Qualcomm

January 31, 2023

Solving unsolvable combinatorial problems with AI

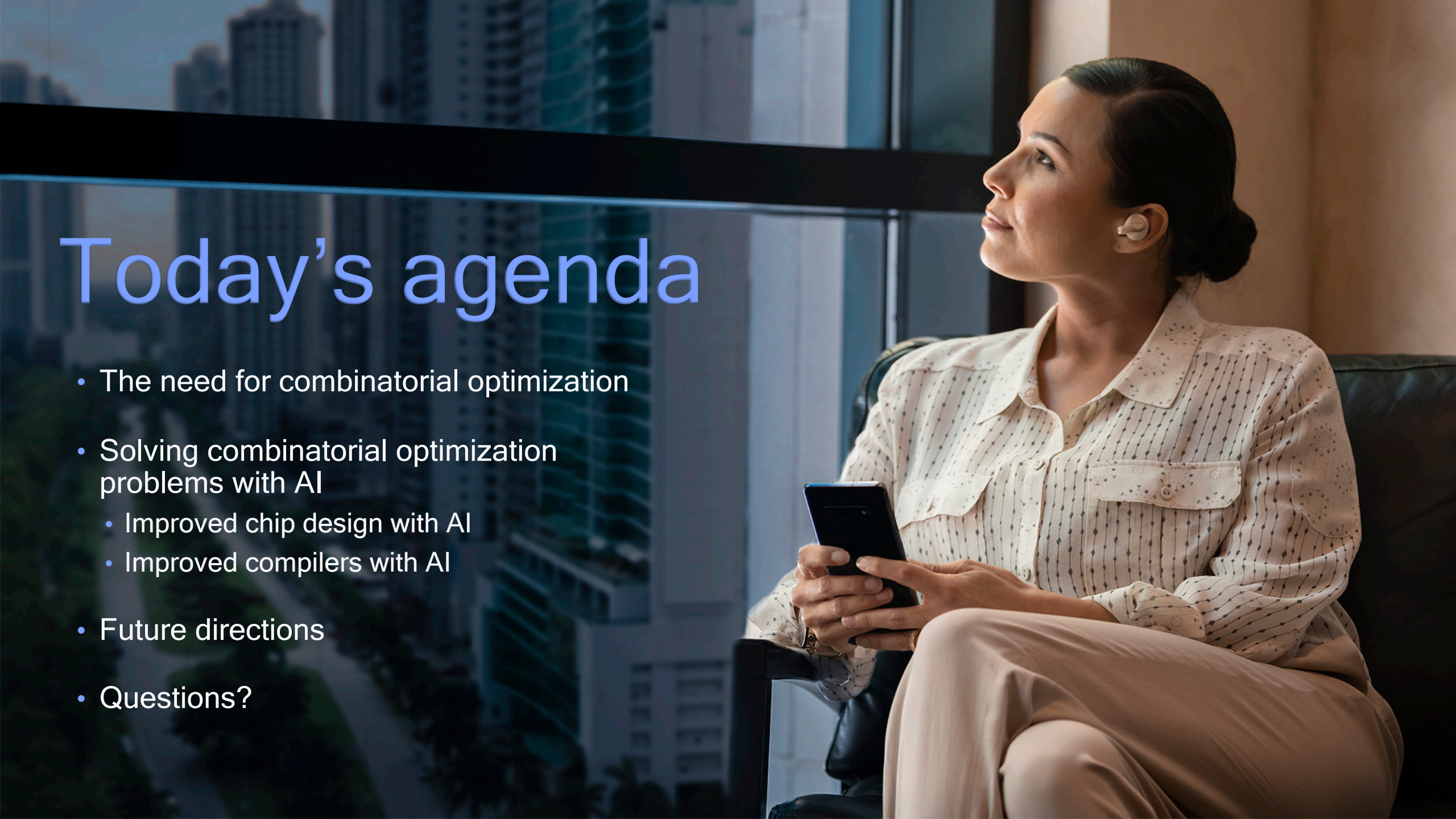
Chris Lott

Senior Director, Engineering
Qualcomm Technologies, Inc.

@QCOMResearch

Today's agenda

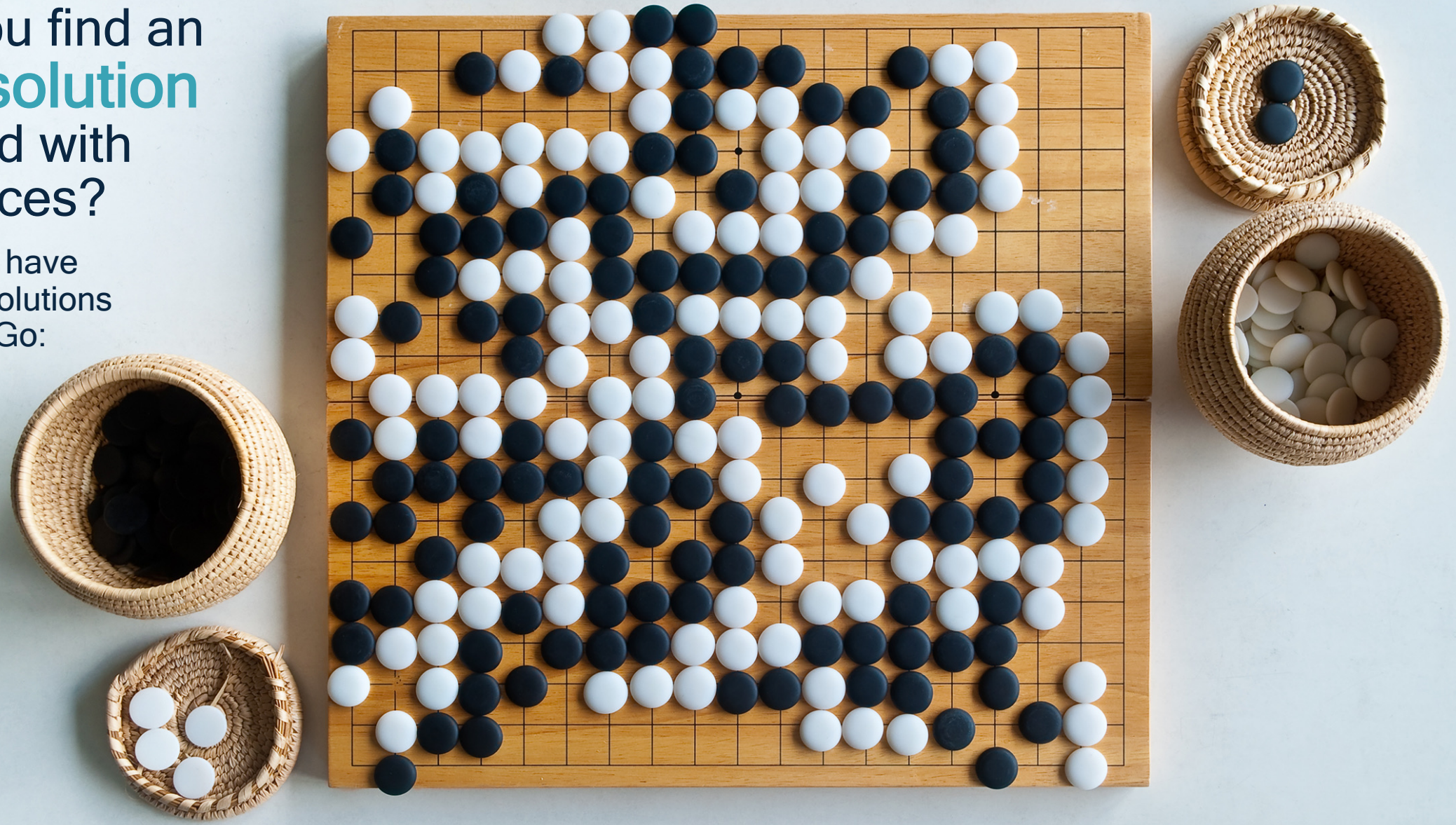
- The need for combinatorial optimization
- Solving combinatorial optimization problems with AI
  - Improved chip design with AI
  - Improved compilers with AI
- Future directions
- Questions?



How do you find an **optimal solution** when faced with many choices?

Some problems have more possible solutions than a game of Go:

$\sim 10^{170}$



Combinatorial optimization problems are all around us

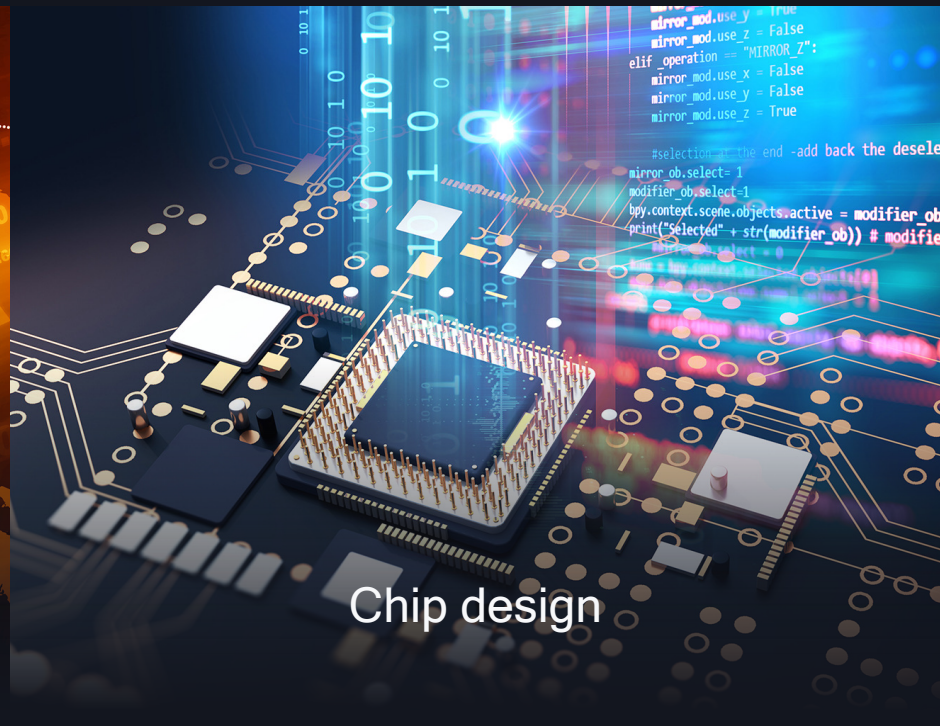
Finding solutions can provide significant benefits:

- Reducing cost
- Reducing time
- Increasing performance

Supply chain optimization

Hardware-specific compiler

Airline network planning



Chip design

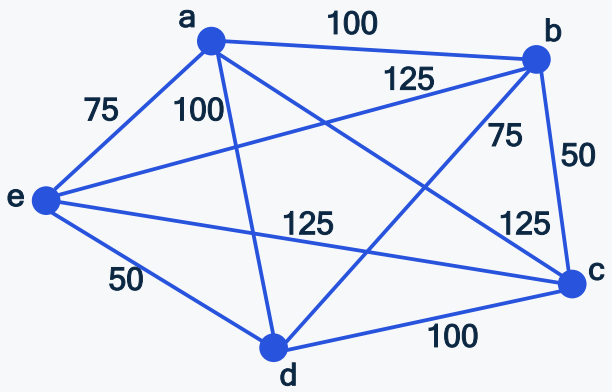
Traveling Salesman Problem (TSP)

Exemplifies the combinatorial optimization problem

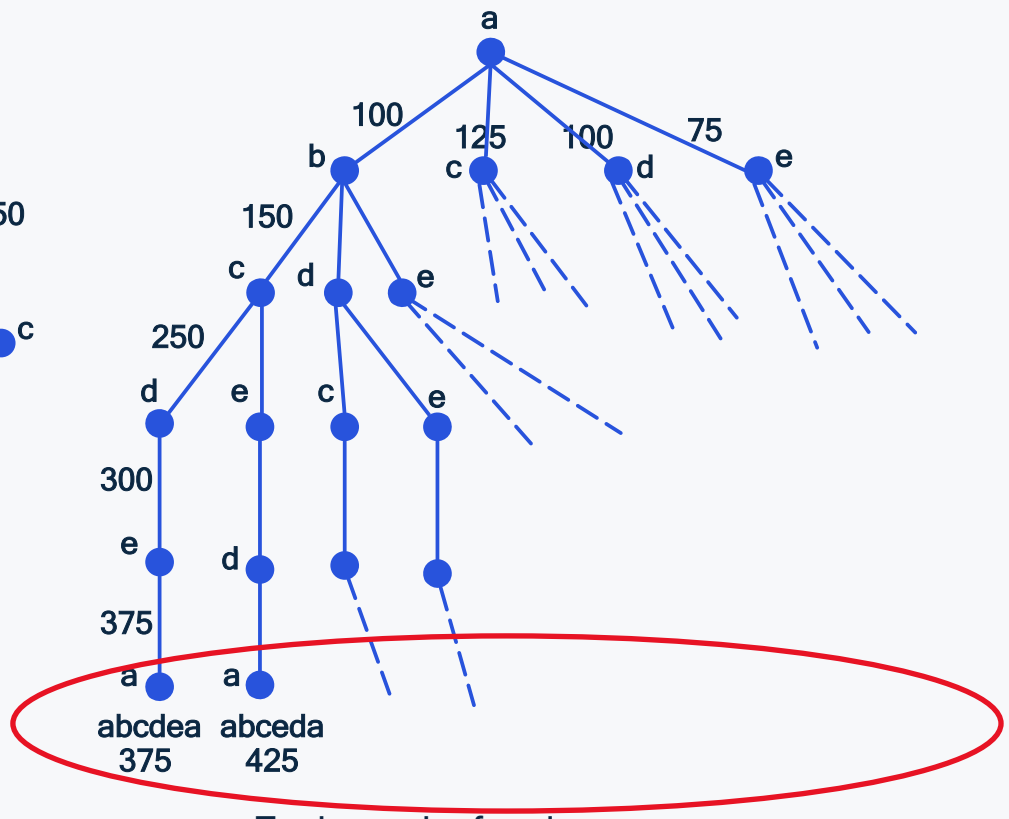
Given a set of N cities with travel distance between each pair, find the shortest path that visits each city exactly once

Example:
Using a brute force search method, if a computer can check a solution in a microsecond, then it would take 2 microseconds to solve 3 cities, 3.6 seconds for 11 cities, and 3857 years for 20 cities.

An instance of the traveling salesman problem



Search space



Each tree leaf node represents one full tour of the cities

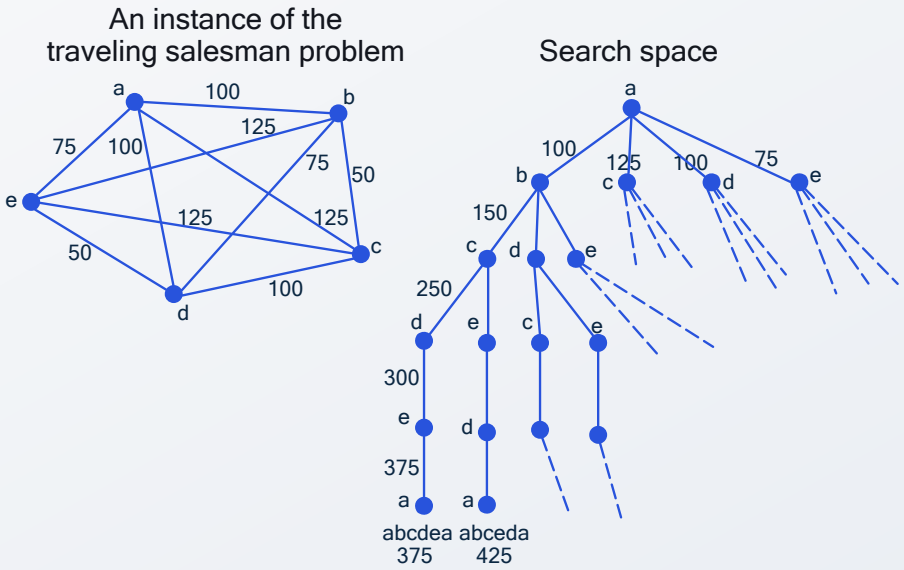
Search space
(represented as a tree)

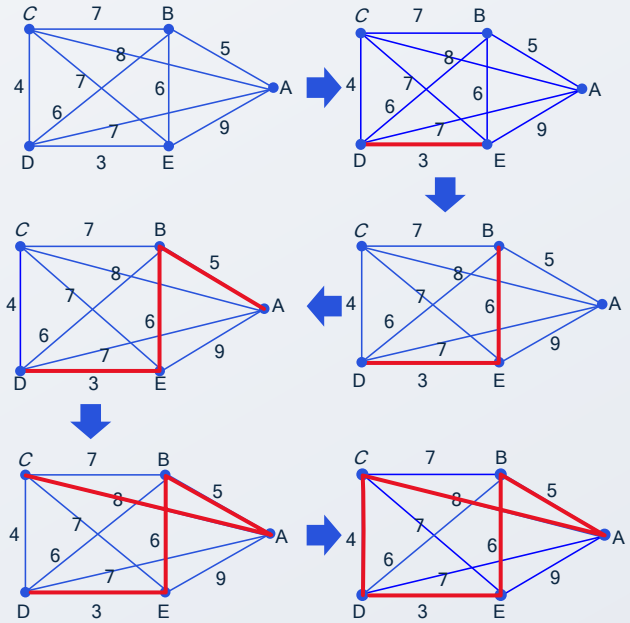
- Start at any city
- Choose from N-1 next cities
- Choose from N-2 next cities
- .....
- Choose last city and connect to start city

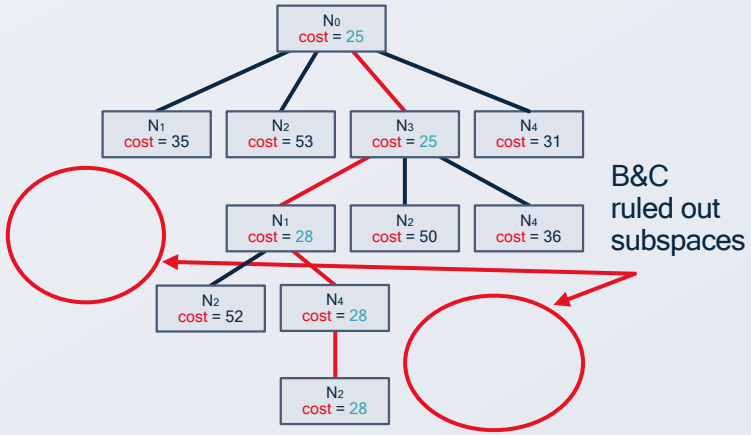
There are (N-1)! paths (tours) in the search tree. Full enumeration quickly becomes infeasible as N grows

How to approach solving TSP with brute force

Existing TSP solutions face challenges







Brute force method

- Full path enumeration
- Naive

- Scales as (N-1)!
  - infeasible for N>20

Heuristic methods

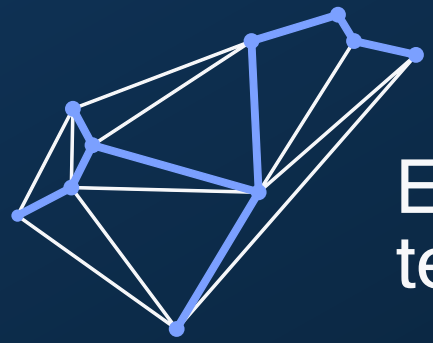
- The Nearest Neighbor method
- N-opt iterative path improvement

- Not guaranteed optimal
- Heuristics are problem-specific (human-designed)

Exact solver methods

- Dynamic programming
- Formulate as Integer Linear Programming (ILP), use Branch and Cut (B&C)
  - Uses branch and bound to rule out whole solution subspaces
  - Combine with problem-specific cutting planes

- Scales up to 1000's of nodes, but at high computational cost
- Problem-specific



Existing combinatorial optimization techniques have limitations

Scale

Search heuristics don’t scale to larger problems in acceptable compute time and cost, and do not guarantee satisfaction of all the constraints resulting in expensive manual intervention

Learning

Techniques don’t incorporate knowledge learned from solving many problems. They start each new problem instance from scratch.

How can we improve this situation with AI?

AI addresses challenges of traditional combinatorial optimization solutions

Leverages learned problem structure

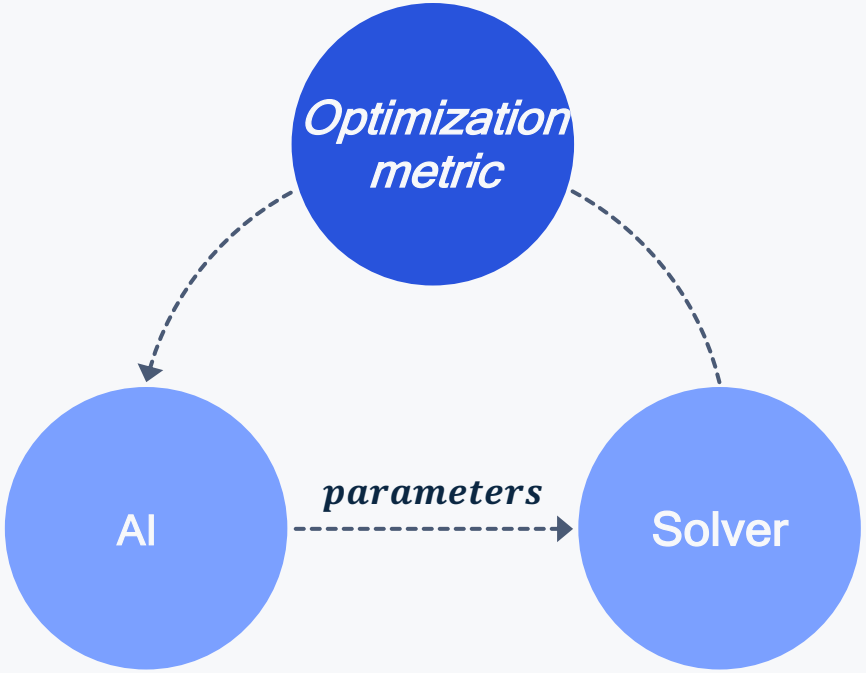
Scales to larger instances

Offers a general framework

Can achieve desired outcome with resource, cost, and time constraints

Notable prior work: "Attention, learn to solve routing problems!", ICLR 2019

AI process



Develop an AI algorithm that can learn correlation between design parameters and optimization metrics from a limited set of problem instances

For new instances, the AI algorithm uses an existing solver more efficiently by reducing parameter search space

Optimizing chip design with AI

Exploring Bayesian optimization to reduce combinatorial search space

“Bayesian Optimization for Macro Placement”, ICML 2022

Production cost reduction

- Chip Area
- Yield - more complex fab process
- Test Time

Design efficiency

- # of tools iterations
- License cost

Power-performance optimization

- System-level power
- Chip-level power

Capital expense (Capex)

- # of compute servers
- Emulation platforms

Competing combinatorial optimization objectives in chip design

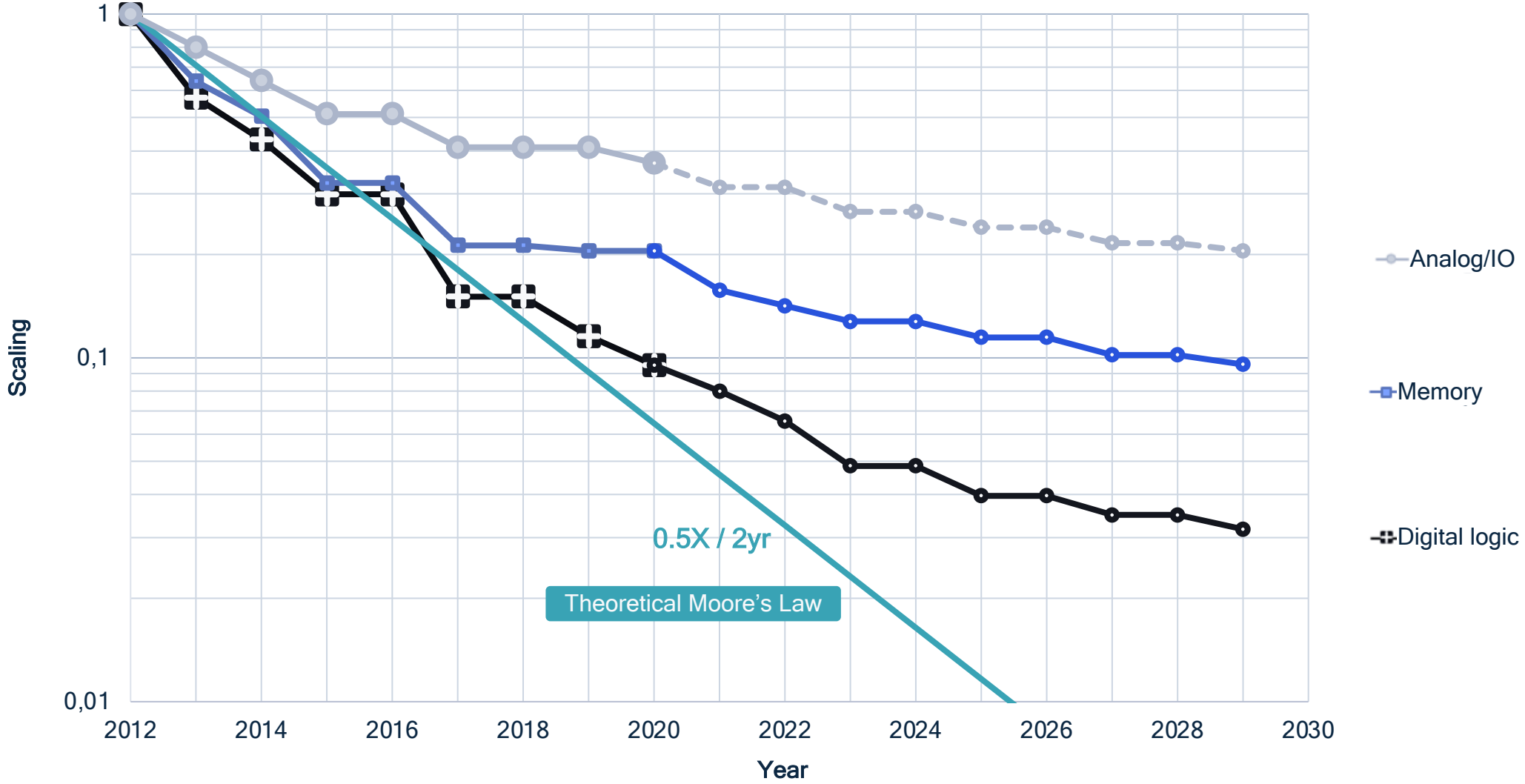
Need to account for all business metrics

Semiconductor scaling advantage is approaching a cliff

Foundation	Key elements	Disruption
Chips	Moore's Law PPA scaling	50% less gain
Design automation	Combinatorial optimization	100x PVT corners
Computing	Cloud servers	10x expense

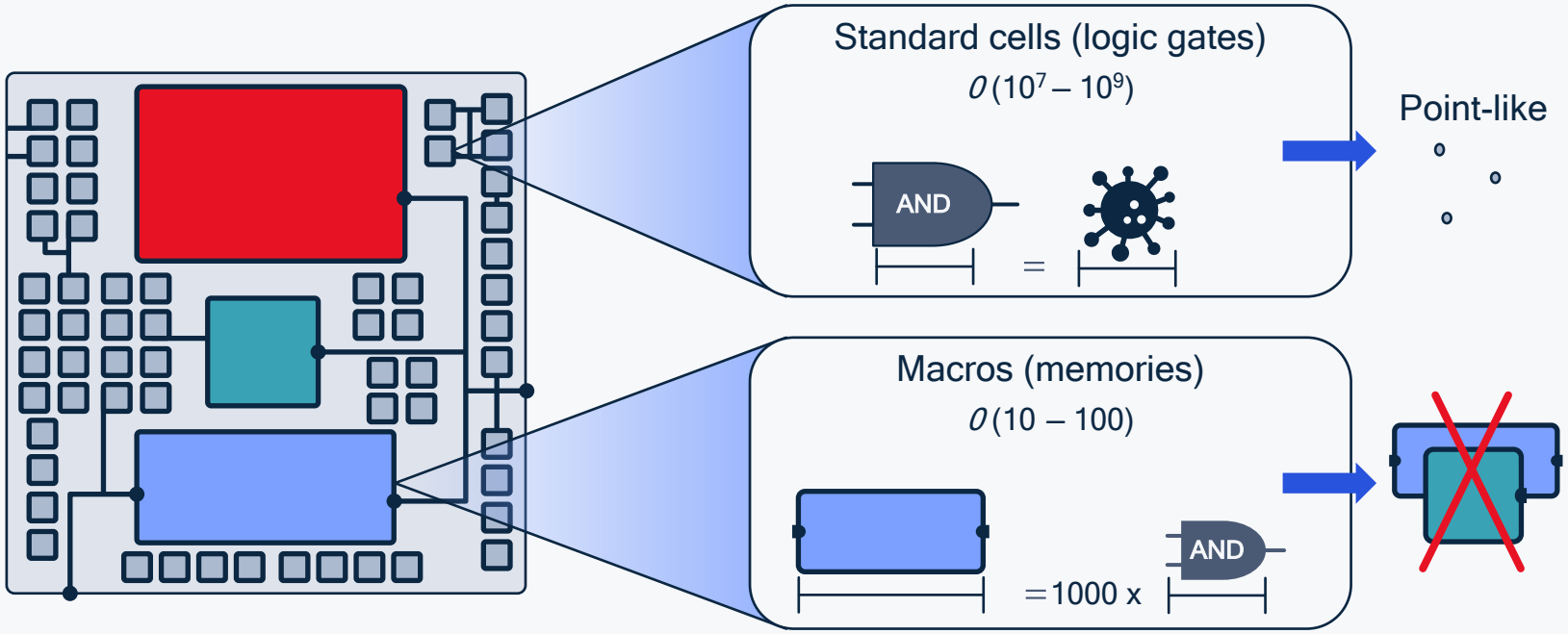
PPA = Power, Performance, Area

Area scaling over time



CO needs to compensate for diminishing technology gains and PVT corner increase within acceptable compute expense

How can we solve the chip optimization problem?



Challenges in chip placement:

1. Placing mixed-size blocks – standard cells and macros (memories)

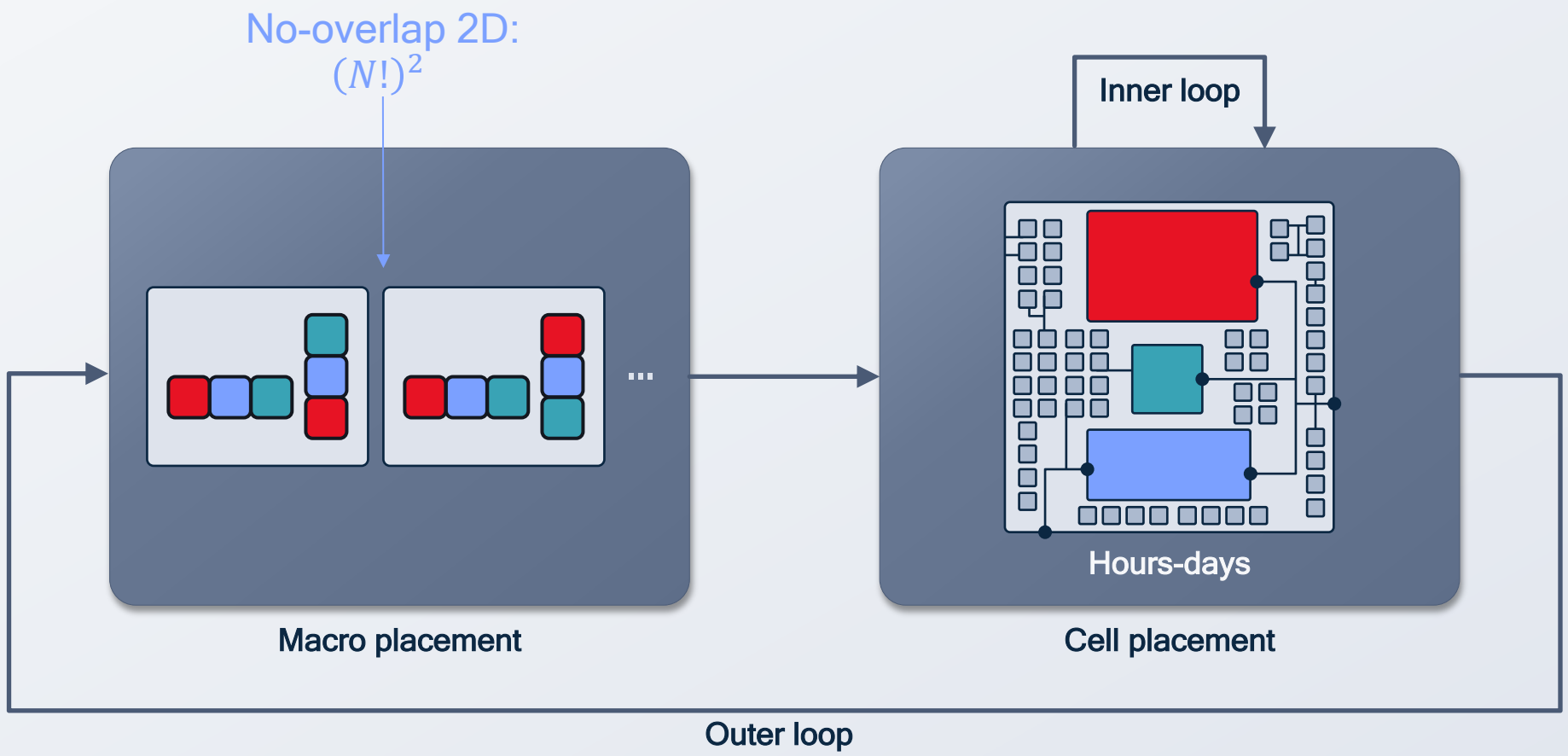
   Minimize power and area while satisfying timing constraints within limited design resources (people and compute servers)

2. Scale with increasing complexity of design (# of blocks) and constraints (e.g., PVT corners)

Chip design is comprised of iterative macro and standard cell placement

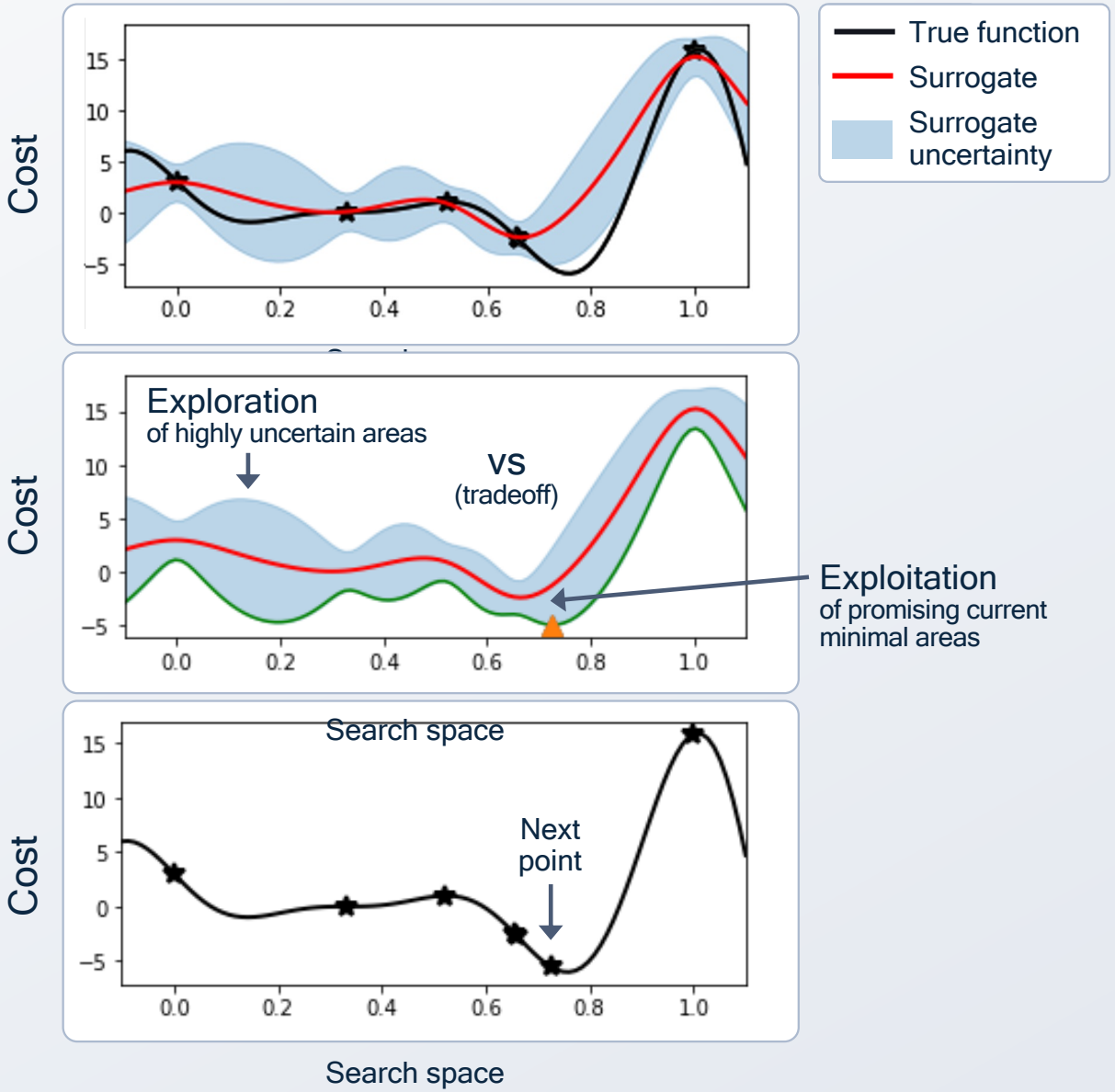
A complex and very computationally intensive part of chip design

Designers manually select a macro placement and use solvers to optimize the standard cell placement (inner loop) and then manually iterate (outer loop)

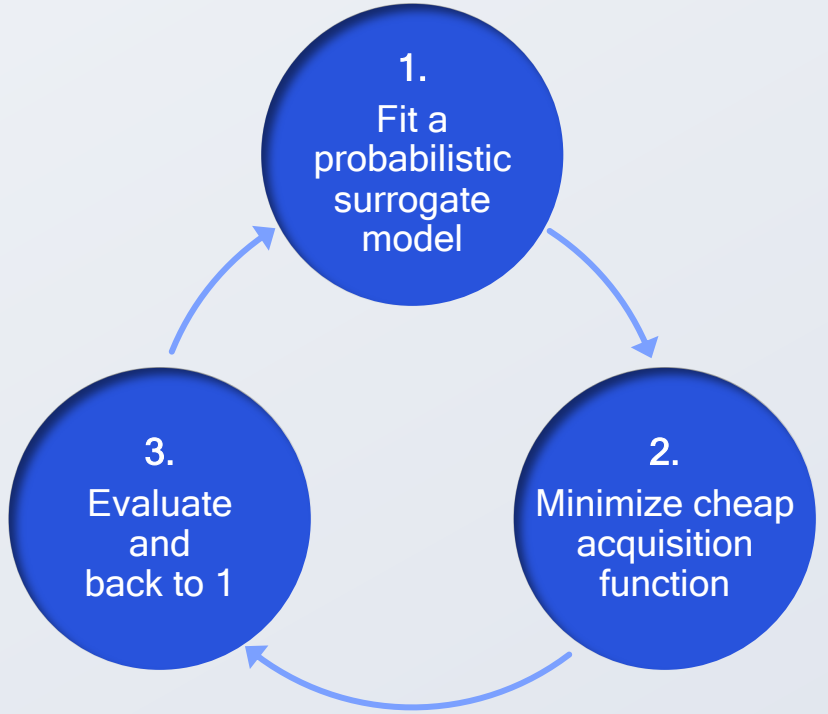


Each iteration can take up to several weeks for state-of-the-art designs and technologies

Bayesian optimization efficiently solves problems iteratively



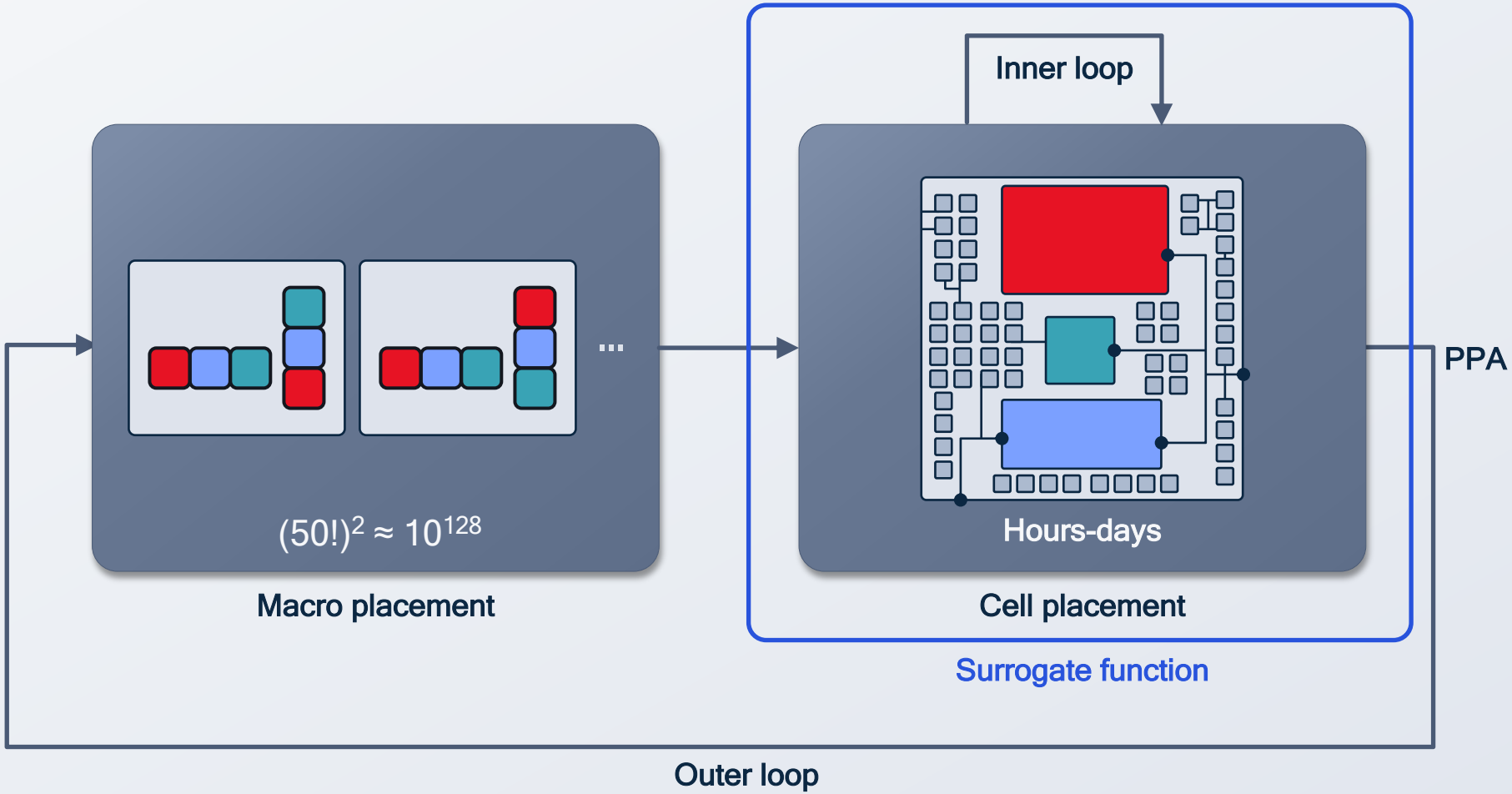
Goal
Find minimum of an expensive function



How to apply to macro placement?

Bayesian optimization for macro layout

Inner loop optimization incorporated into surrogate function

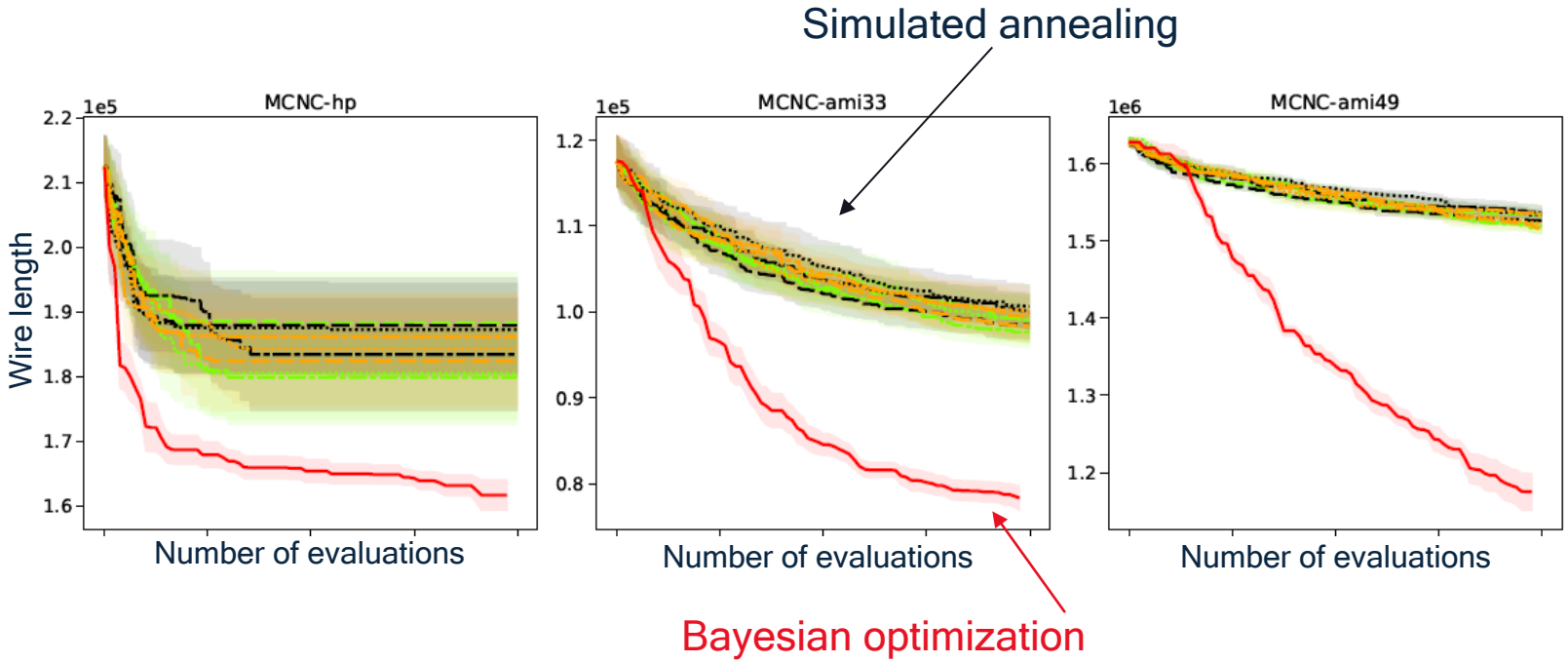


Bayesian optimization learns a surrogate function, which maps each macro placement to a PPA quality metric, and uses it to narrow down the search over the large macro placement space

"Bayesian Optimization for Macro Placement", ICML 2022

Bayesian optimization can converge faster

with better design metrics compared to conventional simulated annealing heuristics for an unconstrained version of the problem



Results are for public MCNC benchmark for layouts (without standard cells)
Three different chip designs: hp, ami33, ami49

Optimization objective is to minimize HPWL (wire length)
This is a simpler objective than the Inner Loop PPA

Further research aimed at generalizing this technique for production designs, across all PPA metrics and with the inclusion of design constraints

"Bayesian Optimization for Macro Placement", ICML 2022

Algorithmic optimization based on analytic solutions needs human guidance

Slow, costly, but accurate for sign-off and automation

Data-driven AI optimization can aid designers with fast evaluations and guide algorithmic optimization with optimal inputs

Fast, cheap, and accurate for optimization

AI compilers can be optimized with AI

AI can improve core components of compilers, including sequencing, scheduling, tiling, and placement

“Neural topological ordering for computation graphs”, NeurIPS 2022

Qualcomm AI Stack

Tools:

Qualcomm AI Model Studio

- AIMET
- AIMET Model Zoo
- NAS
- Model analyzers

Infrastructure:

Prometheus, kubernetes, docker

AI Frameworks, Runtimes

AI Frameworks

TensorFlow, PyTorch, ONNX

AI Runtimes

Qualcomm® Neural Processing SDK, ONNX RUNTIME, TF Lite Micro, Direct ML, TF Lite

Qualcomm® AI Engine Direct

Developer Libraries and Services

Math Libraries, Compilers, Virtual platforms

Profilers & Debuggers, Programming Languages, Core Libraries

System Software

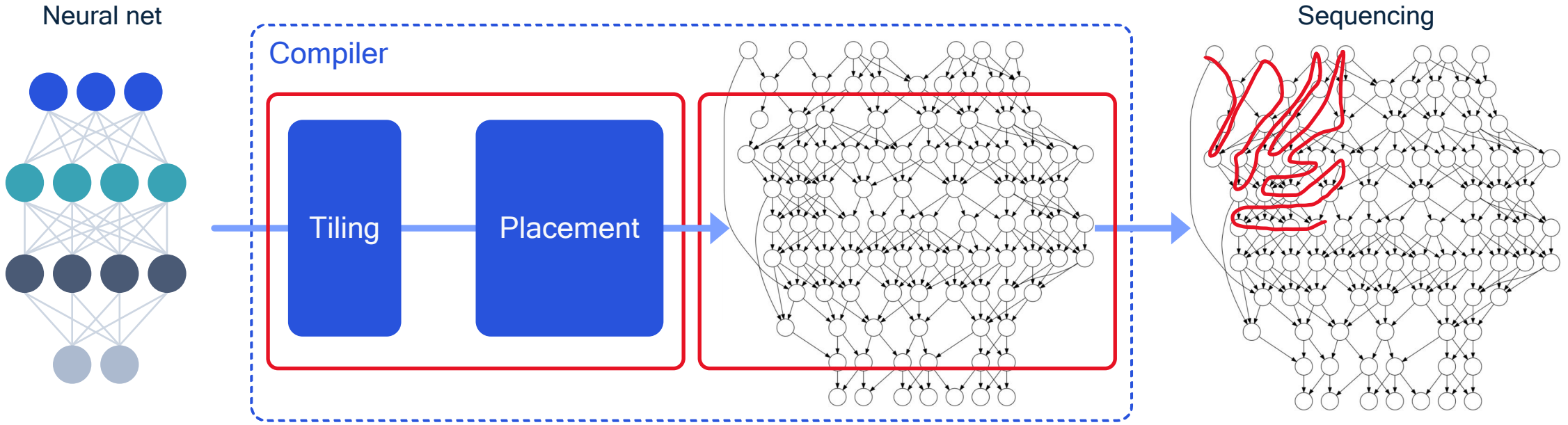
System Interface, SoC, accelerator drivers, Emulation Support

OS

android, Zephyr Project, ubuntu, CentOS, QNX

Platforms

Smartphones, XR, ACPC, IoT, Robotics, Auto, Cloud



Tiling and placement
Splits net blocks into efficient code Ops and places them on multiple compute devices

Sequencing
Determines the best compute ordering of the nodes

Scheduling
Parallelizes across compute engines and sets final timing

Deployment
Puts the resulting generated code onto the target hardware

Our example here will focus on the sequencing problem

The AI compiler converts an input neural net into an efficient program that runs on target hardware

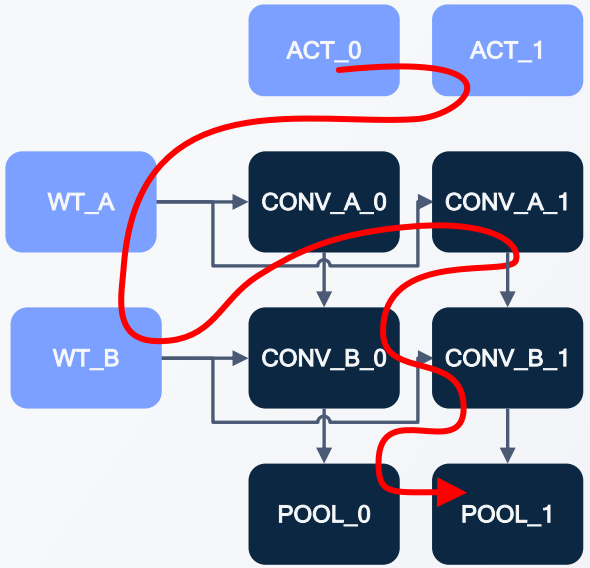
Optimizing for latency, performance, and power

The runtime on the target device is expensive to evaluate

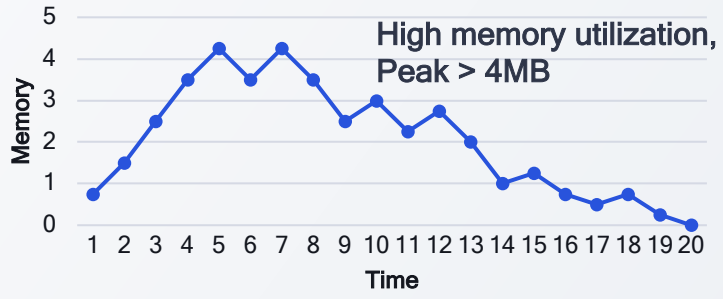
- It can take minutes to set up the compiler with all your decisions and run one computation graph
- Quite a large decision space
- **Goal:** minimizing the runtime of the pipeline over the space of tiling, sequencing, and scheduling choices
- **Metrics:** Double Data Rate (DDR) and Tightly Coupled Memories (TCM)

Sequencing has a big impact on memory usage

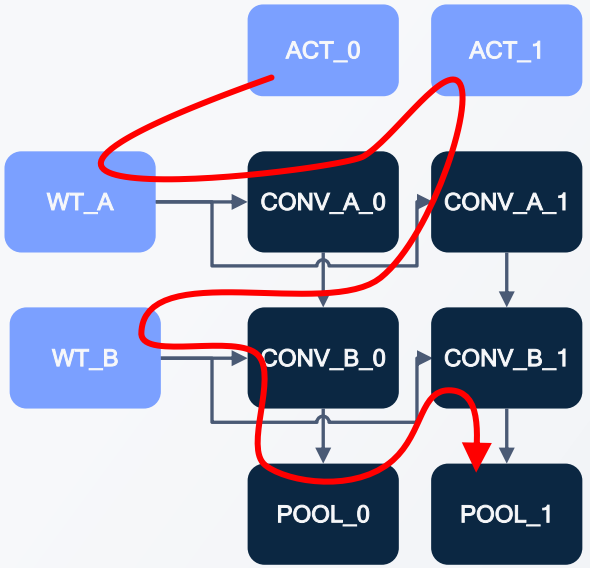
Option #1: Data pre-fetch



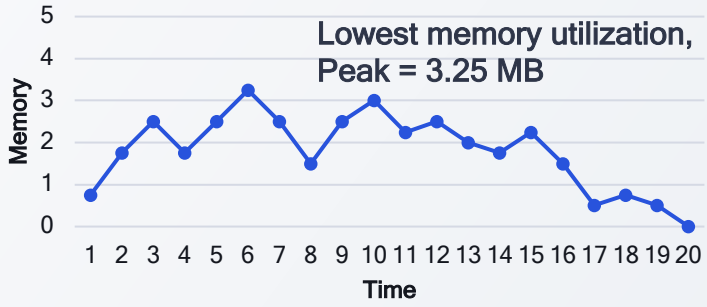
Memory utilization



Option #2: Low memory utilization



Memory utilization

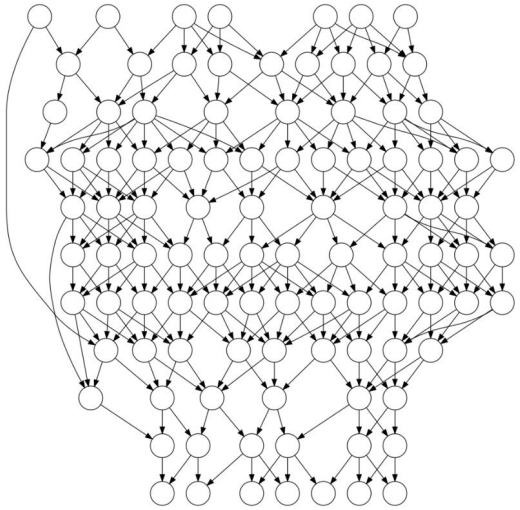


Minimizing memory usage:

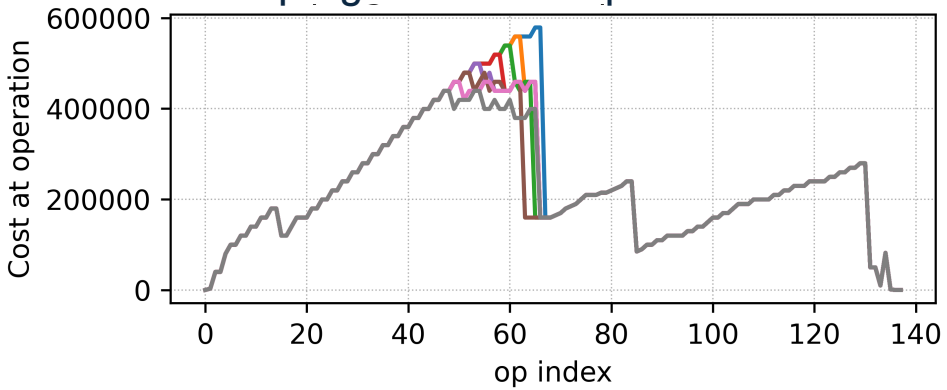
- Maximizes **inferences/sec** by reducing waits for external memory access and by allowing for compute parallelism
- Minimizes **energy/inference** by reducing data movement

Sequencing computation graphs to minimize memory usage

Computation graph (Ops)



Path cost during execution progression of improvements



- # sequences is # node permutations that follow precedence constraints
- Grows exponentially in # nodes in compute graph

Input

A computation graph for a neural net (up to 10k's of nodes)

Objective

Find a sequence that minimizes the peak memory usage

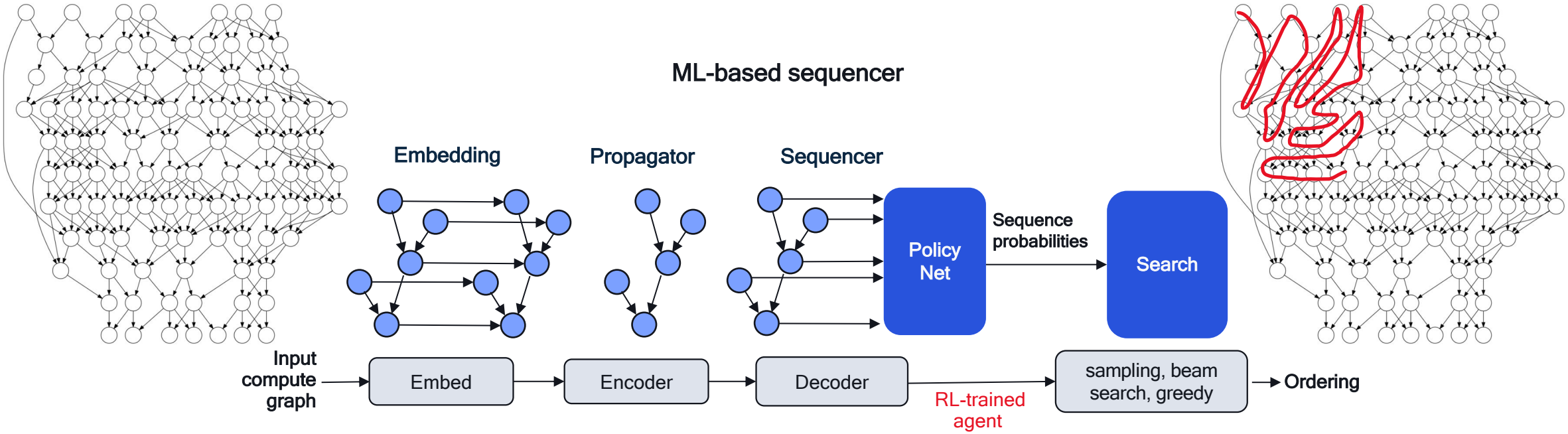
Application

Reduce the inference time of AI models on chips by minimizing access to external memory

External memory access can be 100x local memory and compute times, and is energy-intensive

"Neural Topological Ordering for Computation Graphs", NeurIPS 2022

End-to-end machine learning (ML) for sequencing



Key components

Initial embeddings
Use node properties based on graph structure as initial node embeddings

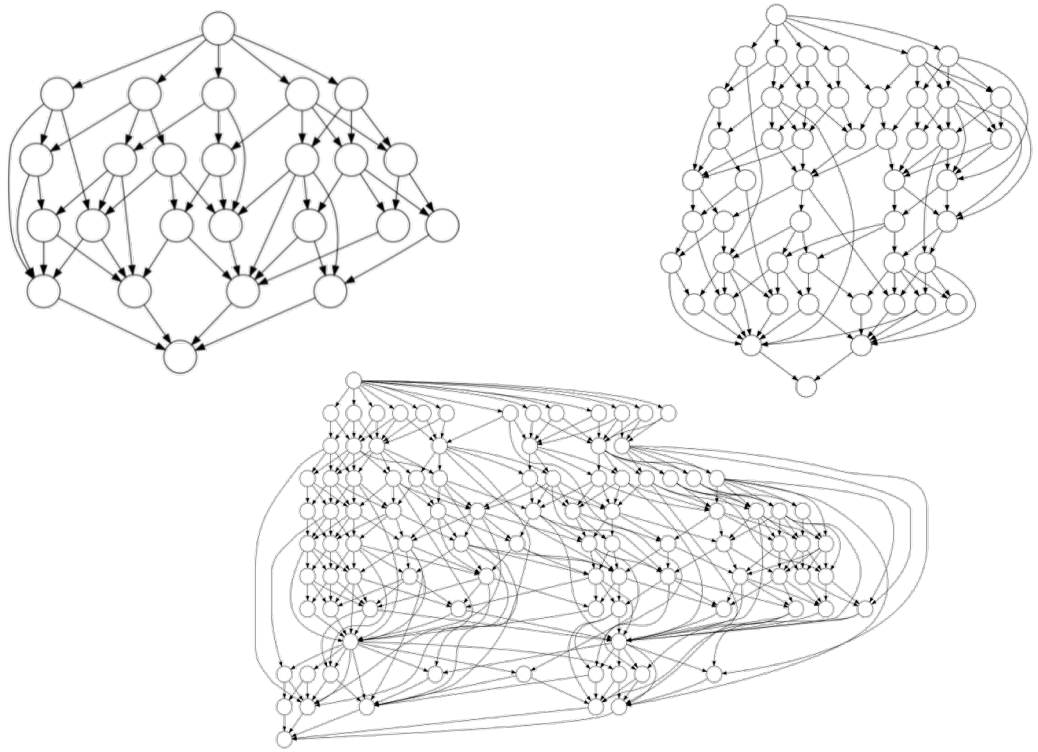
Encoder
Use custom graph NN to capture the graph topology in embeddings while allowing for arbitrary graphs

Decoder
Generates a distribution in the sequence space $P_\theta(seq|G)$ and autoregressively generates a sequence

Objective
$\min. \mathbb{E}_{seq\sim P_\theta}[Cost(\ seq\ )]$, use RL to train encoder decoder architecture end-to-end

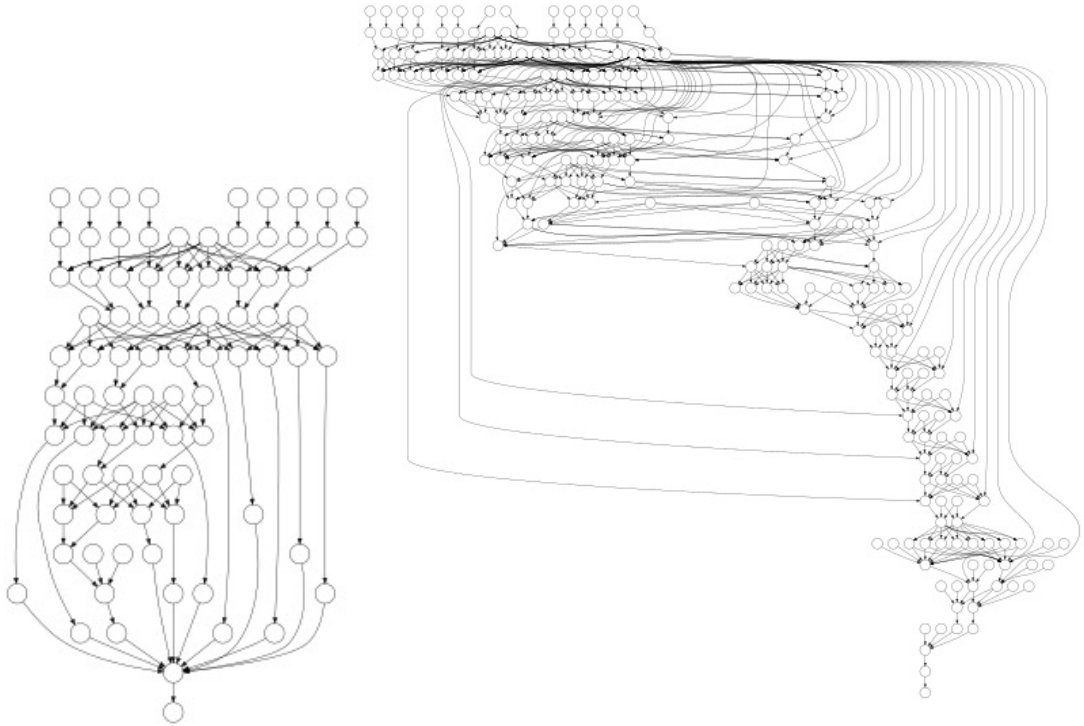
Formulation originally motivated by “Attention, learn to solve routing problems!”, ICLR 2019

Synthetic graphs



We released the algorithm to make these

Real graphs

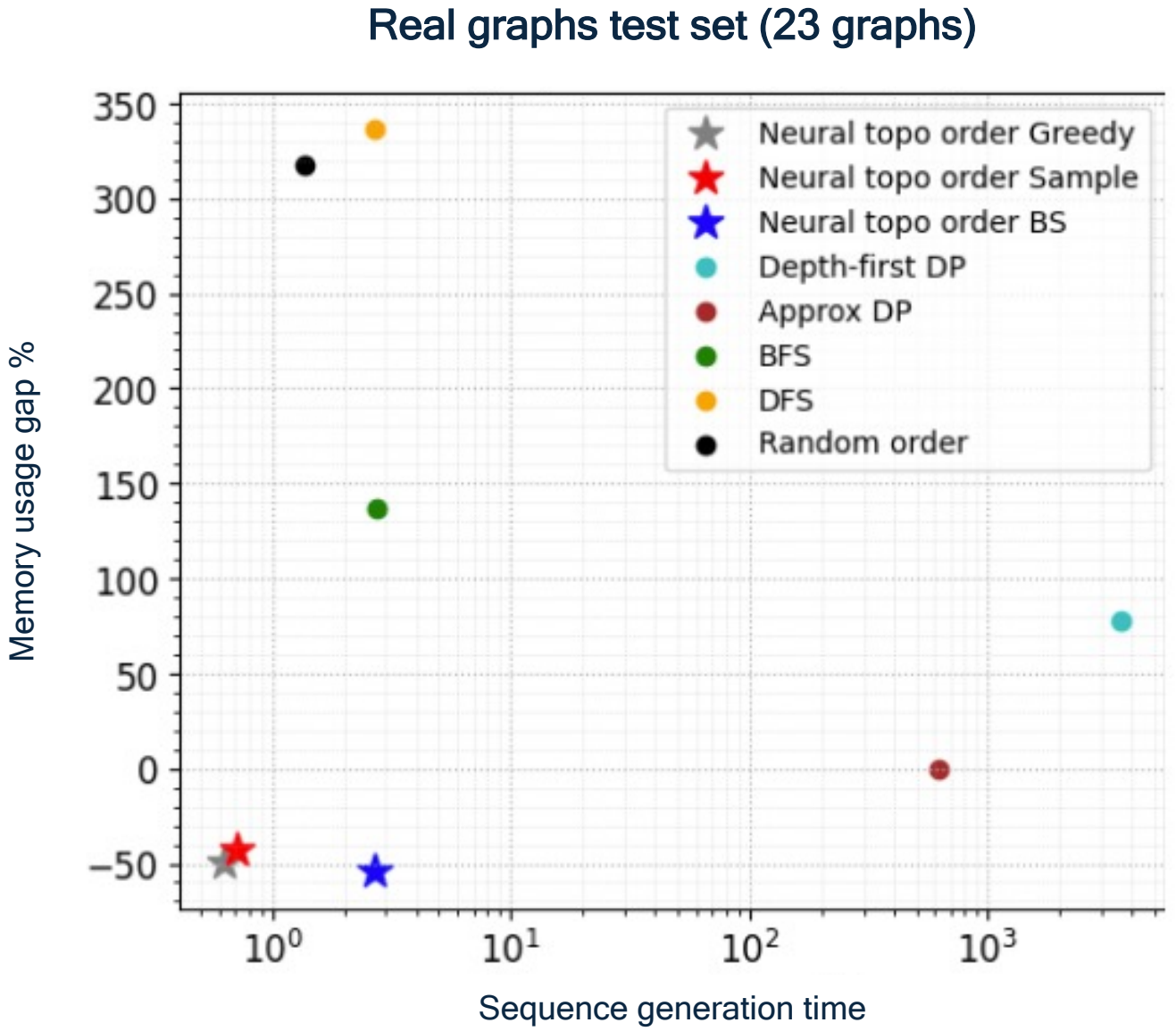


A combination of real and synthetic graphs used for training

We developed a novel approach to generating realistic synthetic graphs since real graph data is scarce

Our model generalizes well and beats baselines comprehensively

- **Dataset of 115** representative graphs
- **Size:** few dozen to 1k nodes
- Our model **performs better** and is **much faster**
- **Results on Snapdragon 8**



BS: Beam Search; DP: Dynamic Programming; BFS: Breadth-First Sequence; DFS: Depth-First Sequence

End-to-end ML for scheduling

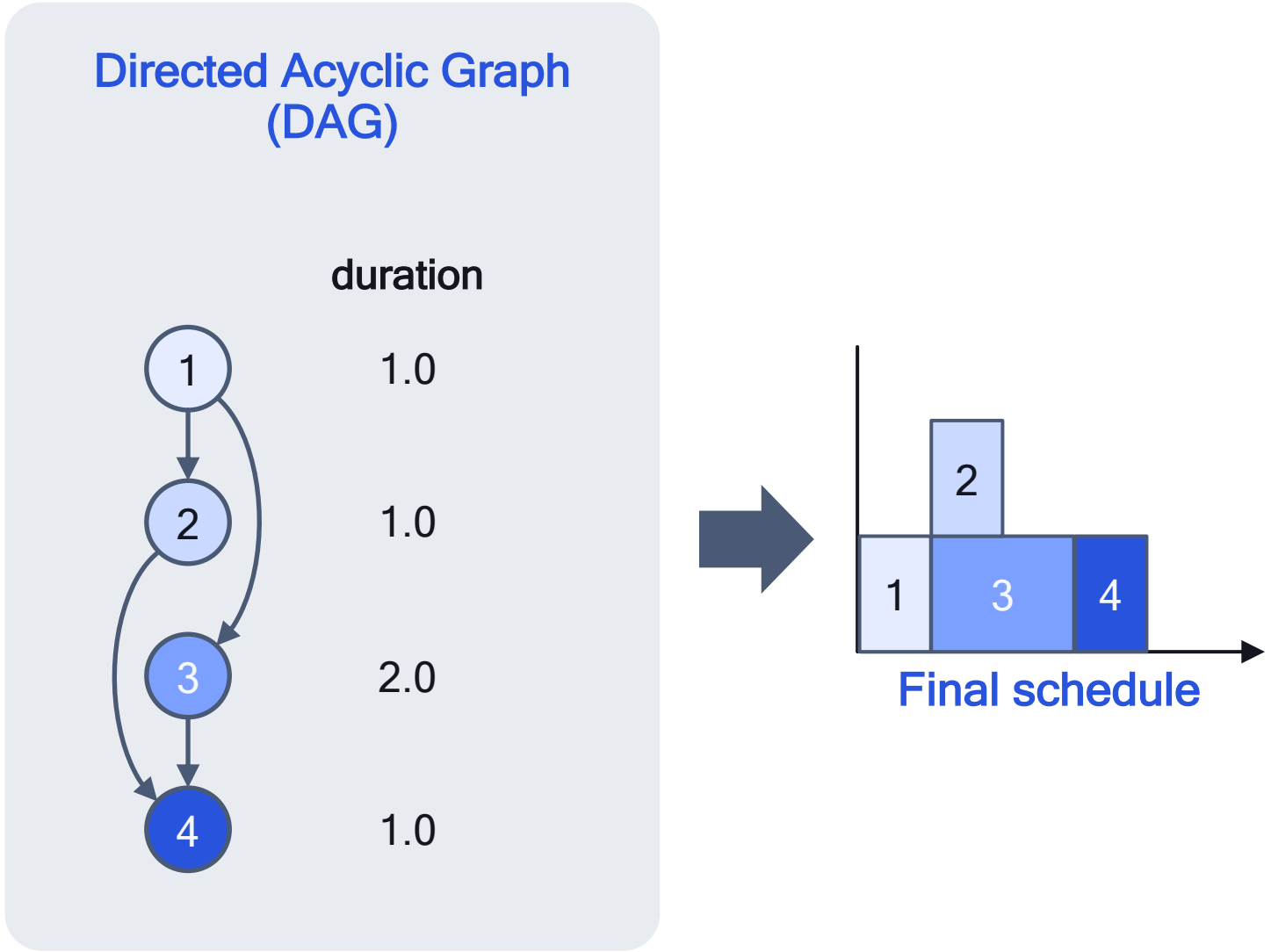
Maximize utilization of parallel hardware while maintaining sequence

Input
A computation graph with
Op durations, set of compute devices

Output
Schedule: Define the start time and device allocation for all nodes

Objective
Find a schedule that minimizes the runtime (latency)

Applications
Reduce the inference time of ML models on chips



End-to-end ML structure similar to previous for sequencing, train to minimize latency

Results for a set of compute graphs of different sizes

Achieves better speedup (inversely proportional to latency, higher is better)

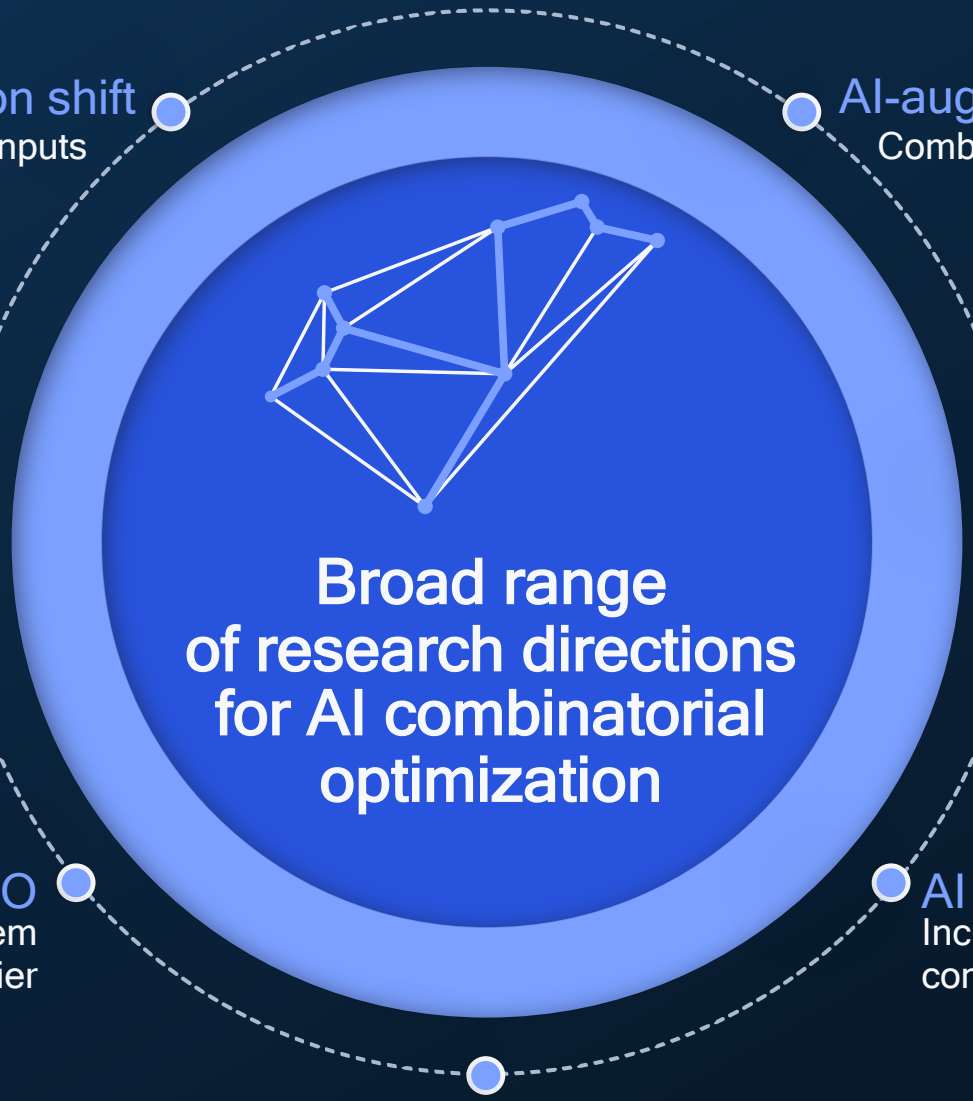
	Algorithm	Node graphs 200 – 500 SpeedUp	500 – 700 SpeedUp	700 – 1000 SpeedUp
Baseline schedulers	CP	3.17	2.80	2.74
	SPT	3.11	2.87	2.66
	MOPNR	3.18	2.82	2.74
Our AI scheduler	S(256)	**3.28**	**3.20**	**2.86**

SpeedUp – higher is better

Our end-to-end ML scheduler achieves state-of-the-art results

Can be optimized for different performance metrics, such as latency

CP: Critical Path; SPT: Shortest Processing Time; MOPNR: Most Operations Remaining



Broad range of research directions for AI combinatorial optimization

Robustify learning to distribution shift
Ensure good results for diverse inputs

AI-augmentation of solvers
Combine best of solvers and AI methods

Scale AI up to larger graphs
Need to efficiently extract global graph structure

Multi-core compiler
Further develop AI approaches targeting many distributed cores

Self-supervised learning for CO
Learn efficient problem representation to solve easier

AI for full end-to-end compiler
Include all aspects including compute primitives and tiling

Dynamical systems and AI
Combine efficiency of dynamic models with AI learning

Qualcomm

Improved combinatorial optimization techniques offer benefits for a variety of use cases across industries

Qualcomm AI Research has achieved state-of-the-art results in combinatorial optimization for chip design and AI compilers

We are enabling combinatorial optimization technology at scale to address challenging problems

Questions

Connect with us

www.qualcomm.com/research/artificial-intelligence

www.qualcomm.com/news/onq

@QCOMResearch

www.youtube.com/c/QualcommResearch

www.slideshare.net/qualcommwirelessevolution

Thank you

Qualcomm

Follow us on:

For more information, visit us at:

qualcomm.com & qualcomm.com/blog