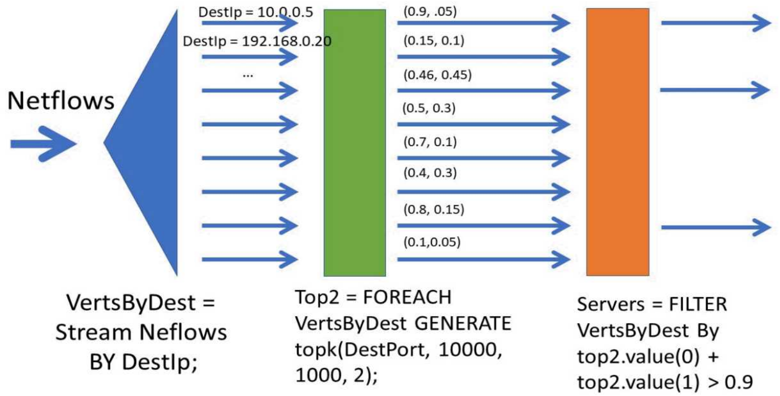
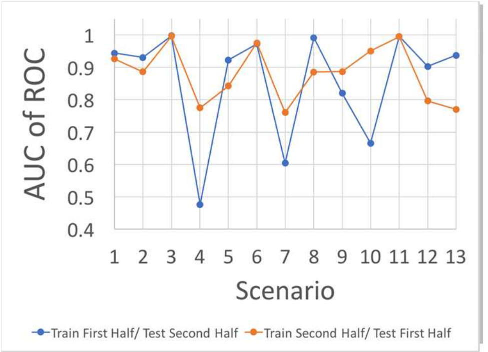
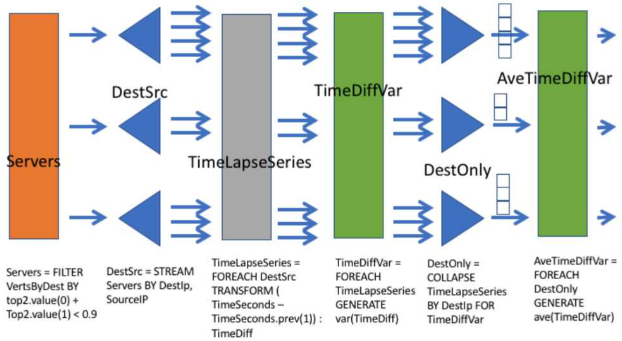
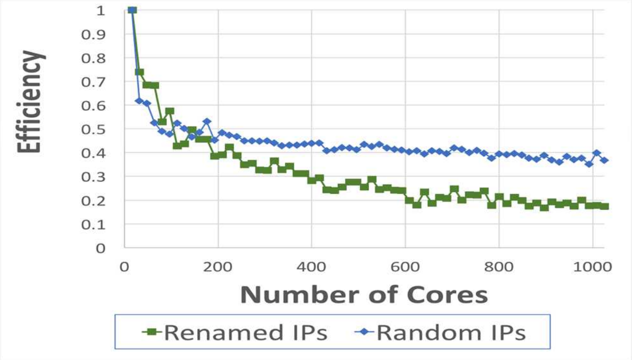
This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2018-8303C




# Streaming Analytics Language for Machine Learning Pipelines

Eric Goodman (9365), Dirk Grunwald

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. SAND NO. 2017-XXXXP

# Highlights

- High-level language for expressing streaming machine learning pipelines
- Parallel implementation
- Example pipeline to detect botnets
- Average AUC over 13 different scenarios: .870
- Scales to 61 nodes/ 976 cores
- 373 thousand netflows per second/ 32.2 billion per day

# Motivation

- Lots of efforts aimed at analyzing cyber data that is done in an ad-hoc fashion
- Commonalities
  - Feature extraction
  - Apply machine learning approaches
- Reduce the amount of time to develop pipelines

# Motivation

- Cyber data is voluminous and comes at high velocity (a fire hose)
- Have streaming algorithms as first class citizens
  - K-medians
  - Frequent Items
  - Mean
  - Quantiles
  - Rarity
  - Variance
  - Vector norms
  - Similarity
  - Count distinct items

# Streaming and Semi-Streaming

- Sliding window model
- Streaming O(polylog n) space requirements
- Semi-streaming O(n polylog n) space requirements

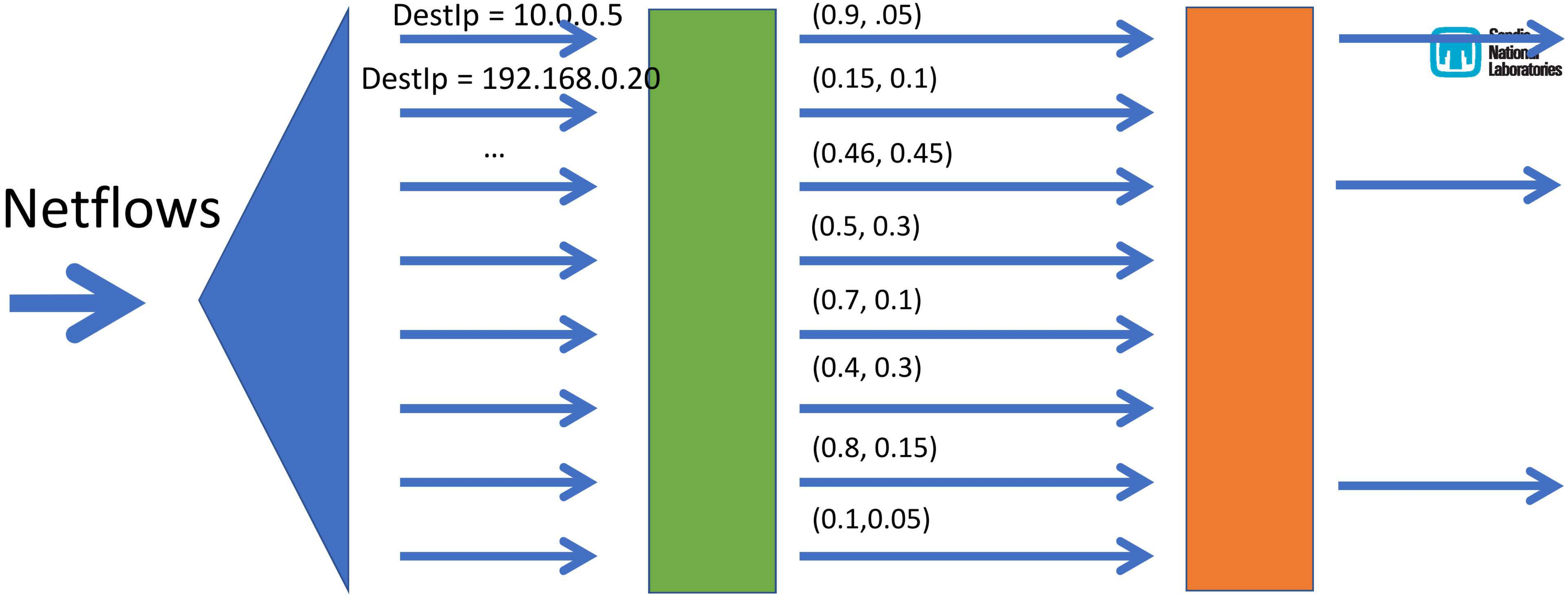
# Case Study: Disclosure

- They want to find command and control servers of botnets
- Several hypotheses about difference in behavior between C&C servers and benign servers

# Getting Servers

- *An IP address belongs to a server if the number of flows directed towards its top two ports (i.e., the two that receives the most connections) account for at least 90% of the flows towards that address.*

```
Netflows = FlowStream("localhost", 9999);
VertsByDest = STREAM Netflows BY DestIp;
Top2 = FOREACH VertsByDest GENERATE topk(DestPort, 10000, 1000, 2);
Servers = FILTER VertsByDest By top2.value(0) + top2.value(1) > 0.9
```




VertsByDest =
Stream Neflows
BY DestIp;

Top2 = FOREACH
VertsByDest GENERATE
topk(DestPort, 10000,
1000, 2);

Servers = FILTER
VertsByDest By
top2.value(0) +
top2.value(1) > 0.9

# Extracting Features

*We extract the mean* $\mu F_{i,j}$ *and standard deviation* $F_{i,j}$ *separately for both incoming and outgoing flows of each server.*

FlowsizeSumIncoming = FOREACH Servers GENERATE ave(SrcTotalBytes);

FlowsizeSumOutgoing = FOREACH Servers GENERATE ave(DestTotalBytes);

FlowsizeVarIncoming = FOREACH Servers GENERATE var(SrcTotalBytes);

FlowsizeVarOutgoing = FOREACH Servers GENERATE var(DestTotalBytes);

# Autocorrelation

*Autocorrelation is widely used for cross-correlating a signal with itself in the signal processing domain, and is useful for identifying repeating patterns in time series data. A series of flow sizes Fi,j can be converted to a time series by ordering sizes by time.*

No semi-streaming approach has been developed for autocorrelation, so not supporting.

# Unique Flow Sizes

- Disclosure counts the number of unique flow sizes

```
UniqueIncoming = FOREACH Servers
GENERATE countdistinct(SrcTotalBytes);
```

```
UniqueOutgoing = FOREACH Servers
GENERATE countdistinct(DestTotalBytes);
```

- They also create an array that provides the count for each unique element.
  - Not exactly expressible in SAL, but can use TopK operator to provide frequency of most frequent unique elements.

# Time Series

*For each server $s_i$ and client $c_j$ , Disclosure prepares a time series $T_{i,j}$ of flows observed during the analysis period. Then, a sequence of flow inter-arrival times $I_{i,j}$ is derived from the time series by taking the difference between consecutive connections;*

```
DestSrc = STREAM Servers BY DestIp , SourceIp ;
TimeLapseSeries = FOREACH DestSrc TRANSFORM ( TimeSeconds −
        TimeSeconds . prev (1)) : TimeDiff
```

# Generate Features on Time Series

TimeDiffVar = FOREACH TimeLapseSeries GENERATE var(TimeDiff);

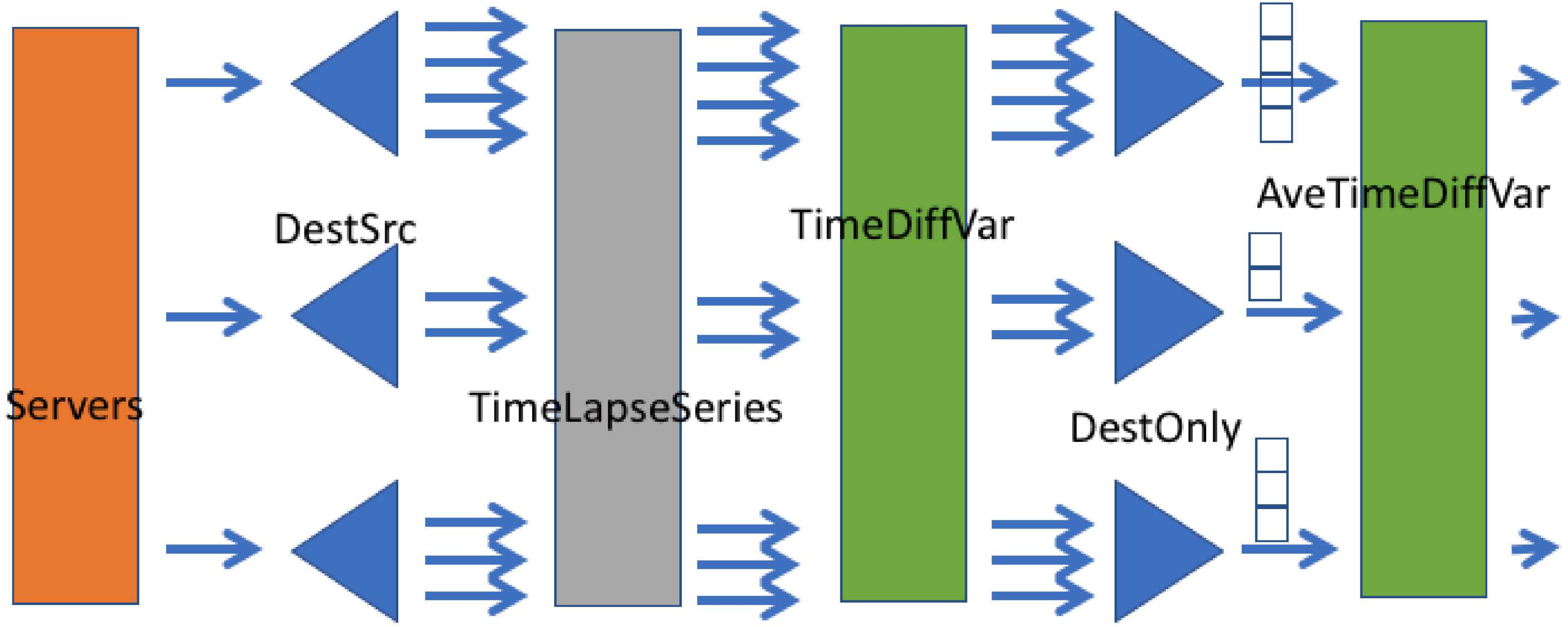
TimeDiffMed = FOREACH TimeLapseSeries GENERATE median(TimeDiff);

DestOnly = COLLAPSE TimeLapseSeries BY DestIp FOR TimeDiffVar , TimeDiffMed ;

AveTimeDiffVar = FOREACH DestOnly GENERATE ave(TimeDiffVar);

VarTimeDiffVar = FOREACH DestOnly GENERATE var(TimeDiffVar);

- Can’t express Max/Min because those require O(n) space requirements.




Servers = FILTER VertsByDest BY top2.value(0) + Top2.value(1) < 0.9

DestSrc = STREAM Servers BY DestIp, SourceIP

TimeLapseSeries = FOREACH DestSrc TRANSFORM ( TimeSeconds – TimeSeconds.prev(1)) : TimeDiff

TimeDiffVar = FOREACH TimeLapseSeries GENERATE var(TimeDiff)

DestOnly = COLLAPSE TimeLapseSeries BY DestIp FOR TimeDiffVar

AveTimeDiffVar = FOREACH DestOnly GENERATE ave(TimeDiffVar)

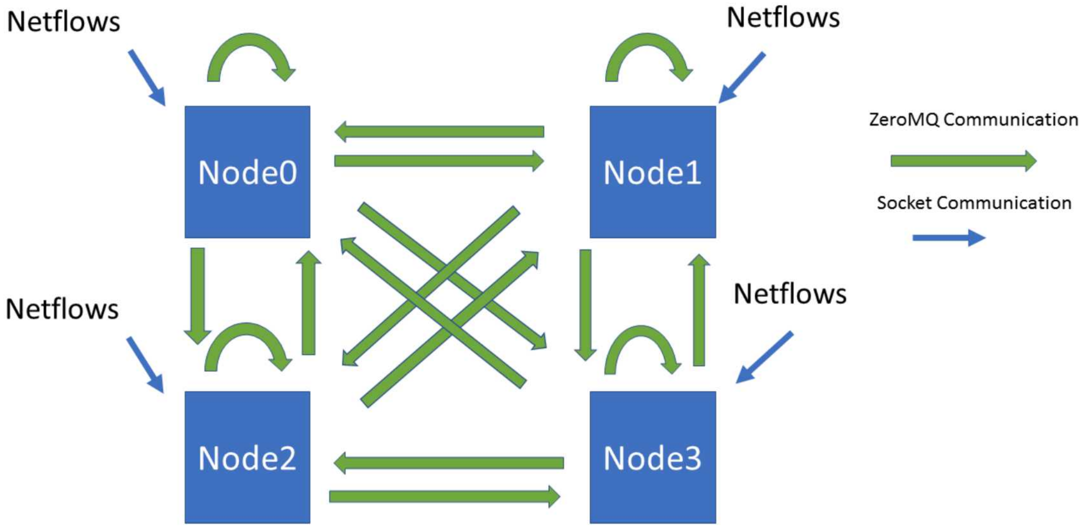
# Case Study Conclusions

- Easy to express machine learning pipeline that was previously implemented in an ad-hoc fashion.
- 10-40 times fewer lines of code

| | C++ | SAL |
|---|---|---|
| Disclosure | 520 | 14 |
| Results Pipeline | 482 | 31 |

# Streaming Analytics Machine (SAM)

- Parallel implementation of SAL in C++
- SAL is converted into a C++ executable using Scala Parser/Combinator
- Each node of SAM receives data using a socket layer
- ZeroMQ is used to distribute the netflows across the cluster
  - Partitioned by IP (common hash function)
  - Push/Pull ZeroMQ sockets are used (lossless)
    - Another option is publish/subscribe (lossy)



# Mapping from SAL to SAM

| C++ Name | Con-sumer | Pro-ducer | Feature Creator | Maps To |
|---|---|---|---|---|
| ReadSocket | | x | | FlowStream |
| ZeroMQPushPull | x | x | | N/A |
| Filter | x | x | | FILTER |
| Transform | x | x | | TRANSFORM |
| CollapsedConsumer | x | | x | COLLAPSE |
| Project | x | | | COLLAPSE |
| EHSum | x | | x | sum |
| EHAve | x | | x | ave |
| EHVariance | x | | x | var |
| TopK | x | | x | topk |

# Results

- Dataset is CTU-13
  - 13 botnet scenarios
  - Real botnet attacks
  - Real background traffic
  - A variety of protocols (IRC, P2P, HTTP) and behaviors (click-fraud, port scans, DDoS, Fast-Flux)

# SAL Program

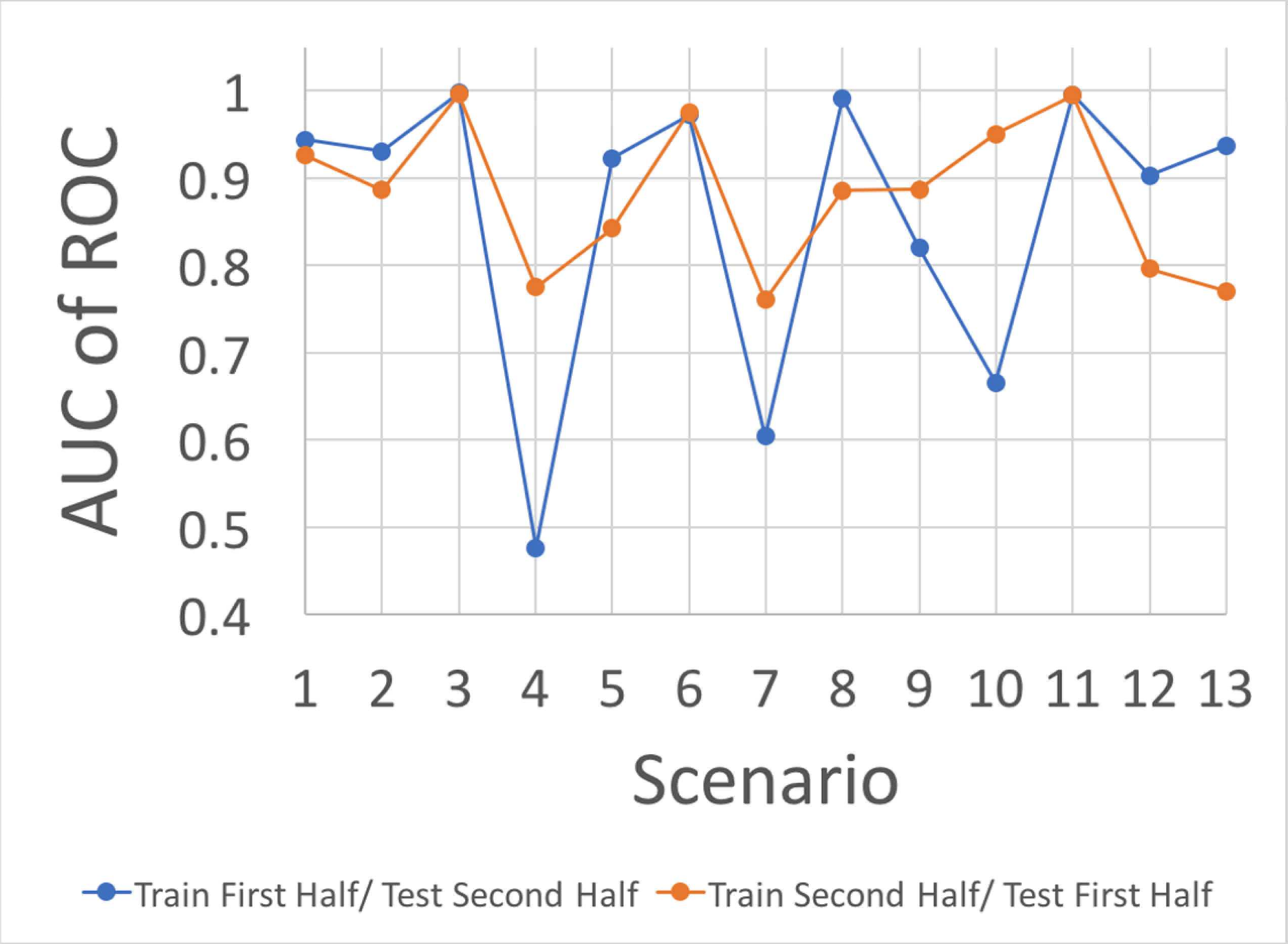
- Average and variance over following features, imuxed on dest IP and source IP

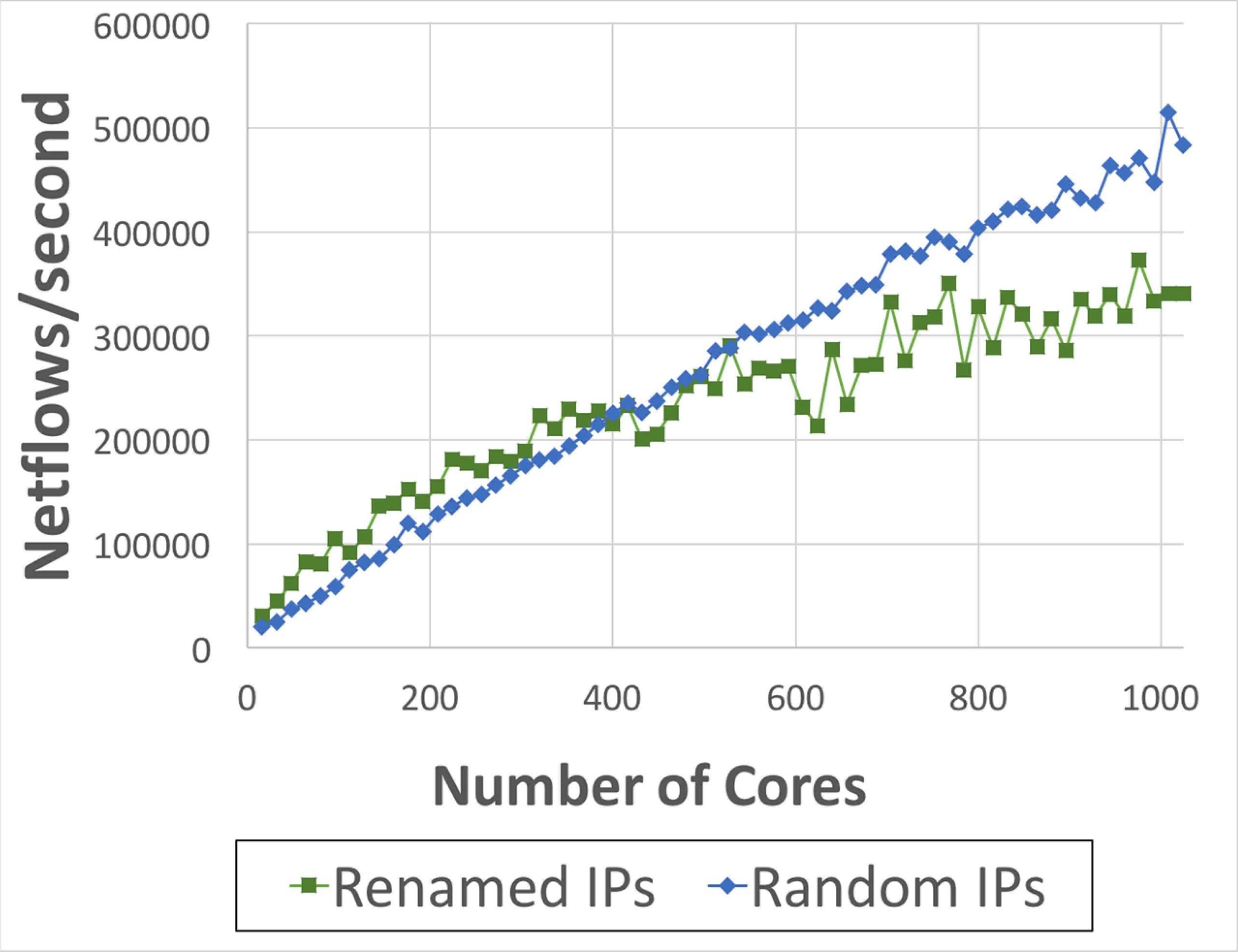
1. SrcTotalBytes
2. DestTotalBytes
3. DurationSeconds
4. SrcPayloadBytes
5. DestPayloadBytes
6. SrcPacketCount
7. DestPacketCount

# Smaller SAL Program

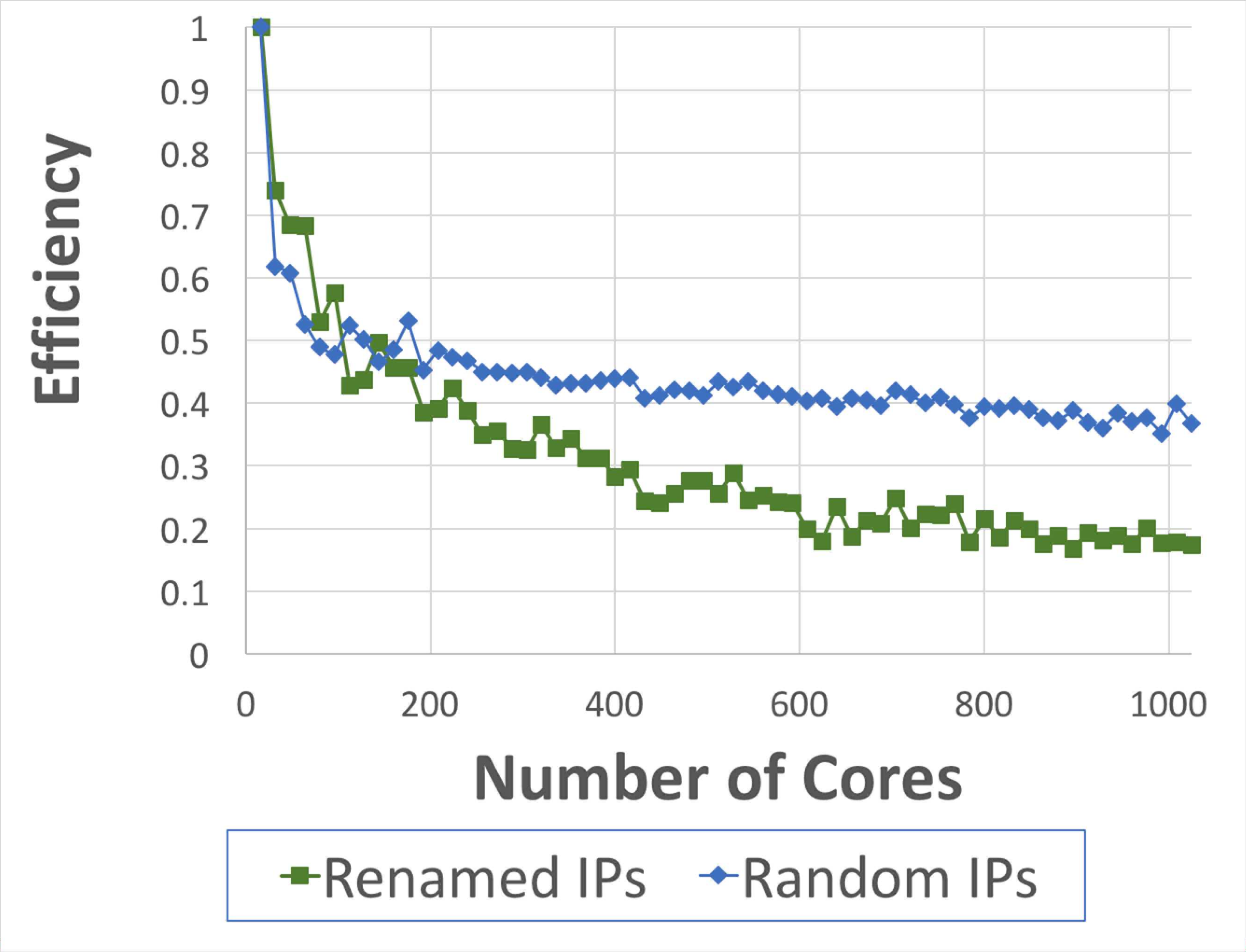
- Filtered down to 8 features using greedy selection approach

1. Average DestPayloadBytes
2. Variance DestPayloadBytes
3. Average DestPacketCount
4. Variance DestPacketCount
5. Average SrcPayloadBytes
6. Average SrcPacketCount
7. Average DestTotalBytes
8. Variance SrcTotalBytes


AUC of ROC
1
0.9
0.8
0.7
0.6
0.5
0.4
1 2 3 4 5 6 7 8 9 10 11 12 13
Scenario
Train First Half/ Test Second Half
Train Second Half/ Test First Half



600000
500000
400000
300000
200000
100000
0
Netflows/second
0
200
400
600
800
1000
Number of Cores
Renamed IPs
Random IPs


Efficiency
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
200
400
600
800
1000
Number of Cores
Renamed IPs
Random IPs

# Conclusions

- SAL provides a succinct way to represent machine learning pipelines
- SAM is a scalable implementation of SAL
  - 373,000 netflows per second
  - 32.2 billion per day
- Example pipeline resulted in an average AUC of ROC of 0.87