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

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

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**

Omnetpp Cache Miss Sparse Access Pattern
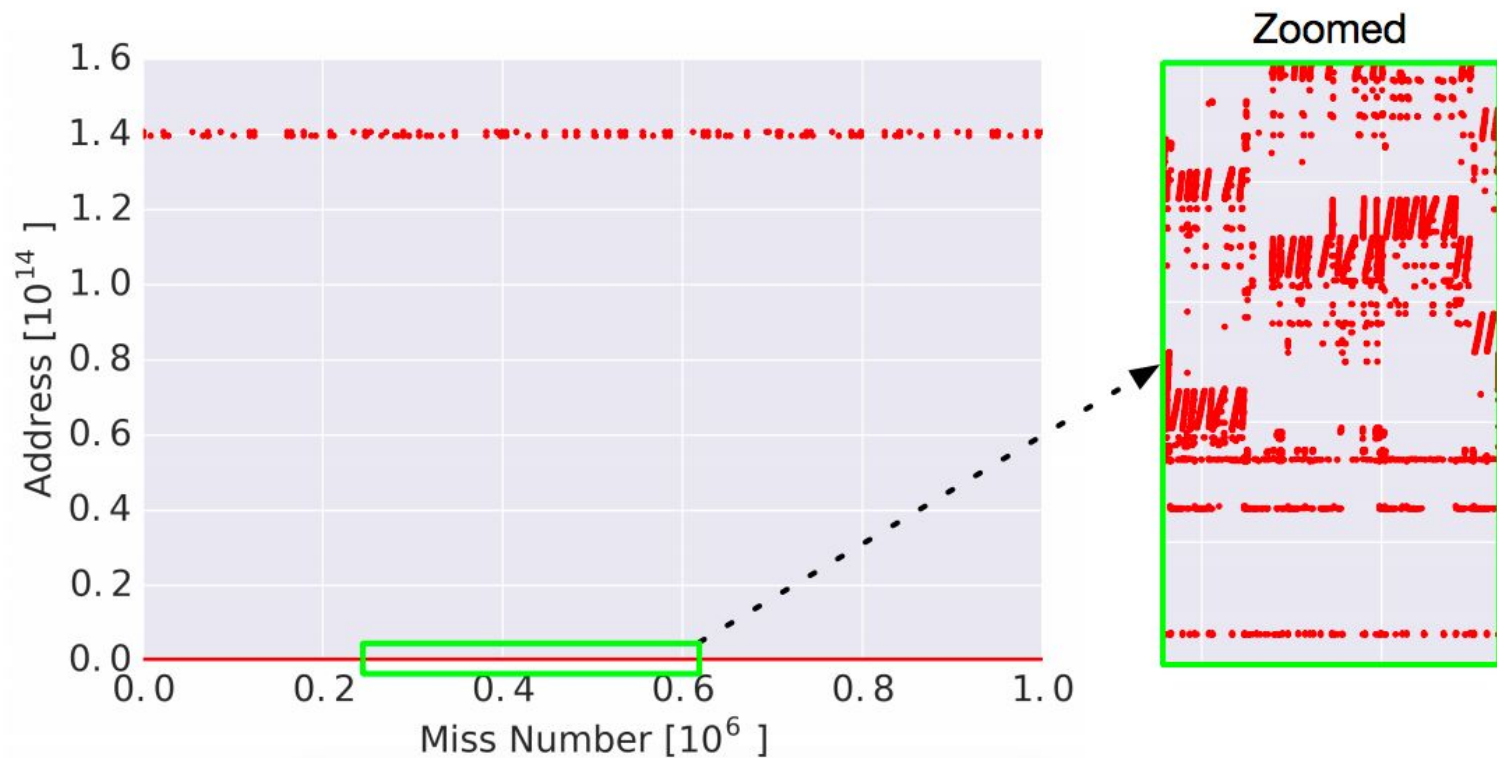
Prefetching as Classification

- **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
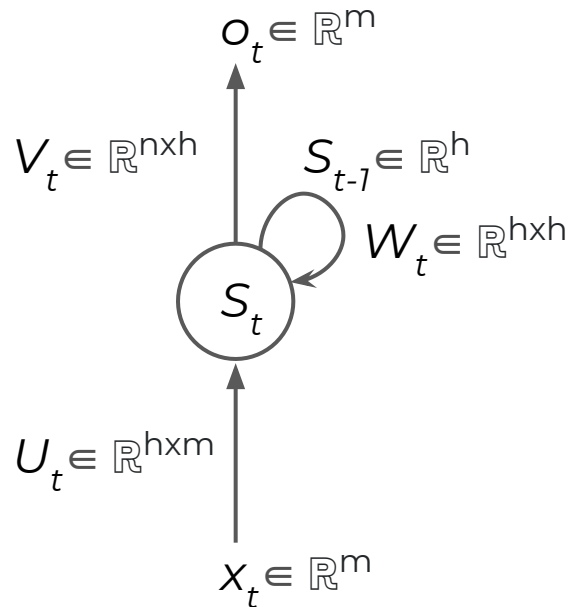
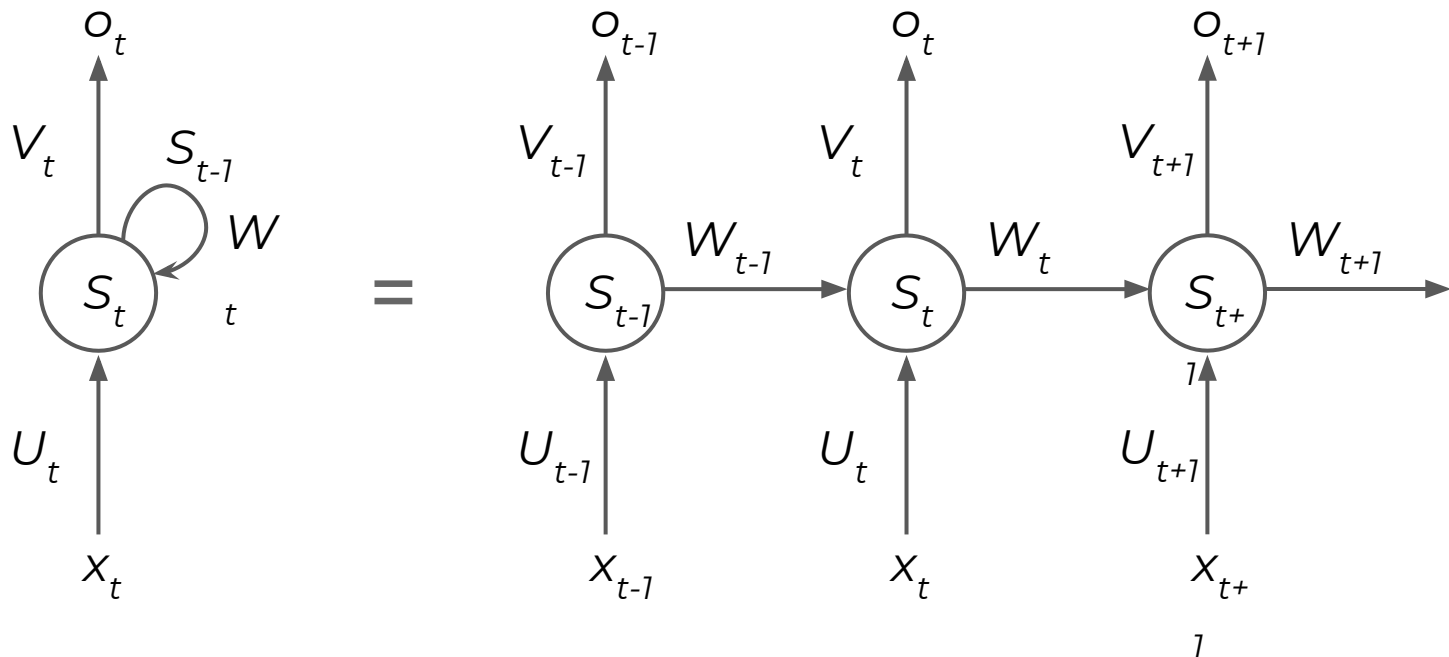# Recurrent Neural Network (RNN)

- 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

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

$$o_t = V_t S_t$$

$o_t \in \mathbb{R}^m$
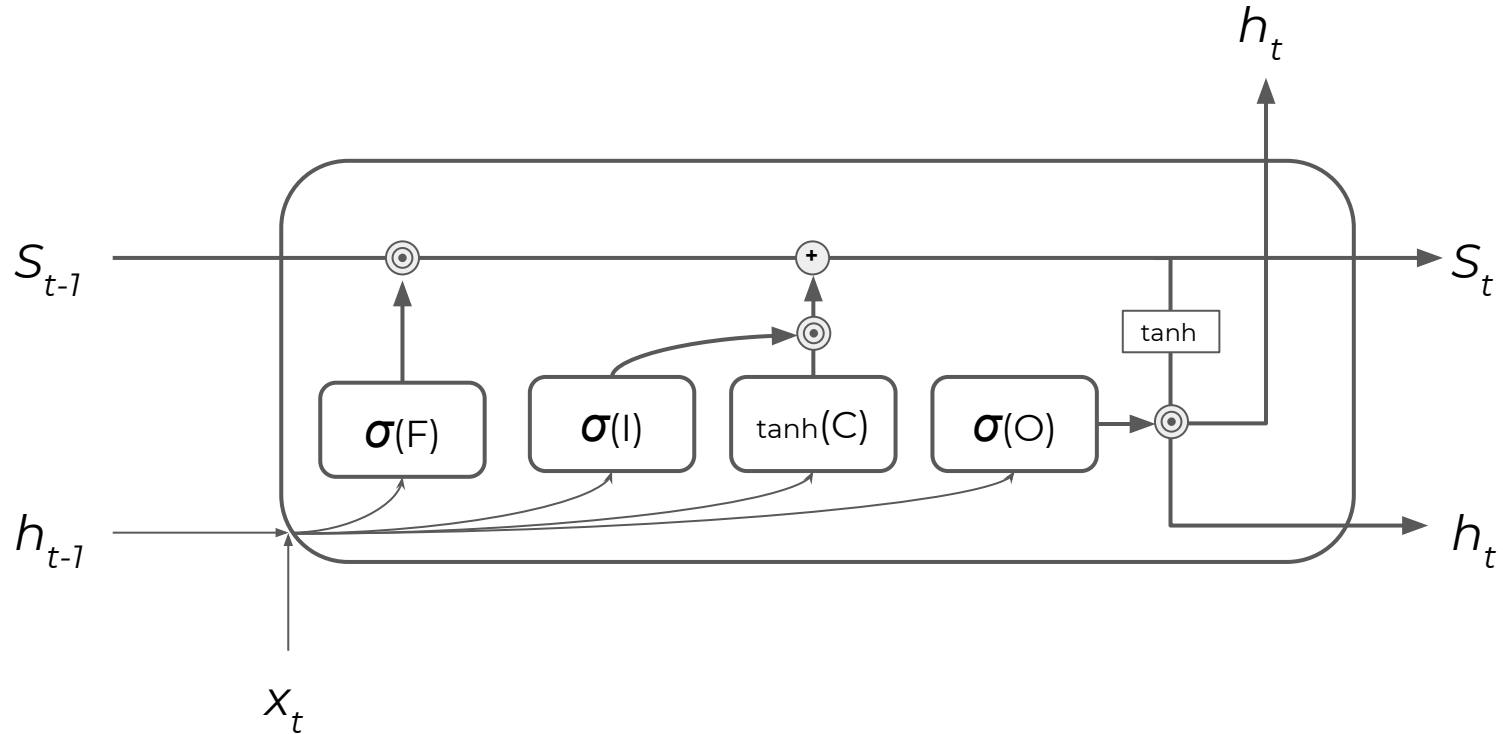
$V_t \in \mathbb{R}^{n \times h}$

$S_{t-1} \in \mathbb{R}^h$

$W_t \in \mathbb{R}^{h \times h}$

S_t

$U_t \in \mathbb{R}^{h \times m}$

$x_t \in \mathbb{R}^m$

Unrolling Through Time
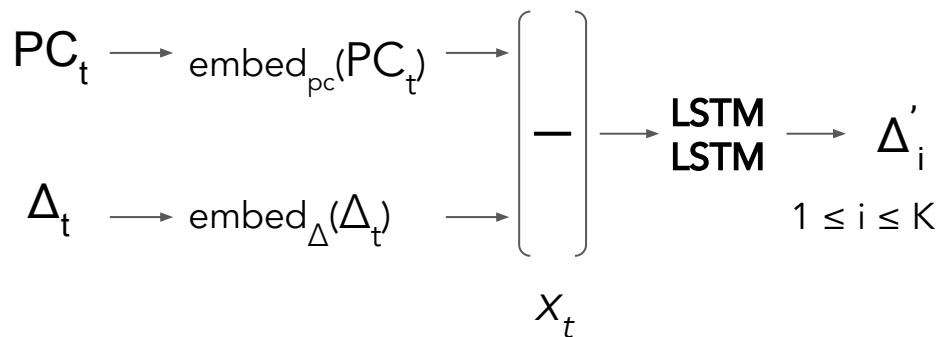
Long Short-Term Memory

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 $I_t \odot \tanh(W_C[x_t, h_{t-1}]+b_C)$

Cell state: Linear combination of candidate memory and previous cell state

S_t State $F_t \odot S_{t-1} + C_t$

Hidden state: Prediction at current timestep

h_t Hidden State $O_t \odot \tanh(S_t)$

LSTMs extend the memory capacity of conventional RNNs

Long Short-Term Memory

Embedding LSTM

PC_t $\longrightarrow$ $\text{embed}_{pc}(PC_t)$ $\longrightarrow$

Δ_t $\longrightarrow$ $\text{embed}_{\Delta}(\Delta_t)$ $\longrightarrow$

$\left[- \right]$ $\longrightarrow$ **LSTM** **LSTM** $\longrightarrow$ Δ'_i

$1 \le i \le K$

x_t

Note: In the paper, they use **f** (embed$_{\Delta}$) 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
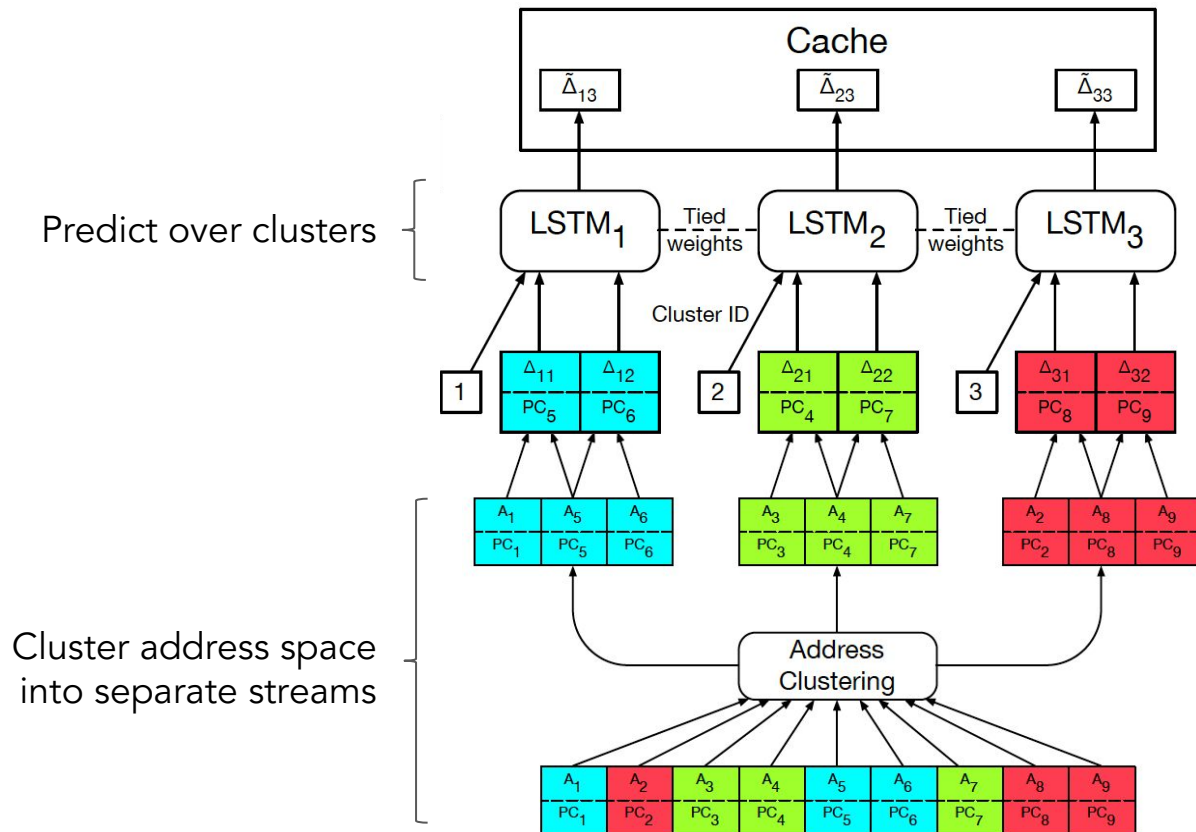3. Rare features

13

Embedding LSTM
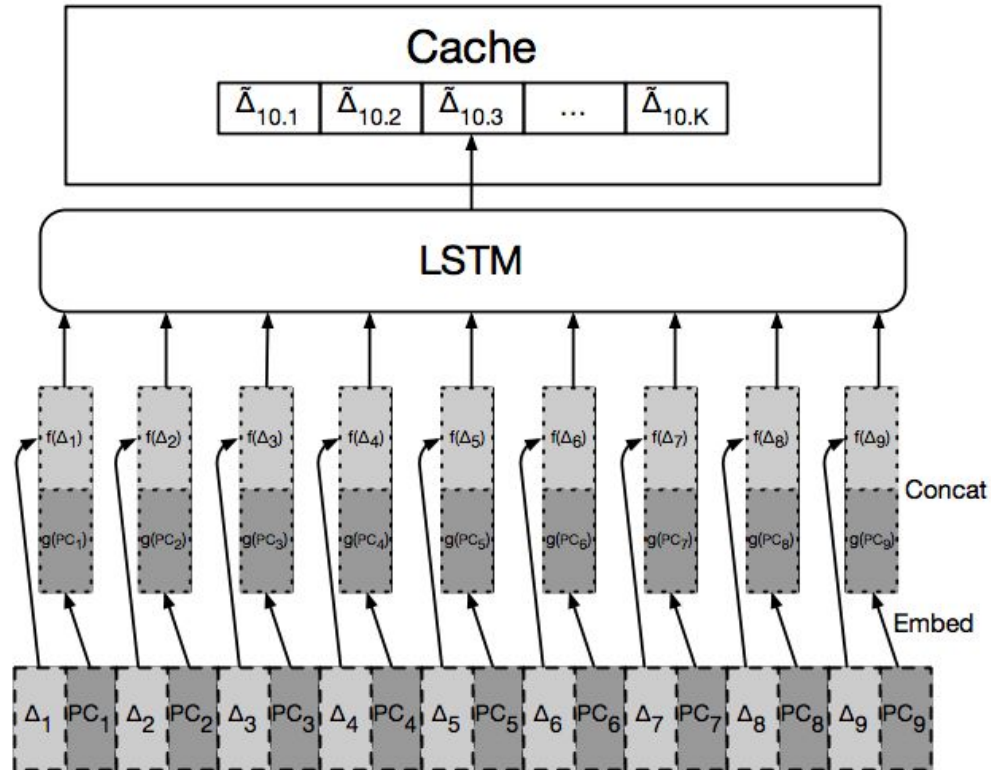
Clustering + LSTM

- 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:
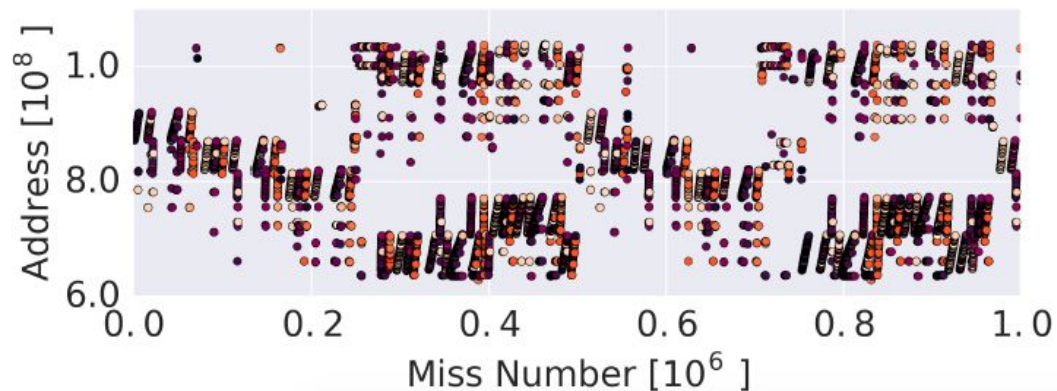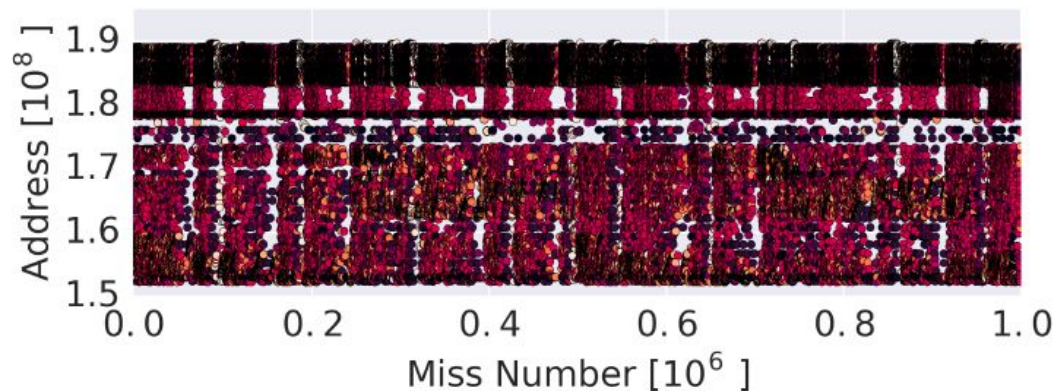 - LSTM for each cluster with weight sharing
  - Cluster id feature

Clustering + LSTM Model

# 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 Comparison
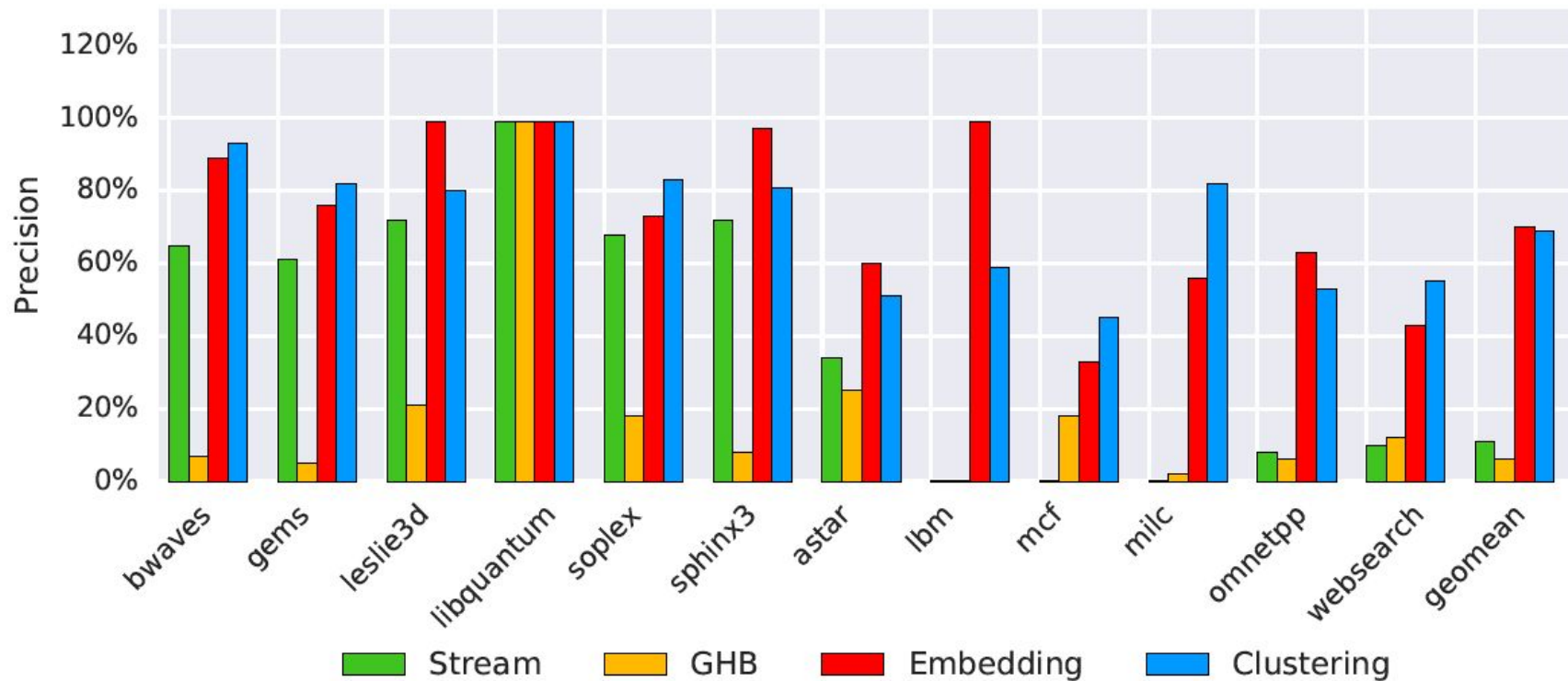
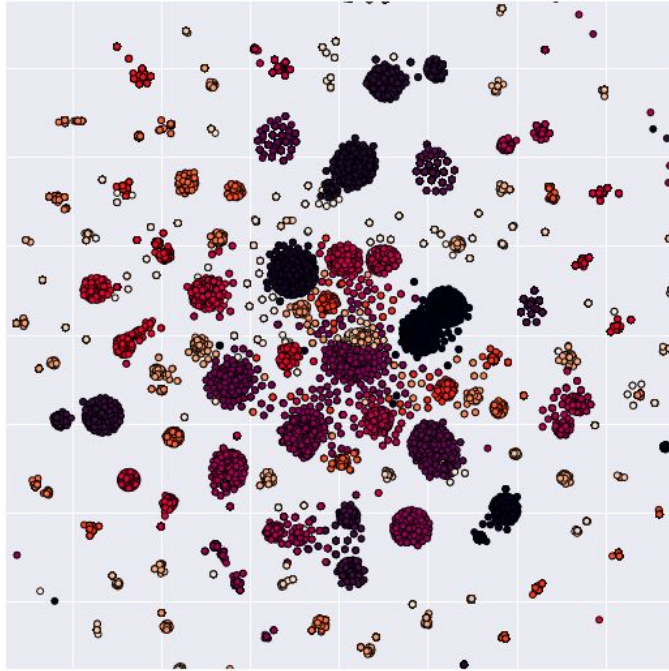# Precision for Embedding LSTM
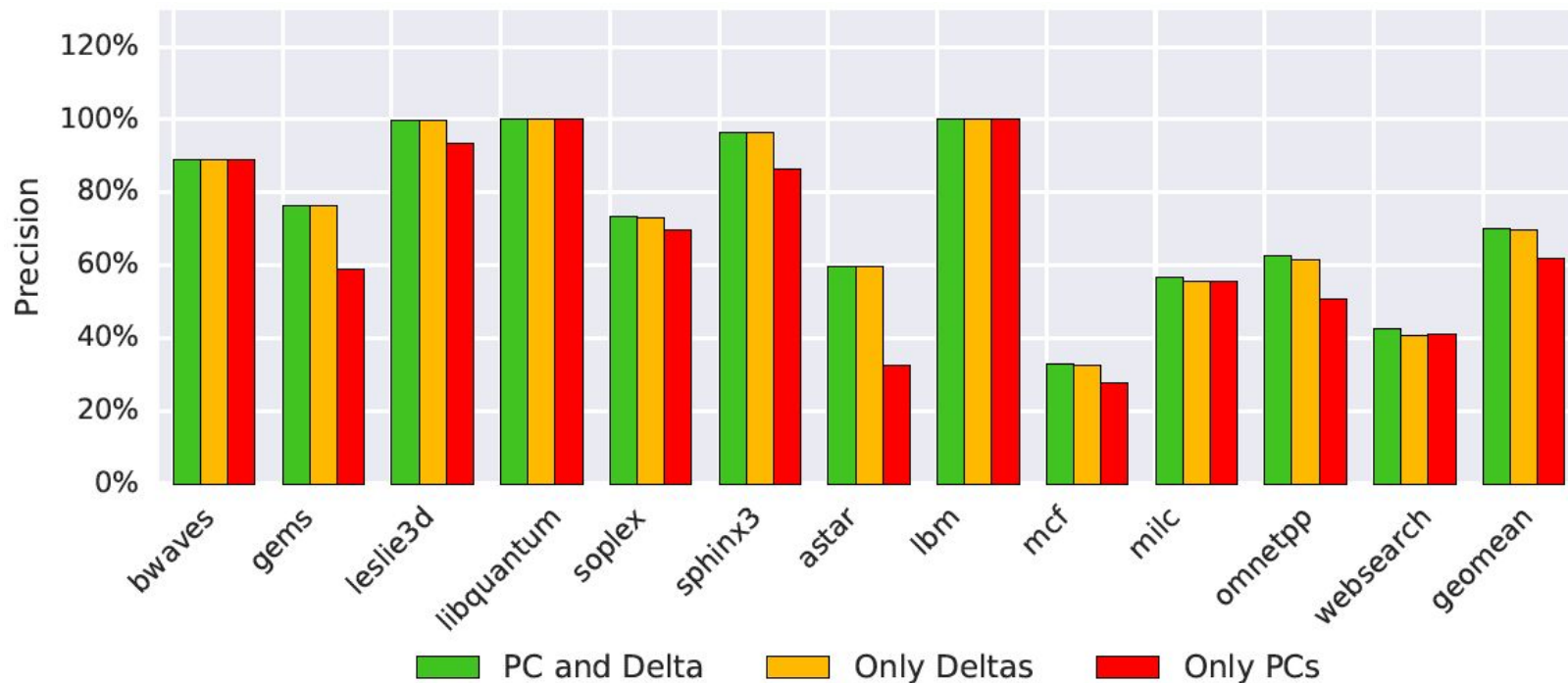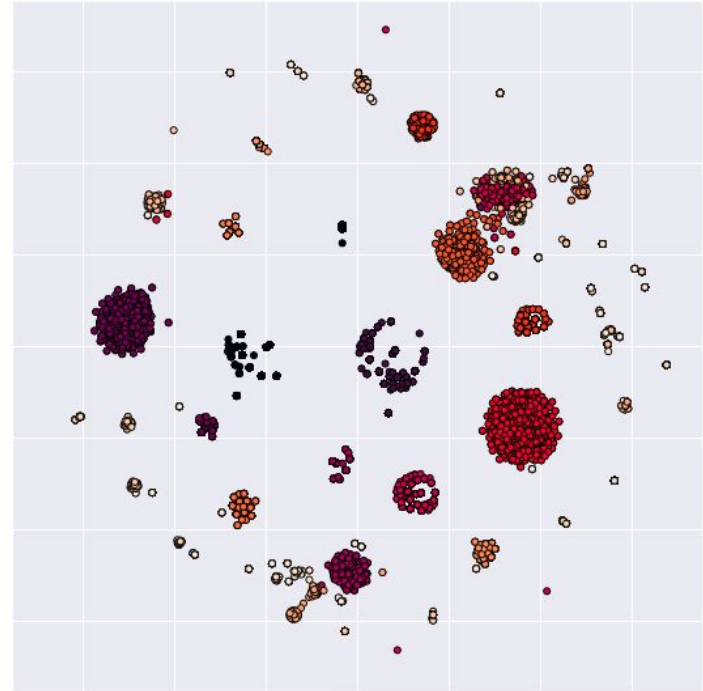
Recall Comparison
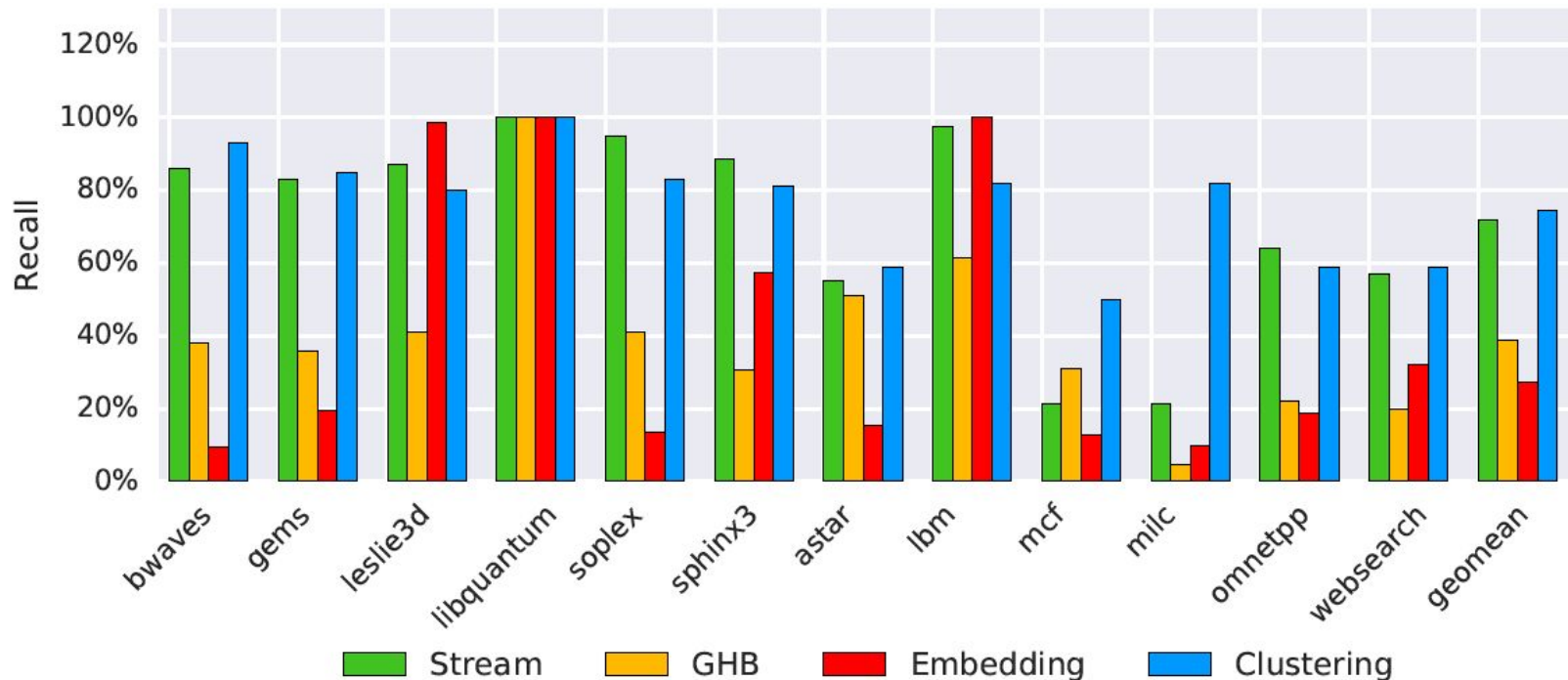
Recall for Embedding LSTM
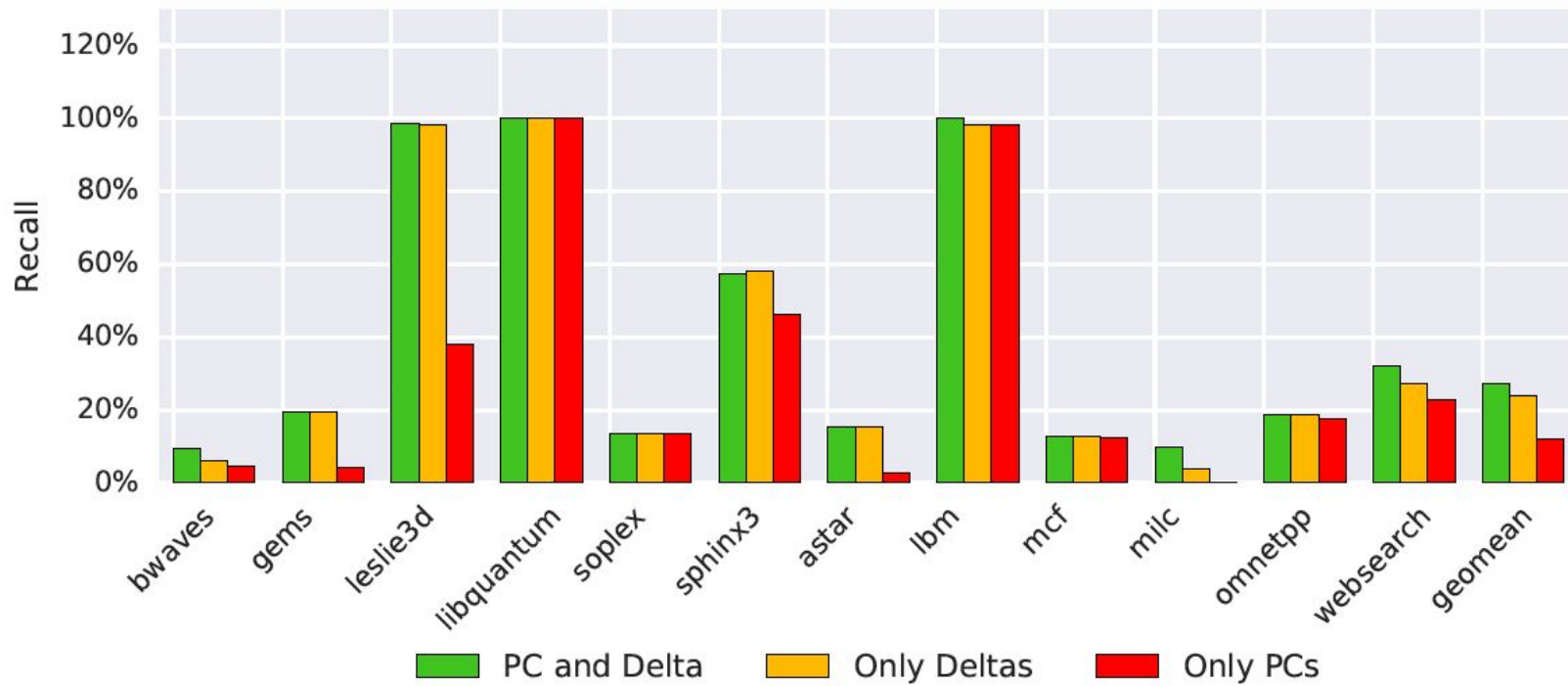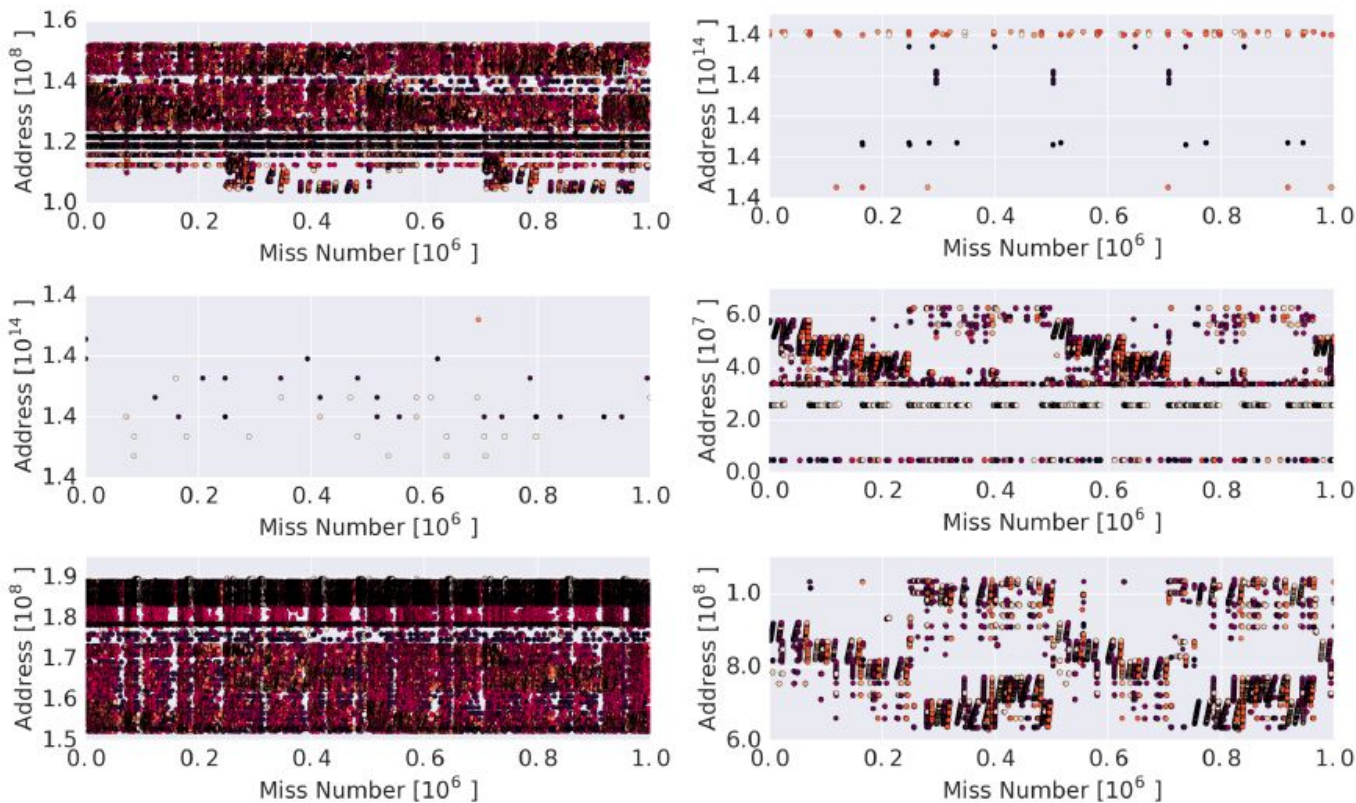
t-SNE Visualizations of Datasets

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.