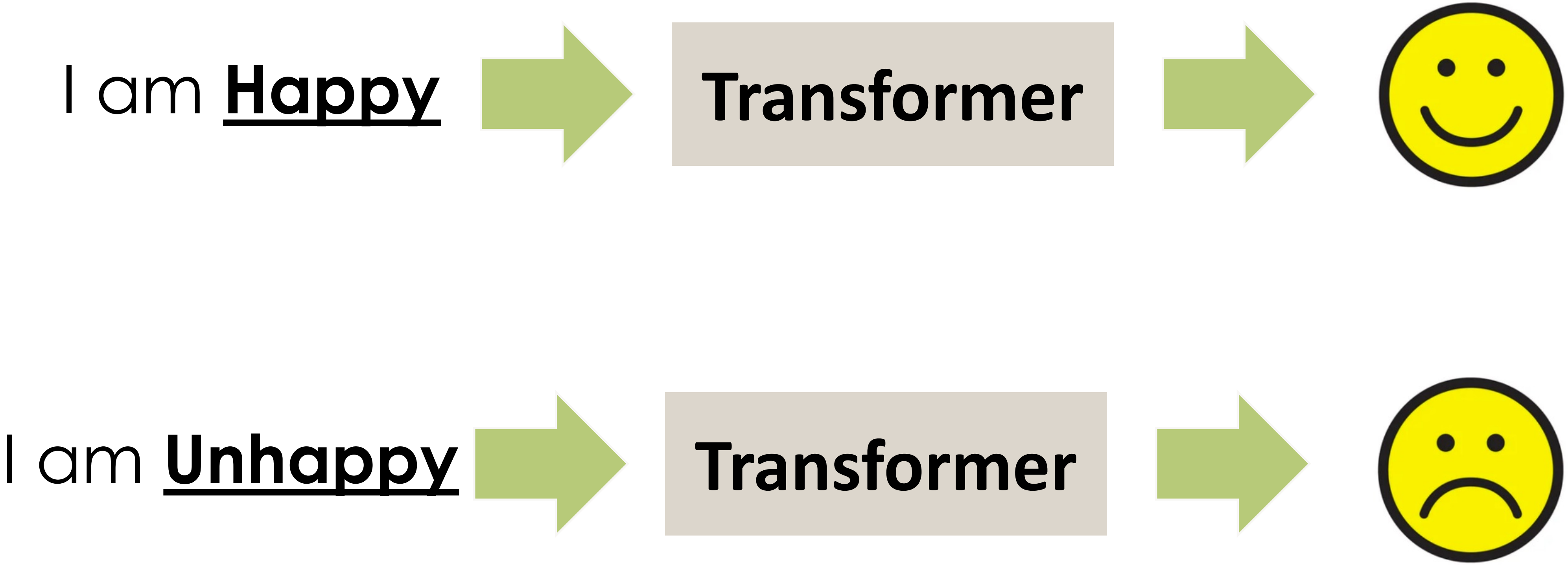
Faith: An Efficient Framework for Transformer Verification on GPUs

Boyuan Feng, Tianqi Tang, Yuke Wang, Zhaodong Chen,
Zheng Wang, Shu Yang, Yuan Xie, Yufei Ding

UC SANTA BARBARA

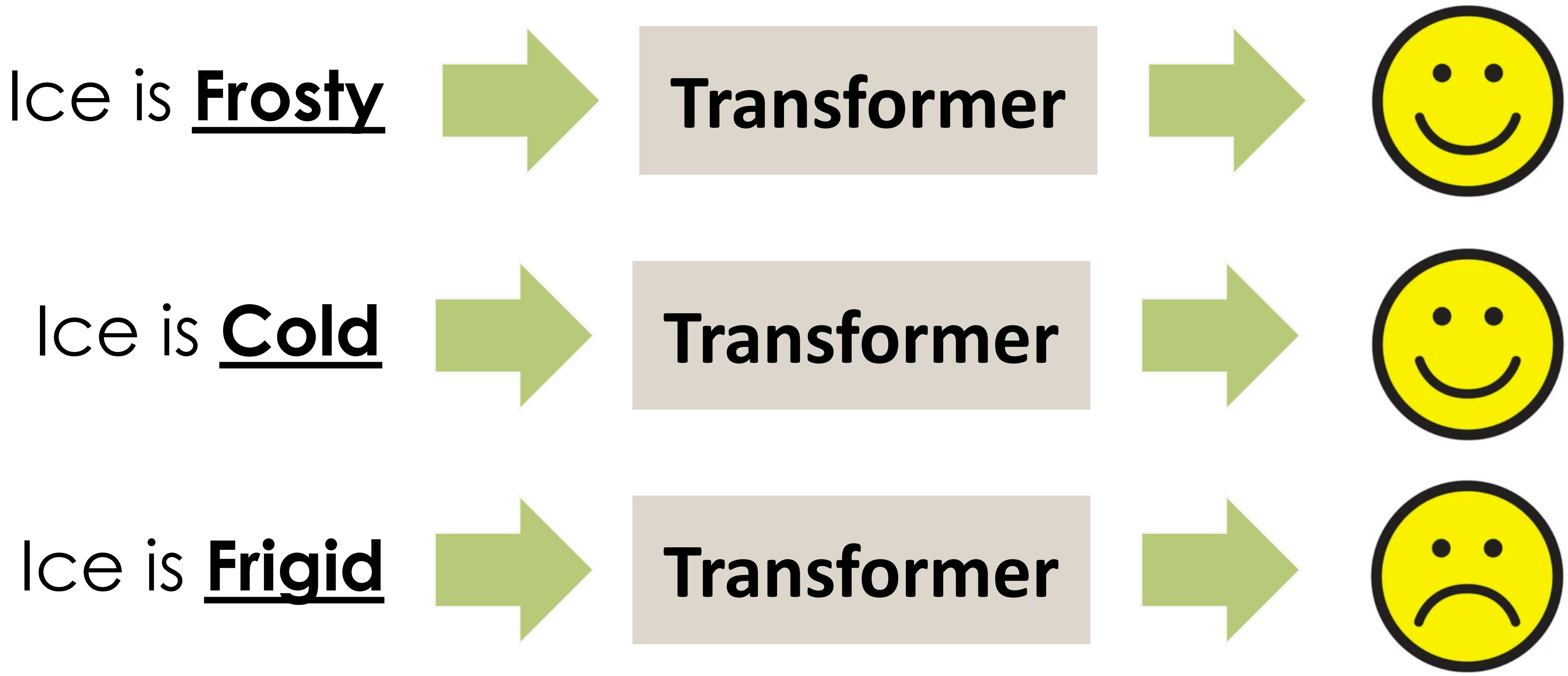
Motivation: Transformer Applications

- Sentiment Analysis



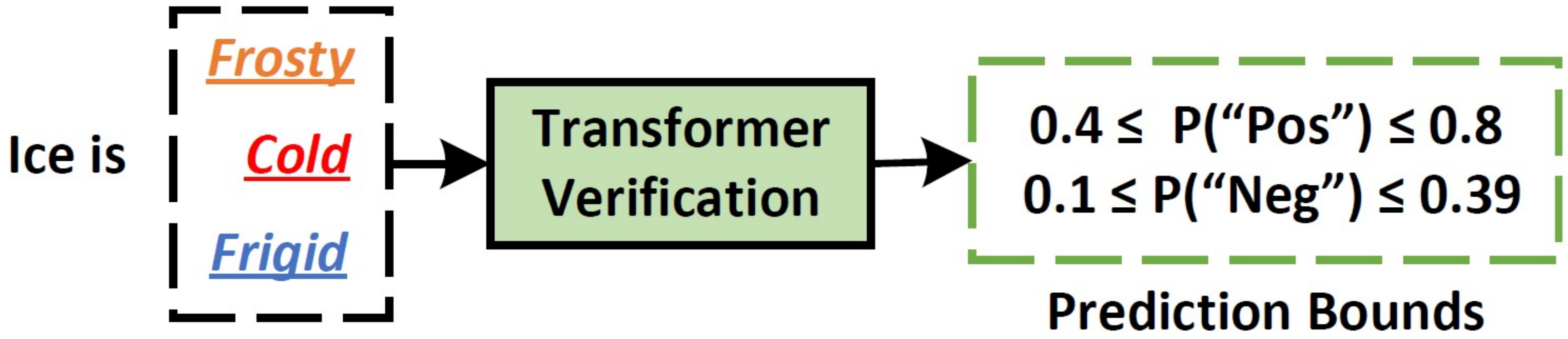
Motivation: Security Concerns

- **Synonym Substitution Attack**



UC SANTA BARBARA

Motivation: Transformer Verification



Performance challenge:

- Second-level latency of transformer verification
- v.s. Millisecond-level latency of standard transformers

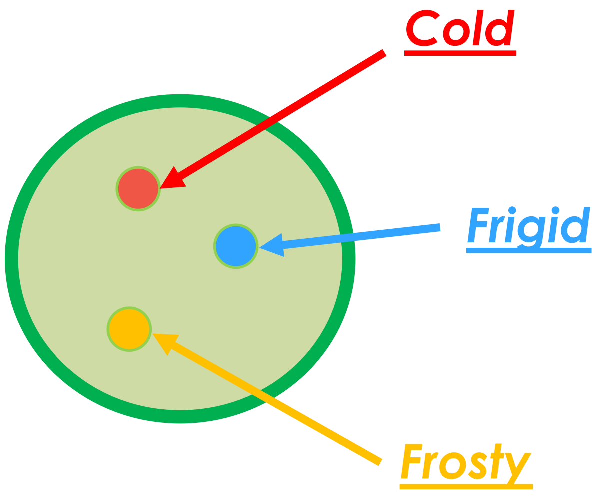
Need efficiency!

Challenges: Unique Computation Patterns

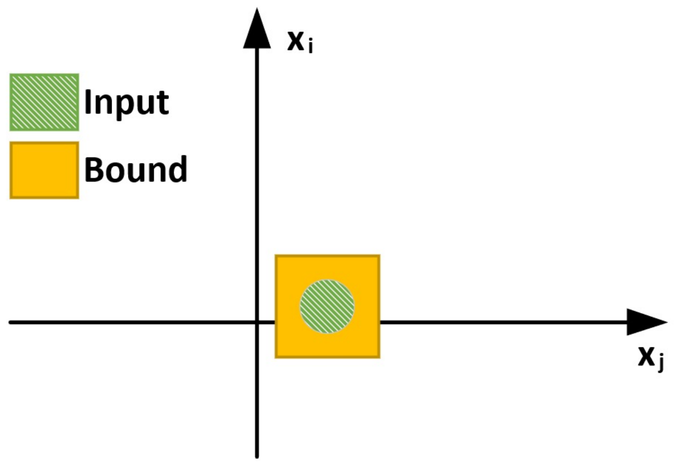
Irregular & 50% sparsity

- Heavy redundancy with dense computation
- Too dense for sparse computation (e.g., cuSPARSE)

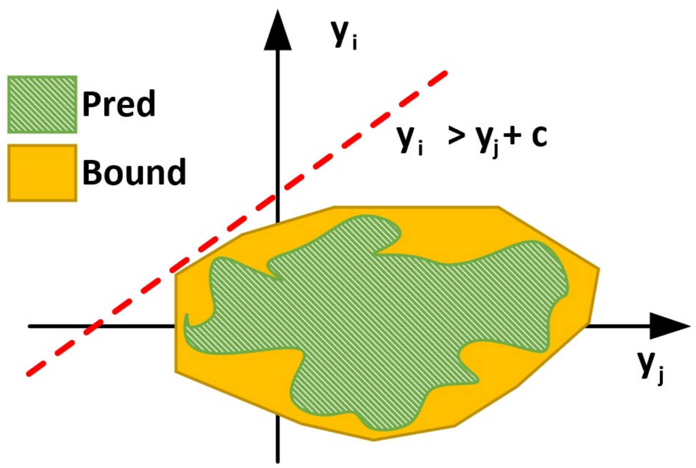
Computation Patterns



Word Embeddings of Synonyms



Input linear bound



Output linear bound

Computation Patterns

Given a linear layer: $y = 2 * x_1 - x_2$

Transformer Inference:

$x_1 = 3, x_2 = 1$ ⇨ $y = 2 * x_1 - x_2 = 5$

Transformer Verification:

$$1 \leq x_1 \leq 4$$
$$-2 \leq x_2 \leq 4$$

⇨

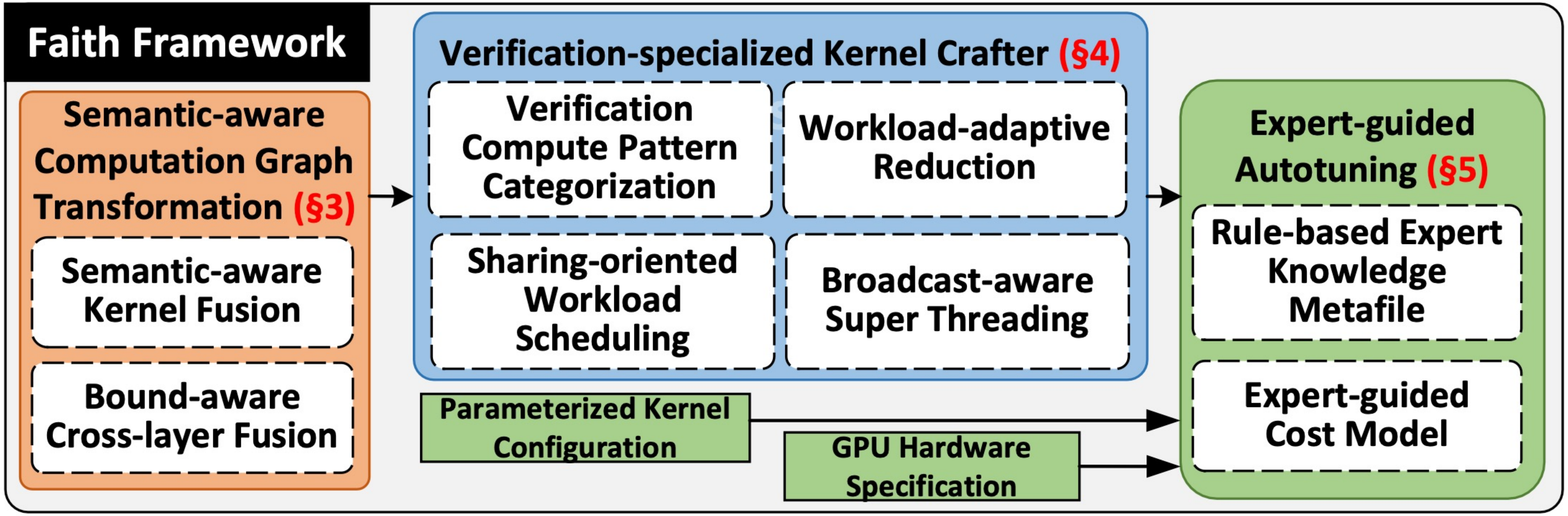
$$2 \leq 2 * x_1 \leq 8$$
$$-4 \leq -x_2 \leq 2$$
$$-2 \leq y \leq 10$$

High Irregularity!

Challenges

- **Lack of support for unique computing patterns**
  - Existing DL frameworks are designed for standard NN.
  - Verification shows different computing pattern.
- **Lack of framework support for verifying diverse NN layers.**
  - Transformer verification shows large diversity in the bound computation.
- **Lack of verification-specialized adaptability towards modern GPUs.**
  - Transformer verification involves memory-intensive operations.
  - Existing DL frameworks only focus on computation-intensive operations.

Overview

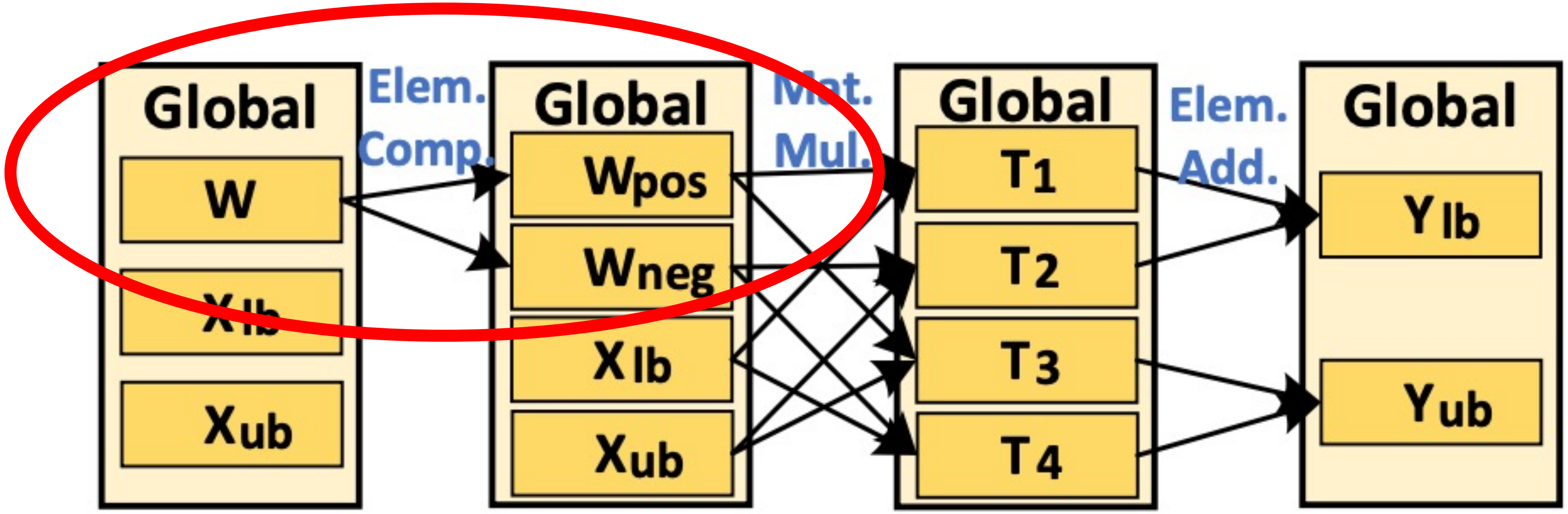


Faith Framework
Semantic-aware Computation Graph Transformation (§3)
Semantic-aware Kernel Fusion
Bound-aware Cross-layer Fusion
Verification-specialized Kernel Crafter (§4)
Verification Compute Pattern Categorization
Workload-adaptive Reduction
Sharing-oriented Workload Scheduling
Broadcast-aware Super Threading
Parameterized Kernel Configuration
GPU Hardware Specification
Expert-guided Autotuning (§5)
Rule-based Expert Knowledge Metafile
Expert-guided Cost Model

Sematic-aware Computation Graph Transformation

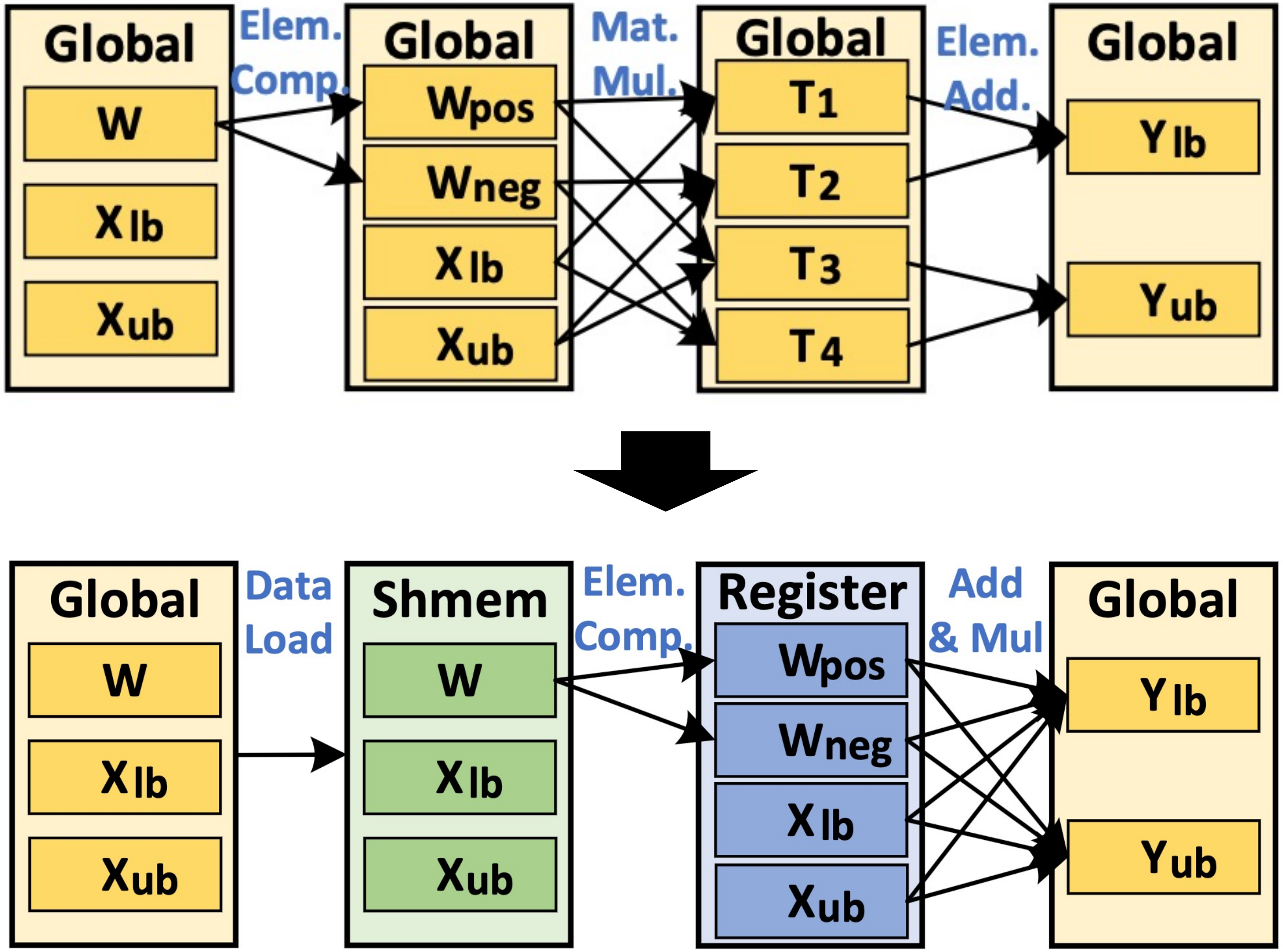
Sematic-aware Computation Graph Transformation

Intensive global memory access

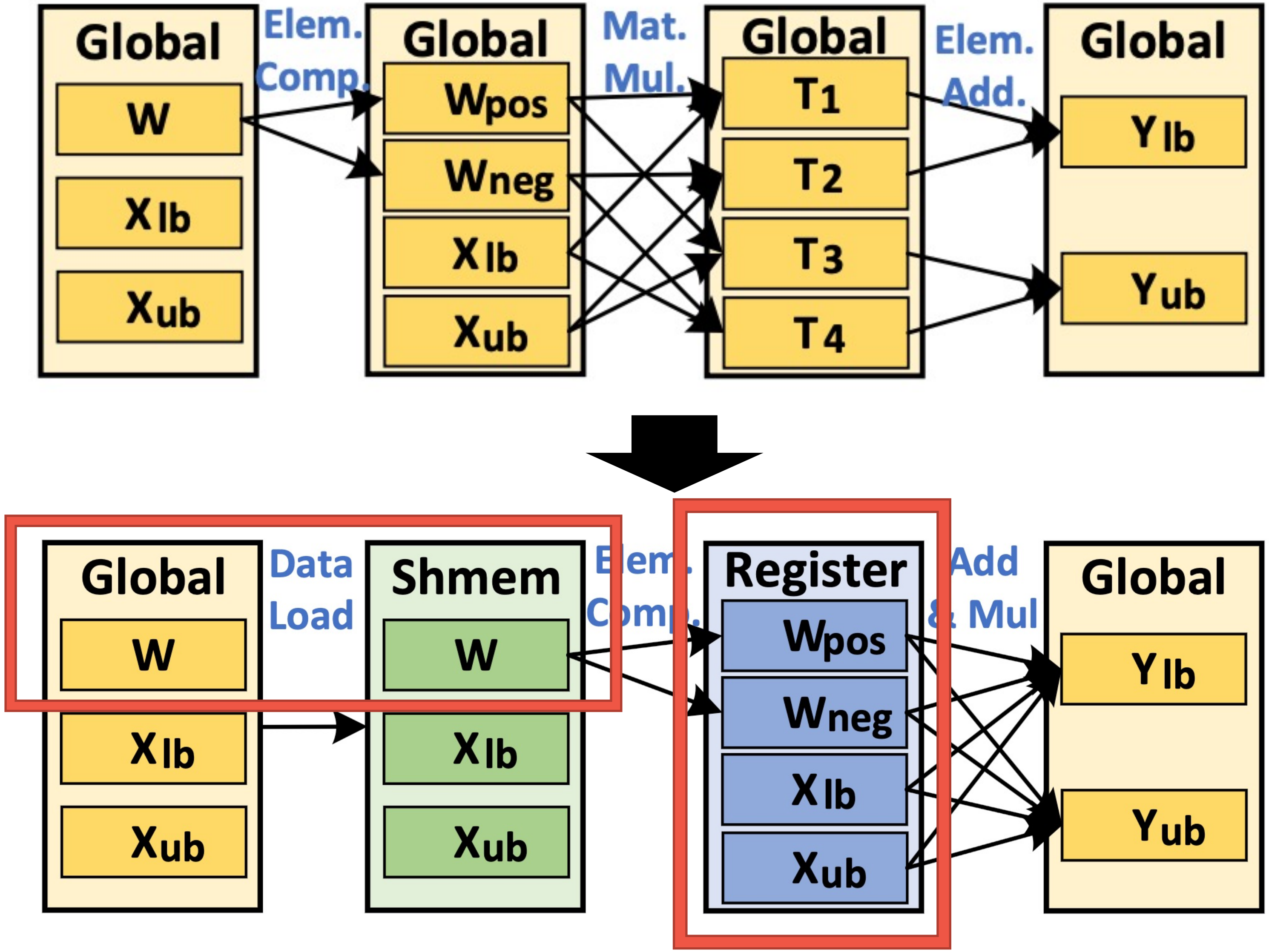


Memory Access pattern of transformer verification

Sematic-aware Computation Graph Transformation

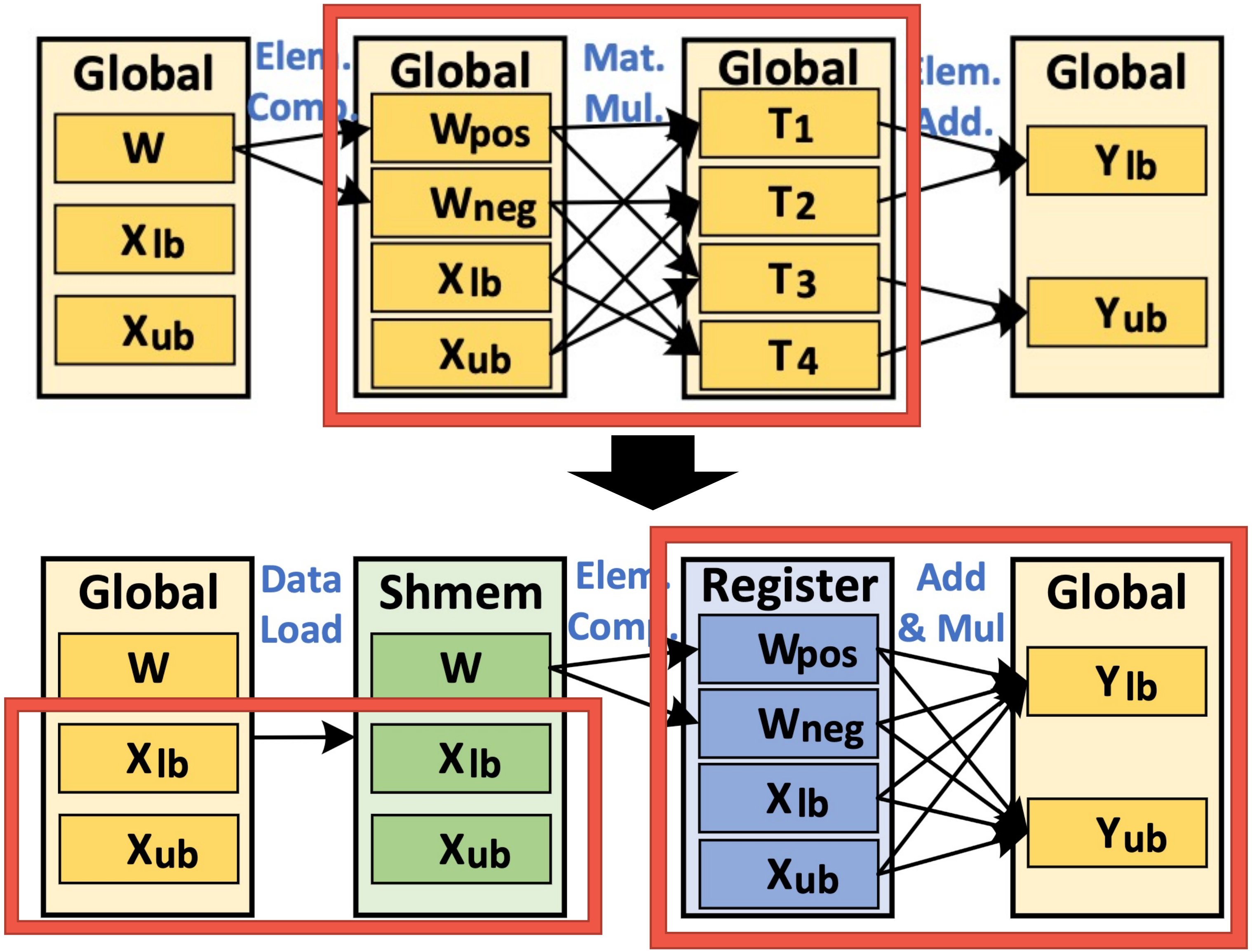


Sematic-aware Computation Graph Transformation



UC SANTA BARBARA

Sematic-aware Computation Graph Transformation

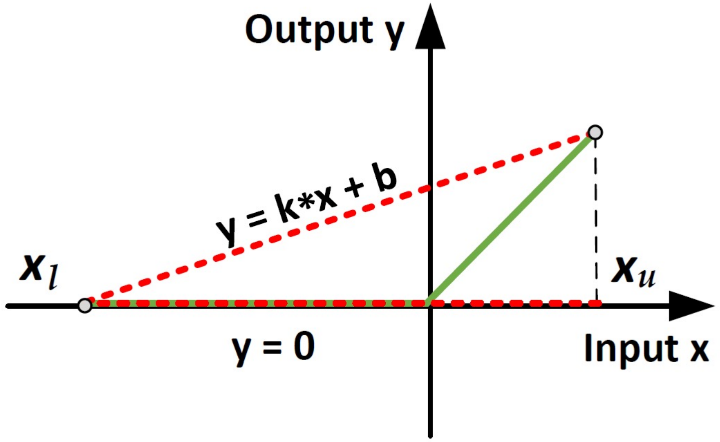


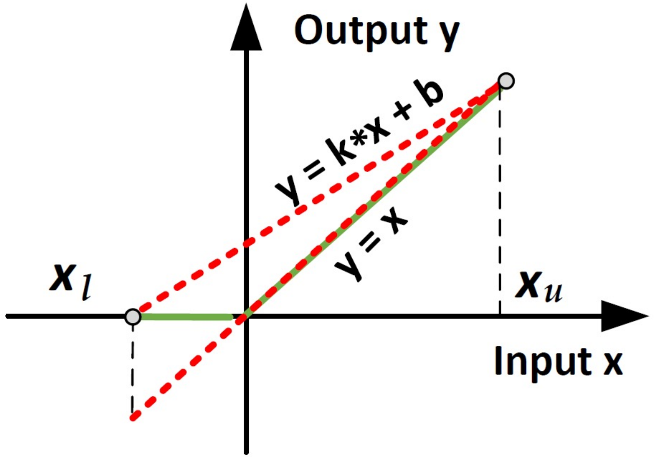
Verification-Specialized Kernel Crafter

Diversity across Verification Designs

- **Adaptive to Input Bounds & Operators**

ReLU

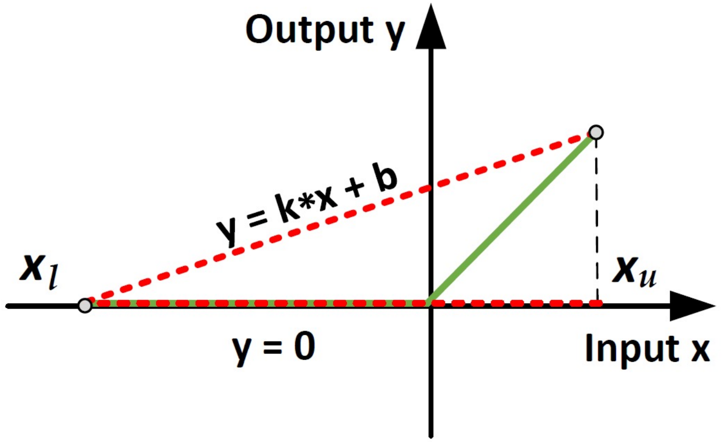


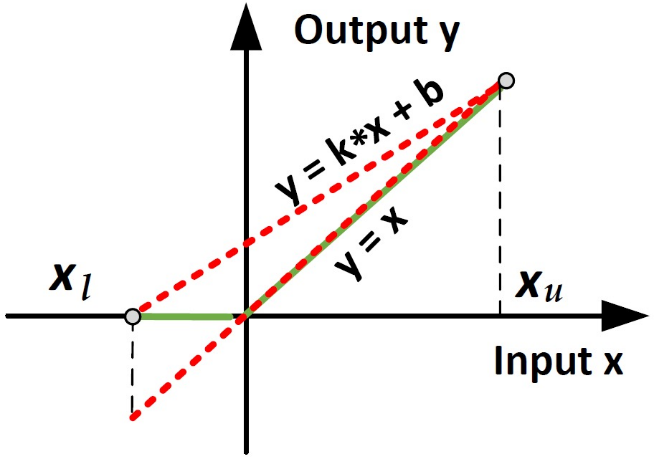


Diversity across Verification Designs

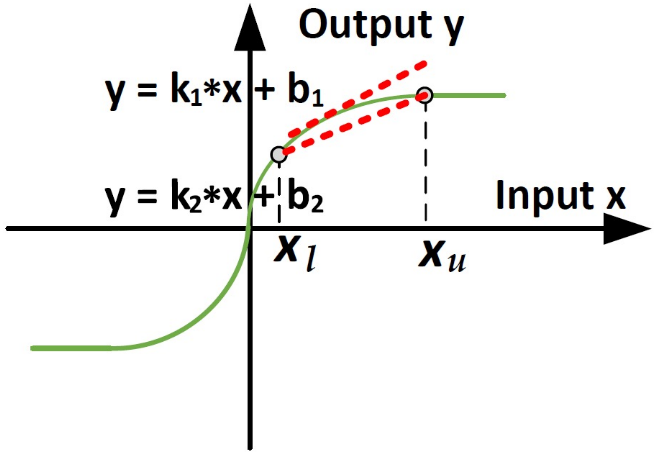
- **Adaptive to Input Bounds & Operators**

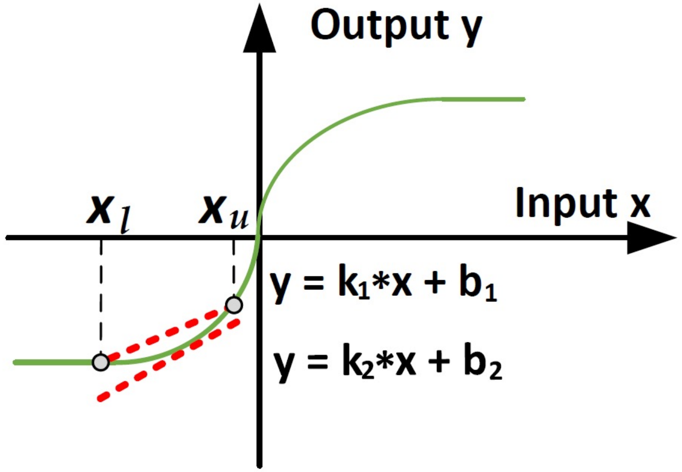
ReLU





Tanh





Hard to Optimize Individual Operators due to Diversity!

Verification Pattern Categorization

Key Insight:

Optimize **Computation Patterns** instead of concrete **Operator Verification Deigns**

Generalized Vector Reduction

$$y_i = reduction(\vec{x}_i) = \sum_{j=1}^{n} f(x_{i,j}), \quad i \in \{1, 2, \cdots, m\}$$

Generalized Elementwise Multiplication

$$y_{i,j} = f(l_{i,j}, u_{i,j}) * x_{i,j}, \quad i \in \{1, 2, \cdots m\}, j \in \{1, 2, \cdots, n\}$$

Generalized Scalar-Vector Multiplication

$$\vec{y}_i = f(s_i) * \vec{x}_i = [f(s_i) * x_{i,1}, f(s_i) * x_{i,2}, \cdots, f(s_i) * x_{i,n}],$$

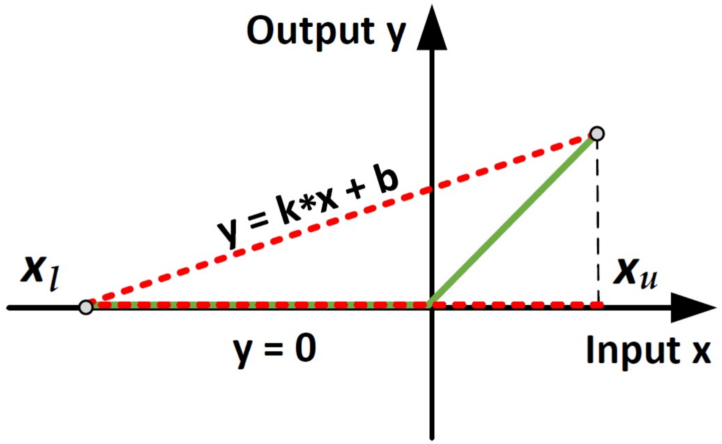
Verification Pattern Categorization

Key Insight:

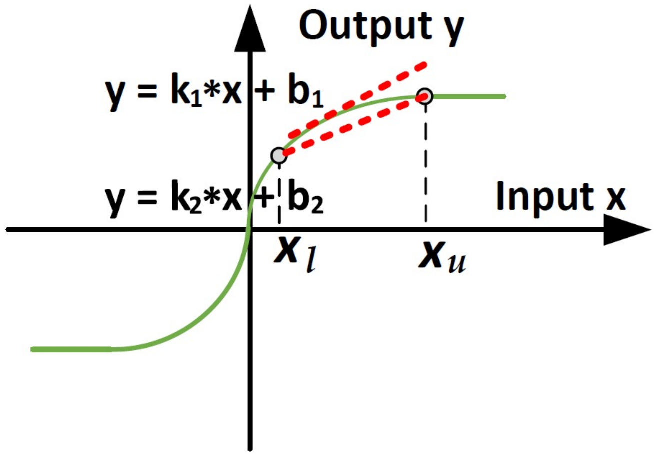
Optimize **Computation Patterns** instead of concrete **Operator Verification Deigns**

Generalized Elementwise Multiplication

$$y_{i,j} = f(l_{i,j}, u_{i,j}) * x_{i,j}, \quad i \in \{1, 2, \cdots m\}, j \in \{1, 2, \cdots, n\}$$



ReLU



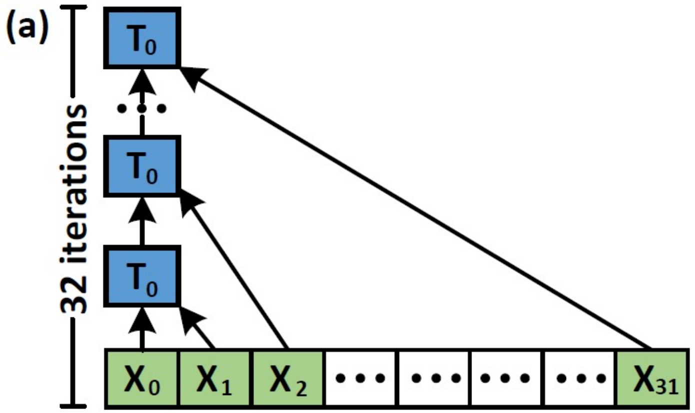
Tanh

Workload Adaptive Reduction

Generalized Vector Reduction

$$y_i = reduction(\vec{x}_i) = \sum_{j=1}^{n} f(x_{i,j}), \quad i \in \{1,2,\cdots,m\}$$

- Widely exists when verifying various operators
- Naïve approach: Sequential Mode

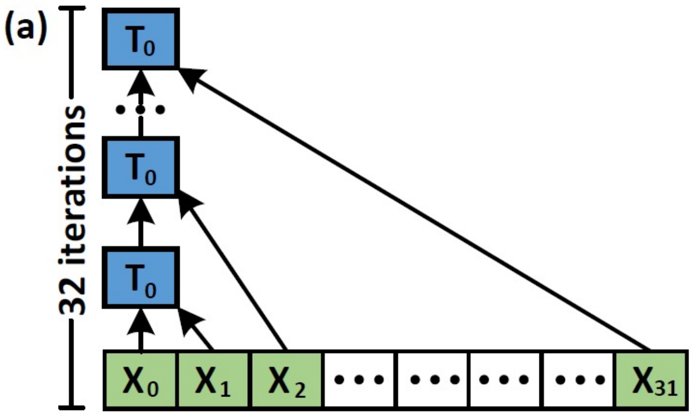


Sequential Mode
- 1 thread for 32 values
- 32 iterations
- Low parallelism

Workload Adaptive Reduction

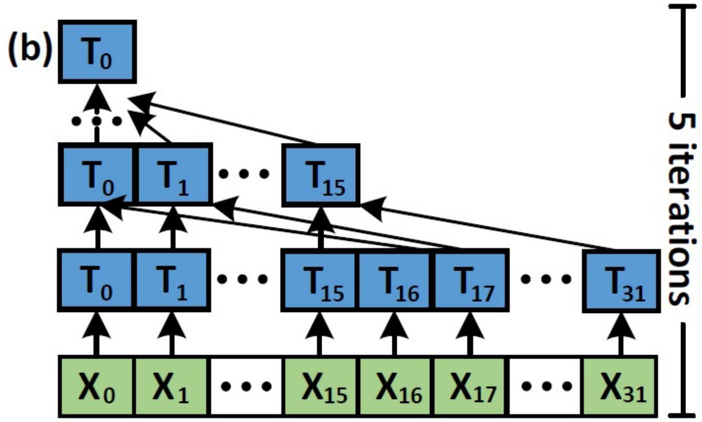
Generalized Vector Reduction

$$y_i = reduction(\vec{x}_i) = \sum_{j=1}^{n} f(x_{i,j}), \ \ i \in \{1, 2, \cdots, m\}$$



Sequential Mode

- 1 thread for 32 values
- 32 iterations
- Low parallelism



Parallel Mode

- Exploit GPU hardware properties
- 32 threads for 32 values via _shfl_down_sync
- 5 iterations
- High parallelism

Sharing-oriented Workload Scheduling

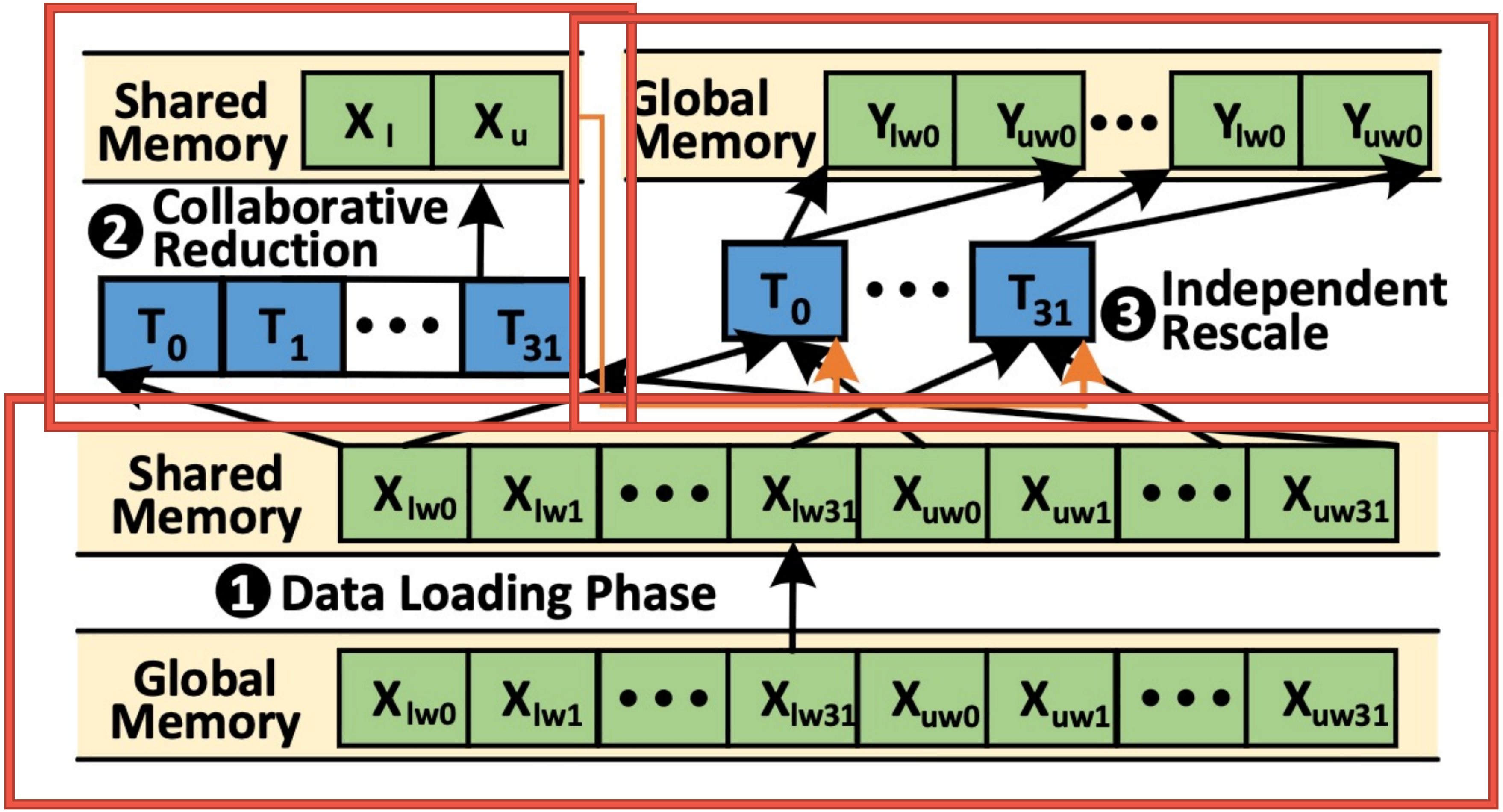
Problem:

Heavy memory overhead during verification

Key idea:

Exploit GPU memory hierarchies (i.e., register, shared memory, and global memory) to effectively reduce memory access.

Sharing-oriented Workload Scheduling



Expert-guided Autotuning Optimization

Expert-guided Autotuning Optimization

Goal:

Effectively incorporate hardware knowledge to find optimal operator implementations

Idea:

- Generate a metafile for each hardware on its properties
- Incorporate this metafile to a cost model for tuning verification operators

Expert-guided Autotuning Optimization

Rule-based Expert Knowledge Metafile

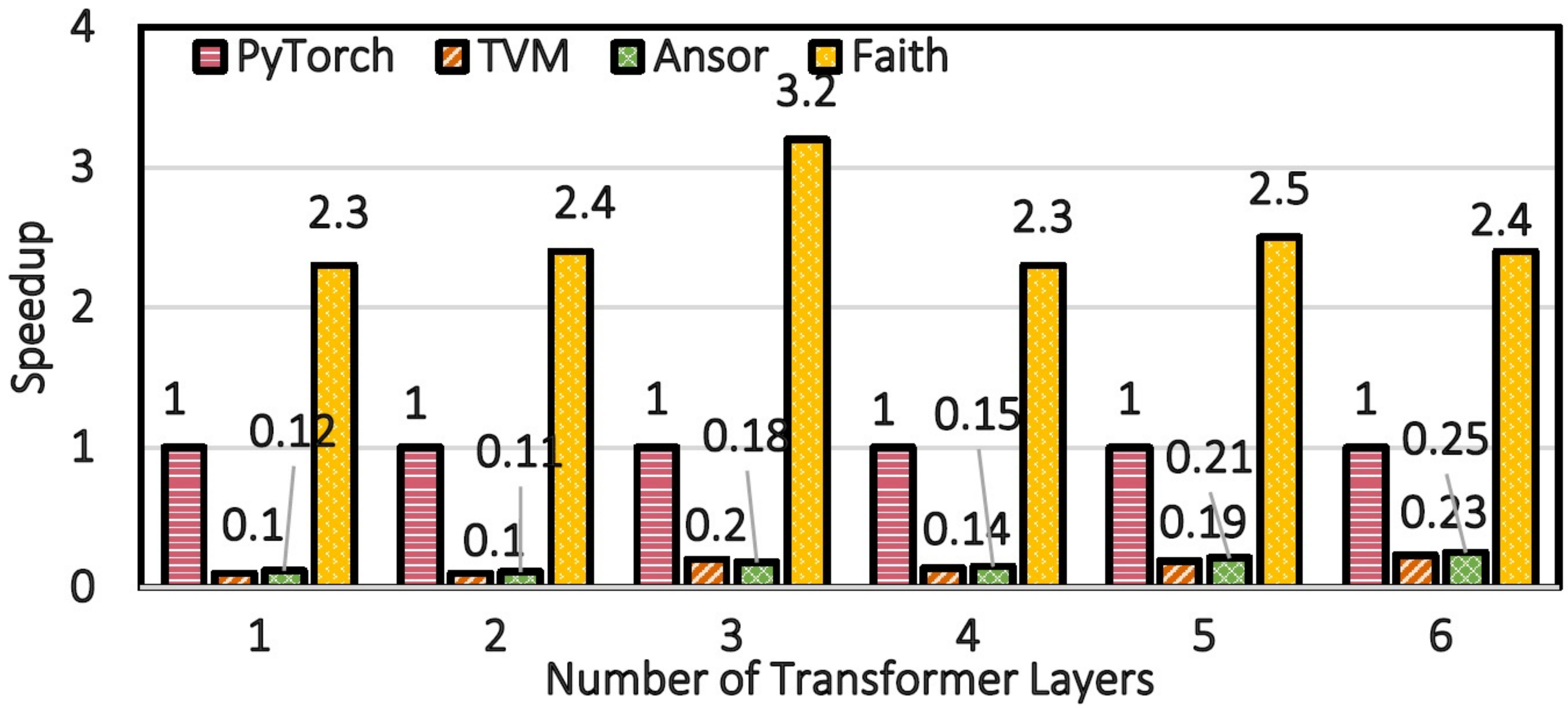
- **Hard rule** for hardware limitation (e.g., maximal shared memory size, maximal #register per thread)
- **Soft rule** for trade-off related to hardware properties (e.g., #SM, #threads per SM)

Expert-guided Cost Model

- **Phase-1**: Estimate shared memory and register usage & skip candidates that violates hard rules.
- **Phase-2**:
  - Train a cost model
  - Consume both soft rules for hardware properties and tuning knobs (e.g., tiling sizes)
  - Predict best candidates

Evaluation

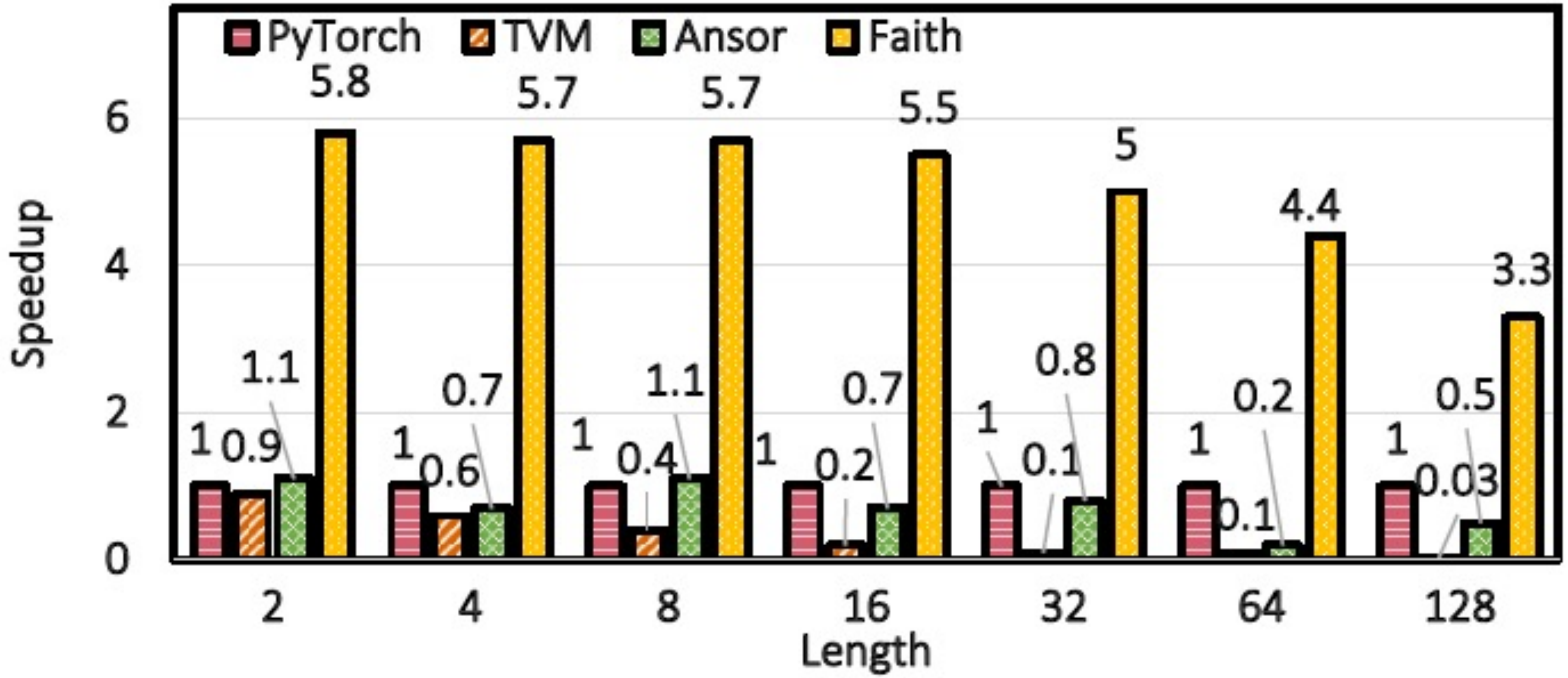
Evaluation: End-to-End Benefits



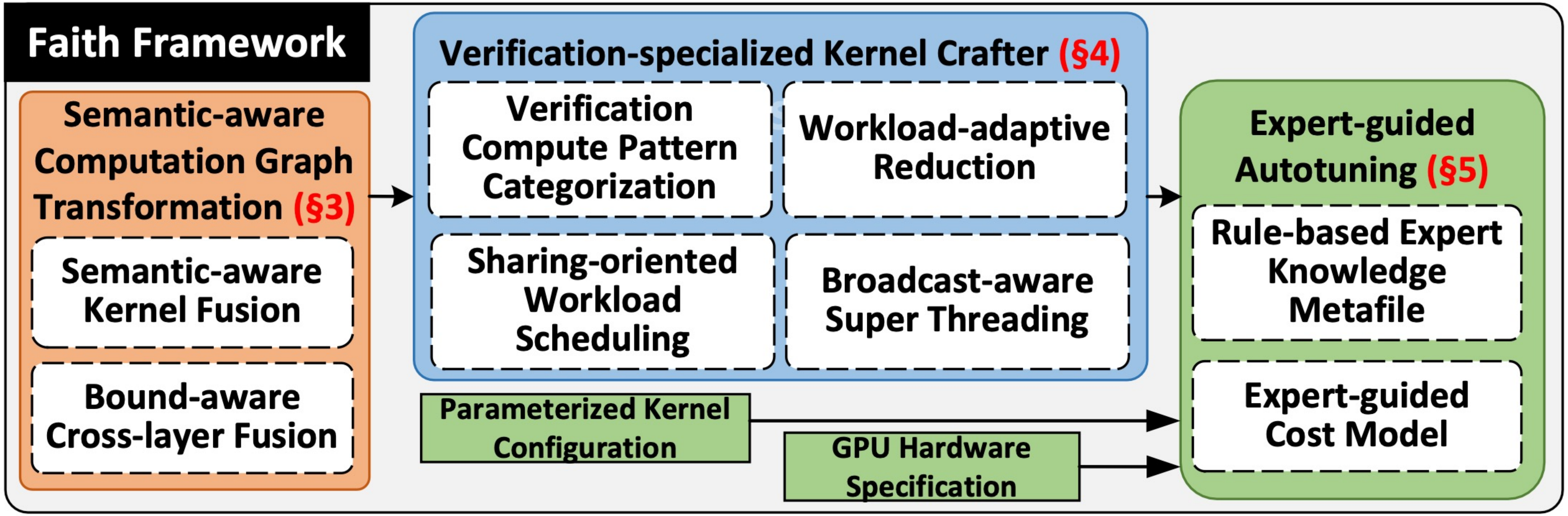
We achieve around 2.5x speedup over Pytorch

Evaluation: Per-layer Benefits

- **Matrix Multiplication**



Questions?



The project is open-sourced at:

https://github.com/BoyuanFeng/Faith

UC SANTA BARBARA