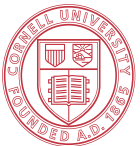




Hypothesizing (Fantasizing) Autonomous Hardware Design

Zhiru Zhang
Cornell University

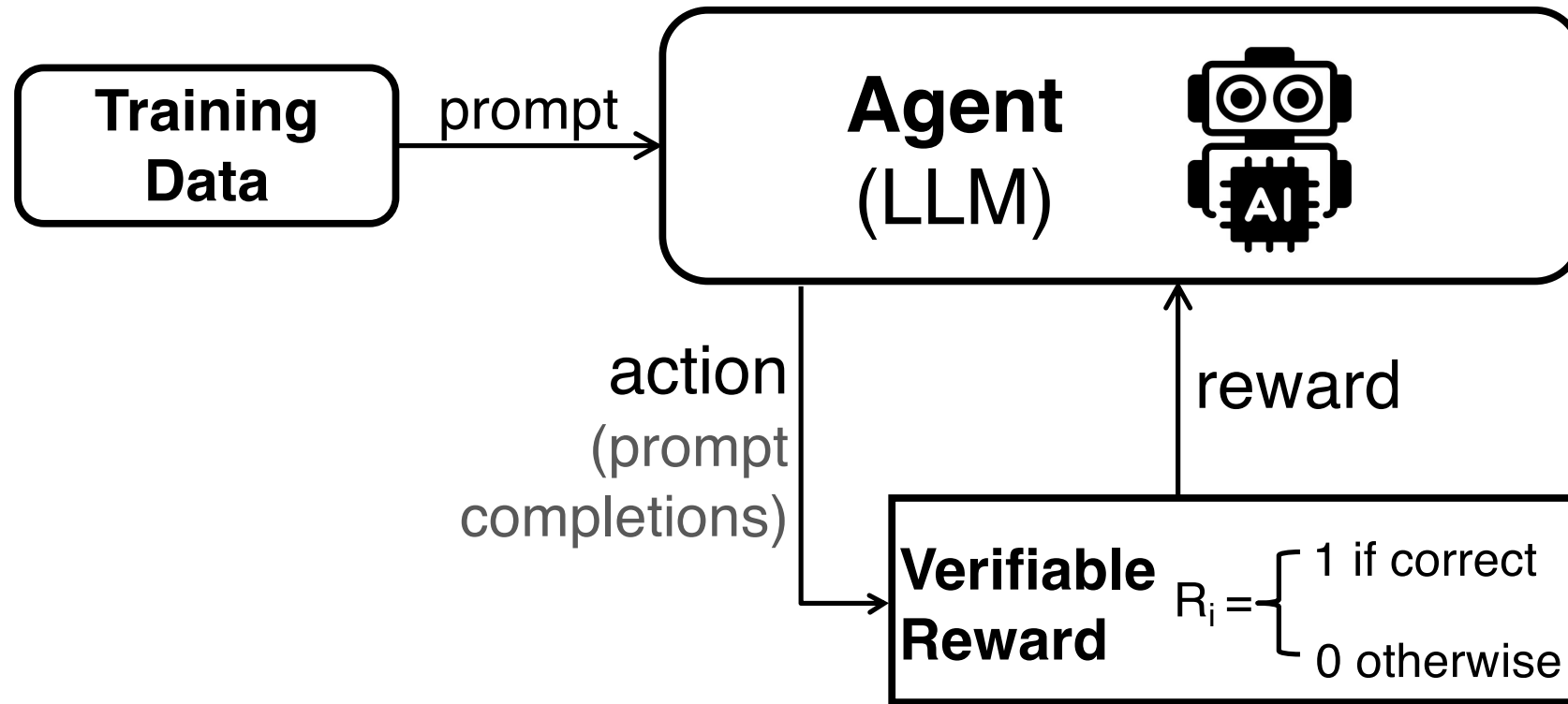
AI4FACD Workshop @ ISCA
6/21/2025



Cornell University



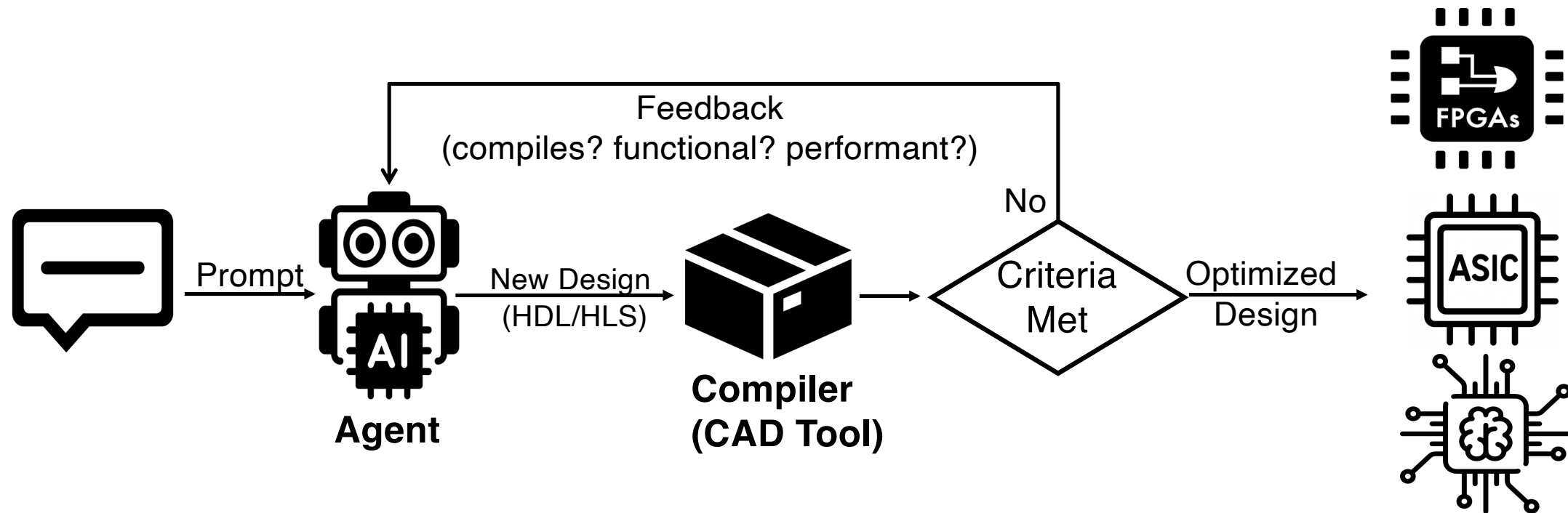
The Powerful Reasoning LLM



Performs well on many math and coding tasks, where

- **Problems are verifiable**
- **LLM+RL navigates large search space** by balancing exploration of new solutions and exploitation of proven approaches from pretraining

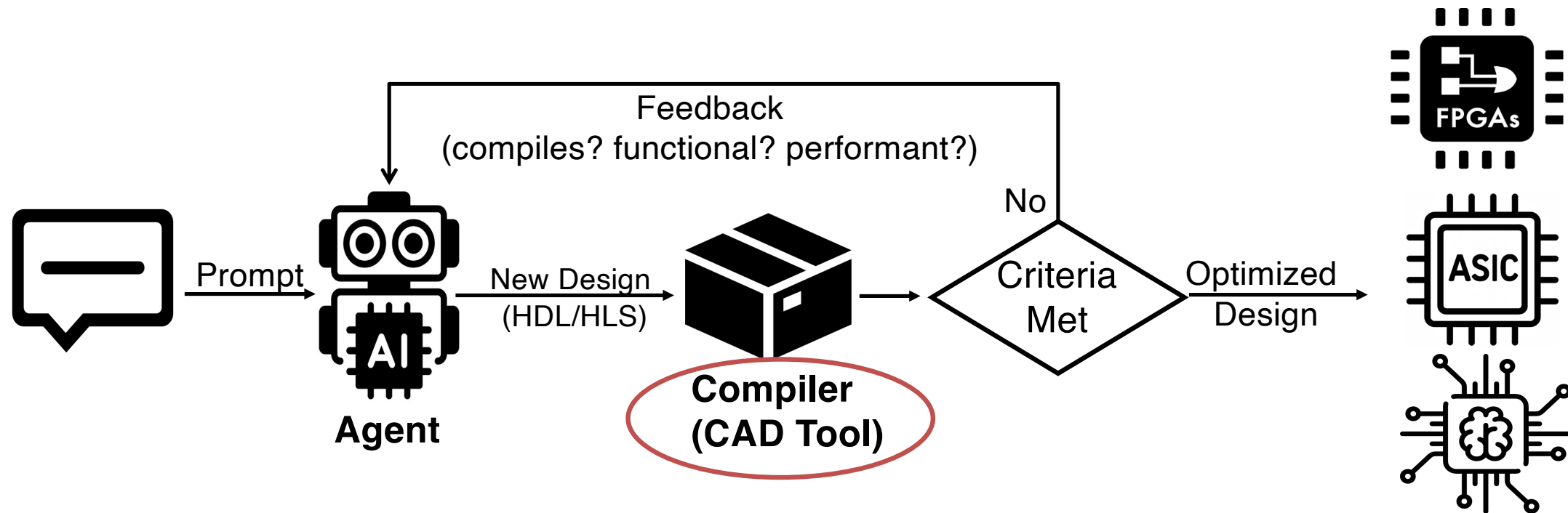
Replicating the Same Approach for Hardware Design?



Any showstoppers to this approach?

Learning the Bitter Lesson by Rich Sutton [1]

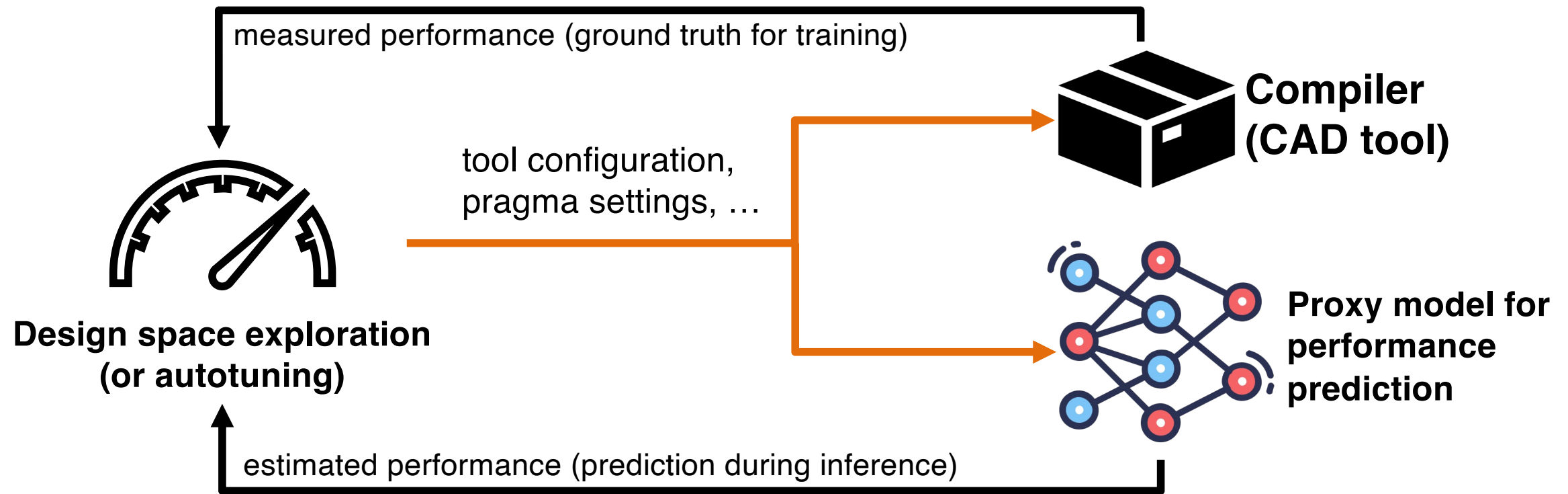
Most significant progress in AI has come from scaling – More compute, data, and general meta-methods beat specialized, human-designed strategies in the long run



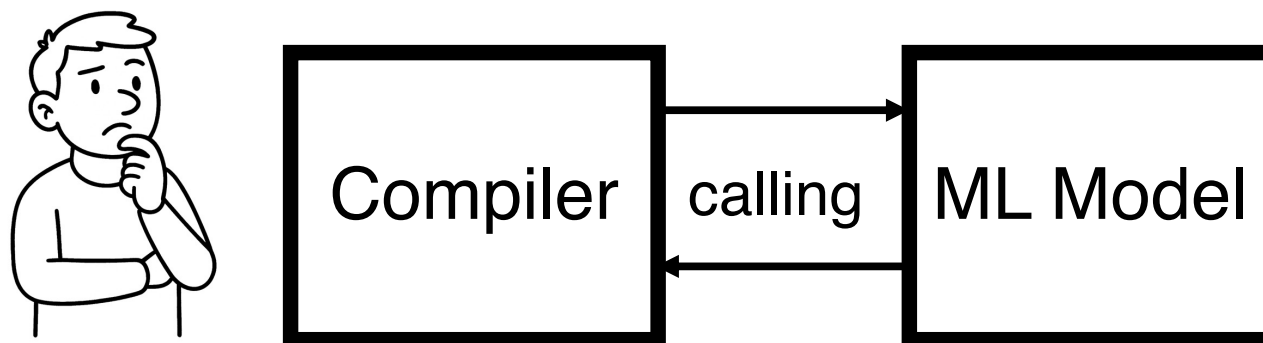
Bottleneck: Current HW compilation stack doesn't scale and relies heavily on handcrafted heuristics 😞

[1] <http://incompleteideas.net/IncIdeas/BitterLesson.html> (March 2019)

Active Research: Neural Approximations of the Compiler



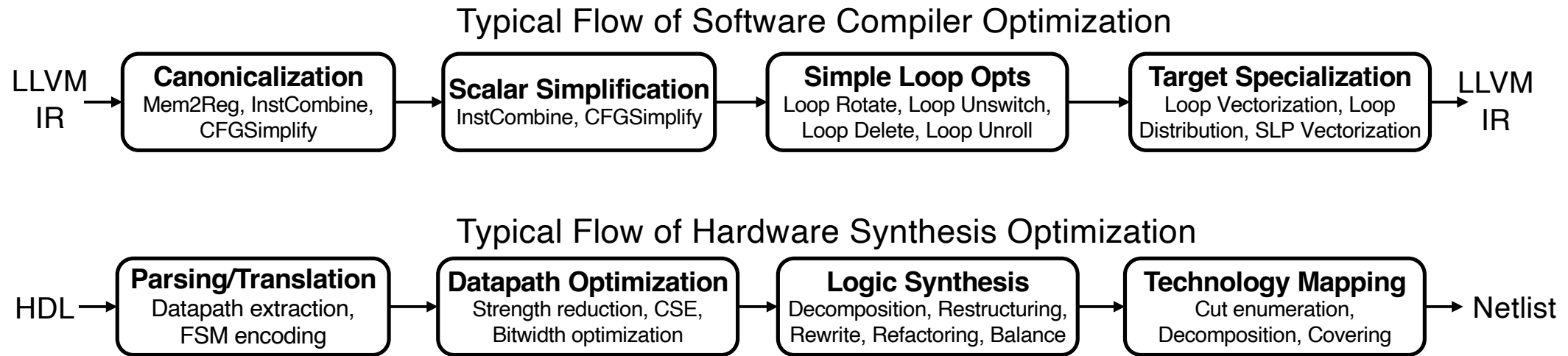
- Data hungry – extensive (in distribution) training required
- Compiler remains a black box and a bottleneck for training runs



Research Question 1

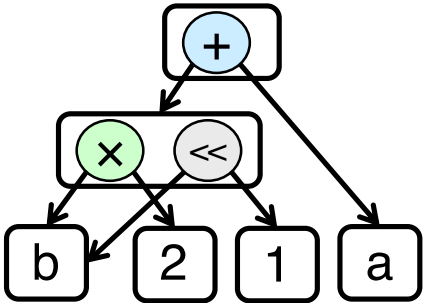
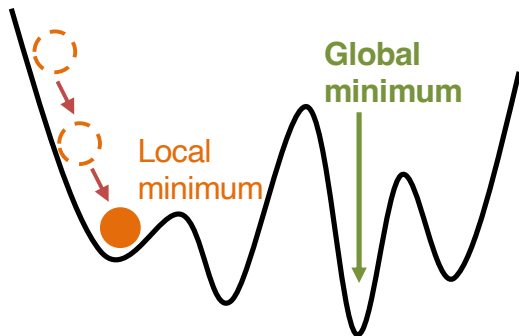
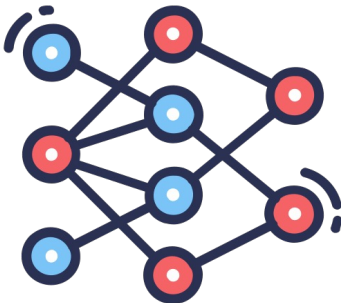
Instead of just compilers calling ML and vice versa,
can we push toward a deeper, more unified integration?

Deep-rooted Problems in Compiler/Synthesis Tools



- ▶ Most compiler/synthesis optimizations rely on ad hoc local heuristics, are hard to parallelize, and typically **run on CPUs using few threads**
- ▶ The granularity and order of these optimizations are manually preset, known as the **phase ordering** problem
 - The “optimal” sequence may differ across input problems and hardware targets

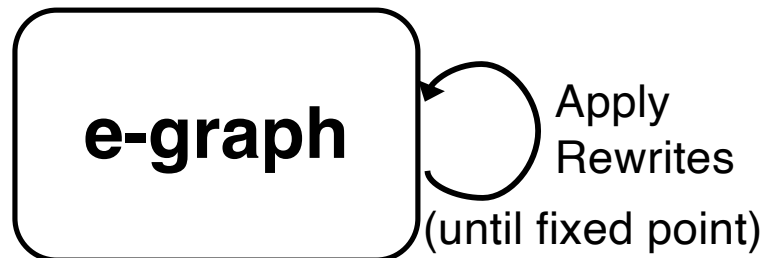
A Radical Ground-up Approach?

Compiler / Synthesis Optimization	Transformation Space	Search Method	Objective Function
Current Approach ↓ Differentiable Approach?	Phase ordered transforms (mostly local) ↓ Formal, compact encoding of equivalent transforms  Meta IR	Heuristics running on CPUs (mostly 1 thread) ↓ Parallelized global optimization  Gradient Descent	Hand-crafted cost models (mostly linear costs) ↓ Realistic, non-linear cost models  Learnable Cost

E-Graph as a Meta IR

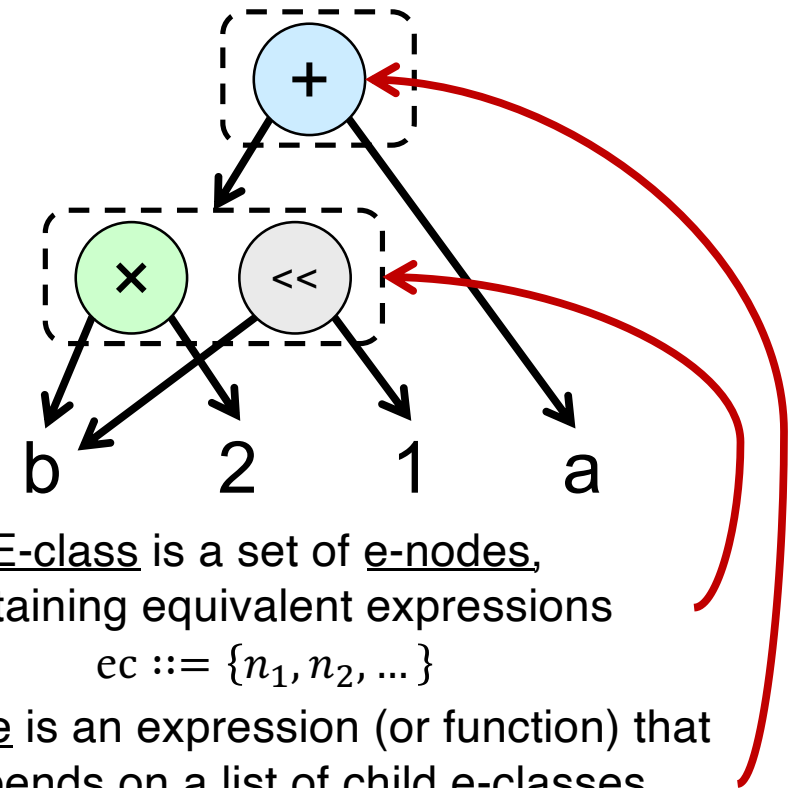
e-graph is a data structure that efficiently represents equivalent programs using e-nodes (expressions) and e-classes (sets of equivalent e-nodes)

Equality saturation expands e-graph until no more rewrites apply, enabling optimal expression extraction with given costs



Given

- ▶ An input program: $a + b \times 2$
- ▶ A rewrite rule: $t \times 2 \rightarrow t \ll 1$



E-class is a set of e-nodes, containing equivalent expressions

$$ec ::= \{n_1, n_2, \dots\}$$

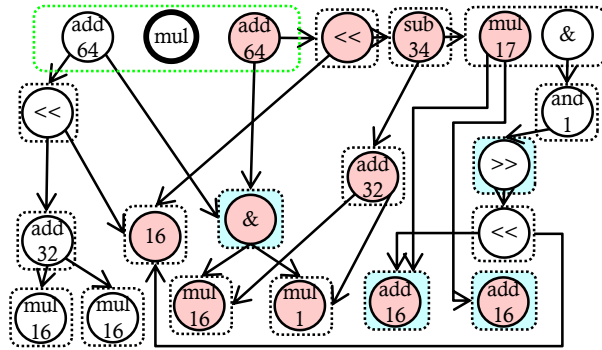
E-node is an expression (or function) that depends on a list of child e-classes

$$n ::= f(ec_1, ec_2, \dots)$$

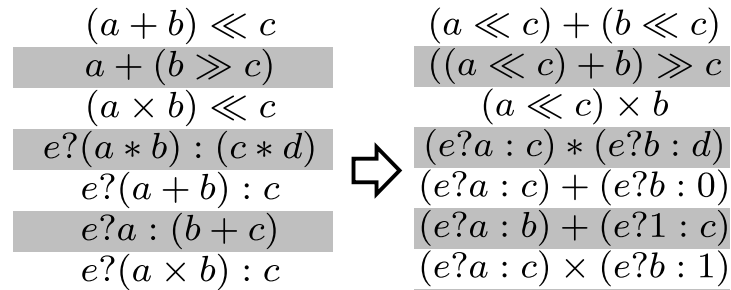
[1] Greg Nelson, Techniques for Program Verification, 1980.

[2] Ross Tate et al., Equality Saturation: A New Approach to Optimization, POPL 2009.

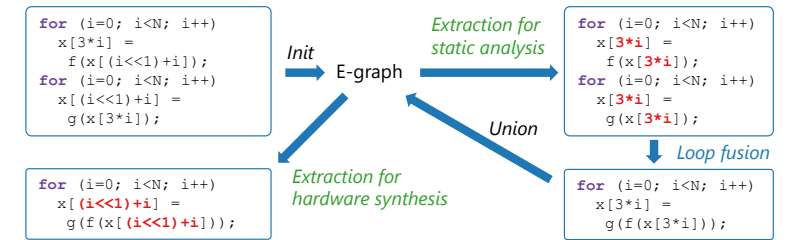
Broad Applications of E-Graphs



IMpress: large multiplication [FCCM'22]



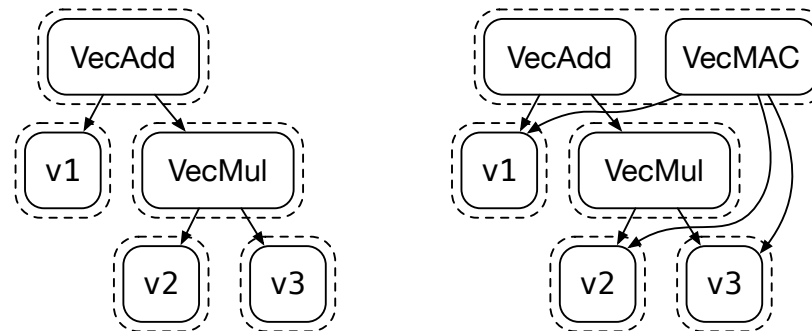
Rover: RTL optimization [TCAD'24]



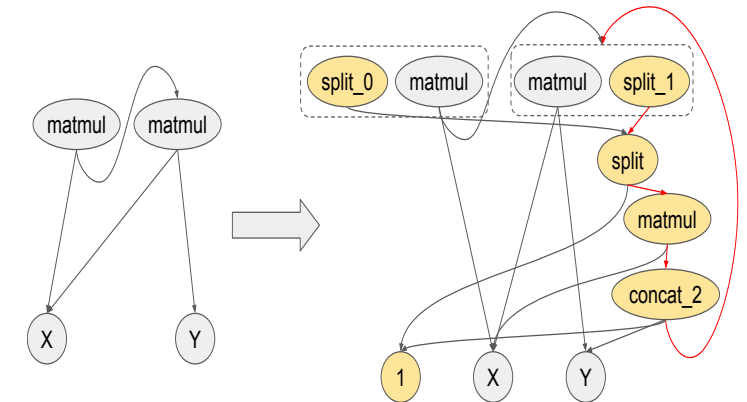
SEER: high-level synthesis [ASPLOS'24]

$A * (B + C) = A * B + A * C$
 $\sum_i (A + B) = \sum_i A + \sum_i B$
 If $i \notin A$, $A * \sum_i B = \sum_i (A * B)$ (else rename i)
 $\sum_i \sum_j A = \sum_{i,j} A$
 If $i \notin \text{Attr}(A)$, then $\sum_i A = A * \text{dim}(i)$
 $A + (B + C) = +(A, B, C)$ (assoc. & comm.)
 $A * (B * C) = *(A, B, C)$ (assoc. & comm.)

SPORES: linear algebra optimization for ML [VLDB'20]



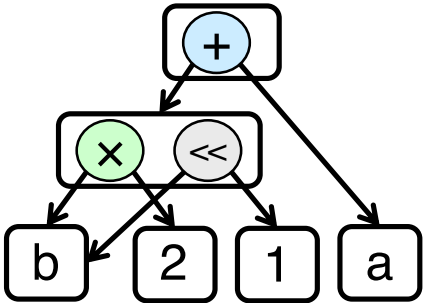
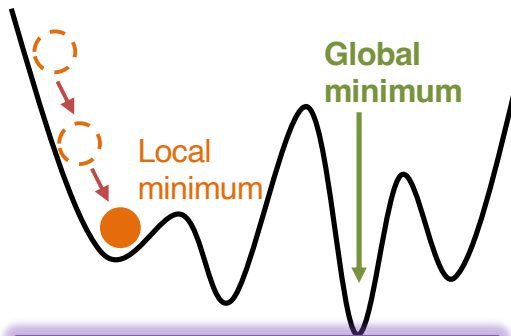
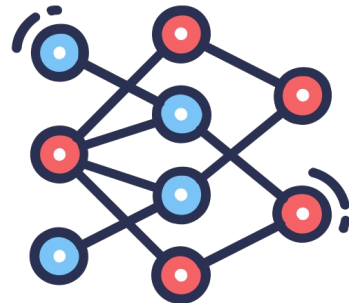
Diospyros: DSP compilation [ASPLOS'21]



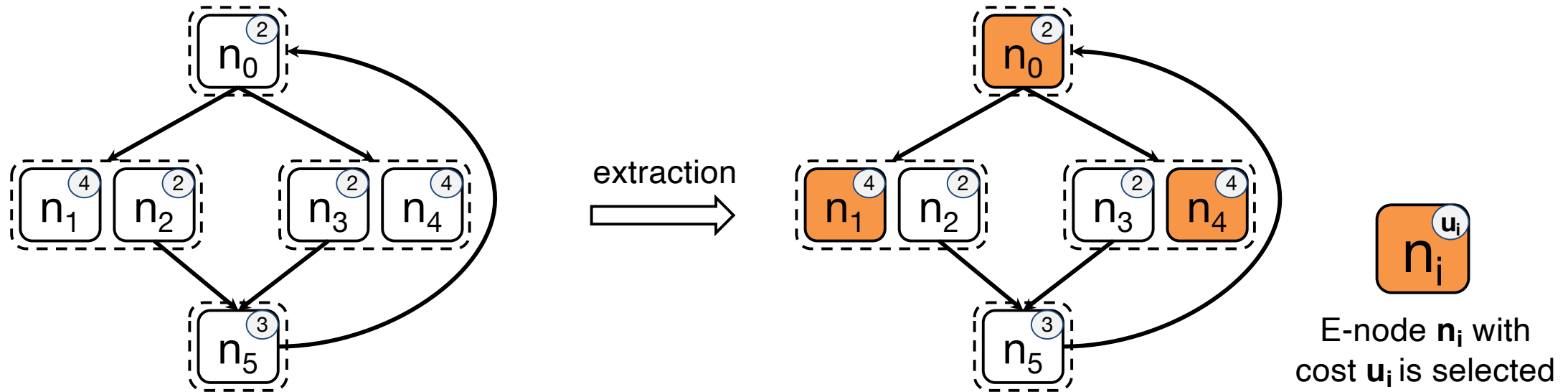
TENSAT: tensor graph optimization [MLSys'21]

Many Others ...

A Radical Ground-up Approach?

Compiler / Synthesis Optimization	Transformation Space	Search Method	Objective Function
Current Approach ↓ Differentiable Approach?	Phase ordered transforms (mostly local) ↓ Formal, compact encoding of equivalent transforms  Meta IR	Heuristics running on CPUs (mostly 1 thread) ↓ Parallelized global optimization  Gradient Descent	Hand-crafted cost models (mostly linear costs) ↓ Realistic, non-linear cost models  Learnable Cost

The E-Graph Extraction Problem



Goal: Extract the lowest-cost legal subgraph based on a given cost function

Constraints:

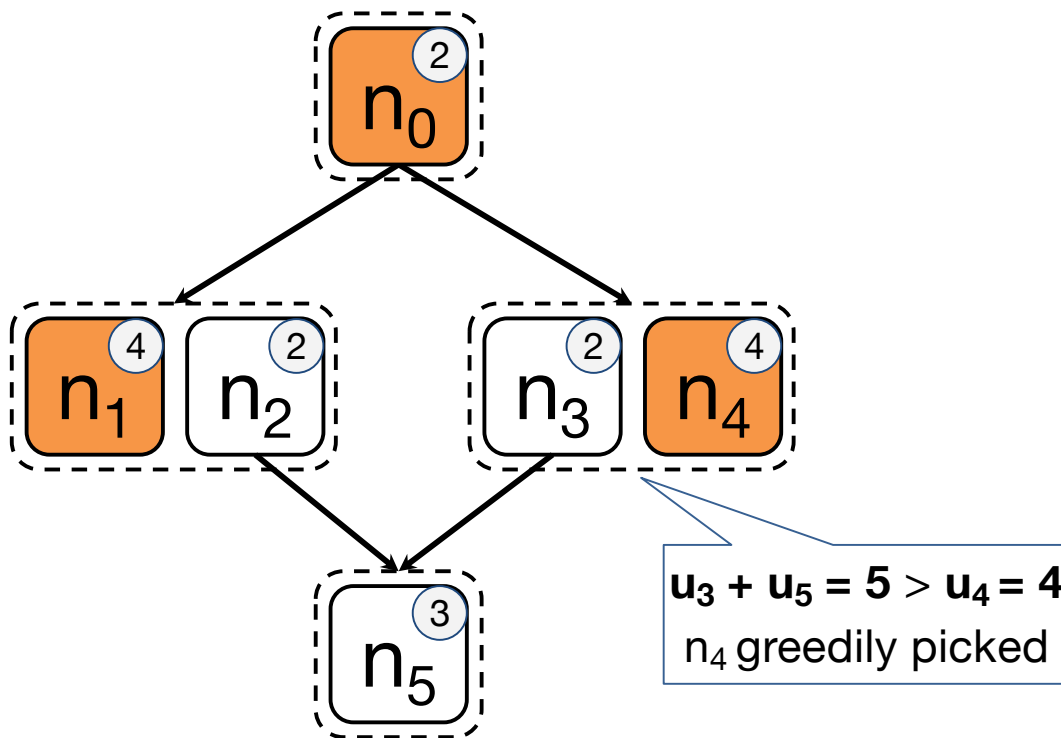
- Root e-class (n_0) must be selected
- Exactly one e-node per chosen e-class must be selected
- All child e-classes of a chosen e-node must be selected
- No cycles after extraction

NP-hard in general, even on acyclic e-graphs

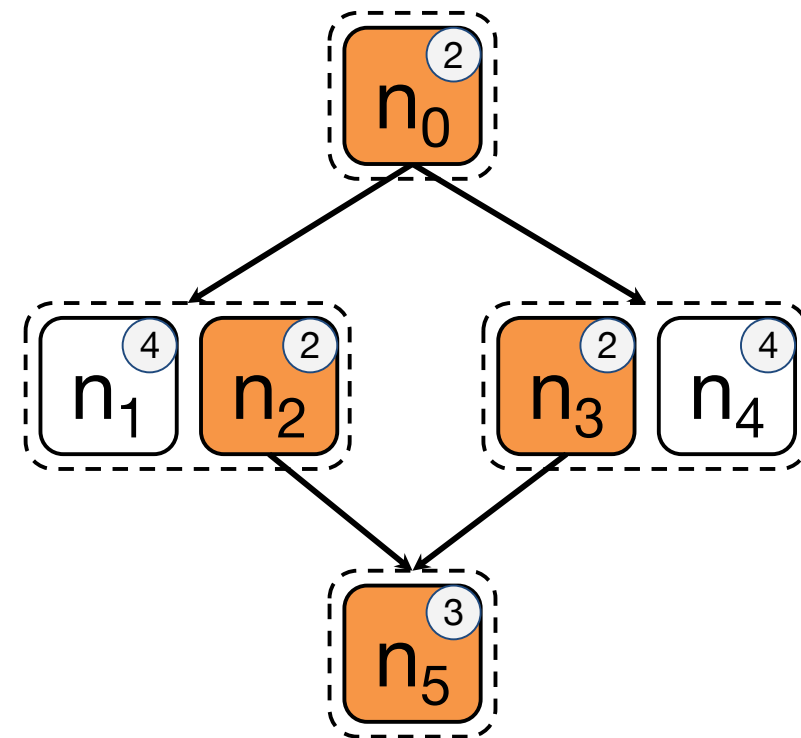
Existing Extraction Methods

- **A popular heuristic**

- A fast, iterative (greedy-ish) algorithm that selects e-nodes with min costs
- **Only supports linear cost functions** (i.e., weighted sum of individual node costs)
- **Suboptimal**



Heuristic extraction cost: $2 + 4 + 4 = 10$



Optimal extraction cost: $2 + 2 + 2 + 3 = 9$

Our Approach: SmoothE

- ▶ A **fully differentiable** approach to e-graph extraction
- ▶ **Continuous global optimization** that supports (learnable) nonlinear cost
- ▶ **GPU-accelerated** and compatible with modern ML frameworks

ASPLOS'25 Best Paper Award



SmoothE: Differentiable E-Graph Extraction

Yaohui Cai
yc2632@cornell.edu
Cornell University
Ithaca, New York, USA

Kaixin Yang
ky427@cornell.edu
Cornell University
Ithaca, New York, USA

Chenhui Deng
cd574@cornell.edu
Cornell University
Ithaca, New York, USA

Cunxi Yu
cunxiyu@umd.edu
University of Maryland, College Park
College Park, Maryland, USA

Zhiru Zhang
zhiruz@cornell.edu
Cornell University
Ithaca, New York, USA

Abstract

E-graphs have gained increasing popularity in compiler optimization, program synthesis, and theorem proving tasks. They enable compact representation of many equivalent expressions and facilitate transformations via rewrite rules without phase ordering limitations. A major benefit of using e-graphs is the ability to explore a large space of equivalent expressions, allowing the extraction of an expression that best meets certain optimization objectives (or cost models). However, current e-graph extraction methods often face unfavorable scalability-quality trade-offs and only support simple linear cost functions, limiting their applicability to more realistic optimization problems.

In this work, we propose *SmoothE*, a differentiable e-graph extraction algorithm designed to handle complex cost models and optimized for GPU acceleration. More specifically, we approach the e-graph extraction problem from a probabilistic perspective, where the original discrete optimization is relaxed to a continuous differentiable form. This formulation supports any differentiable cost functions and enables efficient searching for solutions using gradient descent. We implement SmoothE in PyTorch to leverage the advancements of the modern machine learning ecosystem. Additionally, we introduce performance optimization techniques to exploit sparsity and data parallelism. We evaluate SmoothE on a variety of realistic e-graphs from five different applications using three distinct cost models, including both linear and non-linear ones. Our experiments demonstrate that SmoothE consistently achieves a favorable trade-off between scalability and solution quality.

CCS Concepts: • Computing methodologies → Machine learning; • Software and its engineering → Compilers; General programming languages.

Keywords: Machine learning for systems; Compilers; Programming languages; Equivalence graph

ACM Reference Format:

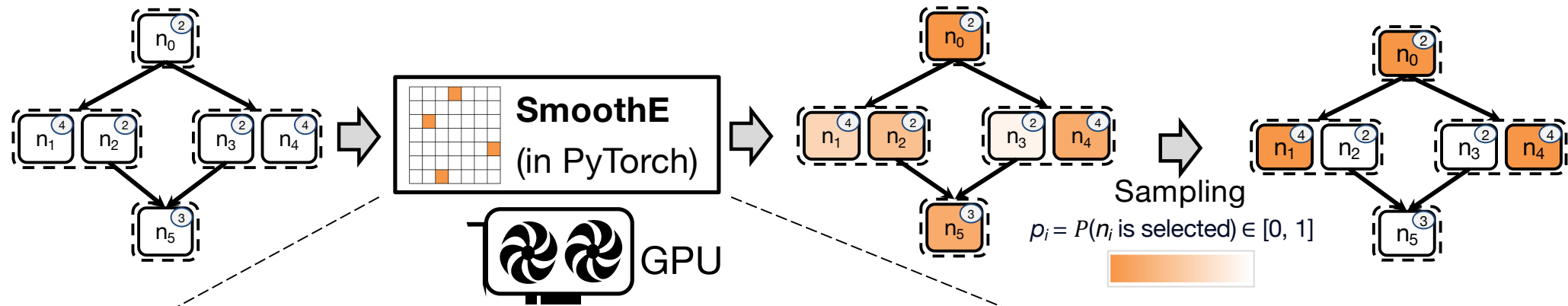
Yaohui Cai, Kaixin Yang, Chenhui Deng, Cunxi Yu, and Zhiru Zhang. 2025. SmoothE: Differentiable E-Graph Extraction. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '25)*, March 30–April 3, 2025, Rotterdam, Netherlands. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3669940.3707262>

1 Introduction

Term rewriting [17], widely employed in compiler optimizations [8, 34, 44] and theorem proving [15, 18], transforms programs into functionally equivalent but more efficient forms. Traditional methods apply the rewrites sequentially in a predetermined order, significantly affecting performance—a challenge known as the *phase ordering* problem [44, 50].

Equality saturation addresses the phase ordering issue by using the *equivalence graph* (*e-graph*), a data structure that compactly represents a set of expressions (i.e., *e-nodes*) and their equivalence relations (i.e., *e-classes*) [6, 33]. The rewrite rules are applied collectively, encoding all functionally equivalent solutions on a single e-graph. This enables the selection of the most cost-efficient (or performant) one

SmoothE in a Nutshell: Differentiable E-Graph Extraction

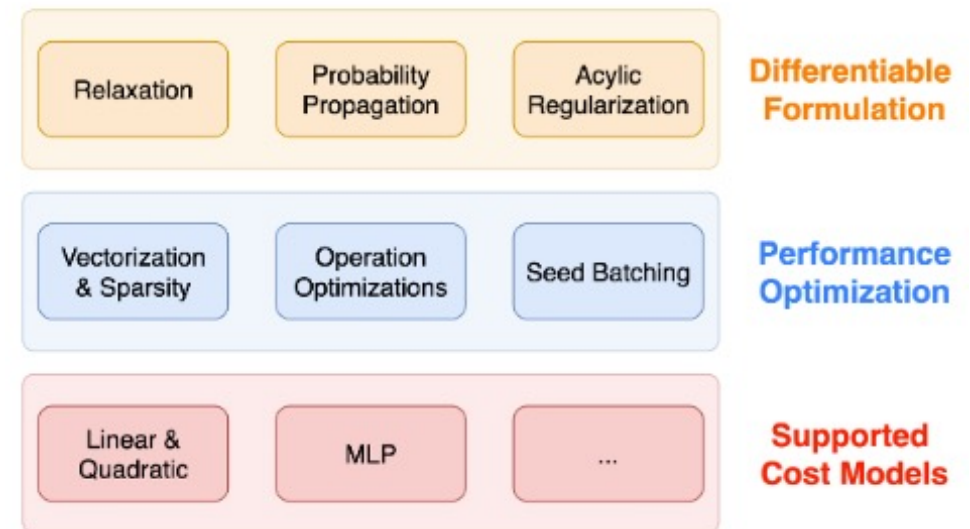


$$\text{minimize}_{\theta} \text{cost}(\mathbf{p}) + \lambda \text{Tr}(\mathbf{e}^{\mathbf{A}})$$

where $\mathbf{cp} = \text{softmax}(\theta)$ Acyclicity penalty

$\mathbf{p} = \text{LBP}(\mathbf{cp})$ learnable params

Loopy belief propagation (LBP) for completeness constraints



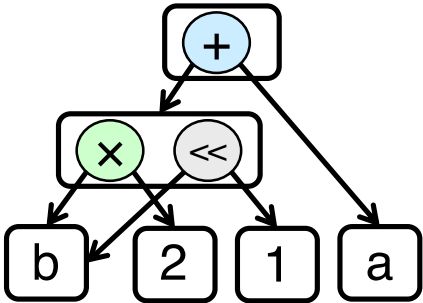
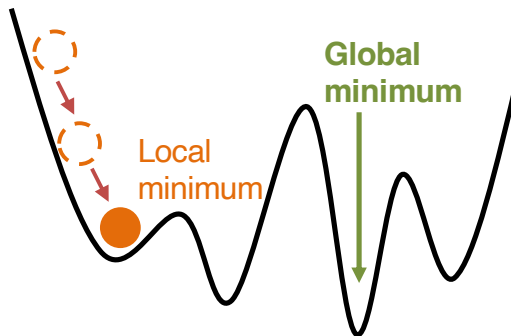
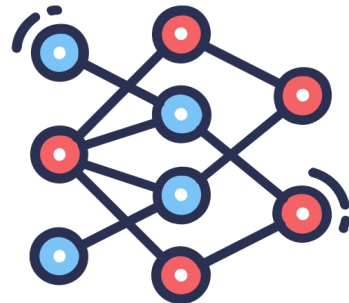
Some SmoothE Results on a Logic Synthesis Task

Benchmarks	Extraction Cost			Improvement over baseline
	Baseline ⁺	ILP	SmoothE	
Adder	313	255	265	15.3%
Barrel Shifter	1395	1201*	1238	11.3%
Divisor	31308	27280*	24260	22.5%
Log2	14078	13098*	9881	29.8%
Multiplier	9911	11645*	7197	27.4%
Sine	2417	1913*	1841	23.8%
Square-Root	11752	12592*	7140	39.2%

⁺: a popular iterative heuristic method for e-graph extraction

* ILP timed out after 6 hours

A Radical Ground-up Approach?

Compiler / Synthesis Optimization	Transformation Space	Search Method	Objective Function
Current Approach ↓ Differentiable Approach?	Phase ordered transforms (mostly local) ↓ Formal, compact encoding of equivalent transforms  Meta IR	Heuristics running on CPUs (mostly 1 thread) ↓ Parallelized global optimization  Gradient Descent	Hand-crafted cost models (mostly linear costs) ↓ Realistic, non-linear cost models  Learnable Cost

Learnable Cost Models using GNNs

QoR Estimation	Publication
Post-HLS Resource & Latency	GNN4HLS [Ferretti et al. TODAES'22]
Post-HLS Resource & Latency	GNN-DSE [Sohrabizadeh et al. DAC'22]
Post-PnR Resource & Timing	IronMan [Wu et al. GLSVLSI'21]
Post-PnR Resource & Timing	— [Wu et al. DAC'22]
Post-PnR Resource & Timing	IronMan-Pro [Wu et al. TCAD'23]
Post-PnR Power	PowerGear [Lin et al. DATE'22]
Post-PnR Power	HL-Pow [Lin et al. TCAD'23]

arXiv:2102.08138v2 [cs.AR] 8 Dec 2021

arXiv:2201.06848v1 [cs.LG] 18 Jan 2022

900

IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 42, NO. 3, MARCH 2023

Automated Accelerator Optimization Aided by Graph Neural Networks

Atefeh Sohrabizadeh, Yunsheng Bai, Yizhou Sun, and Jason Cong
Computer Science Department, University of California - Los Angeles, USA
[atefehsz.yb, yzsun, cong]@cs.ucla.edu

Abstract
Using High-Level Synthesis (HLS), the hardware designers must describe only a high-level behavioral flow of the design. However, it still can take weeks to develop a high-performance architecture mainly because there are many design choices at a higher level to explore. Besides, it takes several minutes to hours to evaluate the design with the HLS tool. To solve this problem, we model the HLS tool with a graph neural network that is trained to be used for a wide range of applications. The experimental results demonstrate that our model can estimate the quality of design in milliseconds with high accuracy, resulting in up to 79% speedup (with an average of 48%) for optimizing the design compared to the previous state-of-the-art work relying on the HLS tool.

ACM Reference Format:
Atefeh Sohrabizadeh, Yunsheng Bai, Yizhou Sun, and Jason Cong. 2022. Automated Accelerator Optimization Aided by Graph Neural Networks. In *Proceedings of the 39th ACM/IEEE Design Automation Conference (DAC) (DAC '22)*, July 10–14, 2022, San Francisco, CA, USA. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3489517.3520469>

1 Introduction
High-Level Synthesis (HLS) was introduced to simplify the FPGA programming by raising the abstraction level in design and soon was embraced by both academia and industry [4, 16]. This is because the HLS tools let the designers optimize their microarchitecture quickly by inserting a few synthesis directives in the form of pragmas. This feature can potentially help shorten the design development cycle. However, not every HLS design has a good quality of results [17]. Thus, one often has to explore many design choices for each new application since the solution space grows exponentially by the number of candidate pragmas. This can negatively impact the design turn-around times.
To speed up the design optimization, a new line of research has been created with the focus on automating the design space exploration (DSE) for optimizing the microarchitecture. As summarized in [14], the previous studies either use the HLS tool directly [17, 24], or develop a model to mimic the HLS tool [11, 26] for evaluating a design point. Relying on the HLS tool to evaluate a solution can increase the DSE time significantly as each design candidate would have a long evaluation time (minutes to hours) that forces us to explore a reduced set of the solution space. While utilizing a model can potentially speed up the process, a simple analytical model cannot capture the different heuristics used by the tool [14]. Adopting a learning algorithm can help with increasing the accuracy. However, the related works build a separate learning model per application and the results from one application are not transferred to another one. A nice effort was made in Kwon et al. [7] for transfer learning using a Multi-Layer Perceptron (MLP) network. Nonetheless, they only use the pragma configurations as the input to the model, which can result in considerable loss since the program semantics are missing (see Section 5.2).
A few of the very recent works have proposed to use Graph Neural Network (GNN) for predicting the design's quality [18, 21].

Ustun et al. [18] proposes a GNN-based model to learn the operation mapping to FPGA's resources for delay prediction in HLS. IronMan [21] uses GNN to predict the performance of the program under different resource allocations (DSP or LUT) to the computation nodes. Although their studies clearly demonstrate the value and power of using GNNs, none of these works include the pragmas in their input representation so their models cannot be used for finding the best design configuration.
In this paper, we aim to automate the design optimization using GNN with the support for model generalization by developing a framework called GNN-DSE. We first build a model to evaluate a design quickly, in milliseconds, without the invocation of the HLS tool. Since the HLS tools employ many heuristics to optimize a design and the design parameters affect each other, we let a deep learning model learn their impact. Furthermore, as the current HLS tools optimize the design based on specific code patterns, it is important to identify the different code patterns and learn their effect to be able to transfer the knowledge we gained from one application to another. As such, we represent the program as a graph which includes the program information in the form of control, data, call, and pragma flows and exploit a GNN to extract the required features of the graph for predicting the objectives. We propose several techniques for improving the accuracy of the model including Jumping Knowledge Networks (JKN) [23], node attention [9], and multi-head objective prediction. To demonstrate the effectiveness of our model, we build a DSE on top of it to find the Pareto-optimal design points. We show that not only can GNN-DSE find the Pareto-optimal design points for the kernels that were included in its training set, it can also generalize to the kernels outside of its database and detect their Pareto-optimal design points. This paper is the first work to employ a graph representation that captures both the program semantics and the pragmas, and to build a single predictive model for several applications with transferring learning capability. In this paper, we target Xilinx FPGAs as an example but our approach is tool-independent and extendable to Intel FPGAs as well.
In summary, this paper makes the following contributions:
• We propose a graph-based program representation for optimizing FPGA designs which includes both the program context and the pragma flow.
• We develop a learning model based on Graph Neural Network (GNN) as a surrogate of the HLS tool for assessing a design point's quality in milliseconds and propose several techniques for improving its accuracy.
• We build an automated framework, GNN-DSE¹, to gather a database of FPGA designs, train a learning model for predicting the design's objectives, and run a design space exploration based on the model to close-in on a high-performing design point.
• The experimental results demonstrate that not only can GNN-DSE find the Pareto-optimal design points for the kernels in its database, but can also optimize the unseen kernels by generalizing the knowledge it learned from its training set.

¹The codes are open-sourced at <https://github.com/UCLA-VAST/GNN-DSE>

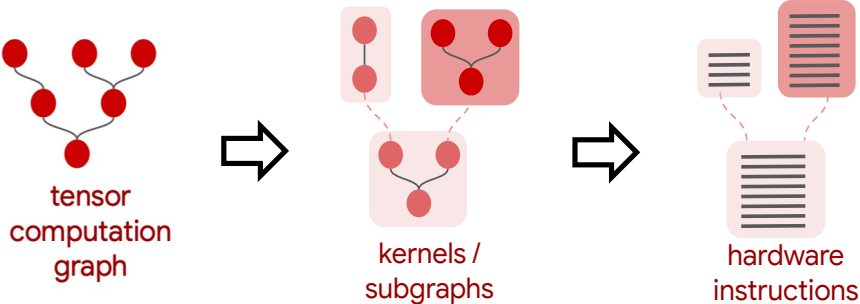
BLOG ›

2023: A year of groundbreaking advances in AI and computing



FRIDAY, DECEMBER 22, 2023

Posted by Jeff Dean, Chief Scientist, Google DeepMind & Google Research, Demis Hassabis, CEO, Google DeepMind, and James Manyika, SVP, Google Research, Technology & Society

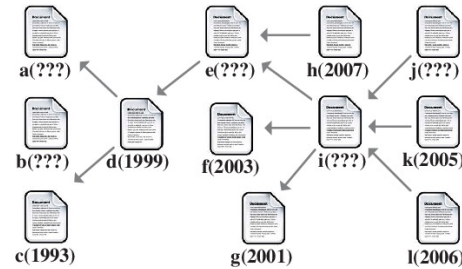


“We developed a novel **GNN** model to predict the properties of **tensor computation graphs**, enabling estimation of performance for **ML programs**.”

Traditional GNNs Not Ideal for Computation Graphs

Non-computation graphs

(e.g., social and citation networks)

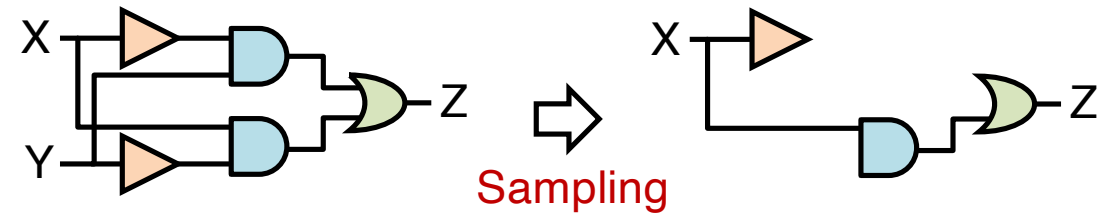


Homophily property – Nearby nodes tend to have similar attributes
(sampling may help both speed and accuracy)

Learning local structures is typically sufficient

Computation graphs

(e.g., logic networks, dataflow graphs)



Heterophily property – Nearby nodes often represent different operations
(graph sampling breaks functionality)

Global structures also matter (if not more)

Our Recent Efforts

ICLR'24

POLYNORMER: POLYNOMIAL-EXPRESSIVE GRAPH TRANSFORMER IN LINEAR TIME

Chenhui Deng, Zichao Yue, Zhiru Zhang
Cornell University, Ithaca, USA
{cd574, zy383, zhiruz}@cornell.edu

ABSTRACT

Graph transformers (GTs) have emerged as a promising architecture that is theoretically more expressive than message-passing graph neural networks (GNNs). However, typical GT models have at least quadratic complexity and thus cannot scale to large graphs. While there are several linear GTs recently proposed, they still lag behind GNN counterparts on several popular graph datasets, which poses a critical concern on their practical expressivity. To balance the trade-off between expressivity and scalability of GTs, we propose Polynormer, a polynomial-expressive GT model with linear complexity. Polynormer is built upon a novel base model that learns a high-degree polynomial on input features. To enable the base model permutation equivariant, we integrate it with graph topology and node features separately, resulting in local and global equivariant attention models. Consequently, Polynormer adopts a linear local-to-global attention scheme to learn high-degree equivariant polynomials whose coefficients are controlled by attention scores. Polynormer has been evaluated on 13 homophilic and heterophilic datasets, including large graphs with millions of nodes. Our extensive experiment results show that Polynormer outperforms state-of-the-art GNN and GT baselines on most datasets, even without the use of nonlinear activation functions. Source code of Polynormer is freely available at: github.com/cornell-zhang/Polynormer.

1 INTRODUCTION

As conventional graph neural networks (GNNs) are built upon the message passing scheme by exchanging information between adjacent nodes, they are known to suffer from *over-smoothing* and *over-squashing* issues (Oono & Suzuki, 2020; Alon & Yahav, 2021; Di Giovanni et al., 2023), resulting in their limited expressive power to (approximately) represent complex functions (Xu et al., 2018; Oono & Suzuki, 2020). Inspired by the advancements of Transformer-based models in language and vision domains (Vaswani et al., 2017; Dosovitskiy et al., 2021), graph transformers (GTs) have become increasingly popular in recent years, which allow nodes to attend to all other nodes in a graph and inherently overcome the aforementioned limitations of GNNs. In particular, Kreuzer et al. (2021) have theoretically shown that GTs with unbounded layers are universal equivariant function approximators on graphs. However, it is still unclear how to unlock the expressivity potential of GTs in practice since the number of GT layers is typically restricted to a small constant.

Global structures can be learned efficiently

⇒ SoTA accuracy on both homophilic & heterophilic graphs, including Google TPUGraphs (ICLR'24)

DAC'24

Less is More: Hop-Wise Graph Attention for Scalable and Generalizable Learning on Circuits

Chenhui Deng¹, Zichao Yue¹, Cunxi Yu², Gokce Sarar³, Ryan Carey³, Rajeev Jain³, Zhiru Zhang¹
¹Cornell University, ²University of Maryland, ³Qualcomm Technologies, Inc.
{cd574, zy383, zhiruz}@cornell.edu, cunxiyu@umd.edu, {gsarar, rcarey, rajeevj}@qti.qualcomm.com

ABSTRACT

While graph neural networks (GNNs) have gained popularity for learning circuit representations in various electronic design automation (EDA) tasks, they face challenges in scalability when applied to large graphs and exhibit limited generalizability to new designs. These limitations make them less practical for addressing large-scale, complex circuit problems. In this work we propose HOGA, a novel attention-based model for learning circuit representations in a scalable and generalizable manner. HOGA first computes hop-wise features per node prior to model training. Subsequently, the hop-wise features are solely used to produce node representations through a gated self-attention module, which adaptively learns important features among different hops without involving the graph topology. As a result, HOGA is adaptive to various structures across different circuits and can be efficiently trained in a distributed manner. To demonstrate the efficacy of HOGA, we consider two representative EDA tasks: quality of results (QoR) prediction and functional reasoning. Our experimental results indicate that (1) HOGA reduces estimation error over conventional GNNs by 46.76% for predicting QoR after logic synthesis; (2) HOGA improves 10.0% reasoning accuracy over GNNs for identifying functional blocks on unseen gate-level netlists after complex technology mapping; (3) The training time for HOGA almost linearly decreases with an increase in computing resources. Source code of HOGA is freely available at: github.com/cornell-zhang/HOGA.

1 INTRODUCTION

Recent years have seen a surge of interest in machine learning (ML) for electronic design automation (EDA), which holds great potential in achieving faster design closure and minimizing the need for extensive human supervision [9]. In particular, graph neural networks (GNNs) have become increasingly popular in the EDA community due to their ability to encode graph-structured data such as gate-level netlists into compact representations, which can be used for a multitude of downstream EDA applications, including quality of results (QoR) prediction and functional reasoning [13, 18].

However, scaling GNN training to large graphs is a notoriously challenging problem, which poses a serious concern on the practical benefit of GNNs on large-scale EDA problems. On the one

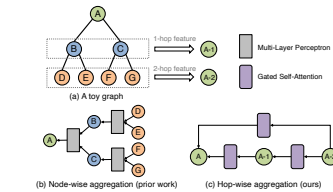


Figure 1: Comparison of HOGA and prior GNNs – (a) An example graph for illustration; (b) GNN computation graph; (c) Computation graph of our proposed approach, HOGA.

hand, unlike common datasets on social networks and molecular graphs, which consist of either a few large graphs or a large number of small graphs, the circuit datasets may contain numerous large graphs. For instance, the OpenABC-D benchmark provides 870k gate-level netlists, where each netlist consists of up to 240k logic gates [5]. Thus, training GNNs on such a large-scale circuit dataset is even more challenging than other graph-based applications. On the other hand, modern GNN models are built upon a message-passing paradigm, which learns representations through a recursive node-wise aggregation scheme shown in Figure 1(b). As a consequence, it is nontrivial to perform efficient distributed GNN training due to the node dependencies in a graph structure.

Apart from the scalability challenge, it is also underexplored how to make GNNs generalizable across different circuit designs. Although there are many customized GNNs previously proposed for various EDA applications, their model backbones mainly follow classic GNNs such as GCN [10] and GraphSAGE [8], which are not necessarily suitable for circuit problems. Consider a task of identifying functional blocks within circuits [18]. As distinct functional blocks may have different depths, the number of hops to be considered varies across nodes, which cannot be easily captured by common GNNs. Moreover, the high-order structures of functional blocks are also important yet ignored by the aforementioned GNN models. As a result, existing GNNs for EDA tasks often struggle

MLSys'25



GRAPH LEARNING AT SCALE: CHARACTERIZING AND OPTIMIZING PRE-PROPAGATION GNNs

Zichao Yue¹, Chenhui Deng^{2*}, Zhiru Zhang¹

ABSTRACT

Graph neural networks (GNNs) are widely used for learning node embeddings in graphs, typically adopting a message-passing scheme. This approach, however, leads to the *neighbor explosion* problem, with exponentially growing computational and memory demands as layers increase. Graph sampling has become the predominant method for scaling GNNs to large graphs, mitigating but not fully solving the issue. Pre-propagation GNNs (PP-GNNs) represent a new class of models that decouple feature propagation from training through pre-processing, addressing neighbor explosion in theory. Yet, their practical advantages and system-level optimizations remain underexplored. This paper provides a comprehensive characterization of PP-GNNs, comparing them with graph-sampling-based methods in training efficiency, scalability, and accuracy. While PP-GNNs achieve comparable accuracy, we identify data loading as the key bottleneck for training efficiency and input expansion as a major scalability challenge. To address these issues, we propose optimized data loading schemes and tailored training methods that improve PP-GNN training throughput by an average of 15× over the PP-GNN baselines, with speedup of up to 2 orders of magnitude compared to sampling-based GNNs on large graph benchmarks. Our implementation is publicly available at <https://github.com/cornell-zhang/preprop-gnn>.

1 INTRODUCTION

Message-passing-based graph neural networks (MP-GNNs) have become a cornerstone for graph representation learning, achieving success in various tasks like node classification (Veličković et al., 2018; Wu et al., 2023; Kipf & Welling, 2017), link prediction (Zhang & Chen, 2018; Schütt et al., 2017), and graph clustering (Zhang et al., 2019; Ying et al., 2018b; Tsitsulin et al., 2023). However, scaling MP-GNNs to large graphs remains a significant challenge.

The message-passing framework (Gilmer et al., 2017) consists of two iterative steps: (1) feature aggregation and (2) transformation. Within this framework, each node collects feature embeddings from its neighbors and then transforms them using a learnable function. We show the architecture of MP-GNN models in Figure 1. The main challenge in scaling MP-GNNs to large graphs stems from the “neighbor explosion” problem (Hamilton et al., 2017), where nodes must re-

footprint during message passing. Those models encompass node-wise sampling to limit neighborhood sizes per node (Chen et al., 2017; Hamilton et al., 2017), layer-wise sampling to reduce node counts per layer (Chen et al., 2018; Zou et al., 2019), and graph-wise sampling to control overall subgraph size (Chiang et al., 2019; Zeng et al., 2020). However, the sampling-based GNNs face several major limitations. First, node-wise sampling methods only partially mitigate the neighbor explosion problem, as their time complexity still increases exponentially with the number of layers. More importantly, the sampling algorithms modify the graph topology by design, which inevitably breaks the functionality of computation graphs such as logic networks (Wu et al., 2023) and dataflow graphs (Phothilimthana et al., 2024), resulting in accuracy degradation on their downstream tasks (Deng et al., 2024).

To circumvent the limitations of MP-GNNs, a new class of models known as pre-propagation GNNs (PP-GNNs) has

Feature pre-propagation enables scalable dense GNN training by avoiding message passing bottlenecks
⇒ much improved accuracy & generalizability on computation graphs (DAC'24)
⇒ 10-100x faster training than conventional message passing GNNs (MLSys'25)

Ongoing: Agentic E-Graph Rewriting for RTL Optimization

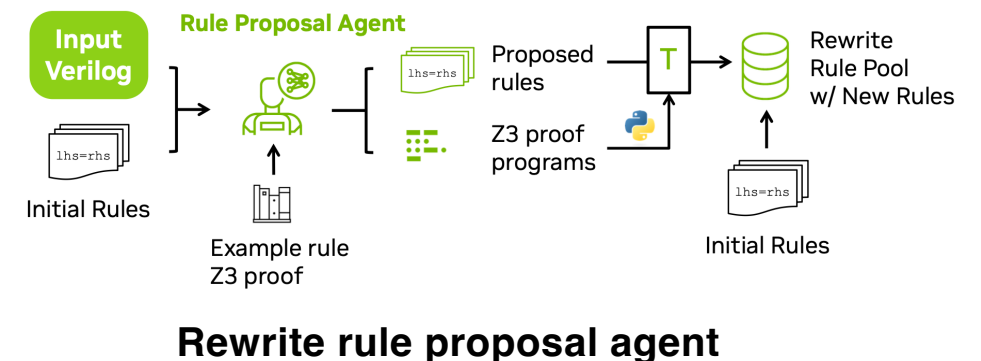
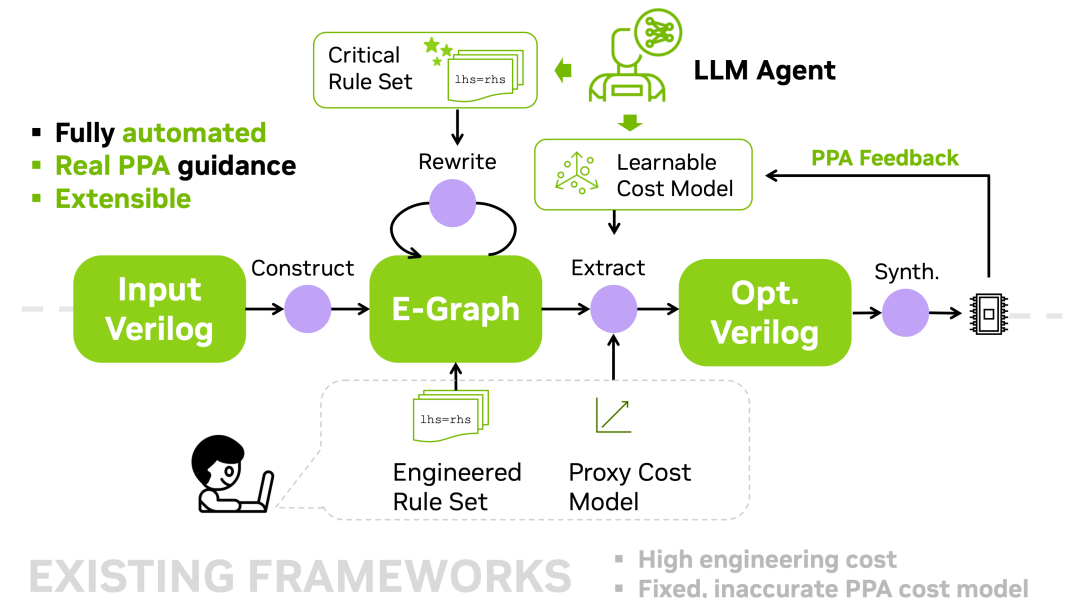
Goal: Improve RTL datapath optimizations by combining LLM agents with e-graph rewriting

Approach (in collab. with NVIDIA)

- ▶ Leverage LLM agents to **propose**, **prove**, **select** e-graph rewrite rules
- ▶ Instead of relying on heuristic cost models, use **real PPA feedback** from EDA tools to guide e-graph extraction

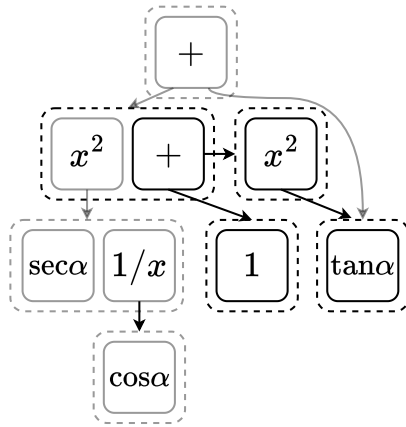
Preliminary Results on a suite of open-source RTL datapath-intensive benchmarks (to appear in MLCAD'25)

- ▶ Improves on average **24%** in area and **12%** in delay over a commercial synthesis tool
- ▶ Identifies Pareto frontier of PPA trade-offs



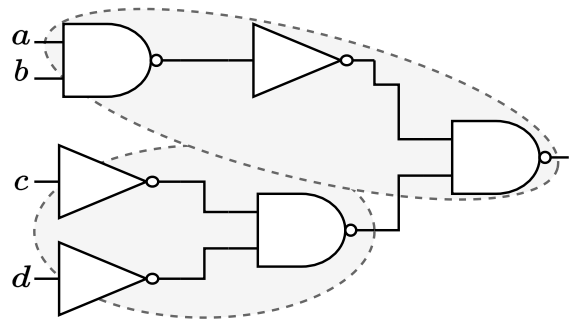
Ongoing: Differentiable Optimization Beyond E-Graph Extraction

Differentiable probabilistic optimization methods apply to a range of key compiler/synthesis problems—more results coming soon!



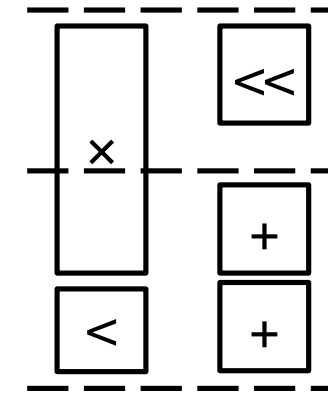
Term rewriting

(expression-, loop-level,
graph-level transforms)



Graph covering

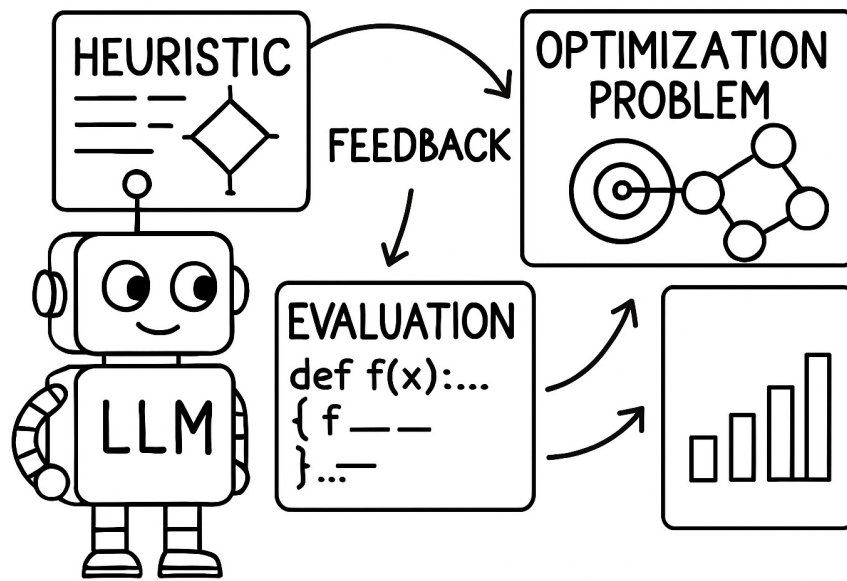
(instruction selection,
tech mapping)



Scheduling

(+ resource allocation,
module selection)

Can also be paired with other metaheuristics (e.g., evolutionary search) to better escape local minima



Research Question 2
To what extent can AI replace humans in crafting optimization algorithms for compilers?

HeuriGym: Agentic Benchmarking for LLM-Generated Heuristics

HeuriGym: An Agentic Benchmark for LLM-Crafted Heuristics in Combinatorial Optimization

Hongzheng Chen^{1*} Yingheng Wang^{1*} Yaohui Cai^{1*} Hins Hu^{1*} Jiajie Li^{1*}
Shirley Huang² Chenhui Deng³ Rongjian Liang³ Shufeng Kong¹
Haoxing Ren³ Samitha Samaranyake¹ Carla P. Gomes¹ Zhiru Zhang¹

¹ Cornell University ² Harvard University ³ NVIDIA Corporation
{hzchen,yingheng}@cs.cornell.edu, {yc2632,zh223,j14257}@cornell.edu

Abstract

While Large Language Models (LLMs) have demonstrated significant advancements in reasoning and agent-based problem-solving, current evaluation methodologies fail to adequately assess their capabilities: existing benchmarks either rely on closed-ended questions prone to saturation and memorization, or subjective comparisons that lack consistency and rigor. In this work, we introduce **HeuriGym**, an agentic framework designed for evaluating heuristic algorithms generated by LLMs for combinatorial optimization problems, characterized by clearly defined objectives and expansive solution spaces. HeuriGym empowers LLMs to propose heuristics, receive evaluative feedback via code execution, and iteratively refine their solutions. We evaluate nine state-of-the-art models on nine problems across domains such as computer systems, logistics, and biology, exposing persistent limitations in tool use, planning, and adaptive reasoning. To quantify performance, we propose the Quality-Yield Index (QYI), a metric that captures both solution pass rate and quality. Even top models like GPT-o4-mini-high and Gemini-2.5-Pro attain QYI scores of only 0.6, well below the expert baseline of 1. Our open-source benchmark aims to guide the development of LLMs toward more effective and realistic problem-solving in scientific and engineering domains.

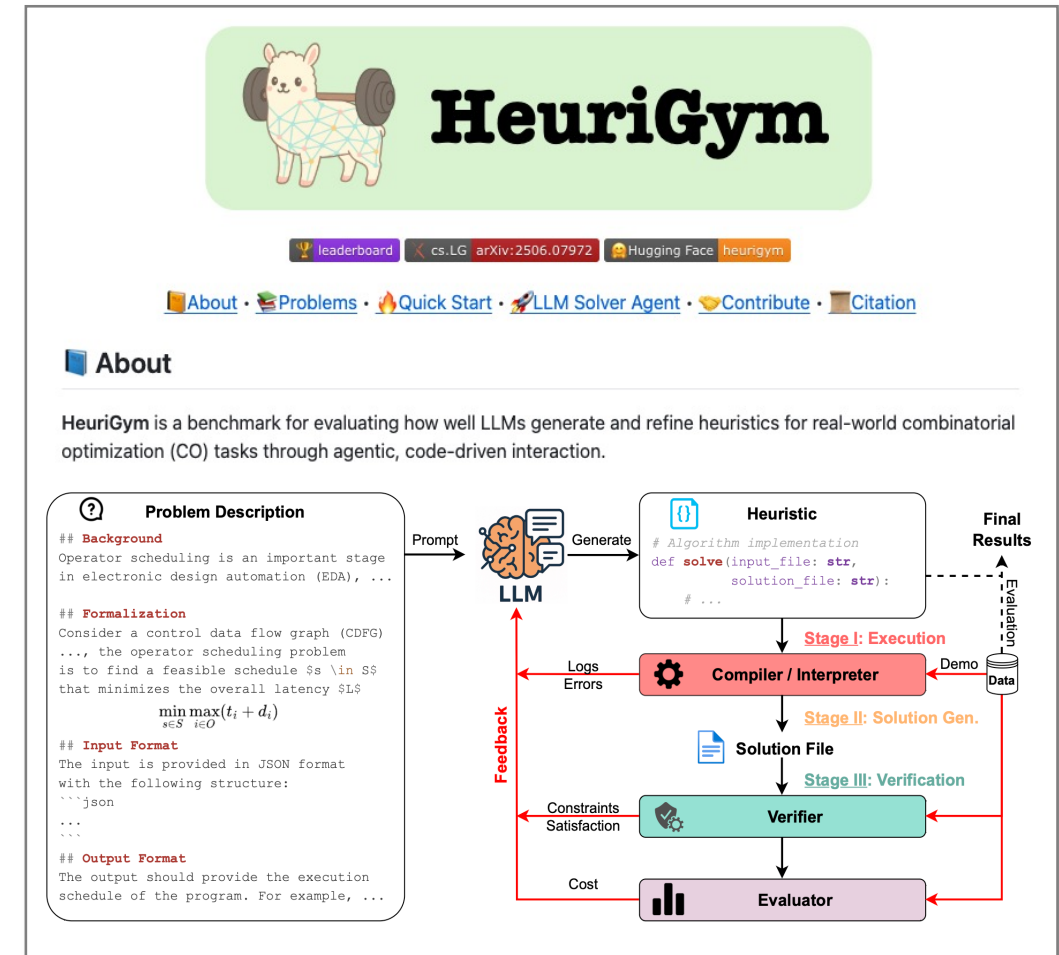
1 Introduction

Recent advancements in Large Language Models (LLMs) have significantly expanded their capabilities in complex reasoning and agent-based problem-solving, enabling applications ranging from automated code generation [25, 79, 177] to dynamic decision-making systems [126, 165]. Despite these breakthroughs, existing evaluation frameworks struggle to rigorously assess the full spectrum of LLMs' emergent abilities. Traditional benchmarks increasingly fail to capture the nuanced demands of real-world tasks that require iterative reasoning, creative algorithm design, and adaptive tool use. This limitation creates a critical gap in understanding whether LLMs can transcend pattern recognition and demonstrate genuine problem-solving ingenuity in real-world scenarios.

arXiv:2506.07972v1 [cs.LG] 9 Jun 2025



arxiv.org/abs/2506.07972



github.com/cornell-zhang/heurigym

HeuriGym Benchmarks

- ▶ A representative mix of 9 combinatorial optimizations problems
 - Well-defined objectives
 - Large solution spaces
 - Scalable data instances
 - Reproducible expert baselines

Domain	Problem	Difficulty
Electronic Design Automation (EDA)	Operator scheduling	★
	Technology mapping	★★
	Global routing	★★★
Compilers	E-graph extraction	★
	Intra-operator parallelism	★★
Computational Biology	Protein sequence design	★
	Mendelian error detection	★★
Logistics	Airline crew pairing	★★
	Pickup and delivery w/ time windows	★★★

HeuriGym Evaluation (1)

- A new metric tracks the LLM's ability to solve problems within i iterations:

$$\text{solve}_s@i := \frac{1}{N} \sum_{n=1}^N \mathbb{1}(\text{pass stage } s \text{ in the first } i\text{-th iteration})$$

Stage I: Code compiles successfully

Stage II: Executable produces non-empty output

Stage III: Output is verified as a feasible solution

Model	solve _{III}			solve _{II}			solve _I		
	@10	@5	@1	@10	@5	@1	@10	@5	@1
DeepSeek-V3	46.8%	42.7%	14.2%	87.6%	83.0%	66.1%	100.0%	100.0%	90.8%
DeepSeek-R1	73.4%	72.9%	44.0%	88.1%	88.1%	60.6%	100.0%	100.0%	71.6%
Gemini-2.5-Flash	67.4%	58.3%	25.2%	83.9%	79.4%	56.4%	100.0%	100.0%	72.9%
Gemini-2.5-Pro	65.1%	64.2%	20.2%	89.4%	89.0%	42.7%	100.0%	100.0%	51.4%
LLaMA-4-Maverick	35.8%	33.5%	6.0%	84.9%	74.3%	8.3%	85.3%	85.3%	13.3%
LLaMA-3.3-70B	33.9%	33.9%	20.6%	78.4%	78.4%	40.4%	99.5%	99.5%	61.9%
Qwen3-235B	45.9%	45.4%	38.5%	86.2%	83.0%	56.0%	100.0%	100.0%	70.6%
Claude-3.7-Sonnet	60.1%	58.7%	9.2%	97.7%	97.7%	41.3%	100.0%	100.0%	60.1%
GPT-o4-mini	74.8%	69.7%	53.2%	100.0%	100.0%	93.1%	100.0%	100.0%	100.0%

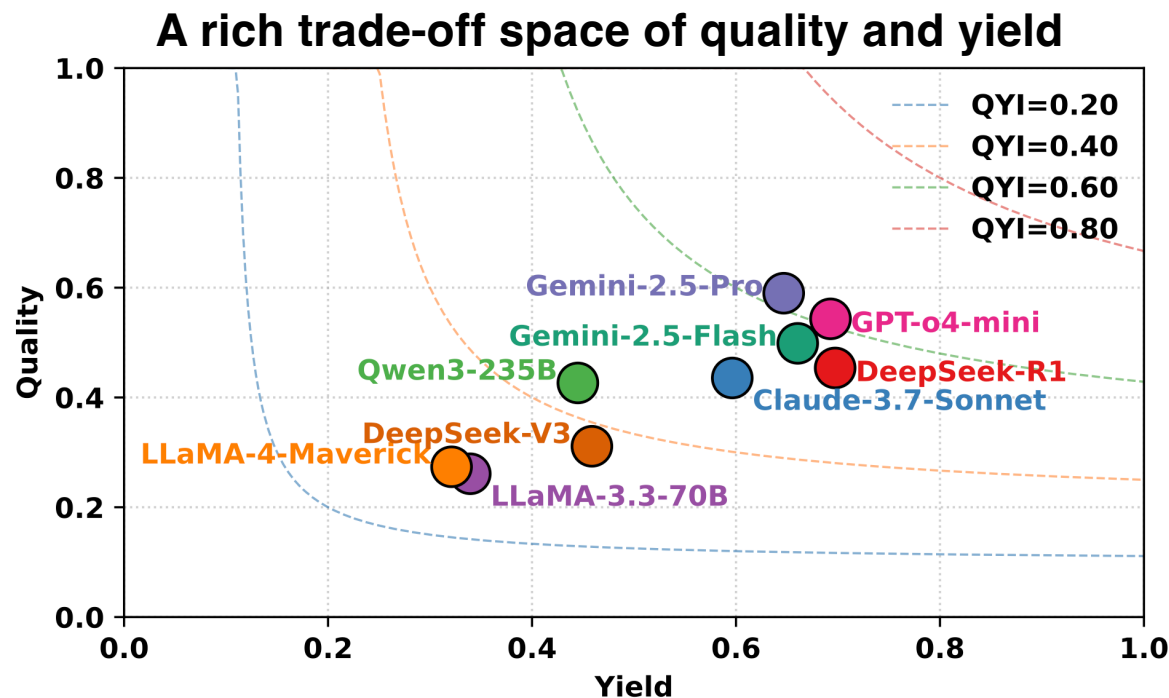
Even SoTA LLMs only yield ~70% feasible solutions

HeuriGym Evaluation (2)

Quality-Yield Index (QYI)

$$\text{QYI} = \frac{2 \cdot \text{Quality} \cdot \text{Yield}}{\text{Quality} + \text{Yield}} \quad \text{Quality} = \frac{1}{\hat{N}} \sum_{n=1}^{\hat{N}} \min \left(1, \frac{c_n^*}{c_n} \right) \quad \text{Yield} = \frac{\hat{N}}{N}$$

penalizes imbalanced values more strongly than arithmetic mean (similar to F-score)



- ▶ SOTA LLMs achieve a QYI score of ~0.6 (1.0 = human expert)
- ▶ More results available on HeuriGym leaderboard

 [cornell-zhang.github.io/heurigym](https://github.com/cornell-zhang/heurigym)

My Predictions – Parting Thought

Where are we now?



Where are we heading in the next few years?



What could be the next big game changer?



a true virtuous cycle