JULY 9–13, 2023

MOSCONE WEST CENTER

SAN FRANCISCO, CA, USA

Gamora: Graph Learning based Symbolic Reasoning for Large-Scale Boolean Networks

Nan Wu[1], Yingjie Li[2], Cong Hao[3], Steve Dai[4], Cunxi Yu[2], Yuan Xie[5]

[1]University of California, Santa Barbara, [2]University of Utah, [3]Georgia Institute of Technology,

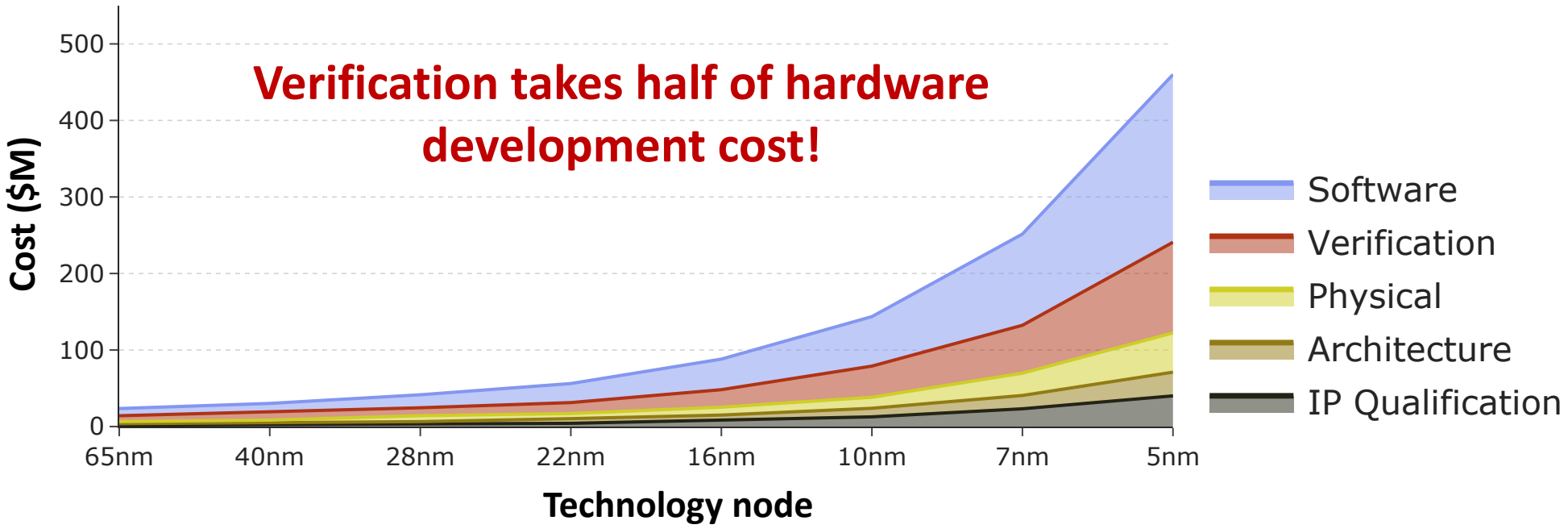
[4]NVIDIA, [5]Alibaba DAMO Academy

Outline

- Motivation
- Problem formulation
  - What is **Gamora**?
  - Why **Ga**mora?
  - How Ga**mora**?
- Evaluation
  - High reasoning accuracy
  - Scalability
  - Generalization

Verification is Important!

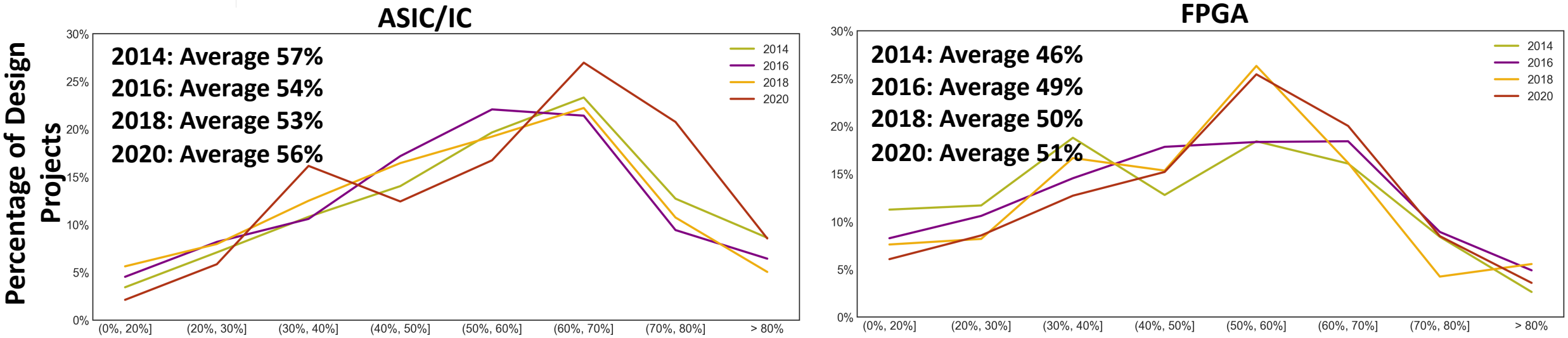
Hardware and Software Development Efforts for Advanced Designs



Frank Schirrmeister, Pete Hardee, Larry Melling, Amit Dua, and Moshik Rubin. Next Generation Verification for the Era of AI/ML and 5G.
https://dvcon-proceedings.org/document/next-generation-verification-for-the-era-of-ai-ml-and-5g/

Verification is Important!

Percentage of Project Time Spent in Verification



Verification dominates the project time!

Harry Foster. The 2020 Wilson Research Group Functional Verification Study.

Functional Unit Identification

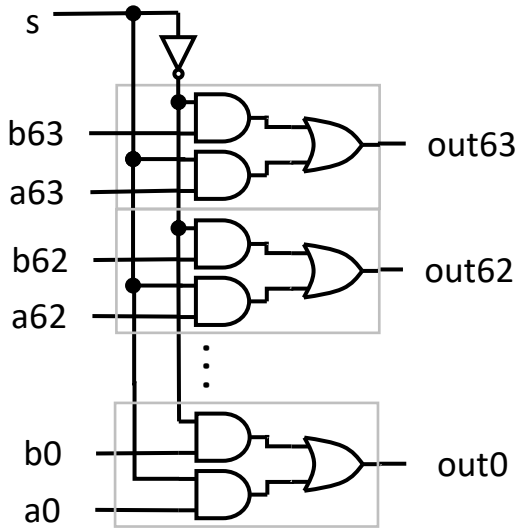
Identifying functional units from flattened gate-level netlists has wide applications

Functional Unit Identification

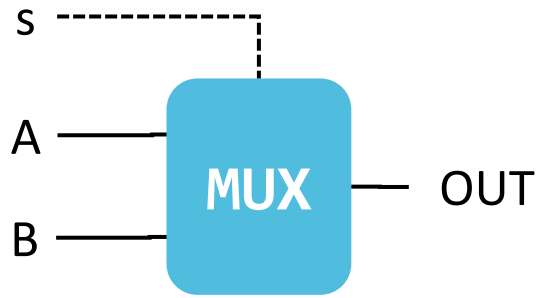
Identifying functional units from flattened gate-level netlists has wide applications

- Word-level abstraction for formal verification

Bit-level implementation



Word-level abstraction




```
int OUT = [
    s : A;
    1 : B;
];
```

Functional Unit Identification

Identifying functional units from flattened gate-level netlists has wide applications

- Word-level abstraction for formal verification
- Functional verification
  - [TCAD'17] Fast algebraic rewriting based on and-inverter graphs
  - [ICCAD'18] PolyCleaner: clean your polynomials before backward rewriting to verify million-gate multipliers
  - [DAC'19] RevSCA: Using reverse engineering to bring light into backward rewriting for big and dirty multipliers
  - [TCAD'19] Understanding algebraic rewriting for arithmetic circuit verification: a bit-flow model

Functional Unit Identification

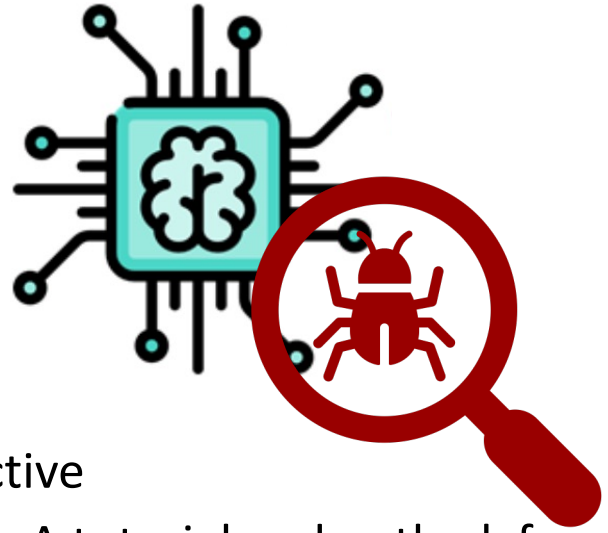
Identifying functional units from flattened gate-level netlists has wide applications

- Word-level abstraction for formal verification
- Functional verification
- Logic optimization and datapath synthesis
  - [DATE'15] A universal macro block mapping scheme for arithmetic circuits

Functional Unit Identification

Identifying functional units from flattened gate-level netlists has wide applications

- Word-level abstraction for formal verification
- Functional verification
- Logic optimization and datapath synthesis
- Hardware trojan detection
  - [DAC'19] Attacking split manufacturing from a deep learning perspective
  - [JETC'21] Hardware trust and assurance through reverse engineering: A tutorial and outlook from image analysis and machine learning perspectives



Challenges in Conventional Methods

- **Structural methods**
  - Focus on circuit topology
  - Efficient with customized algorithms ✓
  - Often mathematically incomplete, rely heavily on reference circuits ✗
  - Memory-consuming for large Boolean networks with billions of nodes ✗
  - Example: [HOST'13] Wenchao Li et al. Wordrev: Finding word-level structures in a sea of bit-level gates
- **Functional methods**
  - Functionally analyze the circuit for potential arithmetic components
  - Accurate and solver-ready ✓
  - Ultra-long runtime ✗
  - Example: [TETC'13] Pramod Subramanyan et al. Reverse engineering digital circuits using structural and functional analyses

Solution: Gamora

Graph Learning based Symbolic Reasoning for Large-Scale Boolean Networks

Why Gamora?

Graph Learning based solution:

- Gate-level netlists are naturally represented as graphs
- Alternate solutions to recognition and classification
- Make better use to modern computing systems → better scalability

Why Gamora?

Successful examples using graph learning:

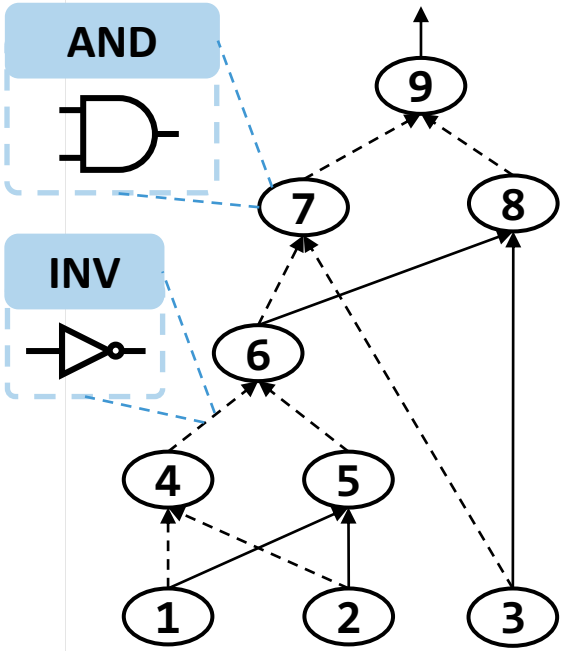
- **Classify sub-circuit functionality from gate-level netlists**
  - [TCAD'21] GNN-RE: Graph neural networks for reverse engineering of gate-level netlists
- **Predict boundaries of arithmetic blocks**
  - [ICCAD'21] Graph learning-based arithmetic block identification
- **Predict the functionality of approximate circuits**
  - [ICCAD'22] AppGNN: Approximation-aware functional reverse engineering using graph neural networks
- **Analyze impacts of circuit rewriting on functional operator detection**
  - [GLSVLSI'22] Graph neural network based netlist operator detection under circuit rewriting

How Gamora?

Symbolic reasoning requires structural and functional information from neighborhood nodes

- Reasoning can be formulated as node-level classification in graphs
- Message-passing mechanism in GNN computation:
  - Simultaneously handling **structural** and **functional** information from Boolean networks
  - Analogous to symbolic propagation and structural hashing

How Gamora?

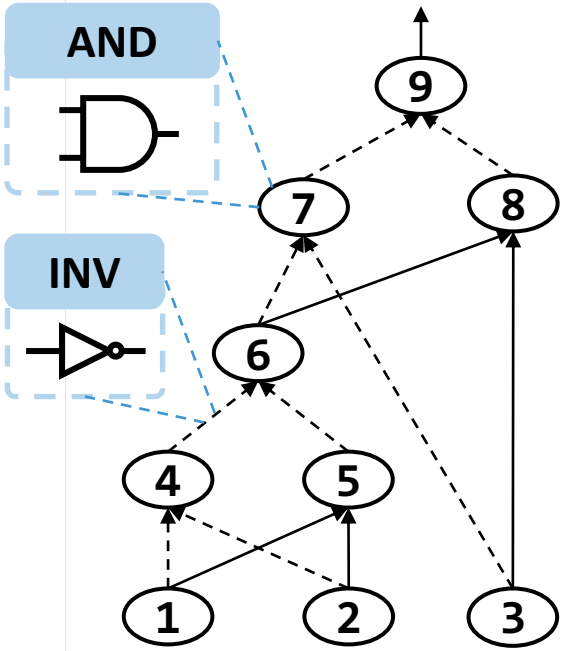


Flattened gate-level netlist in And-Inverter Graph (AIG)

Conventional methods

- Structural hashing + functional propagation
- Low scalability
- Low parallelism

How Gamora?



Flattened gate-level netlist in And-Inverter Graph (AIG)

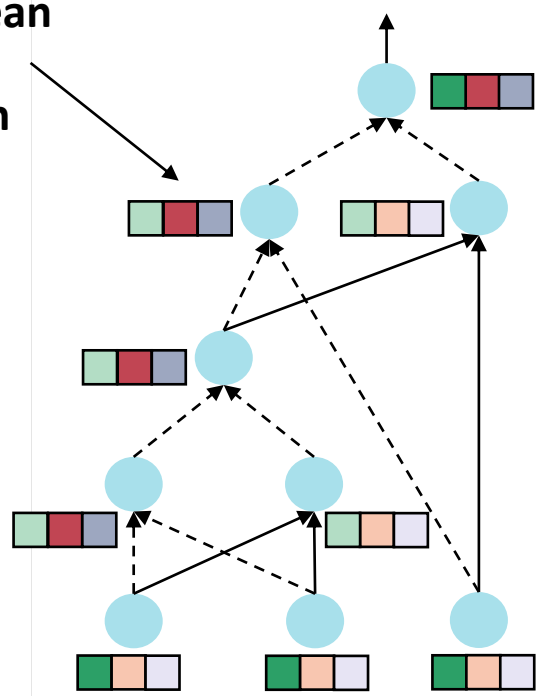
Conventional methods

- Structural hashing + functional propagation
- Low scalability
- Low parallelism

Our solution: GNN

How Gamora?

Node features to encode Boolean functional information

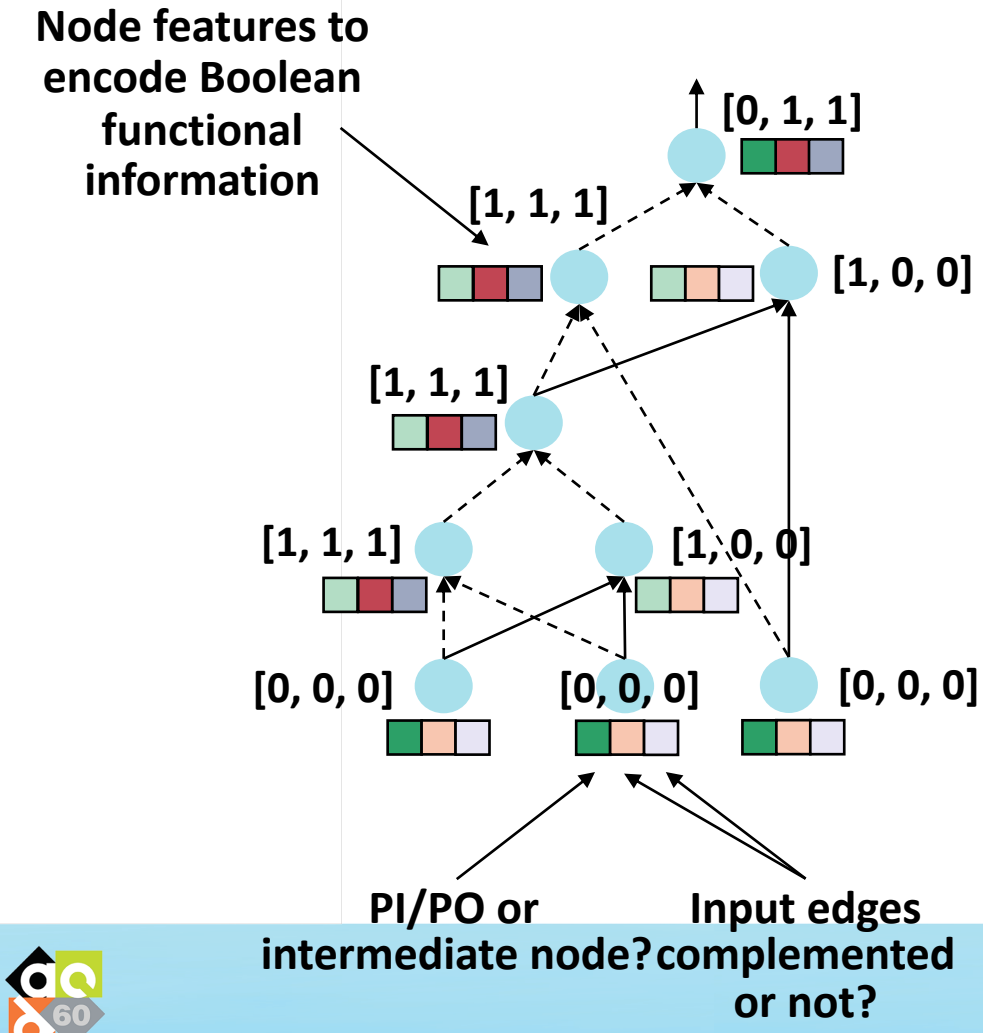


Conventional methods

- Structural hashing + functional propagation
- Low scalability
- Low parallelism

Our solution: GNN

How Gamora?



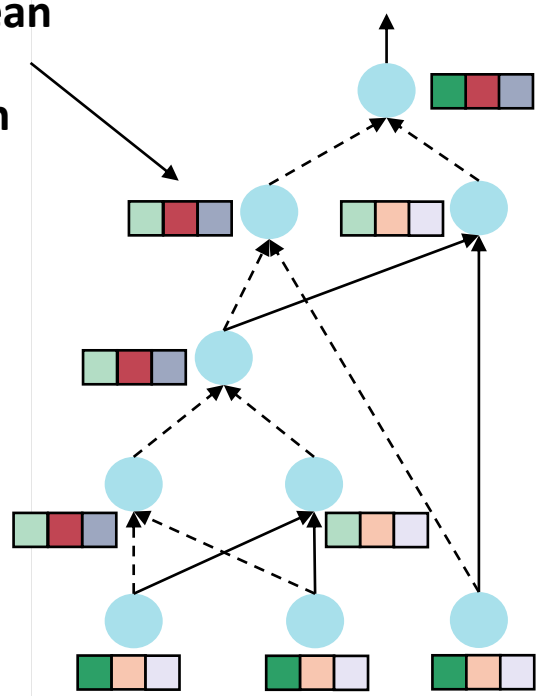
Conventional methods

- Structural hashing + functional propagation
- Low scalability
- Low parallelism

Our solution: GNN

How Gamora?

Node features to encode Boolean functional information

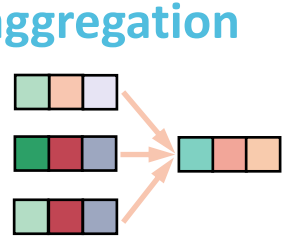


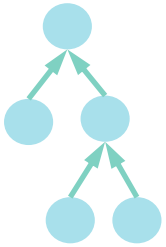
Conventional methods

- Structural hashing + functional propagation
- Low scalability
- Low parallelism

Our solution: GNN

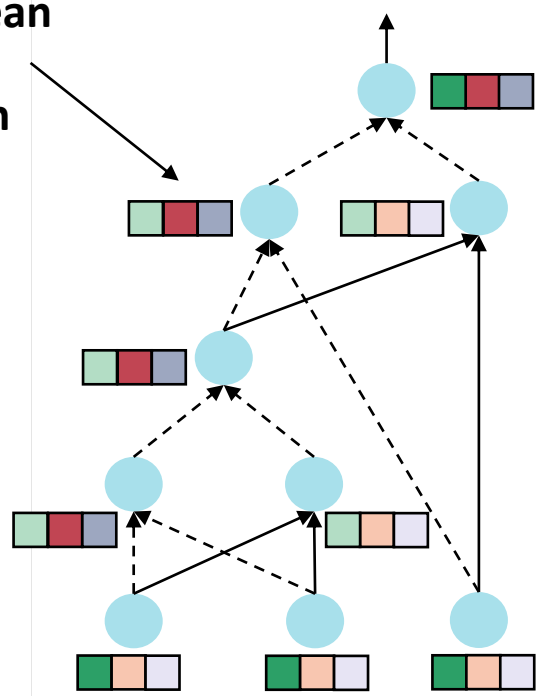
- Message-passing mechanism: functional and structural aggregation





How Gamora?

Node features to encode Boolean functional information



Conventional methods

- Structural hashing + functional propagation
- **Low scalability**
- **Low parallelism**

Our solution: GNN

- Message-passing mechanism: functional and structural aggregation
- **Strong scalability**
- **Better utilization of modern compute platforms**

How Gamora?

- **How to handle laRge-scale BooleAn networks?**
  - GNN has better support from modern computing systems → **GPU acceleration**
  - Node-level, model-level, and graph-level **parallelism** → **Billion-node scalability**

Multiplier Verification

- **Integer multipliers are ubiquitous components**
  - Advanced multipliers, such as Booth multipliers, are difficult to verify.
  - [FMCAD'21] Sound and automated verification of real-world RTL multipliers
- **Large multipliers are important in homomorphic encryption**
  - [ISCAS'14] Practical homomorphic encryption: A survey
  - [CSUR'18] A survey on homomorphic encryption schemes: Theory and implementation

Signal processing

Homomorphic encryption

Multiplier Verification

Our goal: identify the adder tree in flattened/bit-blasted multiplier netlists

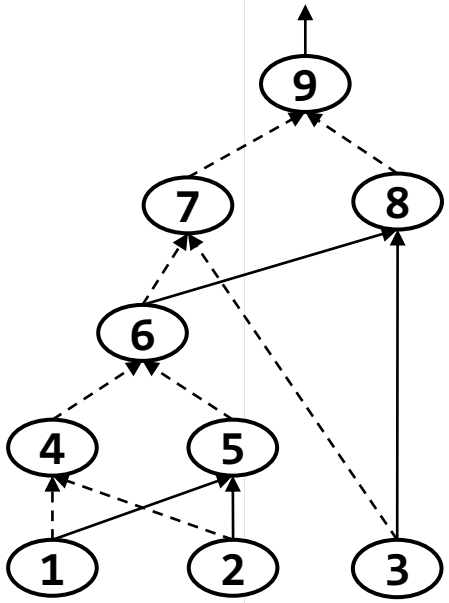
- Essential step in symbolic computer algebra for multiplier verification

Single Adder Extraction

- A full adder has a SUM and a CARRY

Single Adder Extraction

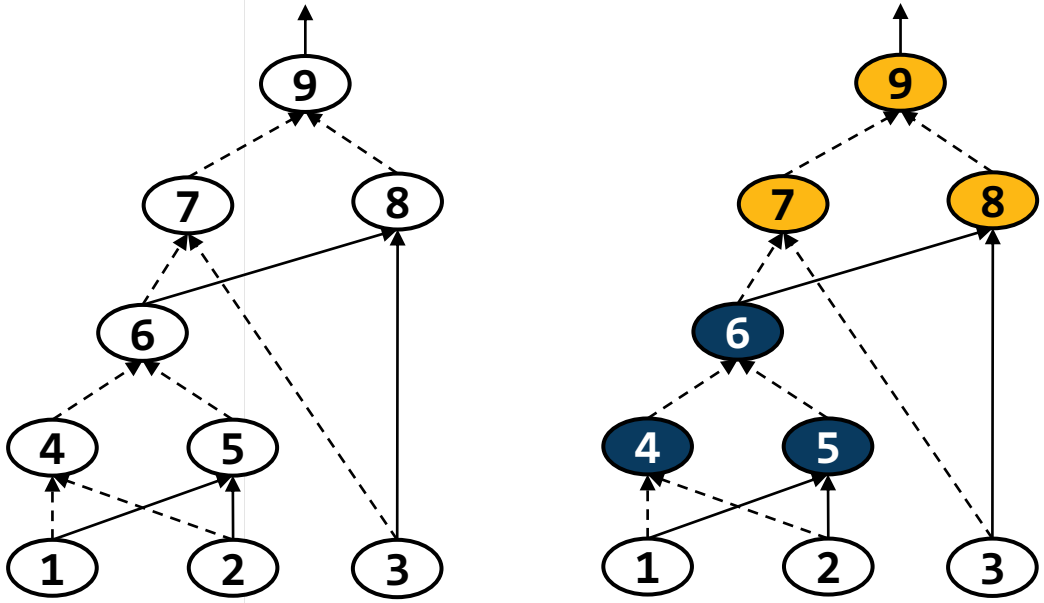
- A full adder has a SUM and a CARRY
- **SUM = XOR3(1,2,3)**



(a) AIG

Single Adder Extraction

- A full adder has a SUM and a CARRY
- **SUM = XOR3(1,2,3)**

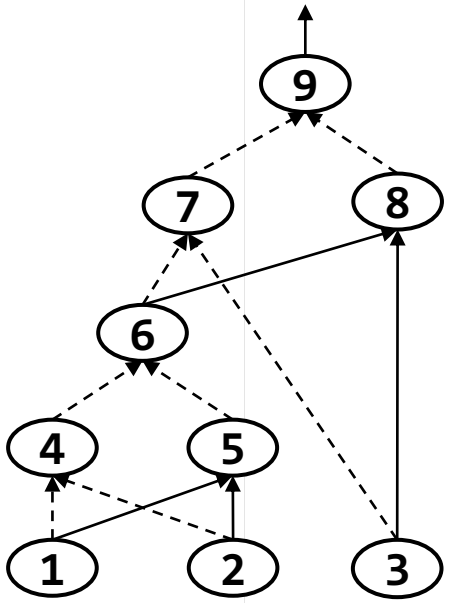


(a) AIG

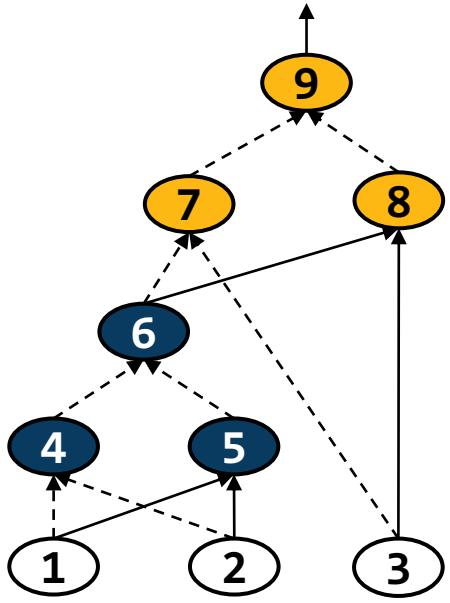
(b) OUT6=XOR(1,2)
OUT9=XOR(6,3)

Single Adder Extraction

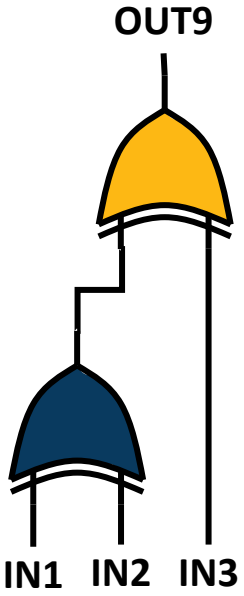
- A full adder has a SUM and a CARRY
- **SUM = XOR3(1,2,3)**



(a) AIG



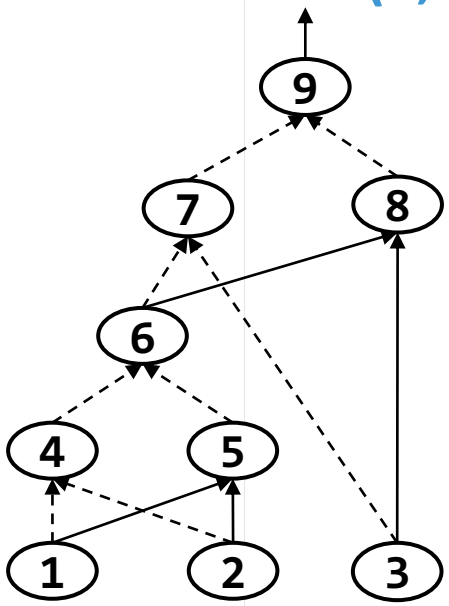
(b) OUT6=XOR(1,2)
OUT9=XOR(6,3)



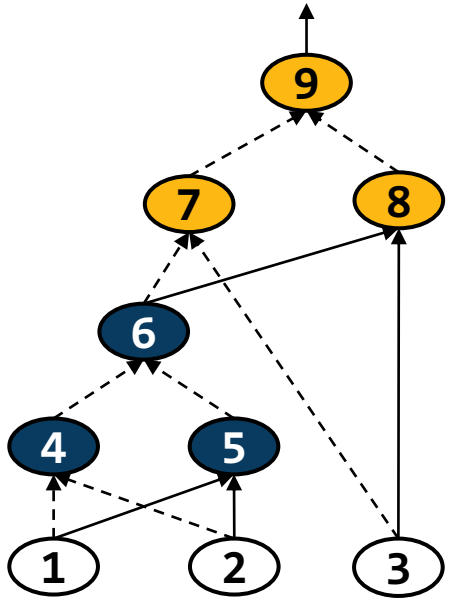
(c) OUT9 = XOR3(1,2,3)

Single Adder Extraction

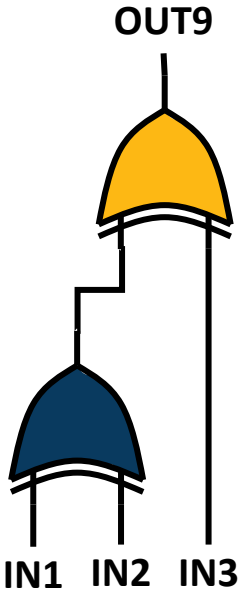
- A full adder has a SUM and a CARRY
- SUM = XOR3(1,2,3)
- **CARRY = MAJ3(1,2,3)**



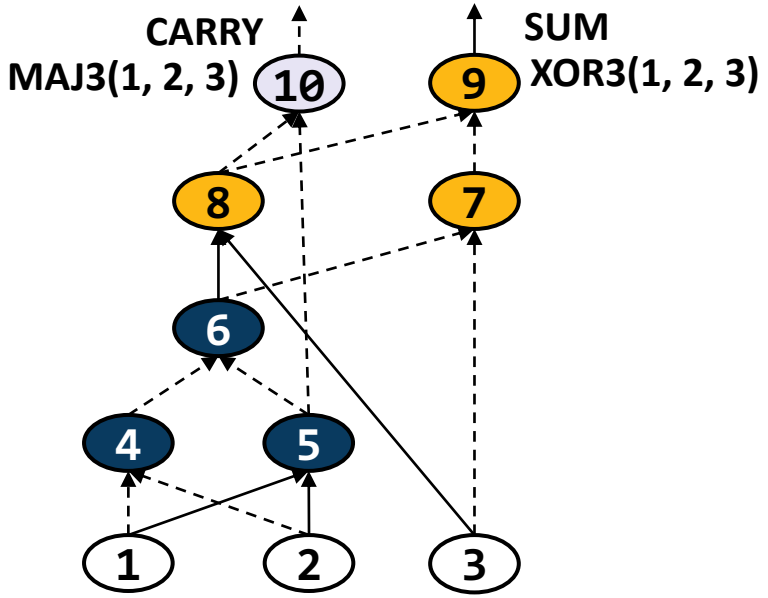
(a) AIG



(b) OUT6=XOR(1,2)
OUT9=XOR(6,3)



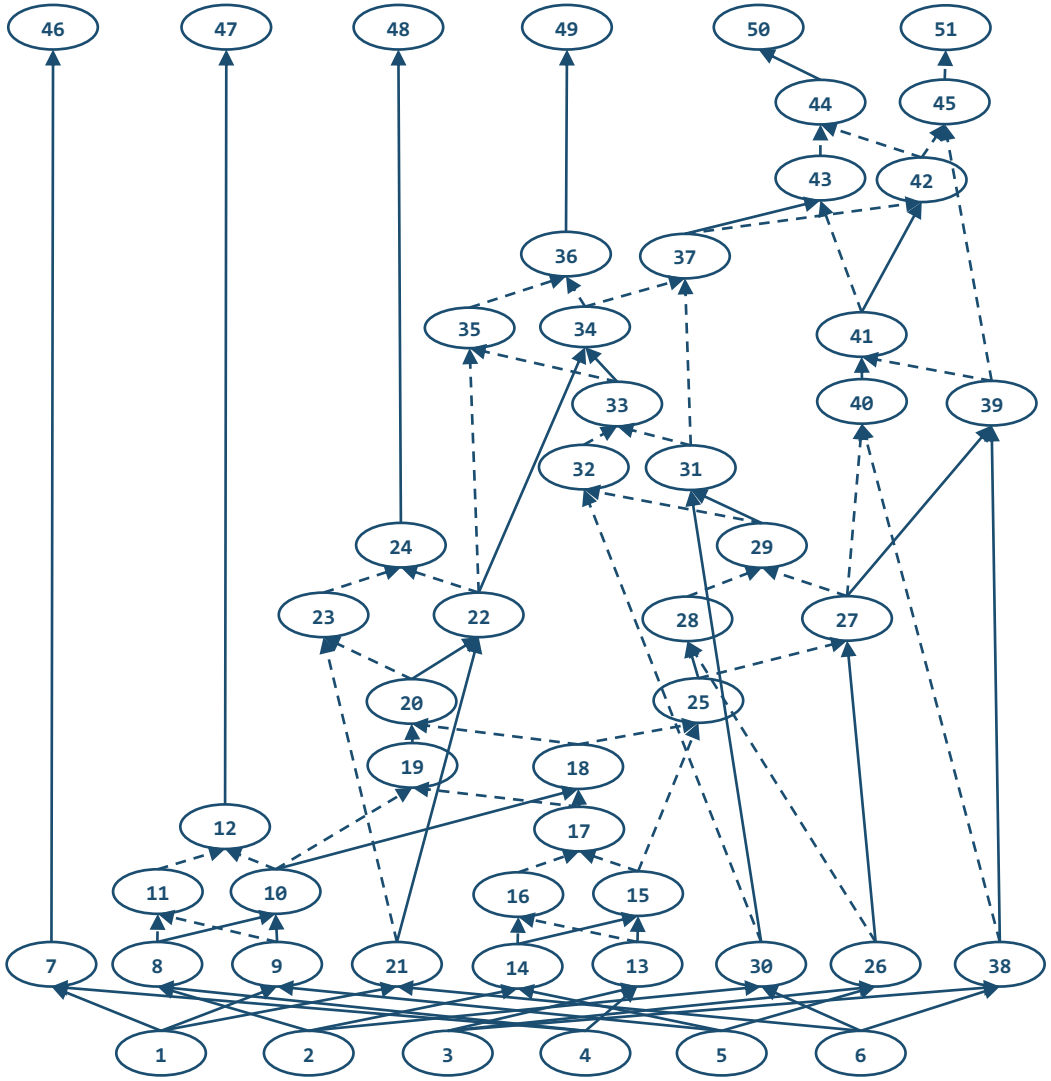
(c) OUT9 = XOR3(1,2,3)



(d) AIG of a full adder

Adder Tree Extraction

Example: AIG of 3-bit multiplier

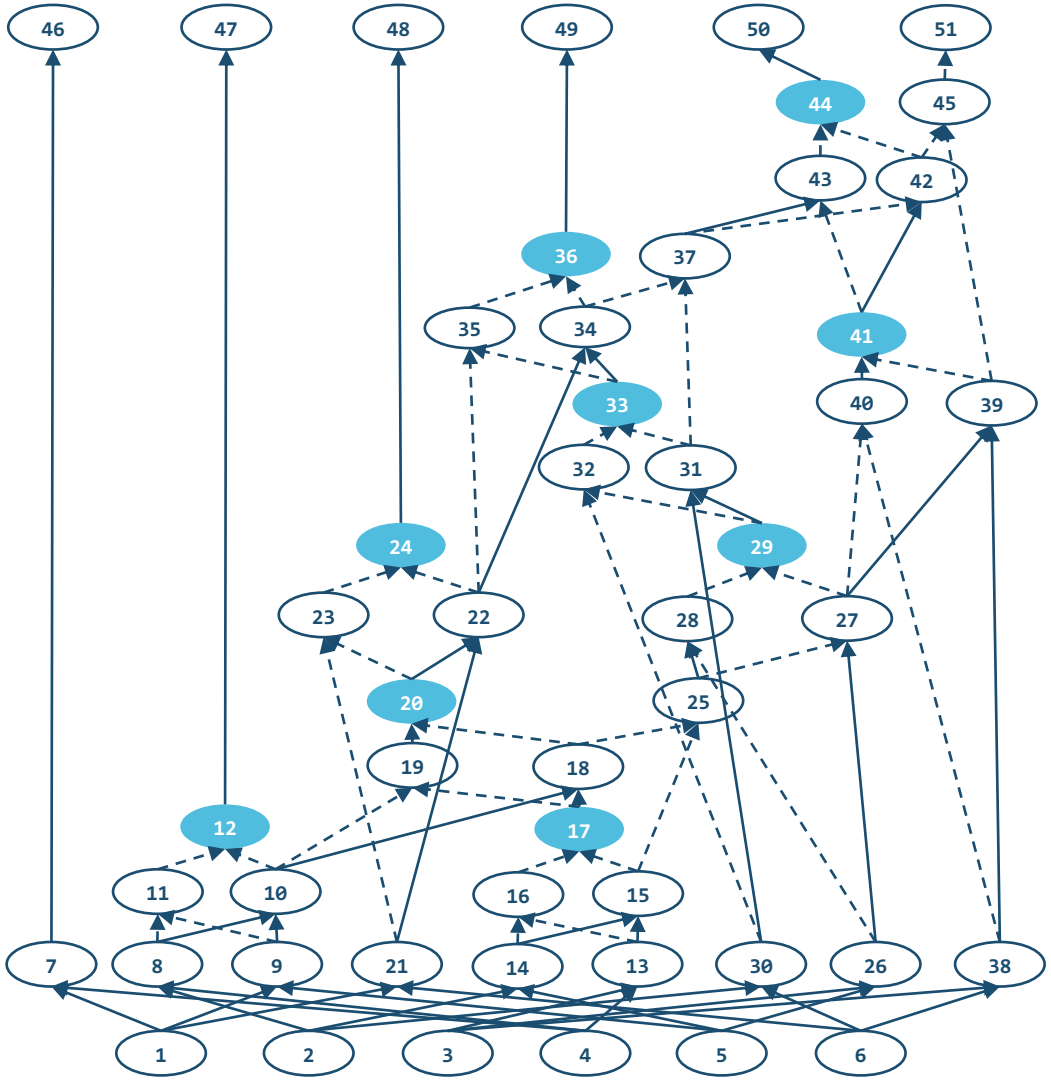


Adder Tree Extraction

Example: AIG of 3-bit multiplier

Step 1: find XOR nodes

XOR



Adder Tree Extraction

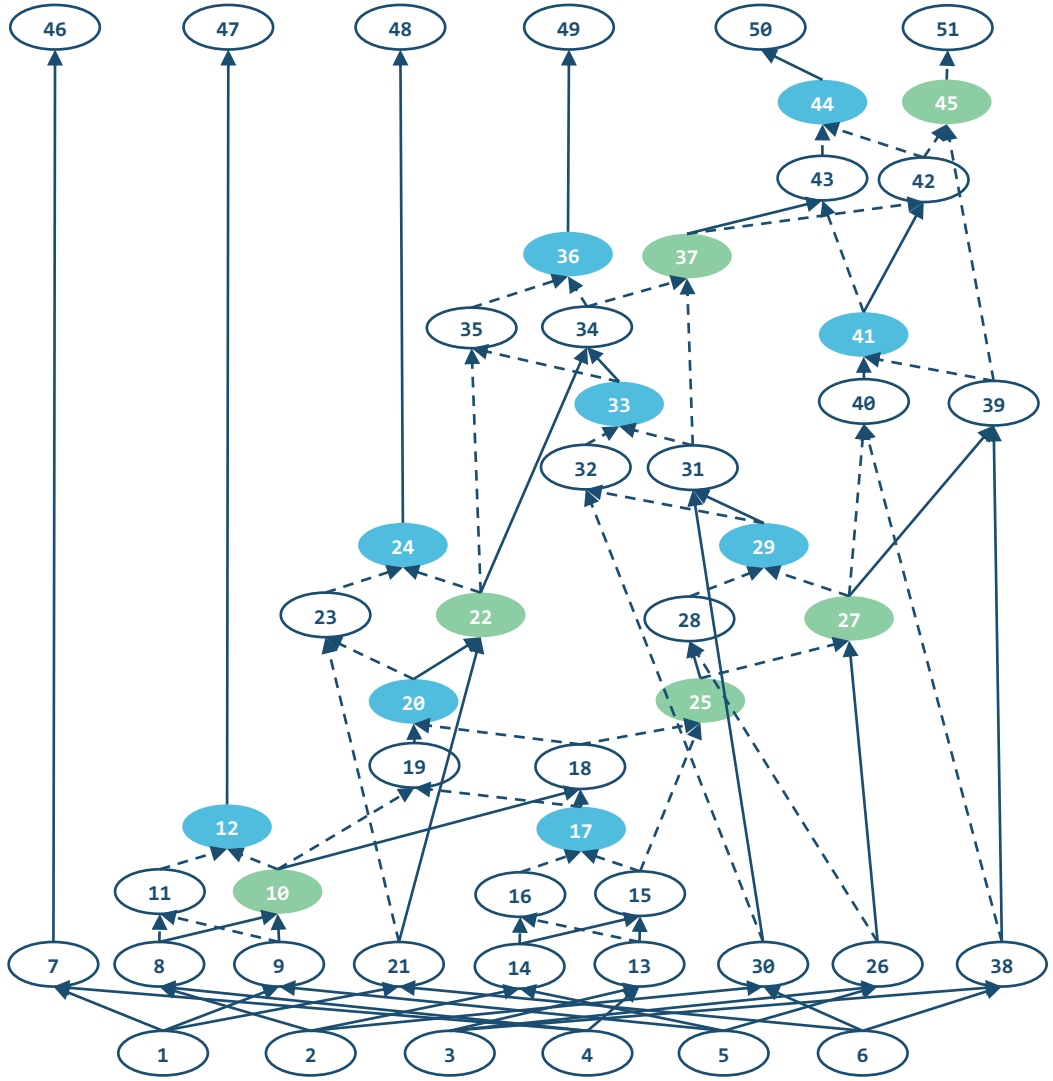
Example: AIG of 3-bit multiplier

Step 1: find XOR nodes

XOR

Step 2: find MAJ nodes

MAJ



Adder Tree Extraction

Example: AIG of 3-bit multiplier

Step 1: find XOR nodes

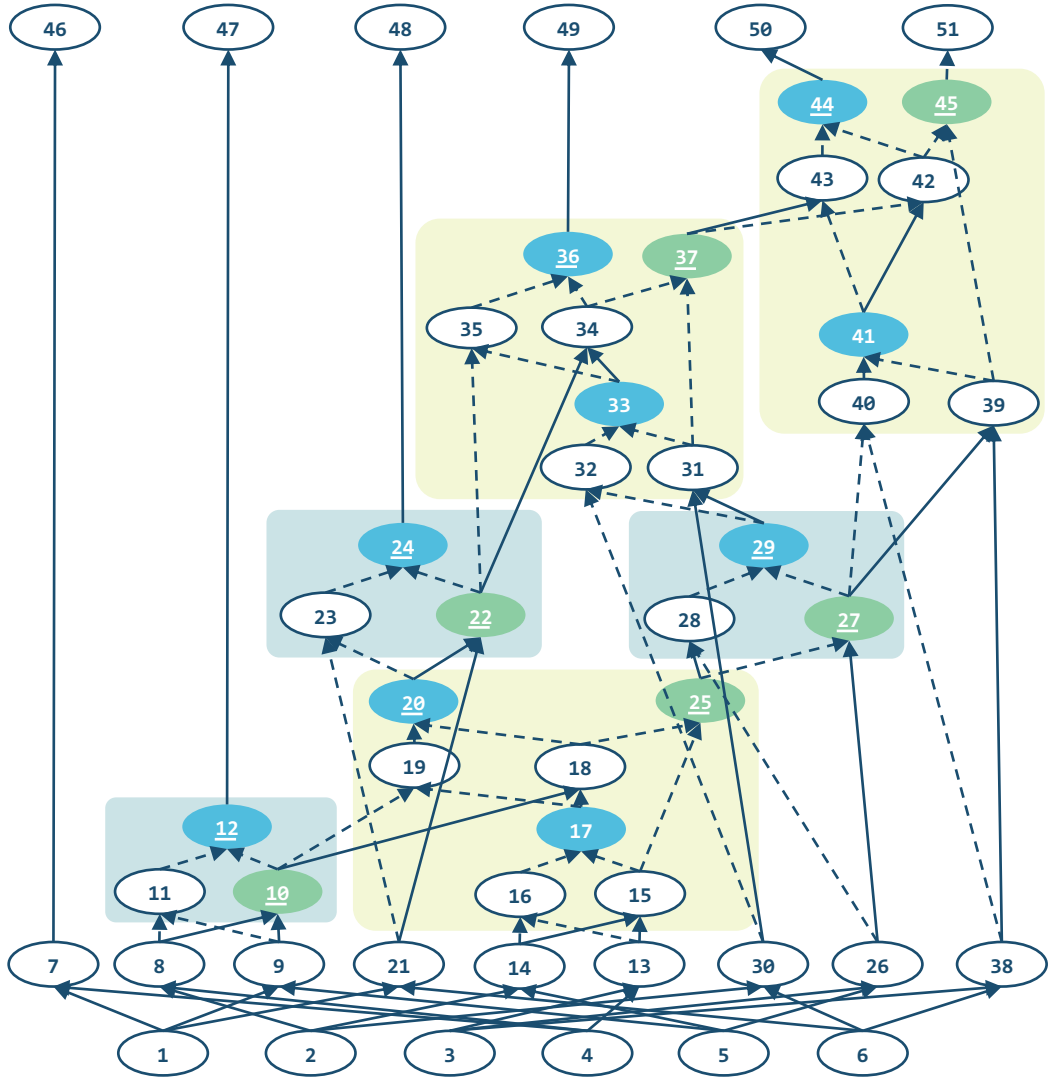
XOR

Step 2: find MAJ nodes

MAJ

Step 3: find boundary

Root ID ID



Adder Tree Extraction

Example: AIG of 3-bit multiplier

Step 1: find XOR nodes

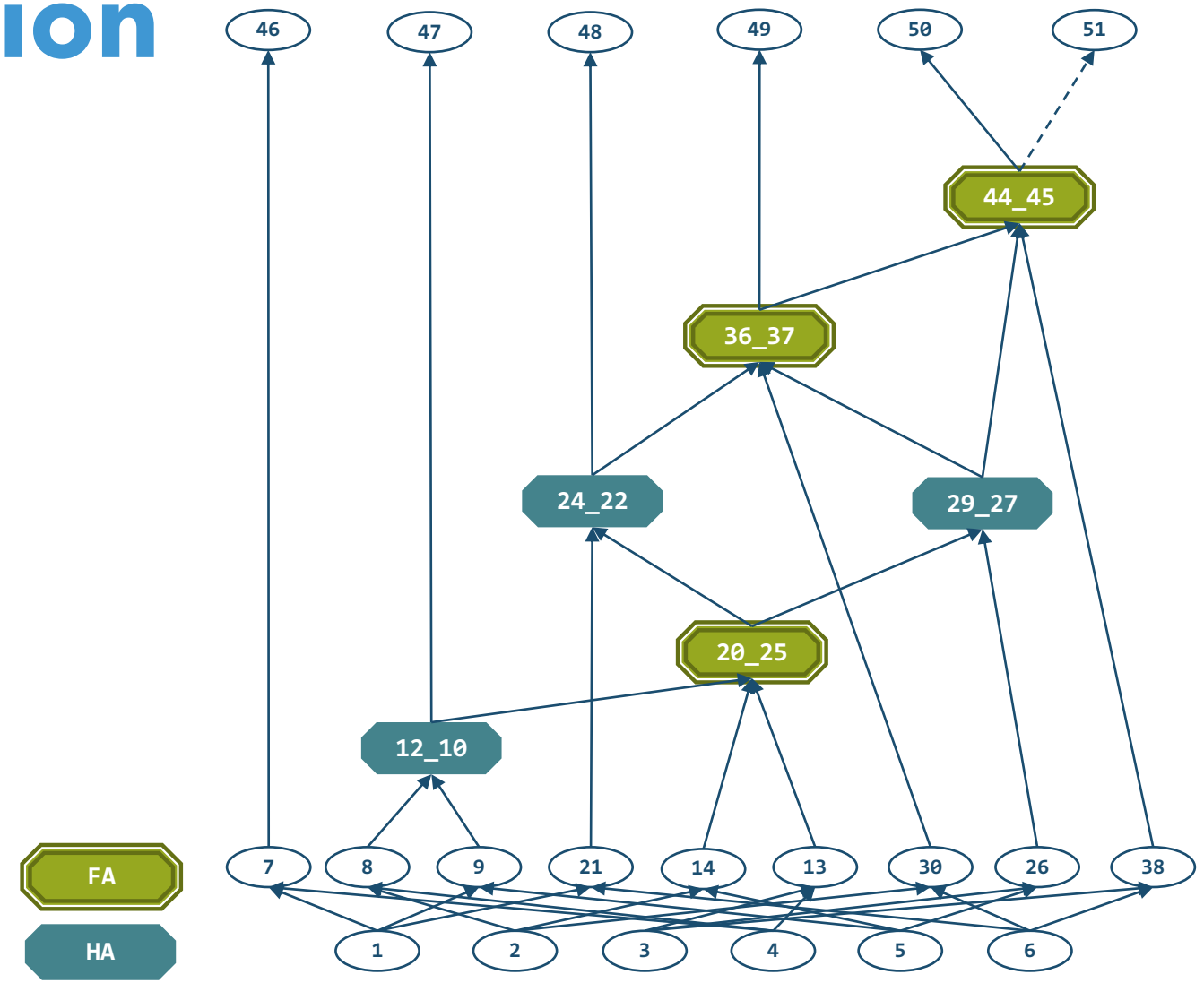
XOR

Step 2: find MAJ nodes

MAJ

Step 3: find boundary

Root ID ID



Adder Tree Extraction

Example: AIG of 3-bit multiplier

Step 1: find XOR nodes

XOR

Step 2: find MAJ nodes

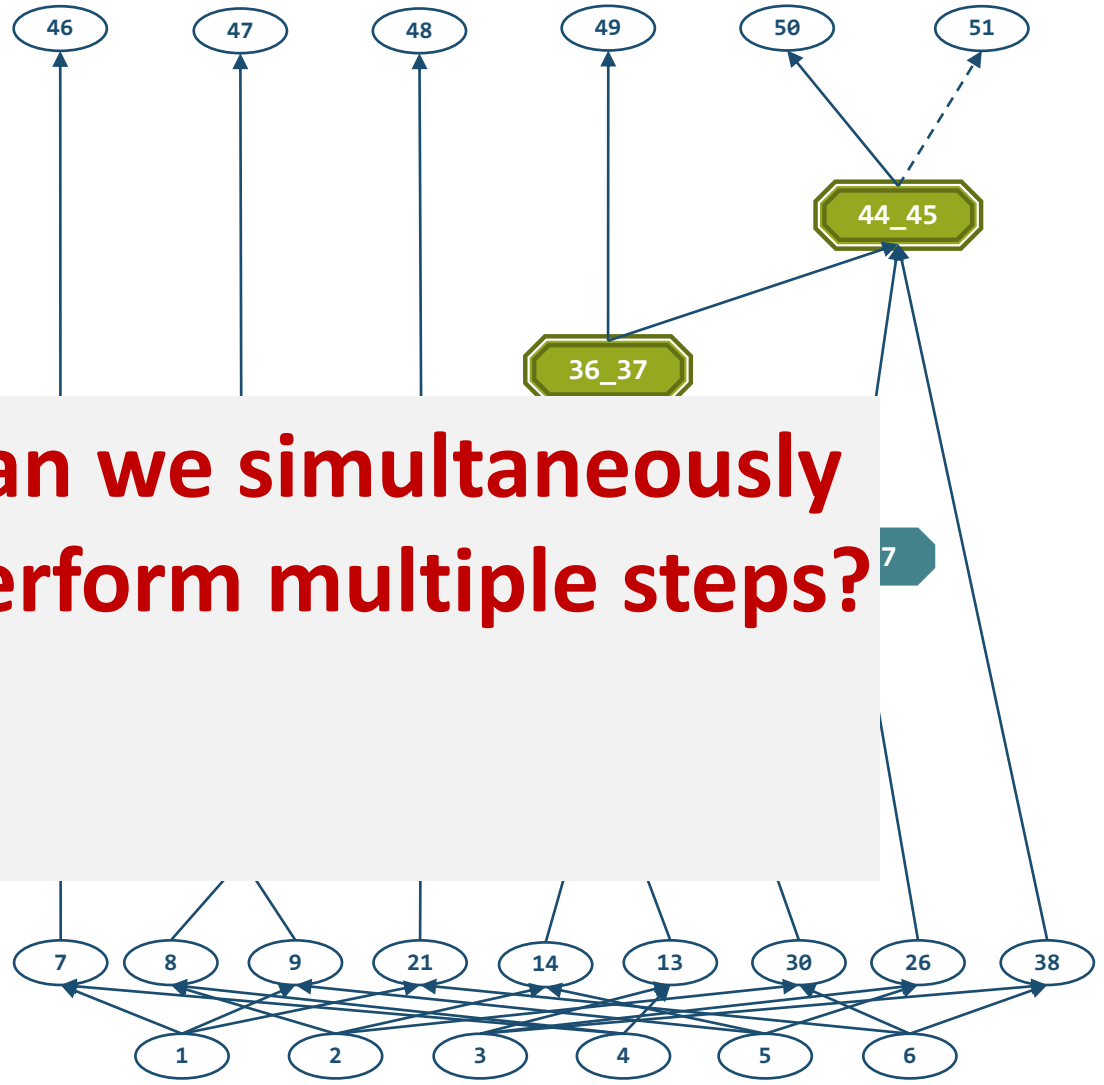
MAJ

Step 3: find boundary

Root ID ID

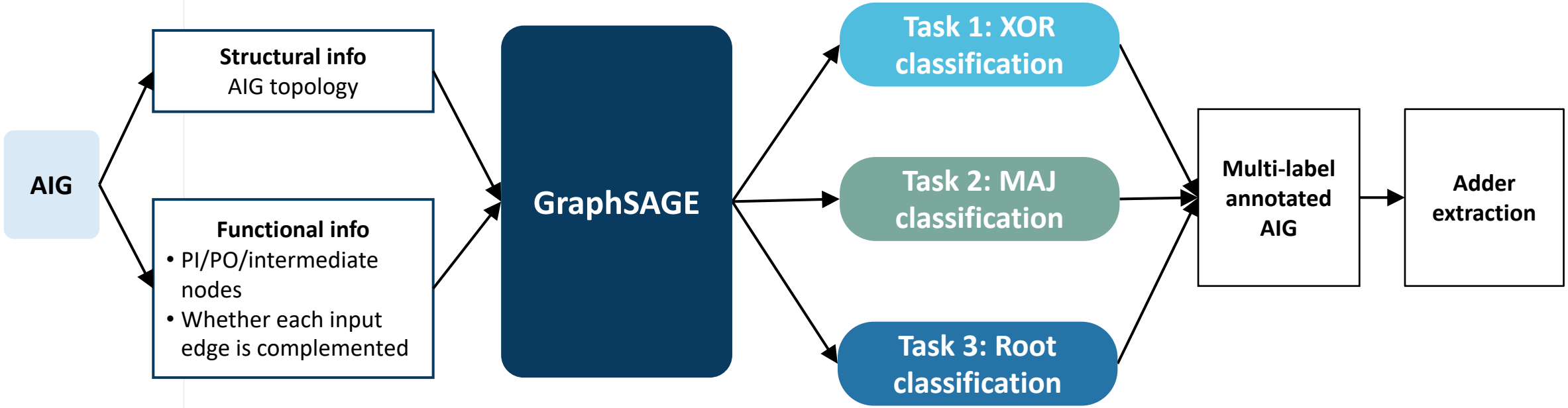
FA

HA

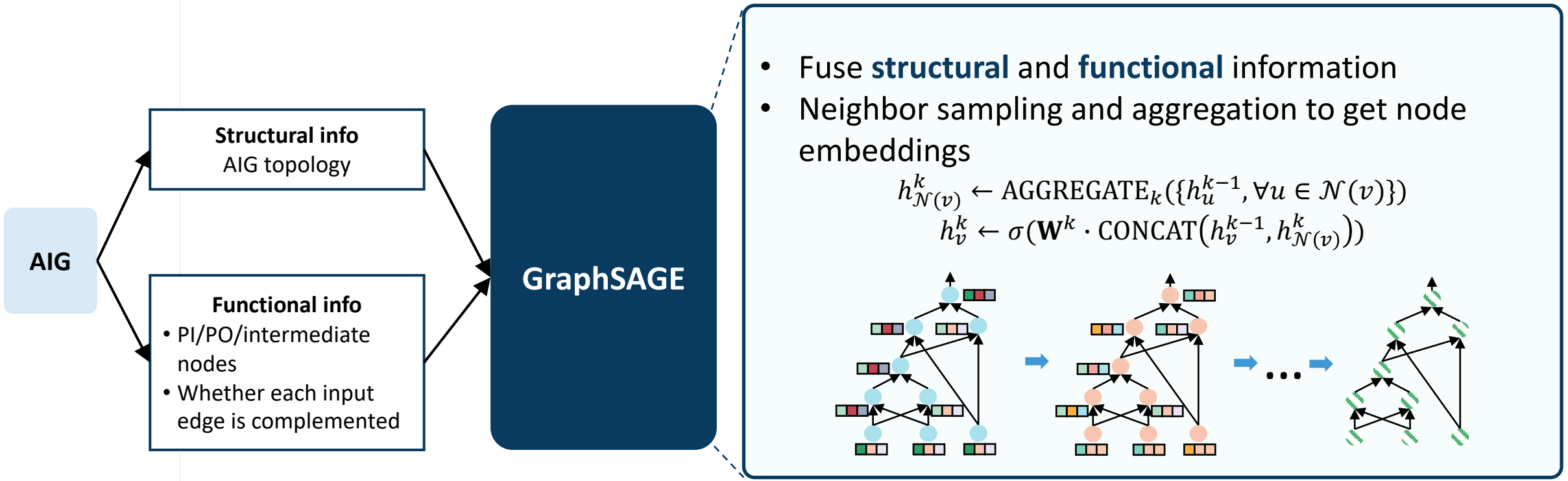


Can we simultaneously perform multiple steps?

Solution: Multi-task GNN



Solution: Multi-task GNN

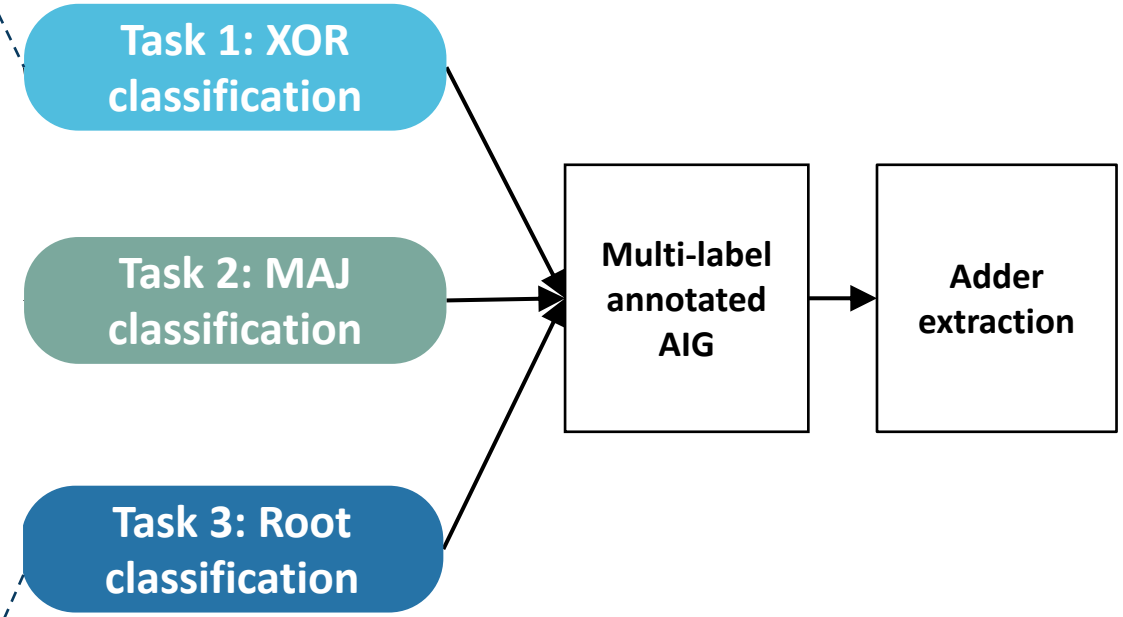


Solution: Multi-task GNN

Multi-task learning

- **Knowledge sharing** across tasks to guarantee reasoning precision
- Improve **sample efficiency** during training
- Loss function:

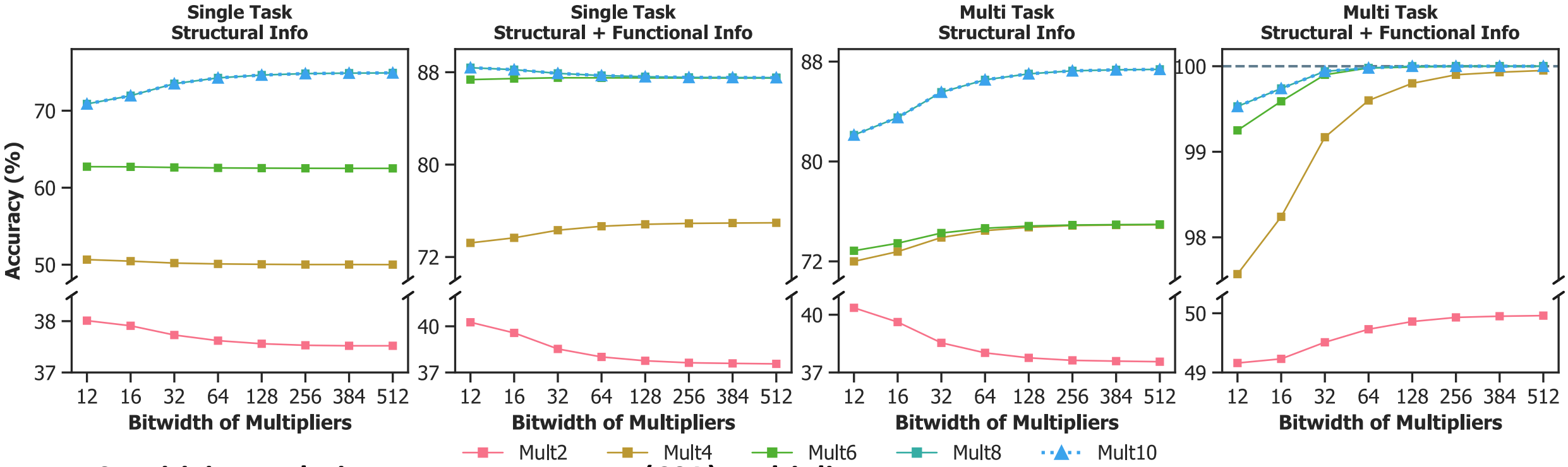
$$\mathcal{L} = \underbrace{\alpha \cdot \ell(\widehat{y_1}, y_1)}_{\text{Task 1}} + \underbrace{\beta \cdot \ell(\widehat{y_2}, y_2)}_{\text{Task 2}} + \underbrace{\gamma \cdot \ell(\widehat{y_3}, y_3)}_{\text{Task 3}}$$



Experiment Setup

- **AIG-based multiplier netlists**
  - Carry-save-array (CSA) multipliers
  - Booth-encoded multipliers
- **Technology mapping**
  - The reduced standard-cell library mcnc.genlib (with gate input size <=3) from SIS distribution
  - ASAP 7nm technologies
- **Baseline & ground truth**
  - Logic synthesis tool ABC, using the adder tree extraction command
  - [TCAD'17] Fast algebraic rewriting based on and-inverter graphs
  - [TCAD'19] Understanding algebraic rewriting for arithmetic circuit verification: a bit-flow model
- **Gamora is trained with small bitwidth multipliers (typically less than 32-bit) and evaluated on large bitwidth multipliers (up to 2048-bit)**

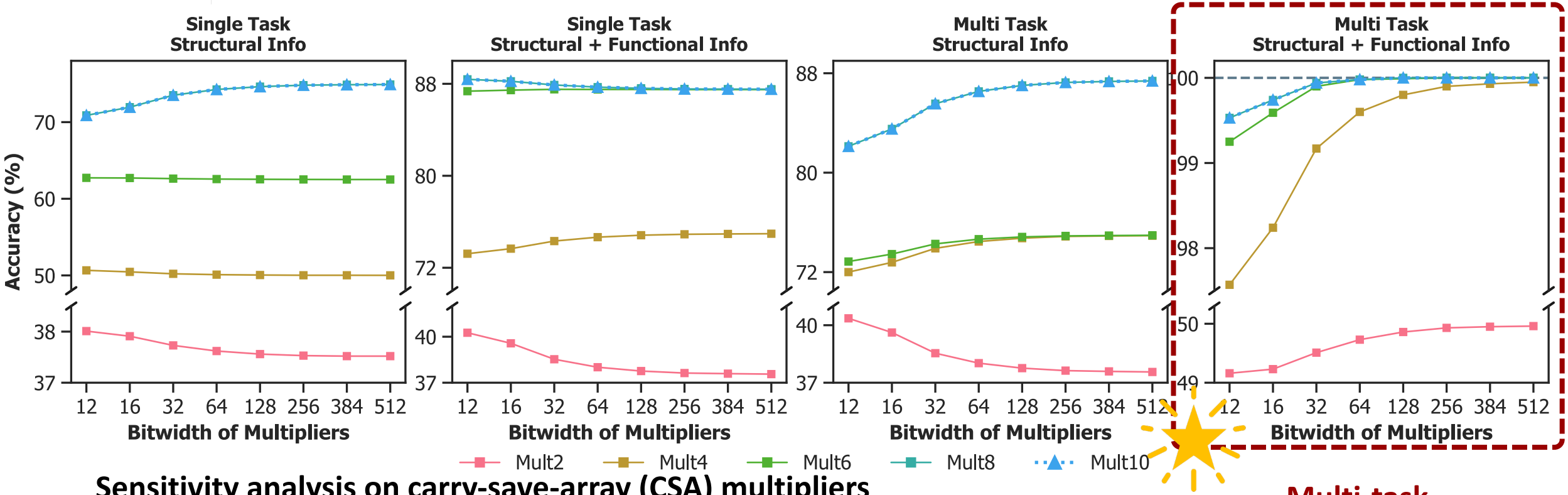
Evaluation: CSA Multiplier



Sensitivity analysis on carry-save-array (CSA) multipliers

- Single- and multi-task
- With and without functional info
- Different training size (2-bit to 10-bit)

Evaluation: CSA Multiplier

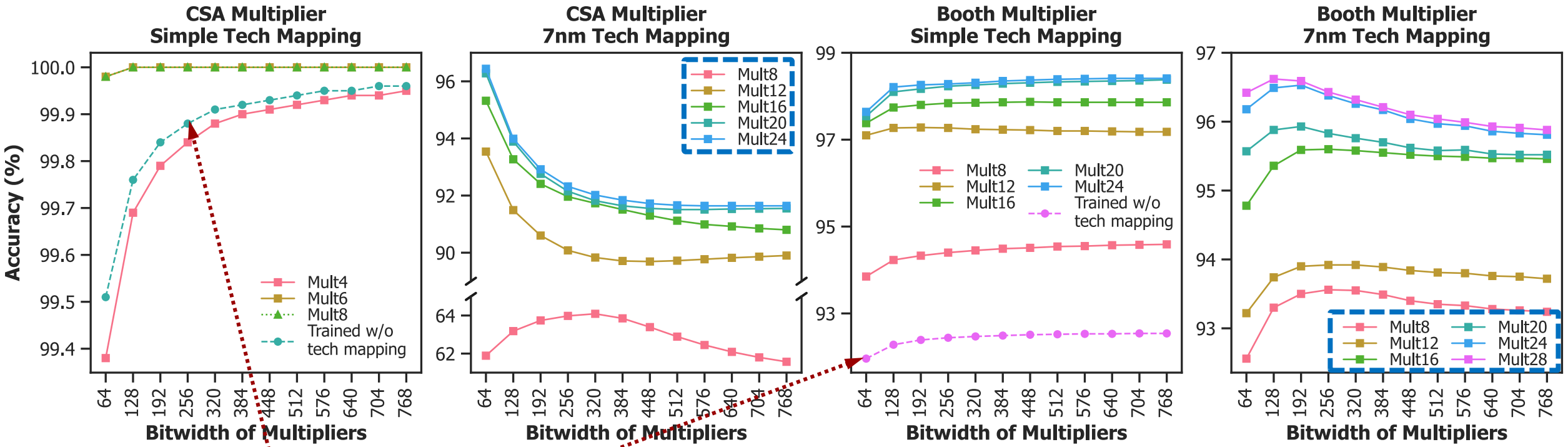


Sensitivity analysis on carry-save-array (CSA) multipliers

- Single- and multi-task
- With and without functional info
- Different training size (2-bit to 10-bit)

Multi-task, structural + functional info, 8-bit mult

Evaluation: Booth and Tech Mapping

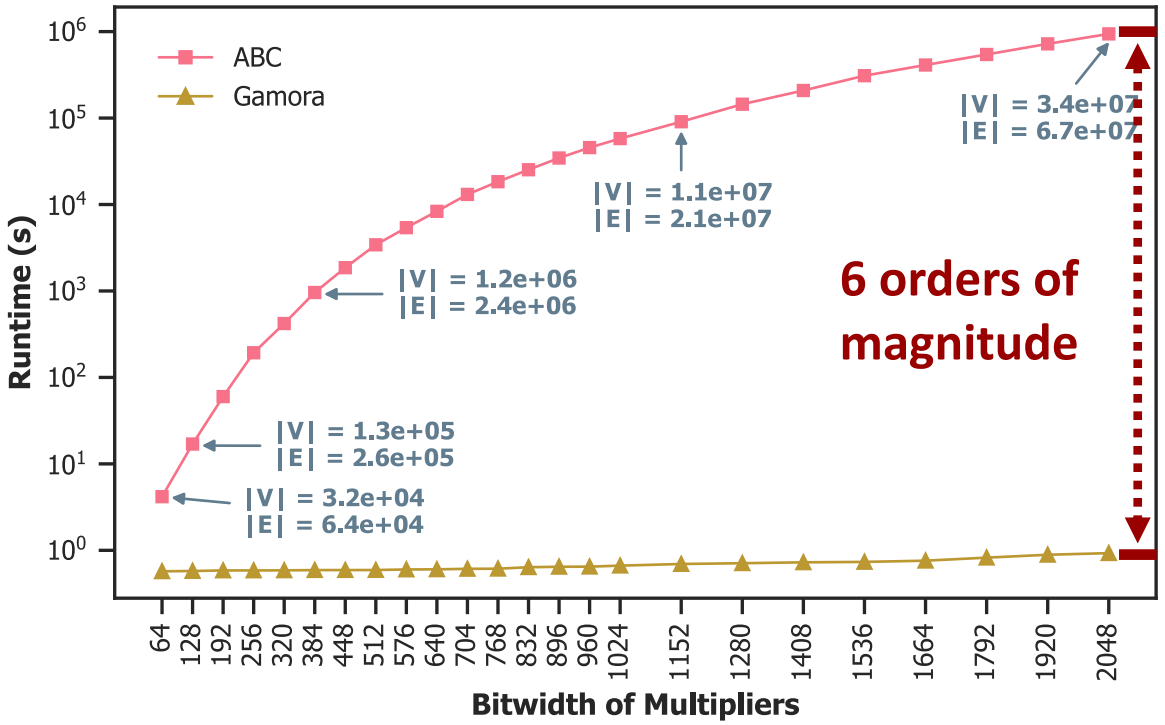


CSA and Booth multipliers, with simple and complex technology mapping

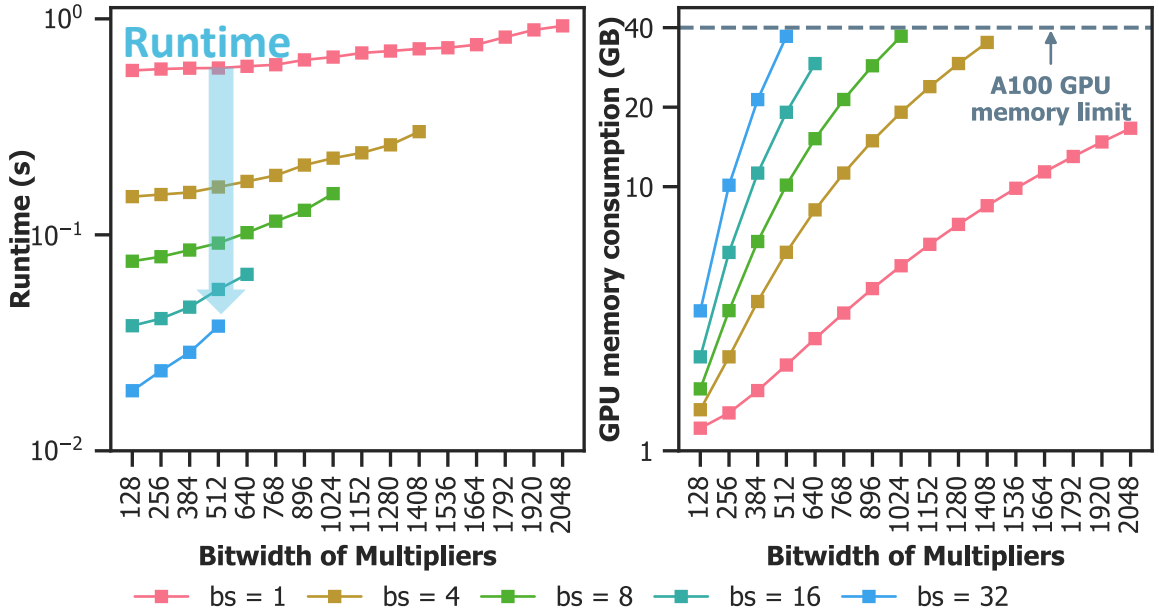
- **Generalization** from small to large designs
- **Generalization** from before to after simple tech mapping

Evaluation: Runtime and Scalability

Runtime comparison between Gamora and ABC.



Runtime and GPU memory consumption with batched reasoning.



ABC: a logic synthesis framework well adopted in academia

R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool". International Conference on Computer Aided Verification. Springer, 2010.

Further speedup with batched reasoning on a single GPU.

Summary

- **Gamora: a novel symbolic reasoning framework, which exploits GNNs to imitate structural hashing and functional aggregation in conventional reasoning approaches.**
  - **High reasoning performance** that reaches almost 100% and over 97% accuracy for CSA and Booth-encoded multipliers, which is still over 92% in finding functional modules after complex technology mapping;
  - **Strong scalability** to Boolean networks with over 33 million nodes, with up to six orders of magnitude speedups compared to the state-of-the-art implementation in the ABC framework;
  - **Great generalization capability** from simple to complex designs, such as from small to large bitwidth multipliers, and from before to after technology mapping.
- Gamora reveals the great potential of applying GNNs and GPU acceleration to speed up symbolic reasoning, which is available at https://github.com/Yu-Utah/Gamora.

Thanks!