A computational cache

A neural-net based algorithm for range matching with application to packet classification

Mark Silberstein
EE, Technion

Joint work with Alon Rashelbach and Ori Rottenstreich

ACSL
Accelerated Computing Systems Lab

TECHNION Israel Institute of Technology

THE ANDREW & ERNA VITERBI FACULTY OF ELECTRICAL ENGINEERING

Home | About Us | Research | Publications | Team | Teaching | Undergraduate Projects | News & Events

Welcome to the Accelerated Computing Systems Lab (ACSL)!

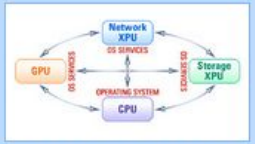
We work on a broad range of computer systems projects spanning hardware architecture, compilers, operating systems, security and privacy, high-speed networking.

All our software is open-source and free .

Feel passionate about building secure and fast computer systems of the future?! Check out how to apply!

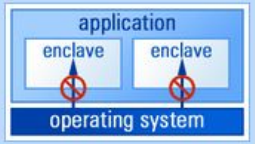
RESEARCH

all research areas



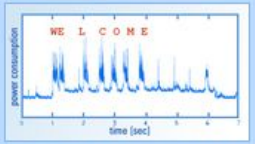
Accelerator-Centric Operating System

Accelerator-centric Operating System Architecture, OmniX, enables direct interaction between accelerators and I/O devices, for example, files and network sockets for GPU kernels



OS Services for Trusted Execution Environments

Our work facilitates the development of complex applications in Secure Enclaves by providing secure virtual memory services.



Harware Side Channels

Side channels have become one of the major threats for systems security. We work on protecting systems from CPU hardware side channels and speculative execution attacks.

GPU computing, Networking, Machine Learning, Distributed Systems

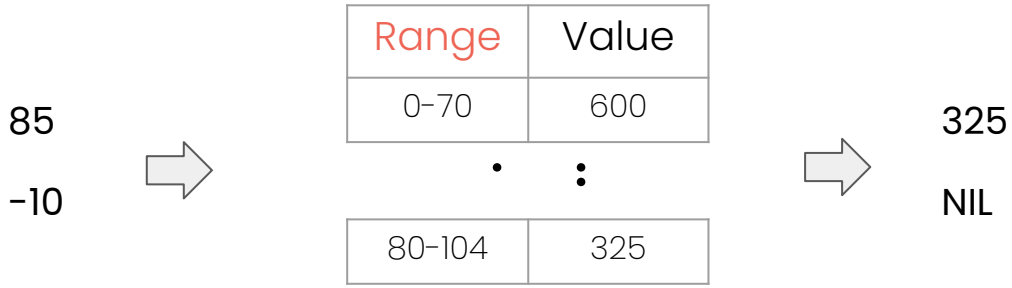
Playground for exploring interesting topics in a search for new ideas

https://acsl.group

Agenda

- Range Matching
  - Applications
  - Background: Recursive Model Index
  - RQ-RMI
- Packet classification
  - NuevoMatch
  - OVS + NuevoMatch
- Future ideas

Range matching queries



Basic building block in a variety of systems

- virtual memory
- network switchers and routers
- file and storage systems
- sparse data structures
- DNA sequencing

Range matching

Traditional data structures

- Regular data (pages)
  - radix trees
- Irregular data (extents)
  - interval trees
- Overlapping ranges (packet rules)
  - hierarchy of hash-tables and (similar to) interval trees

Common problem: scaling is hard due to memory wall!

- Architectural trends
  - Fast Memory does not scale
  - Memory latency to memory remains high
  - Memory bandwidth per core remains low

  Impedes performance of memory-intensive data structures
- Large index: spills out of cache
- Pointer-chasing: memory latency on the critical path

Common problem: scaling is hard due to memory wall!

- Architectural trends
  - Fast Memory does not scale
  - Memory latency to memory remains high
  - Memory bandwidth per core remains low

  Impedes performance of memory-intensive data structures
- Large index: spills out of cache
- Pointer-chasing: memory latency on the critical path

However! NN inference is getting faster all the time!
This talk: how to use NNs to scale range matching

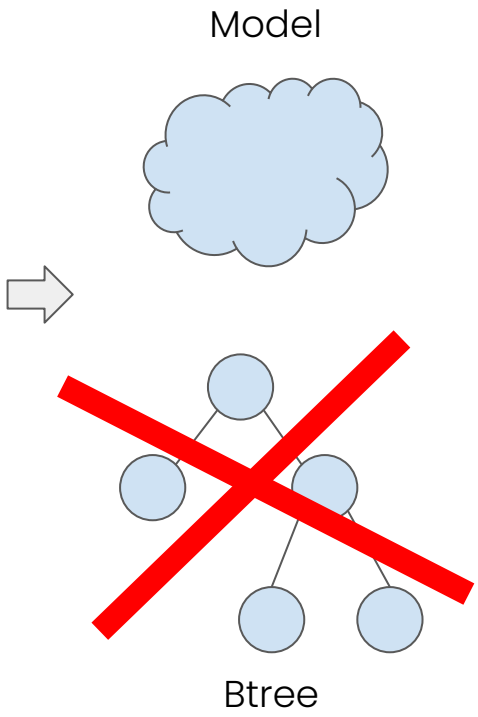
Agenda

- Background: Learned Indexes
- Motivation: why LI cannot be used as is
- Our solution: RQ-RMI
- Application: packet classification

Recursive Model Index (RMI) (1/5)

sorted keys

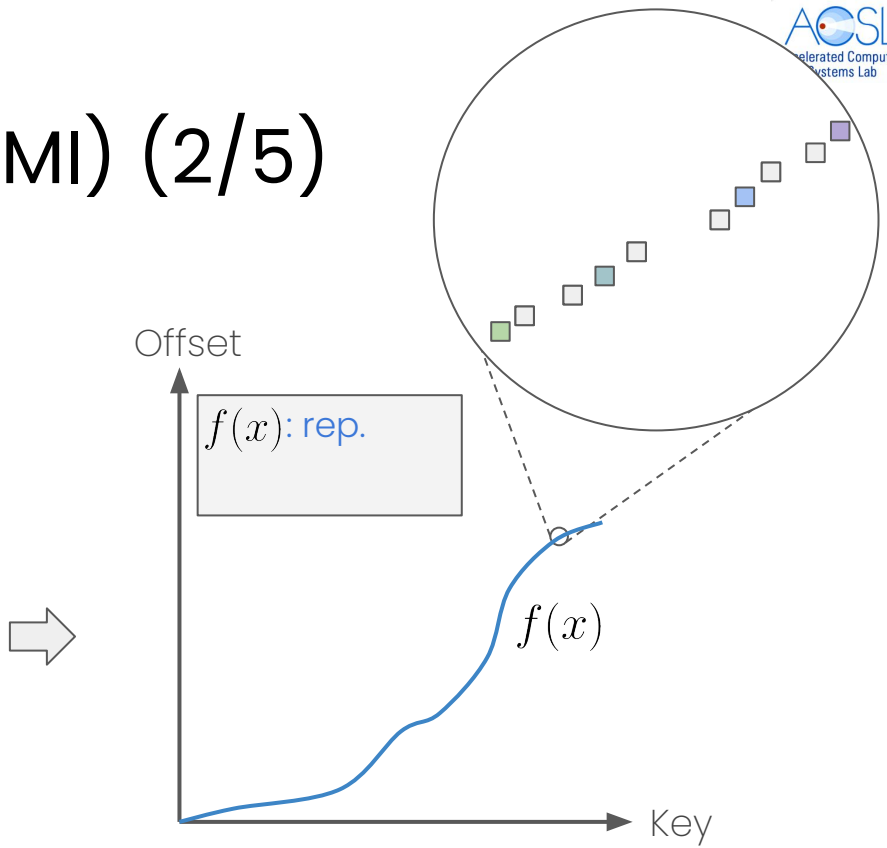
Key	Value
56	value0
60	value1
68	value2
71	value3
80	value4



(*) Kraska et al., "The Case for Learned Index Structures". *SIGMOD 2018.*

Recursive Model Index (RMI) (2/5)

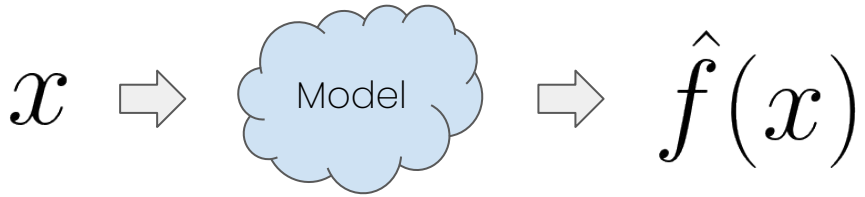
Key	Mem Offset	Value
56	100	value0
60	101	value1
68	102	value2
71	103	value3
80	104	value4
87	105	value5
93	106	value6
100	107	value7
101	108	value8
117	109	value9



$$IndexedKeys = \{ \quad \}$$

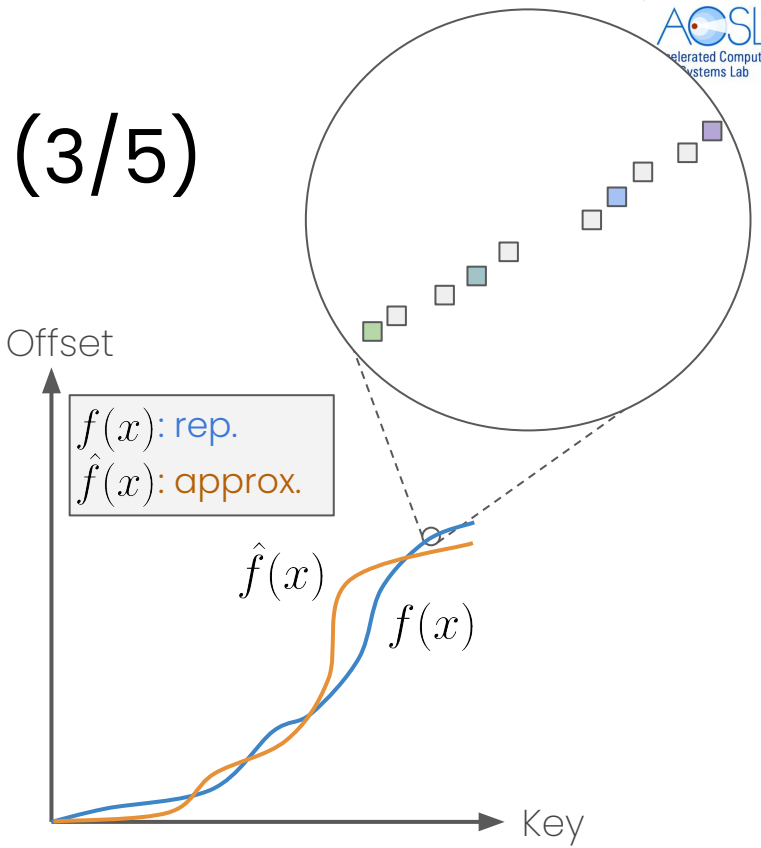
(*) Kraska et al., "The Case for Learned Index Structures". *SIGMOD 2018.*

Recursive Model Index (RMI) (3/5)



Max Approximation Error

$$|f(x) - \hat{f}(x)| < \epsilon$$
$$x \in Keys$$

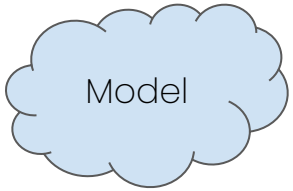


Train model using

$$IndexedKeys = \{ \quad \}$$

(*) Kraska et al., "The Case for Learned Index Structures". *SIGMOD 2018.*

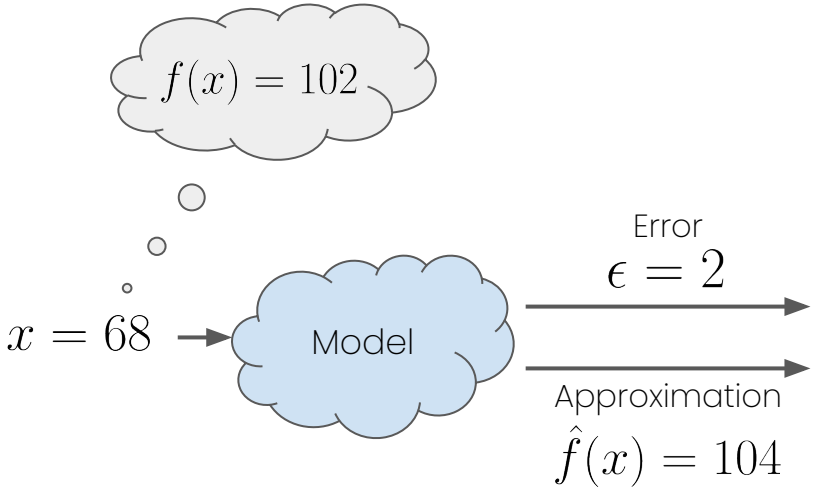
Recursive Model Index (RMI) (4/5)



Key	Mem Offset	Value
60	101	value1
68	102	value2
71	103	value3
80	104	value4
87	105	value5
93	106	value6
100	107	value7

(*) Kraska et al., "The Case for Learned Index Structures". *SIGMOD 2018.*

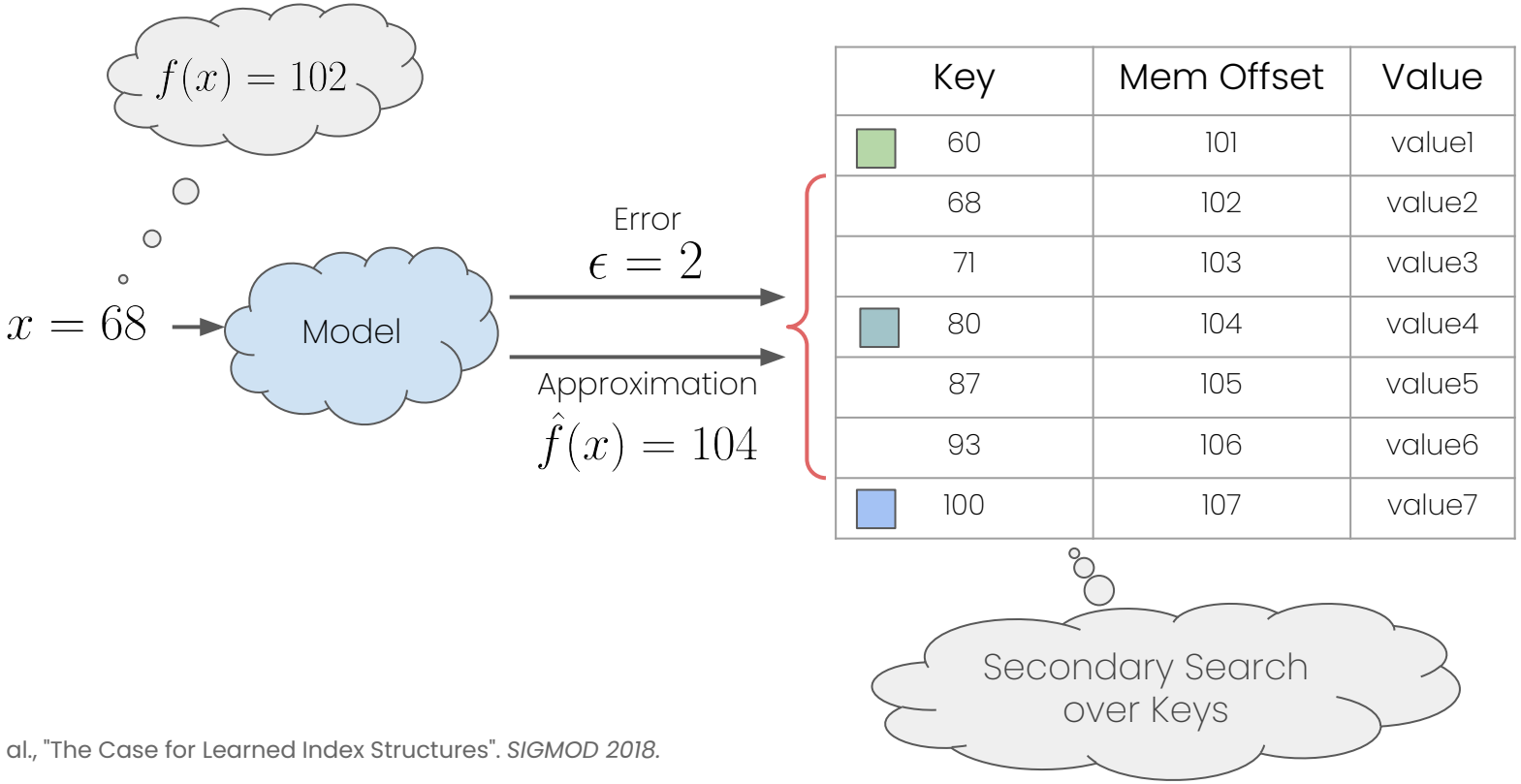
Recursive Model Index (RMI) (4/5)



Key	Mem Offset	Value
60	101	value1
68	102	value2
71	103	value3
80	104	value4
87	105	value5
93	106	value6
100	107	value7

(*) Kraska et al., "The Case for Learned Index Structures". *SIGMOD 2018.*

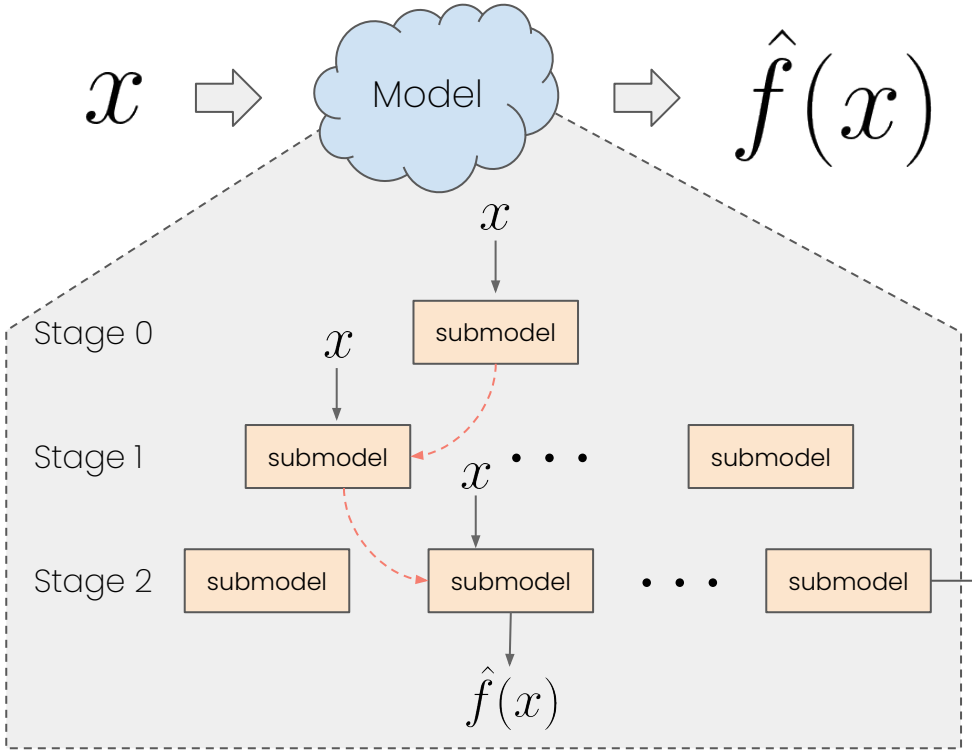
Recursive Model Index (RMI) (5/5)

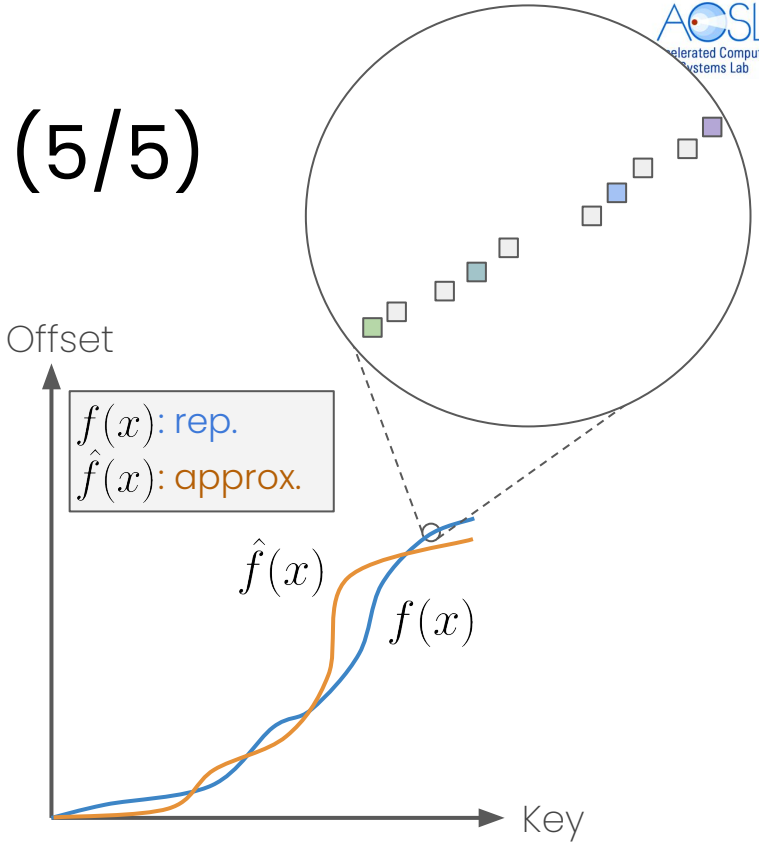


Key	Mem Offset	Value
60	101	value1
68	102	value2
71	103	value3
80	104	value4
87	105	value5
93	106	value6
100	107	value7

(*) Kraska et al., "The Case for Learned Index Structures". *SIGMOD 2018.*

Recursive Model Index (RMI) (5/5)



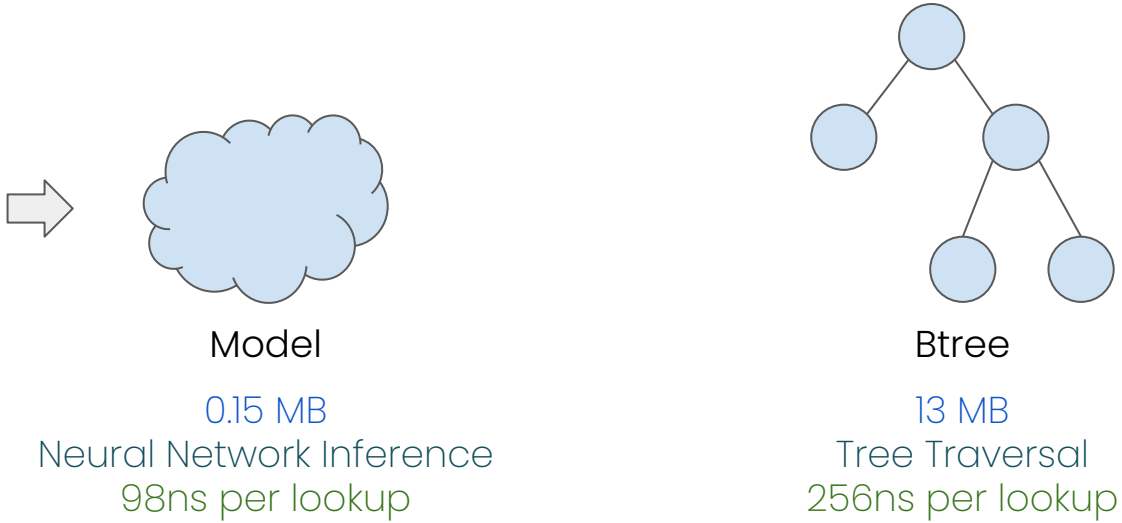


Configurable:
1. Neural Network
2. B-Tree

(*) Kraska et al., "The Case for Learned Index Structures". *SIGMOD 2018.*

Recursive Model Index (RMI)

Key	Value
56	value0
60	value1
68	value2
71	value3
80	value4



(*) Kraska et al., "The Case for Learned Index Structures". *SIGMOD 2018.*

RMI does not work for range matching

- RMI requires to learn offset for each key
- Strawman: enumerate values in a range

Problem 1: to estimate the maximum error it must **scan all keys**

Problem 2: does not work for sparse data - **model gets too large** - no memory savings

Problem 3: How to handle **overlaps**?

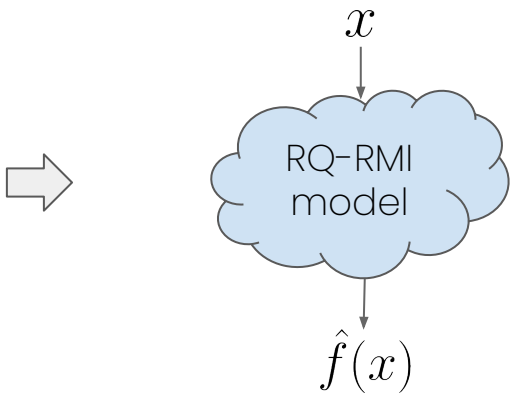
Our work:
Range Query Recursive Model Index: RQ-RMI

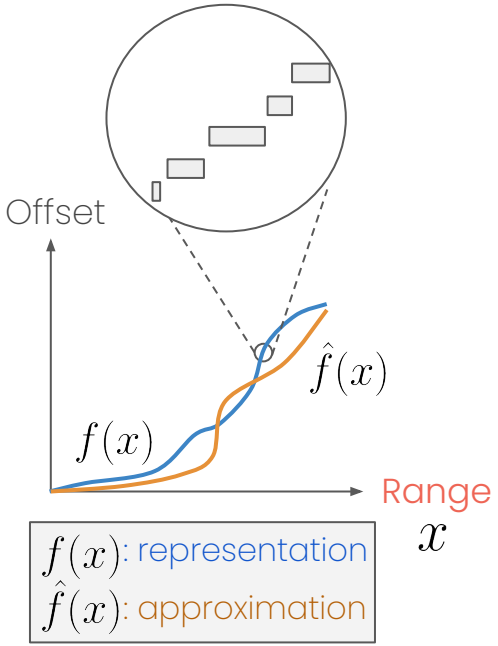
1. Learns **ranges, not keys**
2. Ranges may **overlap**
3. Supports **multi-dimensional ranges**

Assume no overlaps

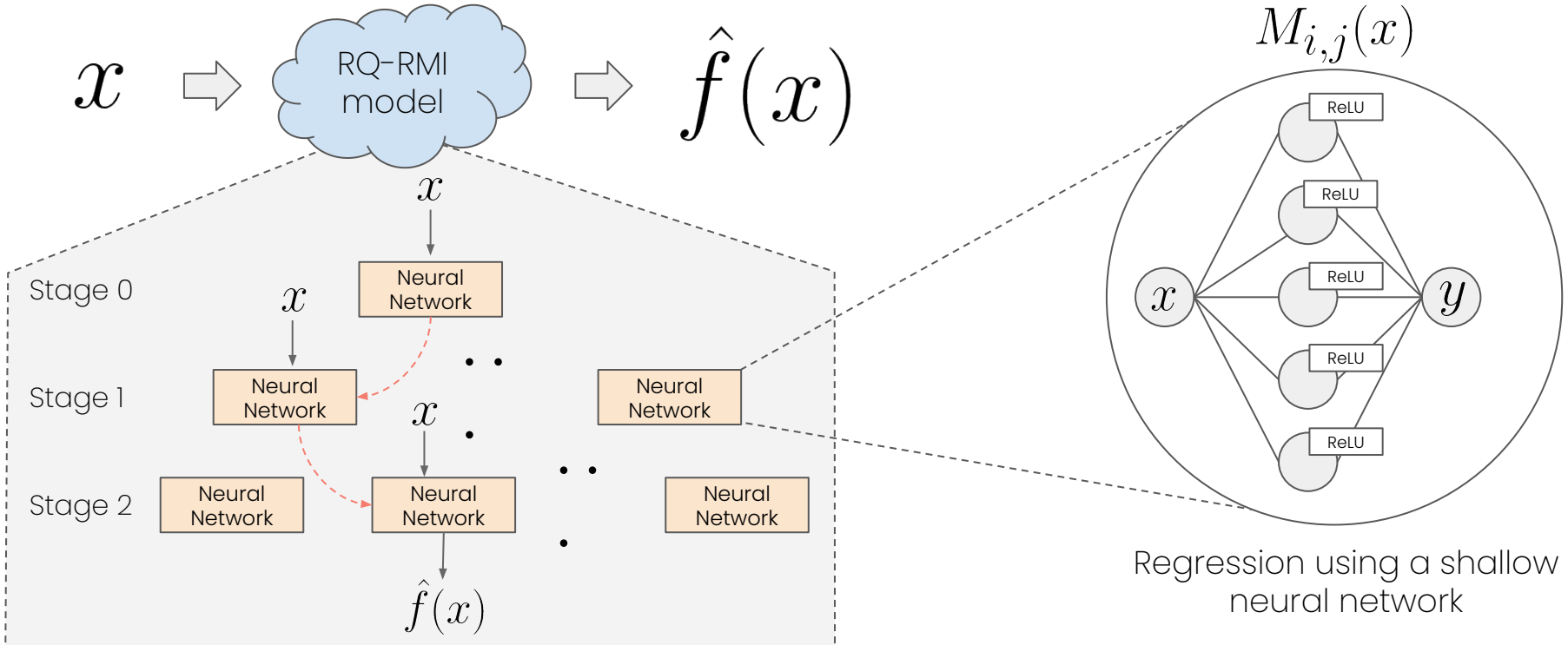
Sample input domain and learn from valid inputs

Range	Value
0-70	600
⋮	⋮
80-104	125



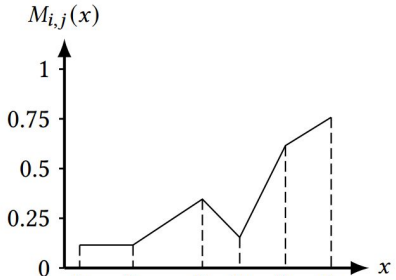


Use MLP as a submodel



Regression using a shallow neural network

How to compute error bound?



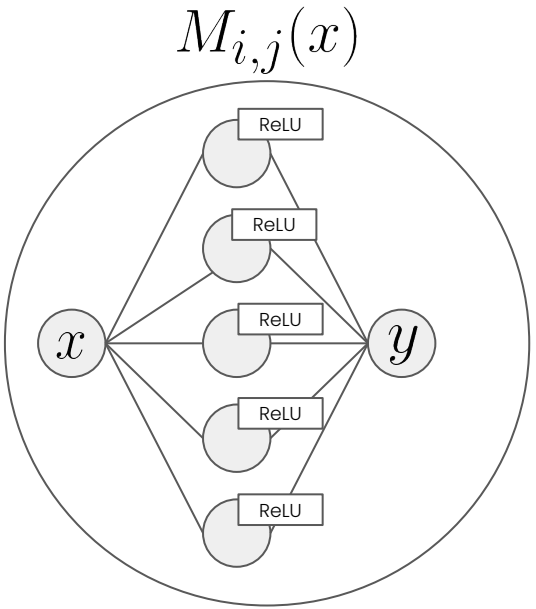
A bounded approximation error
Is guaranteed

$$|f(x) - \hat{f}(x)| < \epsilon$$

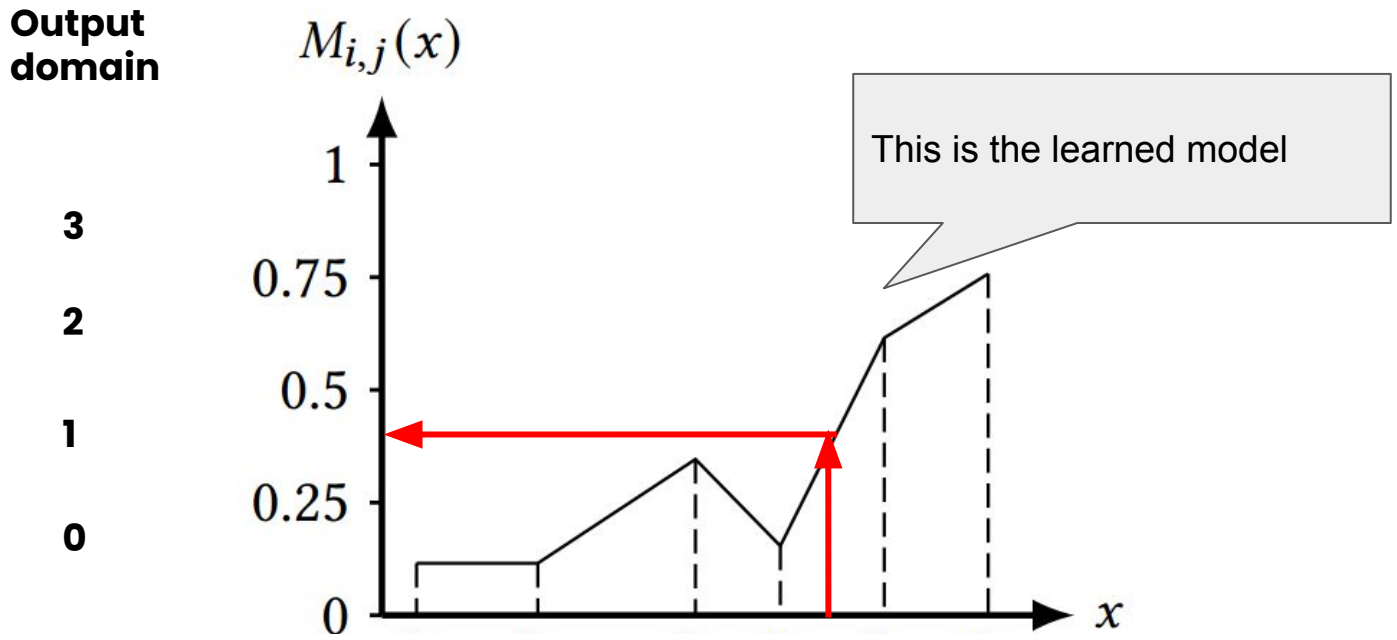
For the entire input domain!

$$\forall x$$

Neural networks with ReLU activations are piecewise linear functions

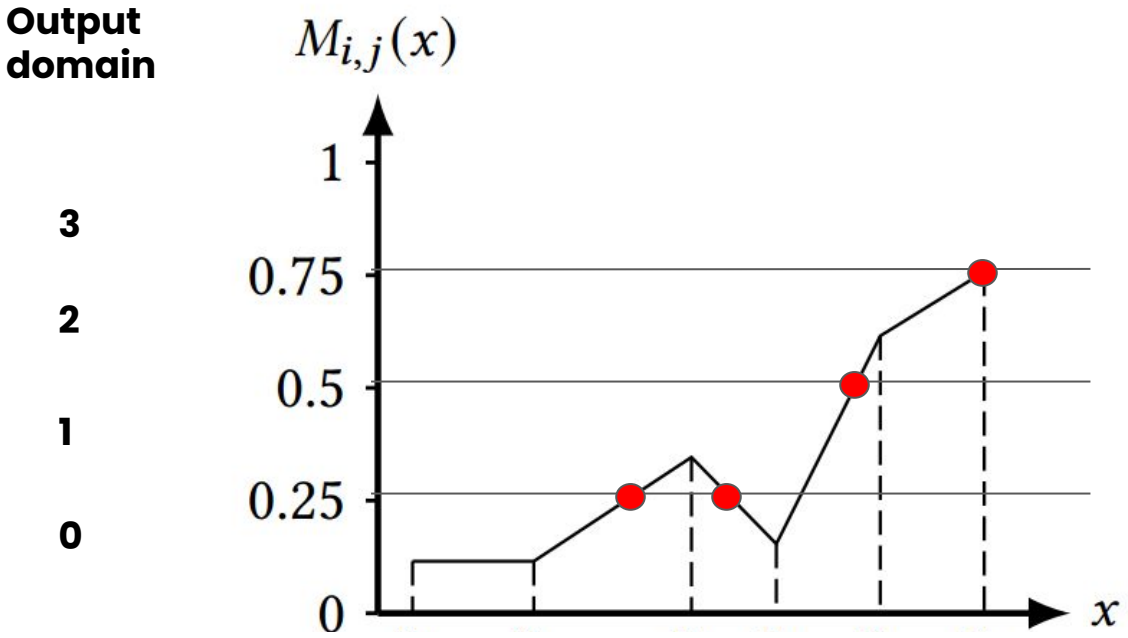


Intuition by example: M(x) learns 4 outputs



Given x - find a prediction for its index.

Observation: maximum error is in vantage points



Error bound can be computed using only a few vantage points

Training Range-Query RMI

- For each layer and each submodel:
- Uniformly sample an **input domain** of a model
- Train on the samples: wider ranges get more samples

- For leaf models: compute error using *vantage* points
- If error is above threshold - add more samples, retrain
- if does not work after few tries - increase the number of submodels in the model and try again

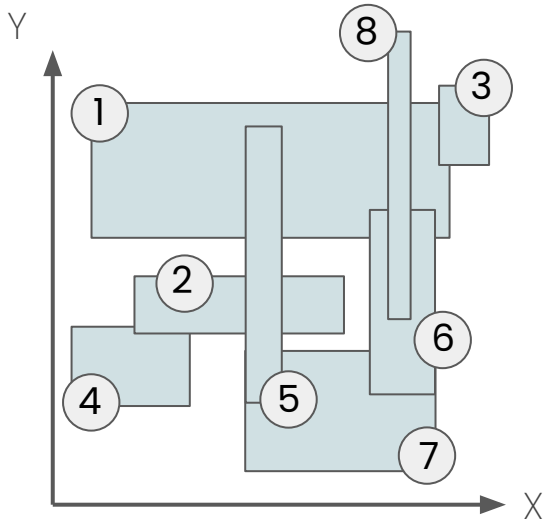
Summary: 1D Range-Query RMI, no overlaps

1. Enables range matching

2. Effective training technique

3. Correctness guarantees

Handling Overlaps + Dimensionality

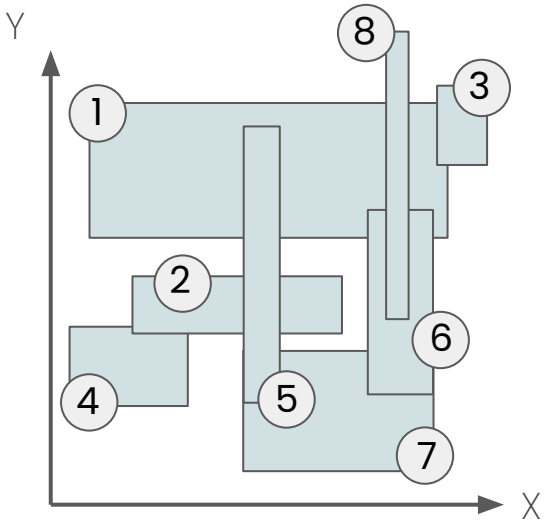


Geometrical representation

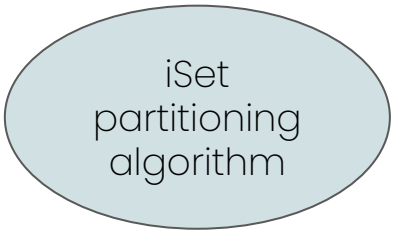
Example: 2D ranges with overlaps

`(2<x<4) AND (3<y<5)`

Handling Overlaps + Dimensionality: iSets (1/5)

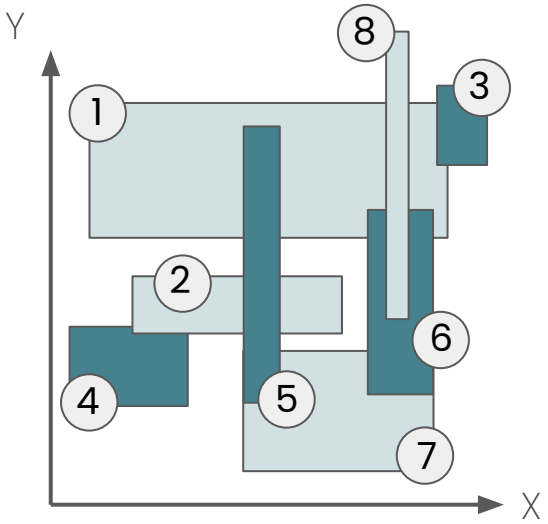


Geometrical representation

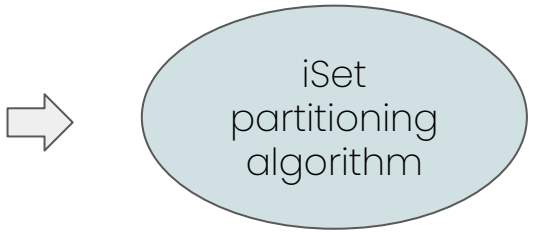


Idea: create multiple sets of non-overlapping ranges, strive to cover as many as possible with fewest sets

Handling Overlaps + Dimensionality: iSets (1/5)



Geometrical representation

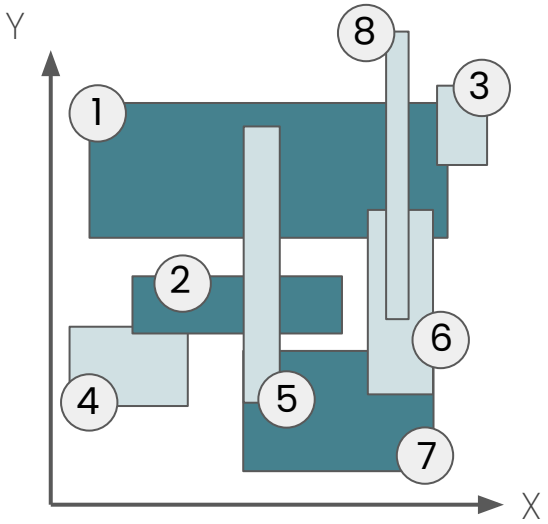


iSet 1 - rules do not overlap on X

{ 4 5 6 3 }

Interval scheduling optimization algorithm

Handling Overlaps + Dimensionality: iSets (1/5)



Geometrical representation

iSet partitioning algorithm

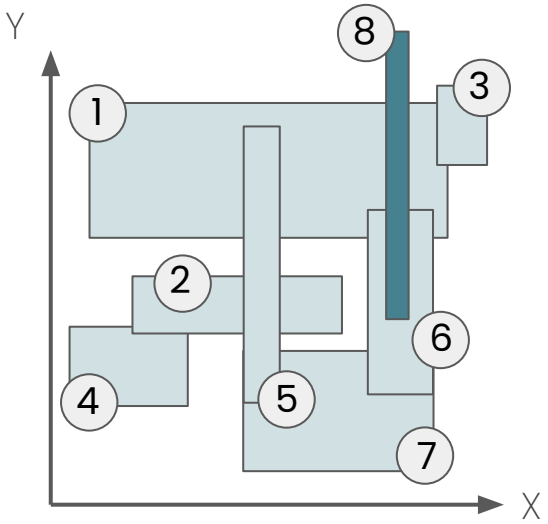
iSet 1 - rules do not overlap on X

{ 4 5 6 3 }

{ 1 2 7 }

iSet 2 - rules do not overlap on Y

Handling Overlaps + Dimensionality: iSets



Geometrical representation

iSet partitioning algorithm

iSet 1 - rules do not overlap on X

{ 4 5 6 3 }

{ 1 2 7 }

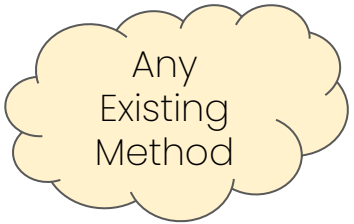
iSet 2 - rules do not overlap on Y

{ 8 }

Remainder - rules that do not fit in any iSet

Handling Overlaps + Dimensionality: iSets

The remainder set is handled by any classification method



iSet 1 - rules do not overlap on X

{ 4 5 6 3 }

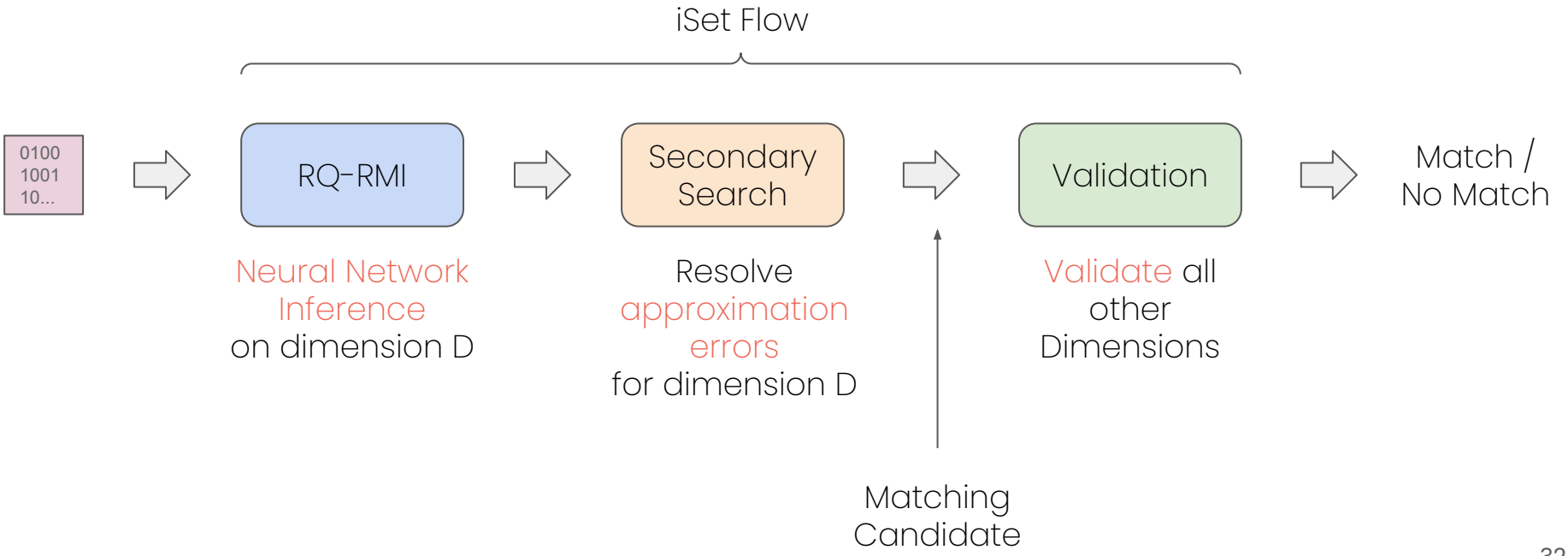
{ 1 2 7 }

iSet 2 - rules do not overlap on Y

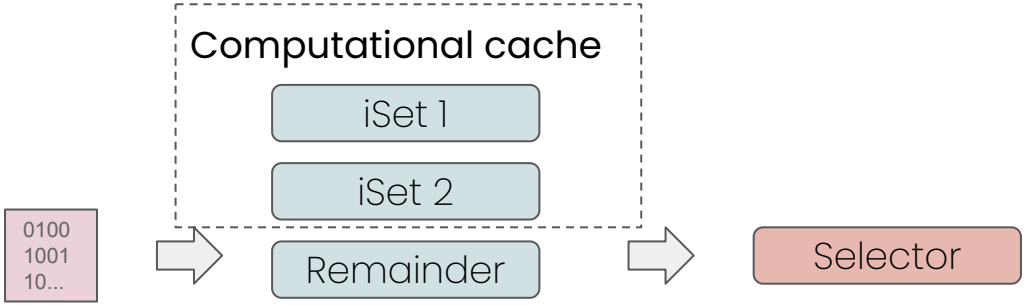
{ 8 }

Remainder - rules that do not fit in any iSet

Handling Overlaps + Dimensionality: iSets



Putting it all together: **computational cache**



In some cases we can eliminate the remainder search if match is found

Summary so far

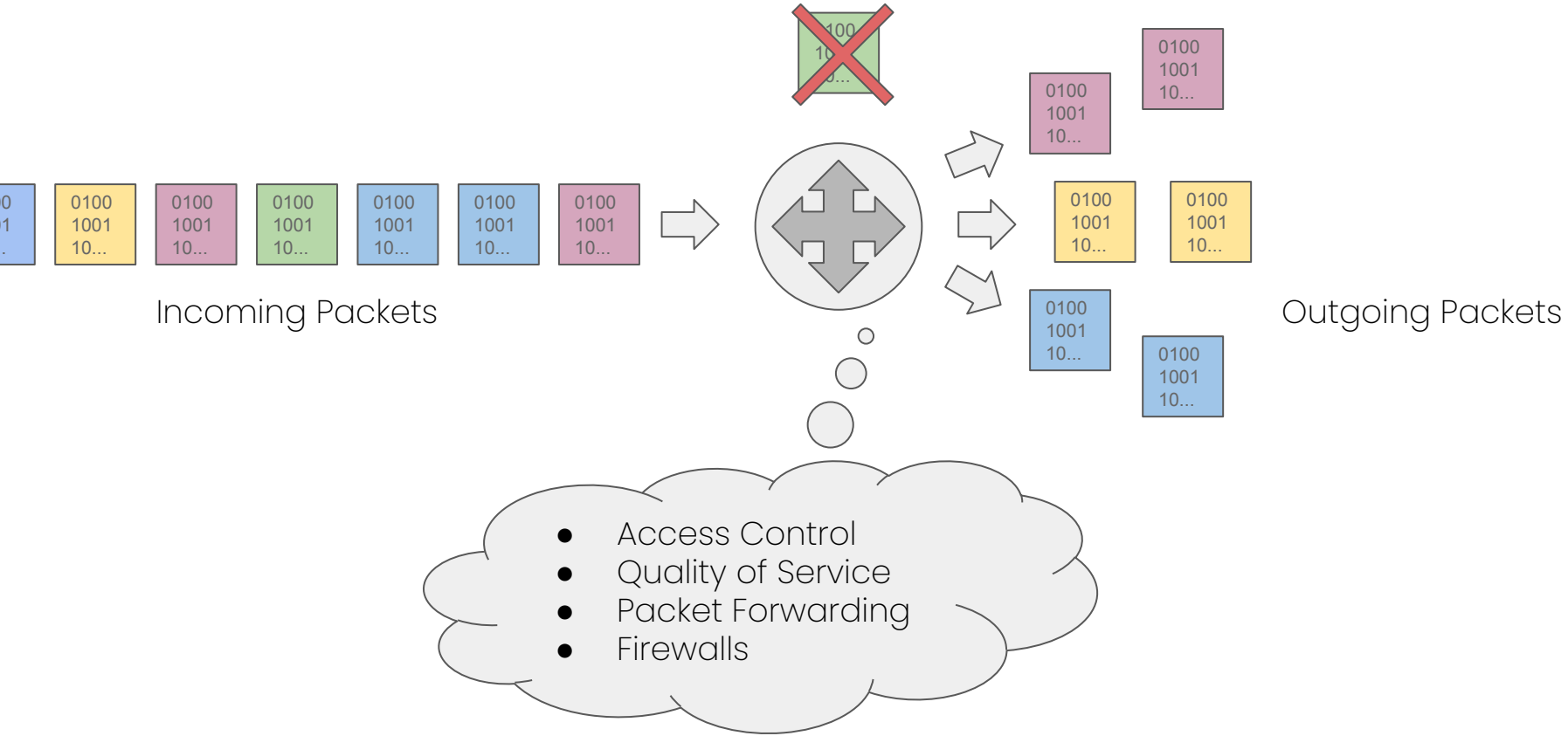
- RQ-RMI learns ranges via sampling
- Offers efficient error estimation
- Deals with overlaps and multi-dimensional cases using remainder

Does it work in practice?

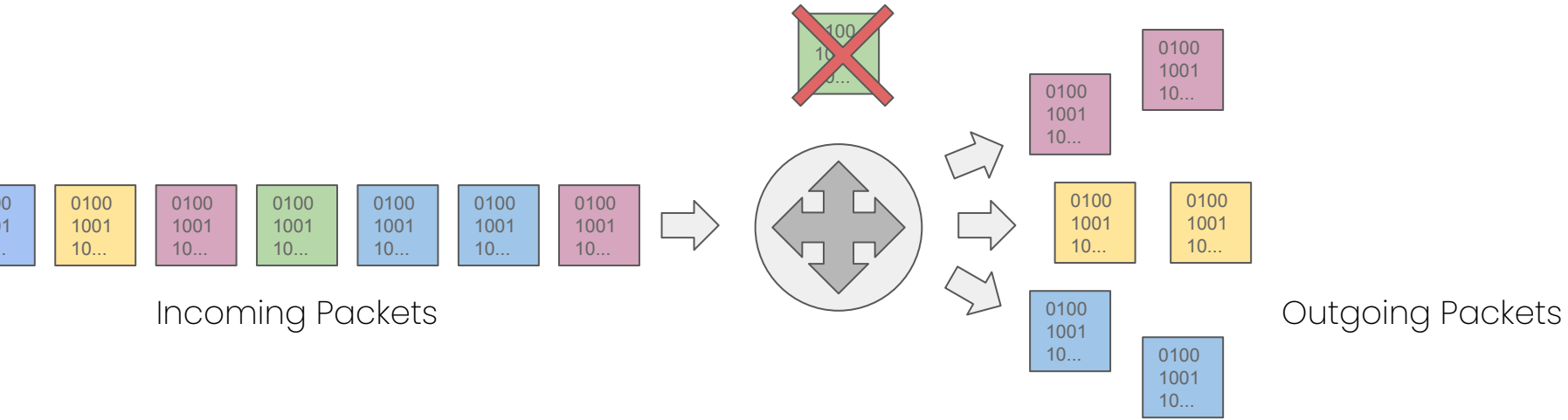
yes!

SIGCOMM'20

A Brief about Packet Classification (1/3)

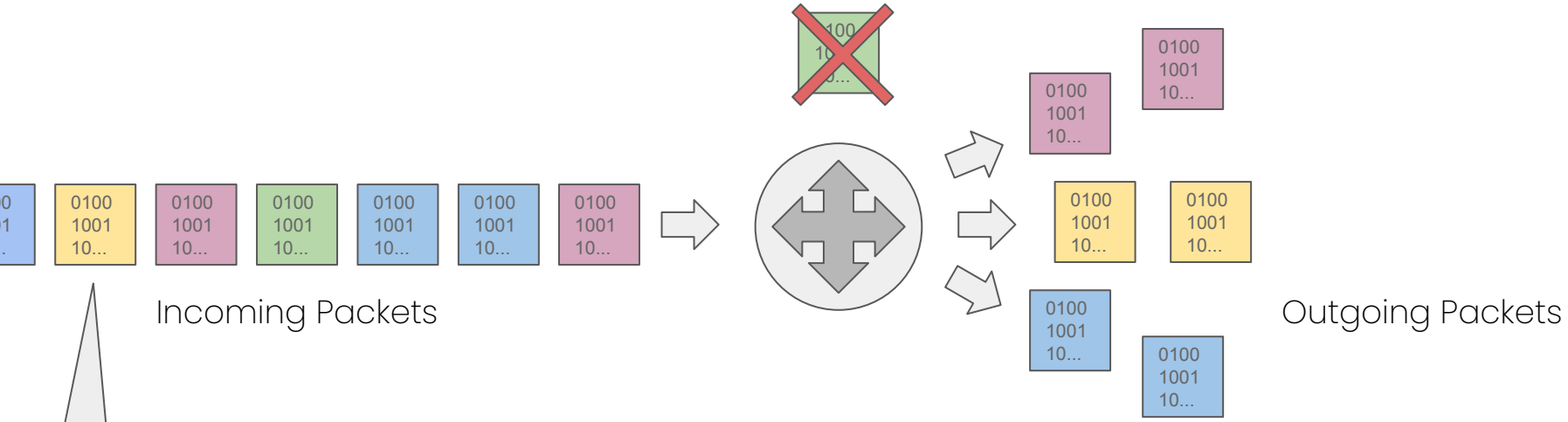


A Brief about Packet Classification (2/3)



Src IP	Dst IP	Src Port	Dst Port	Action	Priority
10.0.10.*	8.8.*.*	0-65535	80	Port 1	3
10.0.20.*	8.8.7.*	0-65535	443	Port 2	2
10.0.*.*	8.8.7.1	0-65535	0-65535	Drop	1

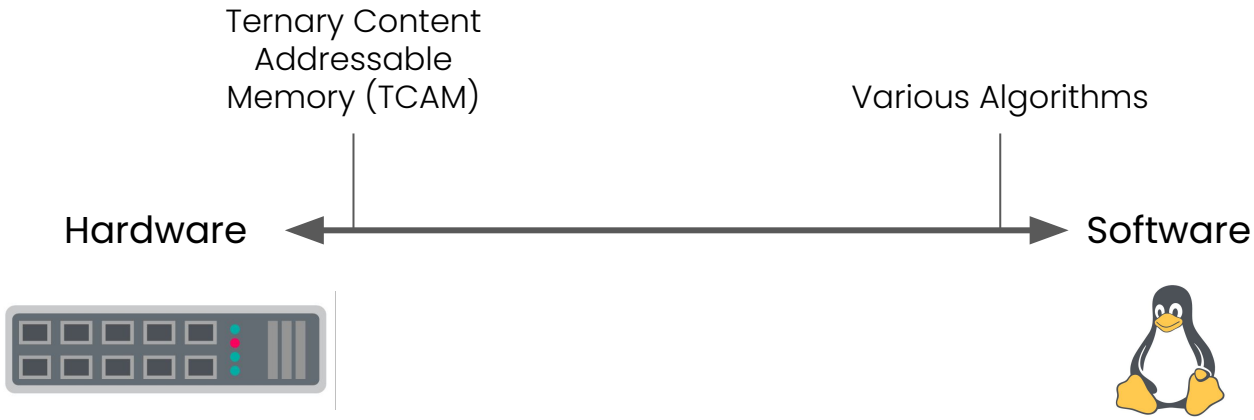
A Brief about Packet Classification (3/3)



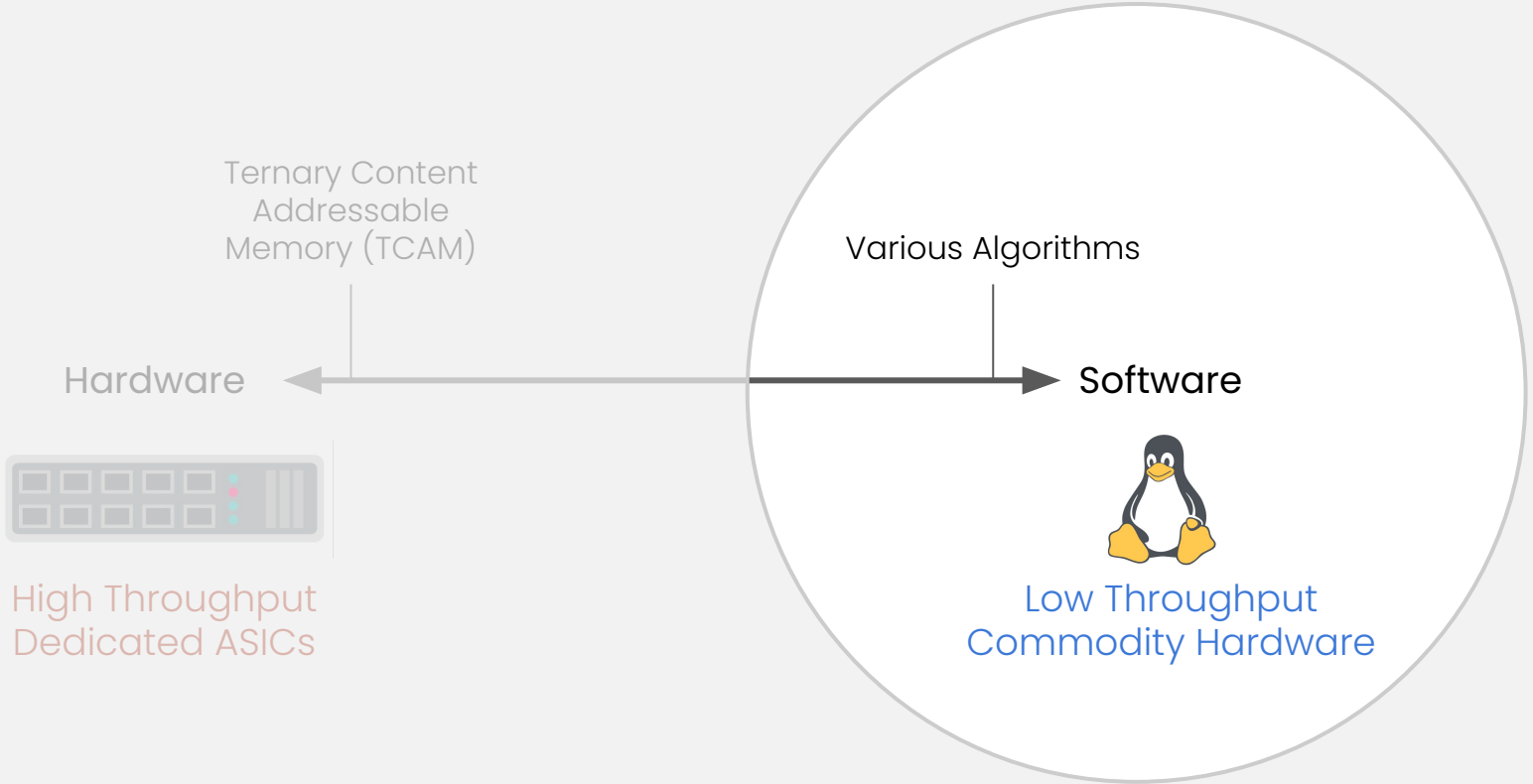
src-ip: 10.0.10.1
dst-ip: 8.8.7.1
src-port: 7870
dst-port: 80

Src IP	Dst IP	Src Port	Dst Port	Action	Priority
10.0.10.*	8.8.*.*	0-65535	80	Port 1	3
10.0.20.*	8.8.7.*	0-65535	443	Port 2	2
10.0.*.*	8.8.7.1	0-65535	0-65535	Drop	1

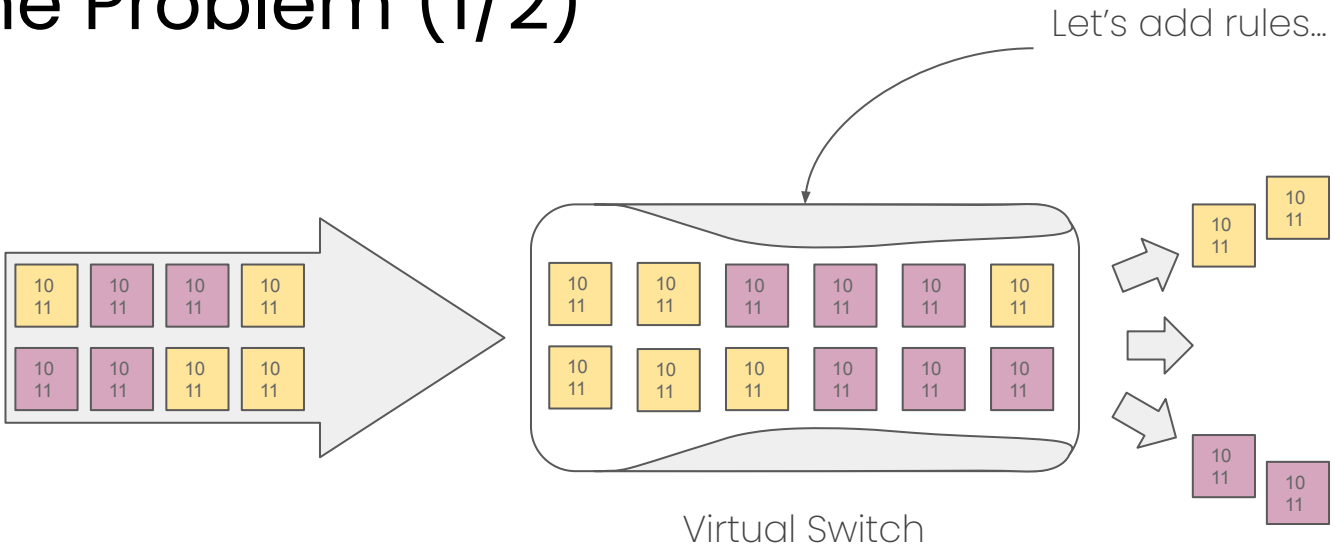
Hardware vs. Software



Hardware vs. Software

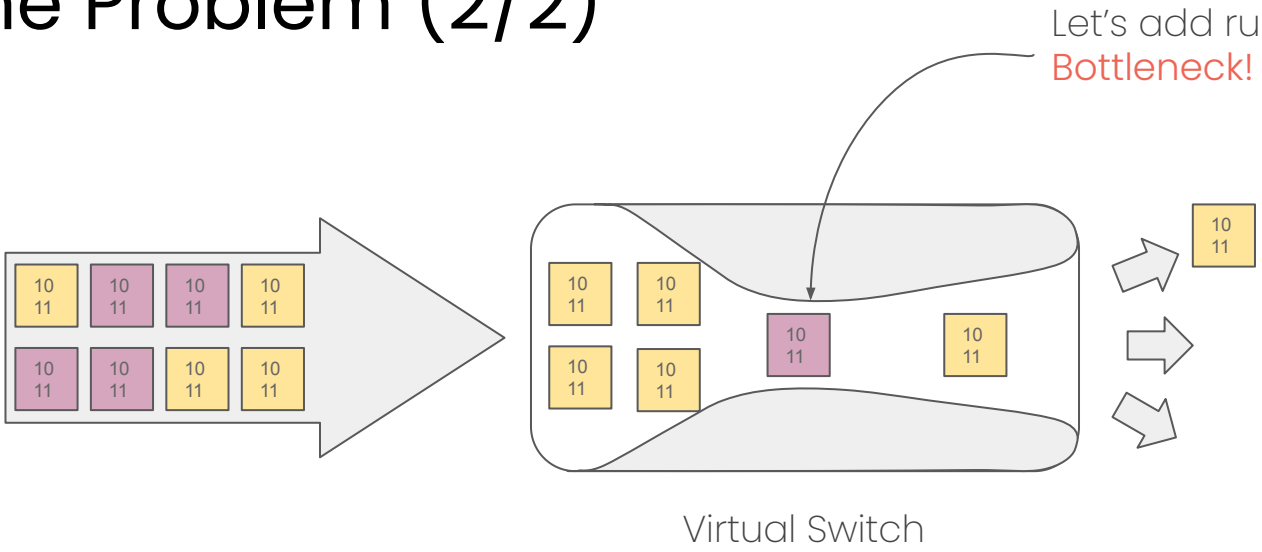


The Problem (1/2)



As we add more rules to virtual switches...

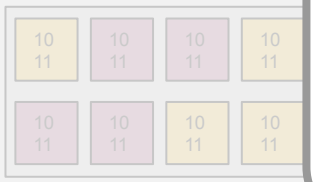
The Problem (2/2)



...we lower their throughput!

The Problem (2/2)

Let's add rules...
Bottleneck!

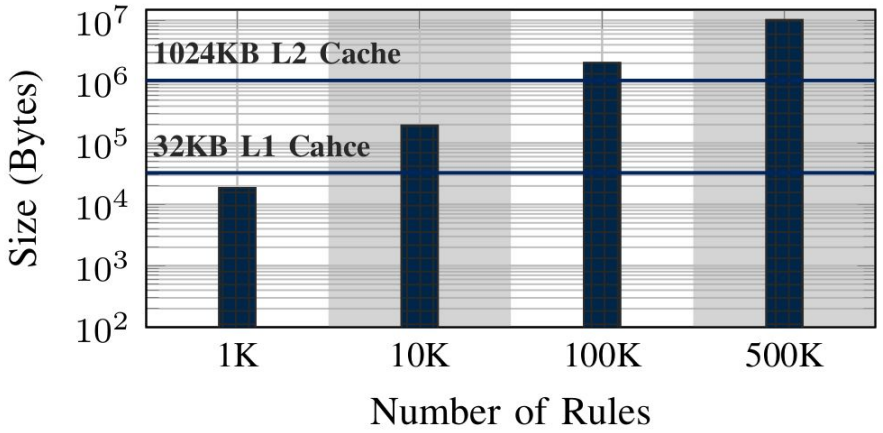


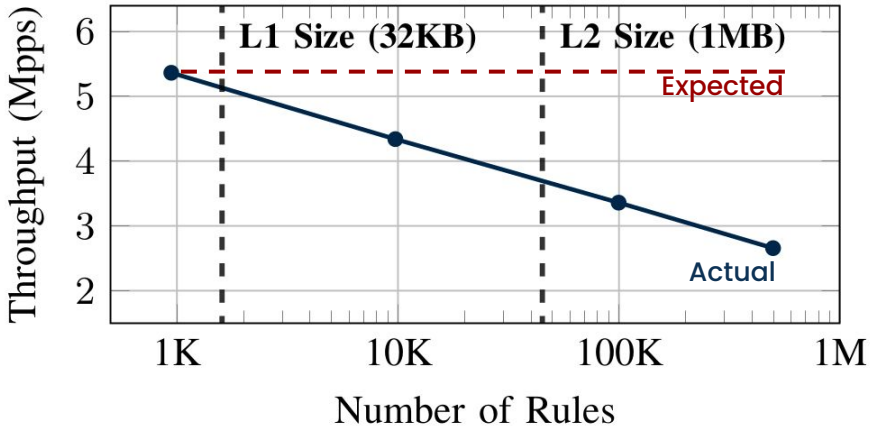
Large Rule Sets
Spill Out of CPU Core Cache!

Virtual Switch

...we lower their throughput!

Large Rule Sets Spill Out of CPU Cache (1/2)

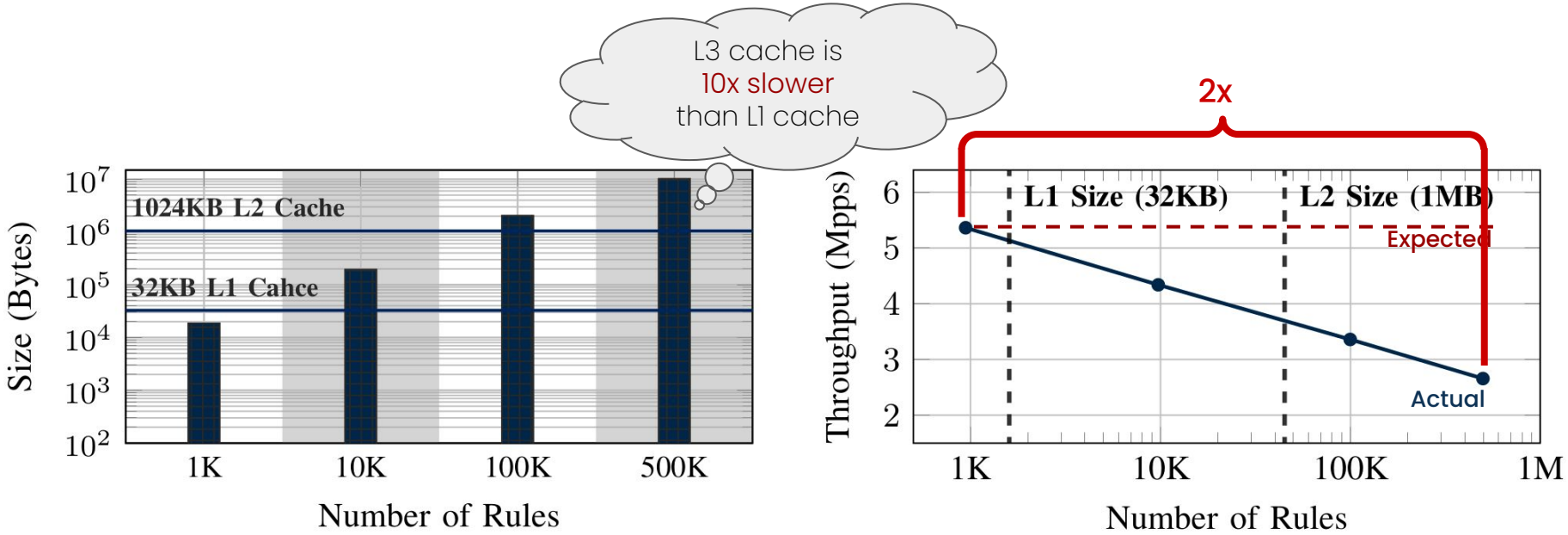




TupleMerge:
State-of-the-art

(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*
(*) Taylor et al., "Classbench: A Packet Classification Benchmark". *TON 2007.*

Large Rule Sets Spill Out of CPU Cache (2/2)

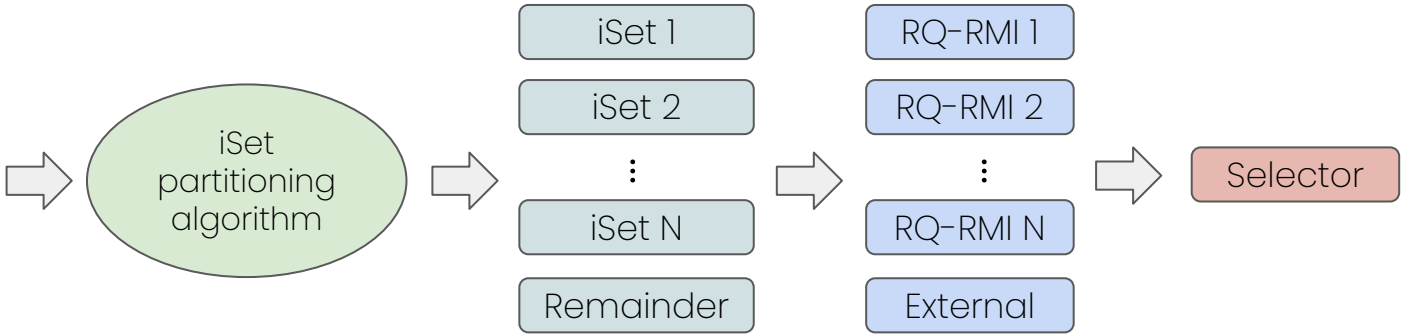


TupleMerge:
State-of-the-art

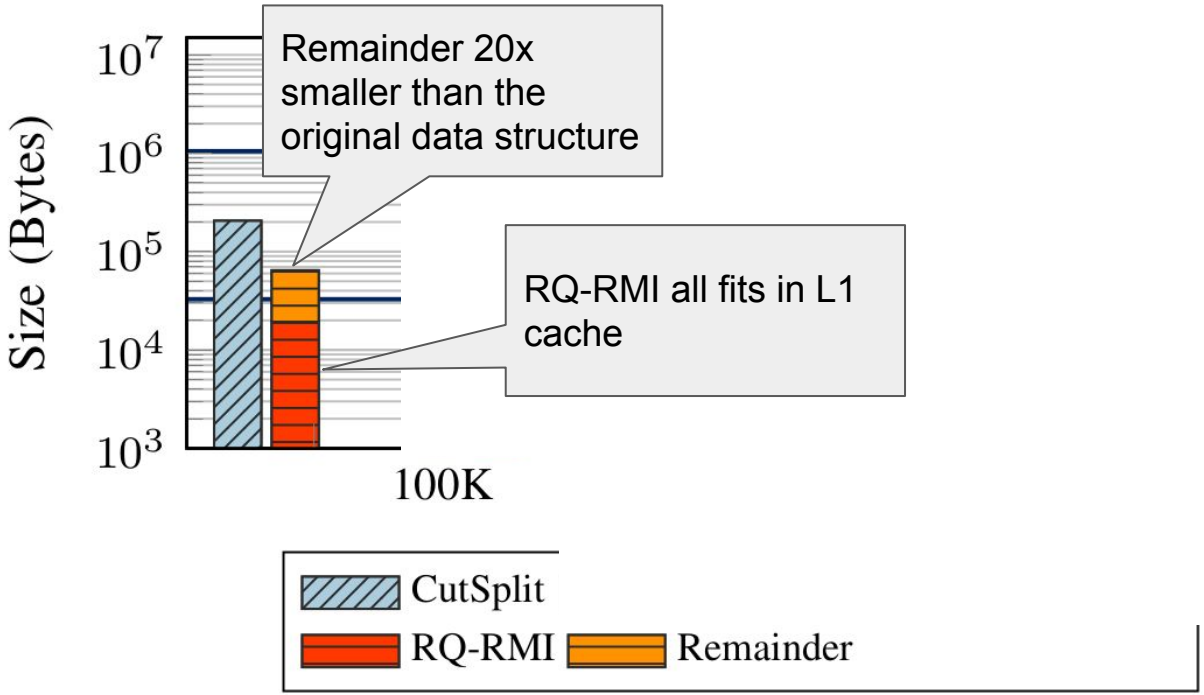
(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*
(*) Taylor et al., "Classbench: A Packet Classification Benchmark". *TON 2007.*

NuevoMatch:
Using RQ-RMI for packet classification

Src IP	Dst IP	Port
10.0.10.*	8.8.*.*	0-80
10.0.20.*	8.8.7.*	443
10.0.*.*	8.8.7.1	*
55.0.0.*	3.3.*.*	22



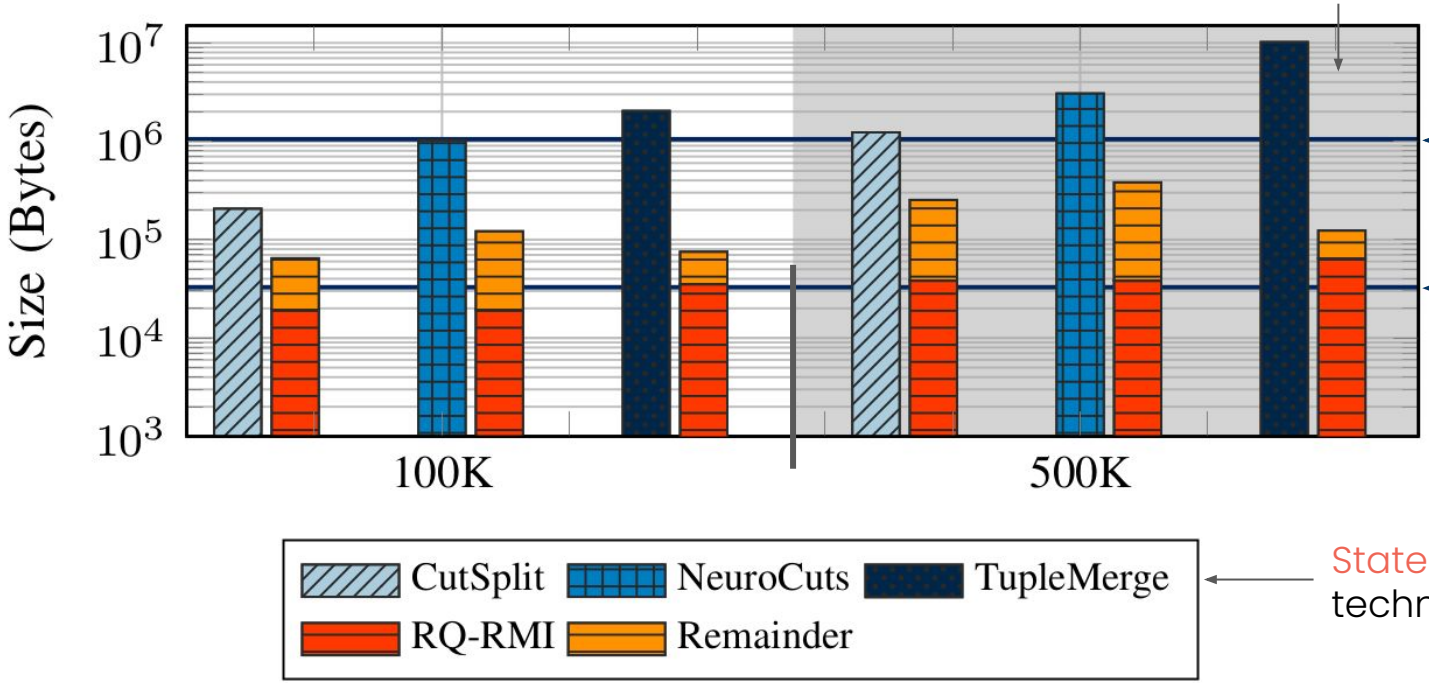
Can We Fit in L2 Cache?



(*) Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". *INFOCOM 2018.*
(*) Liang et al., "Neural Packet Classification". *SIGCOMM 2019.*
(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*
(*) Taylor et al., "Classbench: A Packet Classification Benchmark". *TON 2007.*

Can We Fit in L2 Cache?

Geomean compression factor of 4.9x, 8x, **82x**

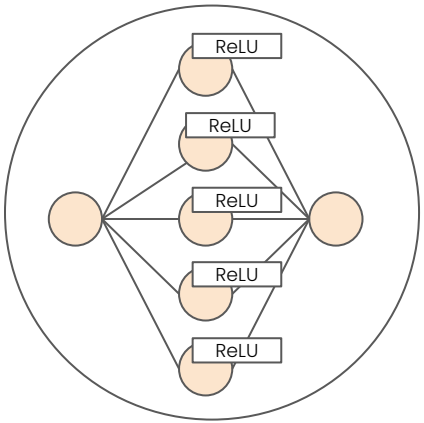


(*) Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". *INFOCOM 2018.*
(*) Liang et al., "Neural Packet Classification". *SIGCOMM 2019.*
(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*
(*) Taylor et al., "Classbench: A Packet Classification Benchmark". *TON 2007.*

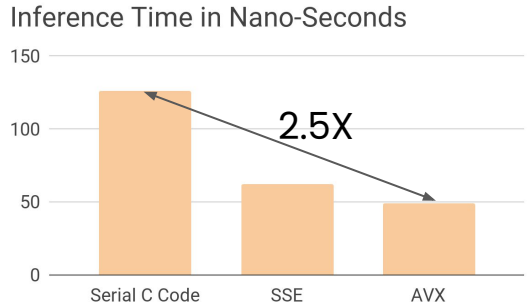
Isn't Neural Network Inference Slow?

Not if you...

Use Small and Shallow Neural Networks



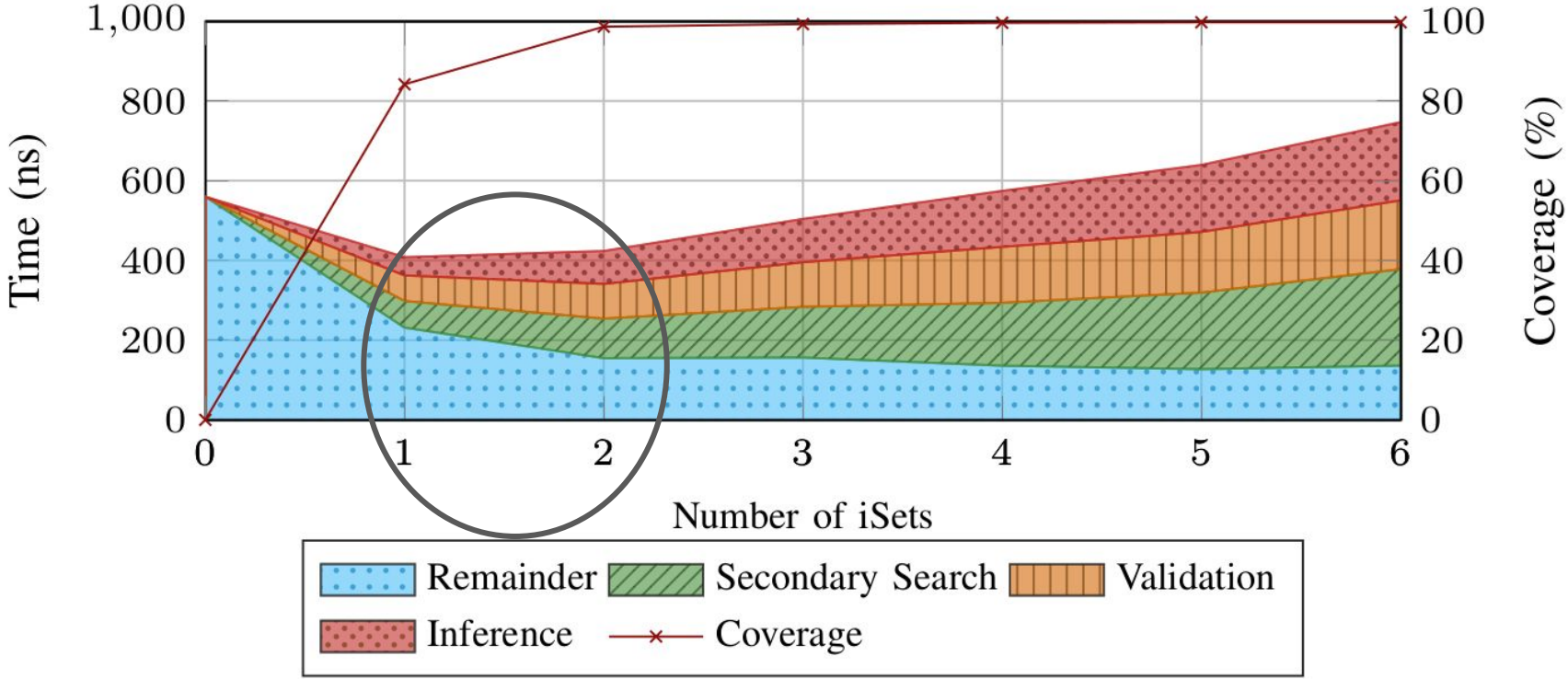
Use Wide Vector Instructions



40-50 ns on a CPU!

The iSet compute-vs-coverage Tradeoff

The iSet compute-vs-coverage Tradeoff



How many iSets needed for high coverage

iSet coverage is high for large rulesets!

Ruleset Size	1 iSet	2 iSets	3 iSets	4 iSets
1K	20.2 ± 18.6	28.9 ± 22.3	34.6 ± 25.6	38.7 ± 27.2
10K	45.1 ± 31.6	59.6 ± 38.9	62.6 ± 37.1	65.1 ± 35.7
100K	80.0 ± 14.5	96.5 ± 8.3	98.1 ± 4.8	98.8 ± 2.7
500K	84.2 ± 10.5	98.8 ± 1.5	99.4 ± 0.6	99.7 ± 0.2
183,376	57.8	91.6	96.5	98.2

How many iSets needed for high coverage

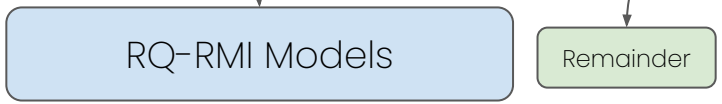
iSet coverage is high for large rulesets!

Ruleset Size	1 iSet	2 iSets	3 iSets	4 iSets
1K	20.2 ± 18.6	28.9 ± 22.3	34.6 ± 25.6	38.7 ± 27.2
10K	45.1 ± 31.6	59.6 ± 38.9	62.6 ± 37.1	65.1 ± 35.7
100K	80.0 ± 14.5	96.5 ± 8.3	98.1 ± 4.8	98.8 ± 2.7
500K	84.2 ± 10.5	98.8 ± 1.5	99.4 ± 0.6	99.7 ± 0.2
183,376	57.8	91.6	96.5	98.2

What About Updates? (1/2)

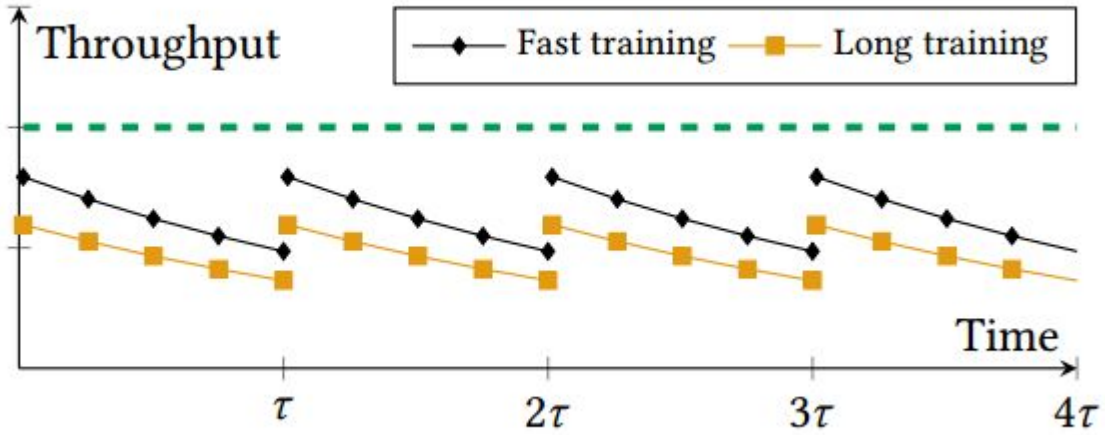
We easily can remove rules from the models

New rules get into the remainder



We retrain the models after a certain threshold in the number of new rules.

Expected performance with periodic training



What About Updates? (2/2)

Training with TensorFlow takes between 10-40 min on a CPU

In our recent work we run at 10ms-1s for the largest datasets

Evaluation: see the paper

End-to-end performance

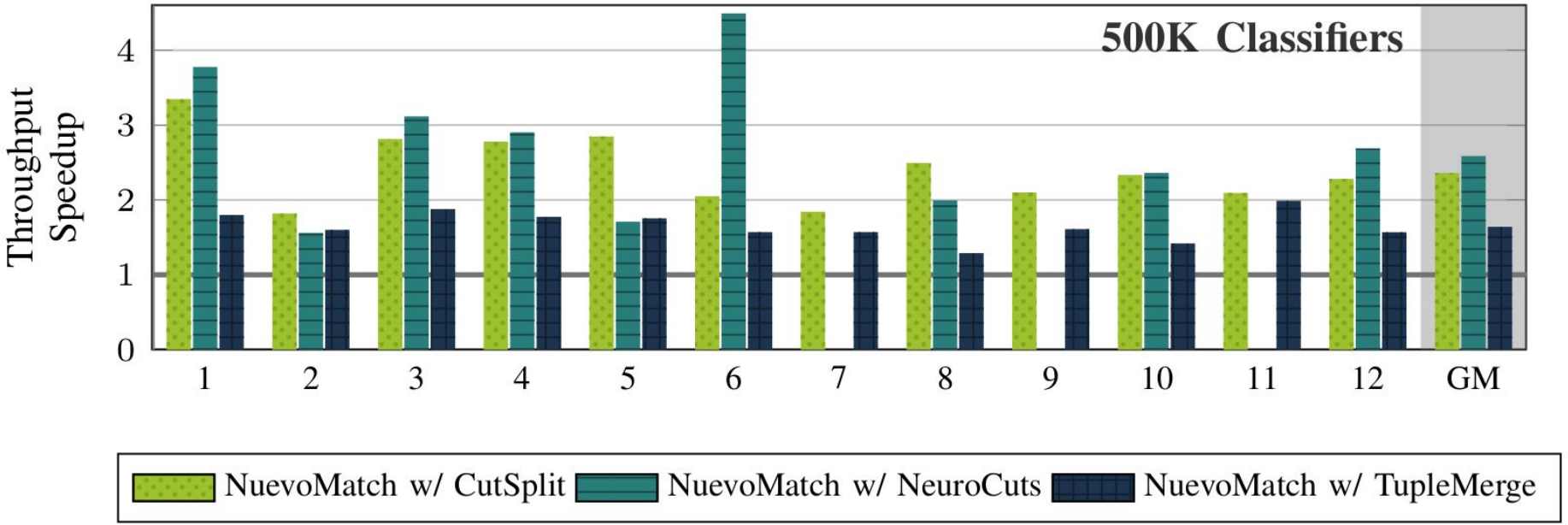
- Single-core vs. multi-core settings
- Small vs. large rule sets
- Traffic with uniform / skewed temporal locality
- Memory footprint comparison
- Performance under L3 cache contention
- Real-world forwarding rules

Performance analysis

- iSet Coverage
- Impact of the number of iSets
- Partitioning effectiveness
- Training time and search bounds
- Performance with many fields

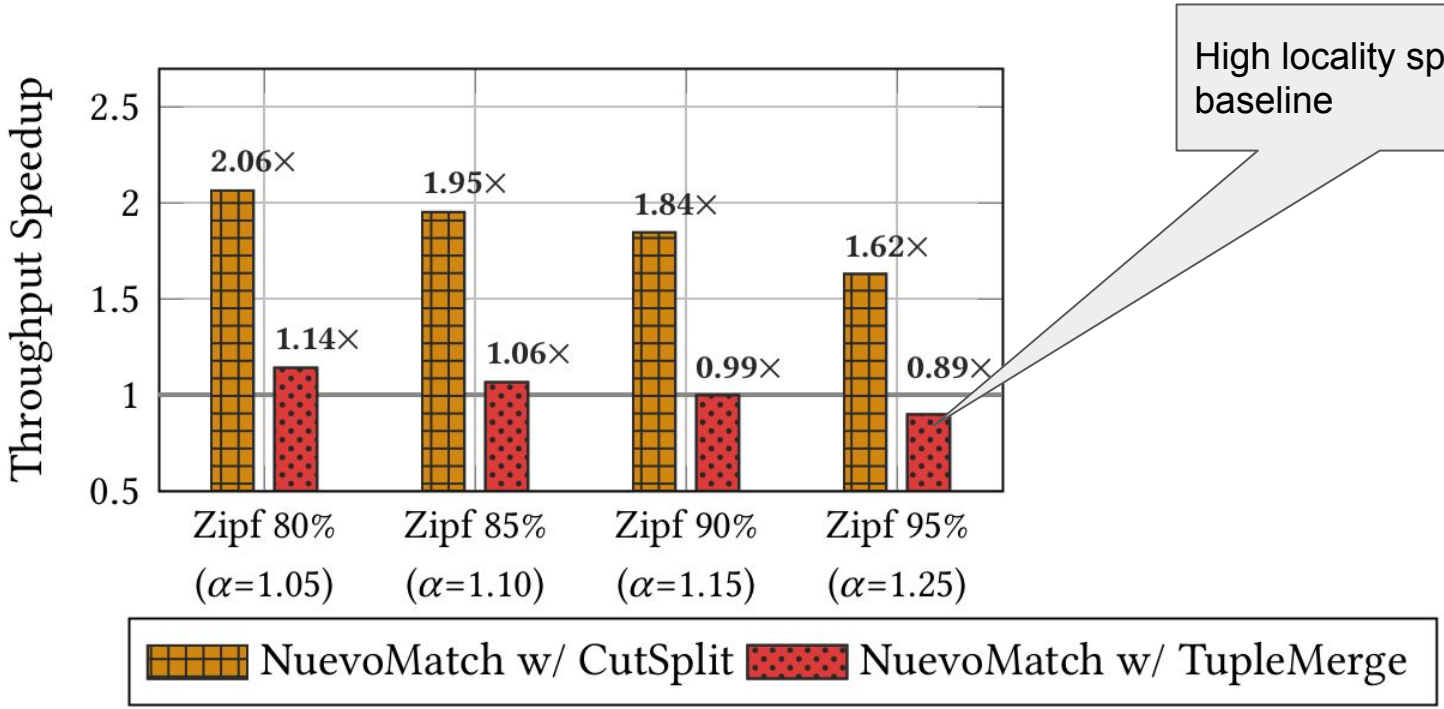
Evaluation - Uniform Access

Geomean throughput speedup of 2.4x, 2.6x, 1.6x



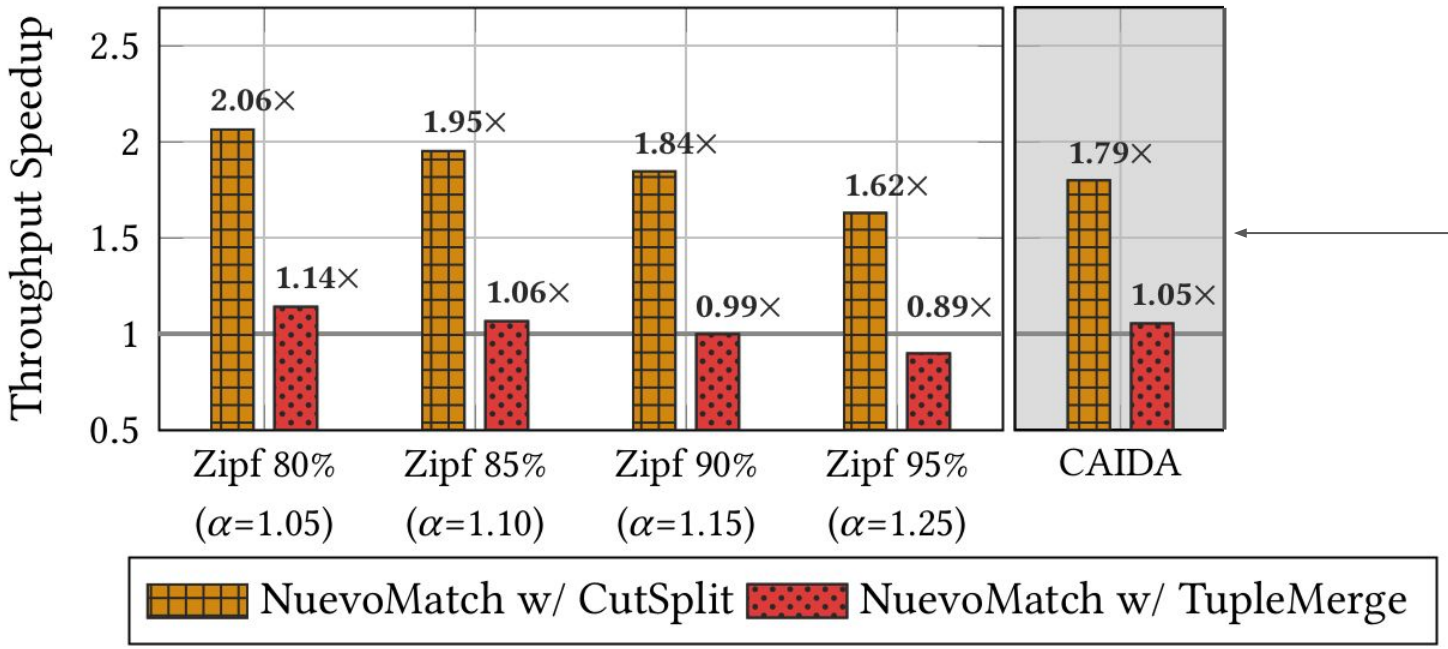
(*) Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". *INFOCOM 2018.*
(*) Liang et al., "Neural Packet Classification". *SIGCOMM 2019.*
(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*
(*) Taylor et al., "Classbench: A Packet Classification Benchmark". *TON 2007.*

Evaluation - Skewed Traces



(*) Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". *INFOCOM 2018.*
(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*
(*) Katta et al., "CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks". *SOSR 2016.*
(*) Taylor et al., "Classbench: A Packet Classification Benchmark". *TON 2007.*
(*) The CAIDA UCSD Anonymized Internet Traces 2019 (www.caida.org/data/passive/passive_dataset.xml).

Evaluation - Skewed Traces



Real-world temporal locality

(*) Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". *INFOCOM 2018.*
(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*
(*) Katta et al., "CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks". *SOSR 2016.*
(*) Taylor et al., "Classbench: A Packet Classification Benchmark". *TON 2007.*
(*) The CAIDA UCSD Anonymized Internet Traces 2019 (www.caida.org/data/passive/passive_dataset.xml).

Agenda

- A computational approach to packet classification
- Ongoing/Future work
  - Faster training + integration with production virtual switches
  - Updatable models
  - Hardware acceleration
  - More applications

OpenV switch integration [under submission]

- 1000x faster training
- From 3X to 25X higher performance on real traces
- Updates: up to 50,000 rules/s
  - Previous versions are slow even with a few

Future work

- Updatable models
  - Efficient updates without retraining
- Hardware acceleration
  - Build hardware accelerator for more application domains
- More applications
  - Handling sparse data structures
  - Variable-size pages in VM
  - In-storage indexing for flash drives
  - Can we get rid of radix trees?
  - In-switch network functions
  - …

Your application here!

Conclusions

1. RQ-RMI for range-value queries
2. NuevoMatch: a new point in the design space of packet classification
3. Promising results using Open vSwitch
4. **First application: packet classification, but more to come!**

Thank You

See More On
https://acsl.group/publications/

Alon Rashelbach
alonrs@campus.technion.ac.il

Ori Rottenstreich
or@technion.ac.il

Mark Silberstein
mark@ee.technion.ac.il