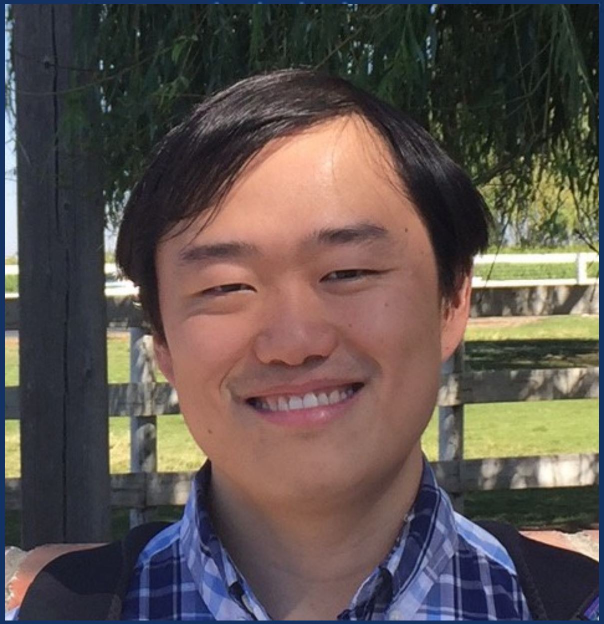
AI-guided Surrogate Modeling for Real-world Optimization

Yuandong Tian
Research Scientist and Manager

Meta AI (FAIR)



facebook Artificial Intelligence

Reinforcement Learning

Go

Chess

Shogi

Poker

DoTA 2

StarCraft II

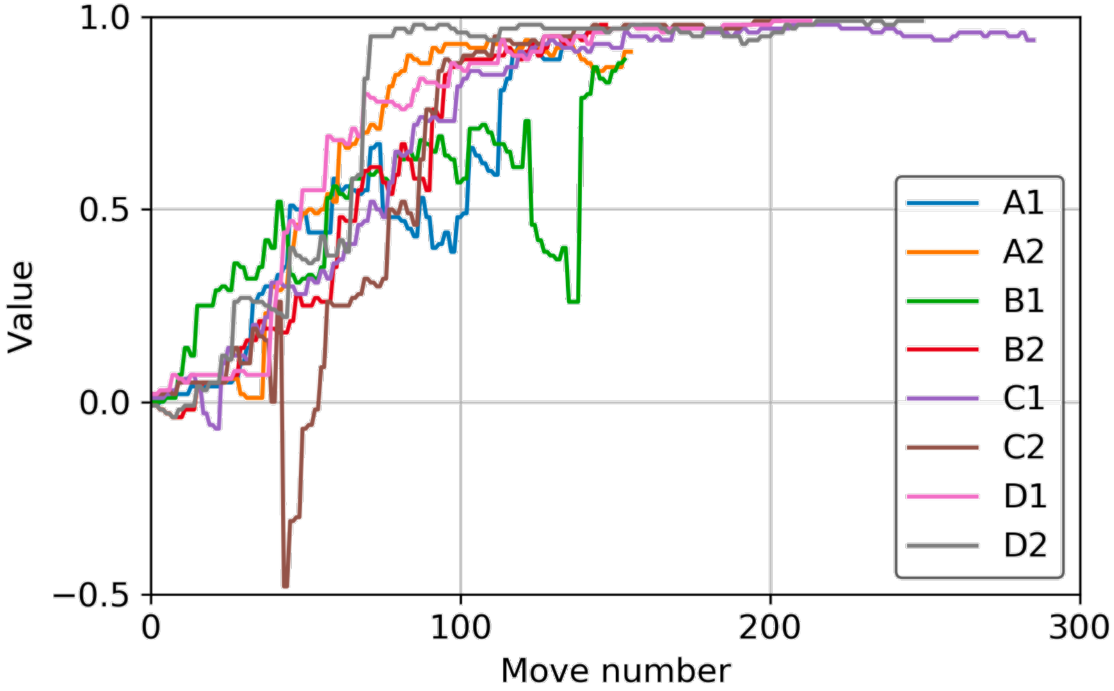
Big Success in Games

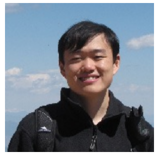
ELF OpenGo

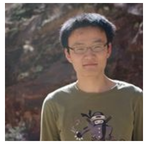
Vs top professional players

Name (rank)	ELO (world rank)	Result
Kim Ji-seok	3590 (#3)	5-0
Shin Jin-seo	3570 (#5)	5-0
Park Yeonghun	3481 (#23)	5-0
Choi Cheolhan	3466 (#30)	5-0

Single GPU, 80k rollouts, 50 seconds
Offer unlimited thinking time for the players









[ELF OpenGo: An Analysis and Open Reimplementation of AlphaZero, Y. Tian et al, ICML 2019]

What’s Beyond Games?

Chip Design (Google)

Several weeks with **human experts** in the loop

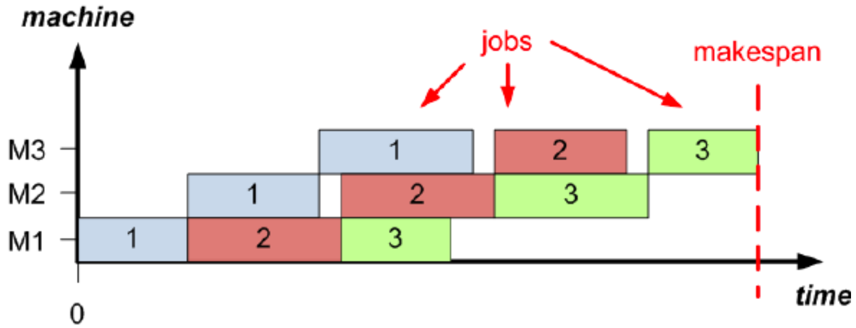
→

Fully automatic design in 6 hours

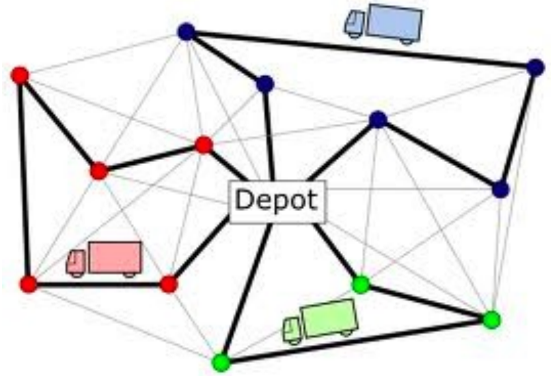
[A. Mirhoseini, A. Goldie, et al, *A graph placement methodology for fast chip design,* Nature’21]

Optimization Problems

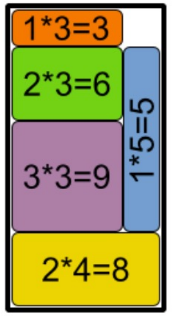
Travel Salesman Problem



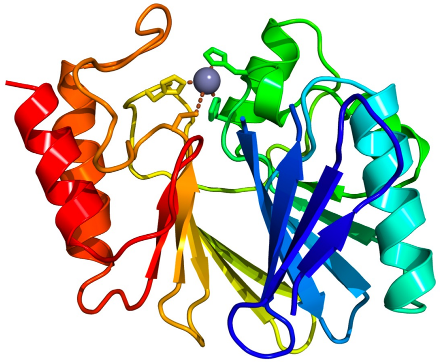
Job Scheduling



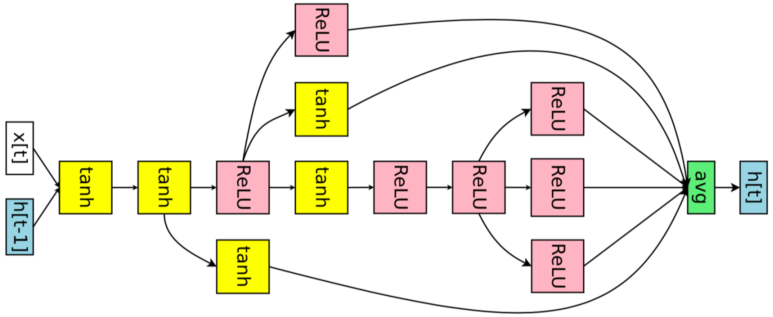
Vehicle Routing



Bin Packing



Protein Folding



Model-Search

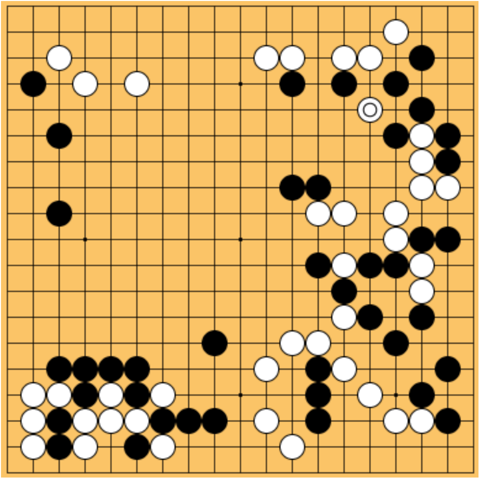
$$x^* = \arg\max_{x \in \Omega} f(x)$$

Wait...What?

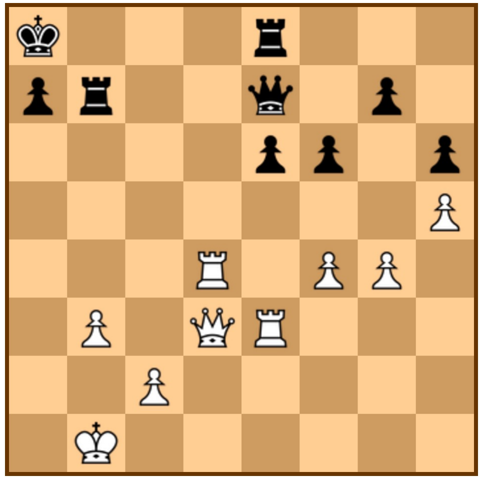
- Many problems are NP-hard problems.
  - No good algorithm unless $P = NP$
- These guarantees are worst-case ones.
  - To prove a lower-bound, construct an adversarial example to fail the algorithm
- For specific distribution, there might be better heuristics.
  - Human heuristics are good but may not be suitable for everything

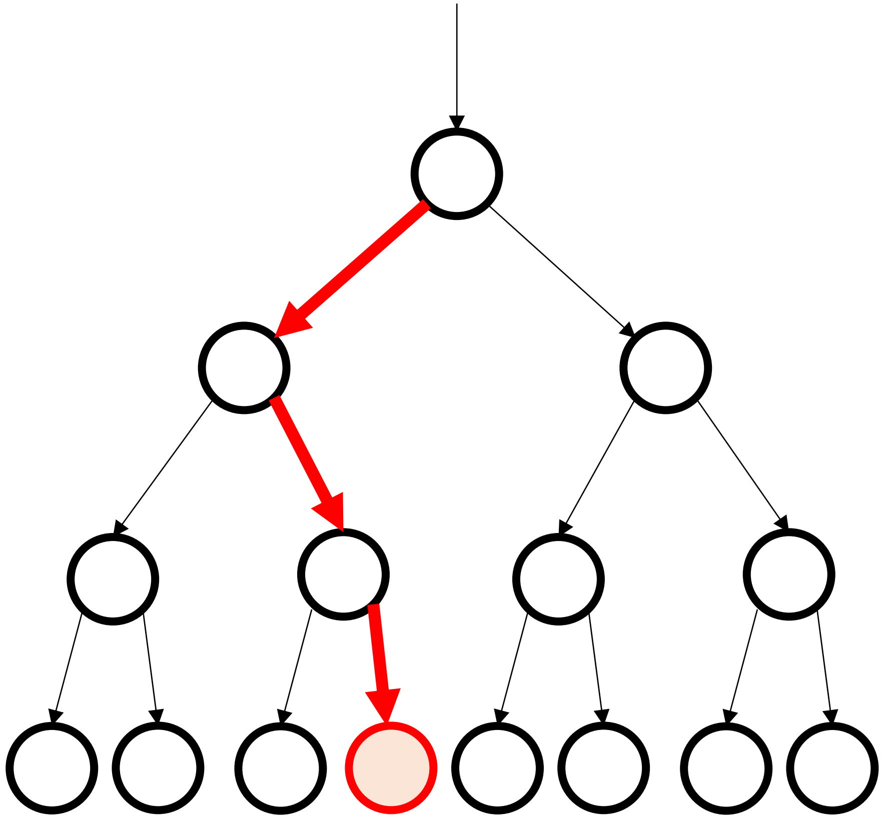
Efficient Search for Games

Go



Chess

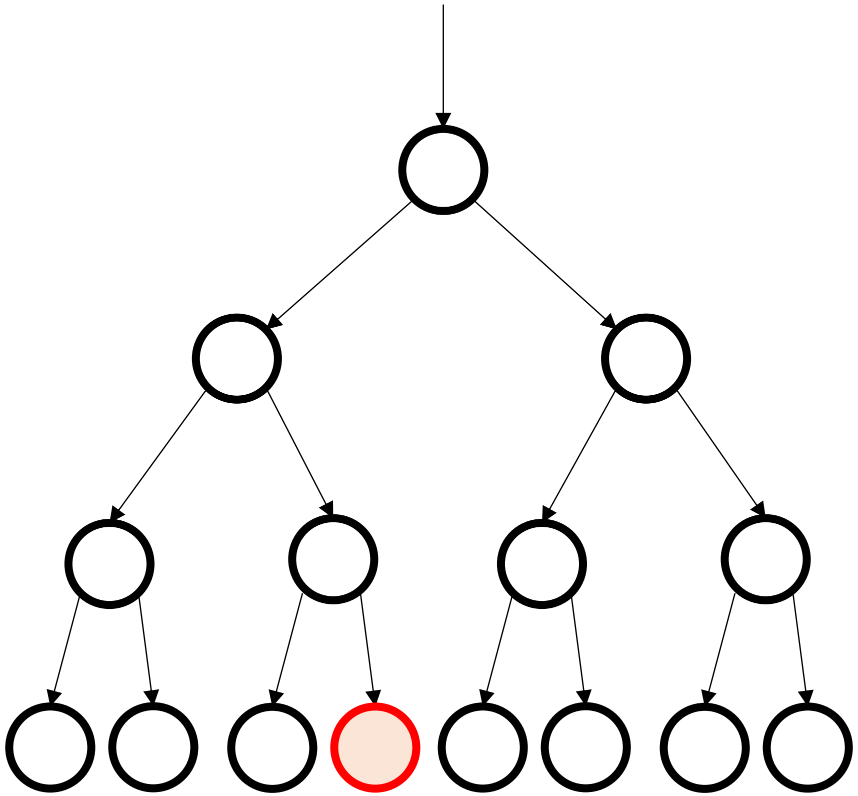




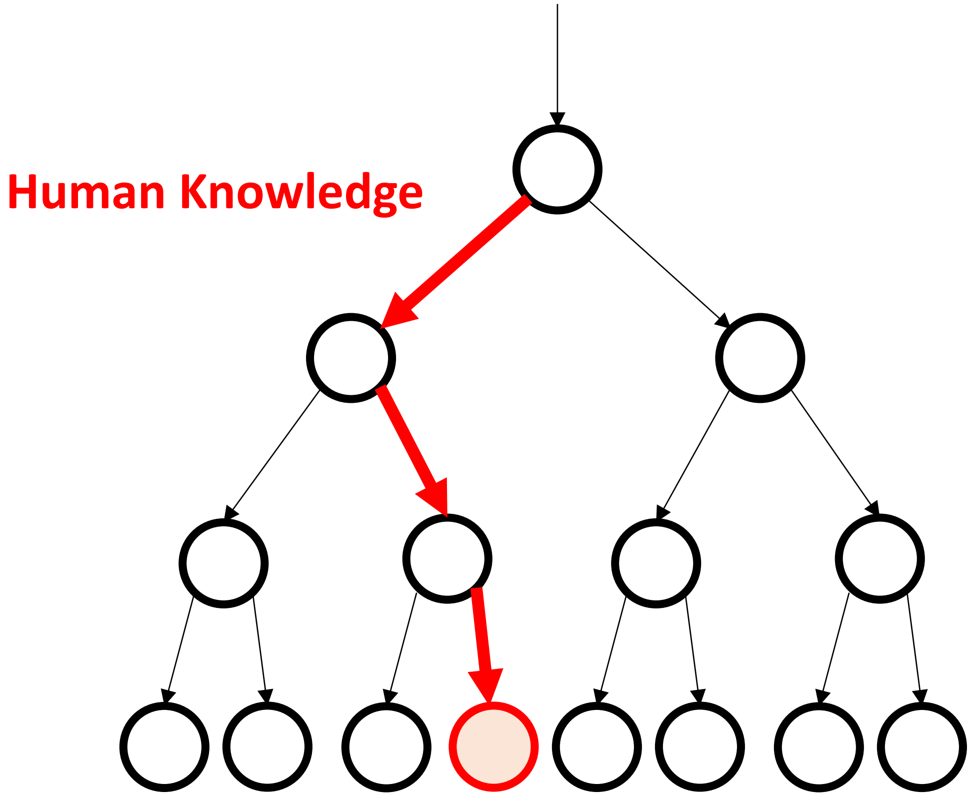
~~Human Knowledge~~

Machine learned models

Efficient Search for Optimization



Exhaustive search to get **a good solution**

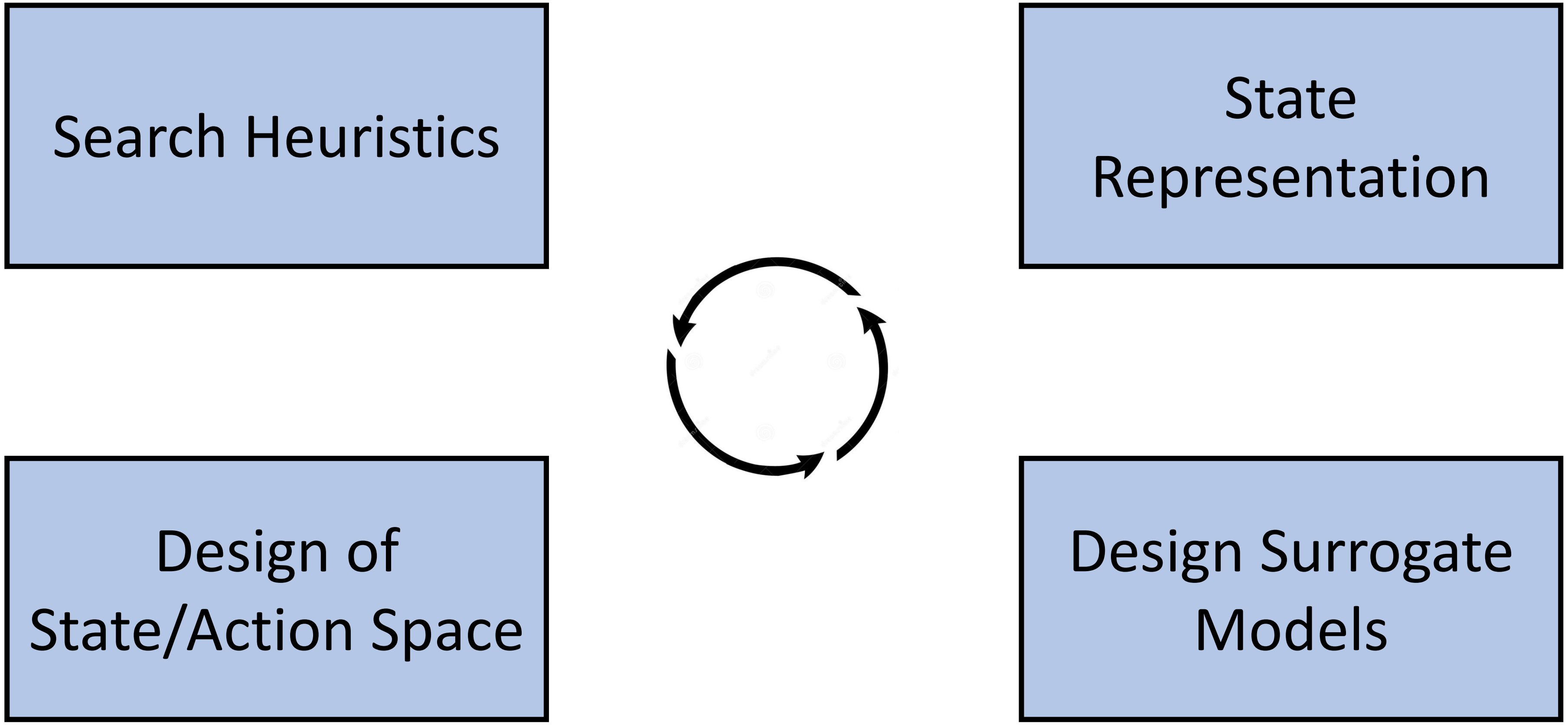


More Efficient Search for Optimization

Can we use Machine Learning?

Exhaustive search to get **a good solution**

Components of Search



Components of Search

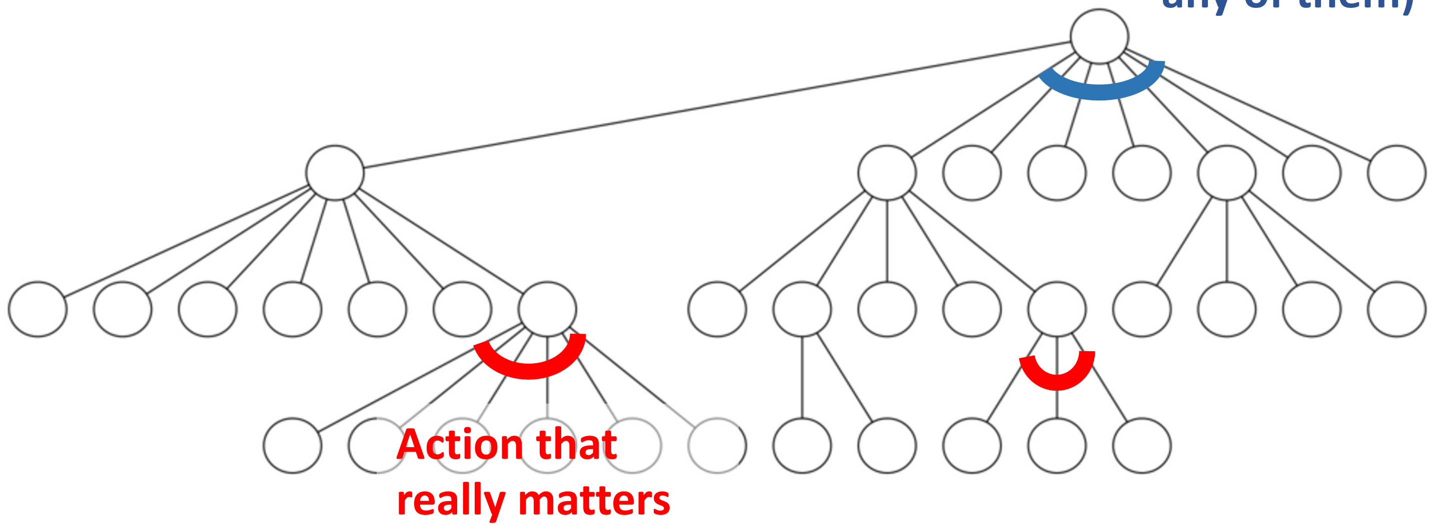
Search Heuristics

Initial solution → Improved solution1 → Improved solution2 → …

[X. Chen and Y. Tian, **Learning to Perform Local Rewriting for Combinatorial Optimization**, *NeurIPS'19]*
[H. Shi et al, **Deep Symbolic Superoptimization Without Human Knowledge,** *ICLR'20]*
[H. Zhu et al, **Network planning with deep reinforcement learning,** *SIGCOMM'21]*
[T. Huang et al, **Local Branching Relaxation Heuristics for Integer Linear Programs,** *CPAIOR'23*]
[T. Huang et al, **Searching Large Neighborhoods for Integer Linear Programs with Contrastive Learning**, arXiv'23]

Components of Search

Design of State/Action Space

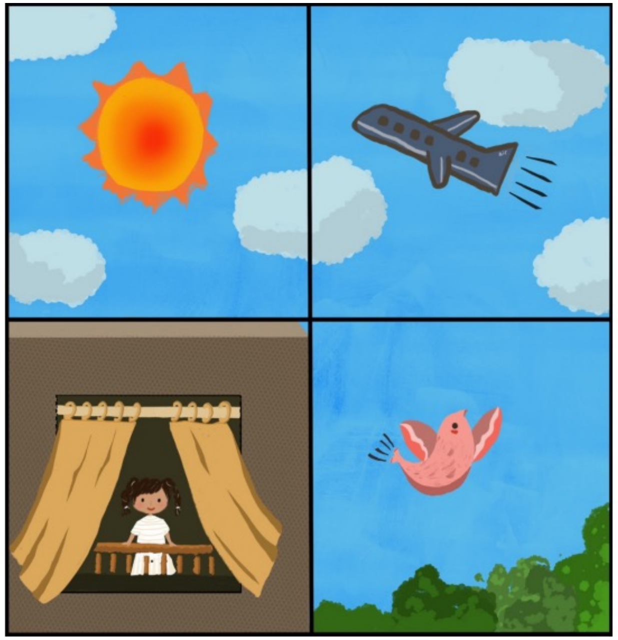


If useful actions only happen after **50** binary moves, then we will waste our efforts in this $\mathbf{2^{50}}$ possibilities.

[L. Wang, et al, **Learning Search Space Partition for Black-box Optimization using MCTS,** *NeurIPS'20]*
[L. Wang, et al, **Sample-Efficient Neural Architecture Search by Learning Action Space,** *T-PAMI'21]*
[K. Yang, et al, **Learning Space Partitions for Path Planning,** *NeurIPS'21*]
[Y. Zhao, et al, **Multi-objective Optimization by Learning Space Partitions,** *ICLR'22]*
[Y. Liang, et al, **Learning Compiler Pass Orders using Coreset and Normalized Value Prediction**, *arXiv'23]*

Components of Search

State Representation



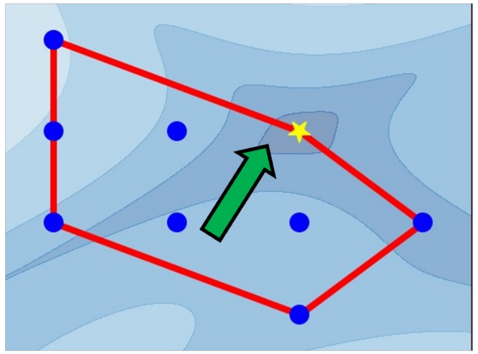
GOAL: Letting in as much sunlight as possible

[X. Chen et al, **Latent Execution for Neural Program Synthesis Beyond Domain-Specific Languages,** *NeurIPS'21]*
[T. Wang et al, **Denoised MDPs: Learning World Models Better Than the World Itself,** *ICML'22*]
[Z. Jiang et al, **Efficient Planning in a Compact Latent Action Space**, *ICLR'23]*

Components of Search

Design Surrogate Models

[Y. Zhao, et al, **Few-shot neural architecture search,** *ICML'21]*
[D. Zha, et al, **DreamShard: Generalizable Embedding Table Placement for Recommender Systems,** *NeurIPS'22]*
[A. Ferber, et al, **SurCo: Learning Linear Surrogates For Combinatorial Nonlinear Optimization Problems,** *arXiv'22]*
[A. Cohen, et al, **Modeling Scattering Coefficients using Self-Attentive Complex Polynomials with Image-based Representation,** *arXiv'23]*



SurCo: Learning Linear Surrogates for Combinatorial Nonlinear Optimization

Aaron Ferber[1], Taoan Huang[1], Daochen Zha[2], Martin Schubert[3], Benoit Steiner[4], Bistra Dilkina[1], Yuandong Tian[4]

[1]University of Southern California, [2]Rice University, [3]Reality Lab Display, [4]Meta AI (FAIR)

USC

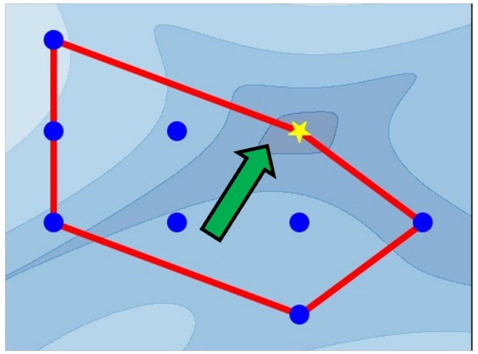
RICE UNIVERSITY School of Engineering Department of Computer Science

Meta AI

facebook Artificial Intelligence

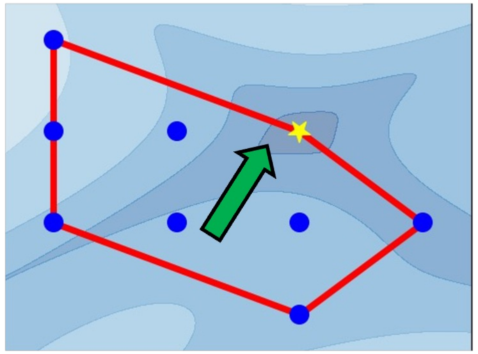
https://arxiv.org/abs/2210.12547

Optimizing Nonlinear Functions over Combinatorial Regions



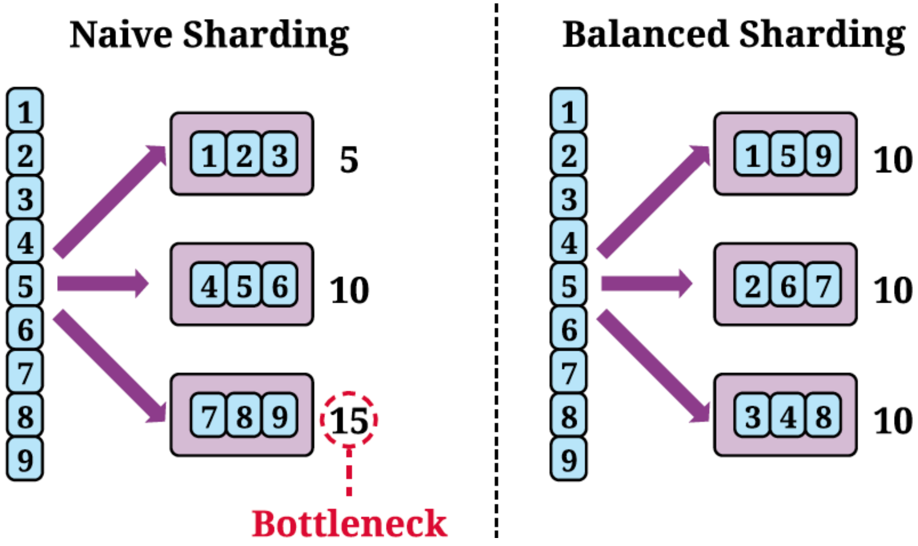
- Nonlinear + differentiable objective
- Combinatorial feasible region
- Real-world domains:
  - Computer system planning
  - Designing photonic devices
  - Throughput optimization
  - Antenna design
  - Energy grid

Example: Embedding Table Placement



Given:

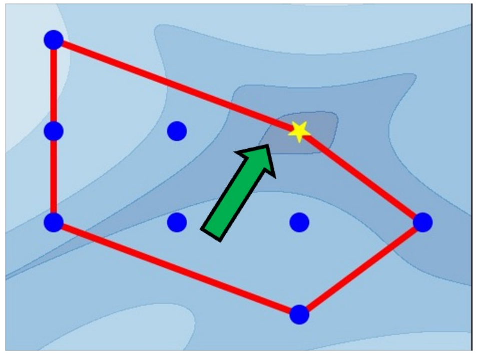
- k tables
- n identical devices
- Table i has memory requirement m_i
- Device j has memory capacity M_j



Find

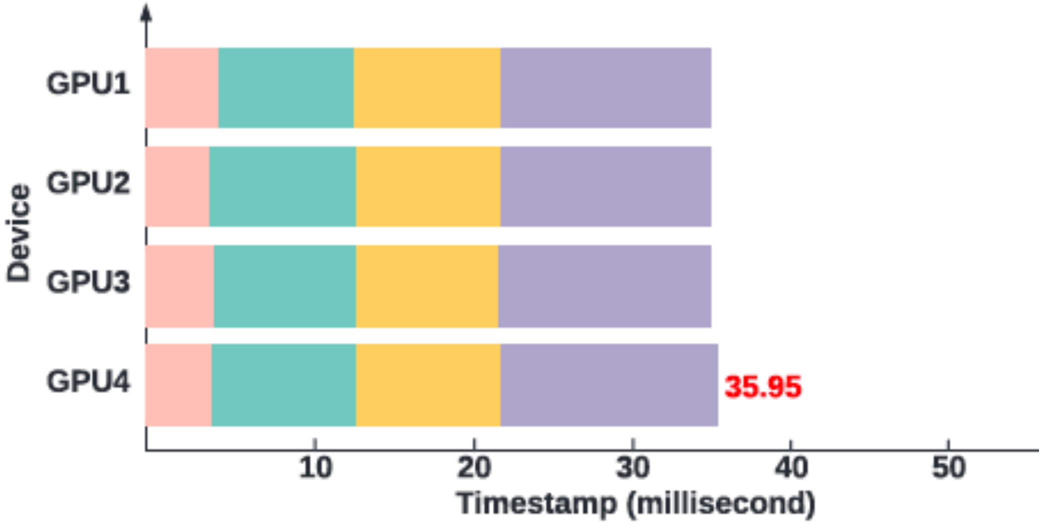
- Allocation of tables to devices observing device memory limits
- Minimize latency which is estimated by a neural network (capturing nonlinear interactions)

Example: Embedding Table Placement



Given:

- k tables
- n identical devices
- Table i has memory requirement m_i
- Device j has memory capacity M_j

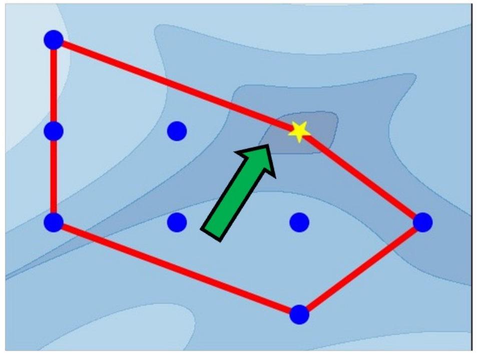


Formulation

$$\mathrm{Min}_x\ \boldsymbol{L}(\{x_{ij}\}) \quad \text{s.t.} \ \textstyle\sum_i x_{ij} m_i \le M_j, \quad \sum_j x_{ij} = 1, \quad x_{ij} \in \{0,1\}$$

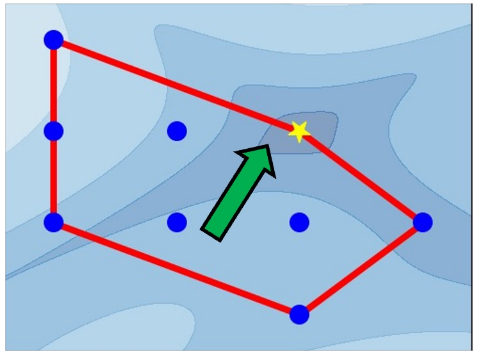
$\boldsymbol{L}$ is nonlinear due to system issues (e.g., batching, communication, etc)

Nonlinear Optimization is Hard



- Specific domains have specialized solvers
- General solvers are often slow (without very careful modeling)
- Genetic algorithms or gradient-based methods may not find feasible solutions

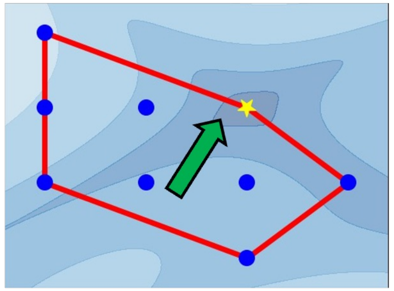
Linear Optimization is Easy(ish)



- MILP solvers (CPLEX, Gurobi, SCIP) easily handle industry-scale problems
- Plus other solvers for linear settings
  - Greedy
  - LP + total unimodularity

Idea: Find a Linear Surrogate

- Learn a MILP objective whose optimal solution x* solves the nonlinear problem

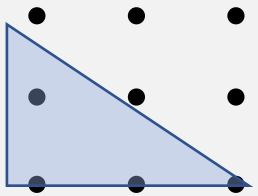


Originally

Nonlinear optimization with combinatorial constraints

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}; \boldsymbol{y})$$

$$\text{s.t } \boldsymbol{x} \in \Omega =$$



combinatorial constraints

Predict surrogate cost $\boldsymbol{c} = \boldsymbol{c}(\boldsymbol{y})$

Now

Surrogate optimization

$$\boldsymbol{x}^*(\boldsymbol{y}) = \operatorname*{argmin}_{\boldsymbol{x}} \boldsymbol{c}(\boldsymbol{y})^{\boldsymbol{T}} \boldsymbol{x}$$

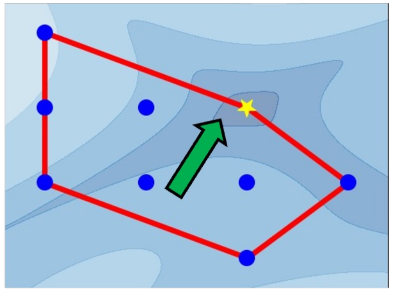
$$\text{s.t } \boldsymbol{x} \in \Omega$$

solved by existing combinatorial solvers

$\boldsymbol{x}^*(\boldsymbol{y})$ optimizes $f(\boldsymbol{x}; \boldsymbol{y})$ as much as possible

Idea: Find a Linear Surrogate

- Learn a MILP objective whose optimal solution x* solves the nonlinear problem

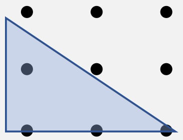


Originally

Nonlinear optimization with combinatorial constraints

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}; \boldsymbol{y})$$

$$\text{s.t } \boldsymbol{x} \in \Omega =$$



combinatorial constraints

Predict surrogate cost $\boldsymbol{c} = \boldsymbol{c}(\boldsymbol{y})$

Now

Surrogate optimization

$$\boldsymbol{x}^*(\boldsymbol{y}) = \operatorname*{argmin}_{\boldsymbol{x}} \boldsymbol{c}(\boldsymbol{y})^{\boldsymbol{T}} \boldsymbol{x}$$

$$\text{s.t } \boldsymbol{x} \in \Omega$$

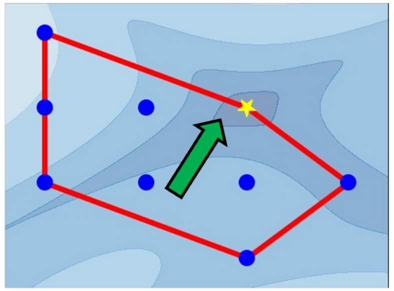
solved by existing combinatorial solvers

$\boldsymbol{x}^*(\boldsymbol{y})$ optimizes $f(\boldsymbol{x}; \boldsymbol{y})$ as much as possible

Challenge: how to find the right objective?

Idea: Find a Linear Surrogate

- Learn a MILP objective whose optimal solution x* solves the nonlinear problem

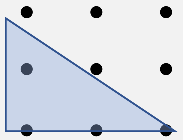


Originally

Nonlinear optimization with combinatorial constraints

$$\min_{\boldsymbol{x}} f(\boldsymbol{x};\boldsymbol{y})$$

$$\text{s.t } \boldsymbol{x} \in \Omega =$$



combinatorial constraints

Predict surrogate cost $\boldsymbol{c} = \boldsymbol{c}(\boldsymbol{y})$

Now

Surrogate optimization

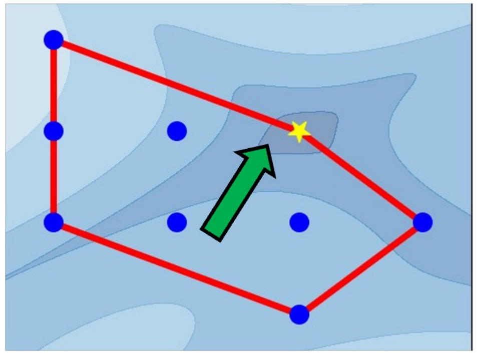
$$\boldsymbol{x}^*(\boldsymbol{y}) = \operatorname*{argmin}_{\boldsymbol{x}} \boldsymbol{c}(\boldsymbol{y})^{\boldsymbol{T}} \boldsymbol{x}$$

$$\text{s.t } \boldsymbol{x} \in \Omega$$

solved by existing combinatorial solvers

$\boldsymbol{x}^*(\boldsymbol{y})$ optimizes $f(\boldsymbol{x};\boldsymbol{y})$ as much as possible

Proposal: gradient-based optimization

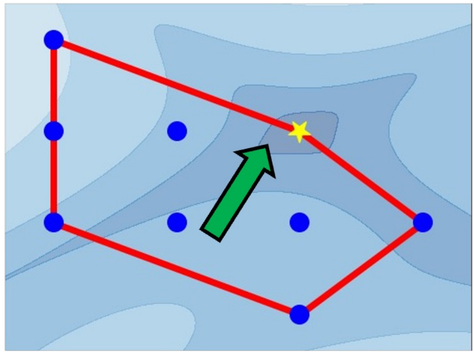


Proposal: surrogate learning

- Use surrogate MILP to solve original problem
- Find linear coefficients c such that $\underset{x\in\Omega}{\operatorname{argmin}}\, f(x) \approx \underset{x\in\Omega}{\operatorname{argmin}}\, c^T x$

Surrogate Coefficients c → **Solver** $x^*(c) = \underset{x\in\Omega}{\operatorname{argmin}}\, c^T x$ → Solution $x^*(c)$ → **Objective** $f(x^*)$ → Loss $f(x^*)$

SurCo-zero: gradient-based optimization



- **Iterative** solver based on linear surrogate guided by **gradient updates**
- Update linear coefficients c such that $x^*(c)$ improves objective $f(x^*(c))$

Surrogate Coefficients c

Solver
$$x^*(c) = \underset{x \in \Omega}{\operatorname{argmin}}\, c^T x$$

Solution
$x^*(c)$

Objective
$f(x^*)$

Loss
$f(x^*)$

$\nabla_c x^*(c)$

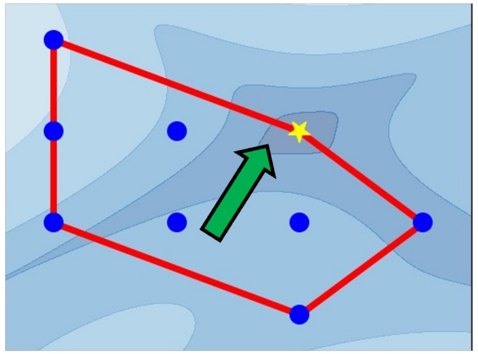
Recent work on differentiable optimization

Differentation of blackbox optimizers
CVXPYLayers
MIPaaL
... more in related work

$\nabla_x f(x)$

Assumed differentiable

SurCo-prior: distributional learning



- One pass solver based on model **learned offline**
- Use neural model based on **problem features** to predict linear coefficients

Problem features

y

θ

Model parameters

Neural Network
$c = NN(y;\theta)$

Surrogate Coefficients

c

Solver
$x^*(c) = \underset{x \in \Omega}{\operatorname{argmin}}\, c^T x$

Solution

$x^*(c)$

Objective
$f(x^*)$

Loss

$f(x^*)$

$\nabla_y NN(y;\theta)$

Standard NN autograd

Pytorch
Tensorflow
JAX etc...

$\nabla_c x^*(c)$

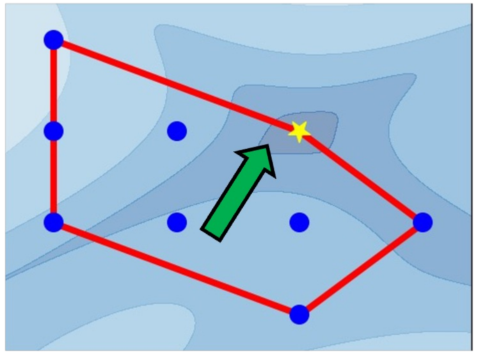
Recent work on differentiable optimization

Differentation of blackbox optimizers
CVXPYLayers
MIPaaL
... more in related work

$\nabla_x f(x)$

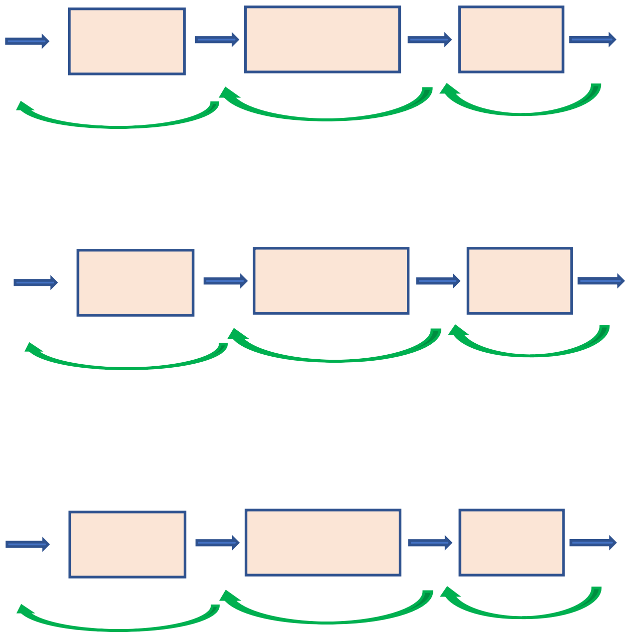
Assumed differentiable

SurCo-prior: distributional learning



- Update neural network parameters from training dataset

$c_i = NN(y_i; \theta)$



Train Model parameters $\boldsymbol{\theta}$

Surrogate Coefficients

$c_{\text{test}} = NN(y_{\text{test}}; \theta)$

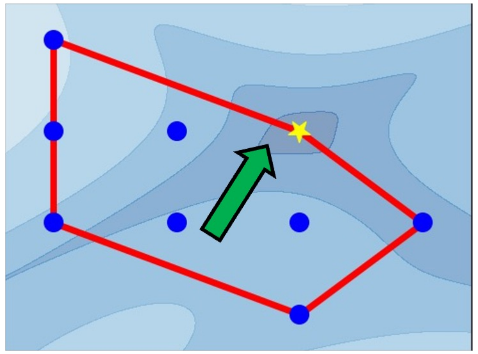
Solver

$x^*(c) = \underset{x \in \Omega}{\operatorname{argmin}}\, c^T x$

Solution

$x^*(c_{\text{test}})$

SurCo-hybrid: fine-tuning from trained model

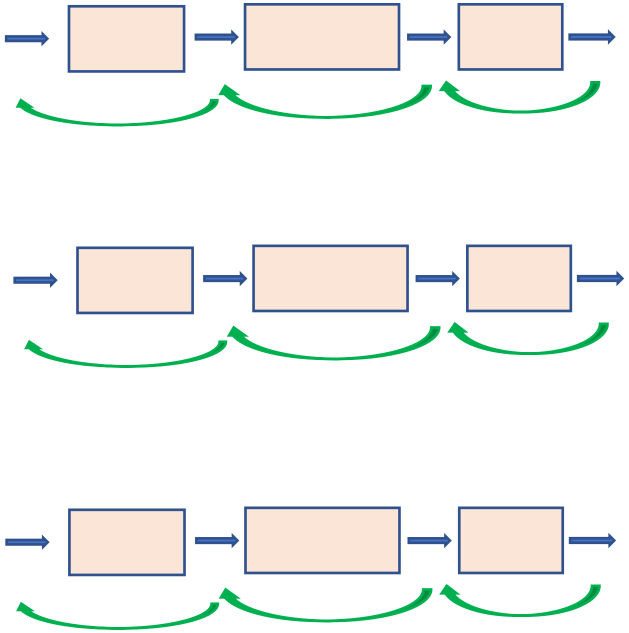


Update neural network parameters from training dataset

Fine-tune surrogate on-the-fly

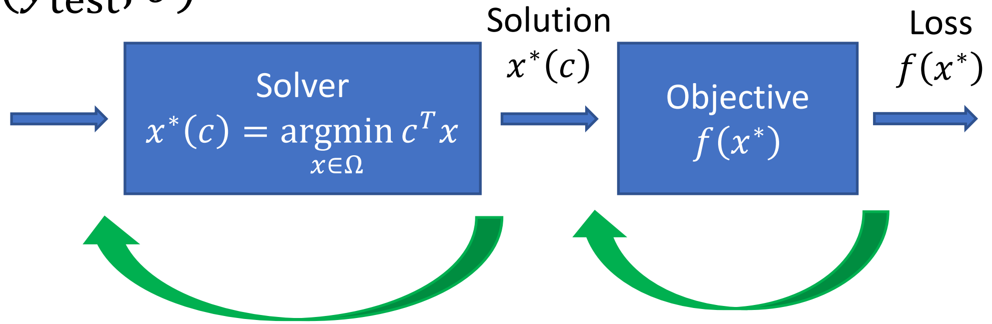
$$c_i = NN(y_i; \theta)$$

Train Model parameters $\boldsymbol{\theta}$

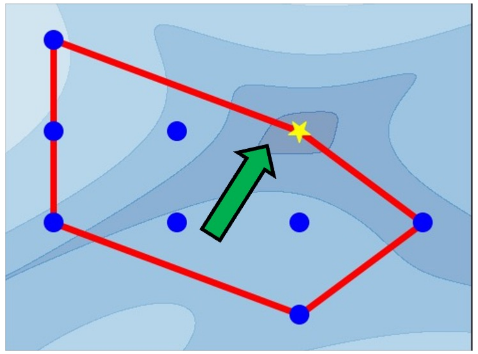


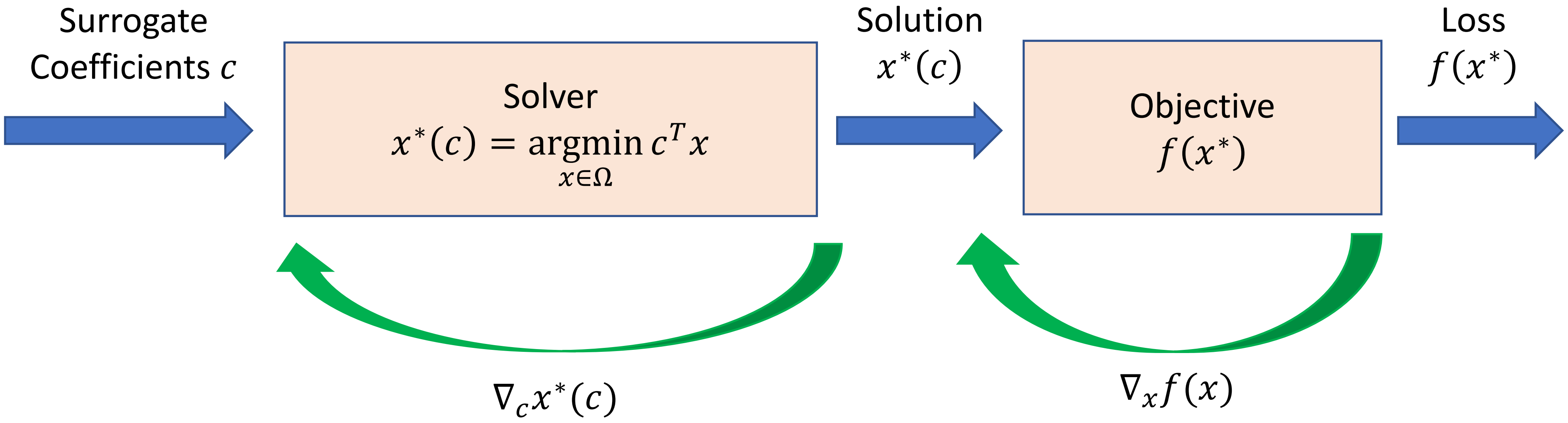
Initial Surrogate Coefficients

$$c_0 = NN(y_{\text{test}}; \theta)$$

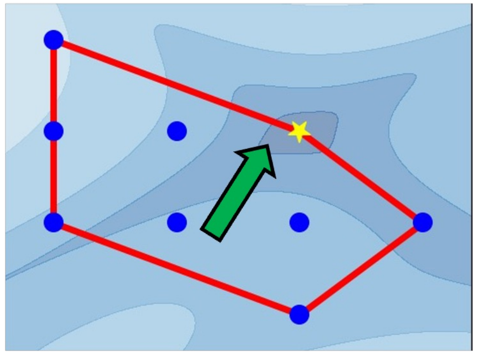


SurCo-zero



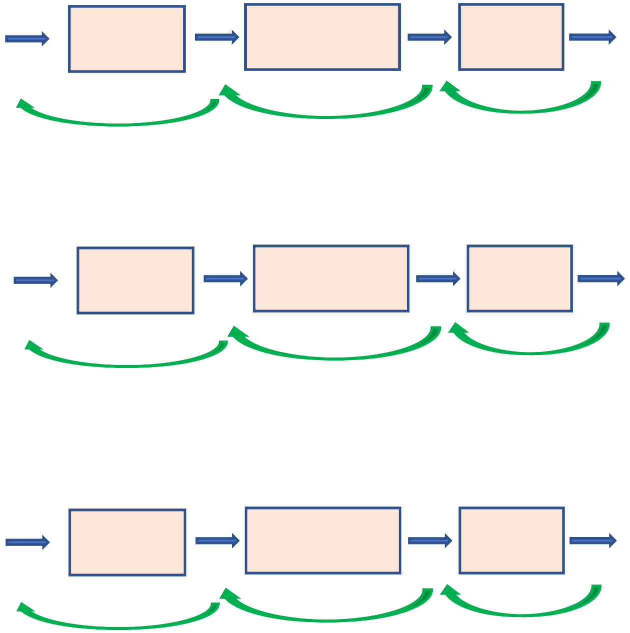


No offline training data, just solve a single problem instance on-the-fly



SurCo-prior

$c_i = NN(y_i; \theta)$



Train Model parameters $\boldsymbol{\theta}$

Surrogate Coefficients

$c_{\text{test}} = NN(y_{\text{test}}; \theta)$

Solver

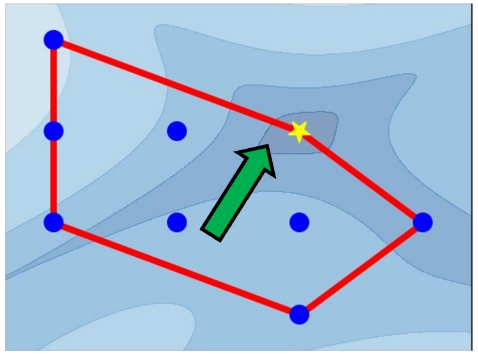
$x^*(c) = \underset{x \in \Omega}{\operatorname{argmin}}\, c^T x$

Solution

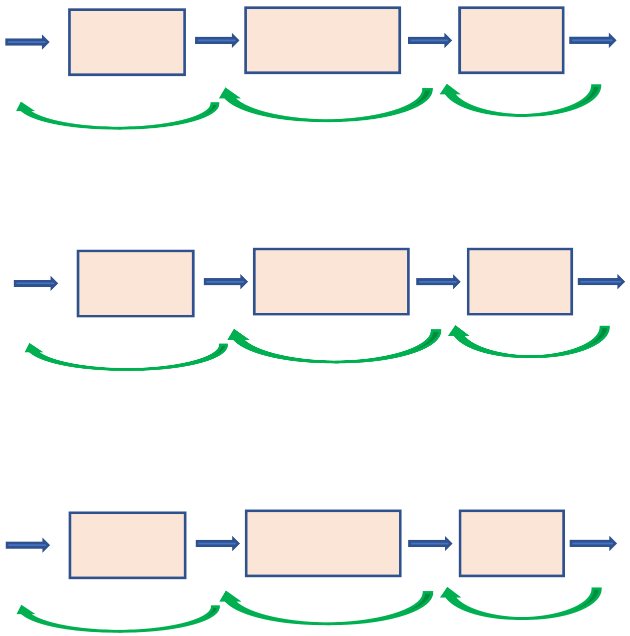
$x^*(c_{\text{test}})$

Uses offline training data to quickly solve problems at test time with just one solver call

SurCo-hybrid



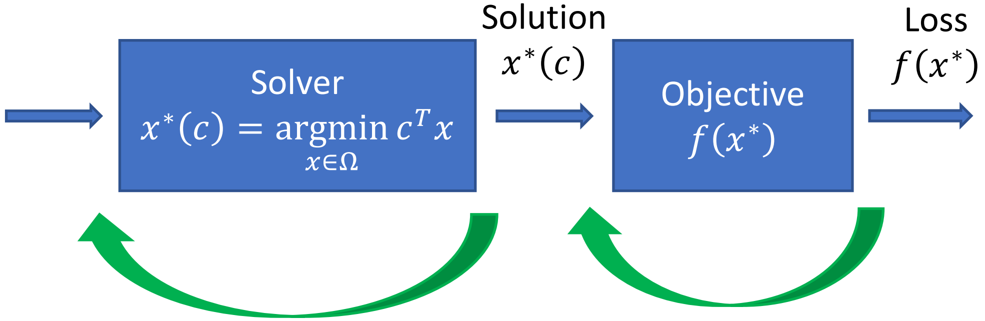
$$c_i = NN(y_i; \theta)$$



Train Model parameters $\boldsymbol{\theta}$

Initial Surrogate Coefficients

$$c_0 = NN(y_{\text{test}}; \theta)$$



Offline train + on-the-fly fine-tuning the surrogate

Related Work

Differentiable optimization: backprop through solvers

Amos et al. OptNet: Differentiable optimization as a layer in neural networks. ICML 2017

Agrawal et al. Differentiable Convex Optimization Layers. NeurIPS 2019

Berthet et al. Learning with Differentiable Perturbed Optimizers. NeurIPS 2020

Demirović et al. Predict+Optimise with Ranking Objectives: Exhaustively Learning Linear Functions. IJCAI 2019

Demirović et al. Dynamic Programming for Predict + Optimise. AAAI 2020

Djolonga et al. Differentiable Learning of Submodular Models. NeurIPS 2017

Donti et al. Task-Based End-to-End Model Learning in Stochastic Optimization. NeurIPS 2017

Elmachtoub et al. Smart "Predict, then Optimize". Management Science 2022

Ferber et al. MIPaaL: Mixed Integer Program as a Layer. AAAI 2020

Lee et al. Meta-Learning with Differentiable Convex Optimization. CVPR 2019

Mandi et al. Smart Predict-and-Optimize for Hard Combinatorial Optimization Problems. AAAI 2020

Niepert et al. Implicit MLE: Backpropagating Through Discrete Exponential Family Distributions. NeurIPS 2021

Valstelica et al. Differentiation of Blackbox Combinatorial Solvers. ICLR 2019

Rolnínek et al. Optimizing Rank-Based Metrics with Blackbox Differentiation. CVPR 2020

Wang et al. Automatically Learning Compact Quality-Aware Surrogates for Optimization Problems. NeurIPS 2020

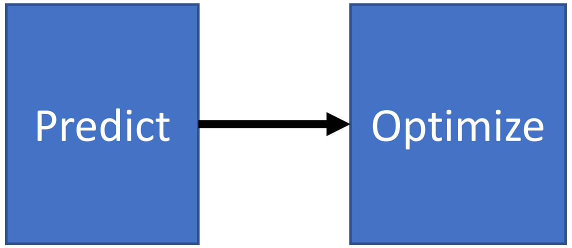
Wang et al. SATNet: Bridging Deep Learning and Logical Reasoning Using a Differentiable Satisfiability Solver. ICML 2019

Wilder et al. Melding the Data-Decisions Pipeline: Decision-focused Learning for Combinatorial Optimization. AAAI 2019

Wilder et al. End to End Learning and Optimization on Graphs. NeurIPS 2019

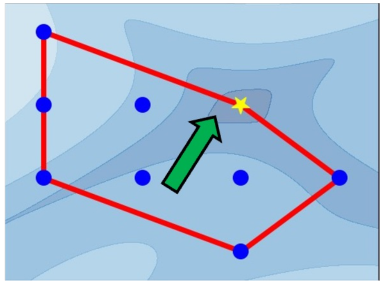
How SurCo is different from Predict+Optimise?

Predict+Optimize



- Suitable for linear optimization problems.
- Requires a ground truth linear coefficients c_i of the objective

SurCo



- Suitable for **nonlinear** objective $f(x; y)$.
- Unlike existing nonlinear solvers, **NO** analytic form needed.
- Does **NOT** require a ground truth linear coefficients c_i. Learned surrogate coefficients by itself.

Both requires contextual information (i.e., problem description y_i)

Related Work

Mixed Integer Nonlinear Optimization: general-purpose solvers

Burer et al. Non-Convex Mixed Integer Nonlinear Programming: A Survey. ORMS 2012

Belotti et al. Mixed Integer Nonlinear Optimization. Acta Numerica 2013

General-purpose heuristic optimizers: combinatorial constraints are hard

Gad et al. Pygad: An Intuitive Genetic Algorithm Python Library. 2021

Rapin et al. Nevergrad – A Gradient-Free Optimization Platform. 2018

Wang et al. Learning Search Space Partition for Black-Box Optimization Using Monte Carlo Tree Search. NeurIPS 2020

Wang et al. Sample Efficient Neural Architecture Search by Learning Actions for Monte Carlo Tree Search. PAMI 2021

RL for combinatorial optimization: combinatorial constraints are hard

Khalil et al. Learning Combinatorial Optimization Algorithms Over Graphs. NeurIPS 2017

Kool et al. Attention, Learn to Solve Routing Problems! ICLR 2018

Mazyavkina et al. Reinforcement Learning for Combinatorial Optimization: A Survey. COR 2021

Nazari et al. Reinforcement Learning for Solving the Vehicle Routing Problem. NeurIPS 2018

Zhang et al. A Reinforcement Learning Approach to Job-Shop Scheduling. IJCAI 1995

Embedding Table Sharding

Used in large-scale deep learning systems: recommendation systems, knowledge graph

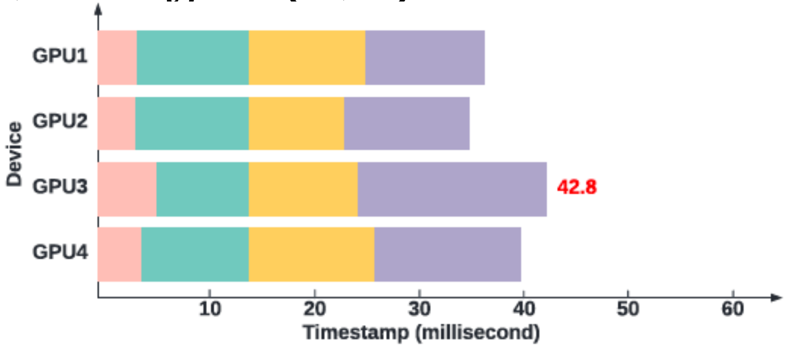
Place N “tables” (with known memory need m_i) on K devices ($x_{ij} = 1$: table i assigned to device j)

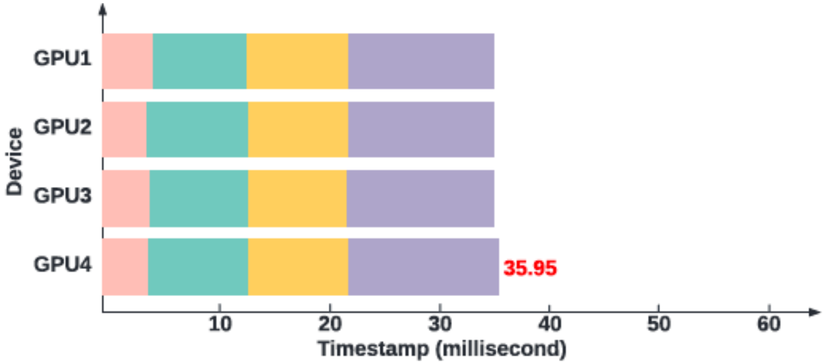
$$\text{Min}_x\, L(\{x_{ij}\}) \quad \text{s.t. } \sum_i x_{ij} m_i \le M_j, \quad \sum_j x_{ij} = 1, \quad x_{ij} \in \{0,1\}$$

L : Runtime bottleneck f(x) estimated by NN (longest-running device)

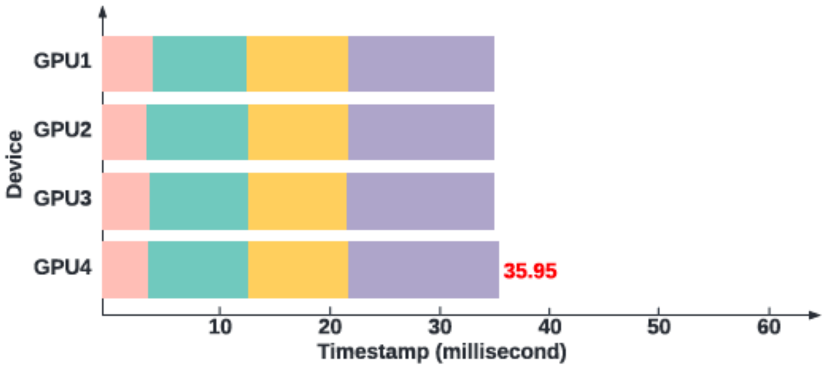
L is nonlinear due to system issues
(e.g., batching, communication, etc.)

$c(y;\theta)$ gives surrogate ”per-table cost” c_{ij}
(and $\sum_{ij} c_{ij} x_{ij}$ is the surrogate latency objective)





Embedding Table Sharding

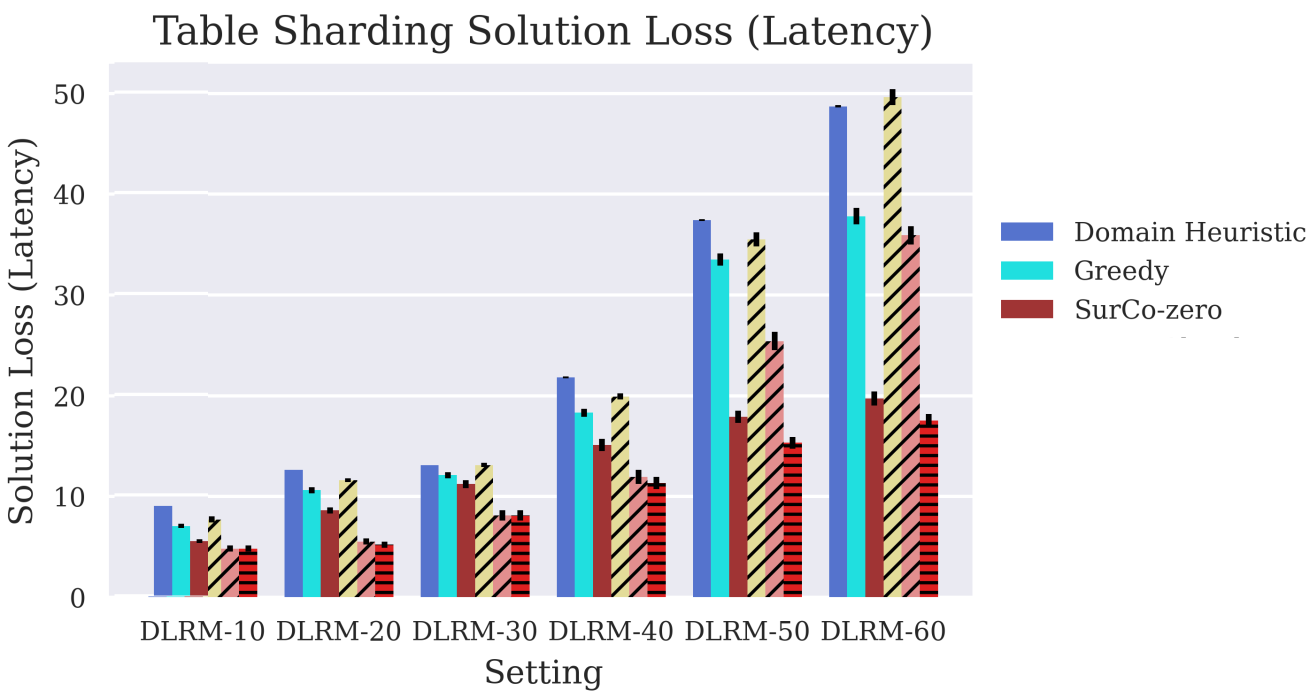


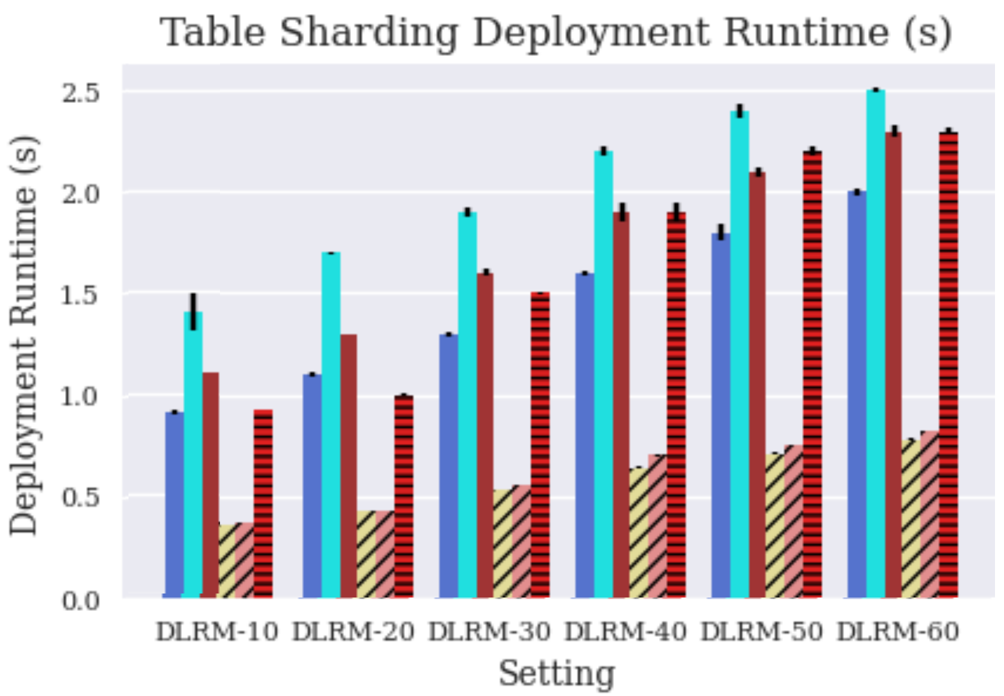
- Public **D**eep **L**earning **R**ecommendation **M**odel (DLRM dataset) placing between 10 to 60 tables on 4 GPUs

- Baseline: Greedy
- SoTA: RL approach Dreamshard[1]
- SurCo: Surrogate NN model learned via CVXPYLayers (differentiable LP Solver)

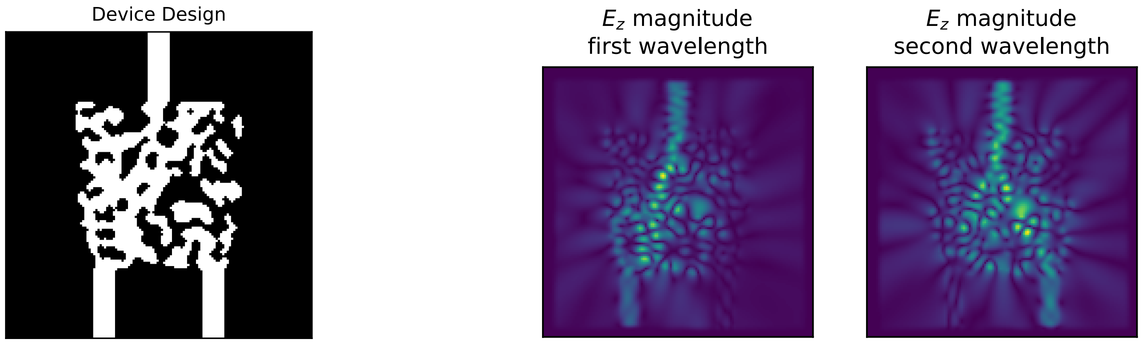
[1] Zha et al. NeurIPS 2022
Dataset: https://github.com/facebookresearch/dlrm_datasets

Results – Table Sharding





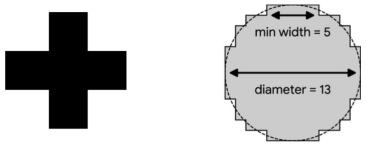
Inverse Photonic Design



- Design physically-viable devices that take light waves and routes different wavelengths to correct locations

$$\mathcal{L}(S) = \left(\left\| \operatorname{softplus}\left(g \frac{|S|^2 - |S_{\text{cutoff}}|^2}{\min(w_{\text{valid}})} \right) \right\|_2 \right)^2$$

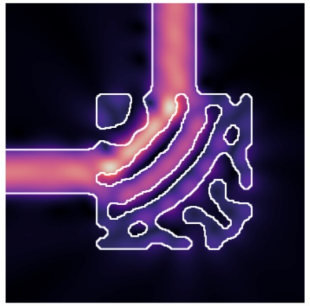
- Device design misspecification loss f(x) computed by differentiable electromagnetic simulator
- Feasible solution: the design must be the union of brush pattern
  - x = binary_opening(x, brush)
  - x = ~binary_opening(~x, brush)

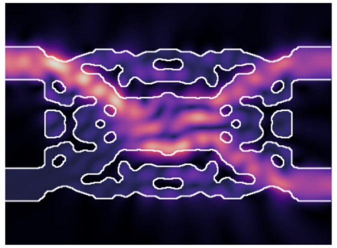


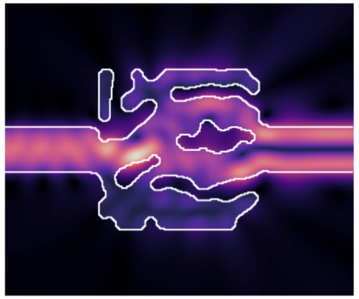
Inverse Photonic Design

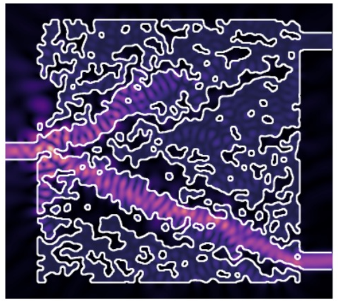
- Dataset: Ceviche Challenges[1]
- Most baselines don't work here due to combinatorial constraints
- SoTA: Brush-based algorithm [1]

- SurCo: Surrogate learned via blackbox differentiation[2] of brush solver


Waveguide bend

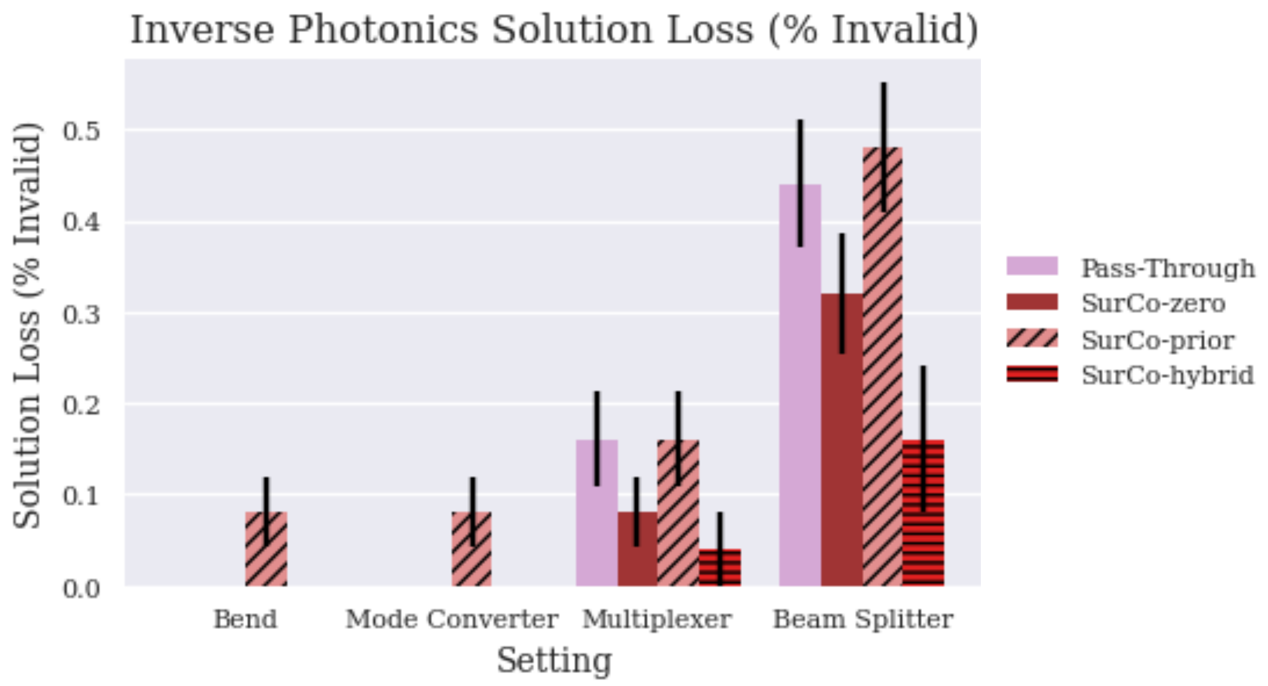

Beam splitter

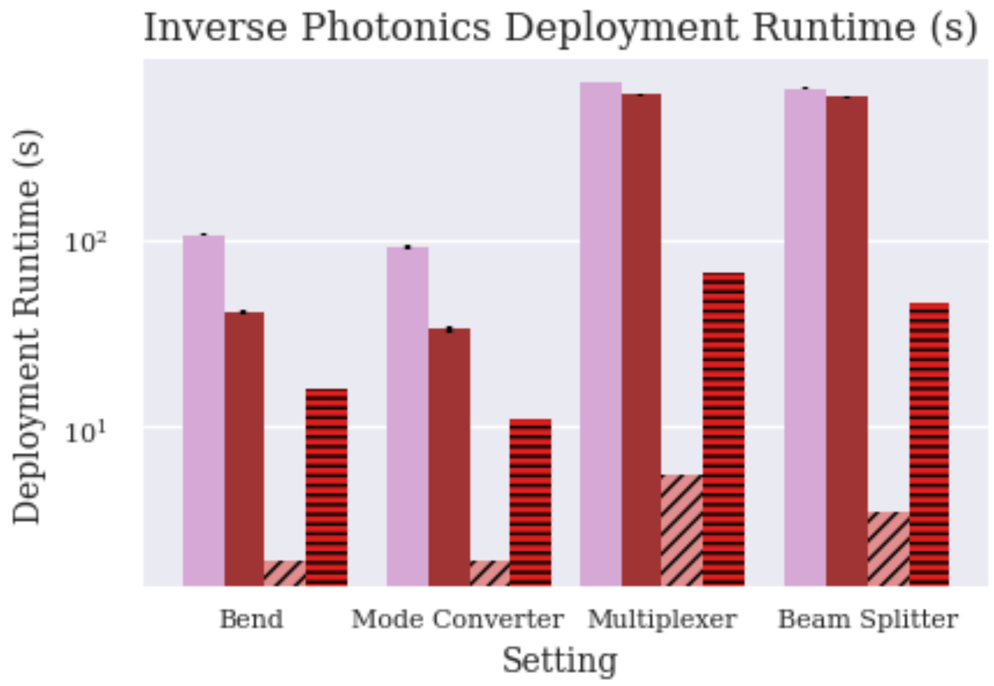

Mode converter


Wavelength division multiplexer

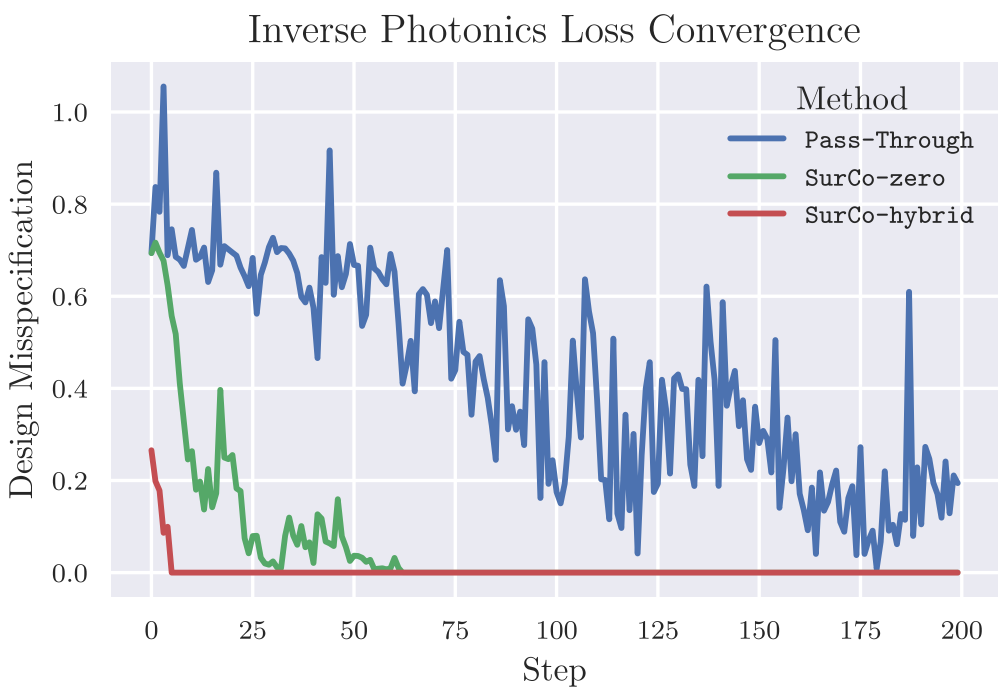
[1]Schubert et al. ACS Photonics 2022
[2]Vlastelica et al. ICLR 2019
Dataset: https://github.com/google/ceviche-challenges

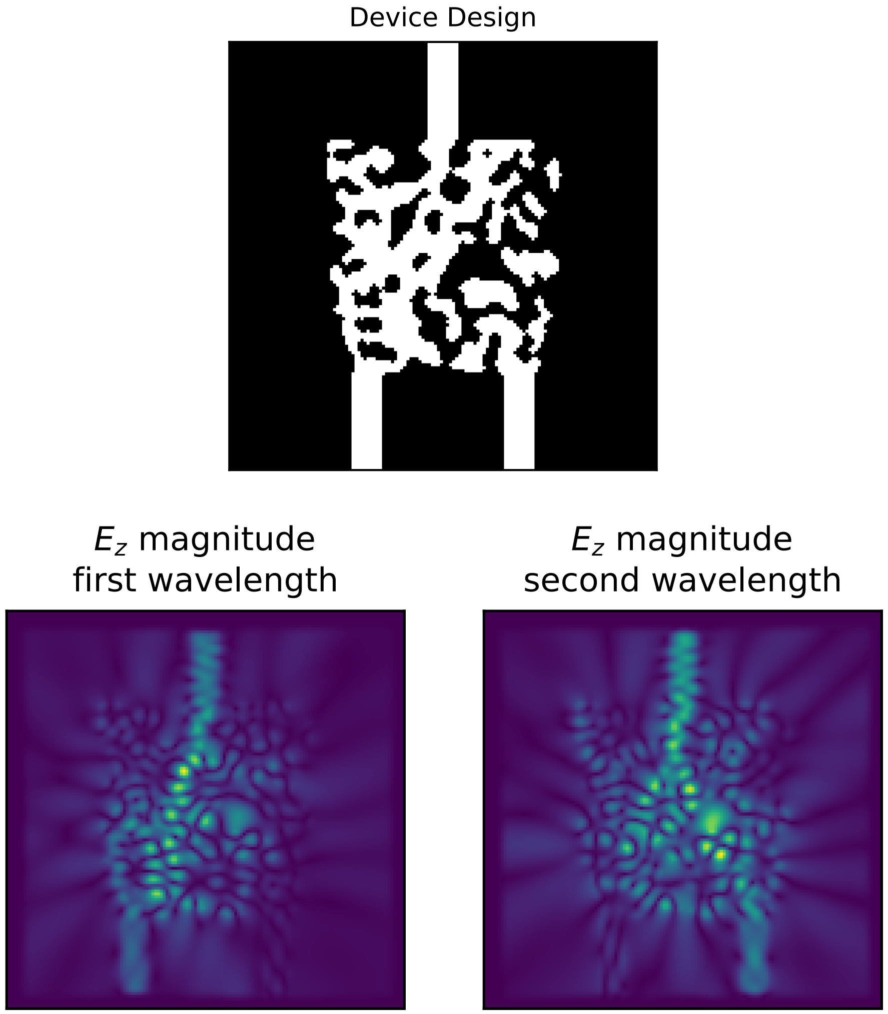
Results – Inverse Photonics





Inverse photonics Convergence comparison + Solution example



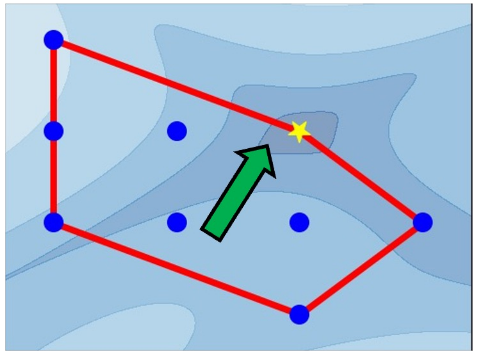


Wavelength division multiplexer

Takeaways:

- SurCo-Zero finds loss-0 solutions quickly
- SurCo-Hybrid uses offline training data to get a head start

Conclusion



- Handle industrial applications with differentiable optimization
- High-quality solutions to combinatorial nonlinear optimization by finding linear surrogates
  - Sometimes we can find “easier” surrogate problems that solve much more difficult instances
- SurCo works in several data settings
  - Zero-shot vs Offline training
  - One step inference vs fine-tuning

Sample-efficient Surrogate Model for Frequency Response of Linear PDEs using Self-Attentive Complex Polynomials

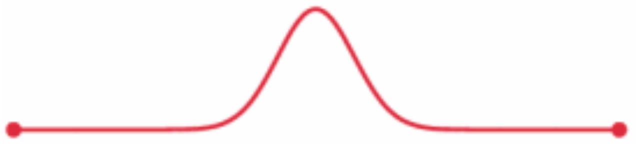
Andrew Cohen[1]*, Weiping Dou[2], Jiang Zhu[2], Slawomir Koziel[3], Peter Renner[2],
Jan-Ove Mattson[2], Xiaomeng Yang[1], Beidi Chen[1,4], Kevin Stone[1], Yuandong Tian[1]*

[1]Meta AI (FAIR), [2]Reality Lab Antenna (Meta), [3]Reykjavik University, [4]Carnegie Mellon University
* = Equal technical contribution

https://arxiv.org/abs/2301.02747

Background

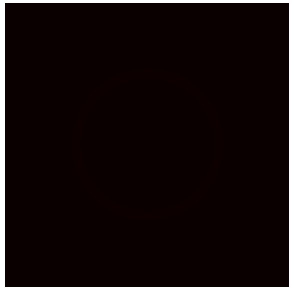
Solving the linear PDE



$$\frac{\partial^n \psi}{\partial t^n} = F(\psi, \nabla_{\boldsymbol{x}}\psi, \ldots; \boldsymbol{h})$$

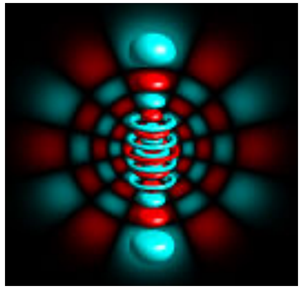
$\psi(x, t)$ is the spatial-temporal signal under time evolutions.
F is a linear function with respect to ψ and its derivatives
$\boldsymbol{h}$ is design choice.

Examples



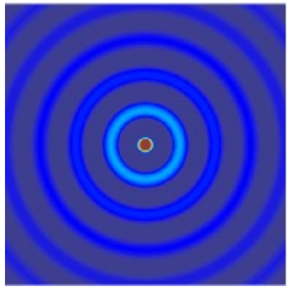
$$\frac{\partial \psi}{\partial t} = \nabla^2 \psi$$

Heat equation



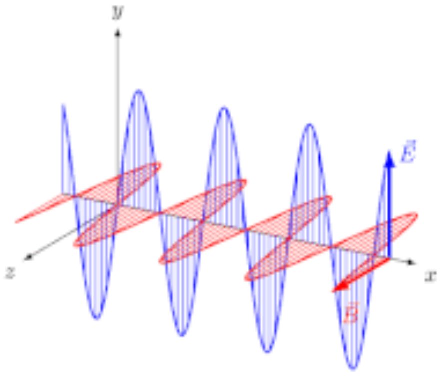
$$i\hbar \frac{\partial \psi}{\partial t} = \left[-\frac{\hbar^2}{2m} \nabla^2 + V \right] \psi$$

Schrodinger's Equation



$$\frac{\partial^2 \psi}{\partial t^2} = \mathrm{c}^2 \nabla^2 \psi$$

Wave equation



$$\nabla \cdot E = \frac{\rho}{\epsilon_0}, \nabla \cdot B = 0$$

$$\nabla \times E = -\frac{\partial B}{\partial t}, \nabla \times B = \mu_0 j + \frac{1}{c^2} \frac{\partial E}{\partial t}$$

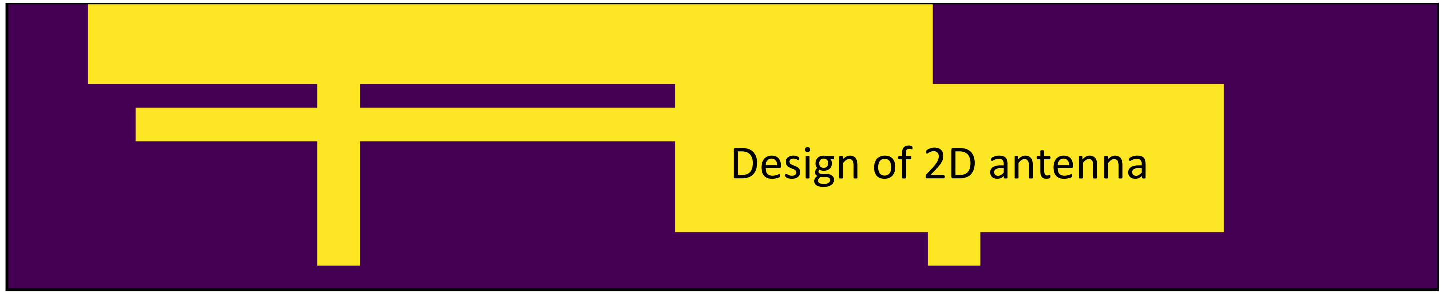
Maxwell's equation

Tricky to simulate accurately and efficiently → Can we do better?

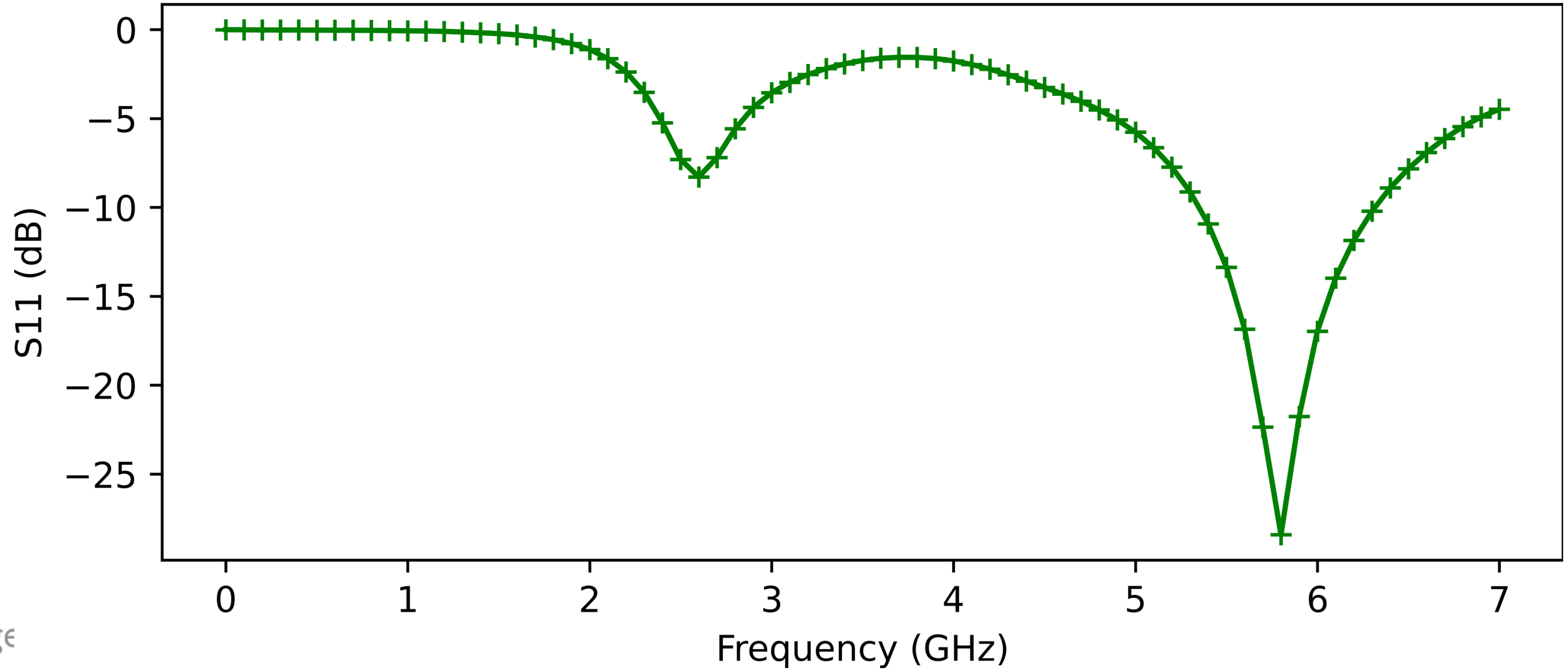
Antenna Design problem

Goal:
find the right design to achieve the right frequency response

$\boldsymbol{h} =$



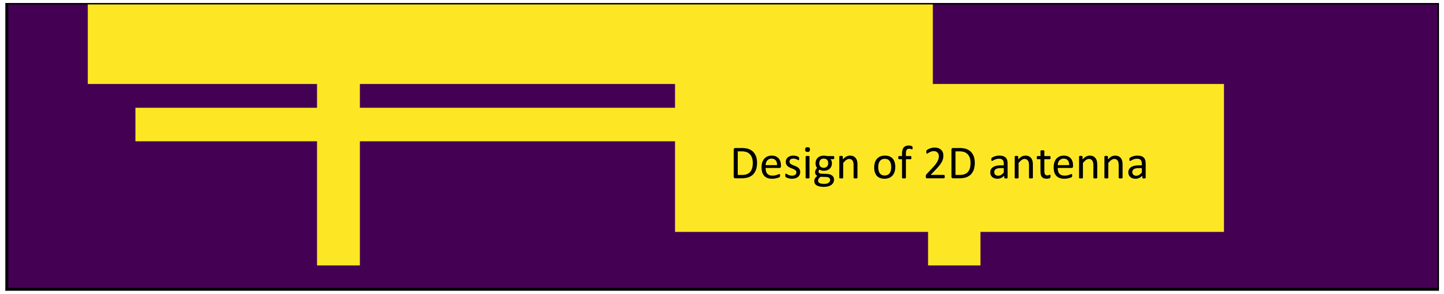
$S_{11}(\psi) =$



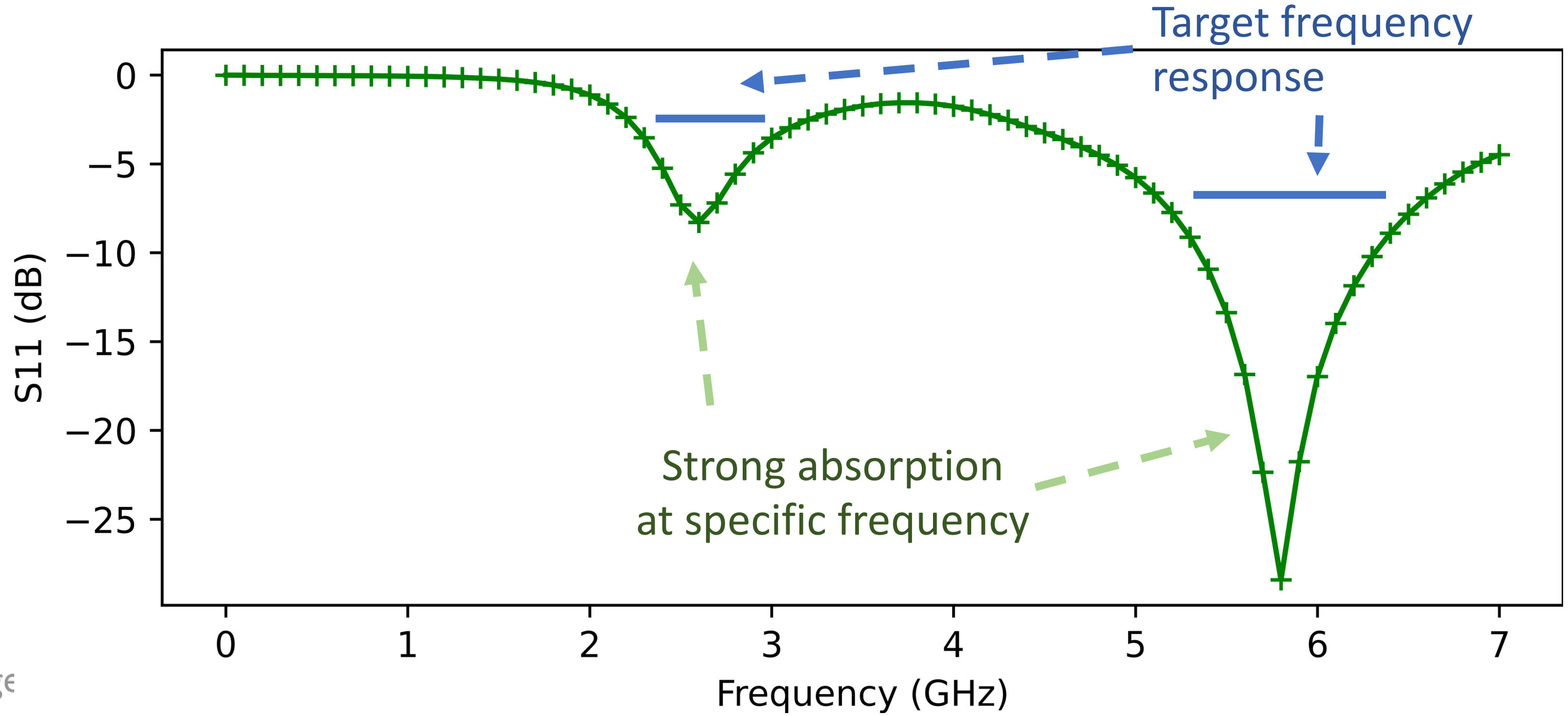
Antenna Design problem

Goal:
find the right design to achieve the right frequency response

$$\boldsymbol{h} =$$



$$S_{11}(\psi) =$$



Discretization of linear PDE systems

$$\frac{\partial^n \psi}{\partial t^n} = F(\psi, \nabla_{\boldsymbol{x}}\psi, \dots; \boldsymbol{h}) \quad \Rightarrow \quad \frac{\partial \phi}{\partial t} = A(\boldsymbol{h})\phi$$

The matrix A encodes the information of F

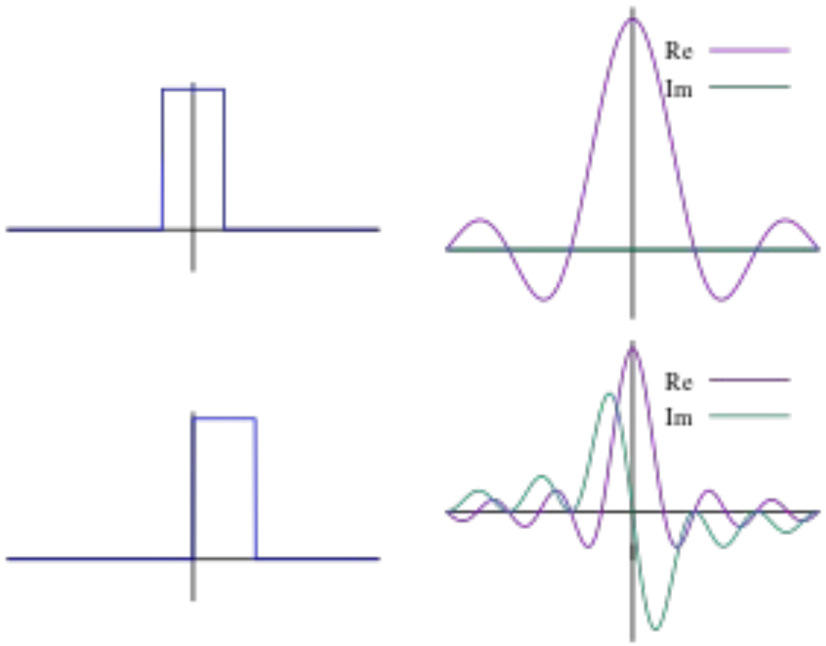
One example: $\frac{\partial^2 \psi}{\partial t^2} = c^2 \nabla^2 \psi \quad \Rightarrow$

$$\phi = \left[\psi(x_1), \dots \psi(x_N), \frac{\partial \psi}{\partial t}(x_1), \dots, \frac{\partial \psi}{\partial t}(x_N)\right]$$

$$A = \begin{bmatrix} 0 & I \\ c^2 B & 0 \end{bmatrix}$$

Discretized onto *N* vertices

Frequency Domain



Signal in the temporal domain: $\phi(x, t)$

Vector form $\boldsymbol{\phi}(t) = [\phi(x_1, t), \dots \phi(x_N, t)]$

Signal in the frequency domain: $\hat{\phi}(x, \omega) = \int \phi(x, t) e^{-\mathrm{i}\omega t} \mathrm{d}t$

Vector form $\hat{\boldsymbol{\phi}}(t) = [\hat{\phi}(x_1, \omega), \dots \hat{\phi}(x_N, \omega)]$

Parametric formula for Linear PDEs

<u>Theorem:</u> For any linear coefficients $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$:

$$\frac{\boldsymbol{b}_1^T \widehat{\boldsymbol{\phi}}(\omega)}{\boldsymbol{b}_2^T \widehat{\boldsymbol{\phi}}(\omega)} = c_0(\boldsymbol{h}) \prod_{k=1}^{K_1} (\omega - \mathbf{z}_k(\boldsymbol{h})) \prod_{k=1}^{K_2} \left(\omega - \mathbf{p}_k(\boldsymbol{h})\right)^{-1}$$

where the constant $c_0(\boldsymbol{h})$, zeros $z_k(\boldsymbol{h})$ and poles $p_k(\boldsymbol{h})$ are complex functions of the design choice $\boldsymbol{h}$

Proof idea: Linear ODE theory gives us the analytic form of the solution $\boldsymbol{\phi}(t) = e^{At}\boldsymbol{\phi}(0)$. Fourier Transform yields $\widehat{\boldsymbol{\phi}}(\omega)$ as a rational function of complex polynomials w.r.t. frequency ω.

For Antenna Optimization

The *Scattering Coefficients* $S_{11}(\omega)$:

$$S_{11}(\omega) = \frac{Z_{\text{in}}(\omega) - Z_0}{Z_{\text{in}}(\omega) + Z_0}$$

$Z_{\text{in}}(\omega)$: Input *Impedance.* Impedance $\mathrm{Z}(\omega) \coloneqq V(\omega)/I(\omega)$

Voltage
(in Fourier domain)

Current
(in Fourier domain)

Both are linear function w.r.t. signal $\widehat{\boldsymbol{\phi}}(\omega)$

Parameter form of $\log|S_{11}(\omega)|$

$$\log|S_{11}(\omega)| = \log|\mathbf{c}_o(\boldsymbol{h})| + \sum_{k=1}^{K} \log \frac{|\omega - \boldsymbol{z}_{\boldsymbol{k}}(\boldsymbol{h})|}{|\omega - \boldsymbol{p}_{\boldsymbol{k}}(\boldsymbol{h})|}$$

where the constant $c_0(\boldsymbol{h})$, zeros $z_k(\boldsymbol{h})$ and poles $p_k(\boldsymbol{h})$ are complex functions of the design choice $\boldsymbol{h}$.

CZP model architecture

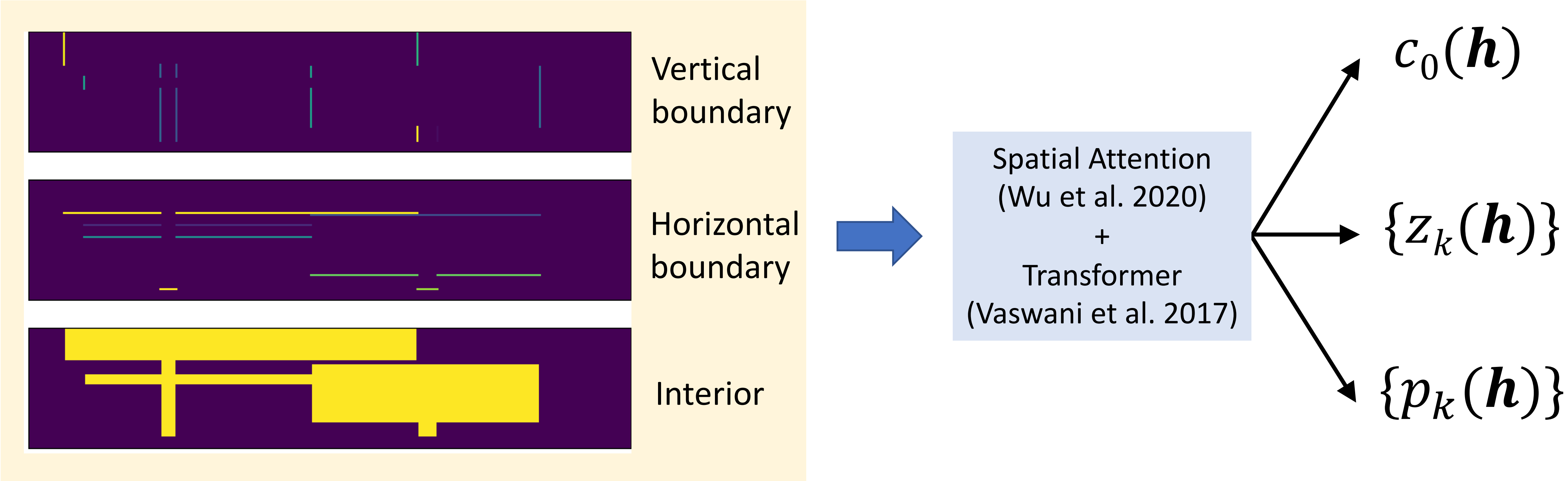
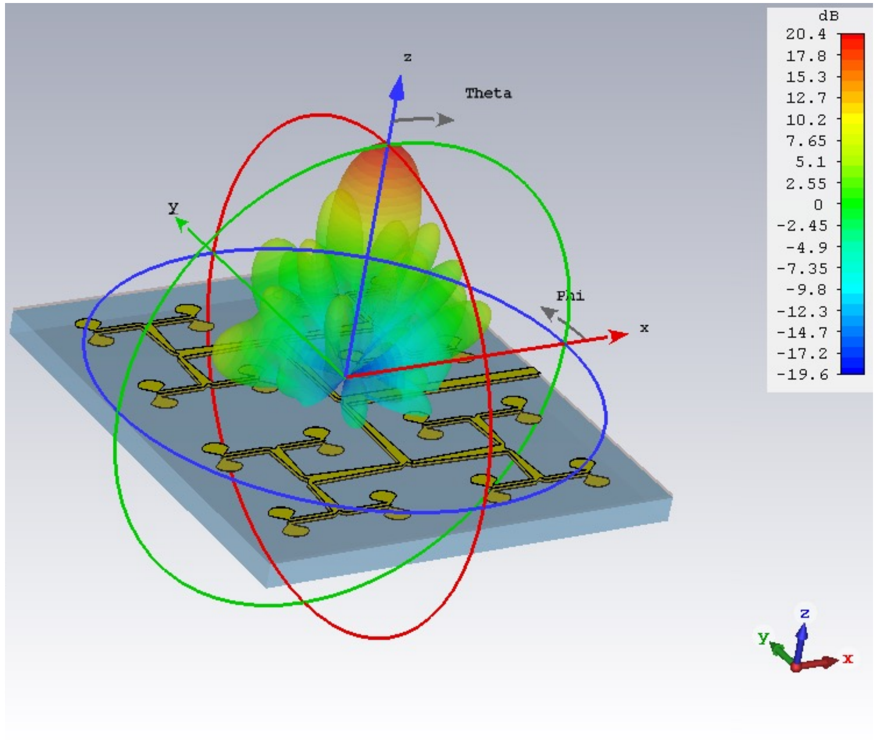


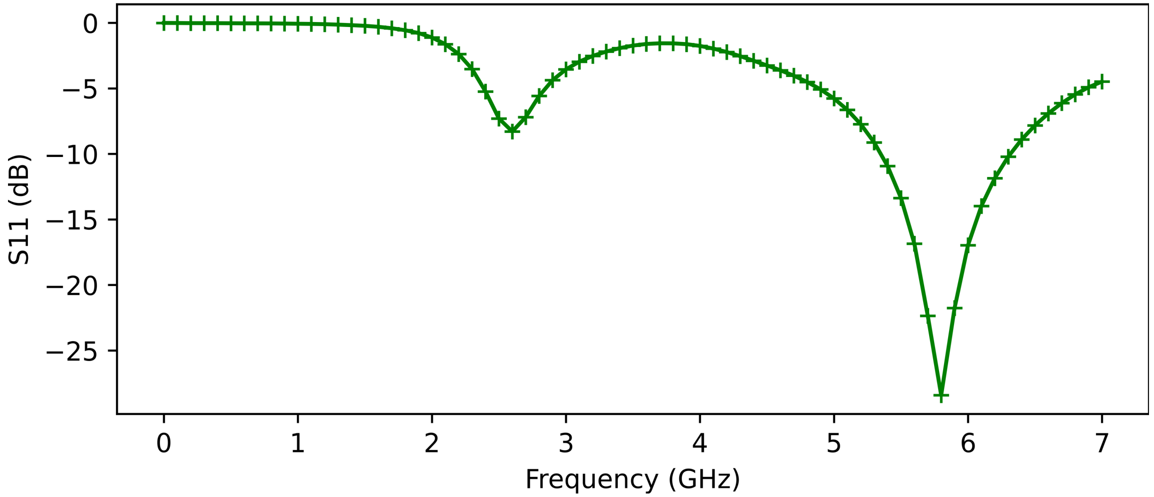
Image-based Representation of design choice $\boldsymbol{h}$

Predict the constant, zeros and poles from an image representation of an antenna

Data Collection

Dataset is collected from commercial simulators (e.g., CST)
Numerical simulation of Electromagnetic wave dynamics





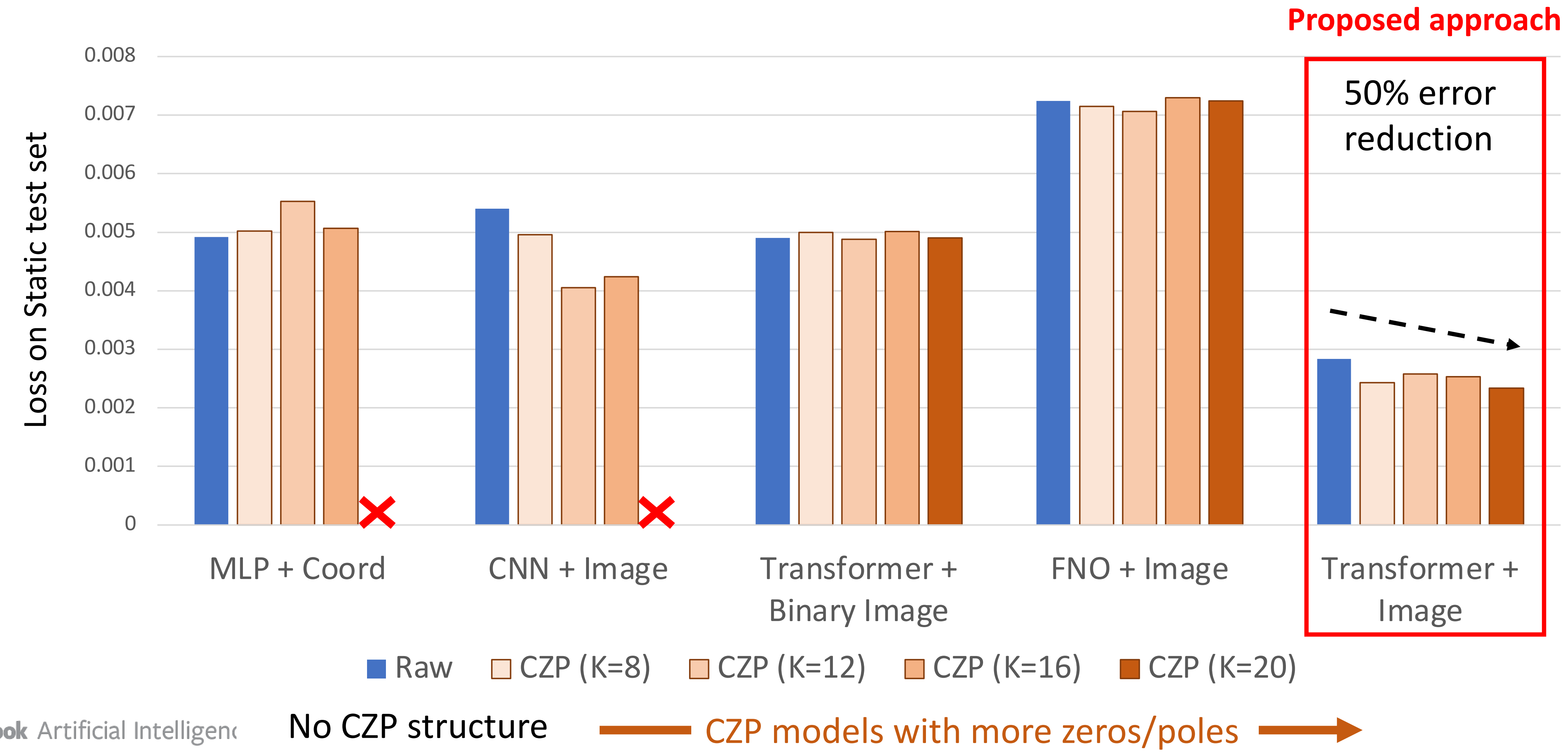
It takes minutes (or even hours) to get one simulation data point.
Dataset size = 48k

Model Evaluation

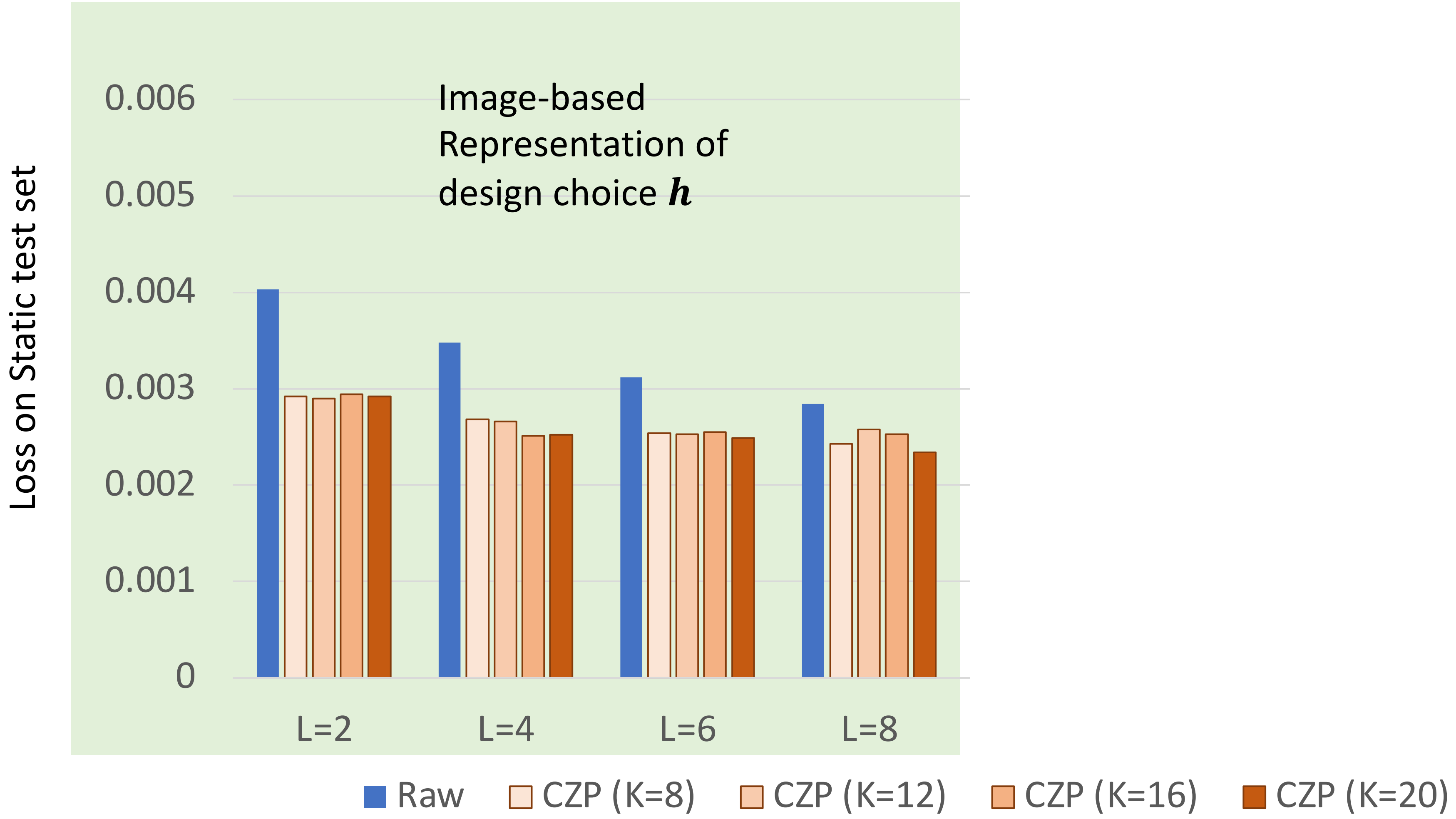
- Static Evaluation
  - On a held-out test set, compute the loss
  - Loss = gap between surrogate models and ground truth (commercial software)

- Dynamic Evaluation
  - Use the surrogate model to search a good design
  - Evaluate the design in ground truth (commercial software)

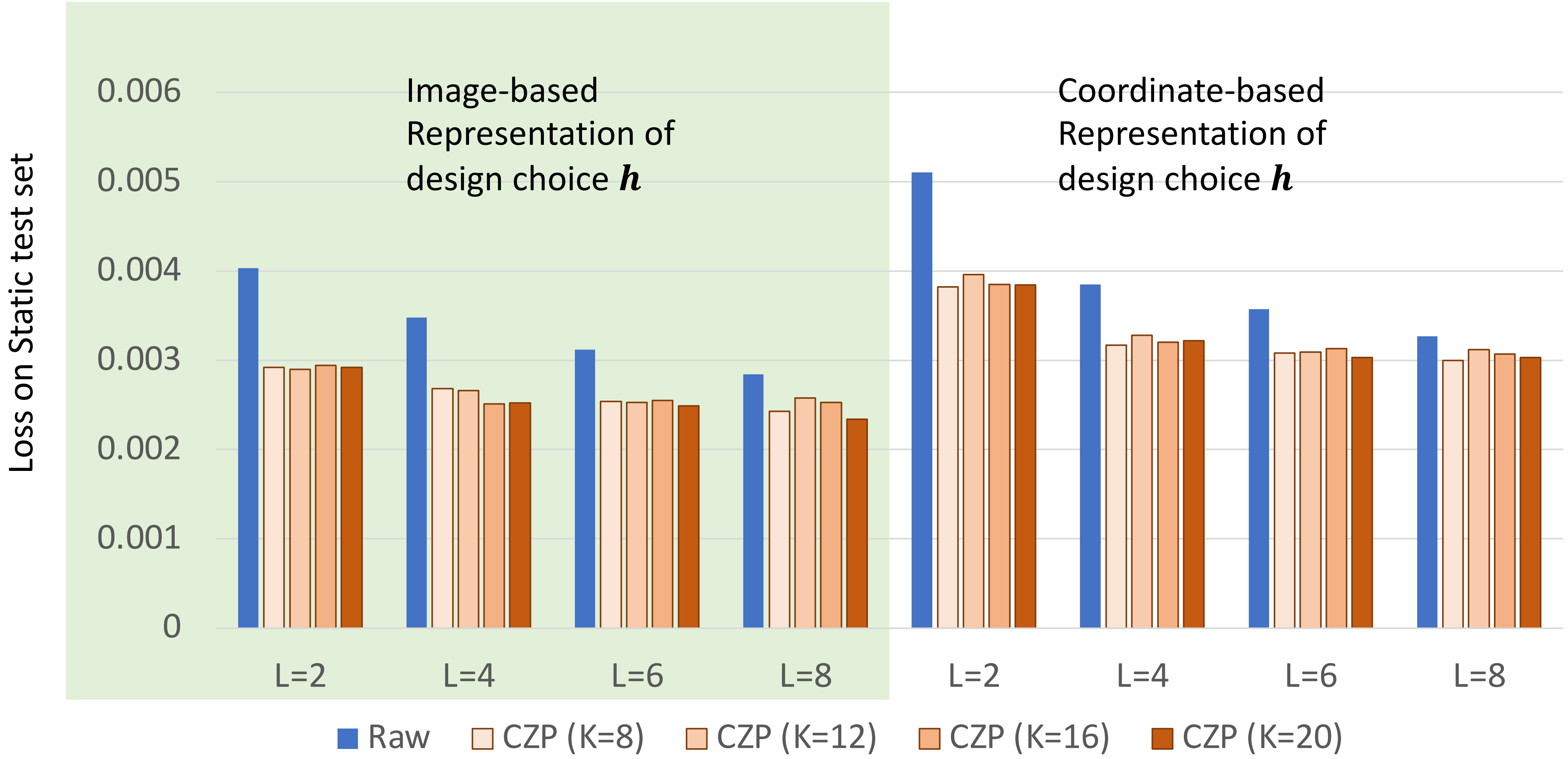
Static Evaluation: Surrogate Model Test Loss



Static Evaluation: Surrogate Model Test Loss

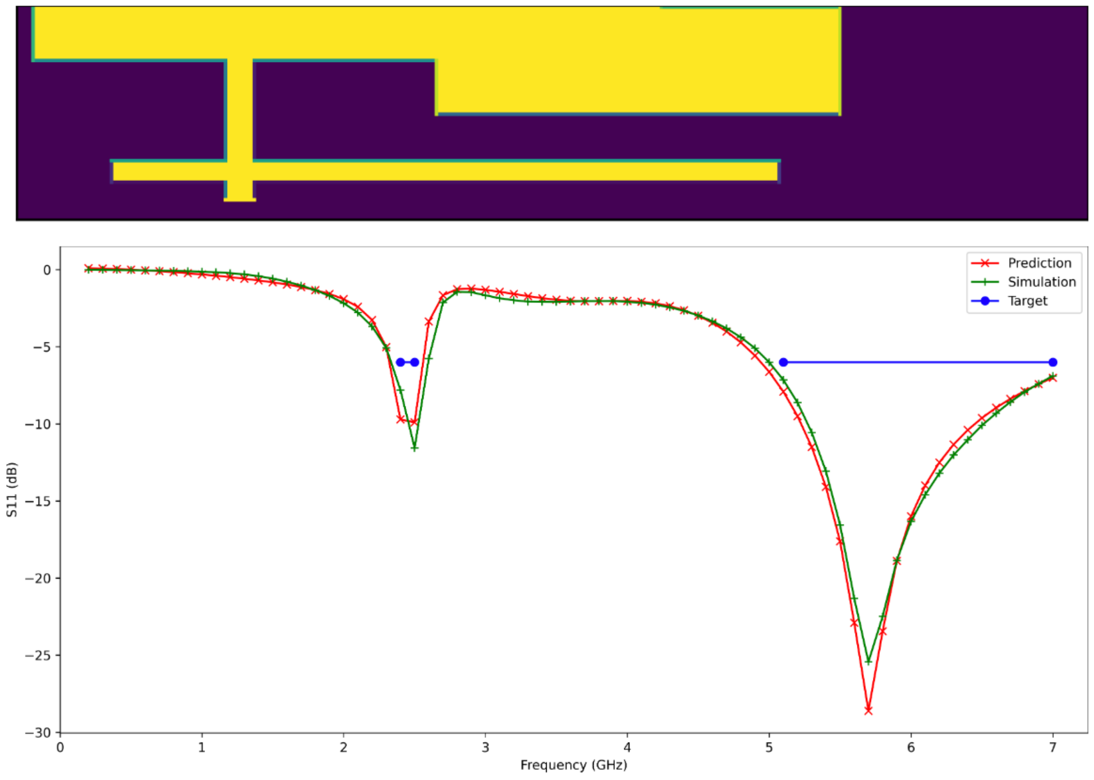


Static Evaluation: Surrogate Model Test Loss

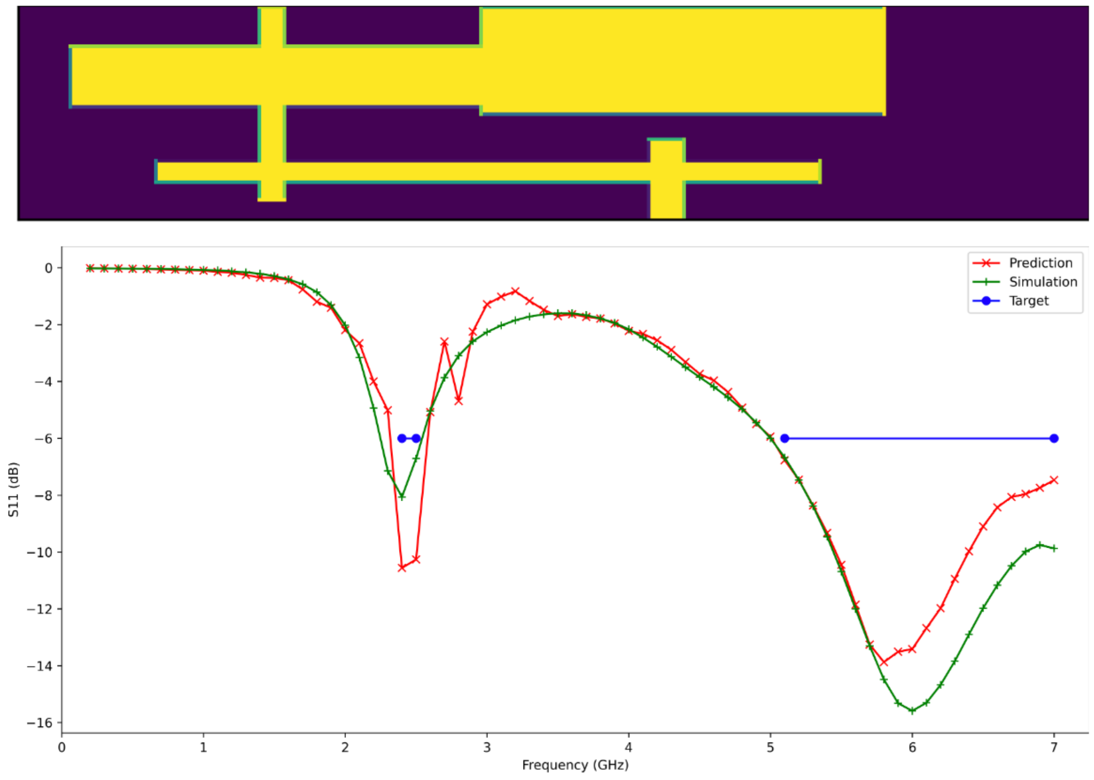


Visualization

Our CZP model captures the smooth structure of scattering coefficients $S_{11}(\omega)$



(a) CZP



(b) Raw

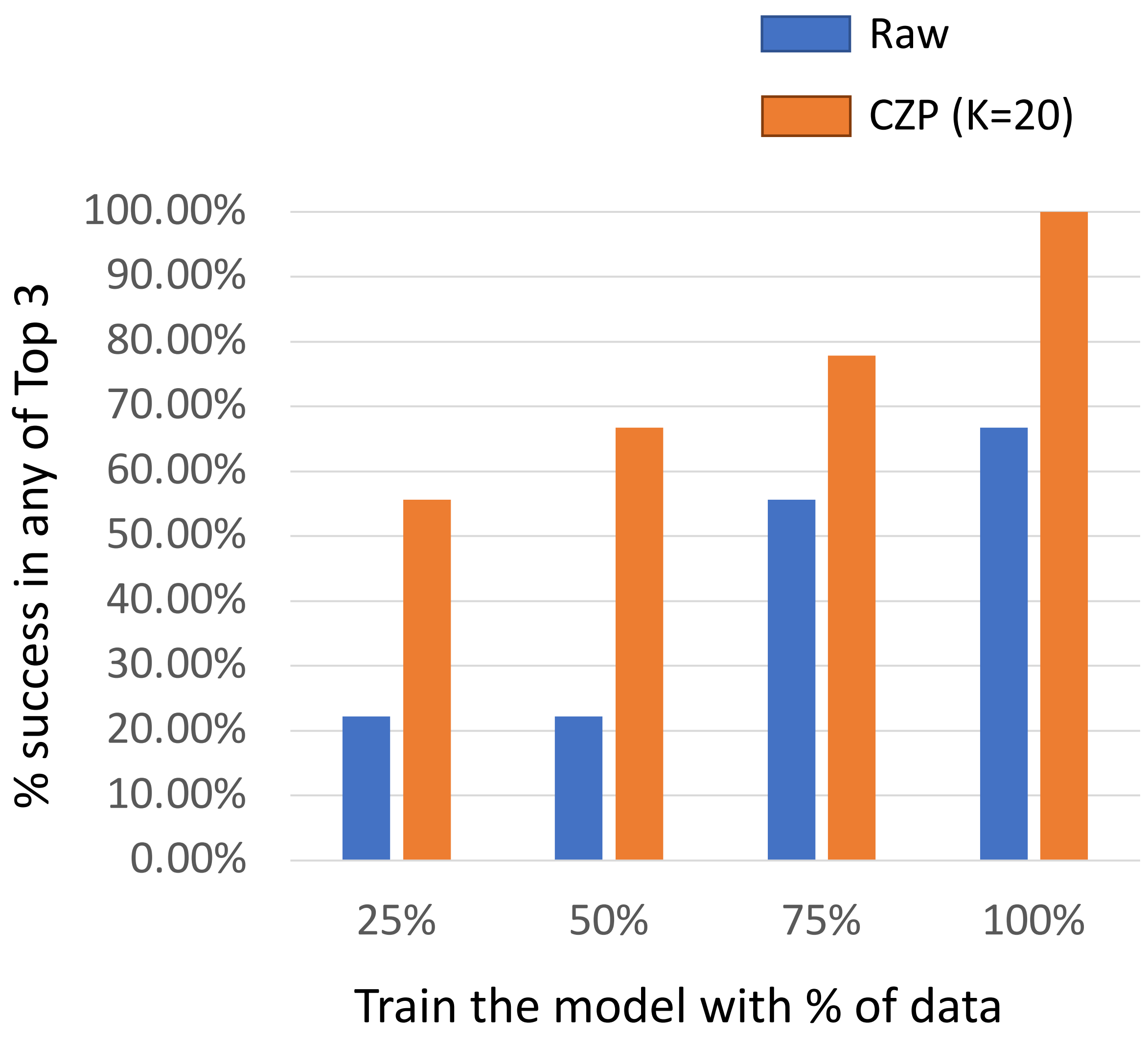
Dynamic Evaluation: CZP model with Search

We use Soft Actor-Critic
as the specific search technique.

Goal: to find a solution to satisfy the frequency constraints (verified with CST)

3 models trained x 3 search attempts using different random seed

For each attempt, check top-3 solutions



Future Work

- The formulation applies to general linear PDEs
  - Maxwell's Equations
  - Schrodinger's Equations
  - Many more …
- Test on more complicated scenarios.
  - 3D antenna

Thanks!