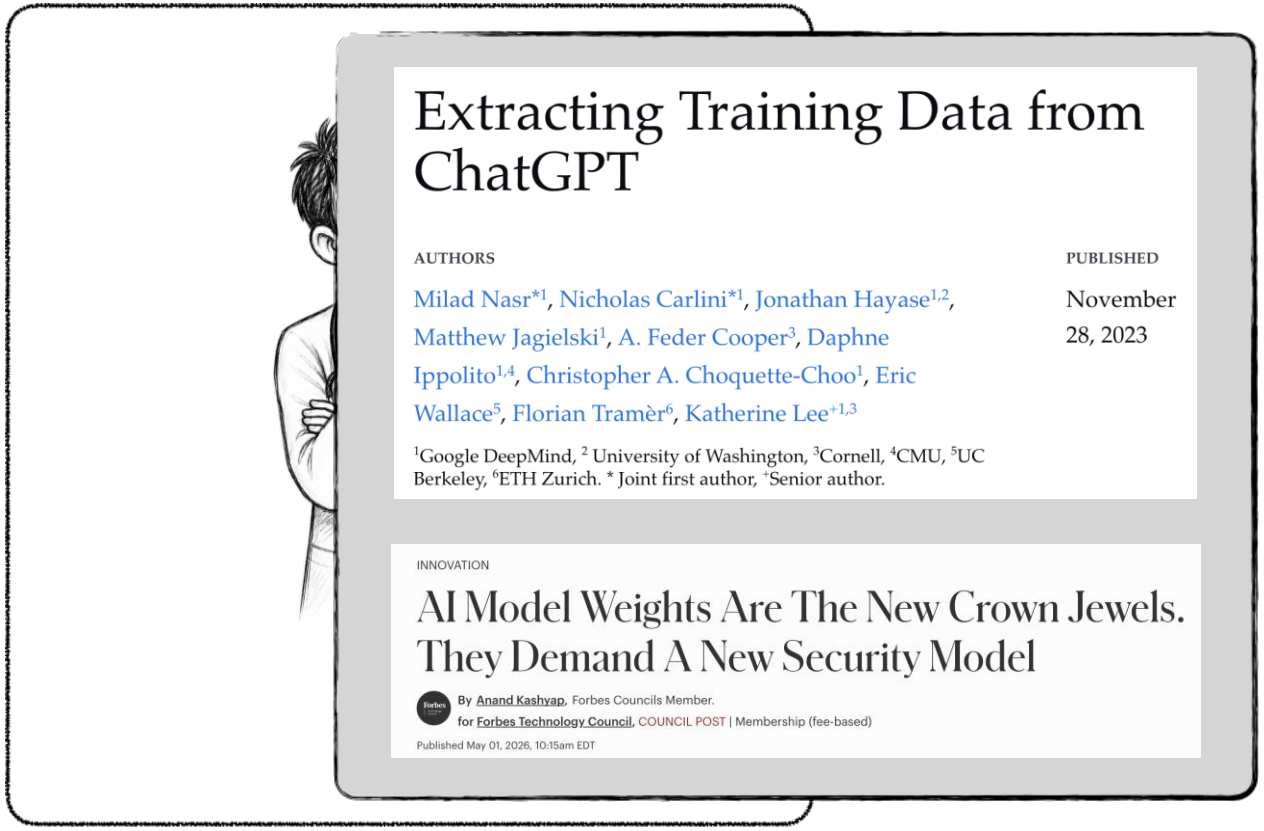


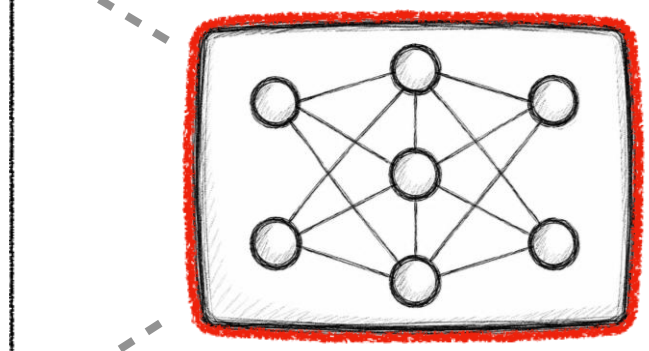


CNN model

Third-party Model Provider

Trust Surface 2: keeping model weights private

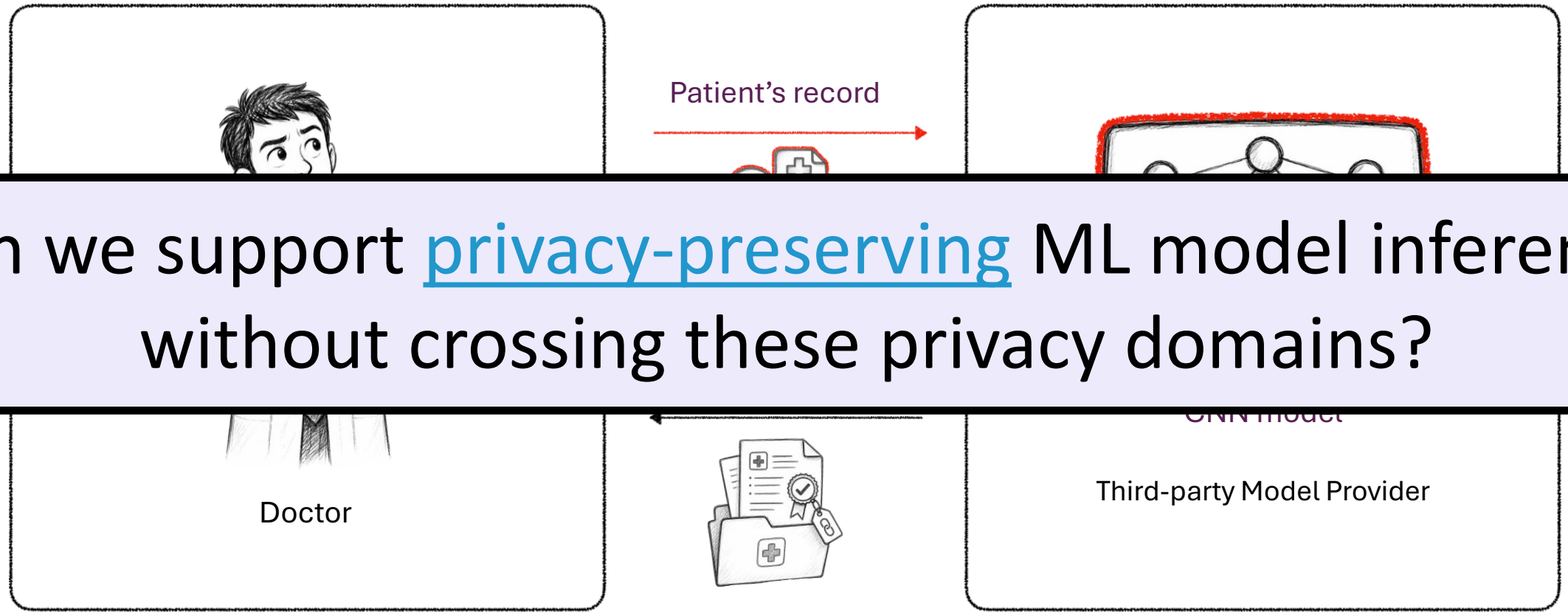




CNN model

Third-party Model Provider

Challenge: privacy-preserving machine learning



Can we support privacy-preserving ML model inference without crossing these privacy domains?

Fully Homomorphic Encryption (FHE)

- Computation on encrypted data

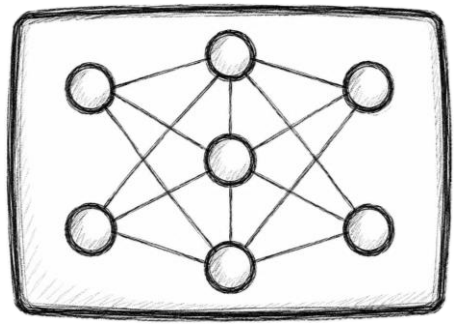
Server knows x



x

$f(x)$

Compute $f(\cdot)$ over x



Client

Doctor

Server

Third-party Model Provider

Fully Homomorphic Encryption (FHE)

- Computation on encrypted data



Client

Doctor

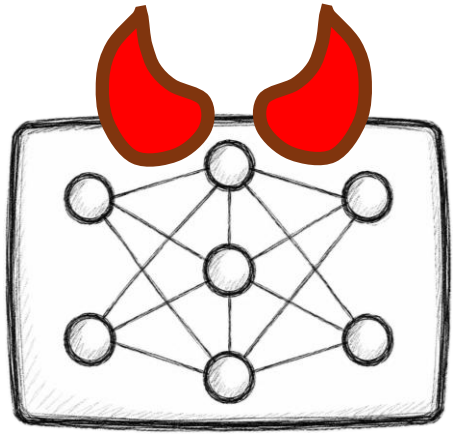
x

$f(x)$

Compute $f(\cdot)$ over x



Server knows x

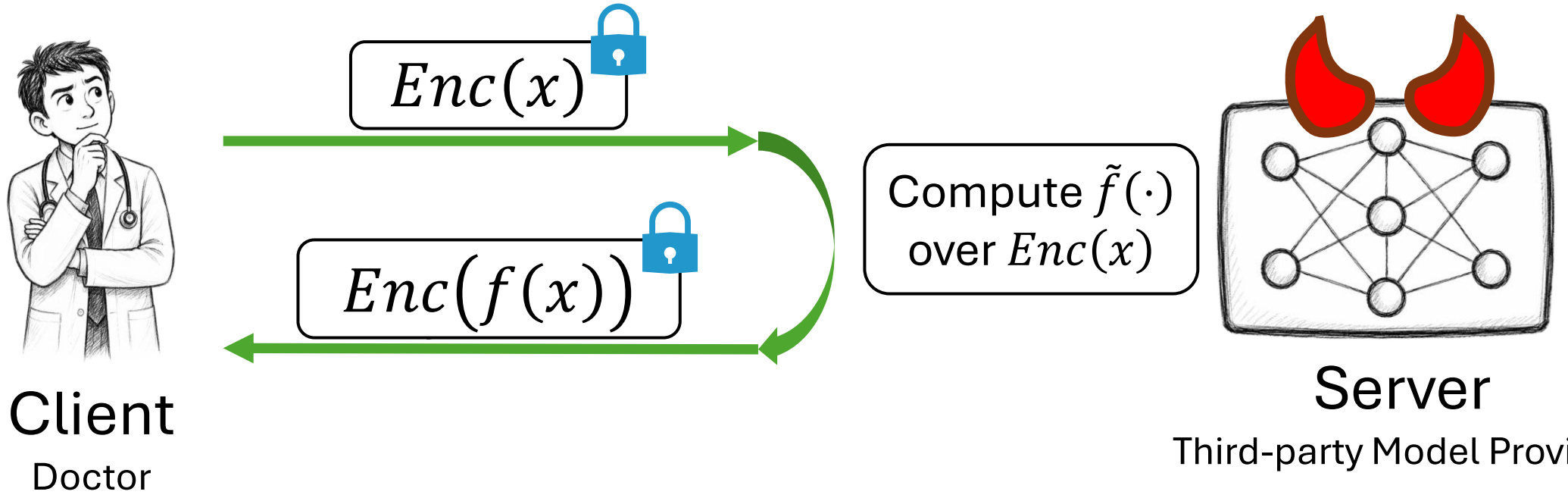


Server

Third-party Model Provider

Fully Homomorphic Encryption (FHE)

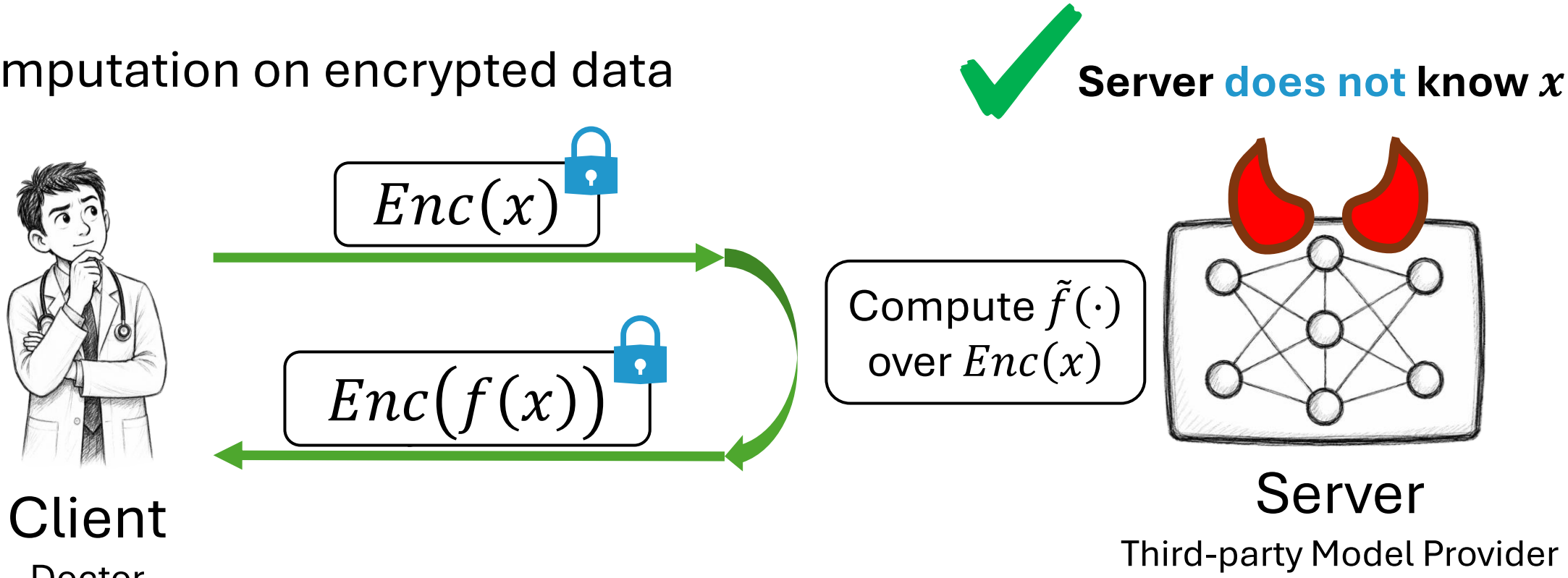
- Computation on encrypted data



$$FHE\colon\ \tilde{f}\big(Enc(x)\big) = Enc\big(f(x)\big)$$

Fully Homomorphic Encryption (FHE)

- Computation on encrypted data



$$FHE\colon\ \tilde{f}\big(Enc(x)\big) = Enc\big(f(x)\big)$$

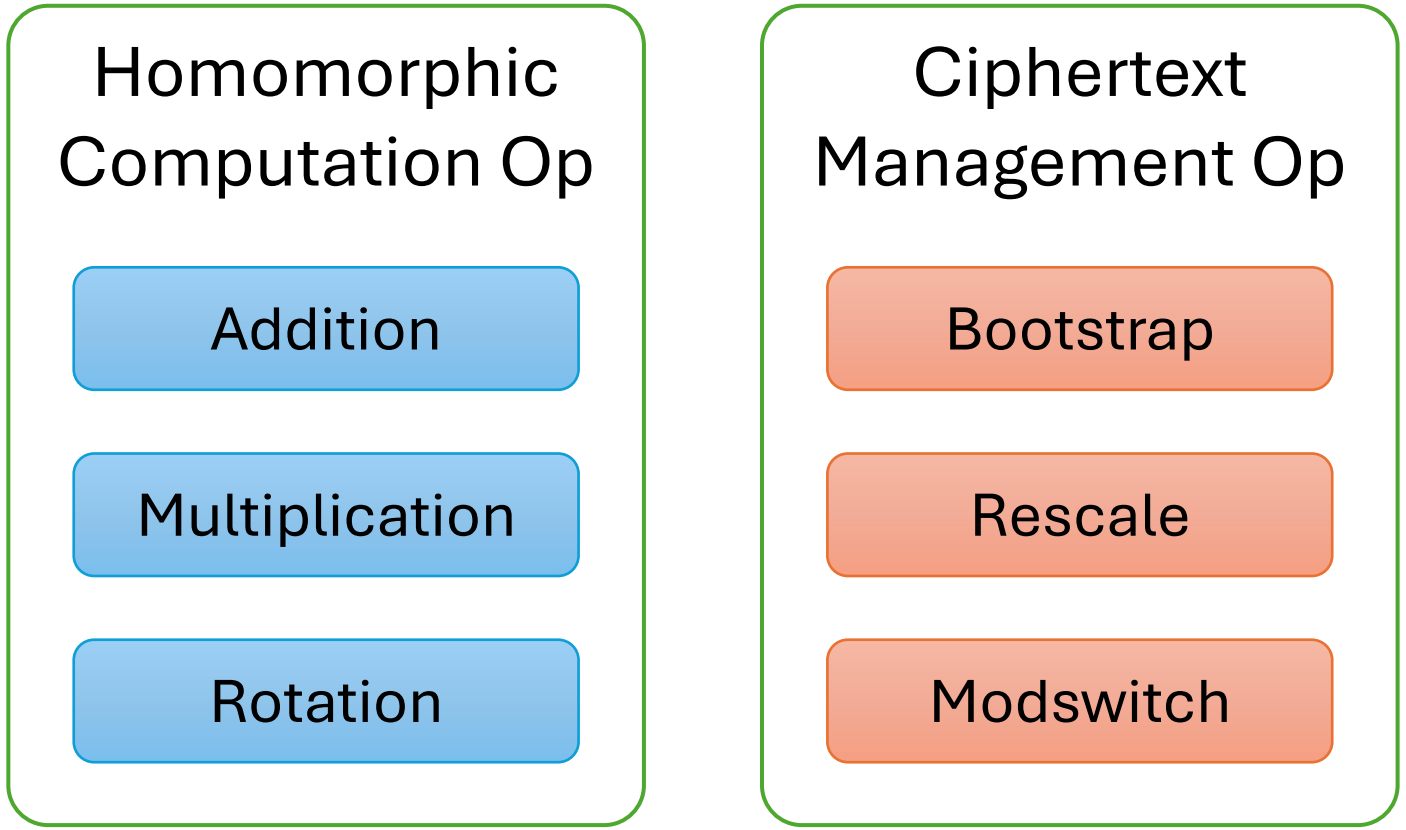
FHE for privacy-preserving ML

CKKS

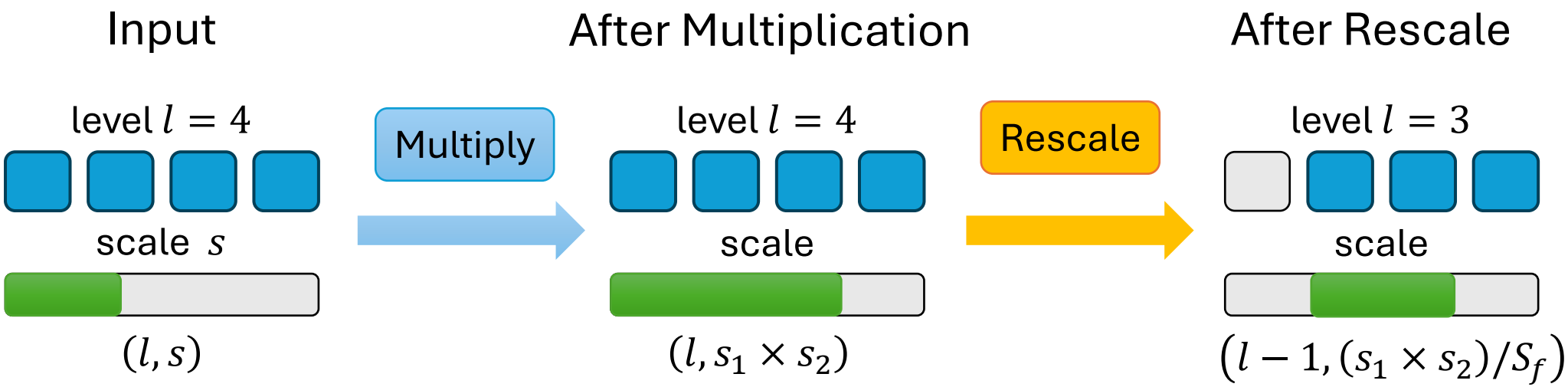
- Front-runner for ML
- Fixed-point arithmetic
- SIMD Batching

1000x slower

than plaintext

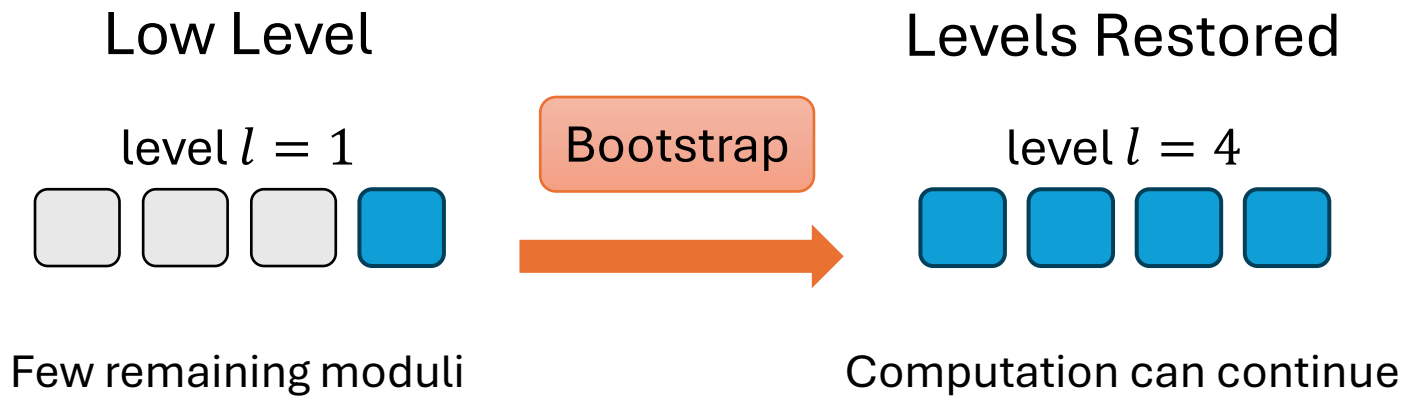


CKKS Background —— Level & Scale



Multiplication creates a rescale decision for the compiler

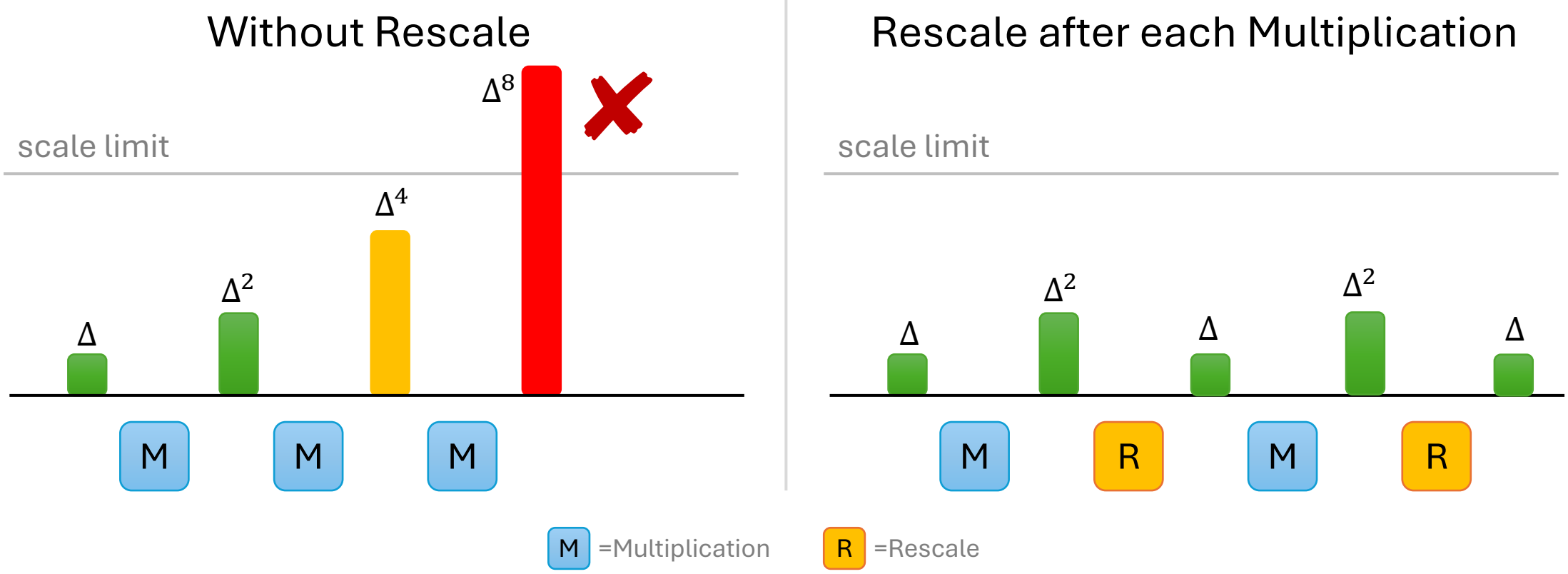
CKKS Background —— Bootstrap



Bootstrap
100-1000x
Slower than
Multiplication

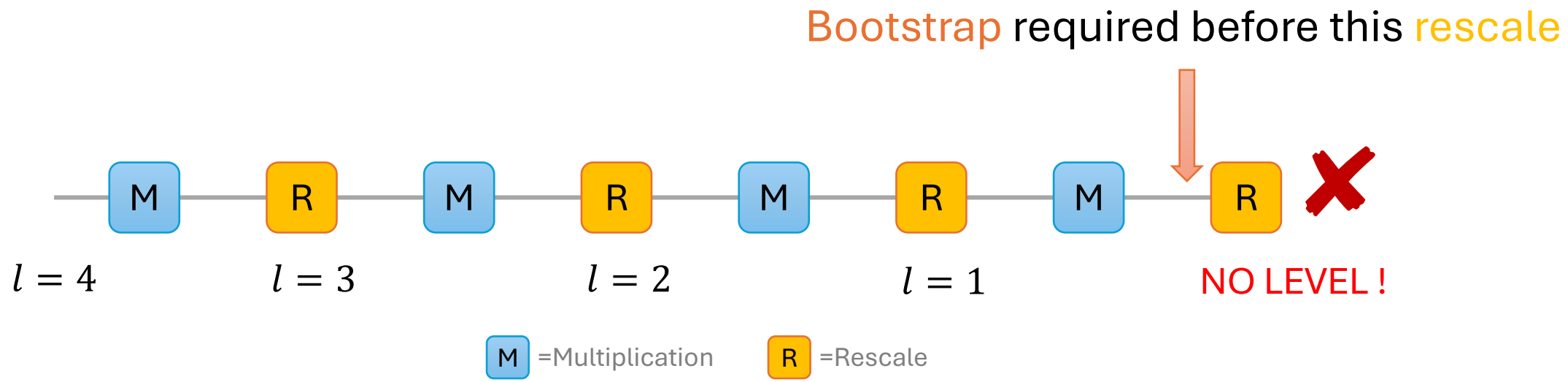
Bootstrap placement is important for fast CKKS-FHE execution

Only Rescale can reduce scale



A valid CKKS program must place rescale along multiplication paths

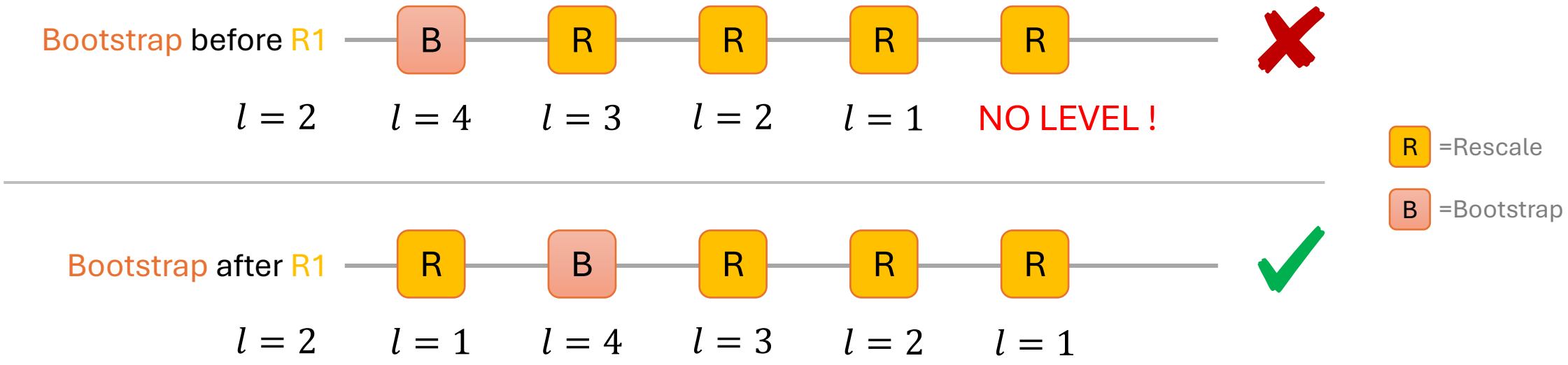
Rescale couple with Bootstrap



Rescale Location → Level Consumption → Bootstrap Requirement

Rescale placement determines when bootstrapping is needed

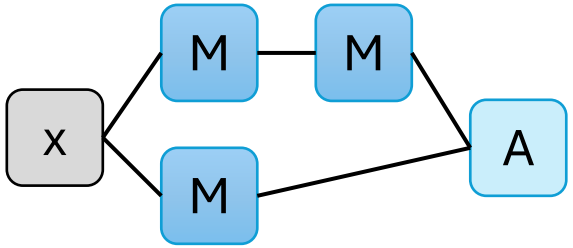
Rescale couple with Bootstrap



Bootstrap placement determines subsequent rescale budgets

Joint Bootstrap and Rescale Placement

INPUT



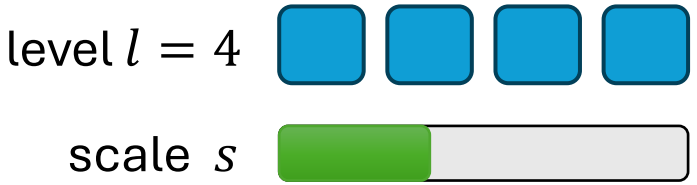
Computation Graph

DECISION



Bootstrap + Rescale Placement

CONSTRAINT

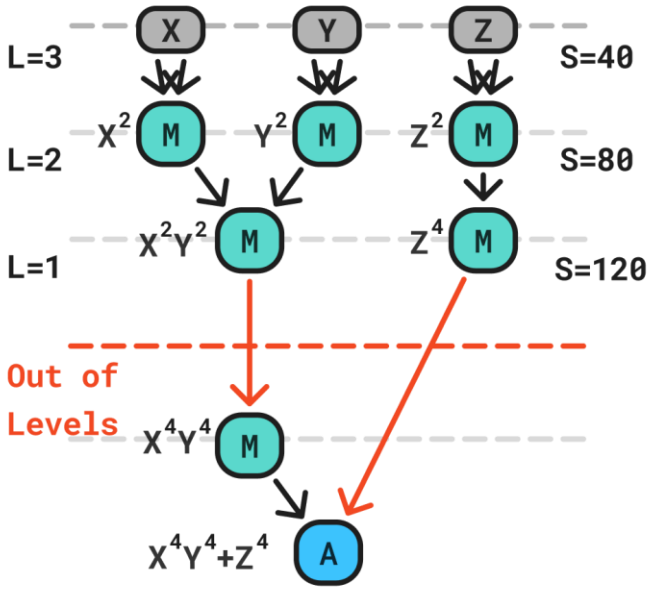


Valid level and scale for all operations

A =Addition M =Multiplication R =Rescale B =Bootstrap

GOAL Minimize predicted end-to-end latency

Gaps in prior work



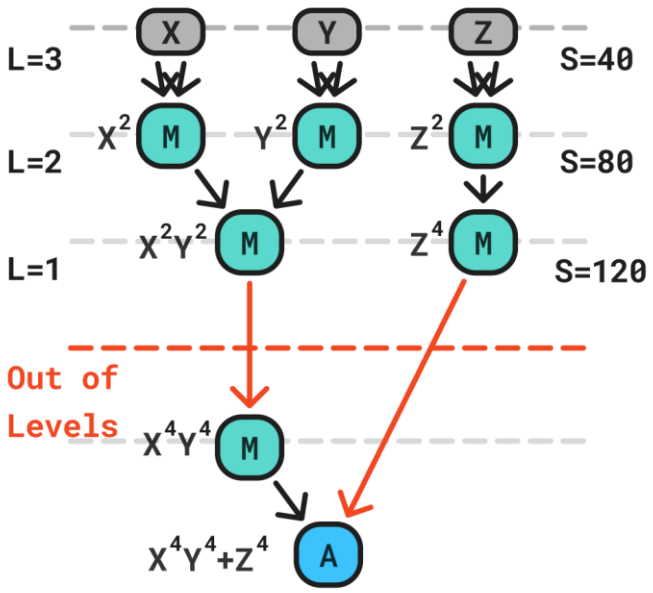
- Orion
- Constrained parameters
- Suboptimal solutions

Orion

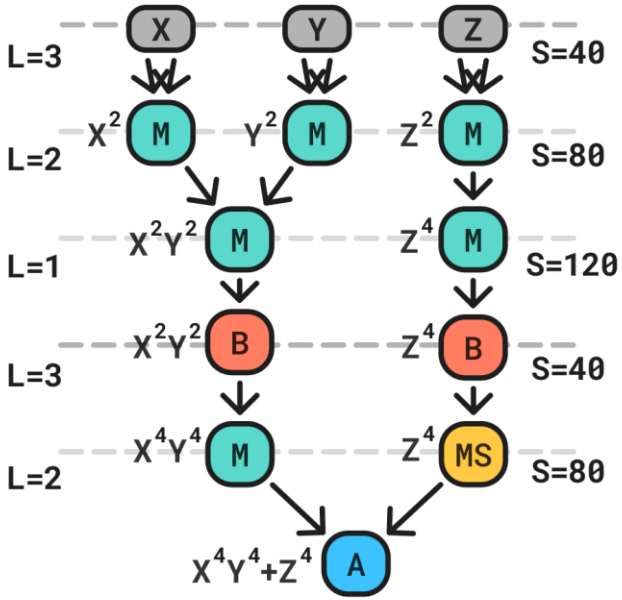
Simplifying Assumptions

- Rescale immediately after each multiplication.
- All polynomial evaluations must fit within the level budget

Gaps in prior work



- Orion
- Constrained parameters
- Suboptimal solutions

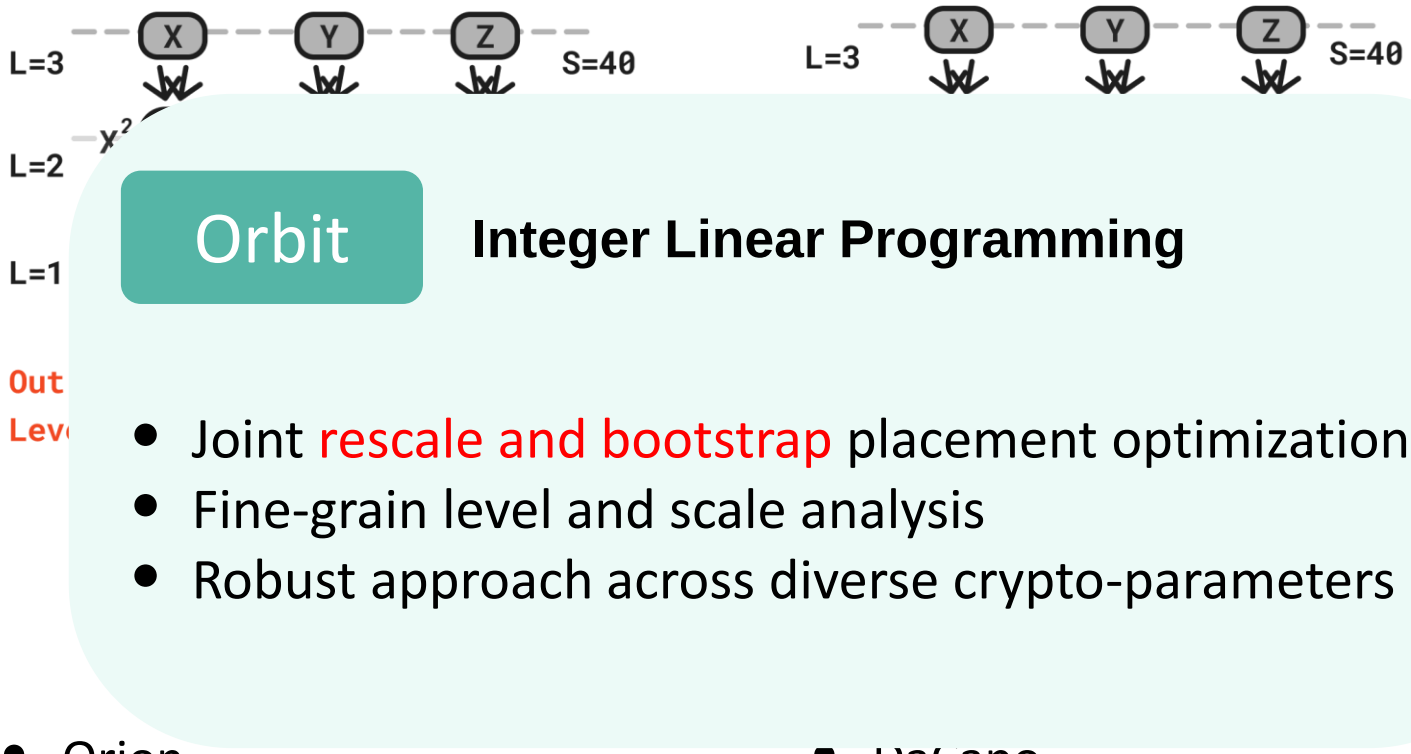


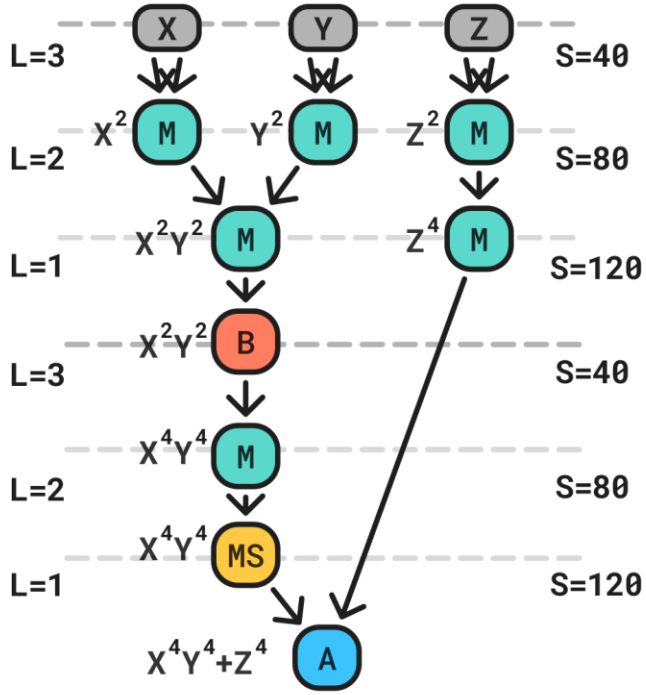
- DaCapo
- Depth-based heuristics
- Consumes 2 bootstraps

DaCapo **Depth-based Heuristics**

- Depth-wise search, bootstrap along depth boundaries
- Doesn't work as well on irregular DAG structures

Our approach: Orbit





- Orion
- Constrained parameters
- Suboptimal solutions

- DaCapo
- Depth-based heuristics
- Consumes 2 bootstraps

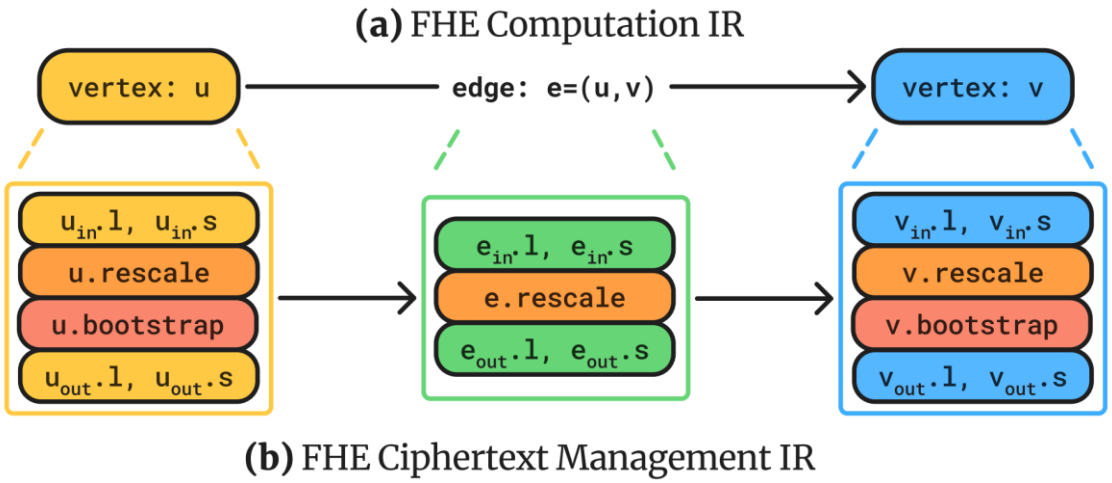
- Our approach: Orbit
- Integer Linear Programming
- Consumes 1 bootstrap

Orbit Techniques

- Jointly optimize rescale and bootstrap → Optimize overall latency
- Integer Linear Programming (ILP) → Optimal
- DAG Reduction Techniques → Scalable
  - Compression
  - Partitioning & Iterative Merging
  - Bypass Handling

ILP Formulation

- Orbit's ILP Formulation: Distill the Essential
  - 7-10 variables / vertex
  - HEILP: $O(L^2)$ variables / vertex



Variables

Per operation: level & scale, bootstrap flag, rescale count

Constraints

Rules to enforce CKKS semantics: level and scale effects of each op

Objective

Minimize overall latency based on a profiled cost-model

Challenge: Scaling Integer Linear Programming

Naive Encoding

≈ 1,000,000

constraints for a single CNN

Solve Time

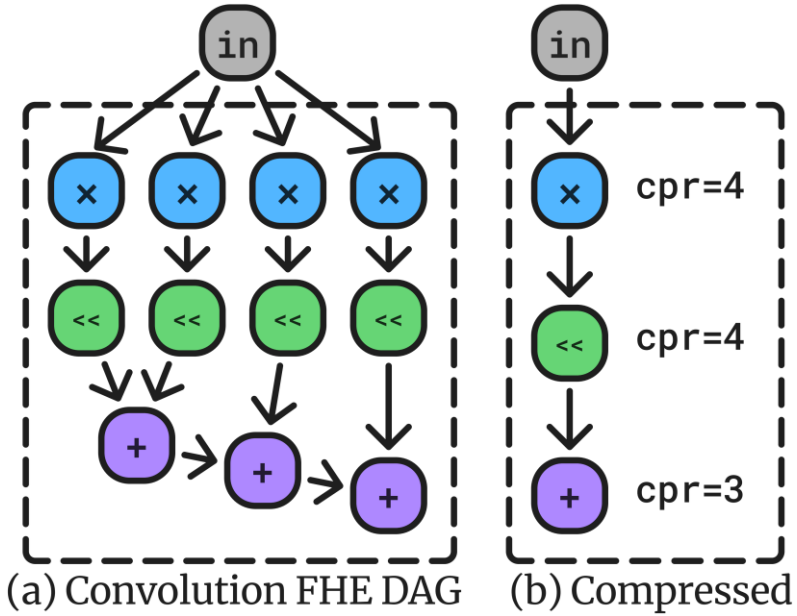
24 h +

solver still didn’t finish

How do we shrink every solver call — without giving up efficient placement?

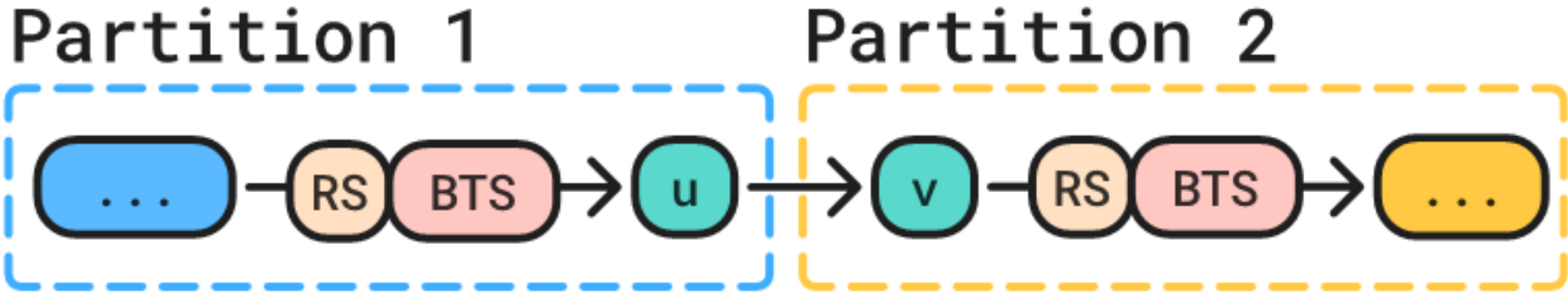
DAG Reduction —— Compression

- Motivation: Many identical parallel computation in convolution
- In optimal solution, their placement should be identical!



DAG Reduction —— Partitioning

- Motivation:
  - Activation function: large function, appear many times, single-in single-out (SISO)
  - Can we solve them separately, and then merge the results?

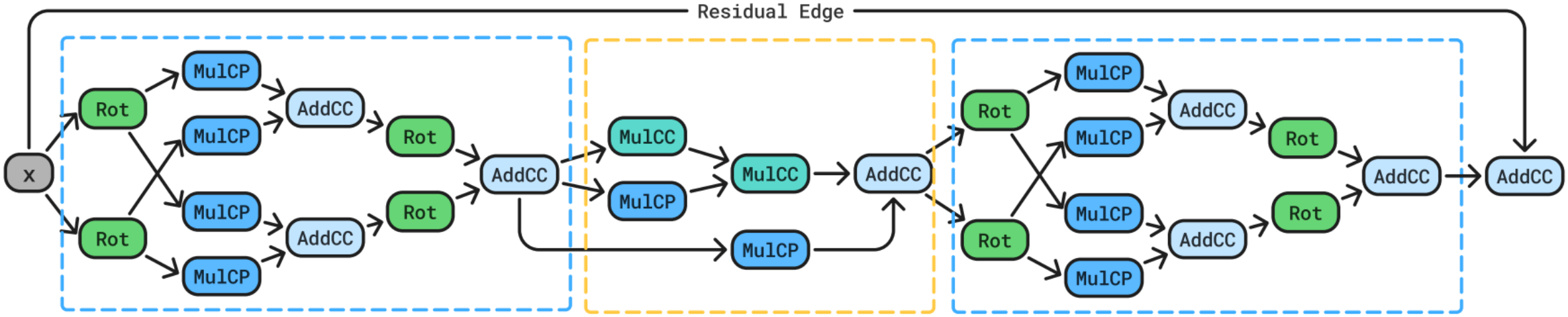


DAG Reduction —— Partitioning / QBP

- How to merge the results?
- Consider all possible boundary node states
- Quadradic Behavior Profile: store the results
  - Cache results for identical partitions
- Dynamic Programming for merging

DAG Reduction —— Bypass Handling

- Problem: ResNet has bypass edges
  - Activation function inside cannot be split out
  - Fewer identical partitions
  - Make the structure irregular



DAG Reduction

DAG Compression

Parallel Structure
Compress identical vertices into a single ILP node.

2.6× fewer vertices
provably optimal

Partitioning + QBP

Sequential Structure
Split along SISO boundaries
Cache solutions with QBP
Stitch together with DP

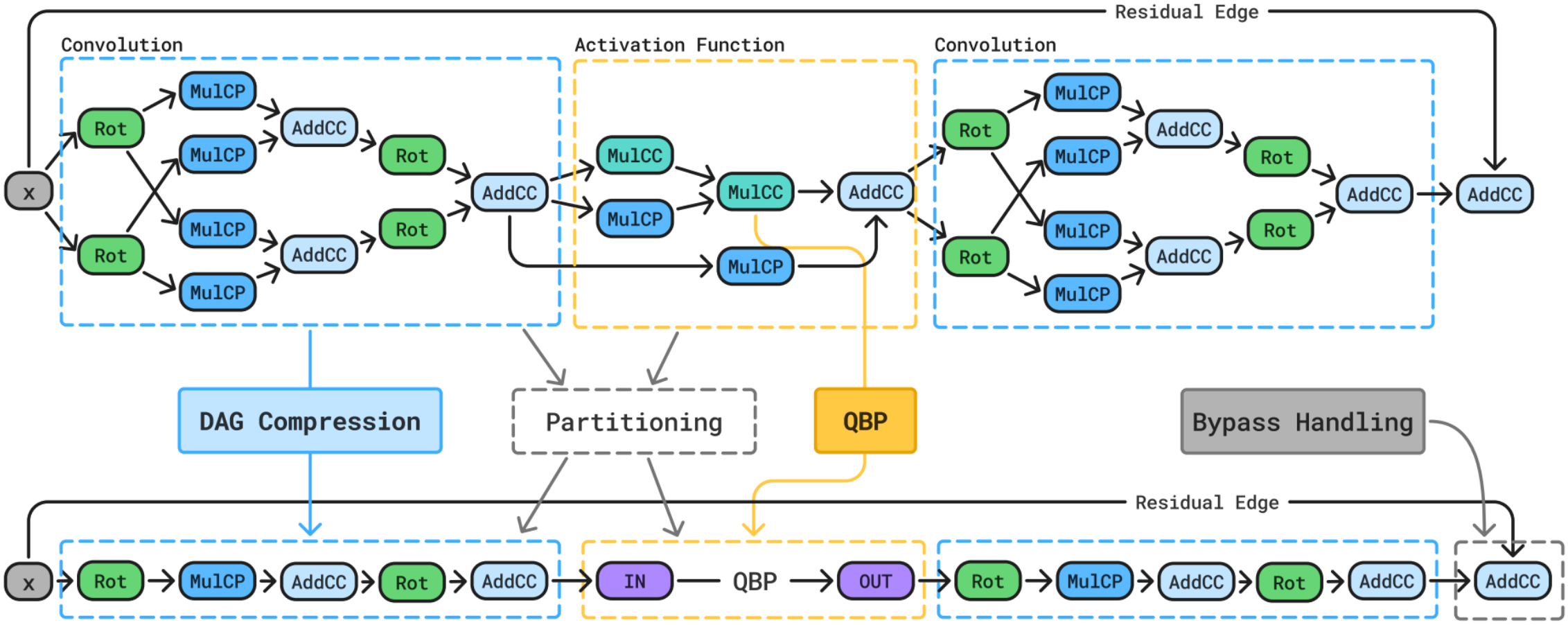
6× faster compilation
reusable across layers

Bypass Handling

Special cases for handling bypass and residual edges.

5.2 – 8.6× improvement
extend to irregular structure

DAG Reduction



Evaluation —— Compile Time / Accuracy

Compile Time

< 6 min

Across all models and configurations
Circuit size: $\sim 10^5$

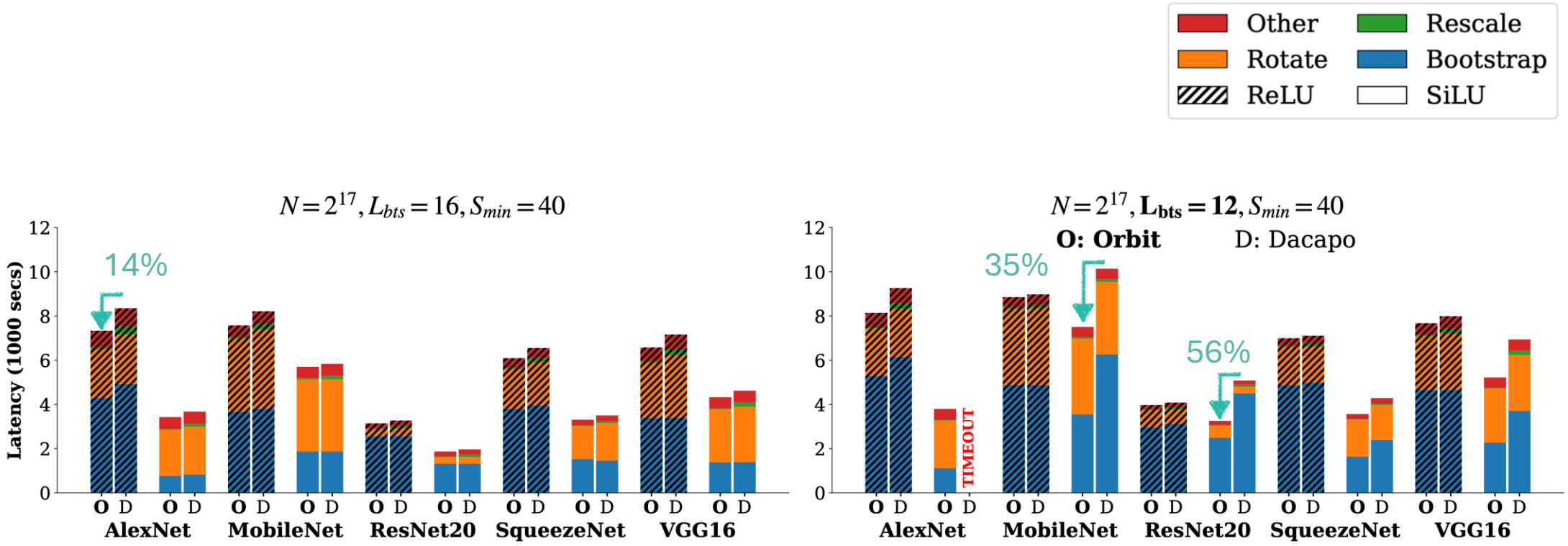
Model Accuracy

≈ 0.3%

of plaintext accuracy

Much better than 24h+

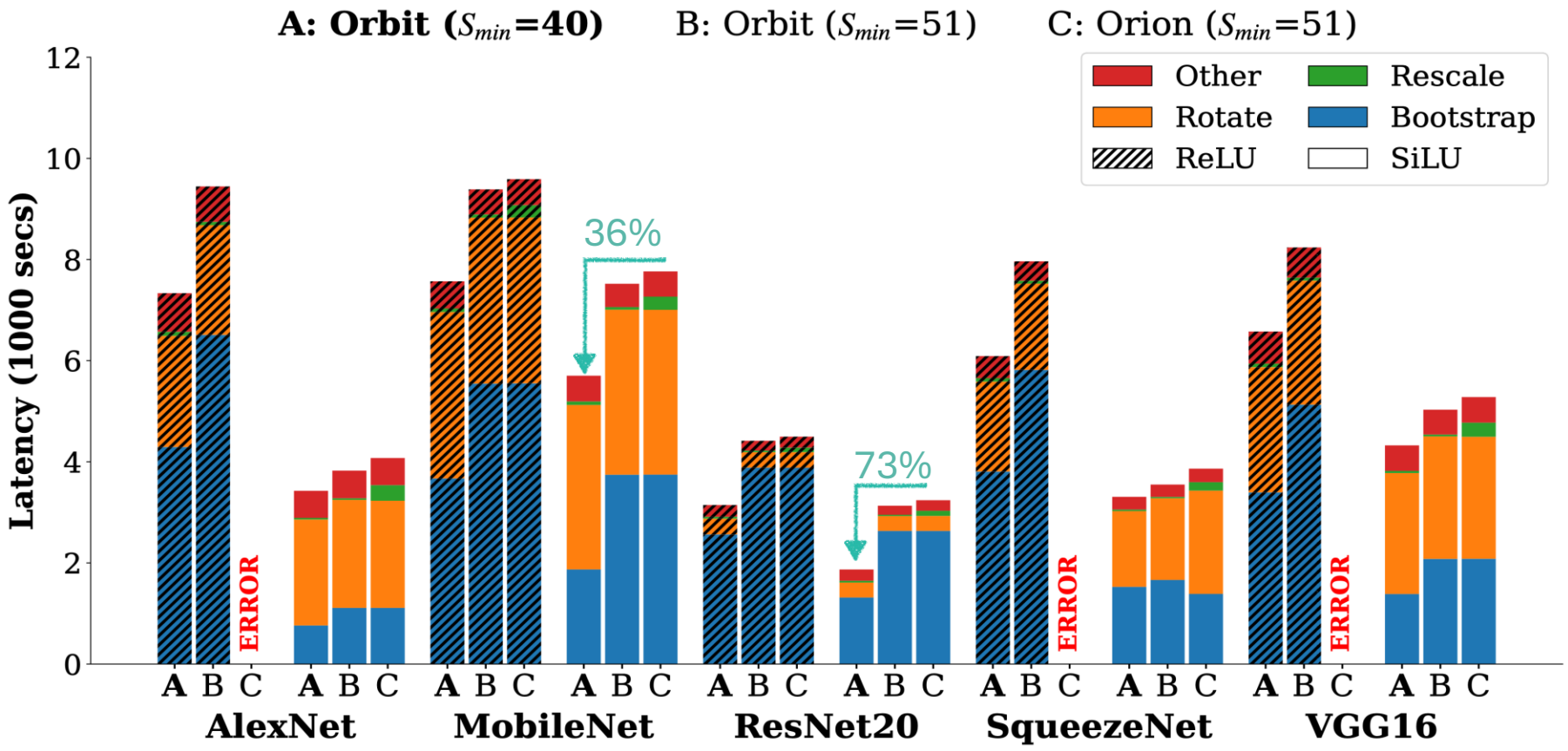
Evaluation —— Orbit vs. DaCapo



Orbit vs. DaCapo on DaCapo's Configurations

Orbit vs. DaCapo on smaller level budgets

Evaluation —— Orbit vs. Orion



Orbit vs. Orion on different waterline scales

Future Directions

Richer Rescale + Bootstrap Operations

Composite scaling
Grafting
Functional bootstrapping

High-level Structure

Preserve tensor/partition structure
Faster placement search

Integration with HEIR

FHE compiler ecosystem



Orbit

- Jointly optimize rescale and bootstrap → Optimize overall latency
- Integer Linear Programming (ILP) → Optimal
- DAG Reduction Techniques → Scalable
  - Compression
  - Partitioning + QBP
  - Bypass Handling
- Evaluation:
  - Compilation time within 6 min for circuits with ~10^5 gates
  - Up to 73% improvement

Link to our repo!