Machine Programming

Justin Gottschlich, Intel Labs

December 12th, 2018

TVM Conference, University of Washington

Motivation

We have a software programmer resource problem

Motivation

We have a software programmer resource problem

Search **Bloomberg** Sign In

Technology

Demand for Programmers Hits Full Boil as U.S. Job Market Simmers

By Craig Torres
March 7, 2018, 9:00 PM PST

https://www.bloomberg.com/news/articles/2018-03-08/demand-for-programmers-hits-full-boil-as-u-s-job-market-simmers

Motivation

We have a software programmer resource problem

Search **Bloomberg** Sign In

Technology

Demand for Programmers Hits Full Boil as U.S. Job Market Simmers

By Craig Torres
March 7, 2018, 9:00 PM PST

*2019 human population **7,714M***

*2019 developers **26.4M***

*% of programmers: **> 0.34% <***

Motivation

We have a software programmer resource problem

Search **Bloomberg** Sign In

Technology

Demand for Programmers Hits Full Boil as U.S. Job Market Simmers

By Craig Torres
March 7, 2018, 9:00 PM PST

*2019 human population **7,714M***

*2019 developers **26.4M***

*% of programmers: **> 0.34% <***

*2019 human population **7,714M***

*2019 developers **1,200M***

*% of drivers: **> 15.56% <***

Motivation

*2019 human population **7,714M***

*2019 developers **26.4M***

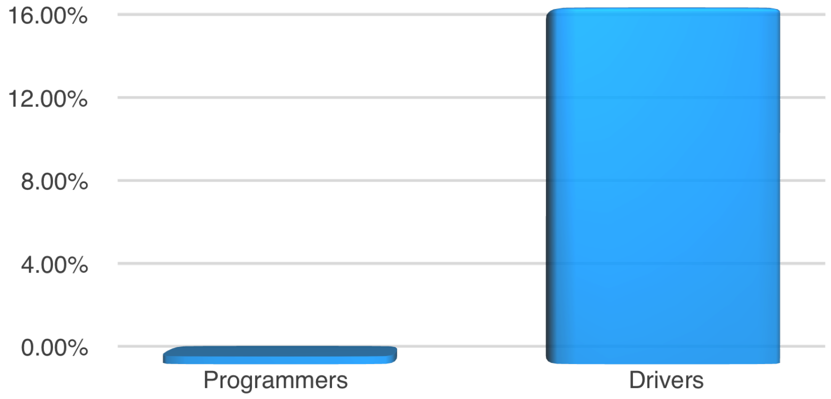
*% of programmers: > **0.34% <***

*2019 human population **7,714M***

*2019 developers **1,200M***

*% of drivers: > **15.56% <***

Programmers vs. Drivers (Population)



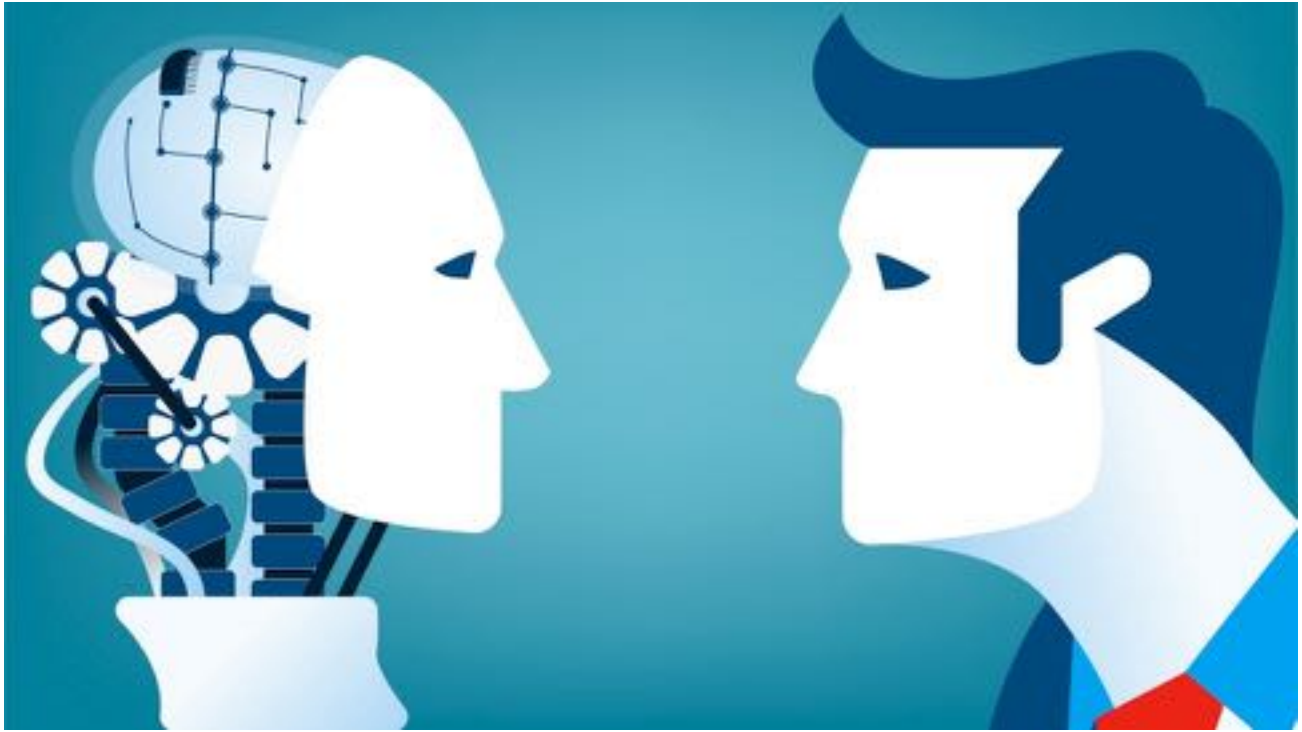
Motivation

What if programming could be as simple as driving?

How can we simplify programming (mostly with machine learning)?

(1) Reduce intention-challenge, (2) delegate most work to machines.

Human programming vs machine programming



Human Programming

The process of developing software, principally by one or more humans.

- **Examples**
  - Writing code in <*your favorite language here*>
- **Pros**
  - Near complete control over the software created, exact behaviors
- **Cons**
  - Expensive, slow, error-prone, human-resource limited

Machine Programming

The process of developing software where some or all of the steps are performed autonomously.

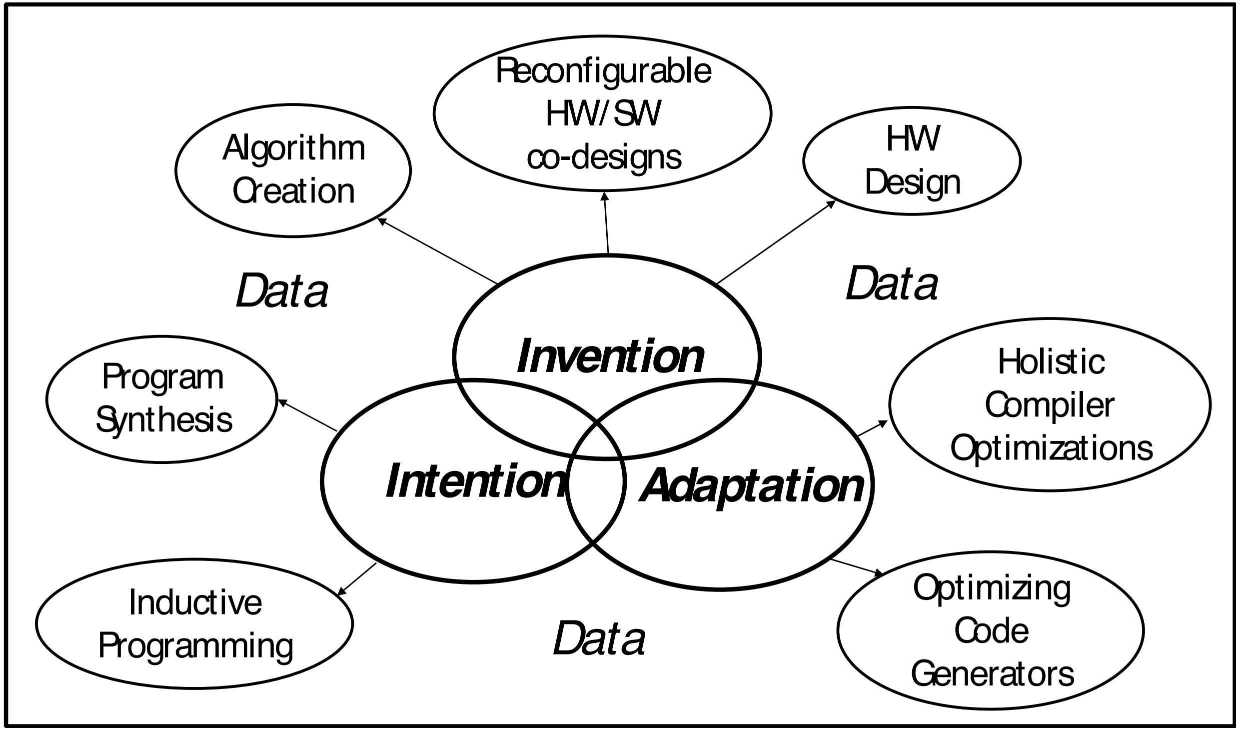
- **Examples**
  - *Classical:* compiler transformations
  - *Emerging:* Verified lifting[1], AutoTVM[2], Sketch[3], DeepCoder[4], SapFix/Sapienz[5]
- **Pros**
  - Resource constrained by computers, most humans can create software
- **Cons**
  - Immature, may lack full control, may be partially stochastic

[1] http://www.cs.technion.ac.il/~shachari/dl/pldi2016.pdf [2] https://arxiv.org/pdf/1805.08166.pdf [3] https://people.csail.mit.edu/asolar/papers/thesis.pdf
[4] https://arxiv.org/abs/1611.01989 [5] https://research.fb.com/finding-and-fixing-software-bugs-automatically-with-sapfix-and-sapienz

The Three Pillars of Machine Programming (MP)

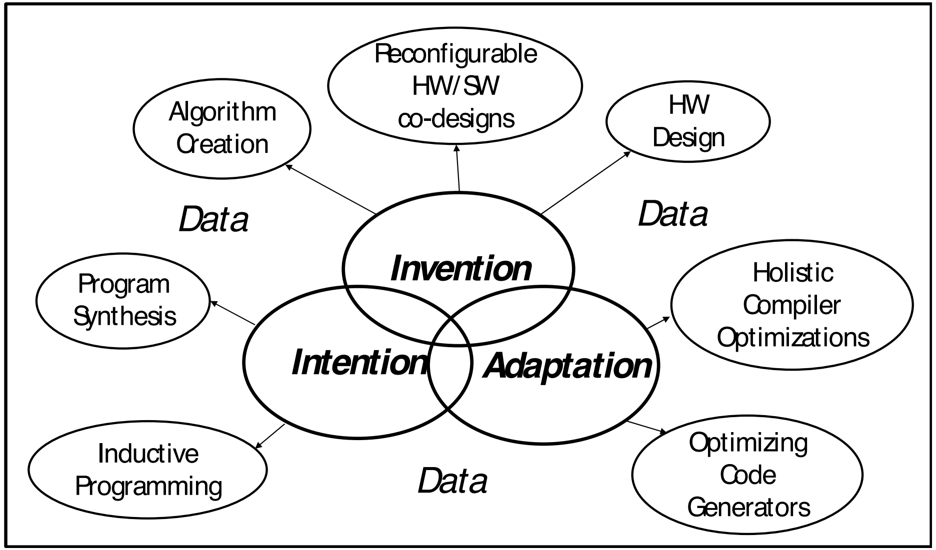
MAPL/PLDI'18

Justin Gottschlich, Intel
Armando Solar-Lezama, MIT
Nesime Tatbul, Intel
Michael Carbin, MIT
Martin, Rinard, MIT
Regina Barzilay, MIT
Saman Amarasinghe, MIT
Joshua B Tenenbaum, MIT
Tim Mattson, Intel



Examples of the Three Pillars of MP

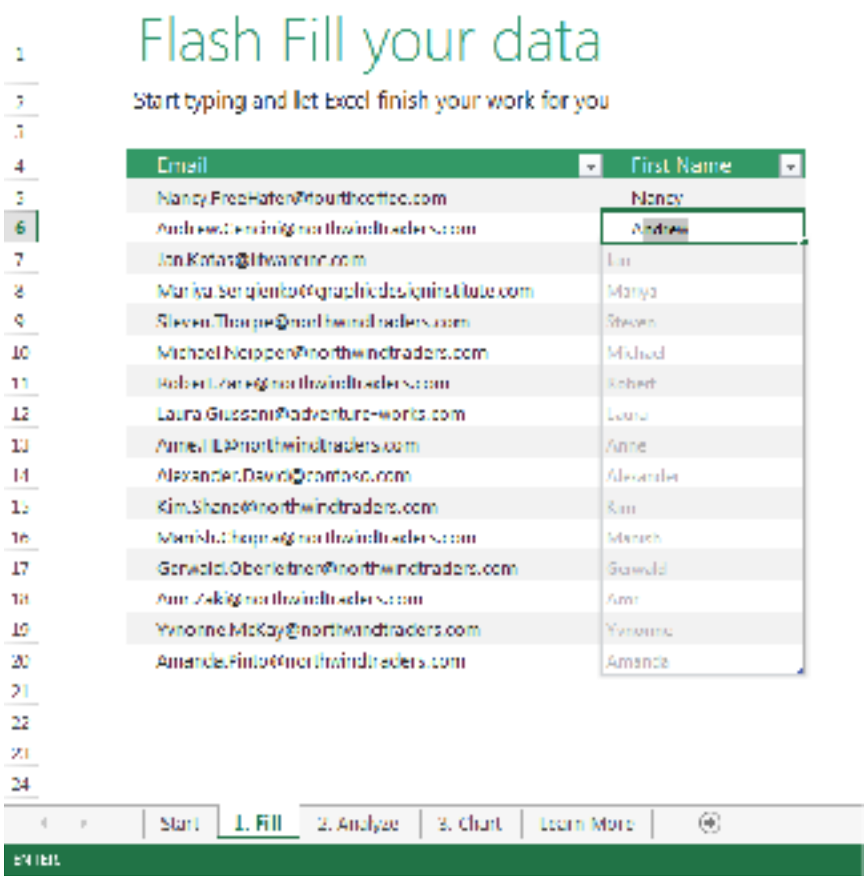
- **Intention**
  - *"Automating String Processing in Spreadsheets using Input-Output Examples"* (Sumit Gulwani)
  - *"Program Synthesis by Sketching"* (Armando Solar-Lezama, Adviser: R. Bodik)
- **Invention**
  - *"The Case for Learned Index Structures"* (Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, Neoklis Polyzotis)
- **Adaptation**
  - *"Precision and Recall for Time Series"* (Nesime Tatbul, TJ Lee, Stan Zdonik, Mejbah Alam, Justin Gottschlich)



- **Adaptation**
  Anomaly Detection Interpretability
  (Xin Sheng, Mejbah Alam, Justin Gottschlich, Armando Solar-Lezama)

Flash Fill

- **Intention**
  - *"Automating String Processing in Spreadsheets using Input-Output Examples"* (Sumit Gulwani)
  - *"Program Synthesis by Sketching"* (Armando Solar-Lezama, Adviser: R. Bodik)
- **Invention**
  - *"The Case for Learned Index Structures"* (Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, Neoklis Polyzotis)
- **Adaptation**
  - *"Precision and Recall for Time Series"* (Nesime Tatbul, TJ Lee, Stan Zdonik, Mejbah Alam, Justin Gottschlich)



Flash Fill (POPL 2011): https://www.microsoft.com/en-us/research/wp-content/uploads/2016/12/popl11-synthesis.pdf

Sketch

- **Intention**
  - *“Automating String Processing in Spreadsheets using Input-Output Examples”* (Sumit Gulwani)
  - *“Program Synthesis by Sketching”* (Armando Solar-Lezama, Adviser: R. Bodik)
- **Invention**
  - *“The Case for Learned Index Structures”* (Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, Neoklis Polyzotis)
- **Adaptation**
  - *“Precision and Recall for Time Series”* (Nesime Tatbul, TJ Lee, Stan Zdonik, Mejbah Alam, Justin Gottschlich)

```
int[] merge (int[] a, int b[], int n) {
   int j, k;
   for (int i = 0; i < n; i++)
      if (        hole        [] )) {
            result[i] = a[j++];
      } else  {
            result[i] = b[k++];
      }
   }
   return result;
}
```

```
int[] merge (int[] a, int b[], int n) {
   int j, k;
   for (int i = 0; i < n; i++)
      if (j<n && ( !(k<n) || a[j] < b[k])
            result[i] = a[j++];
      } else  {
            result[i] = b[k++];
      }
   }
   return result;
}
```

Learned Index Structures

- **Intention**
  - ***"Automating String Processing in Spreadsheets using Input-Output Examples"*** (Sumit Gulwani)
  - ***"Program Synthesis by Sketching"*** (Armando Solar-Lezama, Adviser: R. Bodik)
- **Invention**
  - ***"The Case for Learned Index Structures"*** (Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, Neoklis Polyzotis)
- **Adaptation**
  - ***"Precision and Recall for Time Series"*** (Nesime Tatbul, TJ Lee, Stan Zdonik, Mejbah Alam, Justin Gottschlich)

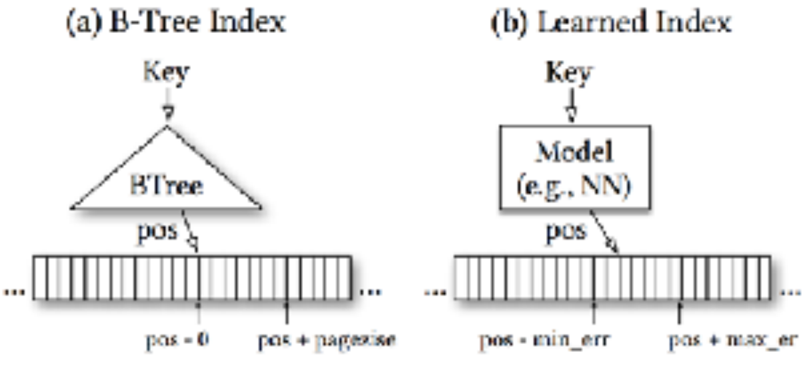


Figure 1: Why B-Trees are models

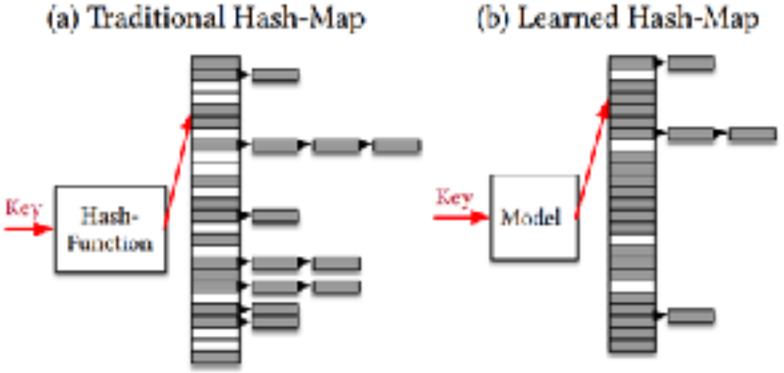
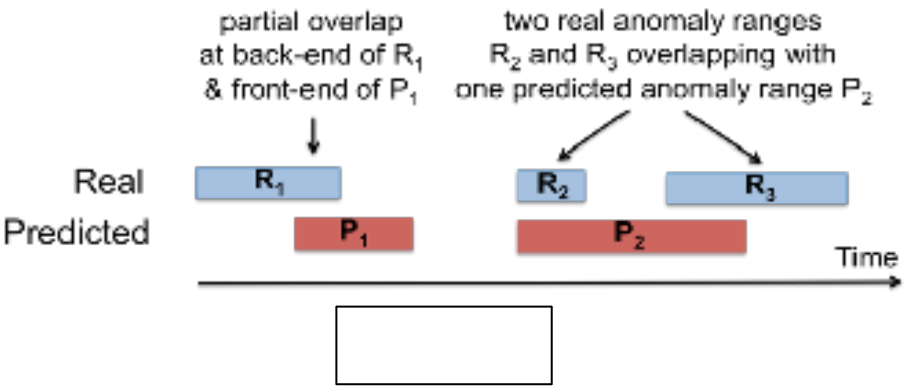


Figure 7: Traditional Hash-map vs Learned Hash-map

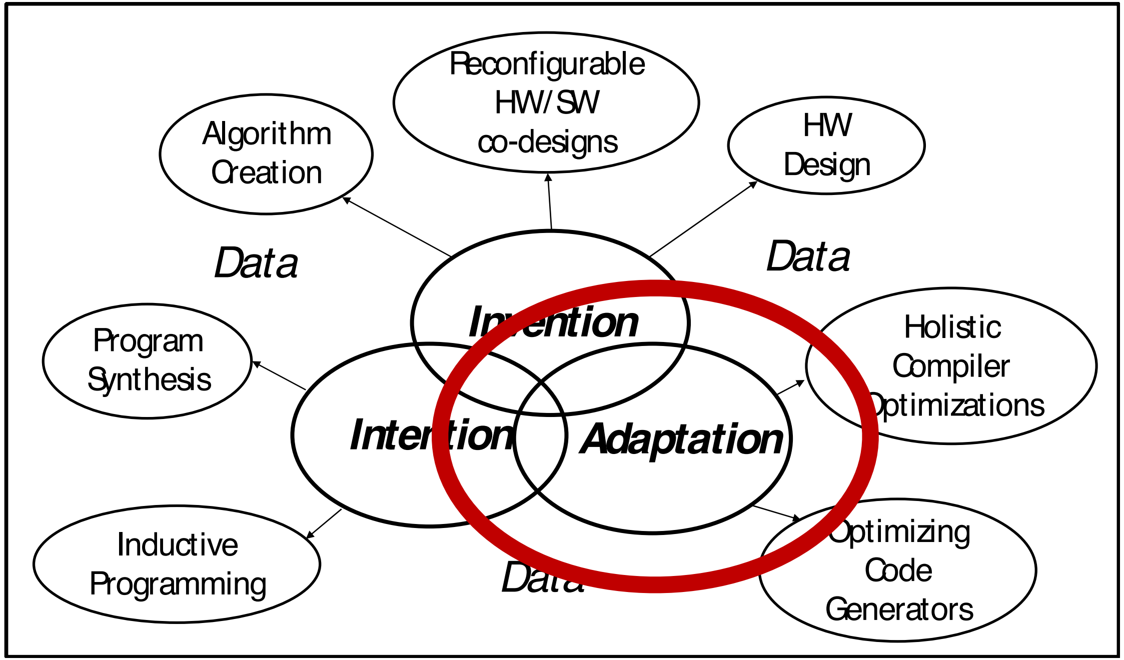
Time Series Anomalies and Interpretability

- Intention
  - *"Automating String Processing in Spreadsheets using Input-Output Examples"* (Sumit Gulwani)
  - *"Program Synthesis by Sketching"* (Armando Solar-Lezama, Adviser: R. Bodik)
- Invention
  - *"The Case for Learned Index Structures"* (Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, Neoklis Polyzotis)
- Adaptation
  - *"Precision and Recall for Time Series"* (Nesime Tatbul, TJ Lee, Stan Zdonik, Mejbah Alam, Justin Gottschlich)

Range-based Anomalies



- **Adaptation**
  Anomaly Detection Interpretability
  (Xin Sheng, Mejbah Alam, Justin Gottschlich, Armando Solar-Lezama)



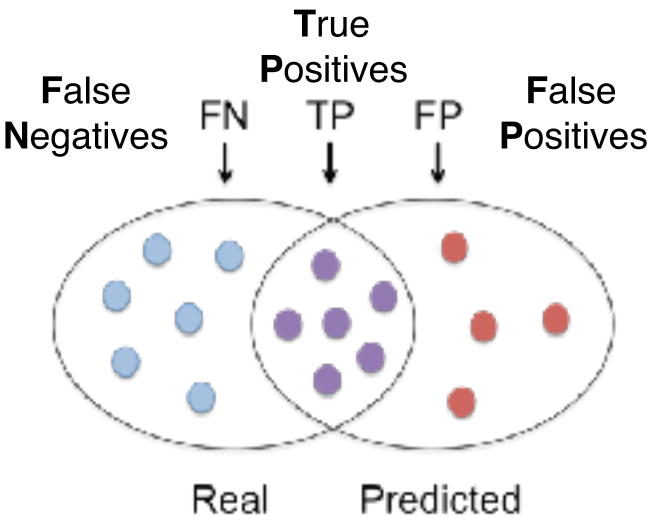
Adaptation

Software that automatically evolves (e.g., repairs, optimizes, secures) itself

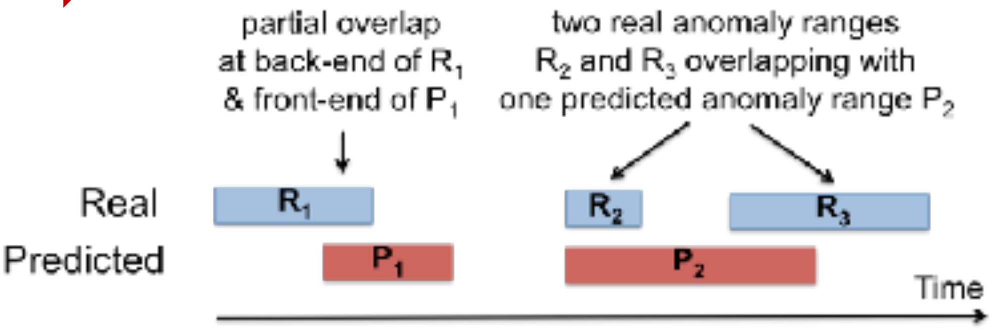
Adaptation is principally about range-based anomaly detection

Time Series Anomaly Detection

Point-based Anomalies



Range-based Anomalies



- How do we define TPs, TNs, FPs, FNs?

(Prior) State of the Art

- **Classical recall/precision**
  - ***Point-based anomalies***
  - Recall penalizes FN, precision penalizes FP
  - F_β-measure to combine & weight them

$$F_\beta = (1+\beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}$$

β : relative importance of Recall to Precision
$\beta = 1$: evenly weighted (harmonic mean)
$\beta = 2$: weights Recall higher (i.e., no FN!)
$\beta = 0.5$: weights Precision higher (i.e., no FP!)

- **Numenta Anomaly Benchmark (NAB)'s Scoring Model [1]**
  - ***Point-based anomalies***
  - Focuses specifically on early detection use cases
  - Difficult to use in practice (irregularities, ambiguities, magic numbers) [2]
- **Activity recognition metrics**
  - No support for flexible time bias

[1] Lavin and Ahmad, "Evaluating Real-Time Anomaly Detection Algorithms – The Numenta Anomaly Benchmark", IEEE ICMLA, 2015.
[2] Singh and Olinsky, "Demystifying Numenta Anomaly Benchmark", IEEE IJCNN, 2017.

(Prior) State of the Art

- **Classical recall/precision**
  - ***Point-based anomalies***
  - Recall penalizes FN, precision penalizes FP
  - F_β-measure to combine & weight them

$$F_\beta = (1+\beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}$$

β : relative importance of Recall to Precision
$\beta = 1$: evenly weighted (harmonic mean)
$\beta = 2$: weights Recall higher (i.e., no FN!)
$\beta = 0.5$: weights Precision higher (i.e., no FP!)

- **Numenta Anomaly Benchmark (NAB)'s Scoring Model [1]**
  - ***Point-based anomalies***
  - Focuses specifically on early detection use cases
  - Difficult to use in practice (irregularities, ambiguities, magic numbers) [2]
- **Activity recognition metrics**
  - No support for flexible time bias

A new accuracy model is needed

[1] Lavin and Ahmad, "Evaluating Real-Time Anomaly Detection Algorithms – The Numenta Anomaly Benchmark", IEEE ICMLA, 2015.
[2] Singh and Olinsky, "Demystifying Numenta Anomaly Benchmark", IEEE IJCNN, 2017.

New Evaluation Model

Expressive, Flexible, Extensible

- **Superset of:**
  - Classical model
  - Other state-of-the-art evaluators (NAB)
- ***NeurIPS '18 Spotlight***
- ***Key: evaluate anomaly detectors with practical meaningfulness***

Precision and Recall for Time Series

Nesime Tatbul — Intel Labs and MIT

Tae Jun Lee — Microsoft

Stan Zdonik — Brown University

Mejbah Alam — Intel Labs

Justin Gottschlich — Intel Labs

Abstract

1 Introduction

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

Precision & Recall for Time Series

Notation	Description
R, R_i	set of real anomaly ranges, the i^{th} real anomaly range
P, P_j	set of predicted anomaly ranges, the j^{th} predicted anomaly range
N, N_r, N_p	number of all points, number of real anomaly ranges, number of predicted anomaly ranges
α	relative weight of existence reward
$\gamma()$, $\omega()$, $\delta()$	overlap cardinality function, overlap size function, positional bias function

Customizable weights & functions

Range-based Recall

$$Recall_T(R,P) = \frac{\sum_{i=1}^{N_r} Recall_T(R_i,P)}{N_r}$$

$$Recall_T(R_i,P) = \alpha \times ExistenceReward(R_i,P) + (1-\alpha) \times OverlapReward(R_i,P)$$

$$ExistenceReward(R_i,P) = \begin{cases} 1, \text{ if } \sum_{j=1}^{N_p} |R_i \cap P_j| \geq 1 \\ 0, \text{ otherwise} \end{cases}$$

$$OverlapReward(R_i,P) = CardinalityFactor(R_i,P) \times \sum_{j=1}^{N_p} \omega(R_i, R_i \cap P_j, \delta)$$

$$CardinalityFactor(R_i,P) = \begin{cases} 1 & \text{, if } R_i \text{ overlaps with at most one } P_j \in P \\ \gamma(R_i,P), & \text{otherwise} \end{cases}$$

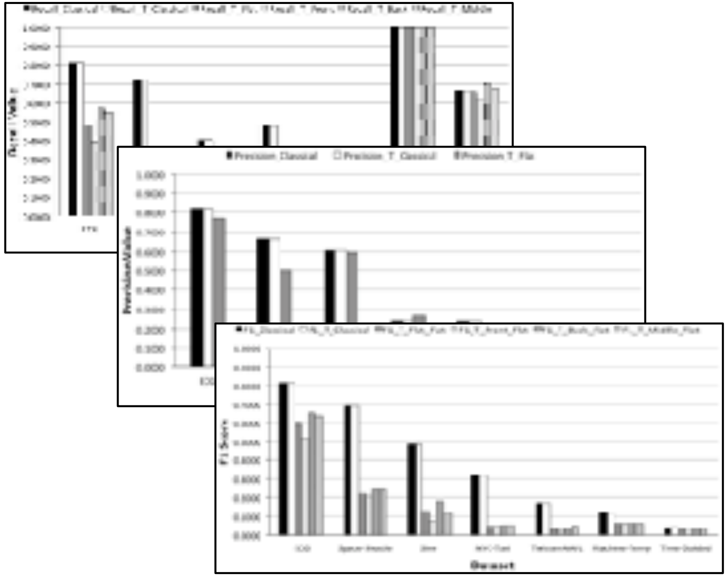
Range-based Precision

$$Precision_T(R,P) = \frac{\sum_{i=1}^{N_p} Precision_T(R,P_i)}{N_p}$$

$$Precision_T(R,P_i) = CardinalityFactor(P_i,R) \times \sum_{j=1}^{N_r} \omega(P_i, P_i \cap R_j, \delta)$$

TSAD-Evaluator Overview

- A tool that implements our customizable evaluation model
- Can be used in two modes:
  - -c: compute classical metrics (point-based)
  - -t: compute time series metrics (range-based)
- Input:
  - 2 files with anomaly labels (e.g., simple.real, simple.pred)
  - Evaluator parameters
- Output:
  - Precision, Recall, F-Score
- A library of pre-defined choices for γ() and δ()
  + templates for user-defined extensions
- Example:



```
./evaluate -t simple.real simple.pred 1 0 reciprocal flat front
```

New Evaluation Model – Helps Intel

- **Positioned to benefit Intel internally**
  - **Cyber-security, data centers, *SW/HW vulnerabilities***

Anomaly Detection Interpretability

Analysis of a anomaly:

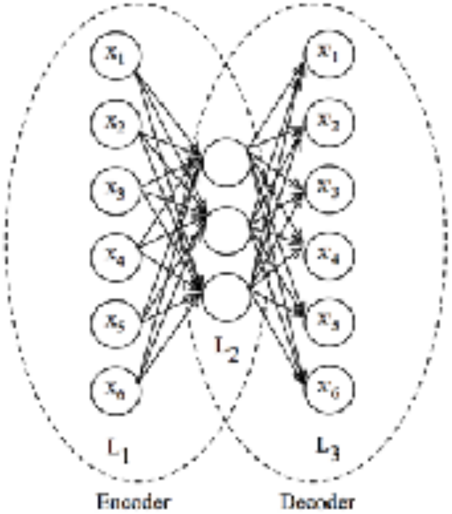
1. Where/when is the anomaly?
   – Existing work can achieve this
2. Why is this an anomaly?
   – Partial solutions in this space
3. *How to fix the anomaly?*
   – Mostly an open problem

The "*How*" and "*Why*" are open questions for *anomaly detection* & *neural networks*

AutoPerf: *ZPL using Autoencoder*

Used to detect parallel software performance anomalies

- Encodes input data to a reduced dimension (encoder)
- Reconstructs input data as target of the network (decoder)
- Reconstruction error :



Anomalous data cannot be reconstructed using representation learned from non-anomalous data

Train autoencoder using non-anomalous dataset → Determine a **threshold** for anomaly using reconstruction error → Inferencing for new inputs → Reconstruction error > threshold?

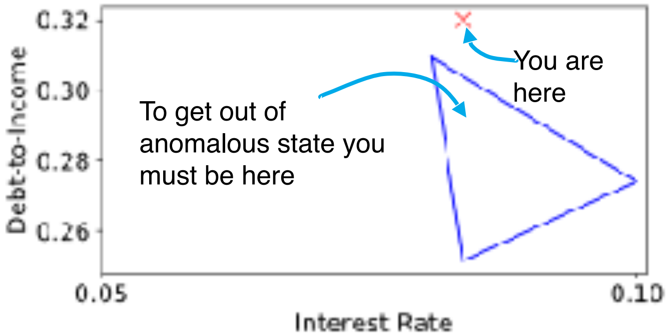
- Yes → Anomaly detected
- No → Non-anomalous

Interpreting Neural Network Judgments

Polaris : Corrections as Explanations for Neural Network Judgment. [2]

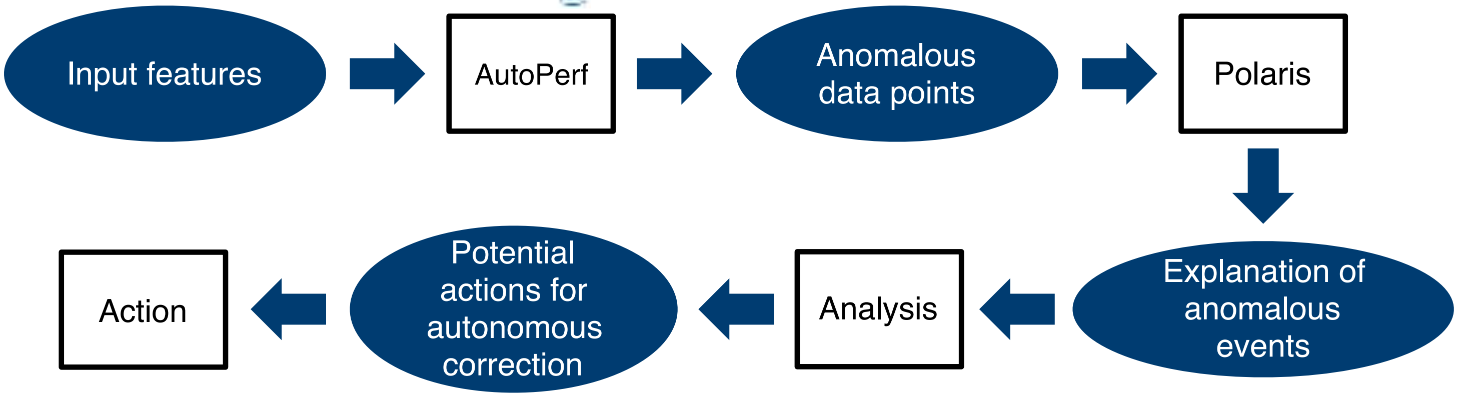
- **Judgment problem:** binary classification problem where one output is preferred
  - Vehicle collision, software performance and correct bugs, security vulnerabilities
- **Proposed solution:** corrections as actionable explanations.

- **Desired properties:**
  - Minimal
  - Stable
  - Symbolic



[2] **Interpreting Neural Network Judgments via Minimal, Stable, and Symbolic Corrections,**
Xin Zhang (MIT), Armando Solar-Lezama (MIT), Rishabh Singh (Google Brain), [**NIPS ’18** (to appear)]

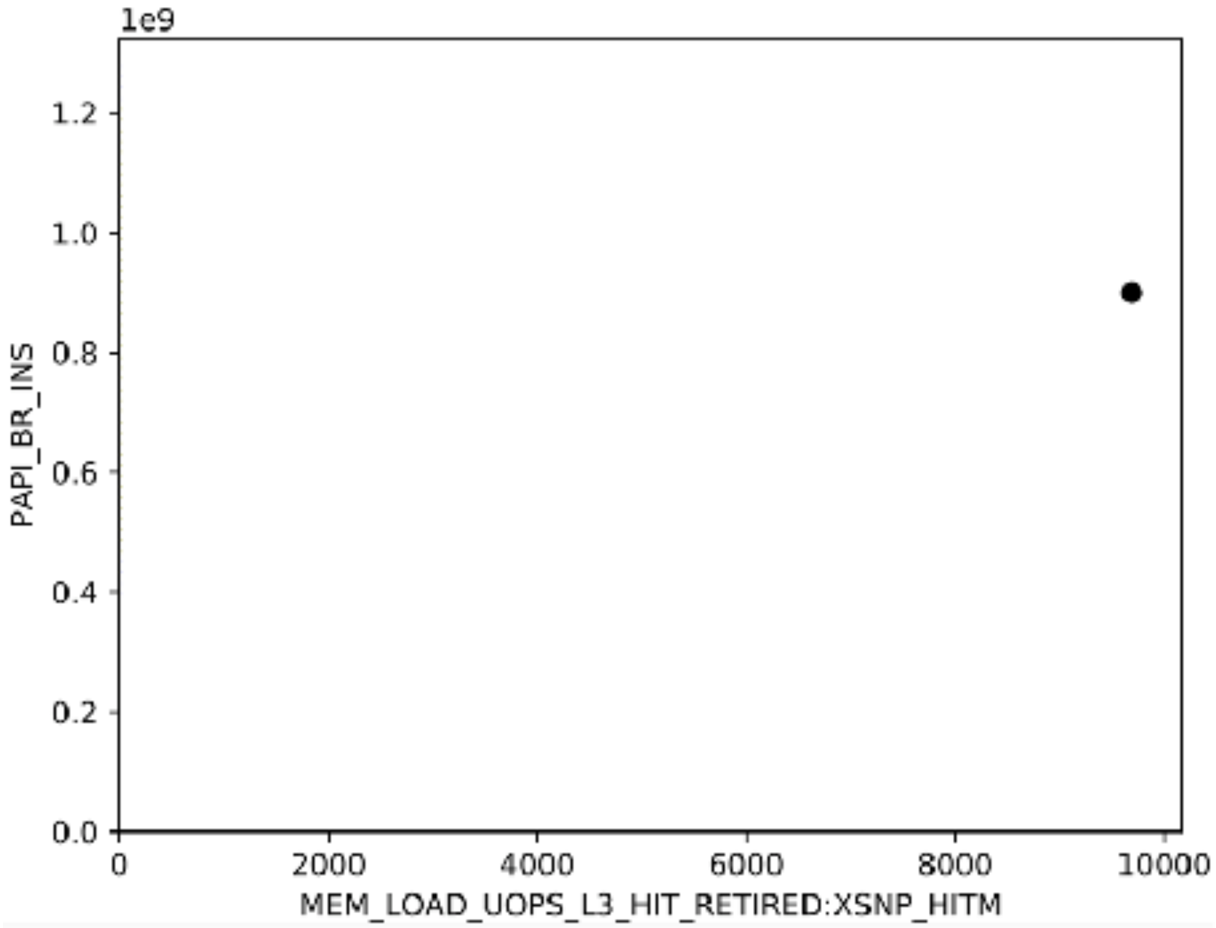
IL+MIT: Interpreting AutoPerf using Polaris



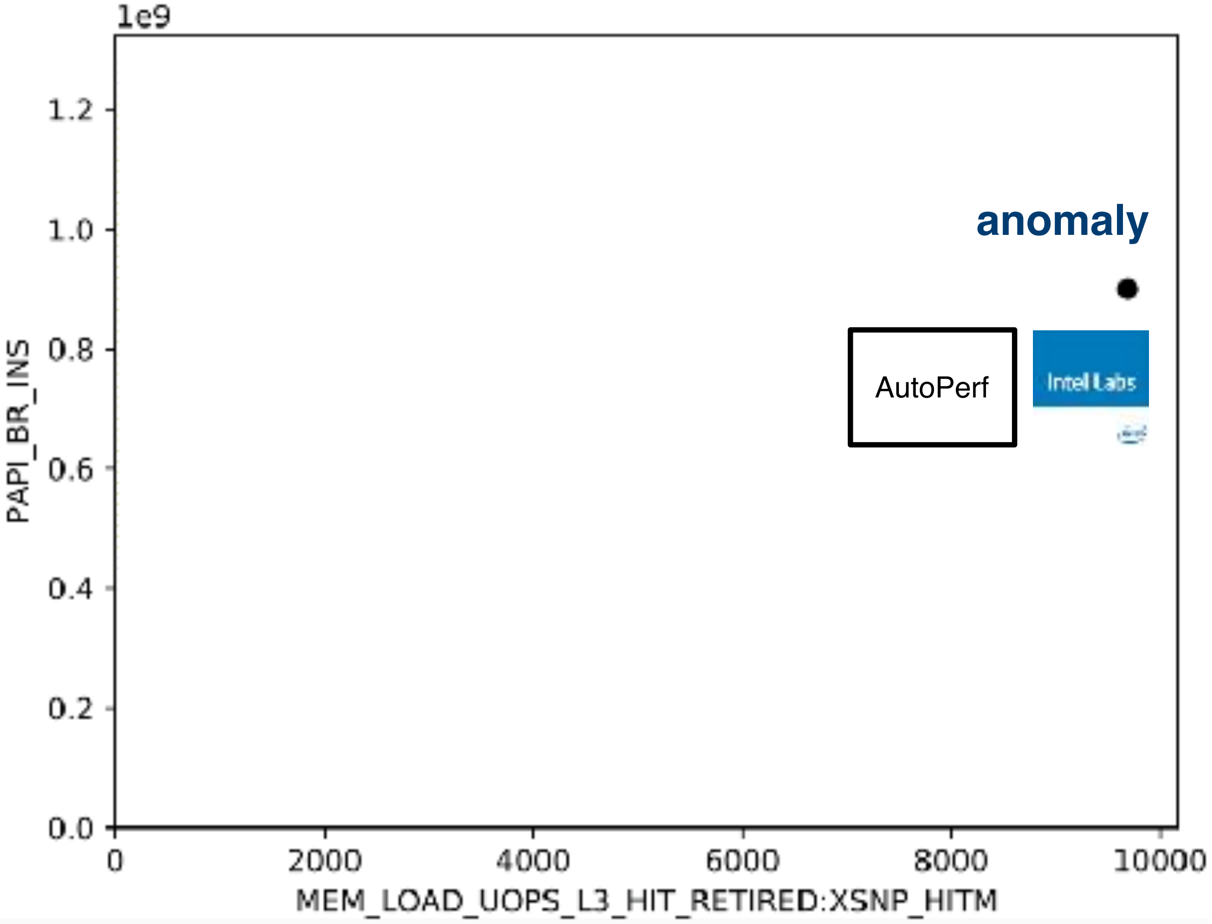
Goal: automatic identification & correction of adaptation-like anomalies

Team: Xin Sheng (MIT), Mejbah Alam (IL), Justin Gottschlich (IL), Armando Solar-Lezama (MIT)

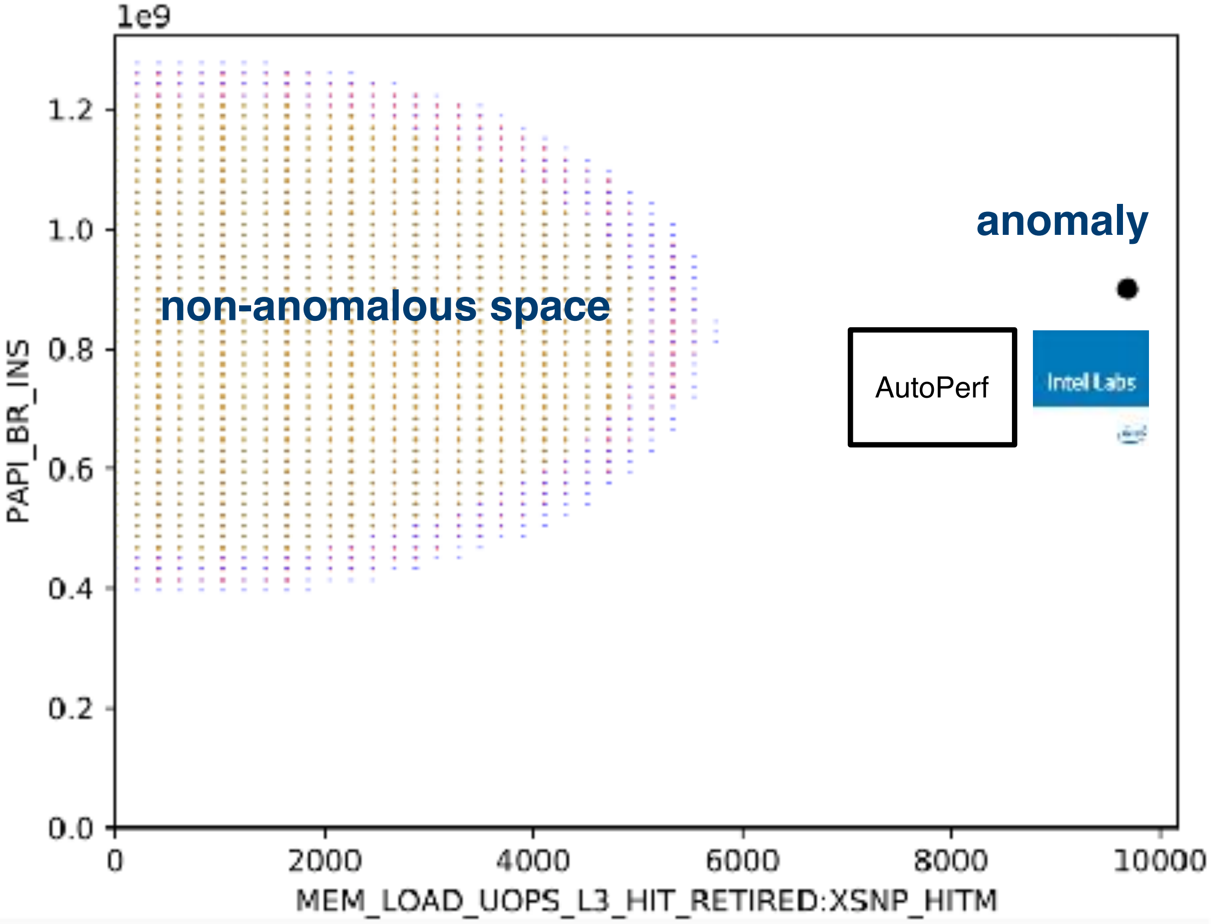
Early Results: Anomaly Detection Interpretability



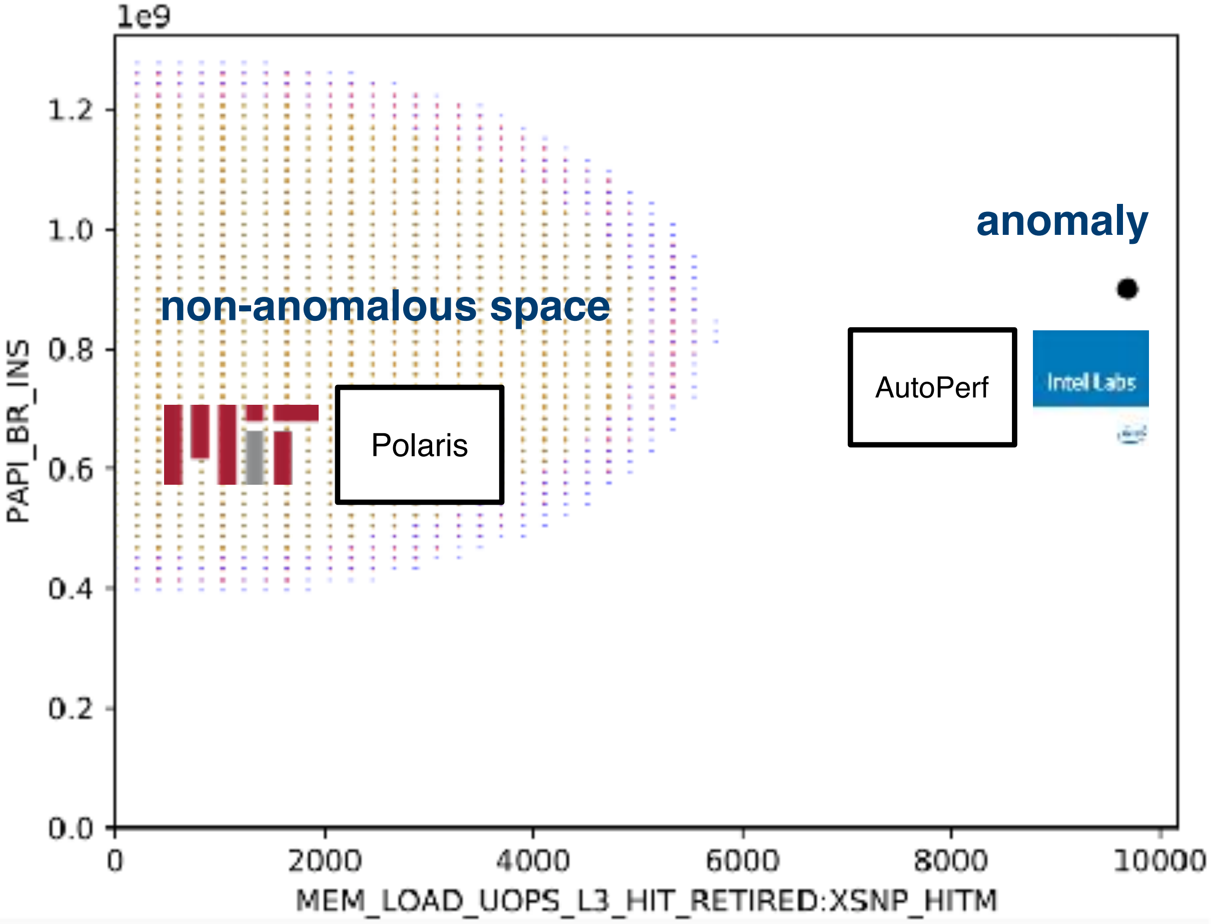
Early Results: Anomaly Detection Interpretability



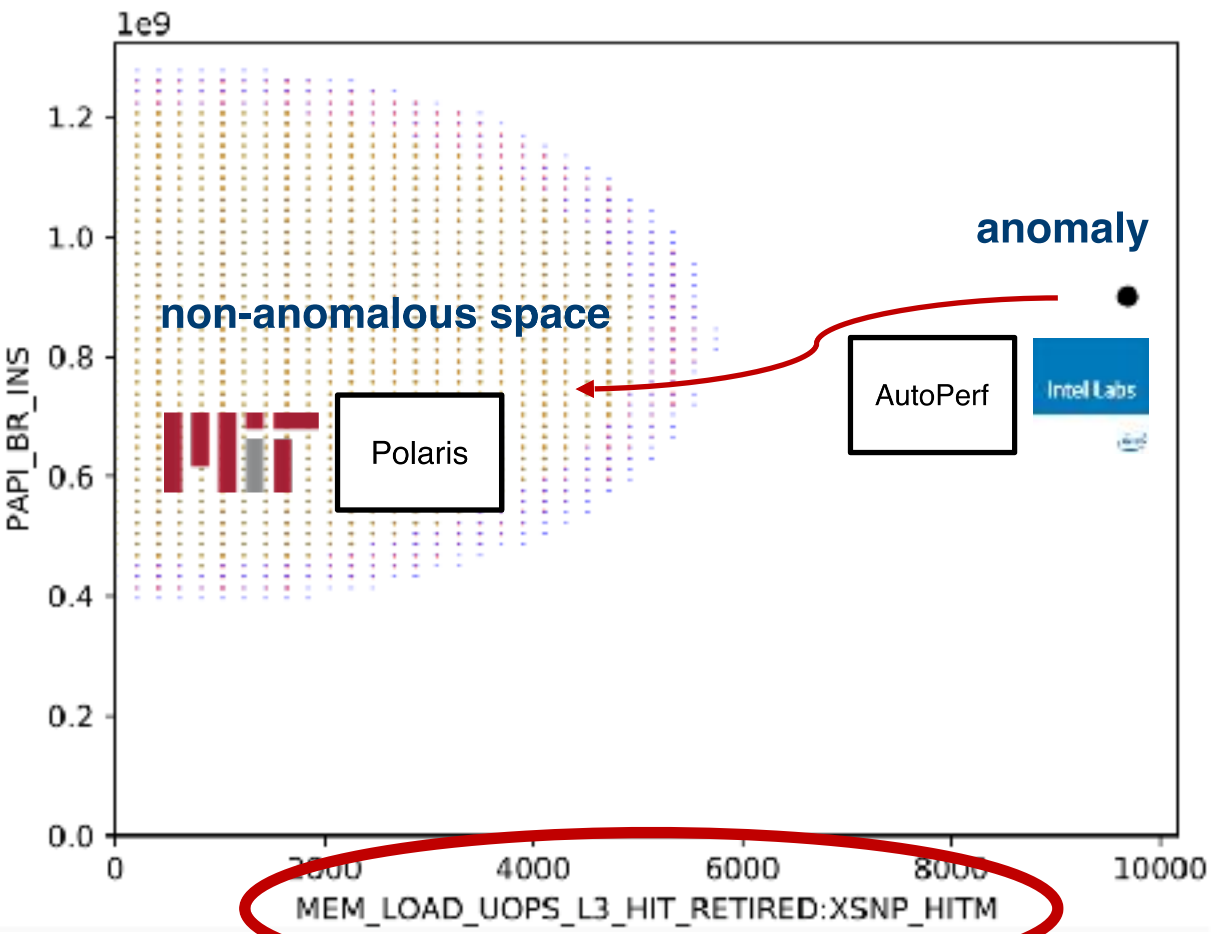
Early Results: Anomaly Detection Interpretability



Early Results: Anomaly Detection Interpretability



Early Results: Anomaly Detection Interpretability



Action: move to non-anomalous space by reducing L3 HITMs

AutoTVM (NeurIPS ’18 Spotlight)

Learning to Optimize Tensor Programs

Tianqi Chen[1] Lianmin Zheng[2] Eddie Yan[1] Ziheng Jiang[1] Thierry Moreau[1]
Luis Ceze[1] Carlos Guestrin[1] Arvind Krishnamurthy[1]
[1]Paul G. Allen School of Computer Science & Engineering, University of Washington
[2]Shanghai Jiao Tong University

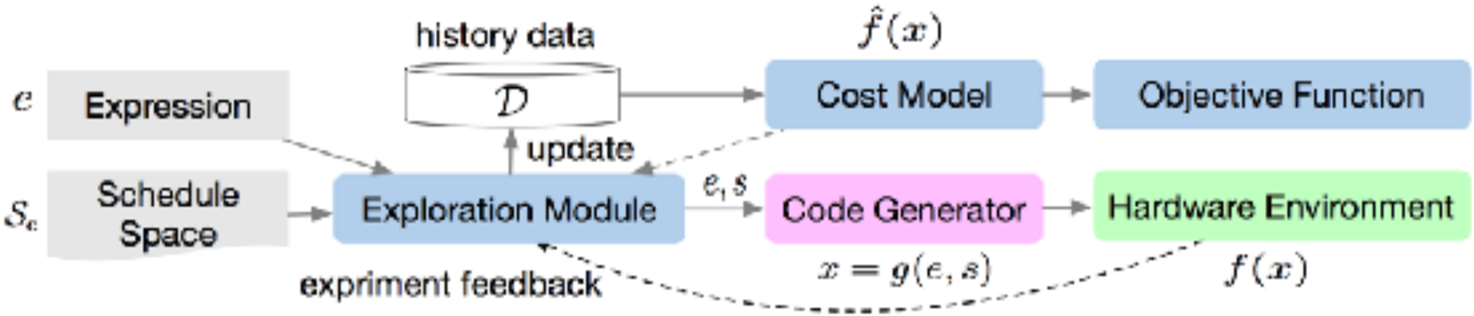


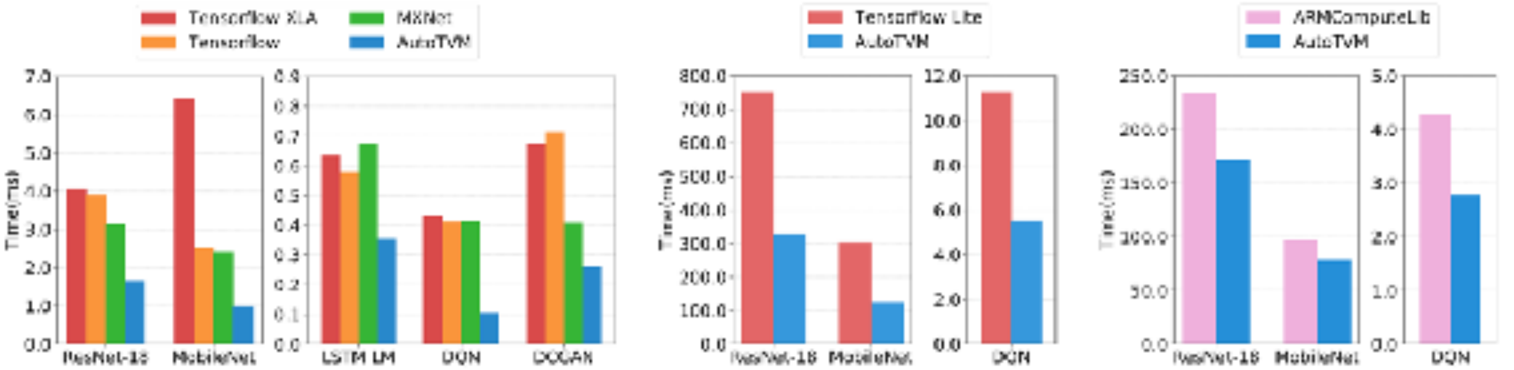
Figure 2: Framework for learning to optimize tensor programs.

AutoTVM (NeurIPS '18 Spotlight)

Learning to Optimize Tensor Programs

Tianqi Chen[1] Lianmin Zheng[2] Eddie Yan[1] Ziheng Jiang[1] Thierry Moreau[1]
Luis Ceze[1] Carlos Guestrin[1] Arvind Krishnamurthy[1]
[1]Paul G. Allen School of Computer Science & Engineering, University of Washington
[2]Shanghai Jiao Tong University

Lower is faster



(a) NVIDIA TITAN X End2End (b) ARM Cortex-A53 End2End (c) ARM Mali-T860 End2End

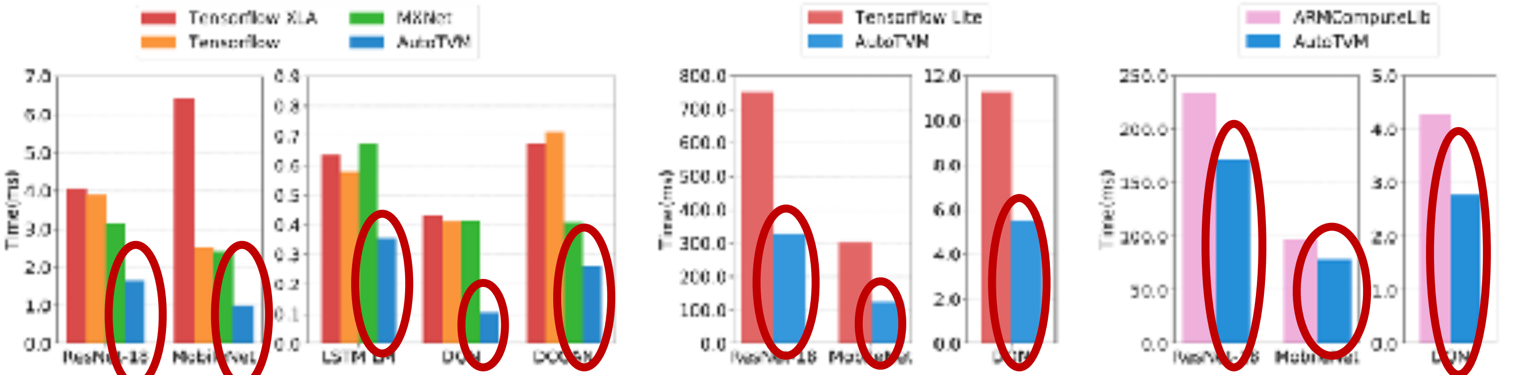
Figure 11: End-to-end performance across back-ends. [2]AutoTVM outperforms the baseline methods.

AutoTVM (NeurIPS '18 Spotlight)

Learning to Optimize Tensor Programs

Tianqi Chen[1] Lianmin Zheng[2] Eddie Yan[1] Ziheng Jiang[1] Thierry Moreau[1]
Luis Ceze[1] Carlos Guestrin[1] Arvind Krishnamurthy[1]
[1]Paul G. Allen School of Computer Science & Engineering, University of Washington
[2]Shanghai Jiao Tong University

Lower is faster



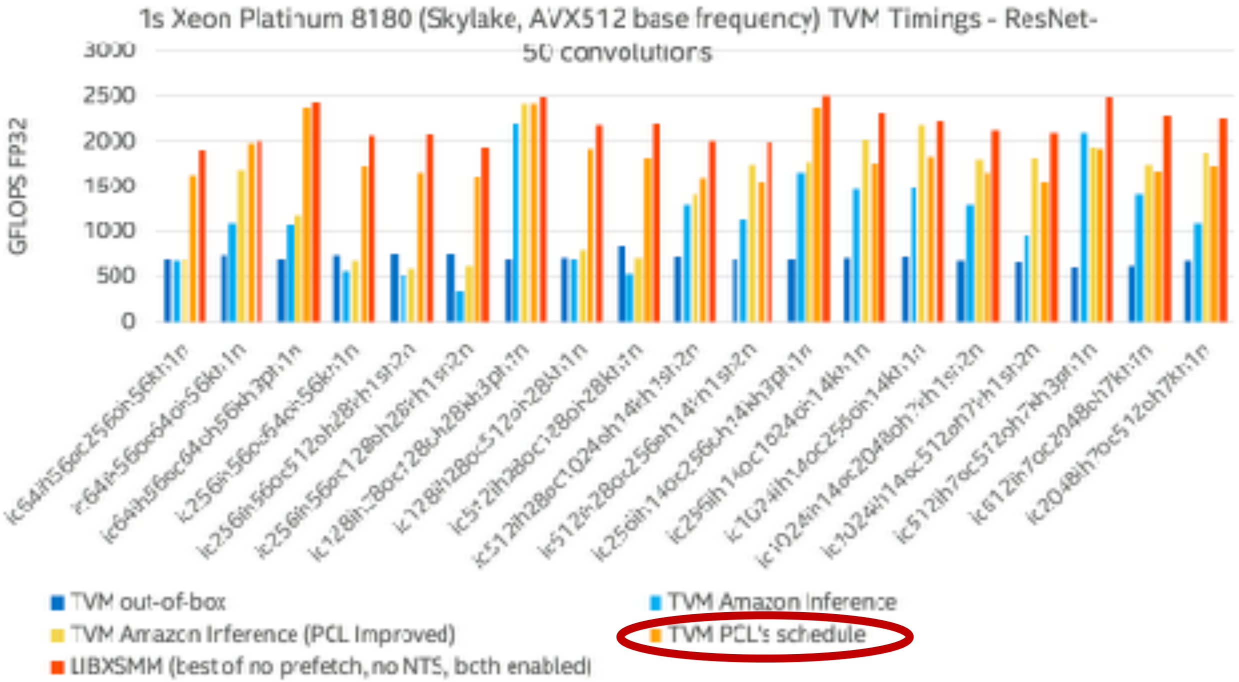
(a) NVIDIA TITAN X End2End (b) ARM Cortex-A53 End2End (c) ARM Mali-T860 End2End

Figure 11: End-to-end performance across back-ends. [2]AutoTVM outperforms the baseline methods.

Intel + TVM

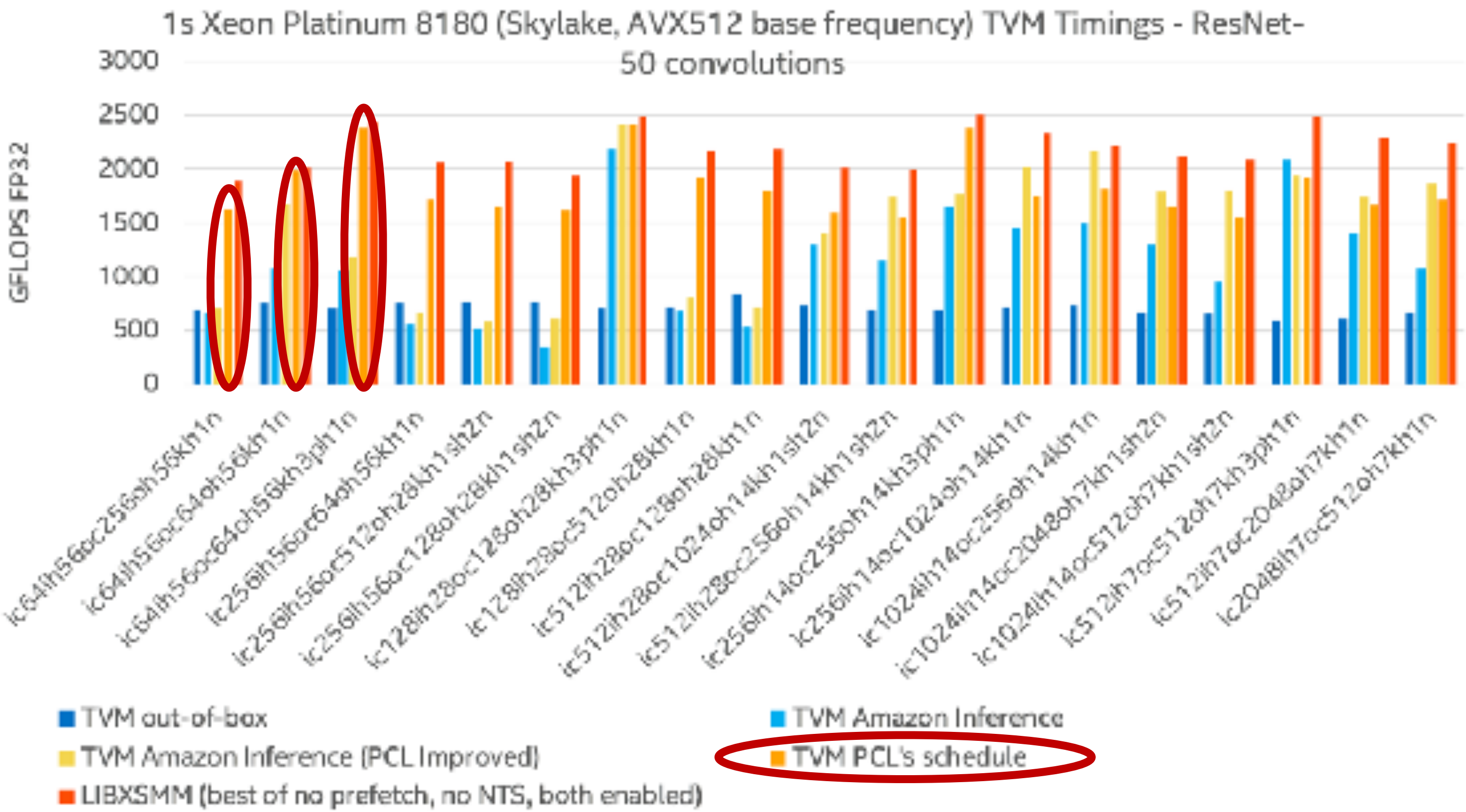
Performance Results for MB=28, 85% of LIBXSMM

Higher is faster



Performance Results for MB=28, 85% of LIBXSMM

Higher is faster



Conclusion

justin.gottschlich@intel.com

- **Machine programming is coming!**
  - Interested in collaborating? Please contact me!
  - Teaching machine programming course @ Penn (Spring 2020)

- **Machine Learning and Programming Languages (MAPL) Workshop**
  - Please consider submitting a paper to MAPL '19 (@ PLDI '19)
    - 10 page ACM SIGPLAN published proceedings (submission: Jan/Feb)
  - General Chair: Tim Mattson (Intel Labs)
  - Program Chair: Armando Solar-Lezama (MIT)

Questions?

justin.gottschlich@intel.com