PerfVec: Generalizable Performance Modeling using Learned Program and Architecture Representations

Lingda Li, Sairam Sri Vatsavai, Kuan-Chieh Hsu

Brookhaven National Laboratory

SIGMETRICS 2025

Stony Brook, NY

Agenda

- PerfVec overview (30 mins)
- Performance prediction demo/hands-on (30 mins)
- Design space exploration (15 mins)
- Source code (5 mins)
- Open discussion (10 mins)

- Repository: https://github.com/PerfVec/PerfVec
  - Source code and other materials used in this tutorial

PerfVec Overview

Performance Modeling for Computer Architecture

- Essential to computer architecture research and engineering
  - New design evaluation, design space exploration, resource scheduling, …
- Goals: speed, accuracy, and generalizability

Methodology	Speed	Accuracy	Generalizability
Analytical Modeling	Fast	Low	High
		High	Low
Discrete Event Simulation	Slow	Variable	High
Emulation	Medium	Variable	Medium
Machine Learning (ML)-based Modeling	Fast	High	Low
ML-based Simulation	Medium	Variable	Medium
PerfVec (this work)	Fast?	High?	**High?**

Goal: explore better tradeoff using ML, especially on generalizability

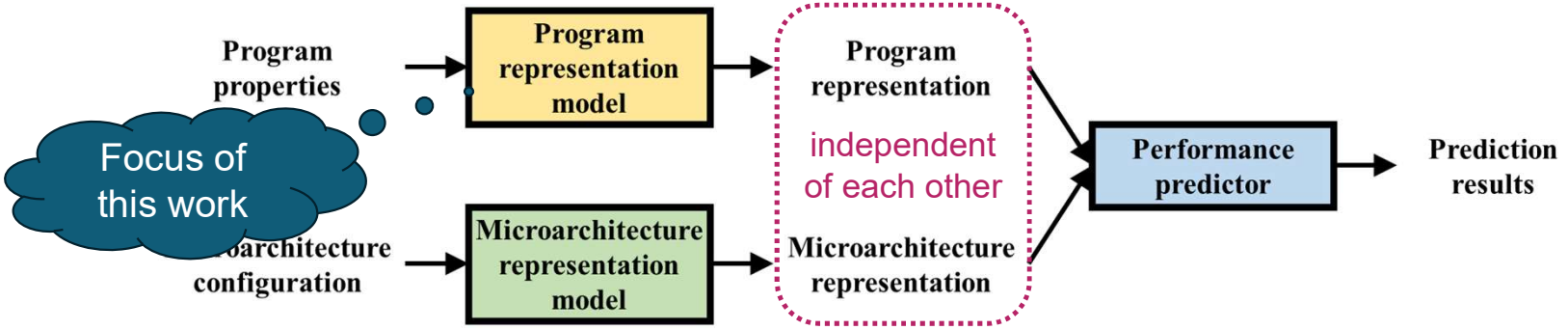
Generalizable Performance Modeling

- Performance is determined by both software and hardware.
- A generic performance model should separate the impact of software (program) and hardware (microarchitecture).
  - When one changes, no need to re-model the other
- Not trivial to achieve such separation
  - Complex interplay between program and microarchitecture
  - Several analytical models tried to do it manually.

PerfVec **learns** the separation automatically.

Learning Separation

Key idea 1: have two independent ML models to capture the performance impacts of program and microarchitecture, respectively



- The same program representation is used to predict its performance on any microarchitecture, and vice versa.

Learning Program Representation

Input	Level of Detail	Limitation
Profiling info (e.g., performance counters)	Low	• Often microarchitecture dependent • Lack of low level details
Static program info (e.g., control flow graph)	Medium	• Cannot capture dynamic execution (e.g., input impact) • Huge graphs to learn from
Instruction execution trace	High	• Huge amounts of instructions to learn from

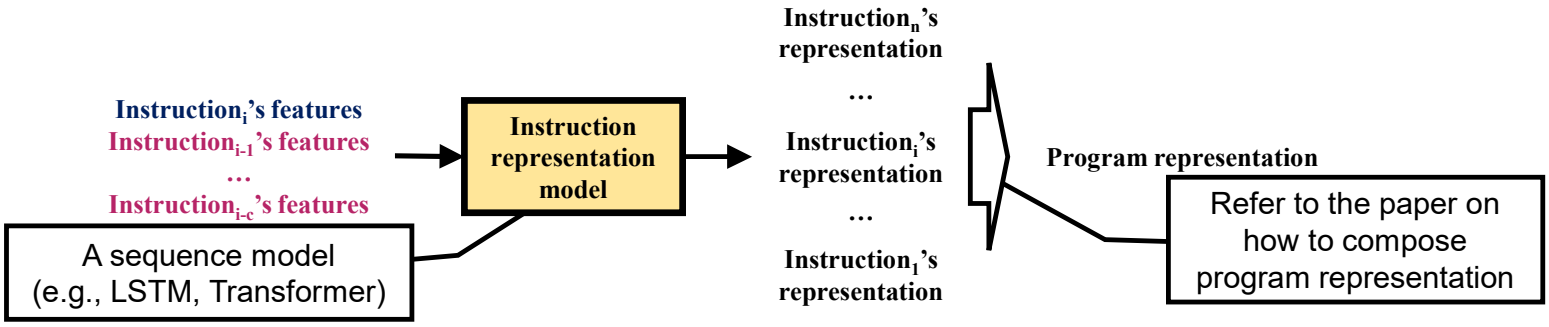
- Challenge: programs typical execute at least billions of instructions, and no ML model can deal with such long sequences.

Key idea 2: 1) learn the representations of individual instructions and 2) compose a program representation from those of all its executed instructions (divide-and-conquer)

Learning Instruction Representation

- Performance determination factors of an instruction

Factor	Microarchitecture Independent Feature
Own properties	Static properties (e.g., operation type); dynamic behavior (e.g., branch direction); reuse distance; branch entropy
Co-running instructions	Co-running instructions' properties

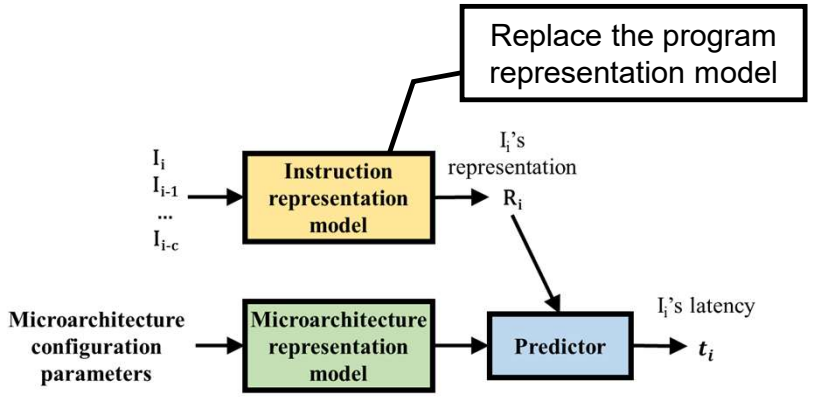


Generalizable across programs

Program traces are different sequences of instructions from the same set.

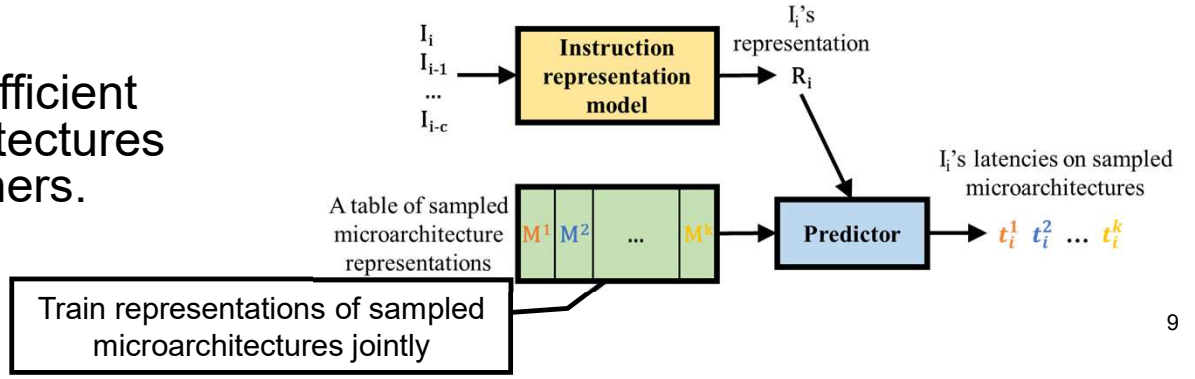
PerfVec Training

- Natural solution: training all models end-to-end
- Challenge
  - Irregular and alterable microarchitectural space
  - Difficult to train a universal microarchitecture representation model



Key idea 3: train the instruction representation model to predict instruction latencies on randomly selected microarchitectures for generalizability

- Hypothesis: training with a sufficient number of diverse microarchitectures enables generalizability to others.



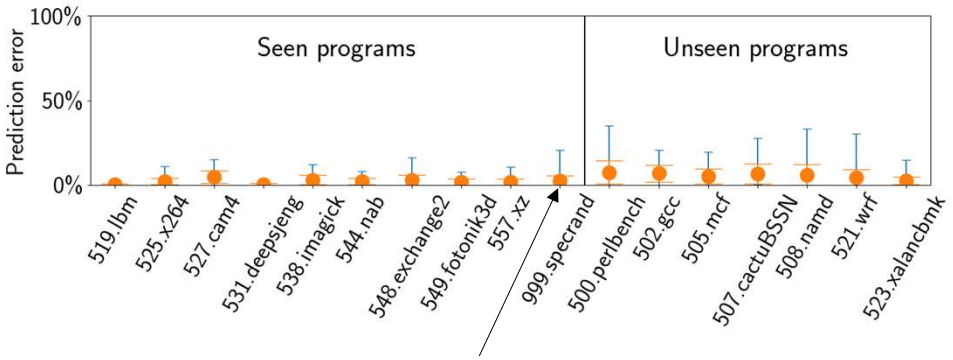
Brookhaven National Laboratory

Data Acquisition & Model Architecture

- Instruction traces from gem5 simulation
  - Easy to obtain instruction level latency
  - Easy to configure microarchitectures
- Programs
  - 17 SPEC CPU2017 benchmarks
  - 10 for training, 7 for testing
- Microarchitectures
  - Randomly generated gem5 configurations
    - In-order/out-of-order cores, caches, memory, etc.
  - 77 for training, 10 for testing
- Model architecture
  - 2-layer LSTM by default
  - See the paper for other architectures

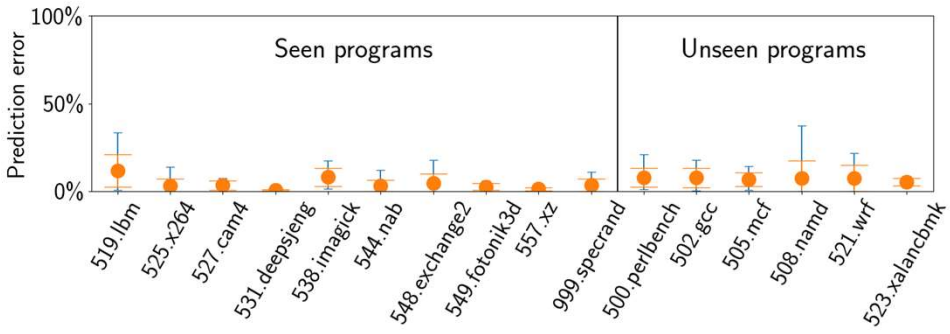
Generalizability Evaluation

Unseen programs



Prediction error range against gem5 simulation across all microarchitectures

Unseen microarchitectures



The trained model generalizes well to unseen programs and microarchitectures.

Brookhaven National Laboratory

PerfVec Summary

- High generalizability
  - Learn independent program and microarchitecture representations
- Good accuracy
  - Learn program representations from instruction execution traces
- Fast speed
  - Simple combination of program and microarchitecture representations
- Many potential applications
  - Design space exploration, performance analysis, …
- Code: https://github.com/PerfVec/PerfVec

Performance Prediction Demo/Hands-on

Agenda

- Demo: cross-compile a program
- Demo: gem5 simulation and trace generation
- Hands-on: trace processing and prediction

Demo: Cross-compile a program

- Input: Helloworld program written in C
- Cross compiler: https://github.com/riscv-collab/riscv-gnu-toolchain
- Platforms:
  - On Linux environment
  - With Dockerfile
- Output: helloworld executable in RISC-V architecture.

Demo: gem5 simulation & trace generation

- Input: executable
- Simulator: gem5 https://github.com/lingda-li/gem5/tree/riscv
- Output: trace.txt and trace.sq.txt

Hands-on: trace processing & prediction

- Google Colab notebook
  - https://colab.research.google.com/drive/1ViJtzsbbUFXkSEYsdgne9iJsfCZq0UvJ#scrollTo=IS-7Ugikt7CI

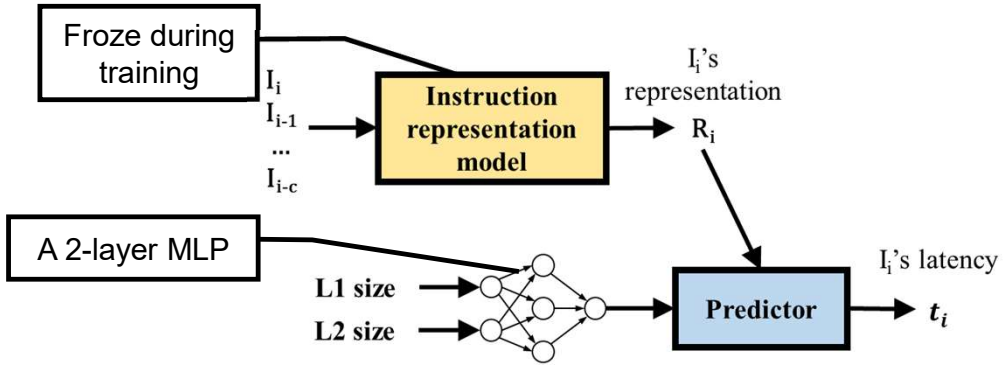
Design Space Exploration

PerfVec Use Cases

- Performance modeling is essential for many tasks.
  - PerfVec can be used in them.
- A case study: design space exploration (DSE)
  - Find the optimal design(s) given one/multiple objective function(s)
- DSE example: L1 and L2 cache size exploration
  - Objective function: execution_time * (1000 + 10*L1_size + L2_size)
    - Similar to latency area product
  - Select the best cache sizes for 17 SPEC CPU2017 benchmarks

DSE Procedure

- Train a microarchitecture representation model



- Training data: gem5 simulation traces of 3 benchmarks on selected configurations
- Predict the performance of all benchmarks
  - Use the trained microarchitecture representation model and existing program representations

DSE Results

- Complete gem5 simulation: 600 hours
- Previous ML-based DSE methods
  - Use selected simulation results to train program-specific models
  - Need to simulate many configurations for each program
- PerfVec
  - Incur significantly less overhead with comparable quality
  - 11 hours = 5 (data collection) + 6 (training)

Lower is better

Method	Overhead	Quality
ASPLOS06	150	4.4%
MICRO07	84	4.7%
DAC16	170	3.6%
PerfVec	11	3.6%

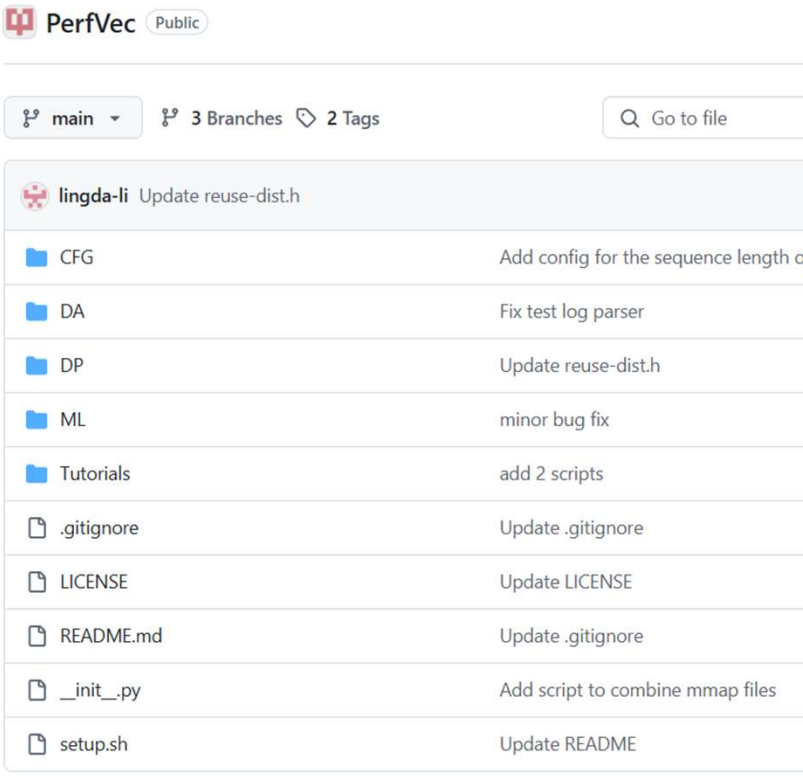
Overhead: the time to construct models (hours)
Quality: how close the selected design is to the optimal design.

1. Ïpek et al. Efficiently exploring architectural design spaces via predictive modeling. ASPLOS 2006.
2. Dubach et al. Microarchitectural design space exploration using an architecture-centric approach. MICRO 2007.
3. Li et al. Efficient design space exploration via statistical sampling and adaboost learning. DAC 2016.

Source Code

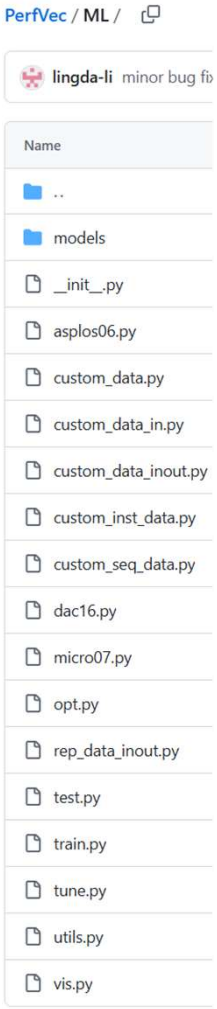
Top Level Folders

- DP: data processing scripts for simulation traces
  - Implemented using Python (NumPy) and C++
- ML: scripts for training, test, models, etc.
  - Implemented using PyTorch
- CFG: configuration files for datasets
- Tutorials: descriptive instructions and documents
- DA: data analysis scripts



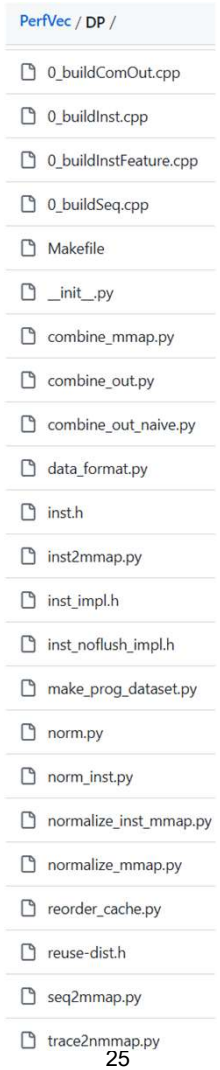
ML Folder

- models: neural network architectures
  - LSTM, Transformer, CNN, MLP, …
- train.py: train instruction representation models
- test.py: test prediction accuracy of trained models
- tune.py: fine-tune microarchitecture representation models and representations
- custom_data*.py: data loaders for PerfVec datasets



DP Folder

- gem5 trace processing
  - 0_buildInstFeature.cpp: numerical instruction feature generation
  - inst2mmap.py, trace2nmmap.py: convert a textual numerical trace to numpy memory mapping format for PyTorch input
  - normalize_*.py: dataset normalization
- Training/test data generation
  - 0_buildComOut.cpp, combine_out.py: generate prediction labels



Open Discussion

Questions?

Comments?

Collaboration opportunities?