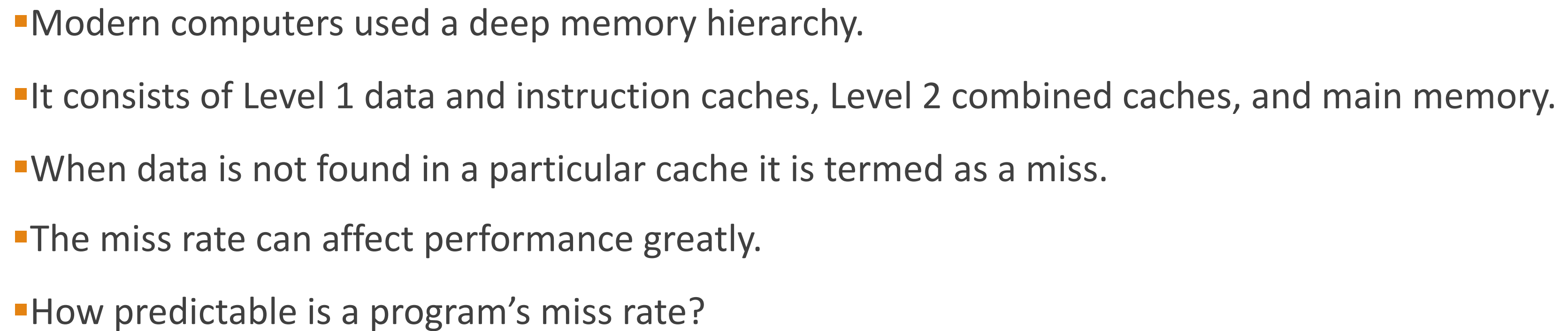
Cache Miss Rate Predictability via Neural Networks and Learning Memory Access Patterns

What is the paper about?

- A program can be characterized by its memory access patterns. These patterns are analyzed using Machine Learning.
- These memory accesses are characterized by a sequence of “cache miss rates”.
- A new data set is created. This set draws from programs run on various JVMs, C, and Fortran compilers.
- Answer the question: How predictable is a program’s cache miss rate as it executes?

What is a cache miss?



Benchmarks

- To evaluate the efficiency of compilers during a program run certain benchmarks are developed.
- C, C++ have benchmarks like SPEC CPU 2000
- Java programs have benchmarks like DaCapo.

This paper uses these benchmarks to predict the cache miss rates for the programs.

```
                           SPEC CINT2000 Summary
   ASUS Computer International Asus M2N32-SLI Deluxe, AMD Athlon (TM) 64 4200+
                          Sat Jul 10 20:58:20 2004

SPEC License #13    Test date: Jul-2006     Hardware availability: Jun-2006
Tester: Intel Corporation                   Software availability: Jun-2006

                  Base      Base      Base      Peak      Peak      Peak
   Benchmarks    Ref Time  Run Time   Ratio    Ref Time  Run Time   Ratio
   ------------  --------  --------  --------  --------  --------  --------
   164.gzip          1400    115         1215      1400    115         1215
   164.gzip          1400    115         1215*     1400    115         1215*
   164.gzip          1400    115         1215      1400    115         1215
   175.vpr           1400    144          970      1400    141          993*
   175.vpr           1400    145          969      1400    141          994
   175.vpr           1400    144          970*     1400    141          993
   176.gcc           1100     75.5       1457      1100     75.0       1466
   176.gcc           1100     75.3       1461      1100     74.9       1468*
   176.gcc           1100     75.4       1458*     1100     74.9       1469
   181.mcf           1800    176         1024      1800    160         1126*
   181.mcf           1800    176         1020*     1800    162         1112
   181.mcf           1800    178         1013      1800    160         1127
   186.crafty        1000     71.8       1392      1000     65.1       1535
   186.crafty        1000     71.8       1393      1000     65.0       1537
   186.crafty        1000     71.8       1393*     1000     65.1       1536*
   197.parser        1800    137         1313      1800    137         1310
   197.parser        1800    137         1314      1800    137         1311
   197.parser        1800    137         1313*     1800    137         1311*
   252.eon           1300     67.3       1932      1300     53.8       2418
   252.eon           1300     67.1       1937      1300     53.6       2426
   252.eon           1300     67.1       1936*     1300     53.6       2424*
   253.perlbmk       1800    125         1443      1800    125         1443
   253.perlbmk       1800    125         1445      1800    125         1445
   253.perlbmk       1800    125         1445*     1800    125         1445*
   254.gap           1100     70.9       1551      1100     70.9       1551
   254.gap           1100     71.2       1546*     1100     71.2       1546*
   254.gap           1100     71.5       1539      1100     71.5       1539
   255.vortex        1900     86.6       2195      1900     80.0       2374
   255.vortex        1900     86.5       2197*     1900     79.9       2378
   255.vortex        1900     86.5       2197      1900     79.9       2377*
   256.bzip2         1500    139         1078      1500    137         1095*
   256.bzip2         1500    139         1077*     1500    137         1095
   256.bzip2         1500    139         1077      1500    137         1095
   300.twolf         3000    282         1064*     3000    250         1202
   300.twolf         3000    282         1065      3000    251         1197*
   300.twolf         3000    282         1062      3000    251         1195
   ========================================================================
   164.gzip          1400    115         1215*     1400    115         1215*
```

Example Results from results.txt:

[code]===== DaCapo antlr starting =====
===== DaCapo antlr PASSED in 4630 msec =====
===== DaCapo bloat starting =====
===== DaCapo bloat PASSED in 61779 msec =====
===== DaCapo hsqldb starting =====
===== DaCapo hsqldb PASSED in 5079 msec =====
===== DaCapo jython starting =====
===== DaCapo jython PASSED in 24531 msec =====
===== DaCapo lusearch starting =====
===== DaCapo lusearch PASSED in 3396 msec =====
===== DaCapo luindex starting =====
===== DaCapo luindex PASSED in 6203 msec =====
===== DaCapo pmd starting =====
===== DaCapo pmd PASSED in 24445 msec =====
===== DaCapo xalan starting =====
===== DaCapo xalan PASSED in 14990 msec =====[/code]T

ANN

- ANNs learn from sequences to predict unseen patterns in NLP.
- In this paper ANN sequence learning techniques are applied to study sequences of cache miss rates and find the predictability of these sequences and how they vary across programs.

Data Set

For the SPEC CPU 2000 programs and DaCapo Java programs, traces of every memory access made by the program in Valgrind (Lackey tool) is captured.

Traces of memory access are just processor accesses giving information like:

<time> <scale> {<cpu>} M<rw><sz><attrib> <addr> <data>

Valgrind is an instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail.

```
tutorialadda@tutorialadda:~/valgrind$
tutorialadda@tutorialadda:~/valgrind$ gcc -o test test.c -g
tutorialadda@tutorialadda:~/valgrind$
tutorialadda@tutorialadda:~/valgrind$ valgrind --tool=memcheck --leak-check=yes  ./test
==11345== Memcheck, a memory error detector
==11345== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==11345== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==11345== Command: ./test
==11345==
==11345==
==11345== HEAP SUMMARY:
==11345==     in use at exit: 0 bytes in 0 blocks
==11345==   total heap usage: 2 allocs, 2 frees, 25 bytes allocated
==11345==
==11345== All heap blocks were freed -- no leaks are possible
==11345==
==11345== For counts of detected and suppressed errors, rerun with: -v
==11345== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
tutorialadda@tutorialadda:~/valgrind$
```

They mapped virtual addresses to their 64-byte virtual cache line.

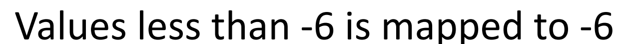
Applied the least recently used (LRU) stack algorithm [4] to obtain miss rates for various cache sizes.

The rates are aggregated over windows of 1,000,000 instructions, for instruction accesses only, data only, and both. Thus they obtain six sequences of numbers in the range [0,1] from each trace (two cache line sizes × instruction only, data only, both).

Data preprocessing

The cache miss rates are transformed using log10 to show the miss rates close to 0.

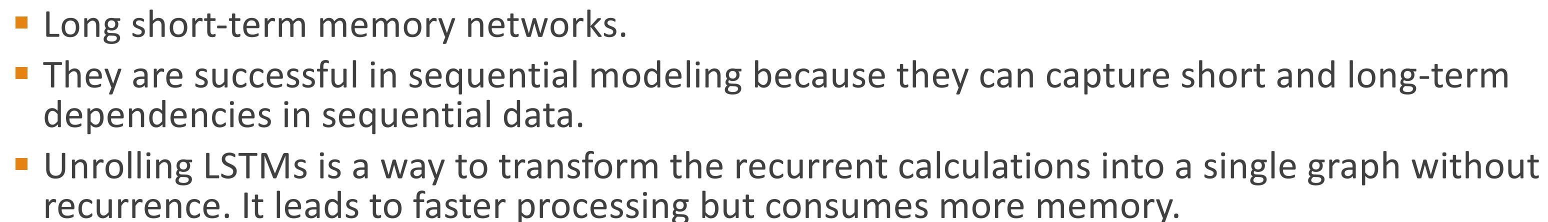
To avoid 0 a small epsilon is added to the miss rates.



Models

- Three different kinds of ANNs are used.
- All models have the following characteristics:
  - They are auto-regressive.
  - Have discretized representation in the input and output space.
  - Are commonly applied to sequence learning tasks.

LSTM:

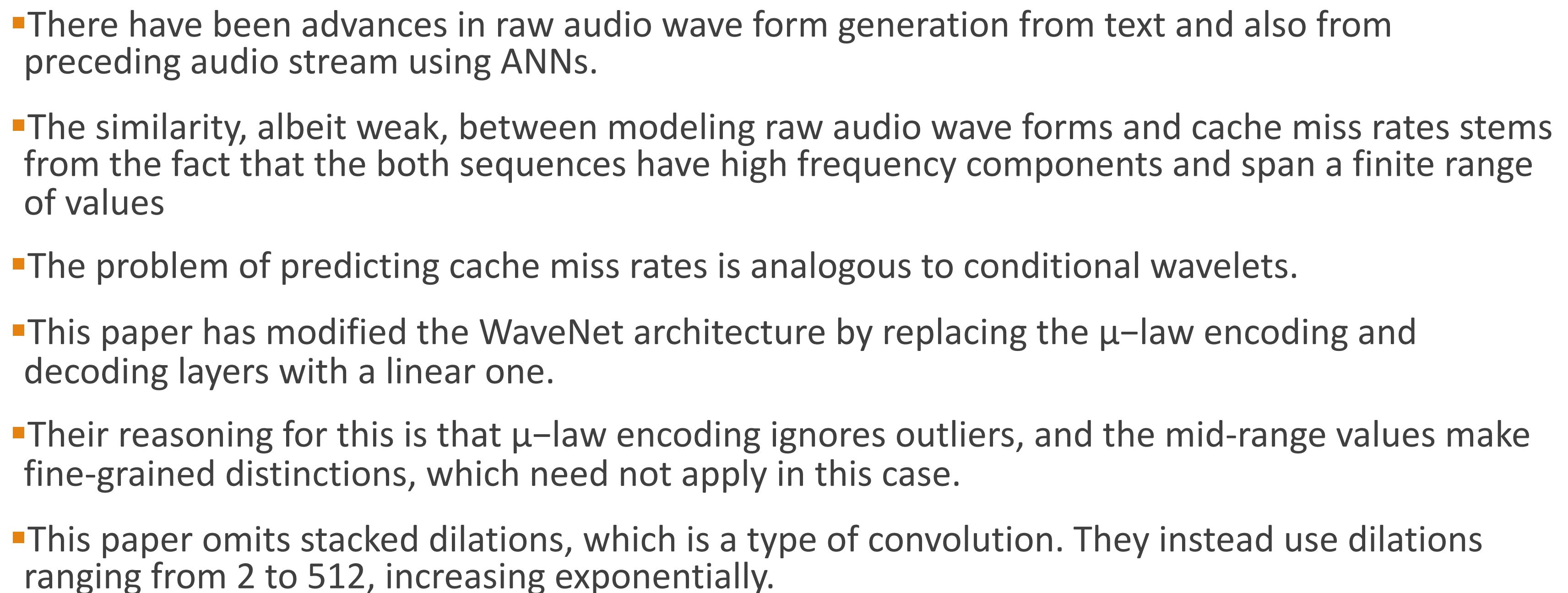


- Long short-term memory networks.
- They are successful in sequential modeling because they can capture short and long-term dependencies in sequential data.
- Unrolling LSTMs is a way to transform the recurrent calculations into a single graph without recurrence. It leads to faster processing but consumes more memory.

LSTM contd…

- Doing this beyond a particular time step in the history of a sequence leads to heavy computation and vanishing gradient issues.
- When there are more layers in the network, the value of the product of the derivative decreases until at some point the partial derivative of the loss function approaches a value close to zero, and the partial derivative vanishes. This is called the vanishing gradient problem.
- The model accounts for this by allowing unrolling up to a finite number of time steps.
- This model consists of an LSTM layer followed by three fully connected layers.
- The hidden state is forwarded to the next prediction of the model.
- The hidden state contains information about the values prior to the current time step.

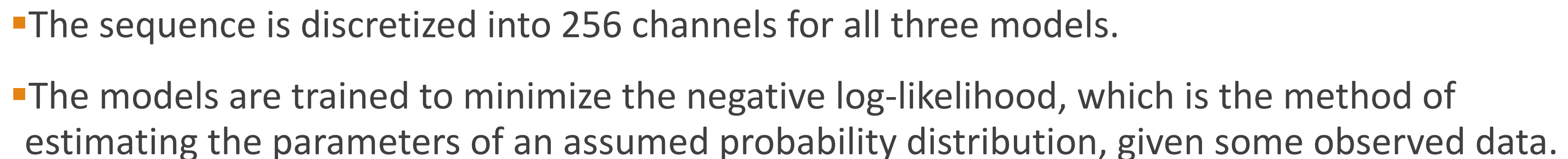
WaveNet



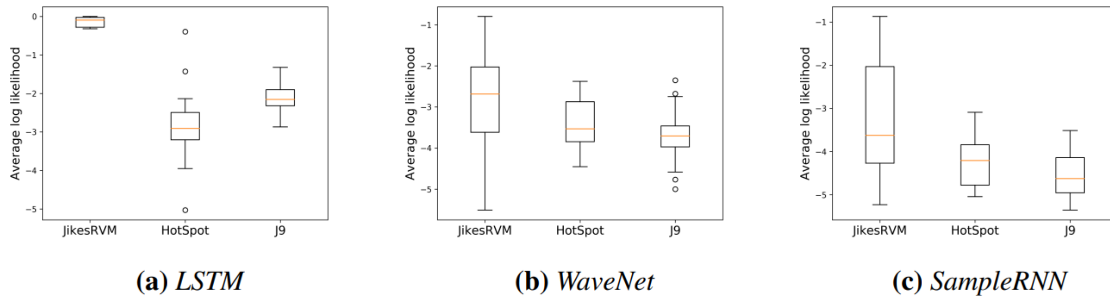
Sample RNN

- Sample RNN has achieved good results for audio waveform generation.
- Unlike WaveNet it uses RNNs at different times to model long-term dependencies instead of dilated convolutions.

Experimental Results and Insights



- The sequence is discretized into 256 channels for all three models.
- The models are trained to minimize the negative log-likelihood, which is the method of estimating the parameters of an assumed probability distribution, given some observed data.



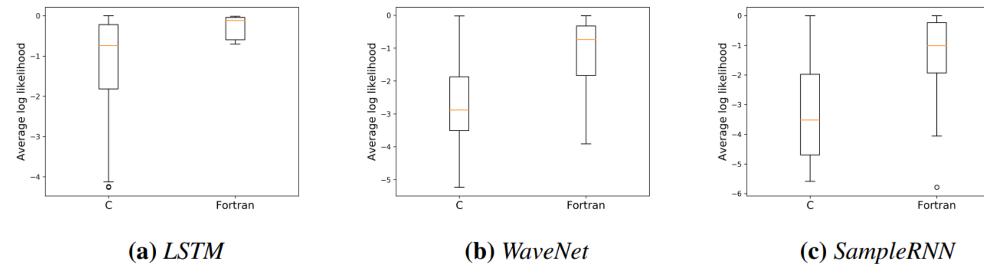
(a) *LSTM* **(b)** *WaveNet* **(c)** *SampleRNN*

Figure 4: *Log Likelihood for Java Virtual Machines*

The three VMs have different likelihood estimates.

The majority of the memory traces can be ordered as HotSpot, J9, and Jikes RVM (lowest to highest likelihood) and considering that log-likelihoods correspond to the predictability of the traces Jikes RVM is seen to have the highest predictability and HotSpot the lowest.

This can be attributed to the fact that Jikes RVM uses only compiled execution while HotSpot combines interpretation and compilation, so its access patterns vary more.



(a) *LSTM* **(b)** *WaveNet* **(c)** *SampleRNN*

Figure 3: *Log Likelihood for C and Fortran Programs*

This figure shows how C/Fortran traces differ in log-likelihood.

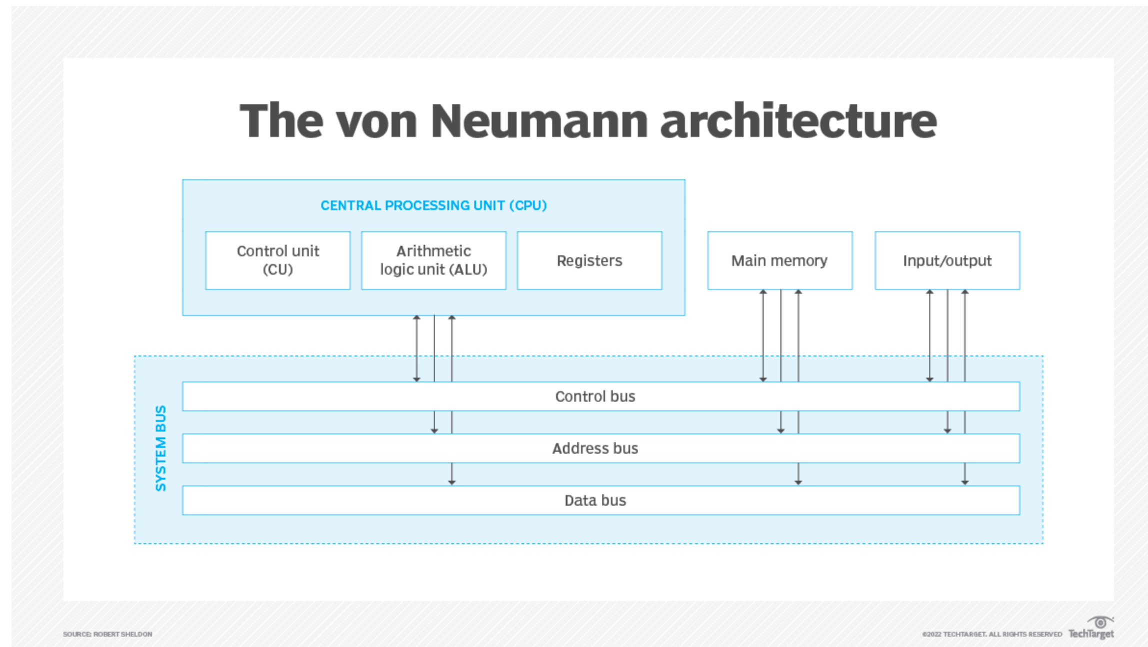
Fortran programs show very high predictability compared to C programs, which are spread across the likelihood spectrum.

The higher predictability of Fortran traces may be because many Fortran programs emphasize regular processing across dense arrays, while C programs lean toward pointer-linked data structures, whose accesses will be more scattered across the memory.

Learning Memory Access Patterns

What is this paper about?

- This paper demonstrates the potential of deep learning to address the Von Neumann bottleneck of memory performance.



The goal is to construct accurate and efficient memory prefetchers while focusing on learning memory access patterns.

Microarchitectural Prefetchers

Prefetchers are hardware structures that predict future memory access from past history.

They can be separated into two categories: stride prefetchers and correlation prefetchers.

Stride prefetchers are commonly implemented in modern processors and lock onto stable, repeatable deltas (differences between subsequent memory addresses).

Correlation prefetchers try to learn patterns that may repeat, but are not as consistent.
They store the past history of memory accesses in large tables and are better at predicting more irregular patterns than stride prefetchers.

Correlation prefetchers require large, costly tables, and are typically not implemented in modern multi-core processors

Recurrent Neural Networks

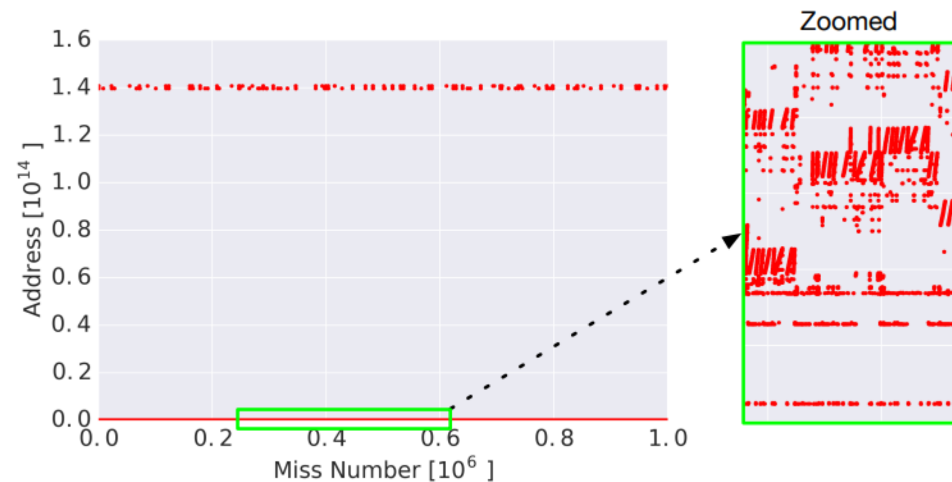
- Deep learning is widely used these days for sequential prediction problems.
- LSTMs have emerged as a popular RNN variant.
- An LSTM is composed of a hidden state **h** and a cell state **c**, along with input **i**, forget **f**, and output gates **o** that dictate what information gets stored and propagated to the next time step. At timestep **N**, input **xN** is presented to the LSTM, and the LSTM states are computed using the following process:
  - 1. Compute the input, forget, and output gates
  - 2. Update the cell state
  - 3. Compute the LSTM hidden (output) state
- The above process forms a single LSTM layer
- LSTM layers can be further stacked so that the output of one LSTM layer at time N becomes the input to another LSTM layer at time N, allowing for greater modeling flexibility with relatively few extra parameters.

The prediction problem of prefetching

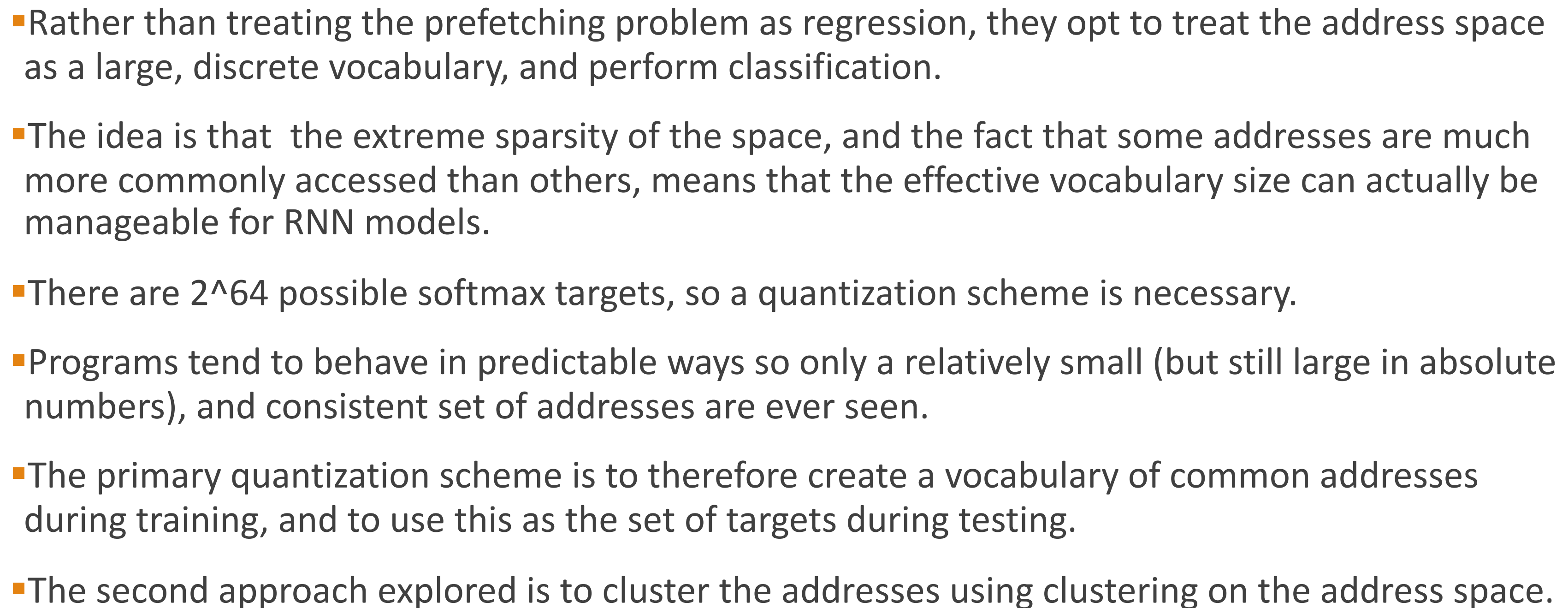
- Prefetching is the process of predicting future memory accesses that will miss in the on-chip cache and access memory based on past history.
- Each of these memory addresses are generated by a memory instruction (a load/store).
- Memory instructions are a subset of all instructions that interact with the addressable memory of the computer system.
- Many hardware proposals use two features to make these prefetching decisions: the sequence of caches miss addresses that have been observed so far and the sequence of instruction addresses, also known as program counters (PCs), that are associated with the instruction that generated each of the cache miss addresses.
- PCs are unique tags, that is each PC is unique to a particular instruction that has been compiled from a particular function in a particular code file

- An initial model could use two input features at a given timestep N.
- It could use the address and PC that generated a cache miss at that timestep to predict the address of the miss at timestep N + 1.
- One concern with this approach is that: the address space of an application is extremely sparse.
- In the training data with 100M cache misses, only 10M unique cache block miss addresses appear on average out of the entire 2^64 physical address space.

- This can be observed in the figure
- This is an example trace omnetpp (a benchmark from the standard SPEC CPU2006 benchmark suite).
- The wide range and severely multi-modal nature of this space makes it a challenge for time-series regression models.



- Neural networks tend to work best with normalized inputs, however when normalizing this data, the finite precision floating-point representation results in a significant loss of information. This issue affects the modeling at both the input and output levels.



Prefetching as classification

- Rather than treating the prefetching problem as regression, they opt to treat the address space as a large, discrete vocabulary, and perform classification.
- The idea is that the extreme sparsity of the space, and the fact that some addresses are much more commonly accessed than others, means that the effective vocabulary size can actually be manageable for RNN models.
- There are 2^64 possible softmax targets, so a quantization scheme is necessary.
- Programs tend to behave in predictable ways so only a relatively small (but still large in absolute numbers), and consistent set of addresses are ever seen.
- The primary quantization scheme is to therefore create a vocabulary of common addresses during training, and to use this as the set of targets during testing.
- The second approach explored is to cluster the addresses using clustering on the address space.

- Due to dynamic side-effects such as address space layout randomization (ASLR), different runs of the same program will lead to different raw address accesses.
- However, **given a layout**, the program will behave in a consistent manner.
- Therefore, one potential strategy is to predict deltas, $\Delta N = Addr_{N+1} - Addr_N$, instead of addresses directly.
- These will remain consistent across program executions, and come with the benefit that the number of uniquely occurring deltas is often orders of magnitude smaller than uniquely occurring addresses.

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

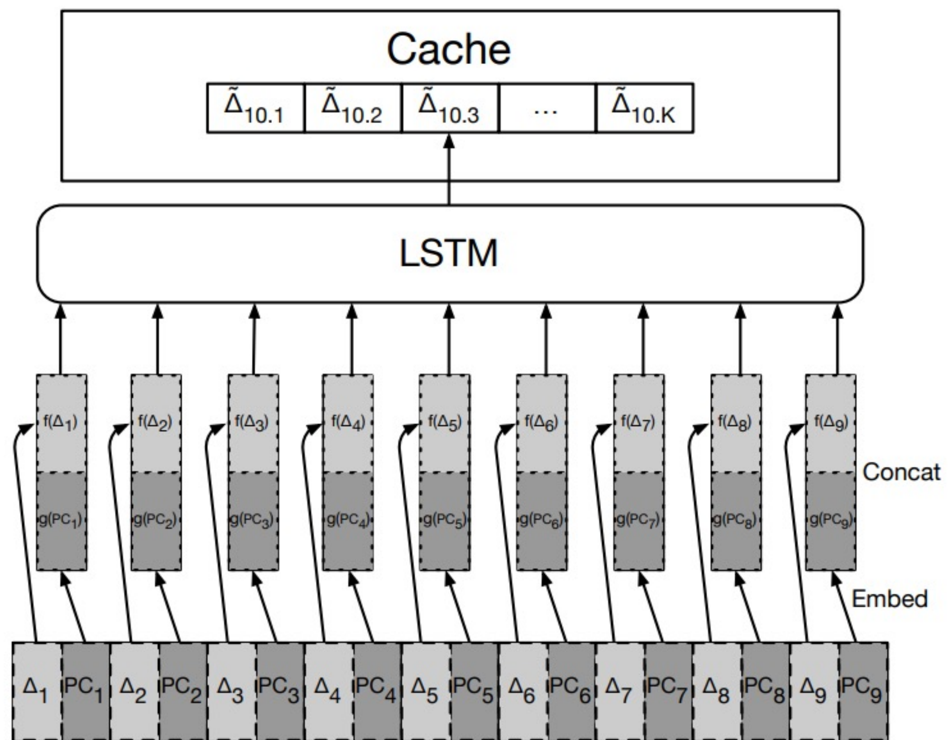
Models

This paper introduces two LSTM-based prefetching models.

The first version is analogous to a standard language model, while the second exploits the structure of the memory access space in order to reduce the vocabulary size and reduce the model memory footprint.

Embedding LSTM

- In this model, the authors restricted the output vocabulary size to model the most frequently occurring deltas.
- According to Table 1, the size of the vocabulary required in order to obtain at best 50% accuracy is usually O(1000) or less, well within the capabilities of standard language models.
- This first model therefore restricts the output vocabulary size to 50,000 of the most frequent, unique deltas.
- For the input vocabulary, we include all deltas as long as they appear in the dataset at least 10 times.
- This model is referred to as the embedding LSTM.
- It uses a categorical (one-hot) representation for both the input and output deltas.

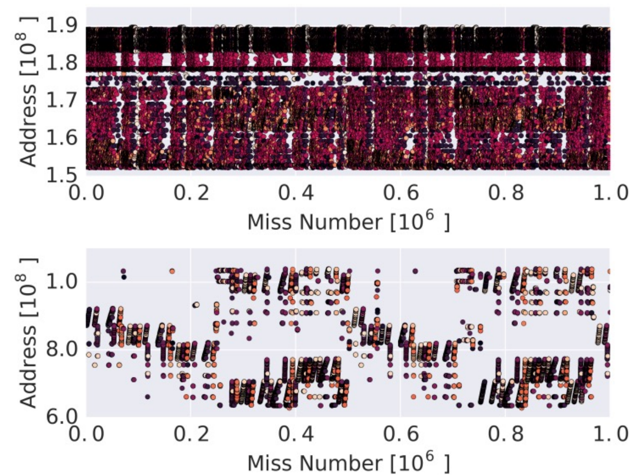


Cache
$\tilde{\Delta}_{10.1}$
$\tilde{\Delta}_{10.2}$
$\tilde{\Delta}_{10.3}$
...
$\tilde{\Delta}_{10.K}$
LSTM
$f(\Delta_1)$ $f(\Delta_2)$ $f(\Delta_3)$ $f(\Delta_4)$ $f(\Delta_5)$ $f(\Delta_6)$ $f(\Delta_7)$ $f(\Delta_8)$ $f(\Delta_9)$
Concat
$g(PC_1)$ $g(PC_2)$ $g(PC_3)$ $g(PC_4)$ $g(PC_5)$ $g(PC_6)$ $g(PC_7)$ $g(PC_8)$ $g(PC_9)$
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

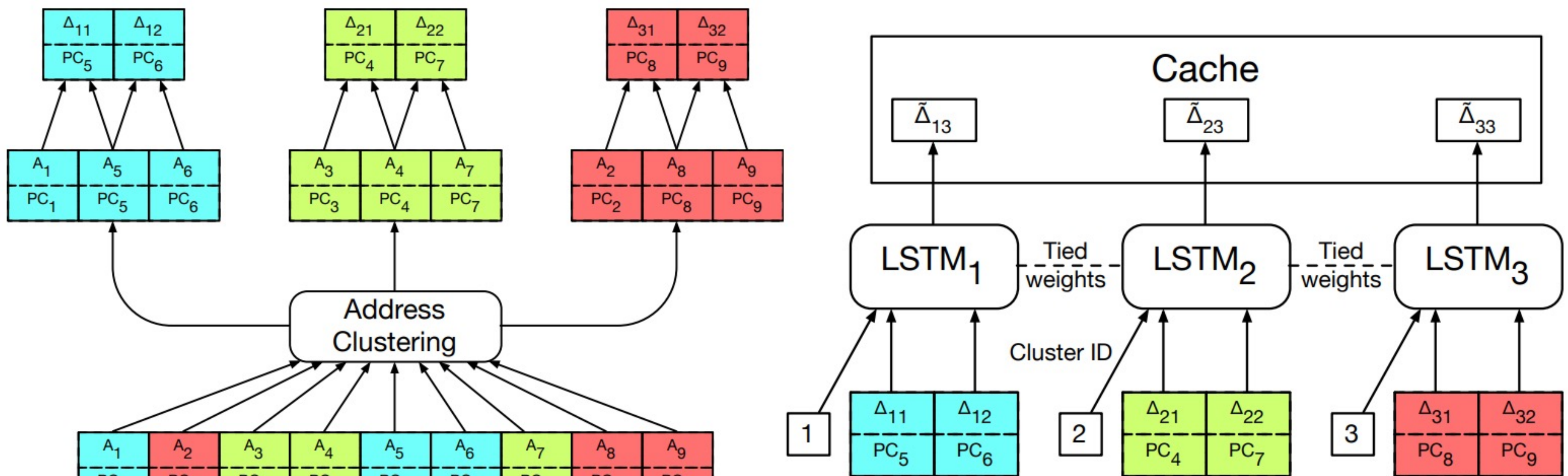
- A prefetcher can return several predictions. This creates a trade-off, where more predictions increase the probability of a cache hit at the next timestep.
- The authors opt to prefetch the top-10 predictions of the LSTM at each timestep.
- LIMITATIONS:
  - A large vocabulary increases the model’s computational and storage footprint.
  - Truncating the vocabulary necessarily puts a ceiling on the accuracy of the model.

Clustering + LSTM

- This model is looking at the narrower regions of the address space.
- They took the set of addresses from omnetpp and clustered them into 6 different regions using k-mean.



- They first cluster the raw address space using K-means. The data is then partitioned into these clusters, and deltas are computed within each cluster.



(a) Clustering the address space into separate streams.

(b) The clustering + LSTM model.

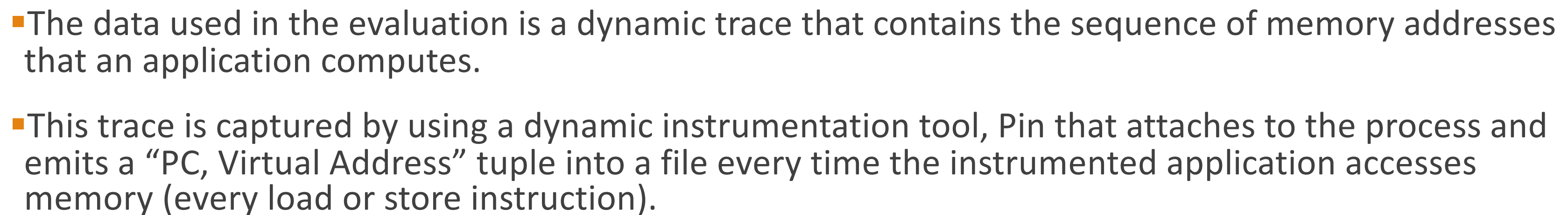
Figure 4. The clustering + LSTM data processing and model.

- The set of deltas within a cluster is significantly smaller than the global vocabulary, reducing some issues with embedding LSTM.
- Here an LSTM is used to model each cluster independently, but the weights of the LSTMs are tied together.
- Cluster ID is passed as an additional feature, which effectively gives each LSTM a different set of biases.
- The resulting deltas can be effectively normalized and used as real-valued inputs to the LSTM.
- This allows further reduction in the size of the model, as we do not need to keep around a large matrix of embeddings.
- The trade-offs are that it requires an additional step of pre-processing to cluster the address space.
- And it cannot model the dynamics that cause the program to access different regions of the address space.

Experiments

- A necessary condition for neural networks to be effective prefetchers is that they must be able to accurately predict cache misses.
- The experiments here measure their effectiveness in this task when compared with traditional hardware.

Data Collection



- The data used in the evaluation is a dynamic trace that contains the sequence of memory addresses that an application computes.
- This trace is captured by using a dynamic instrumentation tool, Pin that attaches to the process and emits a “PC, Virtual Address” tuple into a file every time the instrumented application accesses memory (every load or store instruction).

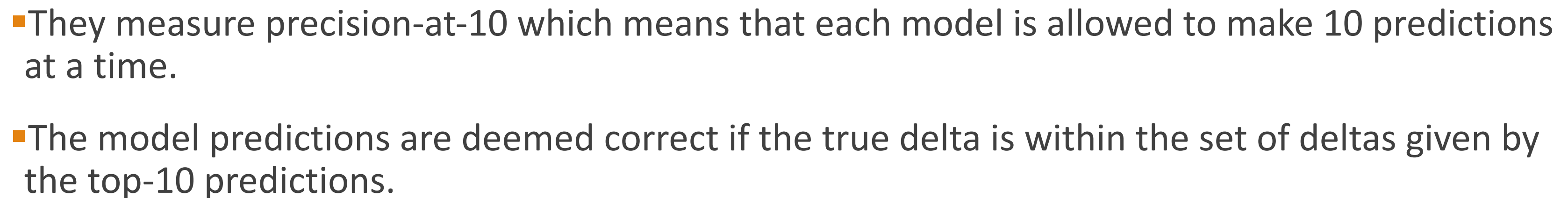
- This raw access trace mostly contains accesses that hit in the cache (such as stack accesses, which are present in the data cache). Since we are focused on predicting cache misses, we obtain the sequence of cache misses by simulating this trace through a simple cache simulator that emulates an Intel Broadwell microprocessor.
- Here they use the memory intensive applications of SPEC CPU2006. This is a standard benchmark suite that is used pervasively to evaluate the performance of computer systems. However, SPEC CPU2006 also has small working sets when compared to modern datacenter workloads. Therefore in addition to SPEC benchmarks, they also include Google's websearch workload. Websearch is a unique application with complex access patterns

Experimental Setup

- Each trace is split into a training and testing set using 70% for training and 30% for evaluation, and train each LSTM on each dataset independently.

Metrics

Precision:



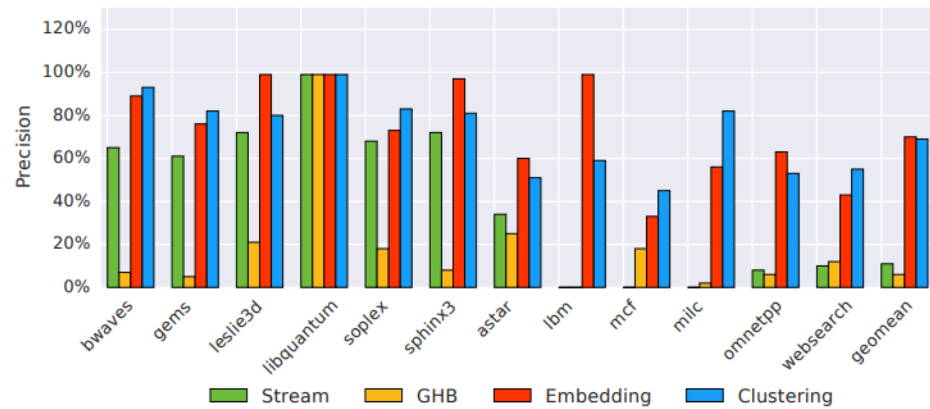
- They measure precision-at-10 which means that each model is allowed to make 10 predictions at a time.
- The model predictions are deemed correct if the true delta is within the set of deltas given by the top-10 predictions.

Recall:

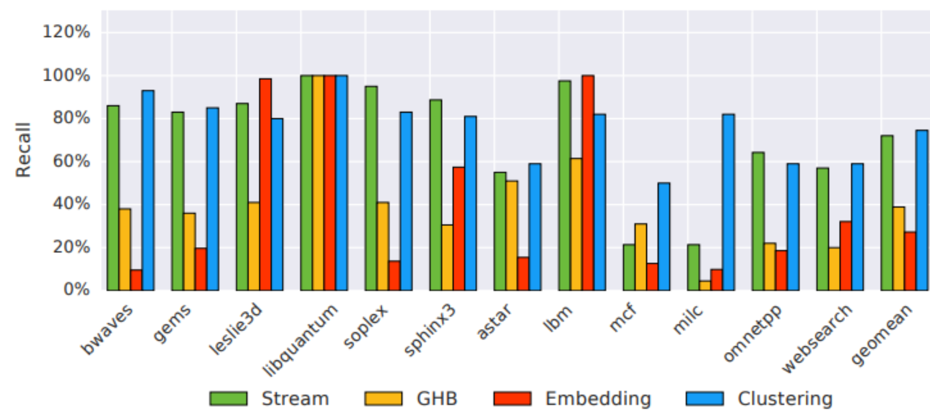
- They measure recall-at-10. Each time the model makes predictions, they record this set of 10 deltas.
- At the end, we measure the recall as the cardinality of the set of predicted deltas over the entire set seen at test-time.

Model Comparison:

- They compare the LSTM-based prefetchers to two state-of-the-art hardware prefetchers.
- The first is a standard stream prefetcher. They simulated a hardware structure that supports up to 10 simultaneous streams.
- The second is a GHB PC/DC prefetcher. This prefetcher excels at more complex memory access patterns, but has much lower recall than the stream prefetcher



(a) Precision



(b) Recall

Conclusion

Exploiting the benefits of learning and predicting program behavior to unlock control and data parallelism is not a new concept. However, the conventional approach of table based predictors, is too costly to scale for data intensive irregular workloads. The models described in this paper demonstrate significantly higher precision and recall than table-based approaches.

There is a notion of timeliness that is also an important consideration. One simple heuristic is to predict several steps ahead, instead of just the next step. This would be similar to the behavior of stream prefetchers.