ESwML 2024

Auto-HPCnet: an Automatic Framework to Build Neural Network-based Surrogate for HPC Applications

Wenqian Dong*^, Gokcen Kestor# and **Dong Li***

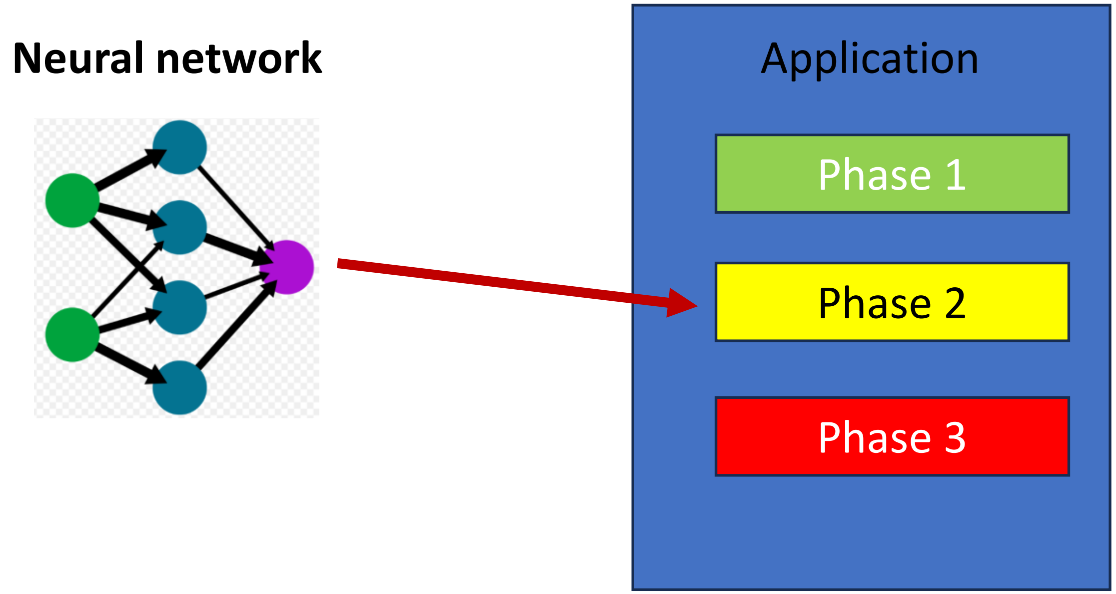
***University of California, Merced**

^Florida International University

#Pacific Northwest National Laboratory

Published in HPDC'23

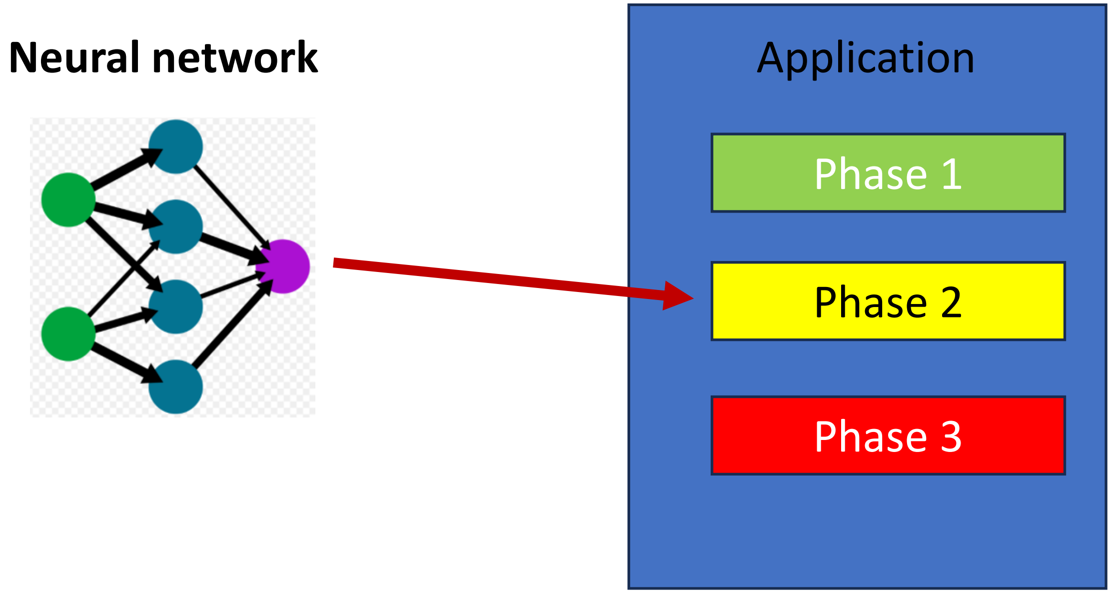
What is the neural network-based surrogate?



Replace a numerical solver or an execution phase in the HPC application with a neural network (NN) model

Goal: achieve performance improvement (i.e., reducing run time) without losing application-outcome quality

Replace a numerical solver or an execution phase in the HPC application with a neural network (NN) model



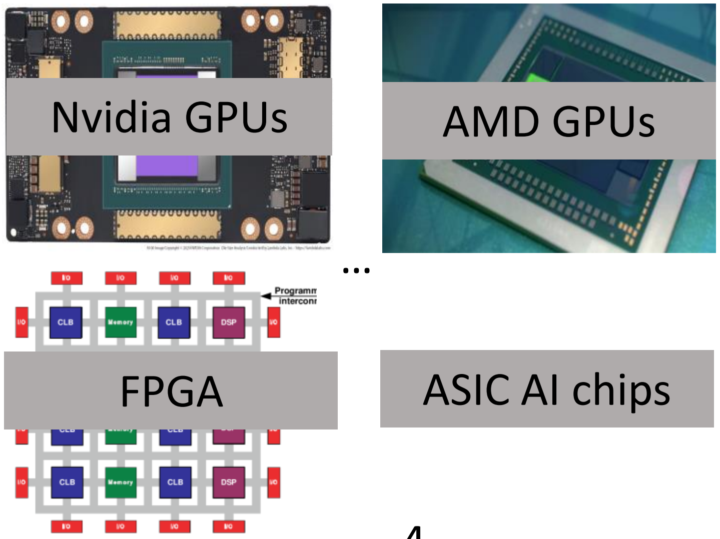
- NN and execution phase share the same input/output
- The HPC application must tolerate approximation
- This method is not universal

Benefits of neural network-based surrogate

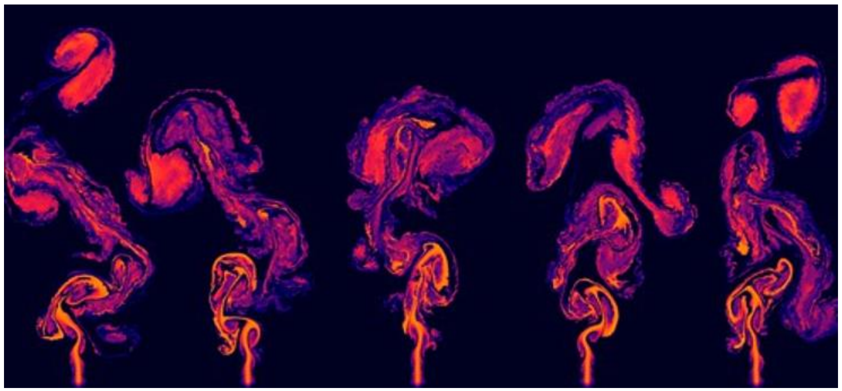
New opportunities for performance optimization

- Remove data dependency in the original code
- Remove irregular memory-access patterns

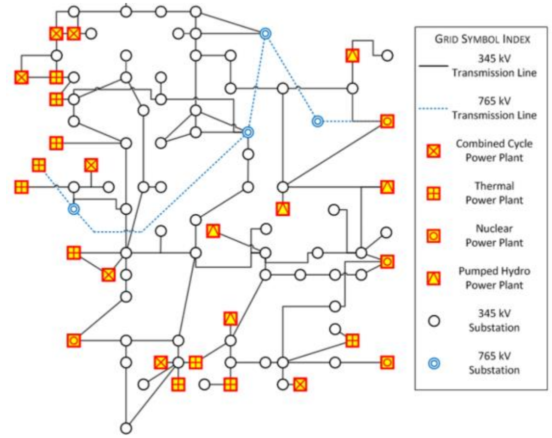
Adaptive to emerging AI accelerators



Success of neural network–based surrogate

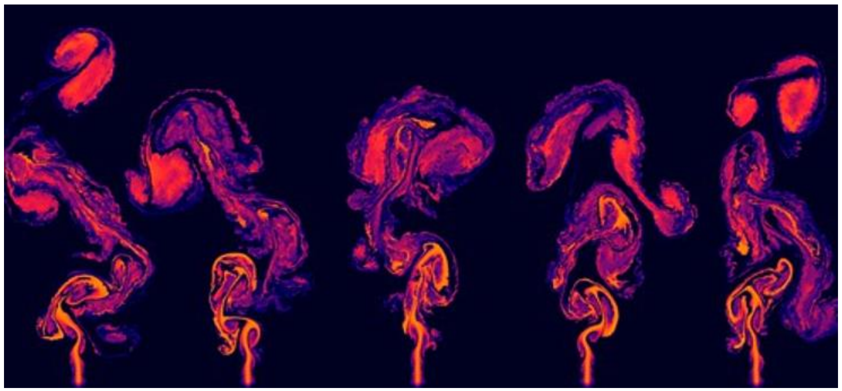


- **Eulerian fluid simulation**:
  Smart-fluidnet (SC'19)
- 590× speedup while providing better simulation quality

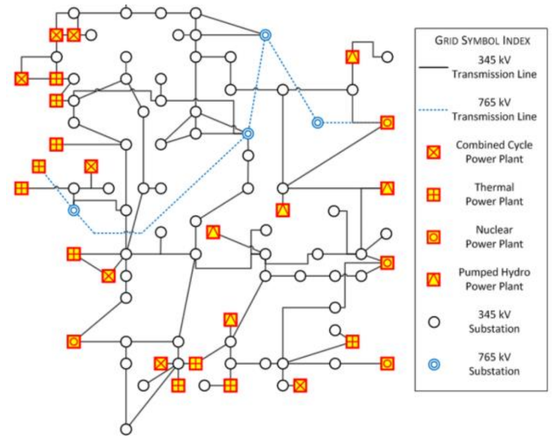


- **Power-grid simulation**:
  Smart-PGSim (SC'20)
- 2.60× speedup over 10,000 problems without losing solution optimality.

Success of neural network-based surrogate



- **Eulerian fluid simulation**: Smart-fluidnet (SC'19)
- 590× speedup while providing better simulation quality



- **Power-grid simulation**: Smart-PGSim (SC'20)
- 2.60× speedup over 10,000 problems without losing solution optimality.

Quantum chemistry

Climate science

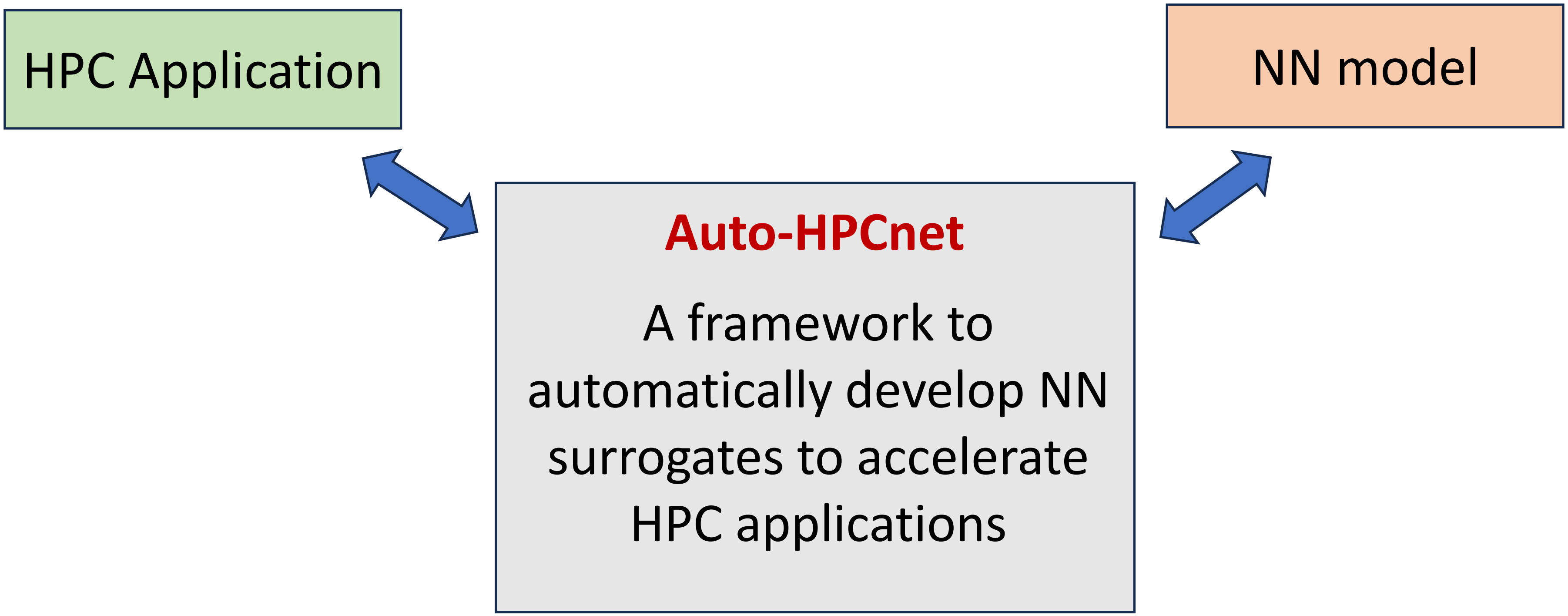
Hydrology

Challenges of building neural network (NN) –based surrogate

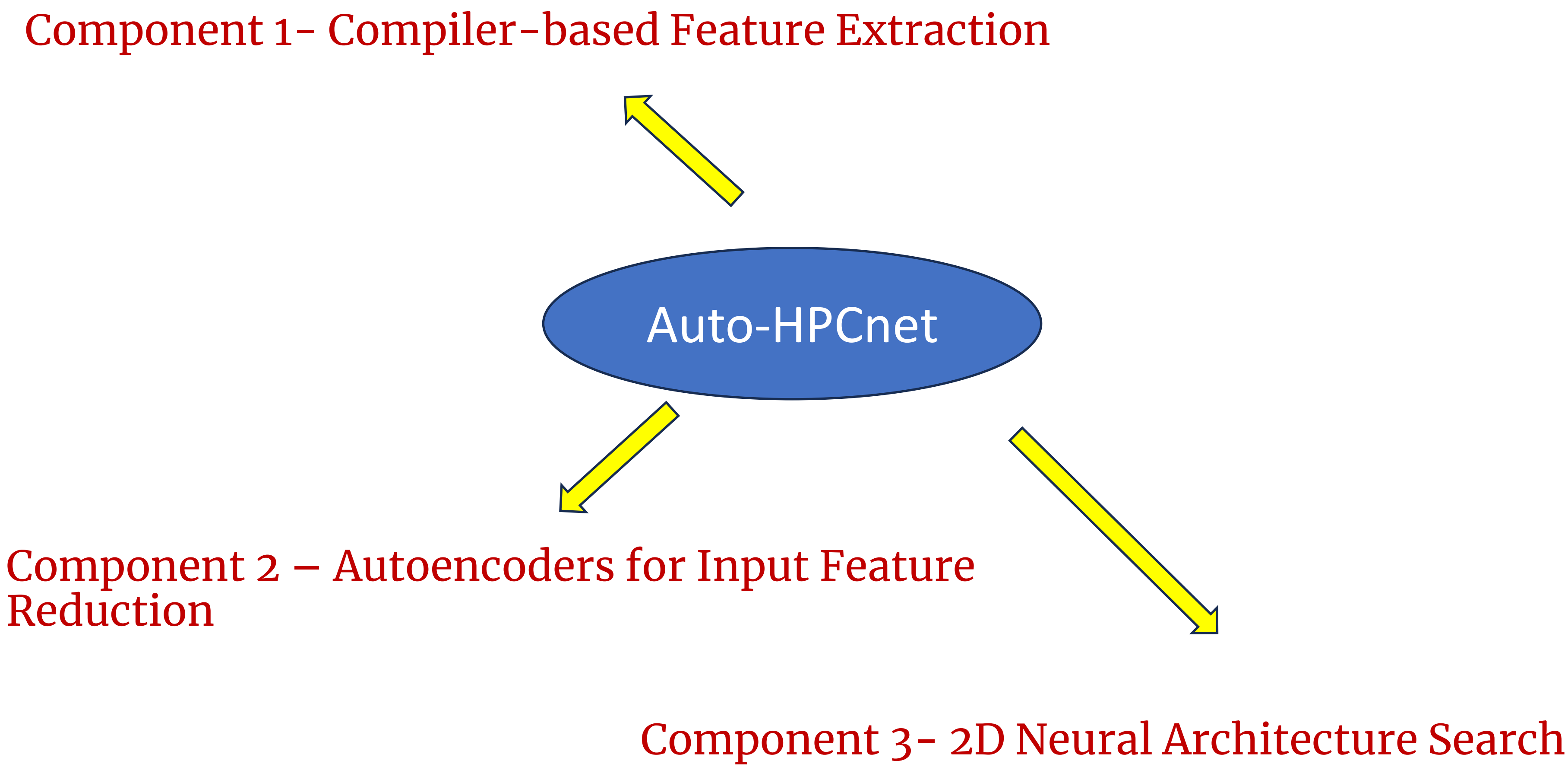
Stages to build NN-based surrogate

Current (problematic) practice

Stages to build NN-based surrogate	Current (problematic) practice
Identifying and collecting input/output features	• Manual efforts • Feature redundancy
NN model construction	• Lack of *coordination* between feature reduction and NN model construction
Repeatedly explore the usage of NN-based surrogate	• Low usability • Almost the whole workflow is manual



Democratize the usage of NN-based surrogate

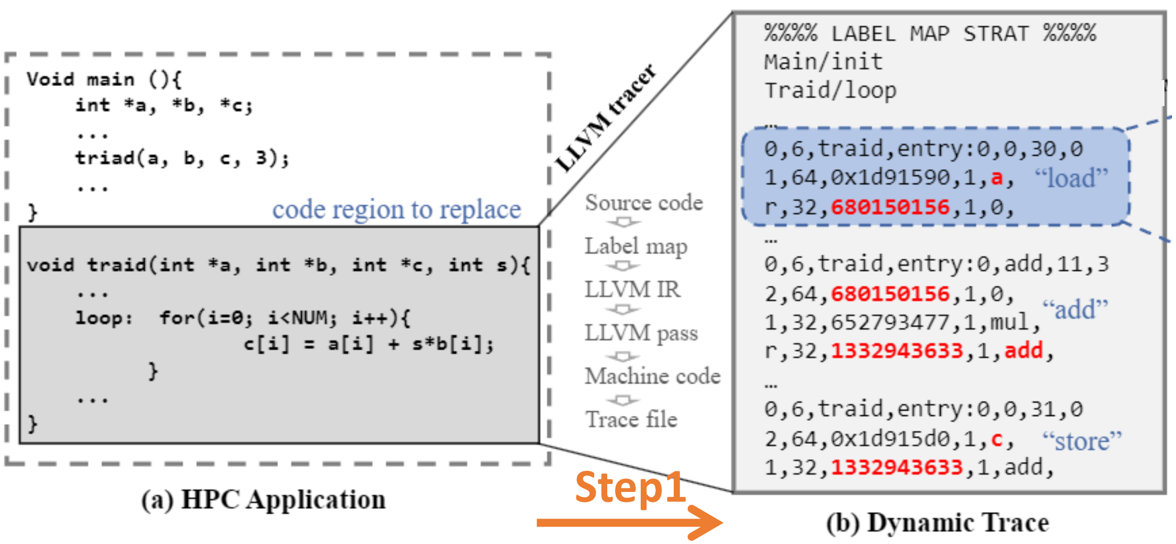

Component 1- Compiler-based Feature Extraction
Auto-HPCnet
Component 2 – Autoencoders for Input Feature Reduction
Component 3- 2D Neural Architecture Search

Component 1- Compiler-based feature extraction

Identify the input/output features of NN surrogates automatically

Step1: Trace generation

- Use LLVM-Tracer to generate a dynamic LLVM instruction trace



(a) HPC Application

(b) Dynamic Trace

An example of acquiring input and output variables

Component 1- Compiler-based feature extraction

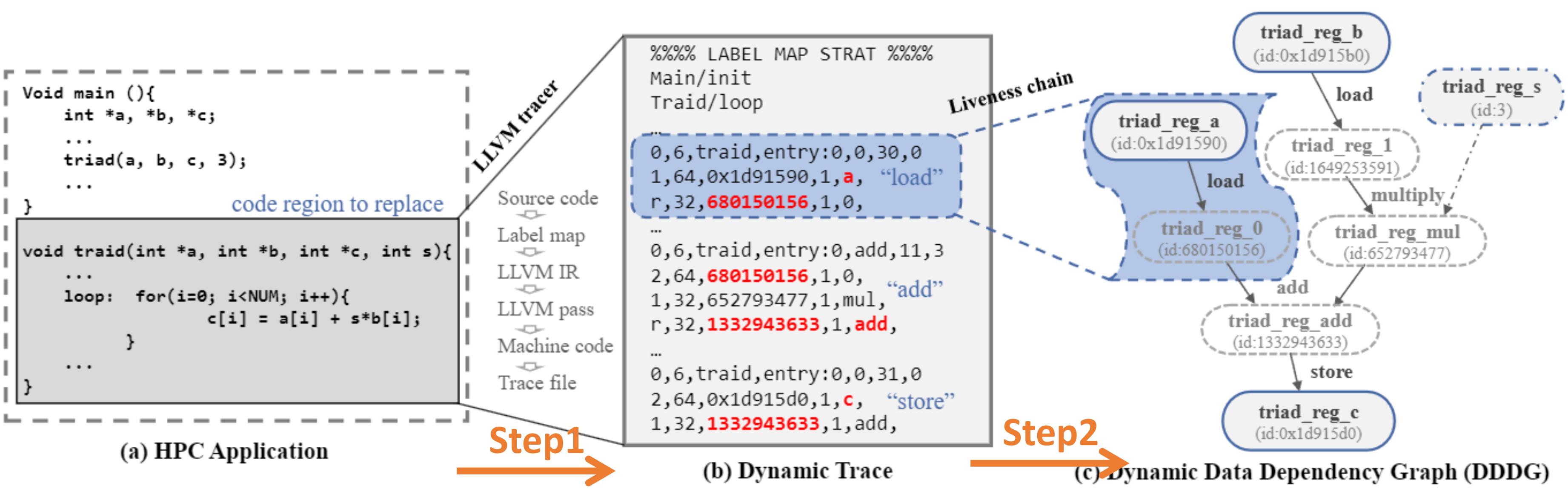
Identify the input/output features of NN surrogates automatically

Step1: Trace generation

- Use LLVM-Tracer to generate a dynamic LLVM instruction trace

Step2: Identification of input and output variables

- Generate dynamic data dependency graph (DDDG) to identify input (leaf of DDDG) and output (root of DDDG) features



An example of acquiring input and output variables

Component 1- Compiler-based feature extraction

Identify the input/output features of NN surrogates automatically

Step1: Trace generation

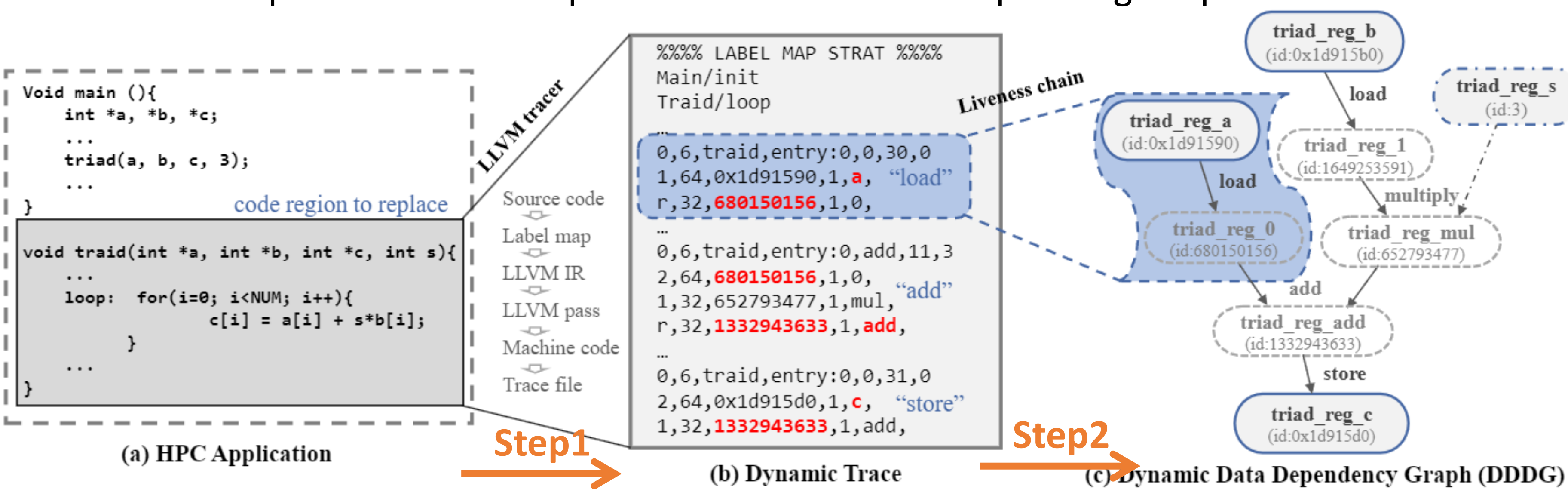
- Use LLVM-Tracer to generate a dynamic LLVM instruction trace

Step2: Identification of input and output variables

- Generate dynamic data dependency graph (DDDG) to identify input (leaf of DDDG) and output (root of DDDG) features

Step3: Generating Training Samples

- Introduce perturbation to input and collect the corresponding output results



An example of acquiring input and output variables

Component 2- Autoencoders to process input features

Handle input sparsity and reduce input-feature redundancy

Input features from HPC applications (sparse matrix)

- ❌ Limit support of sparse matrix formats (COO, CSR, or CRS) in current ML frameworks
- ❌ Unfolding introduces computation inefficiency and storage inefficiency

Row	Col	data
1	2	**2**
2	5	**5**
3	3	**7**
4	1	**3**
5	5	**1**
...	...	...

Input Matrix (COO)

Unfold

Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	...
	2				...
				5	...
		7			...
3					...
				1	...
...	...	...	...	...	⋱

Dense Representation

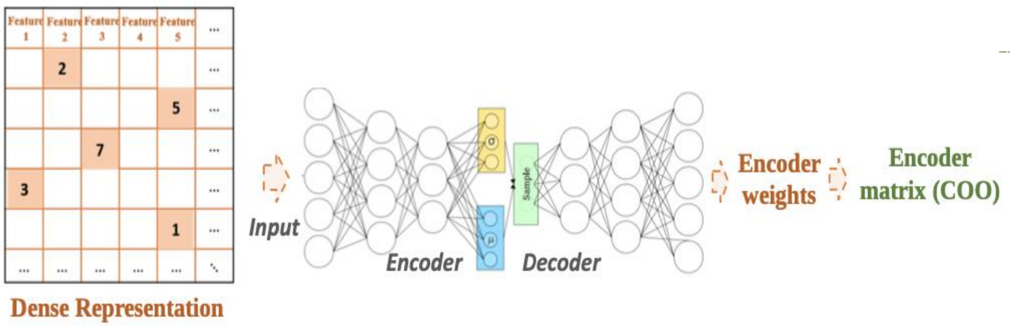
Component 2- Autoencoders to process input features

- **Autoencoder:** reduce redundancy in input features
- **Embedding API**: matrix multiplication $A_{sparse} * B_{sparse} = C_{dense}$

Component 2- Autoencoders to process input features

- **Autoencoder:** reduce redundancy in input features
- **Embedding API**: matrix multiplication $A_{sparse} * B_{sparse} = C_{dense}$

Offline (training autoencoder)
1) Take dense representation as input
2) Generate the Encoder matrix

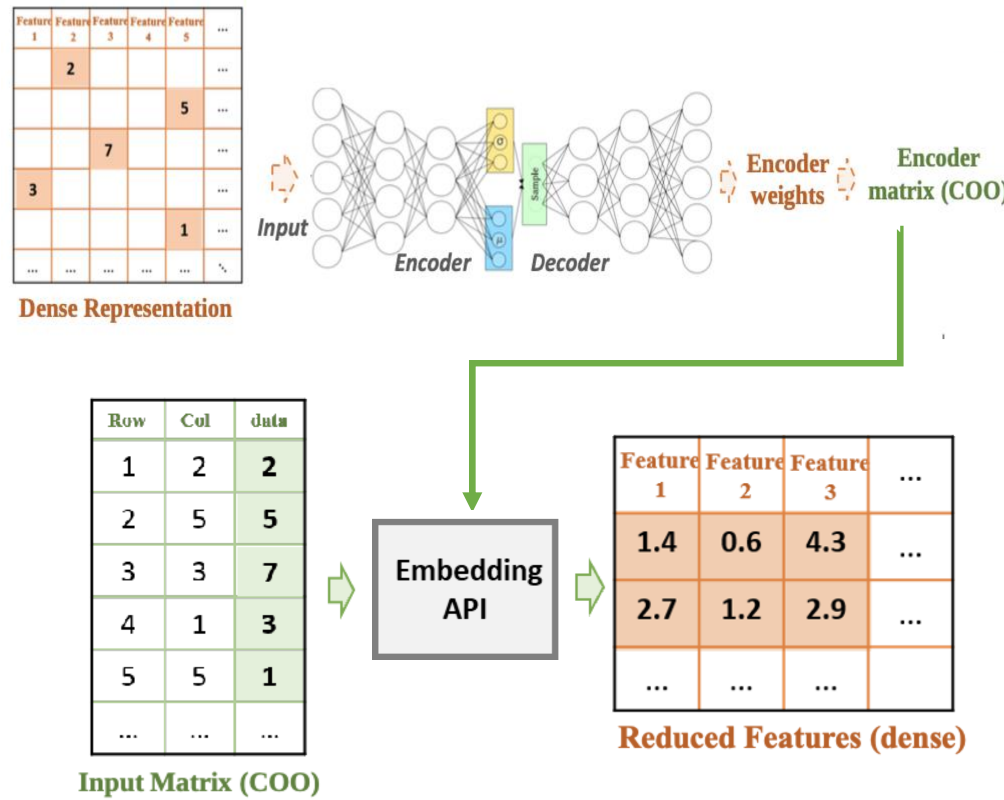


Component 2- Autoencoders to process input features

- **Autoencoder:** reduce redundancy in input features
- **Embedding API**: matrix multiplication $A_{sparse} * B_{sparse} = C_{dense}$

Offline (training autoencoder)
1) Take dense representation as input
2) Generate the Encoder matrix

Online
1) Directly take sparse representation
2) Generate the concise input matrix

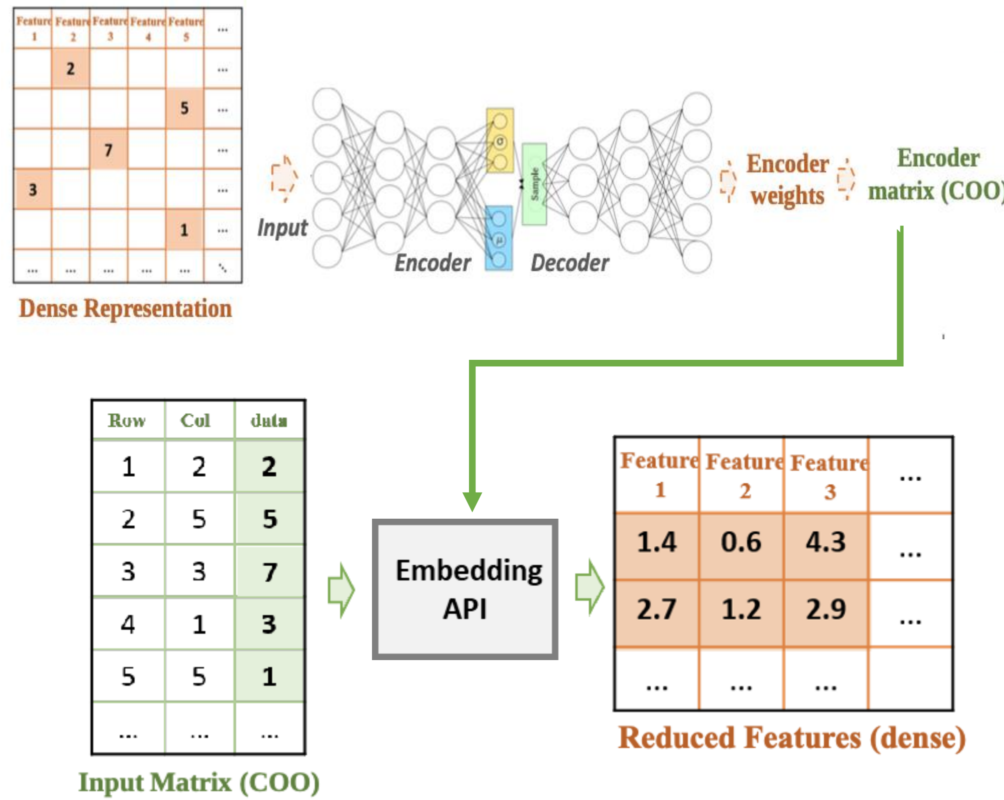


Component 2- Autoencoders to process input features

- **Autoencoder:** reduce redundancy in input features
- **Embedding API**: matrix multiplication $A_{sparse} * B_{sparse} = C_{dense}$

Offline
1) Take dense representation as input
2) Generate the Encoder matrix

Online
1) Directly take sparse representation
2) Generate the concise input matrix



Row	Col	data
1	2	2
2	5	5
3	3	7
4	1	3
5	5	1
...	...	...

Feature 1	Feature 2	Feature 3	...
1.4	0.6	4.3	...
2.7	1.2	2.9	...
...	...	...	

Reduce redundancy and translate sparse format simultaneously

Component 3- 2D neural architecture search

2nd dimension

AutoML

NN topology tuning based on Bayesian optimization (#Layer, #neurons, etc.)

↑

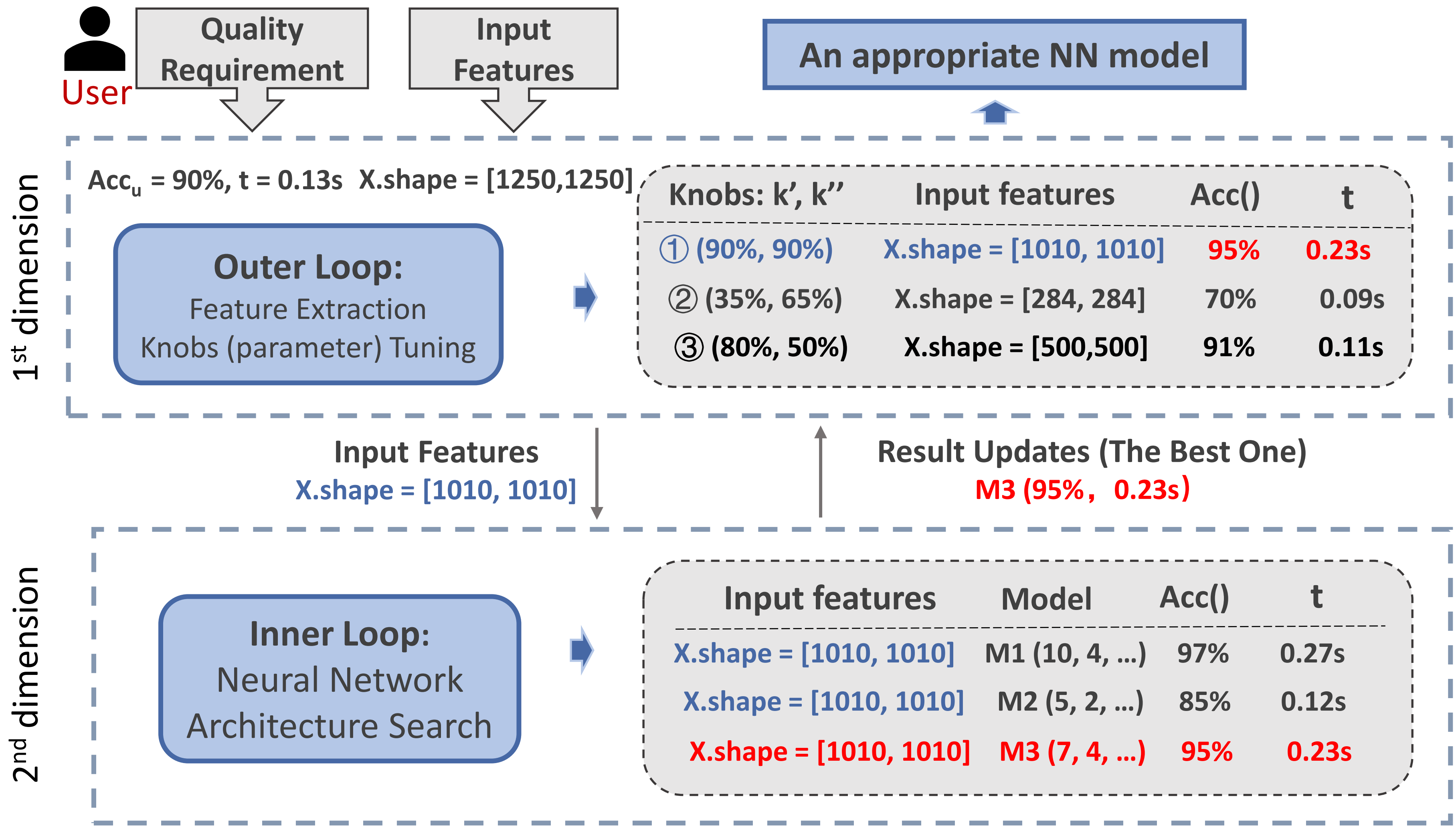
1st dimension

AutoEncoder

Tuning input features

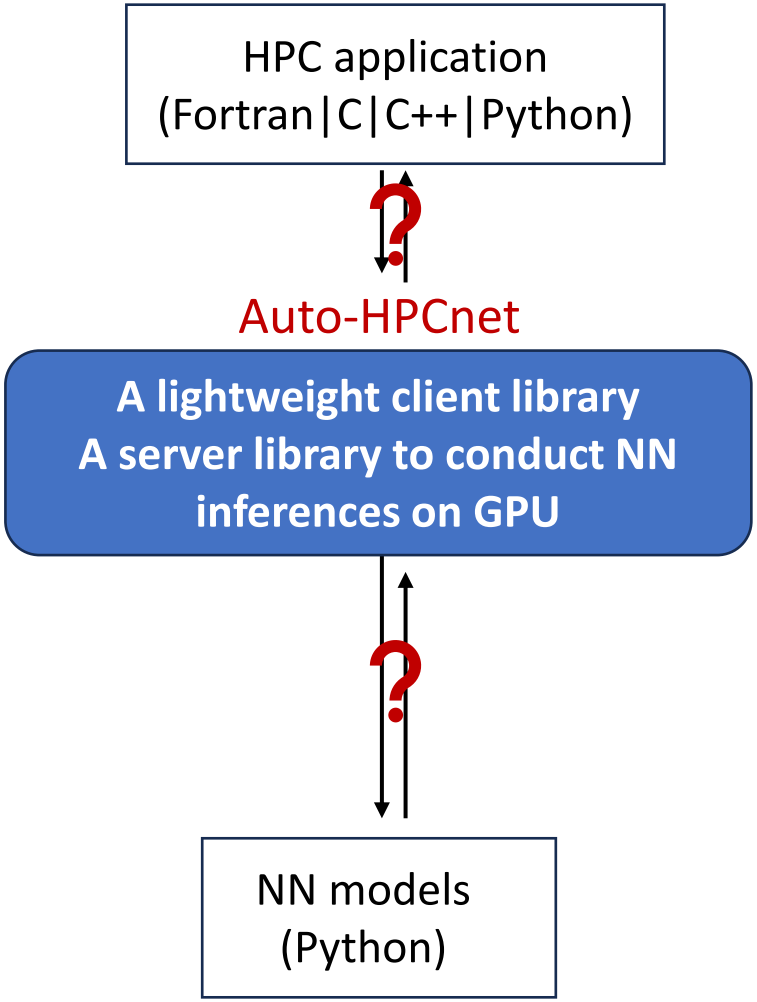
We must consider the impact of input feature reduction during NN model construction

Component 3- 2D neural architecture search

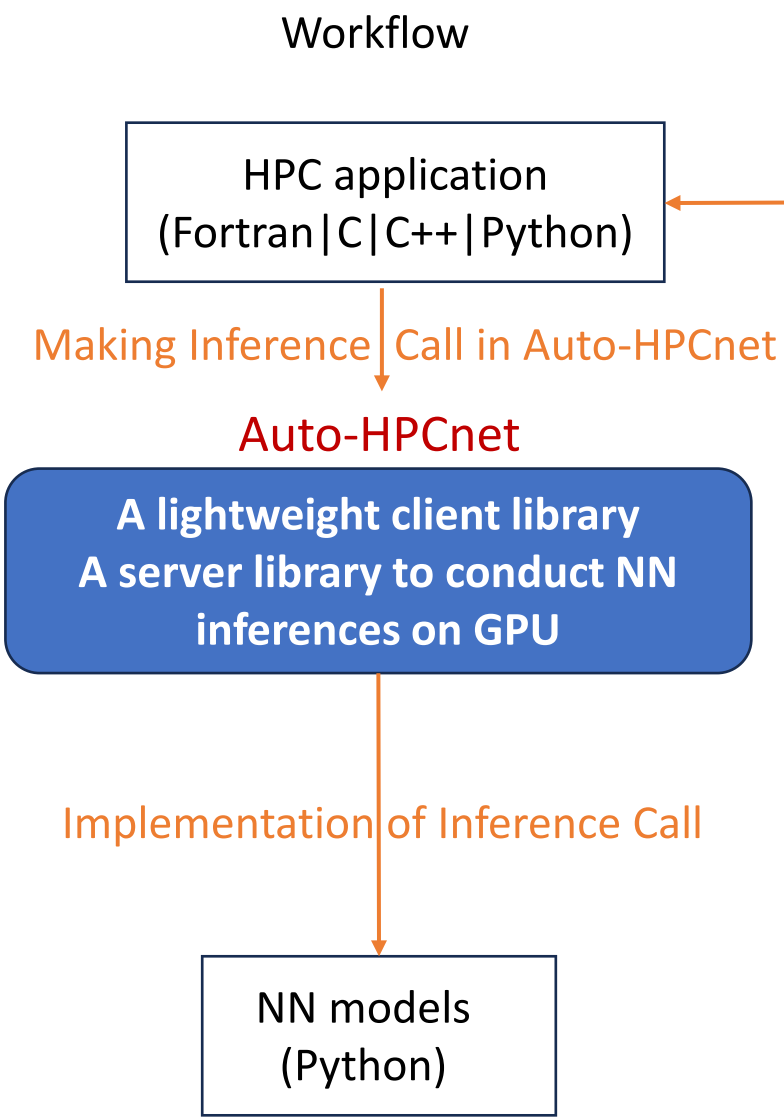


An example of hierarchical Bayesian optimization

Implementation



Implementation



Listing 1: Example of HPC simulation for surrogate request

```
#include "autoHPCnet_client.h"

// Initialize a Client object
autoHPCnet::Client client(false);
// Put the input features on the database
client.put_tensor(in_key, autoHPCnet);
// Run model already in the database
client.run_model("AI-CFD-net", {in_key}, {out_key});

// Get the result of the model
client.unpack_tensor(out_key, autoHPCnet);
```

Listing 2: Example of invoking surrogate model

```
from autoHPCnet import Client
from smartsim.database import Orchestrator
# import other packages …

# create and start a database
orc = Orchestrator(port=REDIS_PORT)
exp.generate(orc)
exp.start(orc, block=False)

# get input from database
sparse_tensor = client.get_tensor(input_feature)

# feature reduction and format transformation
compact_tensor = client.autoencoder(sparse_tensor)

# load a pretrained model from file
client.set_model_from_file("AI-CFD-net", "./saved_net.pt",
"TORCH", "GPU")

# Run model and retrieve outputs
client.run_model("AI-CFD-net", inputs=compact_tensor,
outputs=output_tensor)
```

Evaluation

- **Platform**
  - NVIDIA DGX-1 cluster with 8 nodes, and each node is equipped with two Intel Xeon E5-2698 v4 CPUs and eight NVIDIA TESLA V100 (Volta) GPUs.

- **Applications**

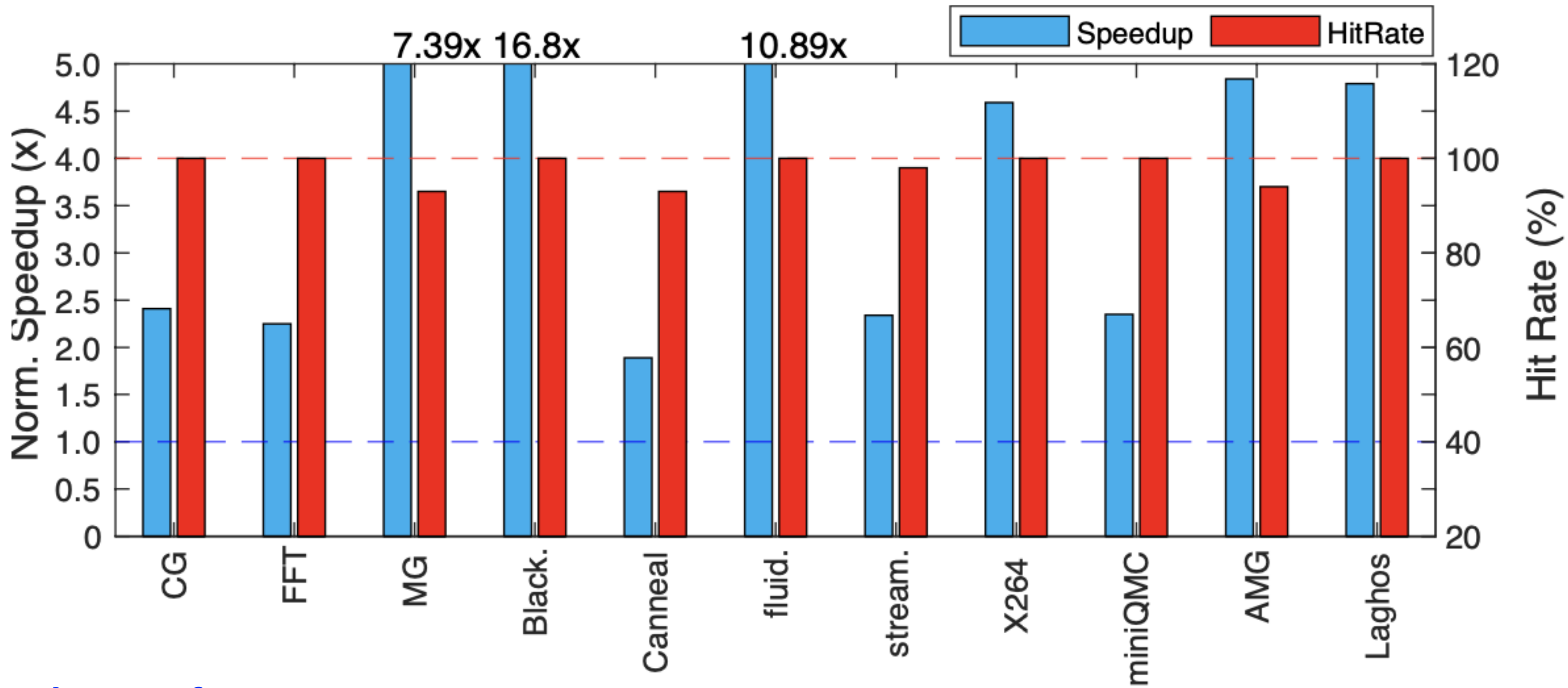
  Type-I: Numerical solvers
  Type-II: the PARSEC parallel benchmark suite
  Type-III: the Exascale Computing Project (ECP) Proxy Applications Suite 4.0.

Type	**Application:** replaced function	Description	Quality of Interest (QoI)
I	**CG**:*CG_solver*	Conjugate Gradient	Solution of linear equations
	FFT:*FFT_solver*	Fast Fourier Transform	Output sequence of FFT
	MG:*MG_solver*	Multi-Grid method	The final residual of the solver
II	**Blackscholes**:*BlkSchlsEqEuroNoDiv*	Investment pricing	The computed price
	Canneal:*Annealing*	VLSI routing	Routing cost
	fluidanimation:*NS_equation*	Fluid dynamics	Particle distance
	streamcluster:*Dimension_reduction*	Online clustering	Cluster center distance
	X264:*Encoding*	Video encoding	Structure similarity
III	**miniQMC:***Determinant*	Quantum Monte Carlo	Particle energy
	AMG:*PCG_solver*	Solver of linear systems	Solution of linear systems
	Laghos: *SolveVelocity*	Compressible gas dynamics	Velocity Divergence

Evaluation

Overall performance

Speedup and prediction HitRate of Auto-HPCnet.



Speedup Performance

Ground truth: the original code;

$$Speedup = \frac{T_{Numerical\_solver}}{T'_{NN\_infer} + T'_{Data\_load} + T_{Other\_part}}$$

HitRate Performance

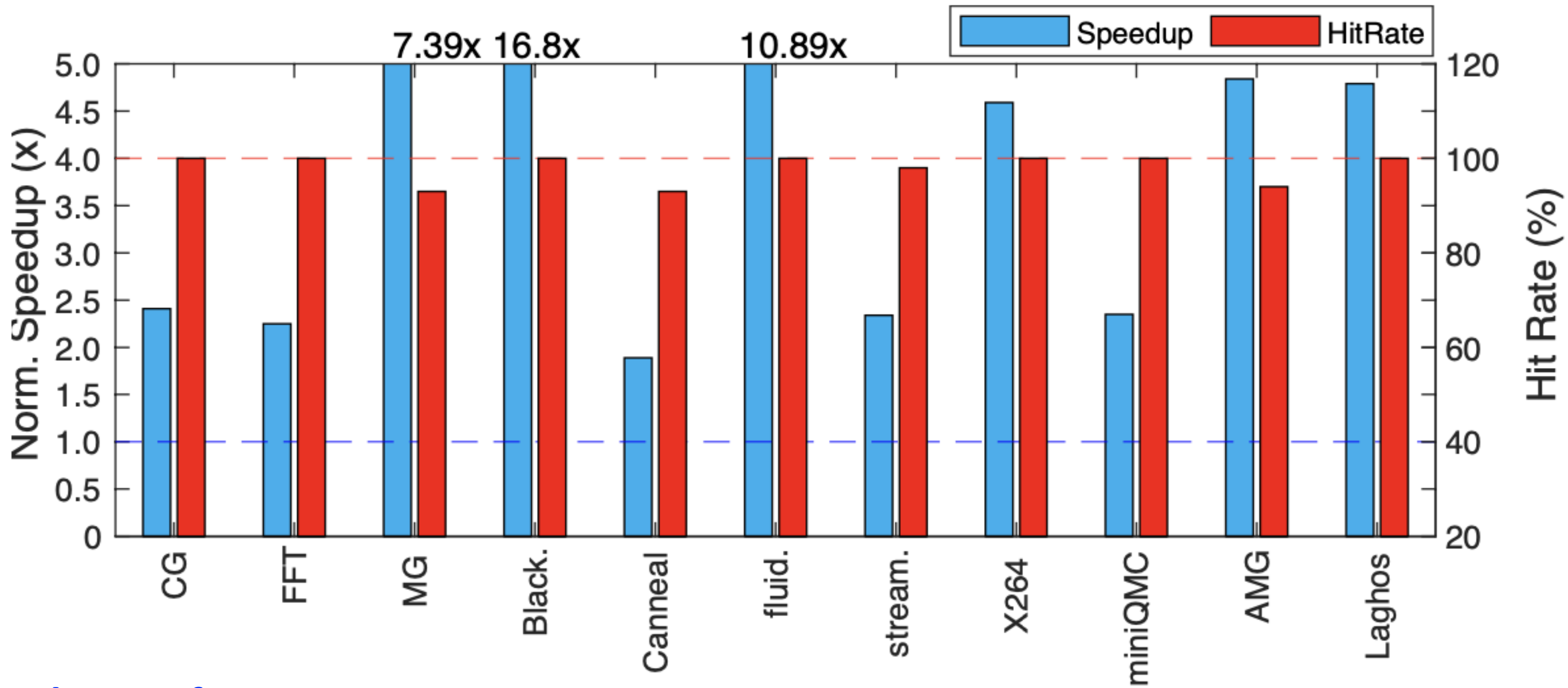
The ratio of successful cases with NN surrogates to the total number of cases;

$$HitRate = \frac{1}{N}\sum_{i=1}^{N}(1,\ \text{if } |V_i' - V_i| \le \mu|V_i|)$$

Evaluation

Overall performance

Speedup and prediction HitRate of Auto-HPCnet.



Speedup Performance

❖ 1.89× - 16.8× speedup with a harmonic mean of 5.50×;

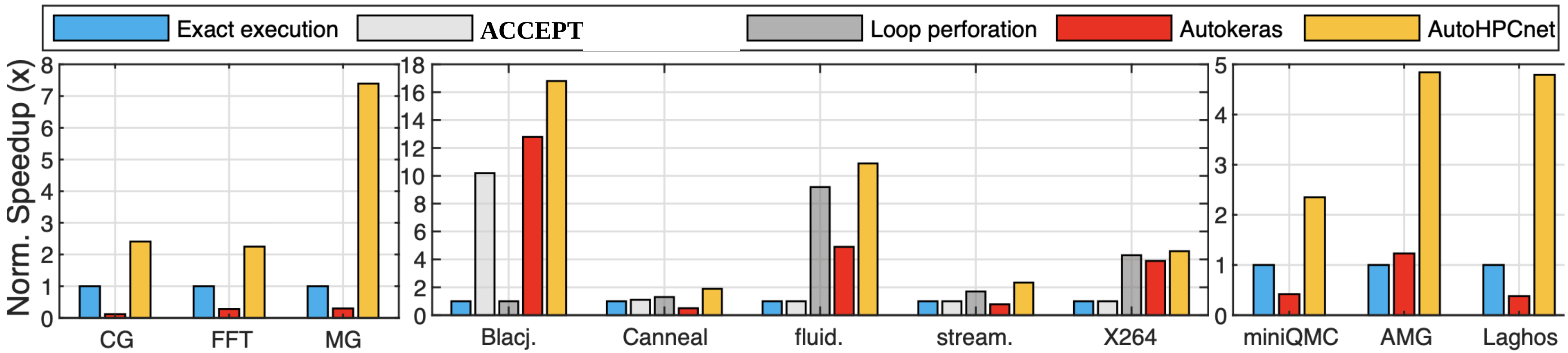
HitRate Performance

❖ Four applications → above 90%;
❖ 100% in other seven applications;

Evaluation

Comparison with the state-of-the art approximation methods

Performance comparison with existing approximation methods



- **ACCEPT :** the state-of-the art framework for NN-bases approximation
- **Loop perforation:** every a couple of iterations skip one iteration
- **Autokeras:** an AutoML framework

❖ Auto-HPCnet outperforms ACCEPT and Loop perforation by more than 40% and 5x on average

❖ Autokeras causes slowdown in applications whose inputs are high-dimensional sparse matrices (CG, FFT, MG, miniQMC, and AMG)

Evaluation

Overhead Analysis

❖ Offline phases

Auto-HPCnet	Time overhead
Component 1: LLVM trace generation, etc	24-59 minutes
Component 2: AutoEncoder training	1.4-2.2 hours
Component 3: 2D Neural Architecture search	6-13 hours

Once the NN model is developed and well-trained, it can be integrated into the HPC applications for repeated use.

❖ Online phases

(1) Fetching input data to GPU memory	21.2%
(2) Encoding input data to low-dimensional features	10.1%
(3) Loading a pre-trained surrogate model	1.6%
(4) Running the surrogate model and retrieving the model output for the application	67.1%

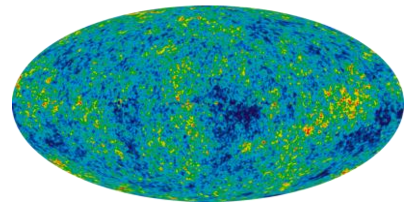
Conclusions

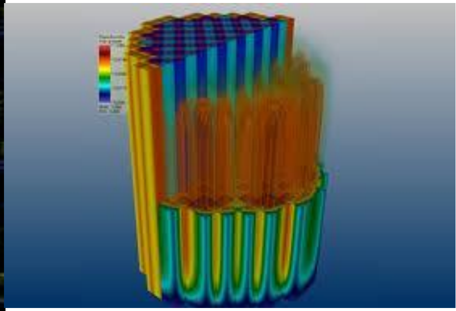
- The NN-based surrogate is powerful to accelerate HPC applications, but is difficult to use

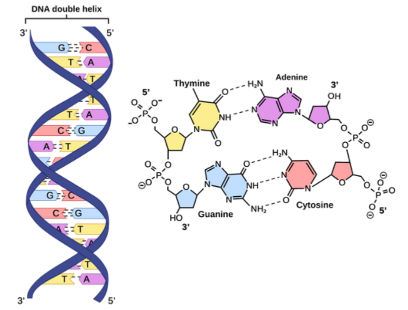
- Auto-HPCnet automates the process of feature identification, performance and application quality control, and NN model construction

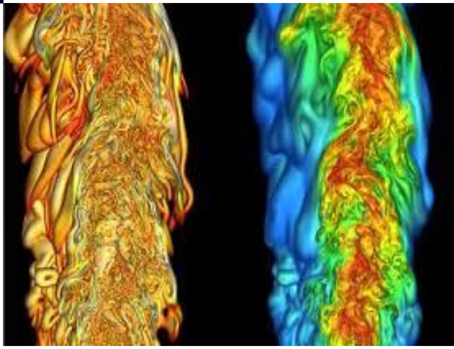
Democratize the usage of NN-based surrogate

Accelerating Scientific Discovery through HPC + AI









Questions?