Learning Memory Access Patterns

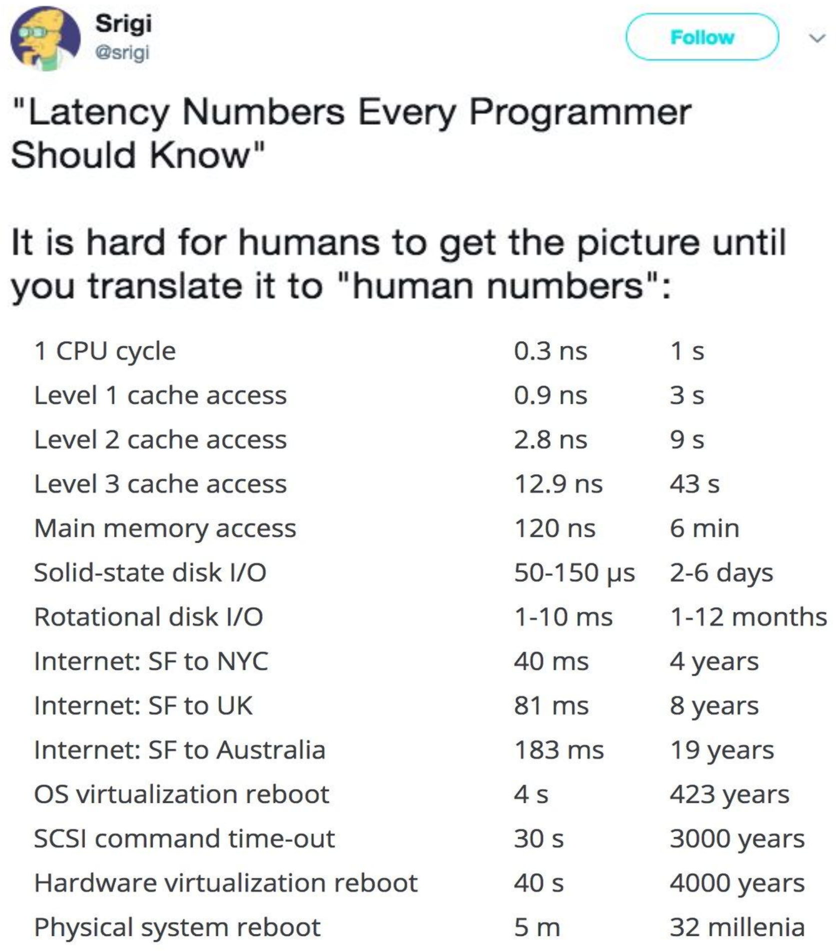
Milad Hashemi, Kevin Swersky, Jamie A. Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, Parthasarathy Ranganathan

MACHINE INTELLIGENCE COMMUNITY

Justin Chen
April 12th, 2018

Motivation

- **Memory wall**
  - Time: prefetch >> compute
- Hierarchical memory systems
- **Prefetching** - process of predicting future memory accesses
- **Prefetchers**
  - Predict when and what data to cache
  - Traditional, handcrafted hardware prefetchers use tables
  - Hard to scale



Srigi
@srigi

"Latency Numbers Every Programmer Should Know"

It is hard for humans to get the picture until you translate it to "human numbers":

1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access	120 ns	6 min
Solid-state disk I/O	50-150 μs	2-6 days
Rotational disk I/O	1-10 ms	1-12 months
Internet: SF to NYC	40 ms	4 years
Internet: SF to UK	81 ms	8 years
Internet: SF to Australia	183 ms	19 years
OS virtualization reboot	4 s	423 years
SCSI command time-out	30 s	3000 years
Hardware virtualization reboot	40 s	4000 years
Physical system reboot	5 m	32 millenia

https://twitter.com/srigi/status/917998817051541504

Prefetching as a Prediction Problem

- Memory instructions generate memory addresses
- Hardware prefetcher features:
  - **Program Counters (PC)**
    - Uniquely identifies register in CPU
    - Registers contain the address of currently executing instruction
    - Predictive of control flow
  - **Cache Miss Addresses**
    - Can use to predict address to prefetch

Traditional Data Prefetchers

- **Stride Prefetchers**
  - Calculates a linear model from repeating patterns to predict access patterns
  - e.g. (0,4,8,12) -> (16,20,24)
- **Correlation Prefetchers**
  - Predict irregular patterns using a table of **previous access patterns**
    - Future events are expected to correlate with past
      - Main motivation for techniques introduced in this work

Application Address Space Data

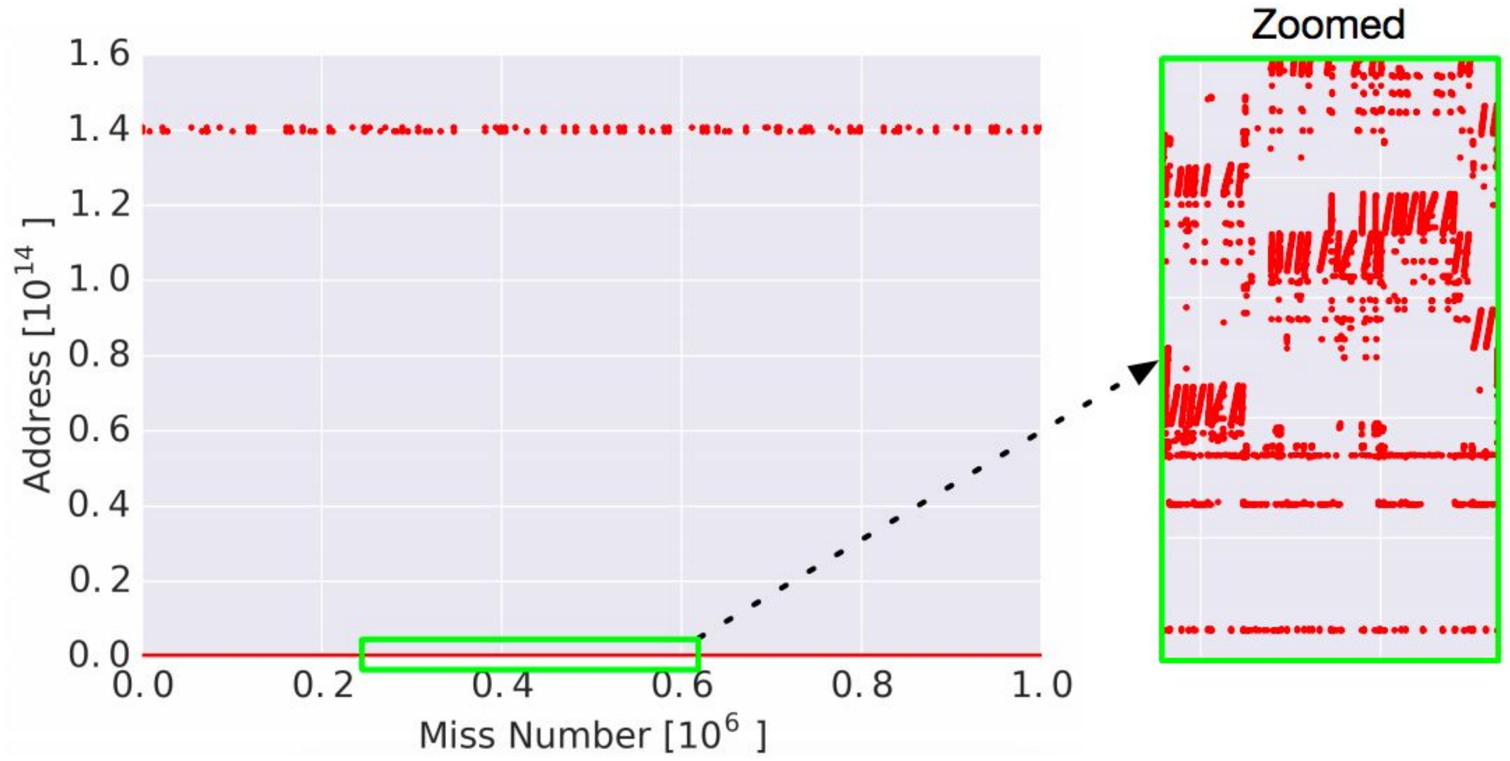
- ***omnetpp*** - standard *SPEC CPU2006* benchmark for simulating communication networks

○ Data:	cache miss addresses
○ Size of training set:	O(100M)
○ Unique points:	O(10M)
○ Size of entire address space:	2^{64}

- Problems:
  - Exceptionally **large range** of values
  - **Sparse** feature space
  - **Multimodal**

https://www.spec.org/cpu2006/Docs/471.omnetpp.html

Omnetpp Cache Miss Sparse Access Pattern



Prefetching as Classification

- **Regression** models suffer from sparsity, variance, and multimodality
- Solution:
  - **Discretization** - discrete abstraction of continuous space
    - Reduces possible values
    - Use a classification model in lieu of regression
    - Strategies:
      1. **Common Addresses**
      2. **Clustering over Address Space**

Prefetching as Classification

- **Address Space Layout Randomization (ASLR)**
  - Different set of addresses used for each execution
- Solution:
  - Predict **deltas**
  - Invariant to ASLR
  - Space of deltas << space of addresses

$$\Delta_N = Addr_{N+1} - Addr_N$$

Notes: In the paper, they use subscript N to denote an intermediate/arbitrary timestep. In this presentation, intermediate/arbitrary timesteps are denoted with subscript t, as they should be. :p

Recurrent Neural Network (RNN)

- Motivation:
  - RNN achieve state-of-the-art in sequence modeling
  - Model conditional probability distributions
  - Able to store memory and compute on its memory
  - Trained offline

$$S_t = \sigma(U_t x_t + W_t S_{t-1})$$

$$o_t = V_t S_t$$

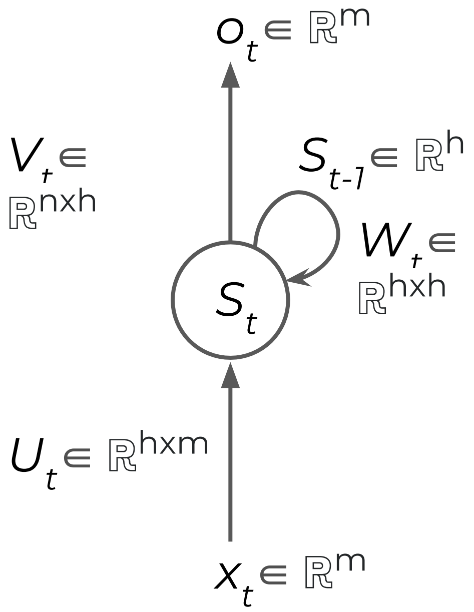

$o_t \in \mathbb{R}^m$
$V_t \in \mathbb{R}^{nxh}$
$S_{t-1} \in \mathbb{R}^h$
$W_t \in \mathbb{R}^{hxh}$
S_t
$U_t \in \mathbb{R}^{hxm}$
$x_t \in \mathbb{R}^m$

Illustration inspired by
http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Unrolling Through Time

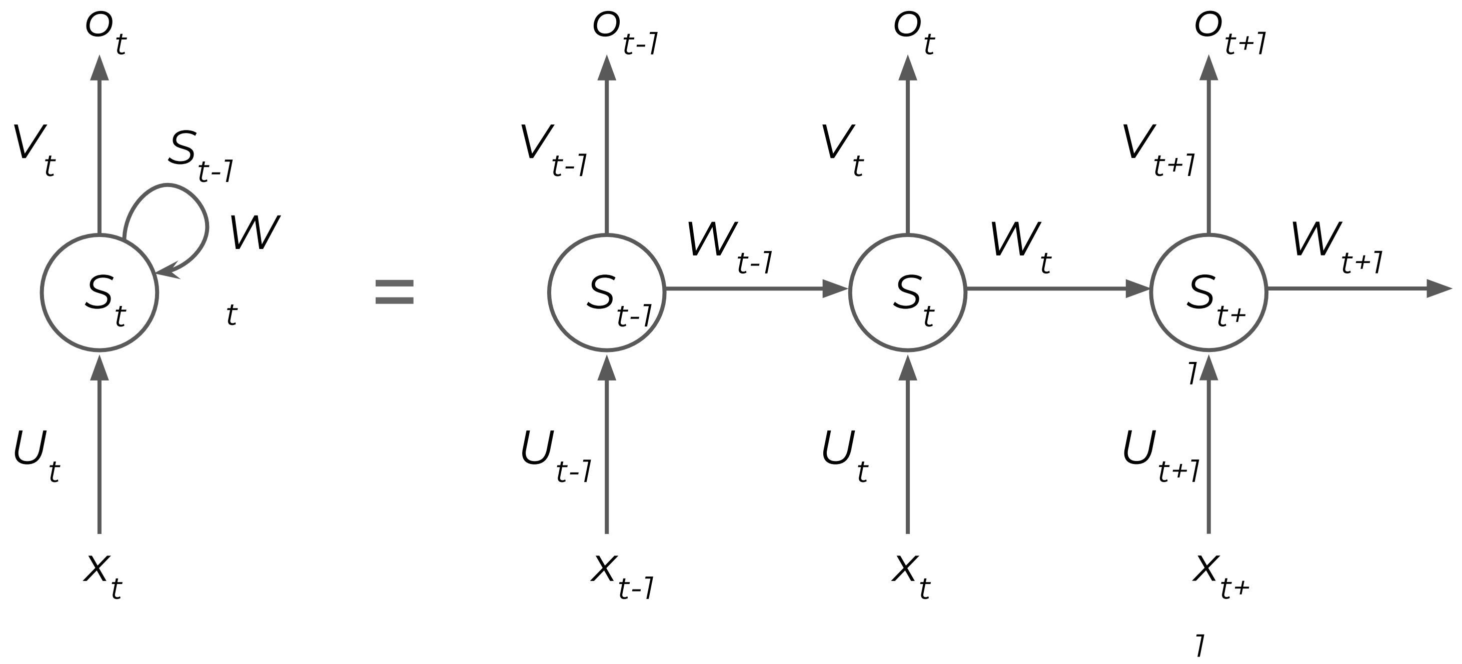
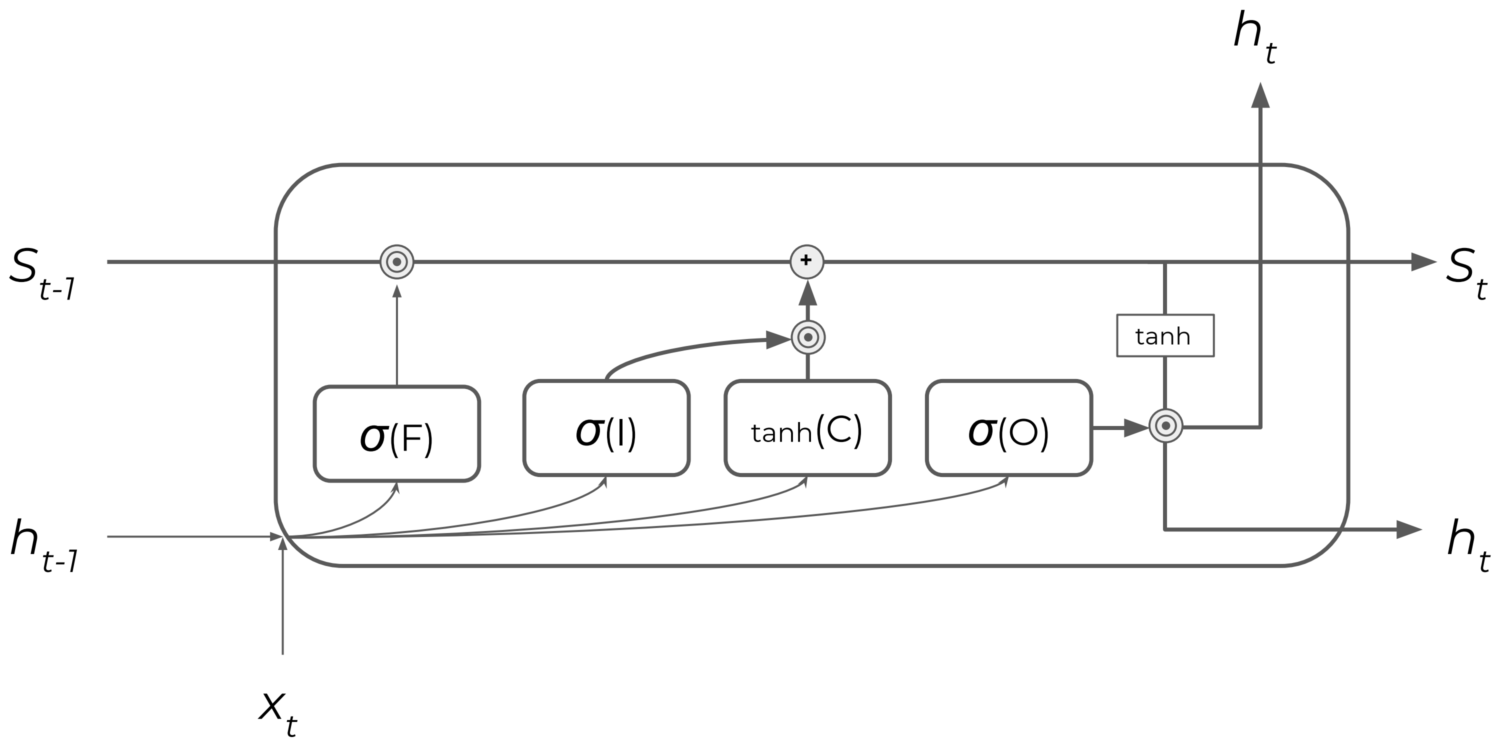


Illustration inspired by
http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Long Short-Term Memory

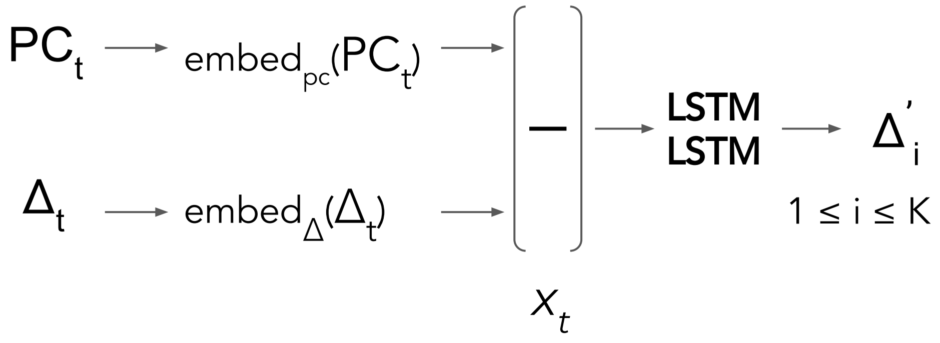
F_t	Forget Gate	$\sigma(W_F[x_t, h_{t-1}]+b_F)$	Learns what to forget
I_t	Input Gate	$\sigma(W_I[x_t, h_{t-1}]+b_I)$	Learns what to assimilate
O_t	Output Gate	$\sigma(W_O[x_t, h_{t-1}]+b_O)$	Learns what to emit
			Candidate memory
C_t	Cell Gate	$I_t \odot \tanh(W_C[x_t, h_{t-1}]+b_C)$	
S_t	State	$F_t \odot S_{t-1} + C_t$	Cell state: Linear combination of candidate memory and previous cell state
h_t	Hidden State	$O_t \odot \tanh(S_t)$	**Hidden state**: Prediction at current timestep

Long Short-Term Memory



Inspired by:
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Embedding LSTM



Note: In the paper, they use **f** (embed_{Δ}) and **g** (embed_{pc}) to denote the embedding functions as in the next slide

Classification over delta vocabulary

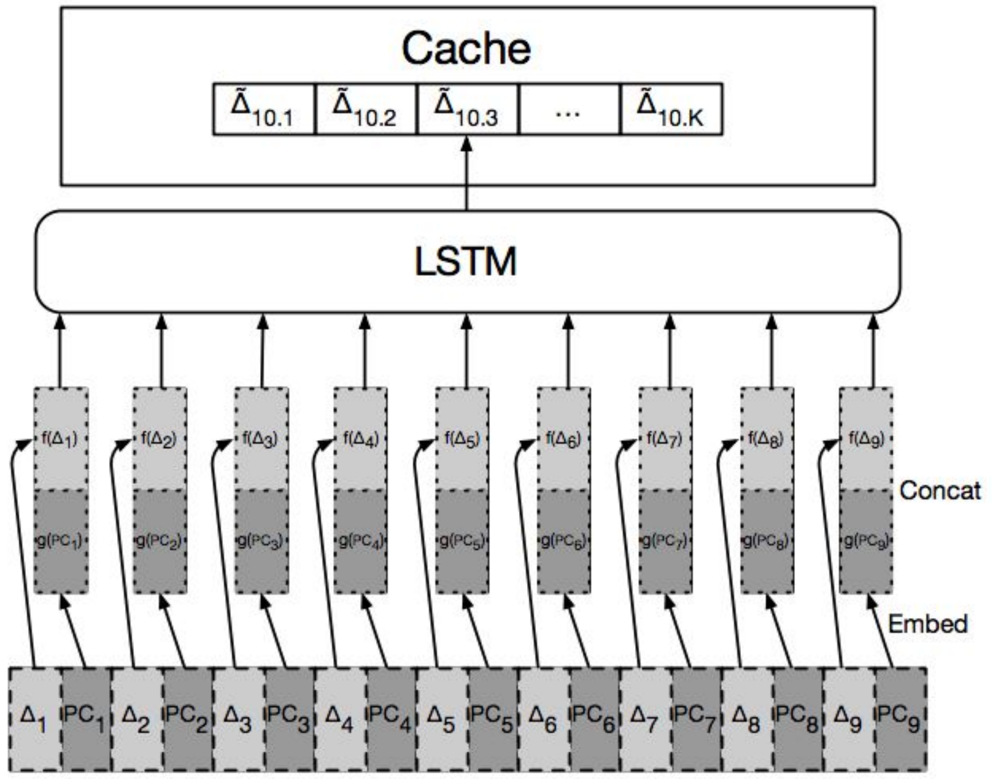
Predict K-highest probability deltas for prefetching each timestep

Add Δ'_i back to address at time *t* to get new address

Problems:

1. Large vocab -> large memory
2. Vocab truncation -> lower acc
3. Rare features

Embedding LSTM


Cache
$\tilde{\Delta}_{10.1}$ $\tilde{\Delta}_{10.2}$ $\tilde{\Delta}_{10.3}$ ... $\tilde{\Delta}_{10.K}$
LSTM
$f(\Delta_1)$ $f(\Delta_2)$ $f(\Delta_3)$ $f(\Delta_4)$ $f(\Delta_5)$ $f(\Delta_6)$ $f(\Delta_7)$ $f(\Delta_8)$ $f(\Delta_9)$
$g(PC_1)$ $g(PC_2)$ $g(PC_3)$ $g(PC_4)$ $g(PC_5)$ $g(PC_6)$ $g(PC_7)$ $g(PC_8)$ $g(PC_9)$
Concat
Embed
Δ_1 PC_1 Δ_2 PC_2 Δ_3 PC_3 Δ_4 PC_4 Δ_5 PC_5 Δ_6 PC_6 Δ_7 PC_7 Δ_8 PC_8 Δ_9 PC_9

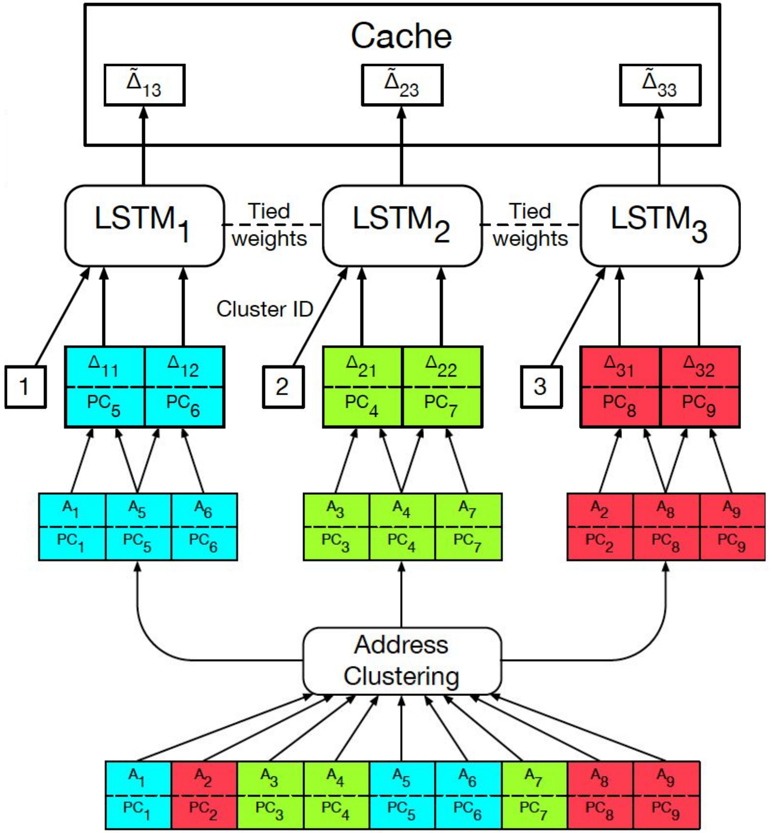
Clustering + LSTM

- k-means to cluster omnetpp addresses into 6 regions
- Compute deltas within each cluster
- Deltas in a cluster are significantly smaller than the global vocabulary
- Reduce model size:
  - LSTM for each cluster with weight sharing
  - Cluster id feature

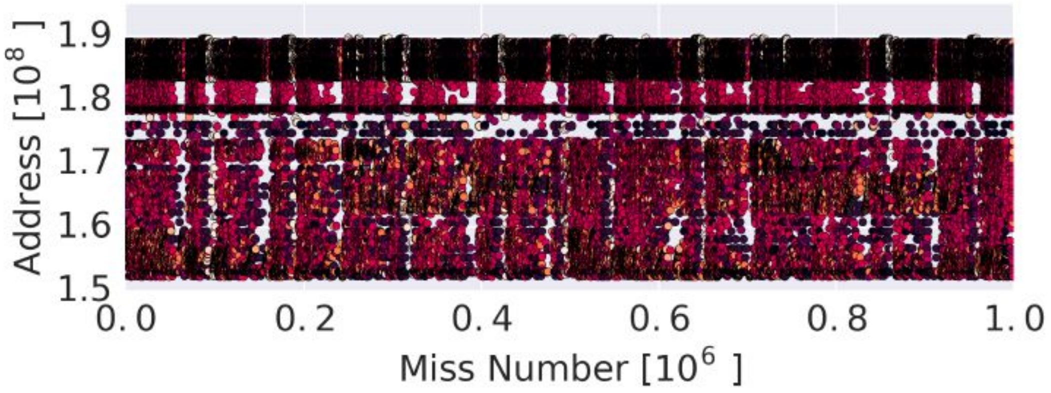
Clustering + LSTM Model

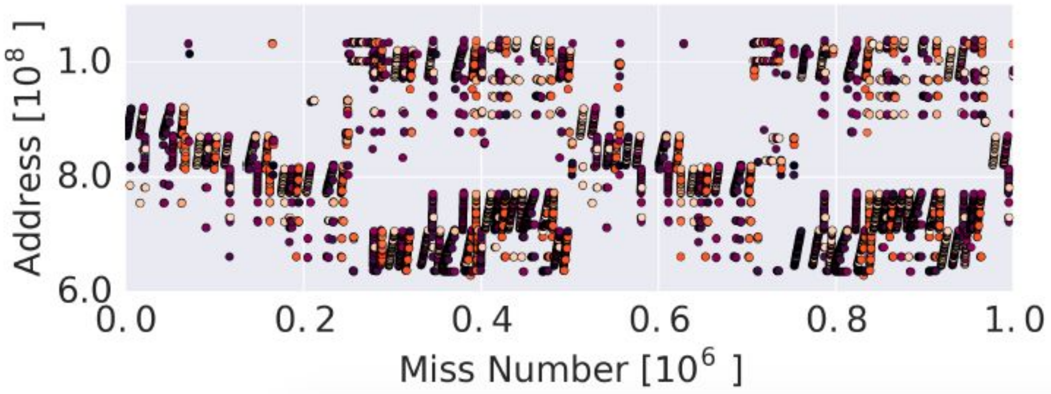
Predict over clusters

Cluster address space into separate streams



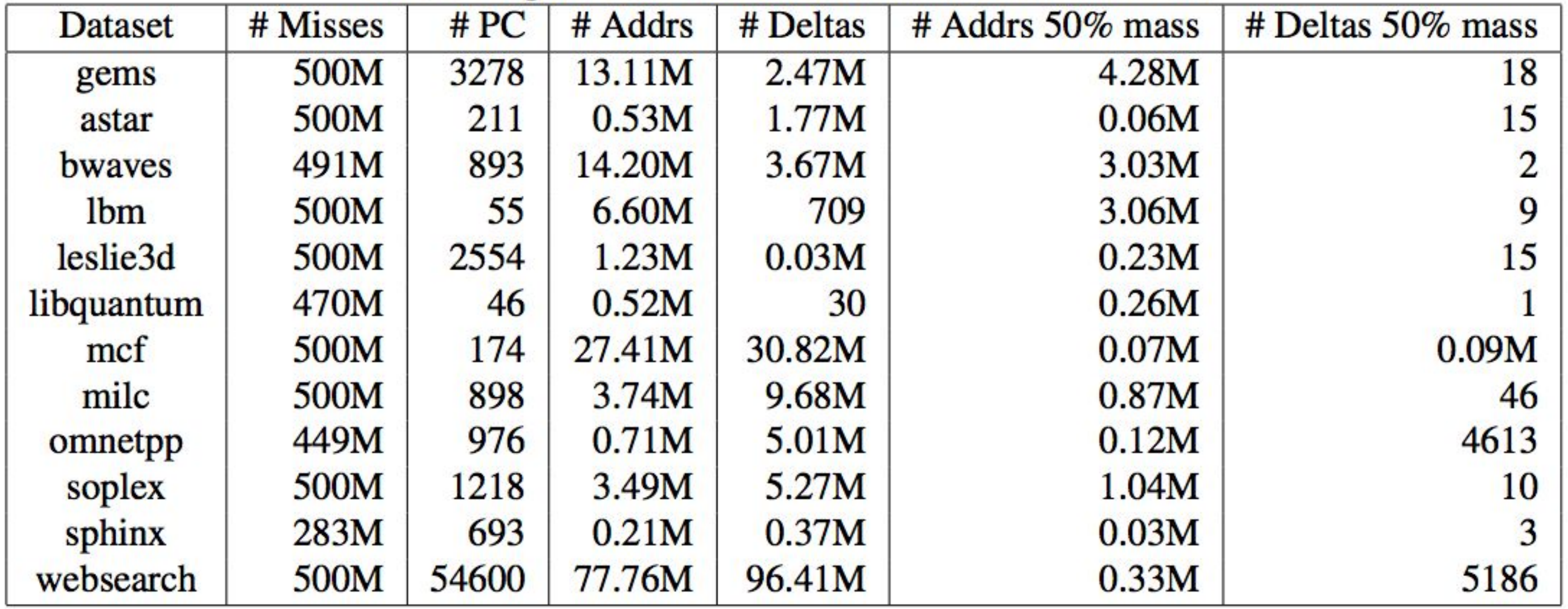
K-means Clusters on omnetpp Benchmark





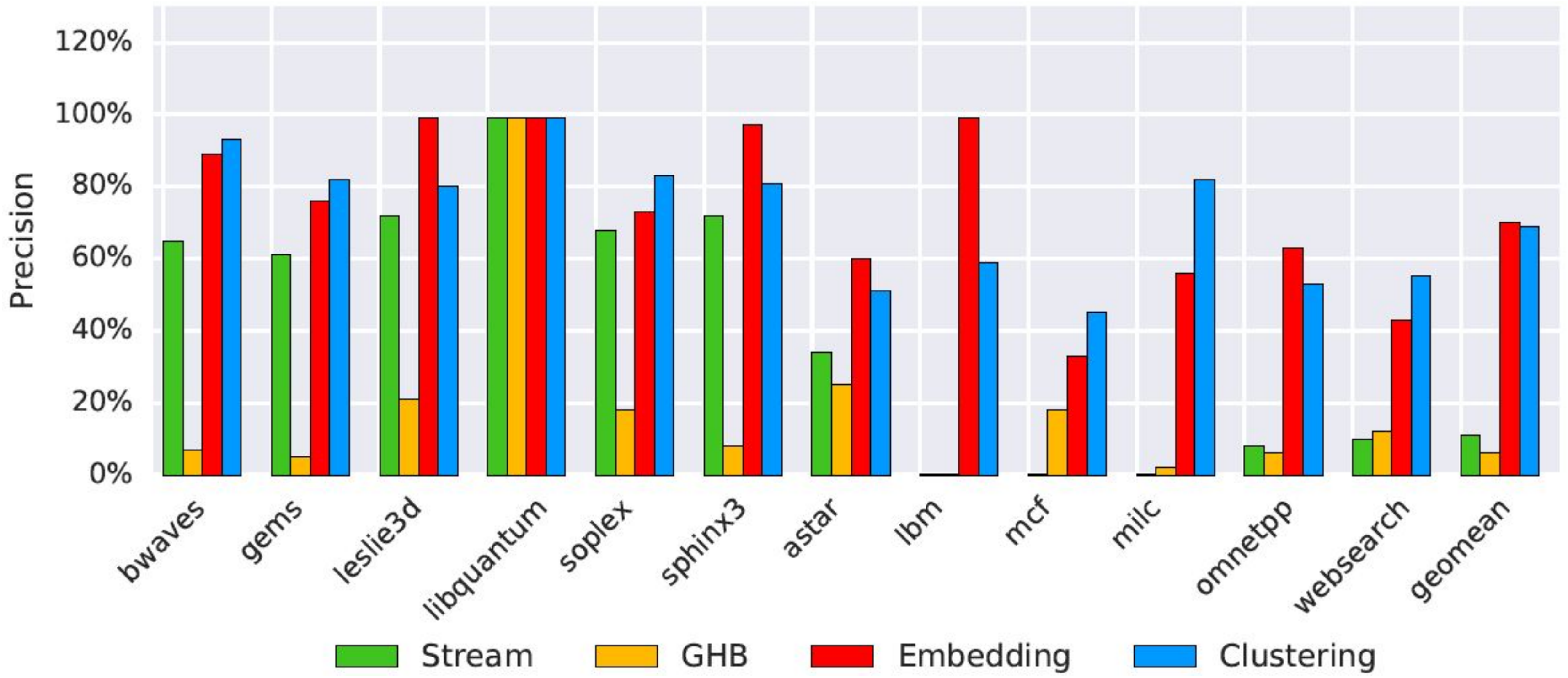
Program Trace Statistics

Table 1. Program trace dataset statistics. M stands for million.



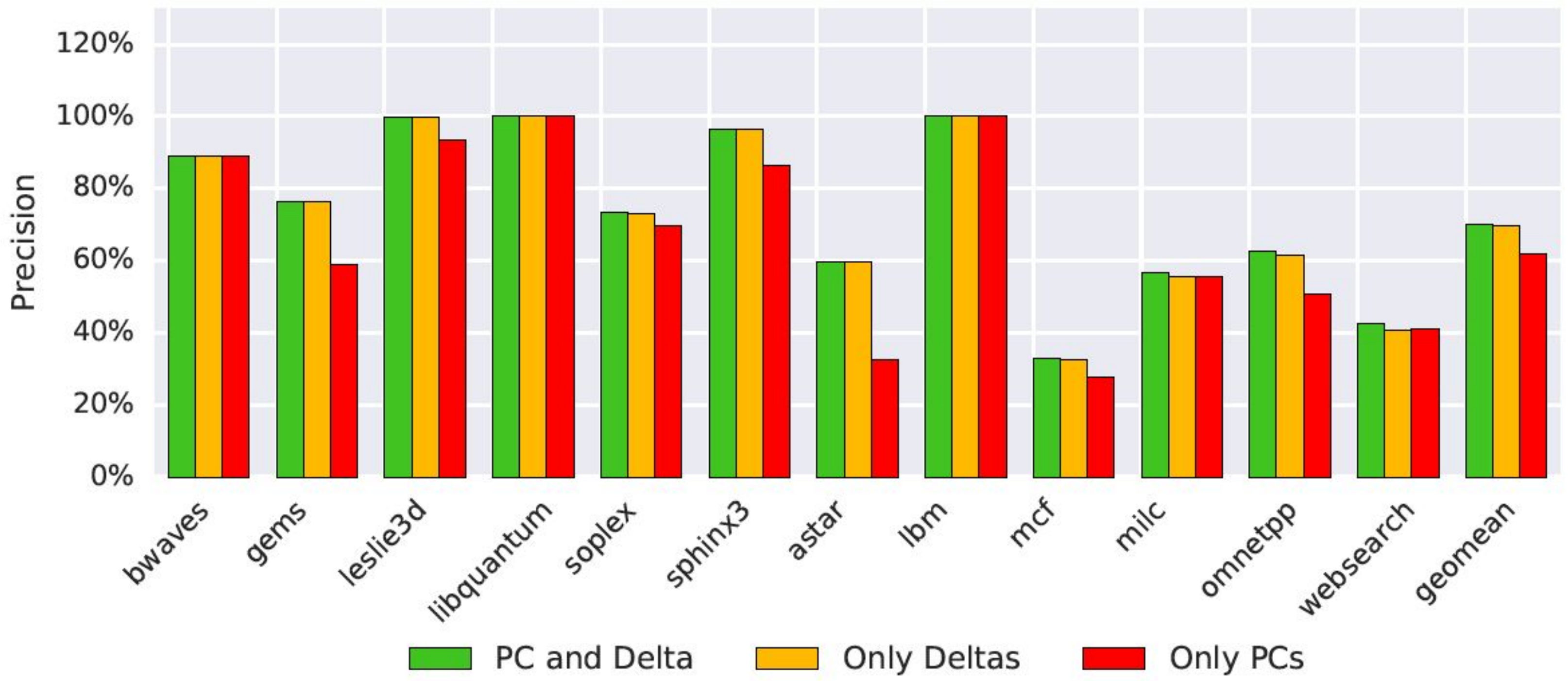
Dataset	# Misses	# PC	# Addrs	# Deltas	# Addrs 50% mass	# Deltas 50% mass
gems	500M	3278	13.11M	2.47M	4.28M	18
astar	500M	211	0.53M	1.77M	0.06M	15
bwaves	491M	893	14.20M	3.67M	3.03M	2
lbm	500M	55	6.60M	709	3.06M	9
leslie3d	500M	2554	1.23M	0.03M	0.23M	15
libquantum	470M	46	0.52M	30	0.26M	1
mcf	500M	174	27.41M	30.82M	0.07M	0.09M
milc	500M	898	3.74M	9.68M	0.87M	46
omnetpp	449M	976	0.71M	5.01M	0.12M	4613
soplex	500M	1218	3.49M	5.27M	1.04M	10
sphinx	283M	693	0.21M	0.37M	0.03M	3
websearch	500M	54600	77.76M	96.41M	0.33M	5186

Precision



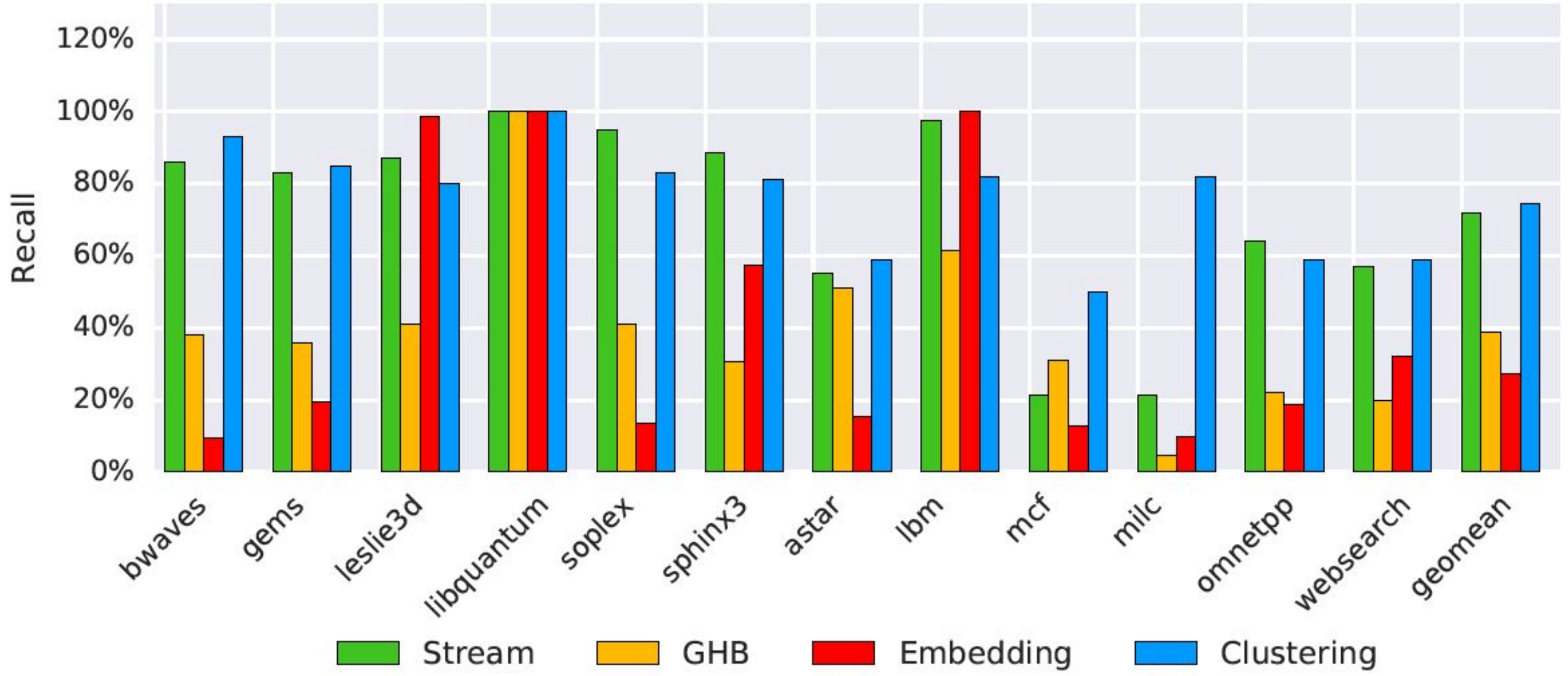
Precision
120%
100%
80%
60%
40%
20%
0%
bwaves
gems
leslie3d
libquantum
soplex
sphinx3
astar
lbm
mcf
milc
omnetpp
websearch
geomean
Stream
GHB
Embedding
Clustering

Precision



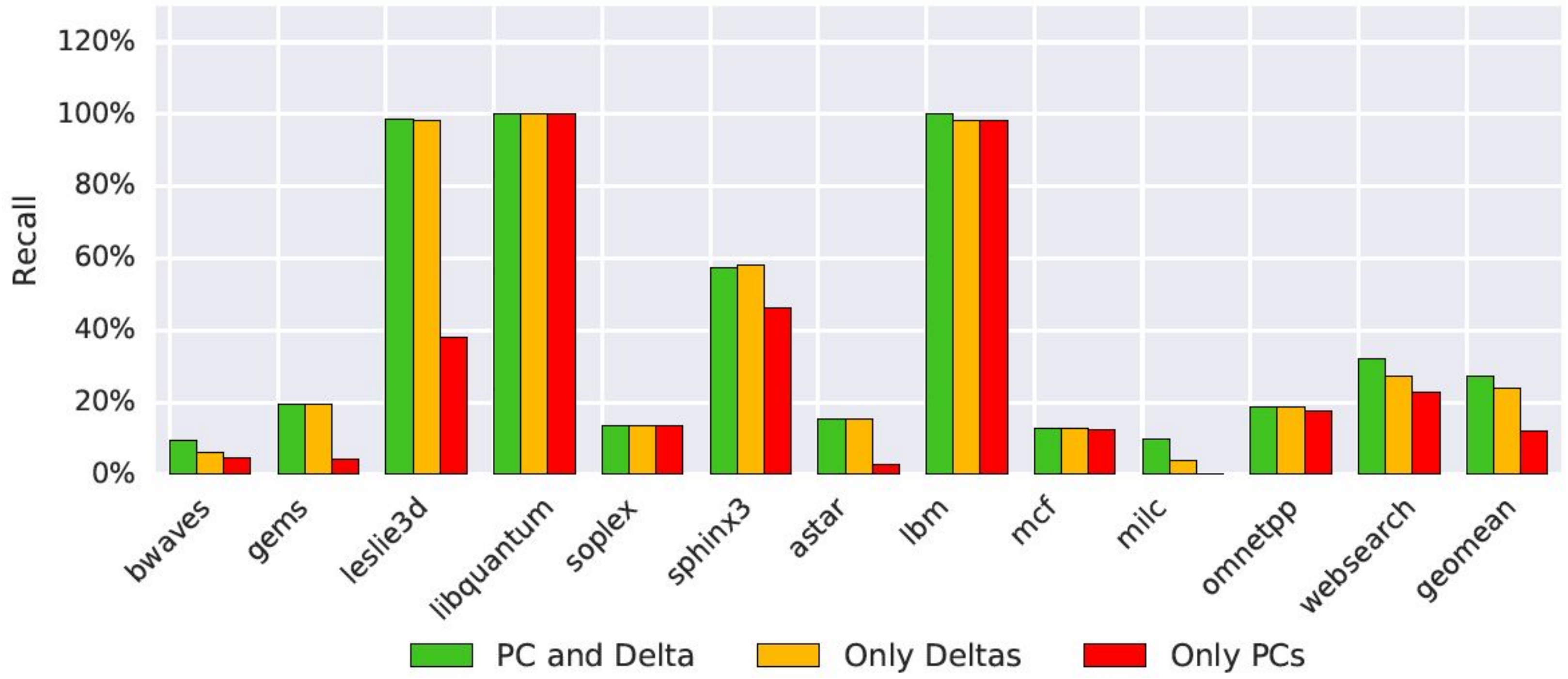
Precision
120%
100%
80%
60%
40%
20%
0%
bwaves
gems
leslie3d
libquantum
soplex
sphinx3
astar
lbm
mcf
milc
omnetpp
websearch
geomean
PC and Delta
Only Deltas
Only PCs

Recall



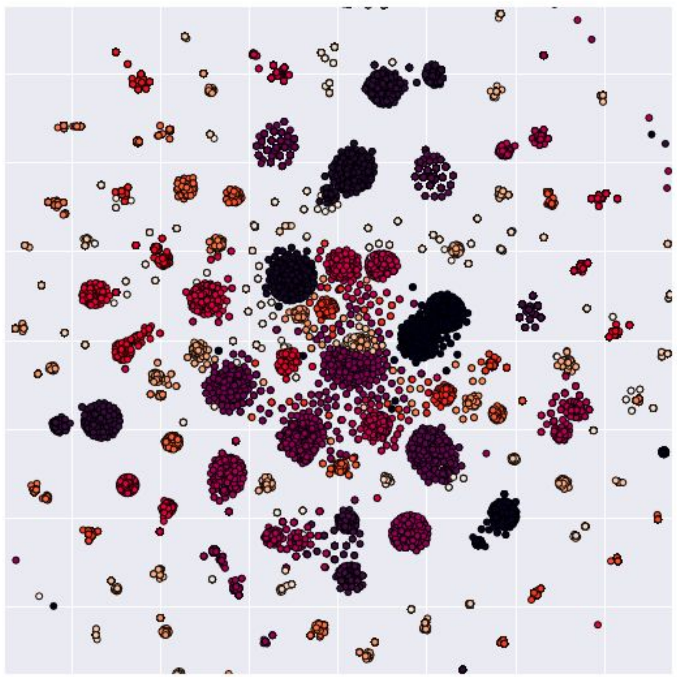
Recall
120%
100%
80%
60%
40%
20%
0%
bwaves
gems
leslie3d
libquantum
soplex
sphinx3
astar
lbm
mcf
milc
omnetpp
websearch
geomean
Stream
GHB
Embedding
Clustering

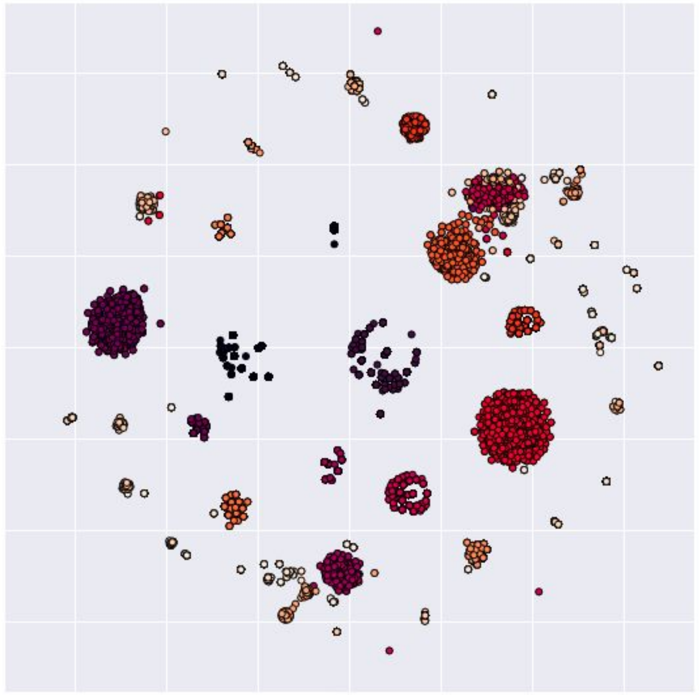
Recall



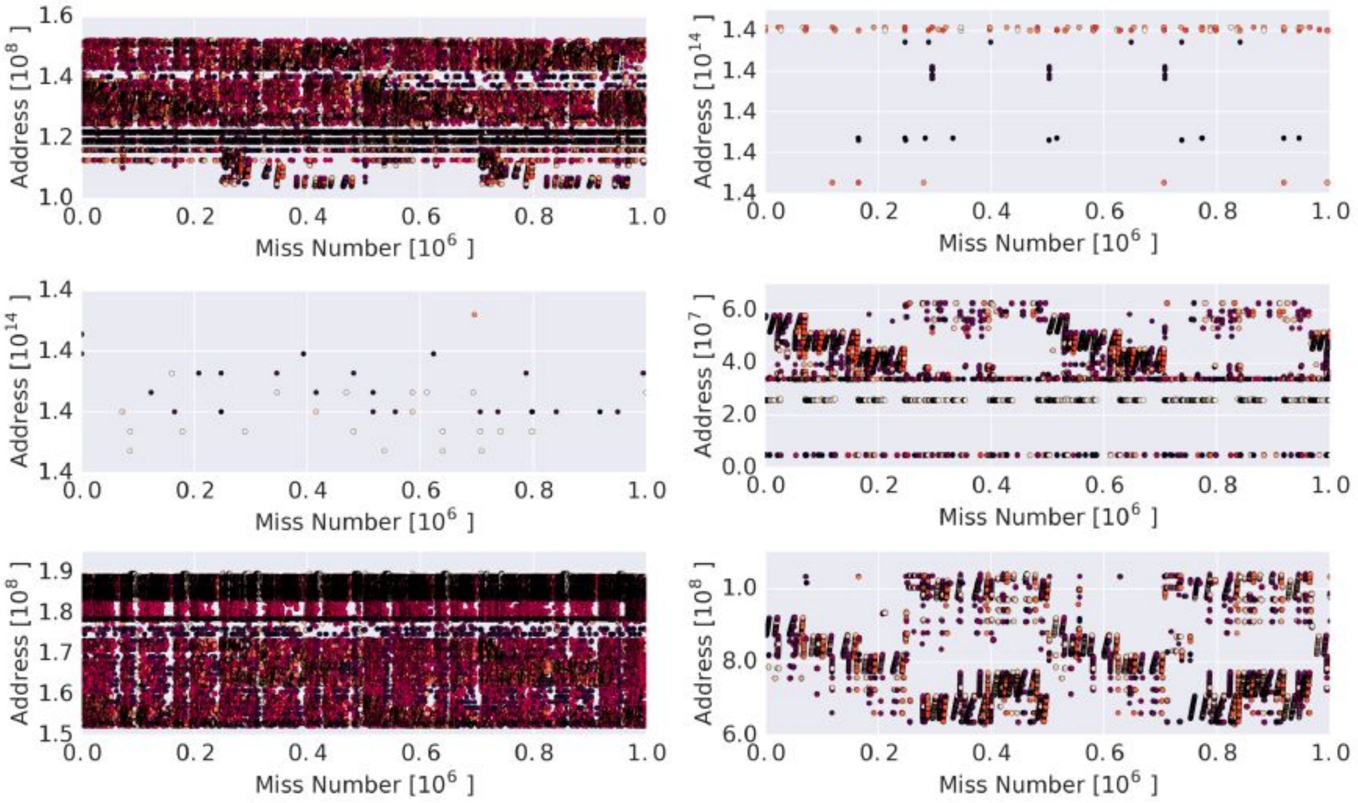
Recall
120%
100%
80%
60%
40%
20%
0%
bwaves
gems
leslie3d
libquantum
soplex
sphinx3
astar
lbm
mcf
milc
omnetpp
websearch
geomean
PC and Delta
Only Deltas
Only PCs

t-SNE Visualizations of Datasets


Concatenated embeddings of *omnetpp* colored by PC instructions


Concatenated embeddings of *mcf* colored by PC instructions

K-means Clustering on Address Space



References

1. Hashemi, Milad, et al. "**Learning Memory Access Patterns**." arXiv preprint arXiv:1803.02329 (2018).
2. Kraska, Tim, et al. "**The Case for Learned Index Structures**." *arXiv preprint arXiv:1712.01208* (2017).
3. Van Den Oord, Aaron, et al. "**Wavenet: A generative model for raw audio**." arXiv preprint arXiv:1609.03499 (2016).
4. Oord, Aaron van den, Nal Kalchbrenner, and Koray Kavukcuoglu. "**Pixel recurrent neural networks**." arXiv preprint arXiv:1601.06759 (2016).
5. Siegelmann, Hava T., and Eduardo D. Sontag. "**On the computational power of neural nets**." *Journal of computer and system sciences* 50.1 (1995): 132-150.