



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

Q 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

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

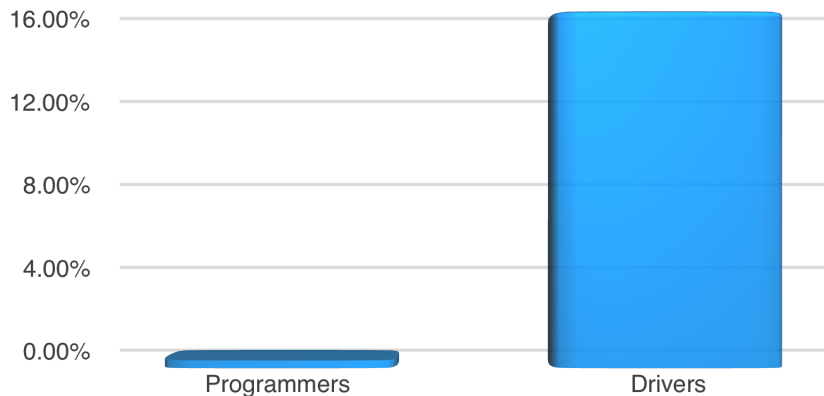
% of programmers: > 0.34% <

2019 human population 7,714M

2019 developers 1,200M

% of drivers: > 15.56% <

Programmers vs. Drivers (Population)



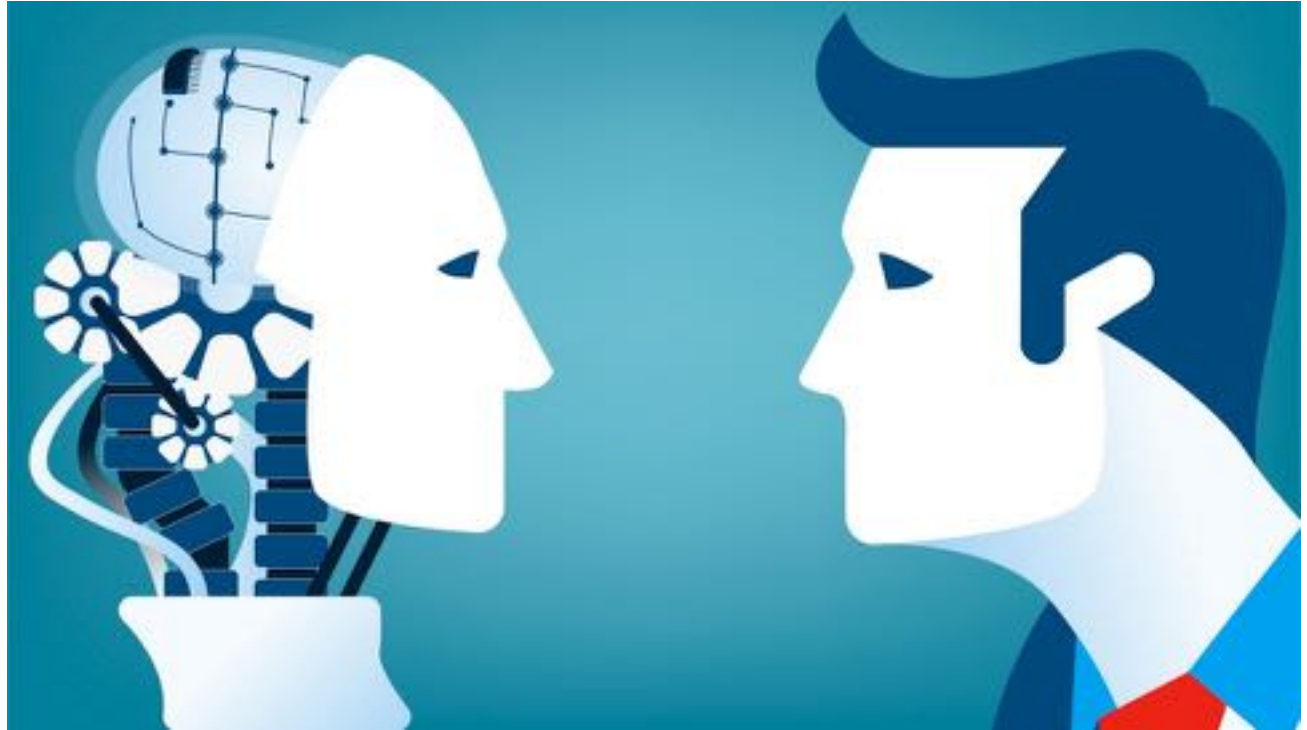
Motivation

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/thesketch.pdf>

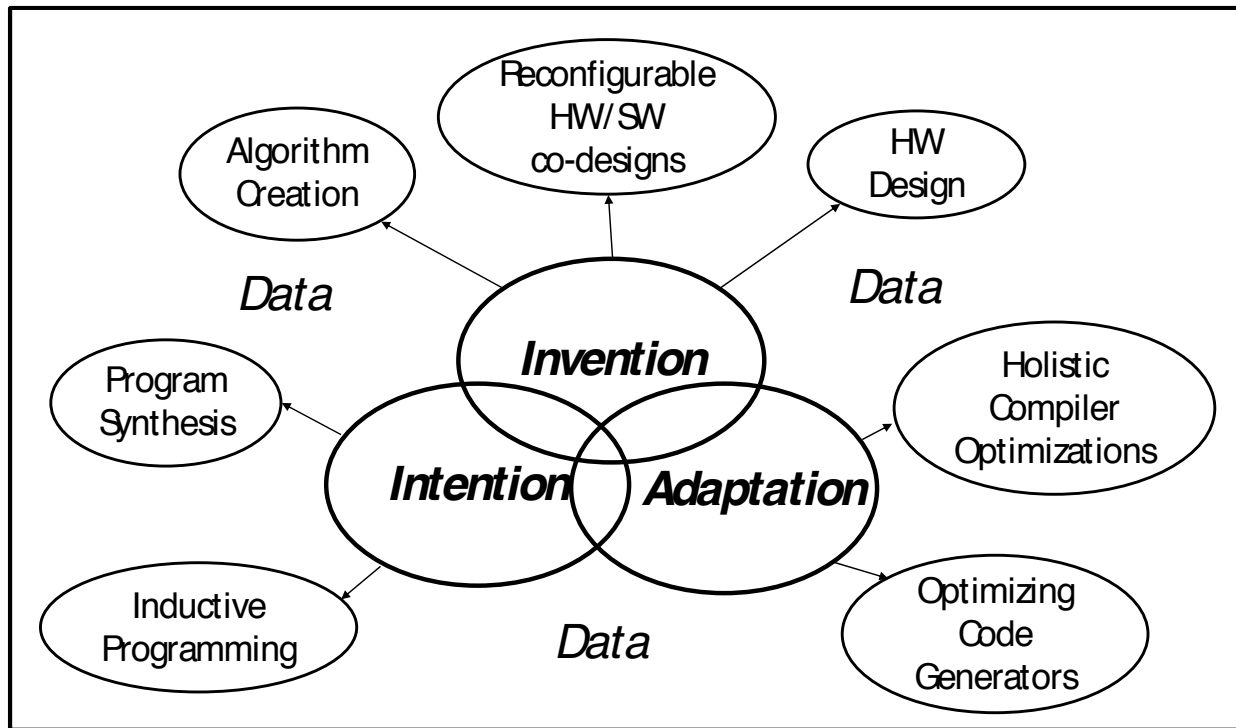
[4] <https://arxiv.org/abs/1611.01989>

[5] <https://research.fb.com/finding-and-fixing-software-bugs-automatically-with-sapfix/>

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

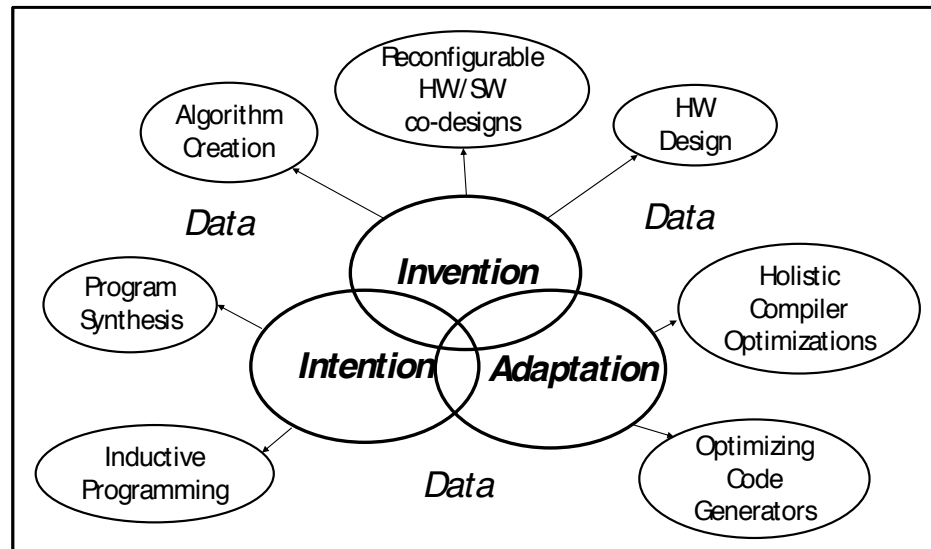
- *“Automating String Processing in Spreadsheets using Input-Output Examples”* (Sumit Gulwani)
- *“Program Synthesis by Sketching”* (Armando Solar-Lezama, Adviser: R. Bodik)

■ 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 your data

Start typing and let Excel finish your work for you

Email	First Name
Nancy.FreeHater@northtraders.com	Nancy
Andrew.Ewing@northtraders.com	Andrew
Jan.Karas@thwainor.com	Jan
Mariya.Sergienko@graphicdesignstudio.com	Mariya
Steven.Thorpe@northtraders.com	Steven
Michael.Hopper@northtraders.com	Michael
Robert.Lowe@northtraders.com	Robert
Laura.Gussoni@adventure-works.com	Laura
Anne.LP@northtraders.com	Anne
Alexander.Davis@comcast.com	Alexander
Kim.Shane@northtraders.com	Kim
Marshall.Hopewig@northtraders.com	Marshall
Gerwald.Oberleitner@northtraders.com	Gerwald
Ann.Zelik@northtraders.com	Ann
Yvonne.McKay@northtraders.com	Yvonne
Amancia.Finto@northtraders.com	Amancia

Excel ribbon: Start | **1. Fill** | 2. Analyze | 3. Chart | Learn More | ©

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)

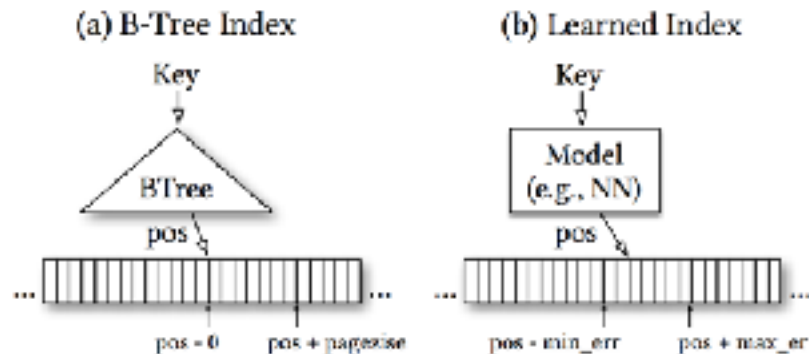


Figure 1: Why B-Trees are models

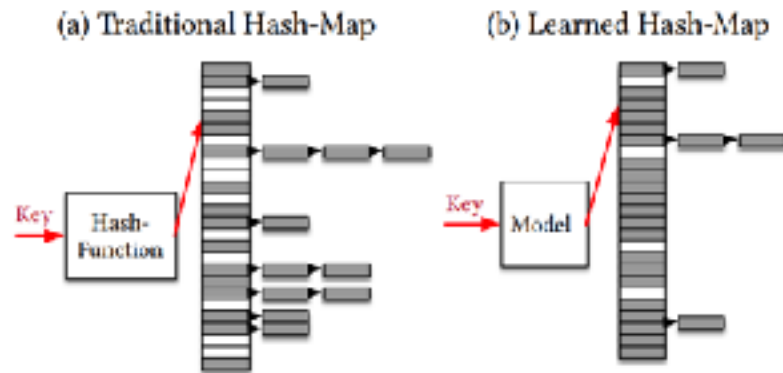


Figure 7: Traditional Hash-map vs Learned Hash-map

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)

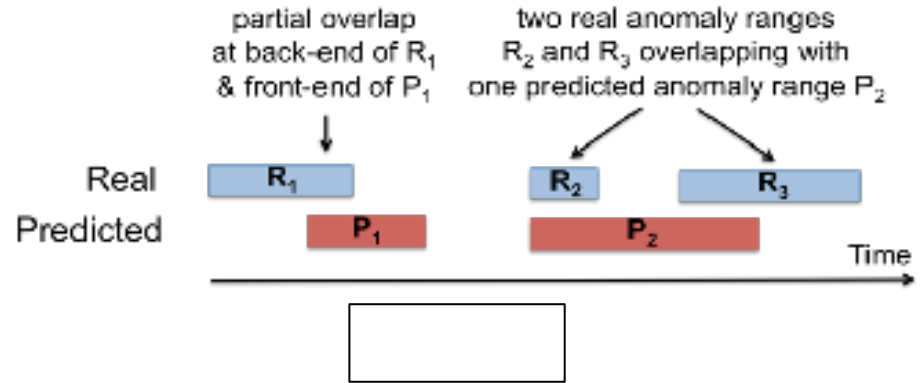
■ Invention

- *“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)

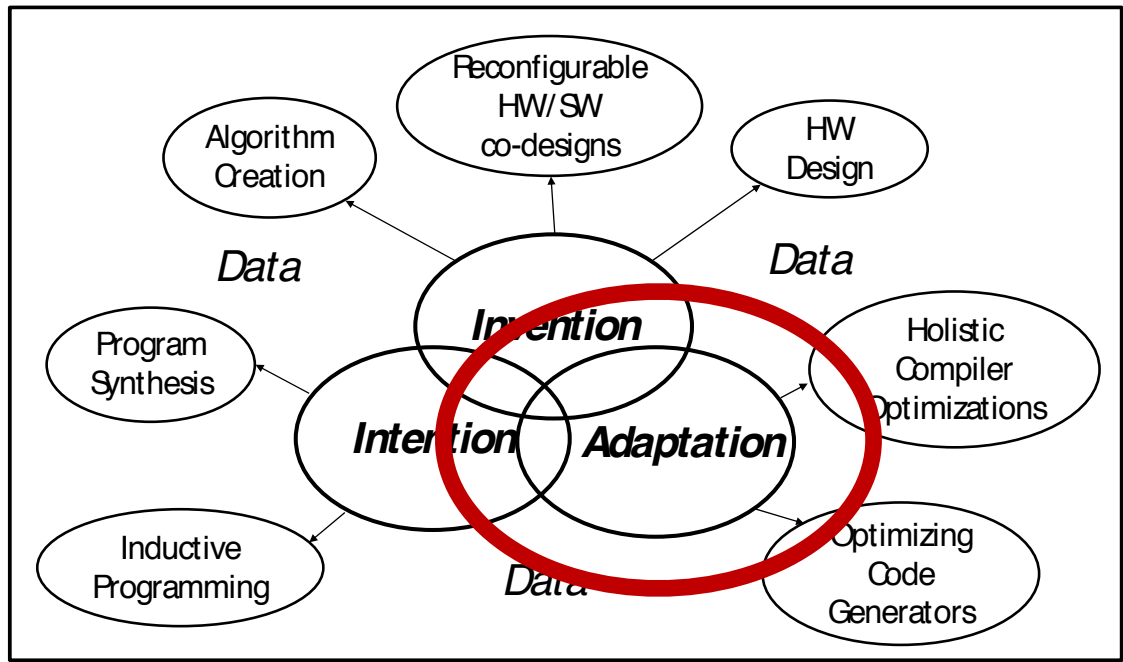
Range-based Anomalies



■ Adaptation

Anomaly Detection Interpretability

(Xin Sheng, Mejbah Alam, Justin Gottschlich, Armando Solar-Lezama)



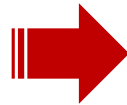
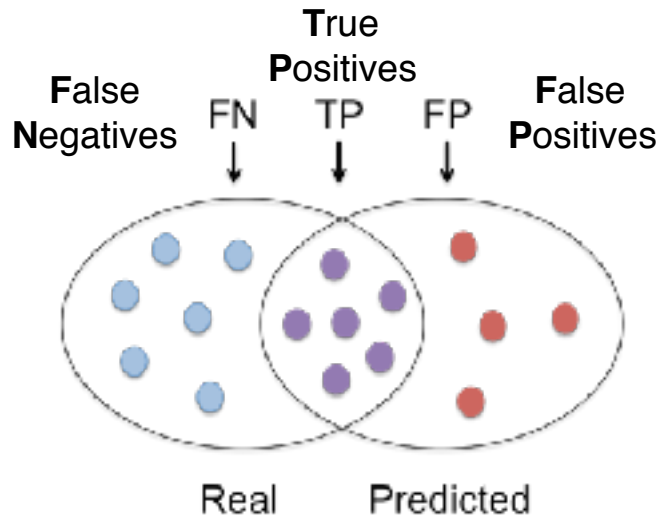
Adaptation

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

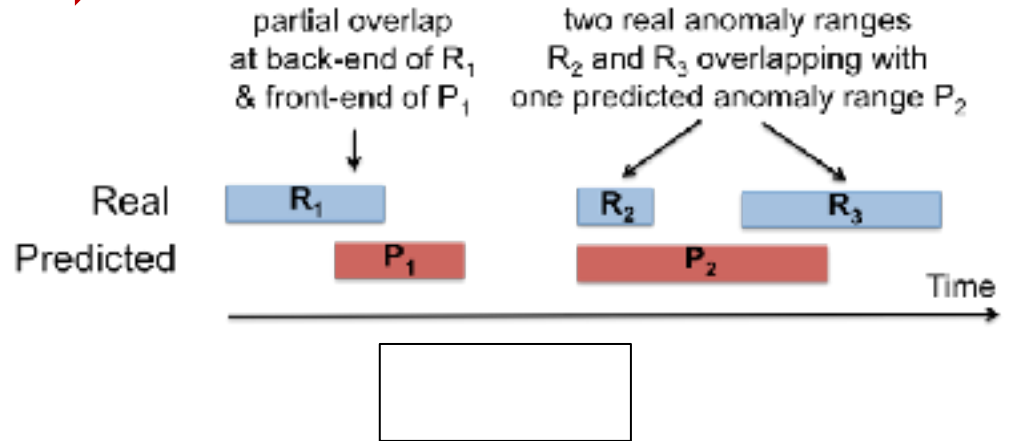
Adaptation is principally about range-based anomaly detection

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

Noritas Duffell
Intel Labs and MIT
noritad@mit.edu

The Jun Lee
Microsoft
leejun@microsoft.com

Saeed Ebrahimi
Intel Labs and MIT
saeed@mit.edu

Miguel Almaraz
Intel Labs
malm@intel.com

Justin Gottrich
Intel Labs
jgottrich@intel.com

Abstract

Classical anomaly detection is principally concerned with point-based anomalies, those anomalies that occur at a single point in time. They fail to detect anomalies that are long-lived, making them poor over a period of time. In this paper, we present a new model for range-based anomalies, where the occurrence of an anomaly depends on the range-based anomalies, while retaining the classical model's ability to classify point-based anomaly detection systems.

1. Introduction

Anomaly detection (AD) is the process of identifying and confirming those events, or behaviors, that deviate from the expected norm. The proper identification of anomalies can be critical for many domains. Some examples are early diagnosis of illness and disease [1], fraud detection for cyber-attacks [2], or safety analysis for self-driving cars [3]. Many real-world anomalies can be observed at time series data. Anomaly systems that detect anomalies should reason about them as they occur over a period of time. We call such events range-based anomalies, which are a subset of both conventional and collective anomalies [4]. More precisely, a range-based anomaly is an anomaly that occurs over a consecutive sequence of time points, where the time series data points exist between the beginning and the end of the anomaly. The standard metrics for evaluating anomaly detection algorithms today, *Precision* and *Recall*, have been around since the 1950s, originally formulated to evaluate document retrieval algorithms by counting the number of documents that were correctly returned against those that were not [5].

Formally defined as follows, *Recall* and *Precision* are a good match for single-point AD [1] (where TP , FP , FN are the number of true positives, false positives, false negatives, respectively):

$$\text{Recall} = TP / (TP + FN) \quad (1)$$

$$\text{Precision} = TP / (TP + FP) \quad (2)$$

Informally, *Recall* is the ratio of which a system has identified anomalies without mispredicting any anomalous events. *Precision* is the ratio a system can identify anomalies without mispredicting non-anomalous events. In this sense, *Recall* and *Precision* are complementary. This characterization shows useful when they are combined, such as in the F_1 score, which is their harmonic mean. Such combinations help gauge the quality of both anomalies and non-anomalous predictions. While useful for point-based anomalies, classical recall and precision suffer from the inability to represent those range-based anomalies. This has a negative side effect on the advancement of AD systems. In particular, many state-of-the-art AD systems' accuracy is being misrepresented, because point-based recall and precision are being used to measure their performance for range-based anomalies. Moreover, the need to consistently identify those anomalies is growing. In the past several years, a wide exploration of streaming and real-time systems [2, 6, 14, 21, 28, 31]. In addition, there are real-time recall and precision to encompass range-based anomalies. Unlike previous work [2, 21], our mathematical

12nd Conference on Artificial Intelligence, Processing Systems (NIPS), Montreal, Canada.



Precision & Recall for Time Series

Customizable weights & functions

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

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

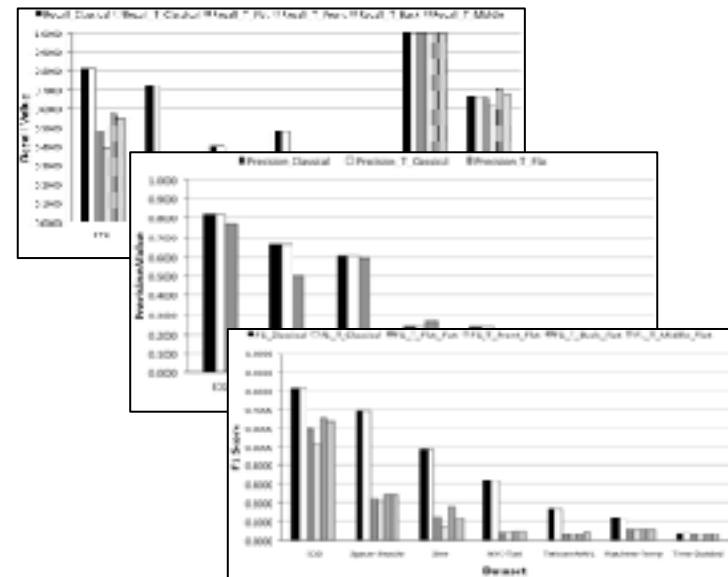
$$Precision_T(R, P) = \frac{\sum_{i=1}^{N_p} Precision_T(R, P_i)}{N_p}$$

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

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 $\gamma()$ and $\delta()$
 - + 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

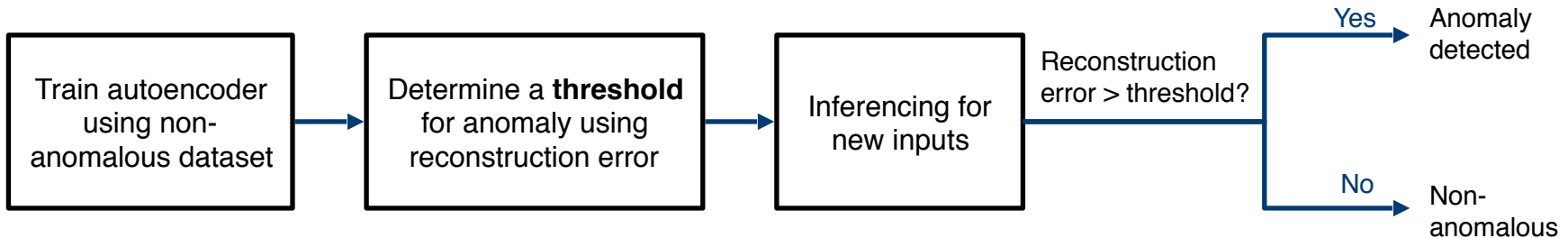
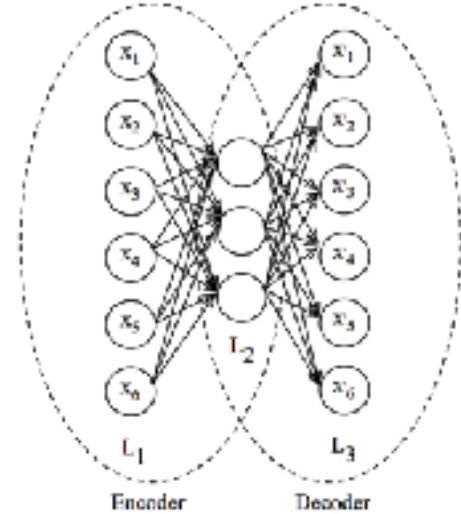
The “*How*” and “*Why*” are open questions for *anomaly detection & neural networks*

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



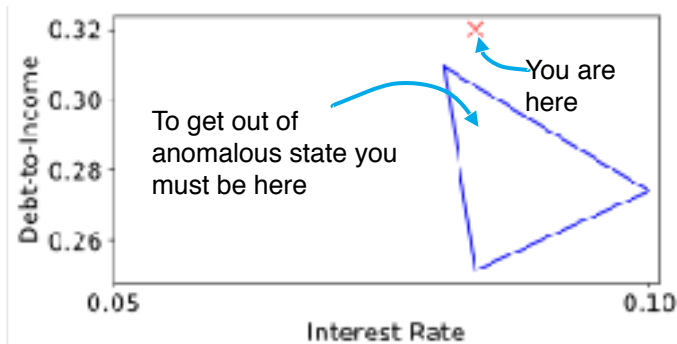
Interpreting Neural Network Judgments

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.

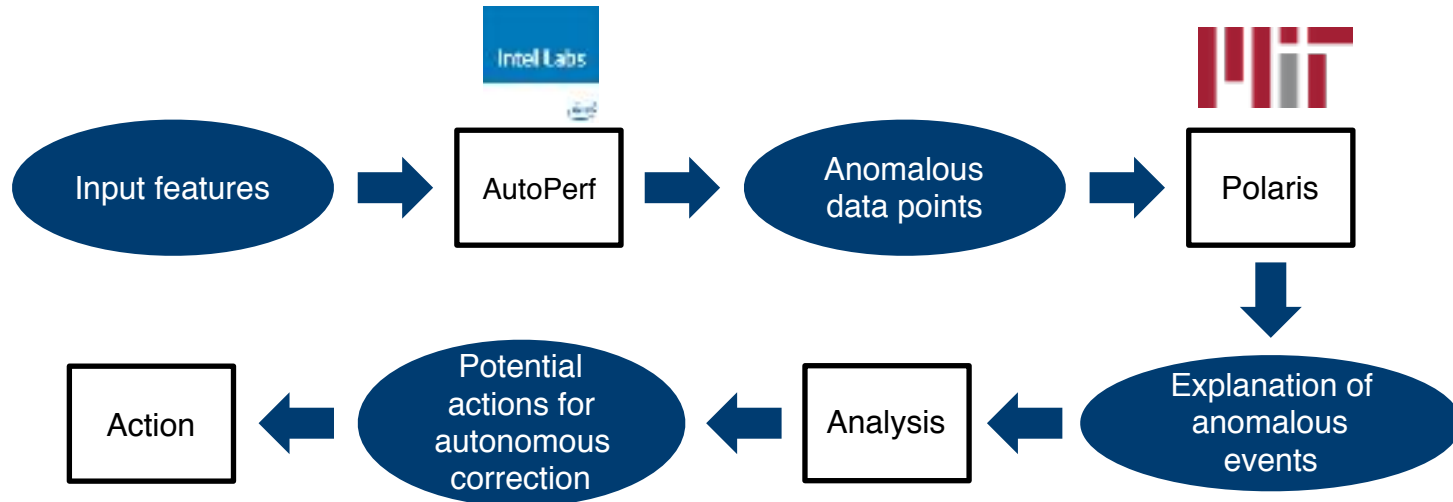
- **Desired properties:**

- Minimal
- Stable
- Symbolic



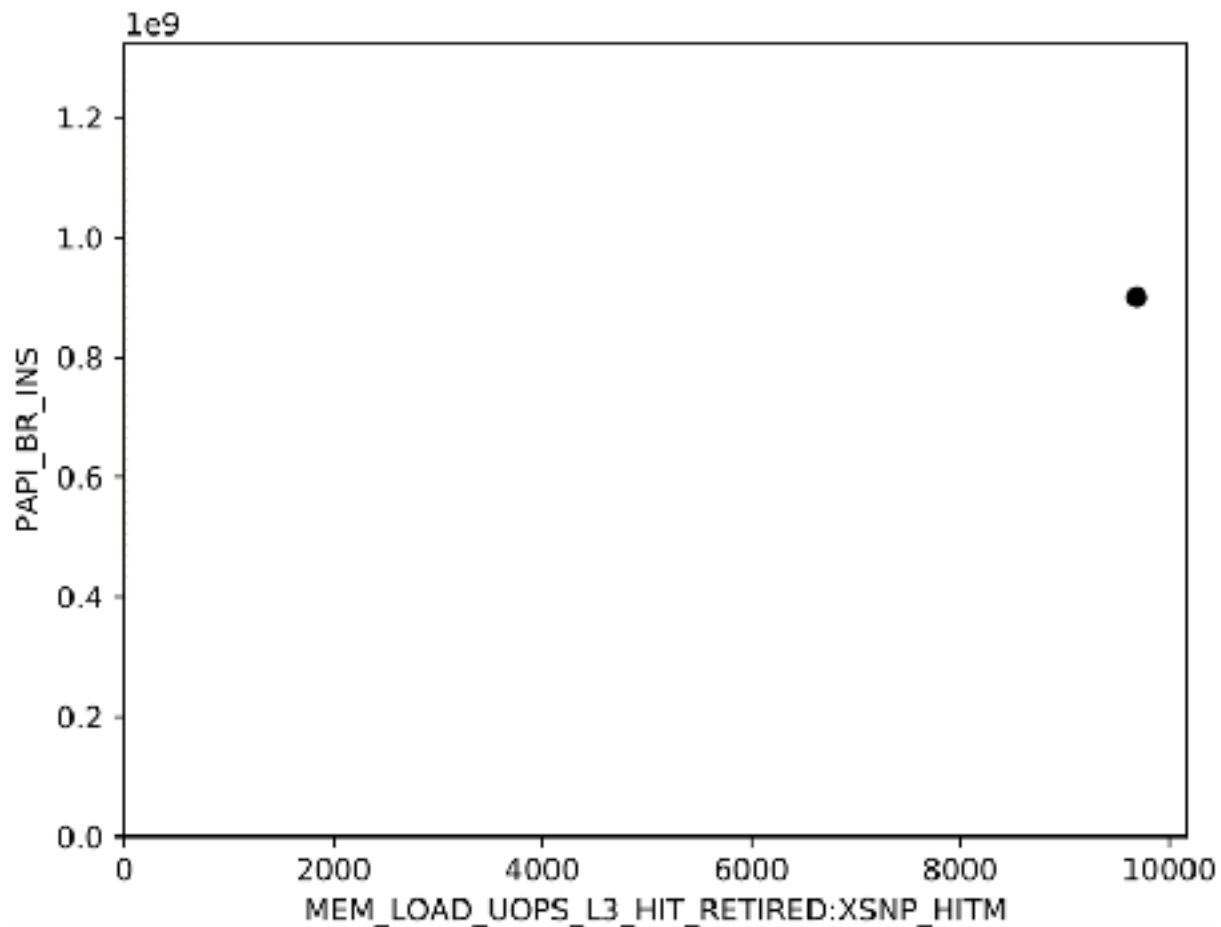
[2] Interpreting Neural Network Judgments via Minimal, Stable, and Symbolic Corrections,
Xin Zhang (MIT), Armando Solar-Lezama (MIT), Rishabh Singh (Google Brain), [NIPS '18 (to appear)]

IL+MIT: Interpreting AutoPerf using Polaris

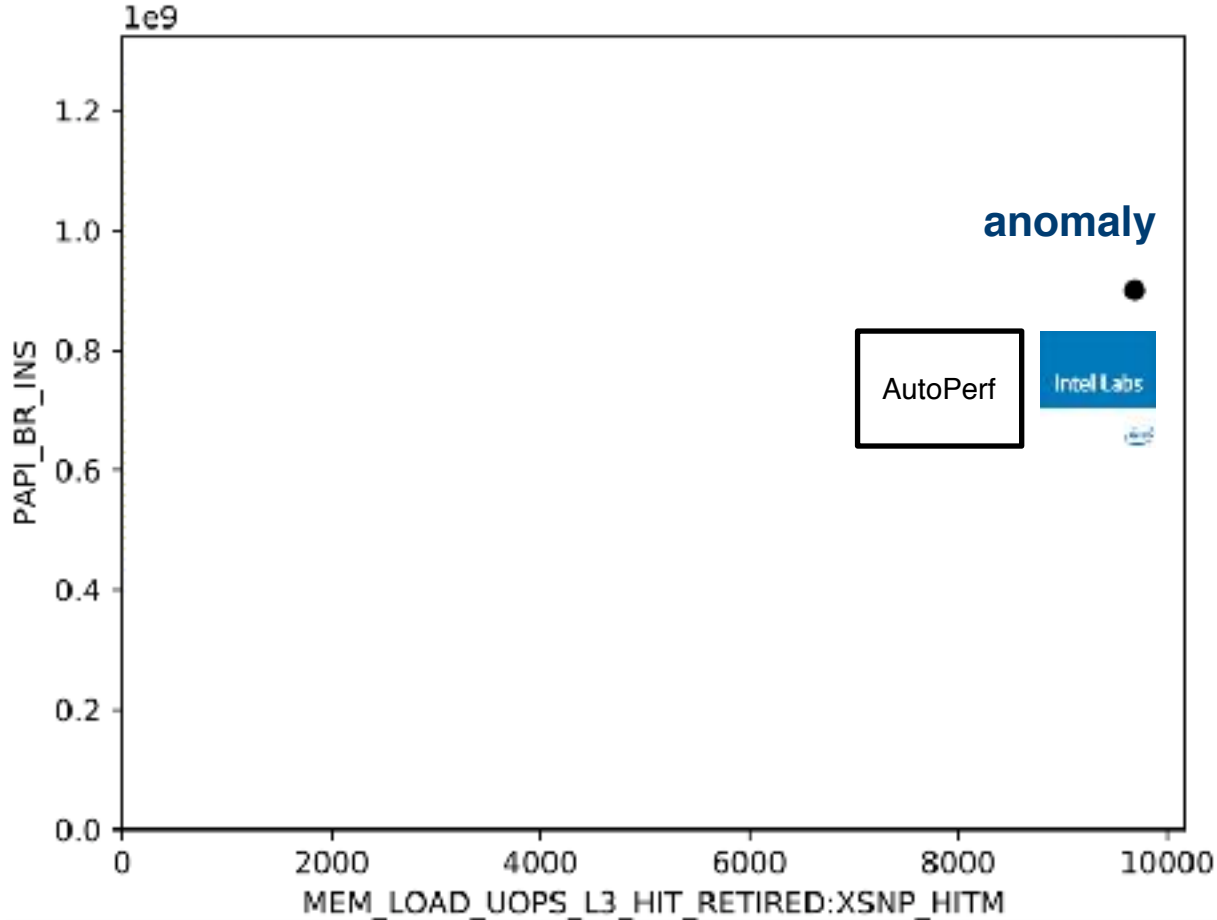


Goal: automatic identification & correction of adaptation-like anomalies

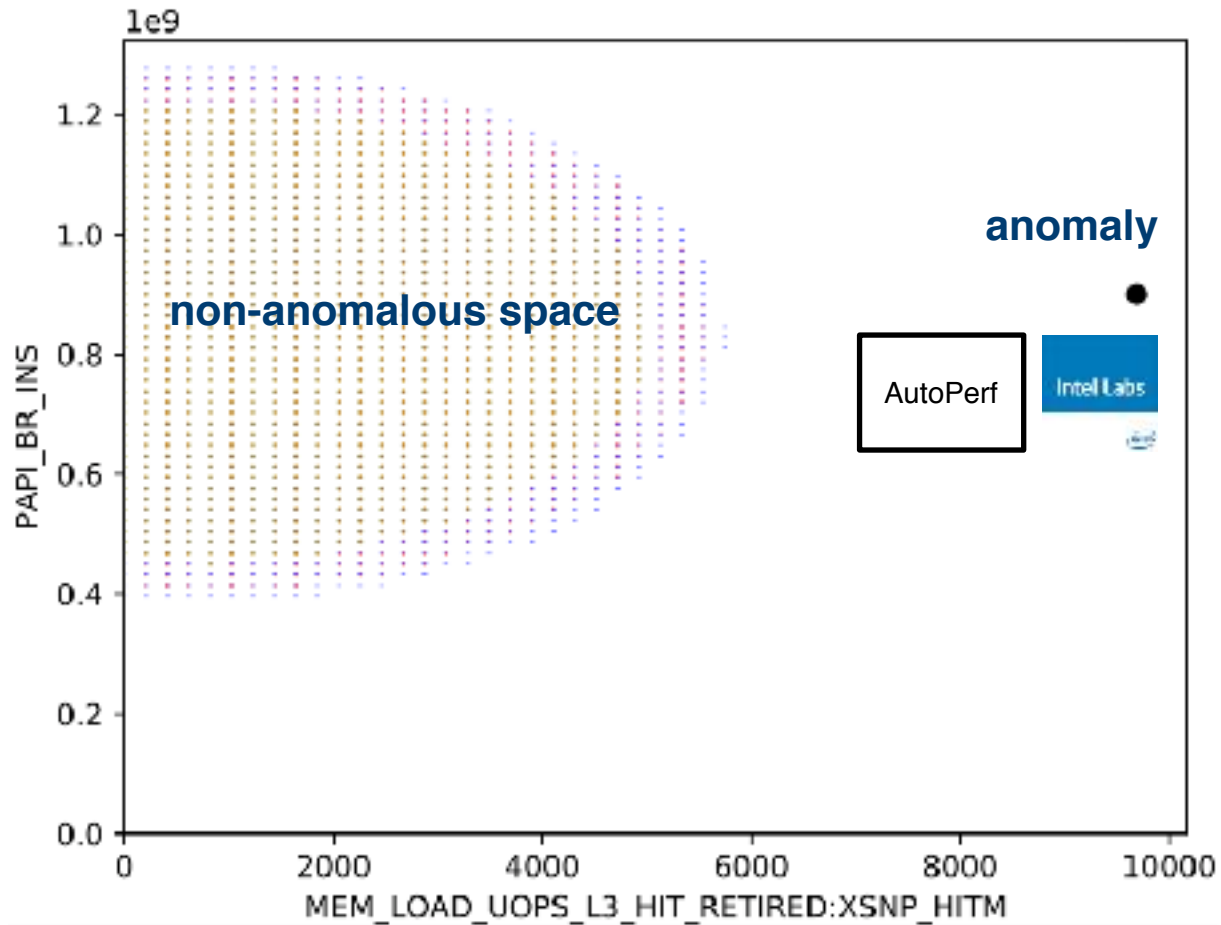
Early Results: Anomaly Detection Interpretability



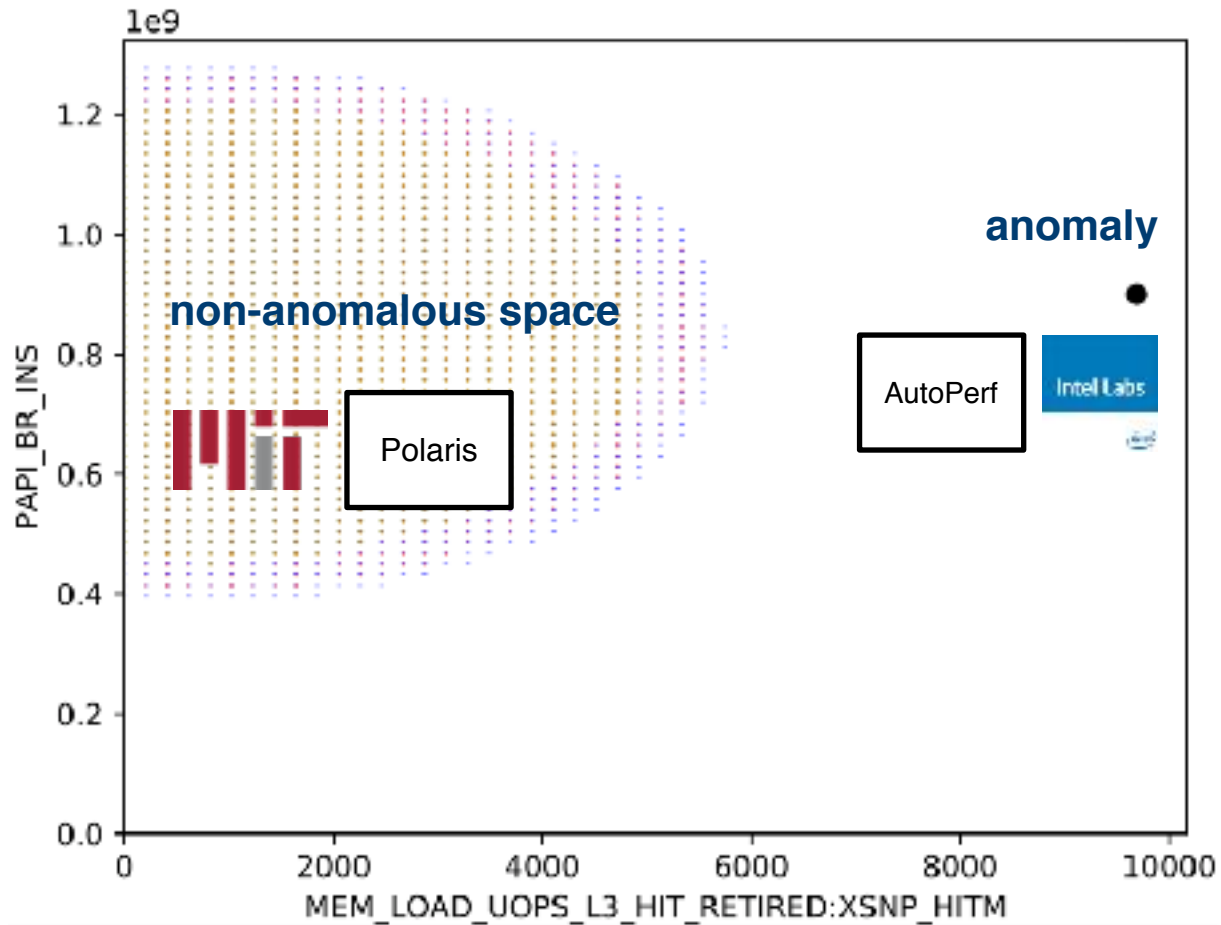
Early Results: Anomaly Detection Interpretability



Early Results: Anomaly Detection Interpretability

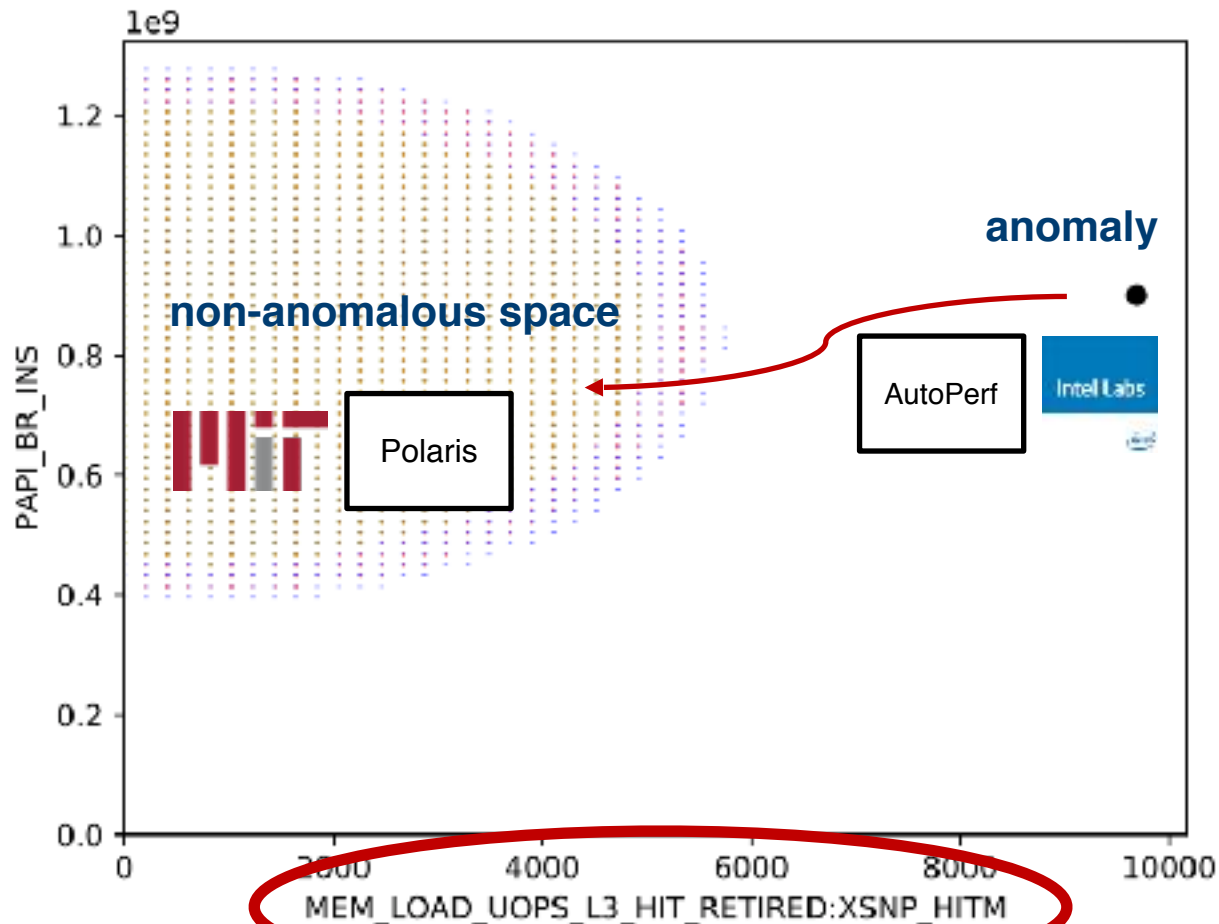


Early Results: Anomaly Detection Interpretability



Early Results: Anomaly Detection Interpretability

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



Learning to Optimize Tensor Programs

Tianqi Chen¹ Lianmin Zheng² Eddie Yan¹ Ziheng Jiang¹ Thierry Moreau¹
Luis Ceze¹ Carlos Guestrin¹ Arvind Krishnamurthy¹
¹Paul G. Allen School of Computer Science & Engineering, University of Washington
²Shanghai Jiao Tong University

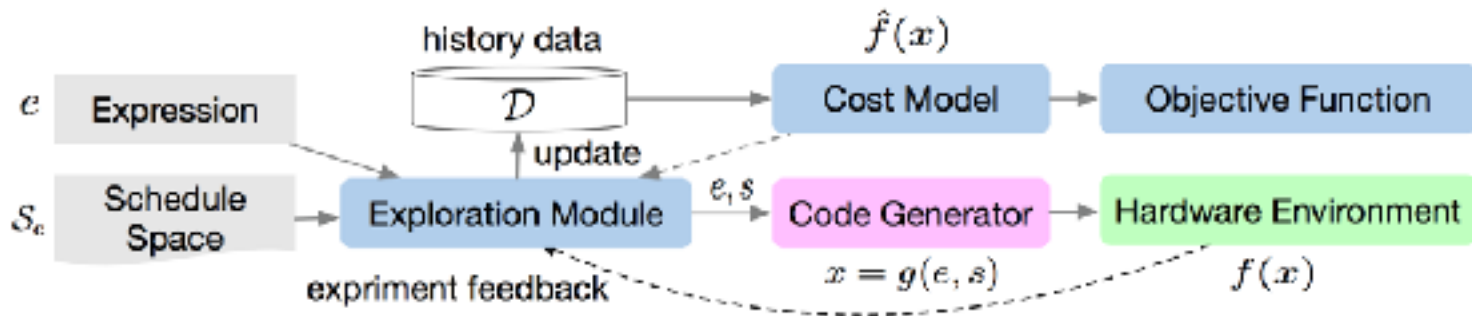


Figure 2: Framework for learning to optimize tensor programs.

Tianqi Chen¹ Lianmin Zheng² Eddie Yan¹ Ziheng Jiang¹ Thierry Moreau¹
Luis Ceze¹ Carlos Guestrin¹ Arvind Krishnamurthy¹
¹Paul G. Allen School of Computer Science & Engineering, University of Washington
²Shanghai Jiao Tong University

Lower is faster

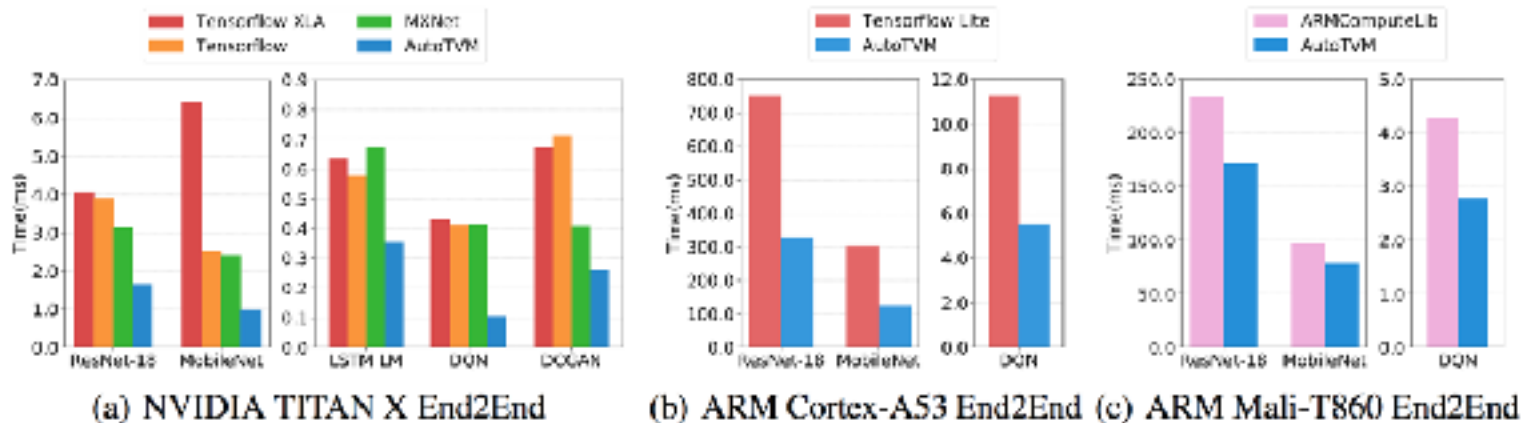


Figure 11: End-to-end performance across back-ends. ²AutoTVM outperforms the baseline methods.

Tianqi Chen¹ Lianmin Zheng² Eddie Yan¹ Ziheng Jiang¹ Thierry Moreau¹
 Luis Ceze¹ Carlos Guestrin¹ Arvind Krishnamurthy¹
¹Paul G. Allen School of Computer Science & Engineering, University of Washington
²Shanghai Jiao Tong University

Lower is faster

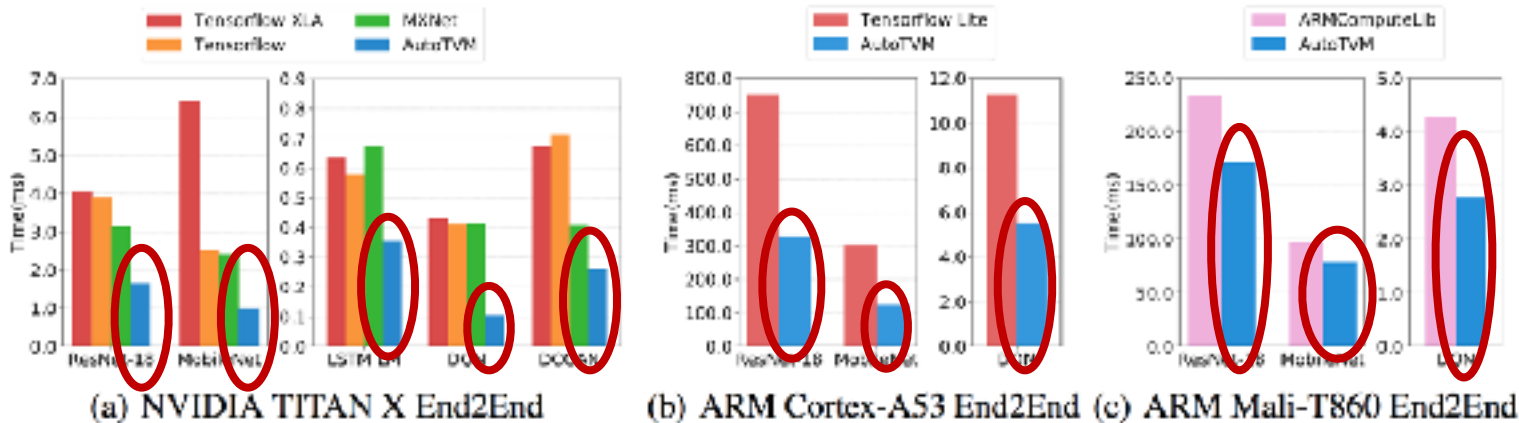


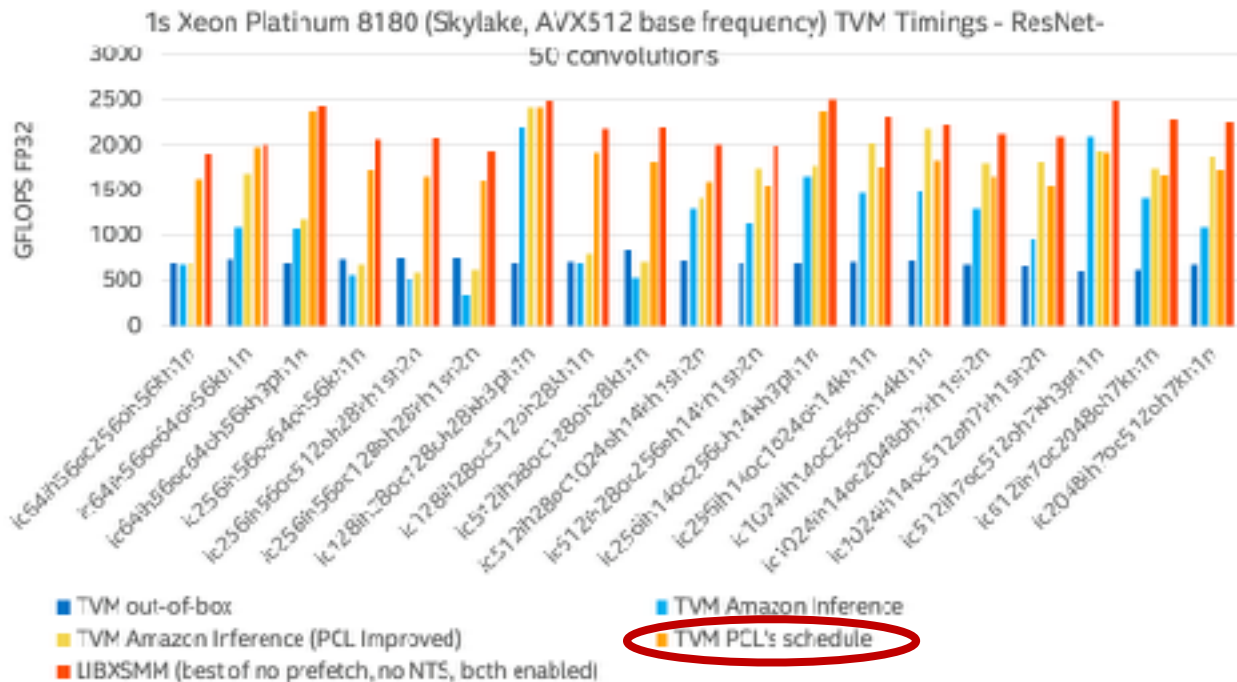
Figure 11: End-to-end performance across back-ends. ²AutoTVM outperforms the baseline methods.

Intel + TVM

Higher is faster

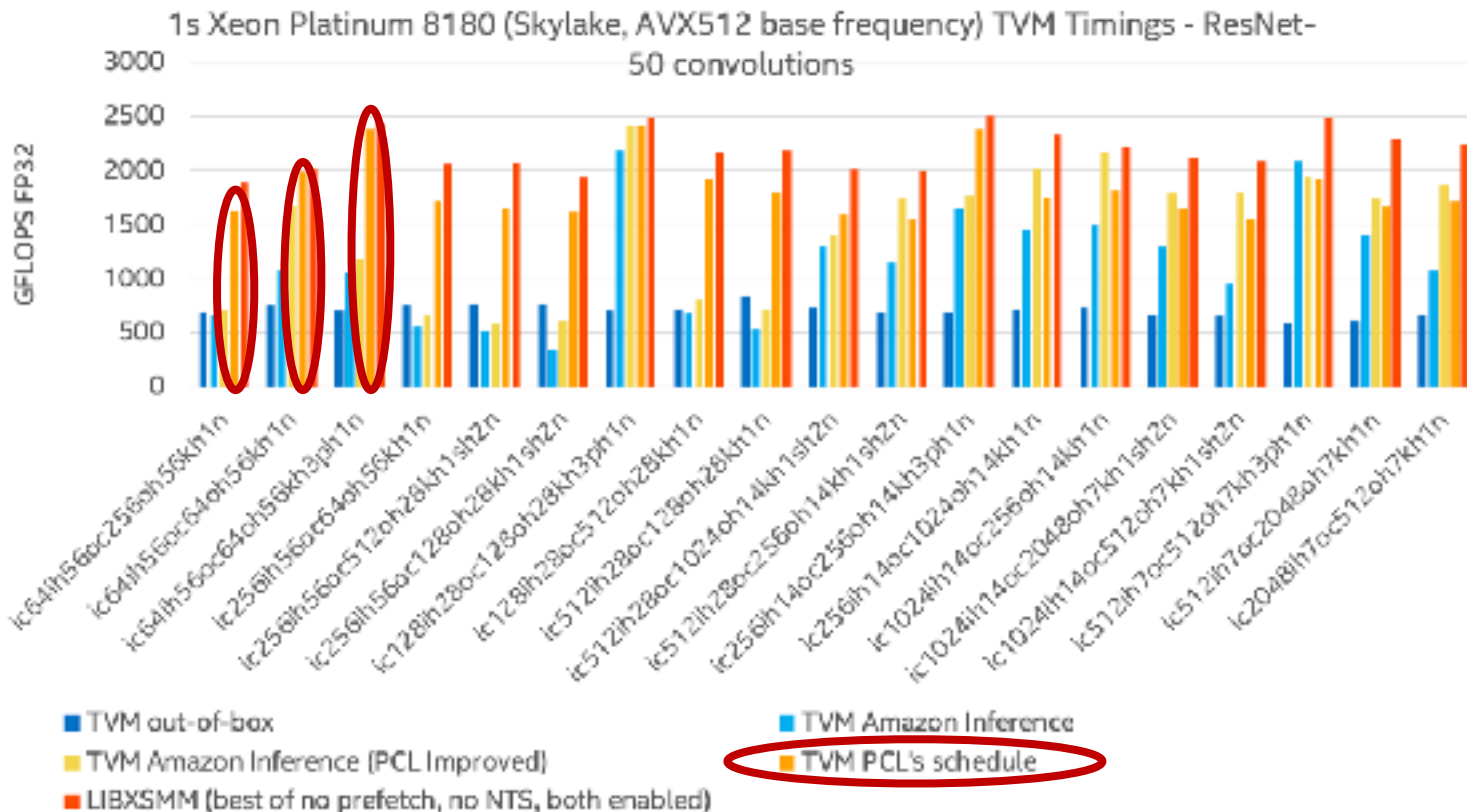


Performance Results for MB=28, 85% of LIBXSMM



Higher is faster

Performance Results for MB=28, 85% of LIBXSMM



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

