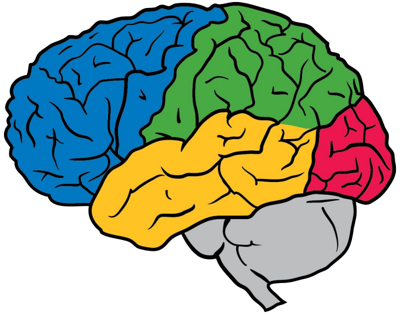


Machine Learning for Systems and Systems for Machine Learning

Jeff Dean
Google Brain team
g.co/brain

Presenting the work of **many** people at Google

Machine Learning for Systems

Learning Should Be Used Throughout our Computing Systems

Traditional low-level systems code (operating systems, compilers, storage systems) **does not** make extensive use of machine learning today

This should change!

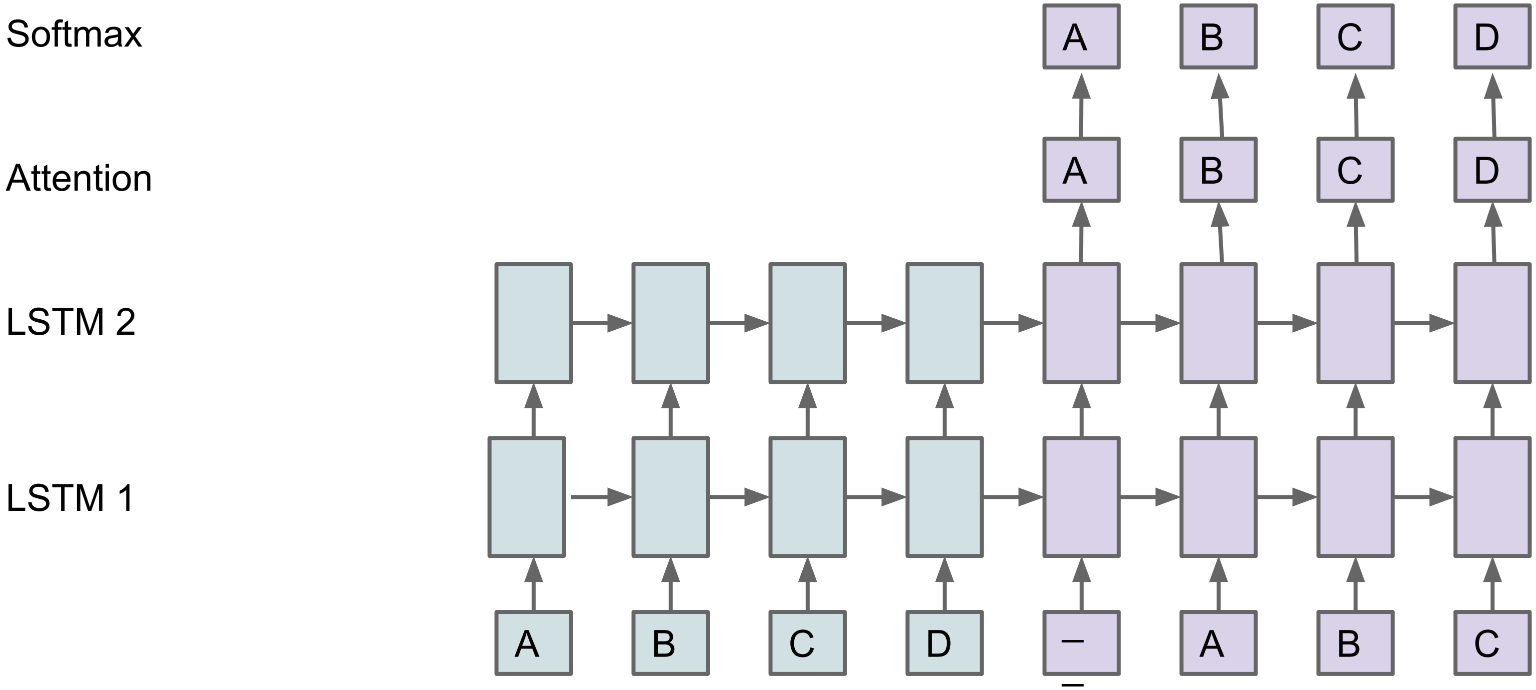
A few examples and some opportunities...

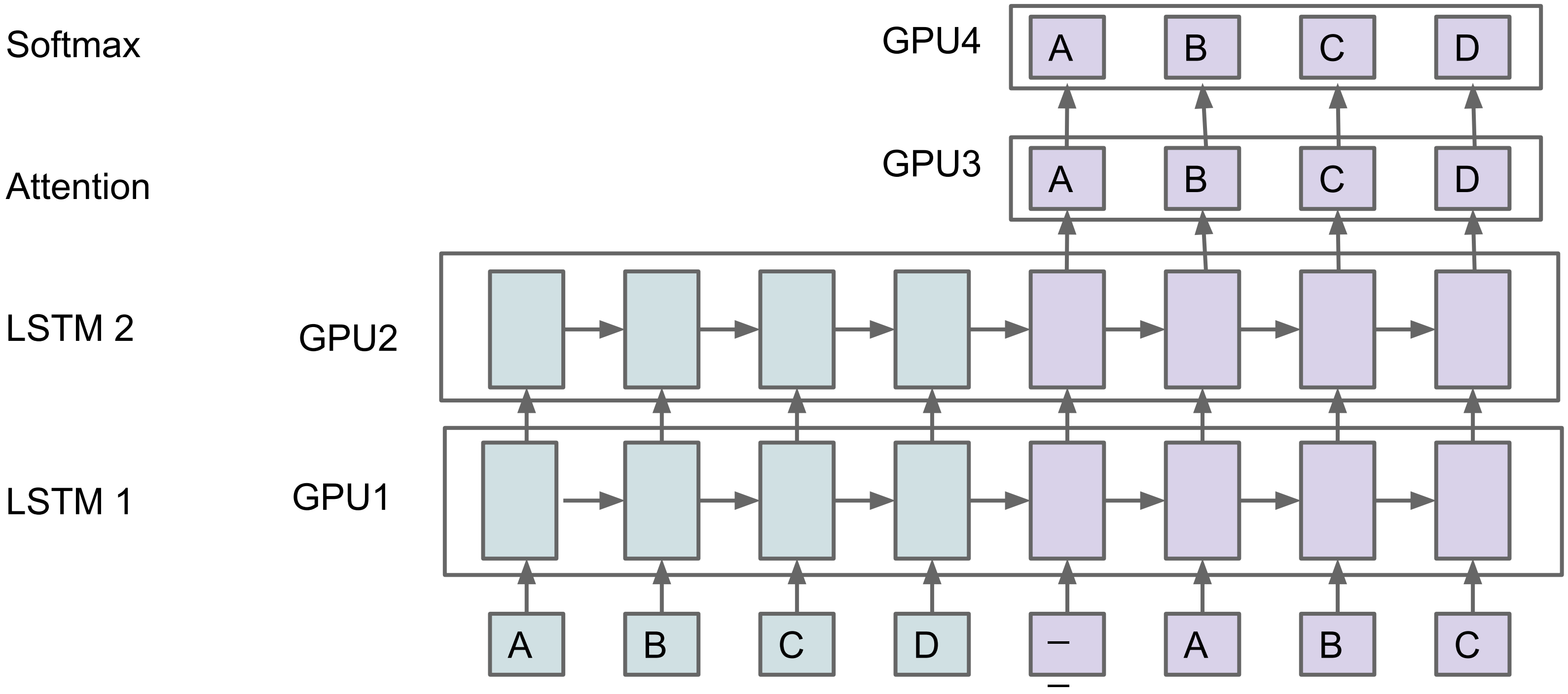
Machine Learning for
Higher Performance Machine Learning
Models

For large models, model parallelism is important

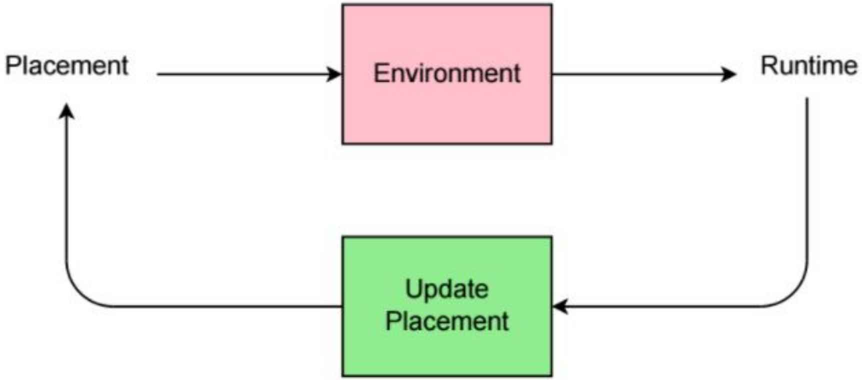
For large models, model parallelism is important

But getting good performance given multiple computing devices is non-trivial and non-obvious


Softmax
A B C D
Attention
A B C D
LSTM 2
LSTM 1
A B C D _ A B C


Softmax
Attention
LSTM 2
LSTM 1
GPU4
GPU3
GPU2
GPU1
A
B
C
D
A
B
C
D
A
B
C
D
_
A
B
C

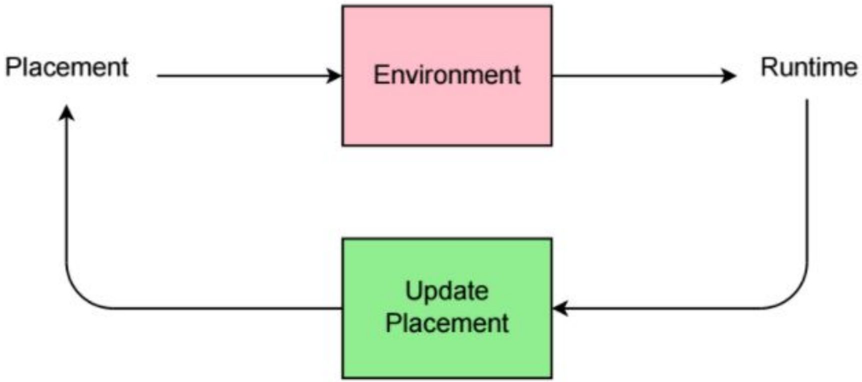
Reinforcement Learning for Higher Performance Machine Learning Models



Device Placement Optimization with Reinforcement Learning,
Azalia Mirhoseini, Hieu Pham, Quoc Le, Mohammad Norouzi, Samy Bengio, Benoit Steiner, Yuefeng Zhou, Naveen Kumar, Rasmus Larsen, and Jeff Dean, ICML 2017, arxiv.org/abs/1706.04972

Reinforcement Learning for Higher Performance Machine Learning Models

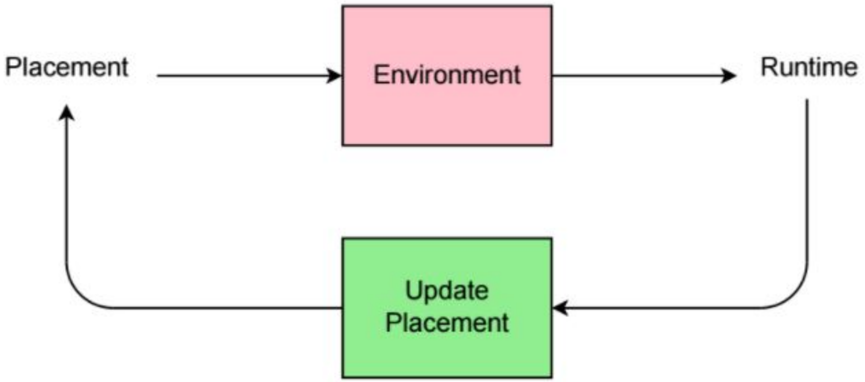
Placement model (trained via RL) gets graph as input + set of devices, outputs device placement for each graph node



Device Placement Optimization with Reinforcement Learning,
Azalia Mirhoseini, Hieu Pham, Quoc Le, Mohammad Norouzi, Samy Bengio, Benoit Steiner, Yuefeng Zhou, Naveen Kumar, Rasmus Larsen, and Jeff Dean, ICML 2017, arxiv.org/abs/1706.04972

Reinforcement Learning for Higher Performance Machine Learning Models

Placement model (trained via RL) gets graph as input + set of devices, outputs device placement for each graph node

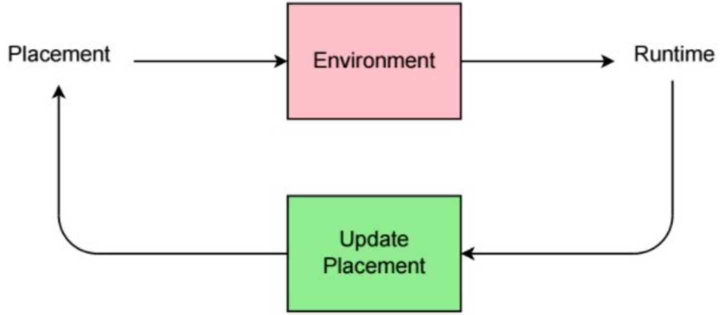


Measured time per step gives RL reward signal

Device Placement Optimization with Reinforcement Learning,
Azalia Mirhoseini, Hieu Pham, Quoc Le, Mohammad Norouzi, Samy Bengio, Benoit Steiner, Yuefeng Zhou, Naveen Kumar, Rasmus Larsen, and Jeff Dean, ICML 2017, arxiv.org/abs/1706.04972

Device Placement with Reinforcement Learning

Placement model (trained via RL) gets graph as input + set of devices, outputs device placement for each graph node



Measured time per step gives RL reward signal

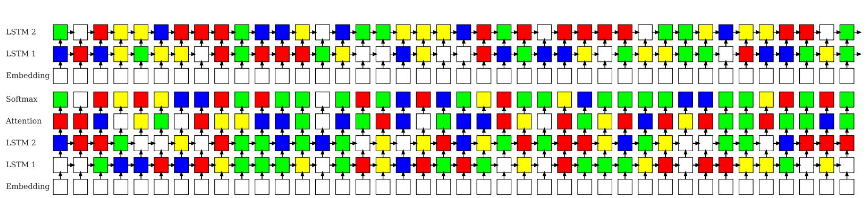


Figure 4. RL-based placement of Neural MT graph. Above: encoder, Below: decoder. Devices are denoted by colors, where the transparent color represents an operation on a CPU and each other unique color represents a different GPU. This placement achieves an improvement of 19.3% in running time compared to the fine-tuned hand-crafted placement.

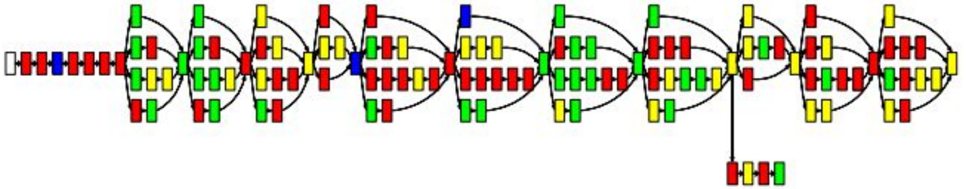


Figure 5. RL-based placement of Inception-V3. Devices are denoted by colors, where the transparent color represents an operation on a CPU and each other unique color represents a different GPU. RL-based placement achieves the improvement of 19.7% in running time compared to expert-designed placement.

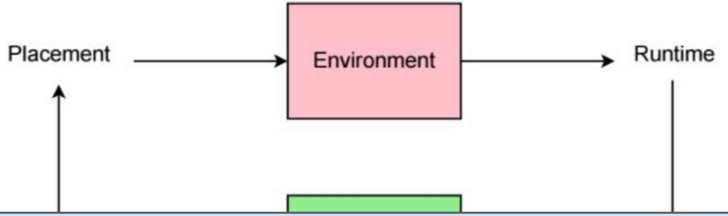
+19.3% faster vs. expert human for neural translation model

+19.7% faster vs. expert human for InceptionV3 image model

Device Placement Optimization with Reinforcement Learning,
Azalia Mirhoseini, Hieu Pham, Quoc Le, Mohammad Norouzi, Samy Bengio, Benoit Steiner, Yuefeng Zhou, Naveen Kumar, Rasmus Larsen, and Jeff Dean, ICML 2017, arxiv.org/abs/1706.04972

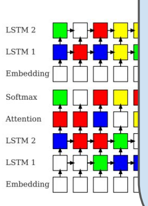
Device Placement with Reinforcement Learning

Placement model (trained via RL) gets graph as input + set of devices, outputs device placement for each graph



Measured time per step gives RL reward signal

Plug: Come see Azalia Mirhoseini's talk on ***"Learning Device Placement"*** tomorrow at 1:30 PM in the *Deep Learning at Supercomputing Scale* workshop in **101B**



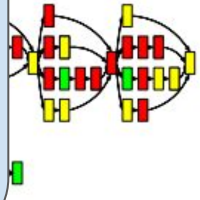


Figure 4. RL-based plac… transparent color represents an operation on a CPU and each other unique color represents a different GPU. This placement achieves an improvement of 19.3% in running time compared to the fine-tuned hand-crafted placement.

…r represents an operation on a CPU and each other unique color represents a different GPU. RL-based placement achieves the improvement of 19.7% in running time compared to expert-designed placement.

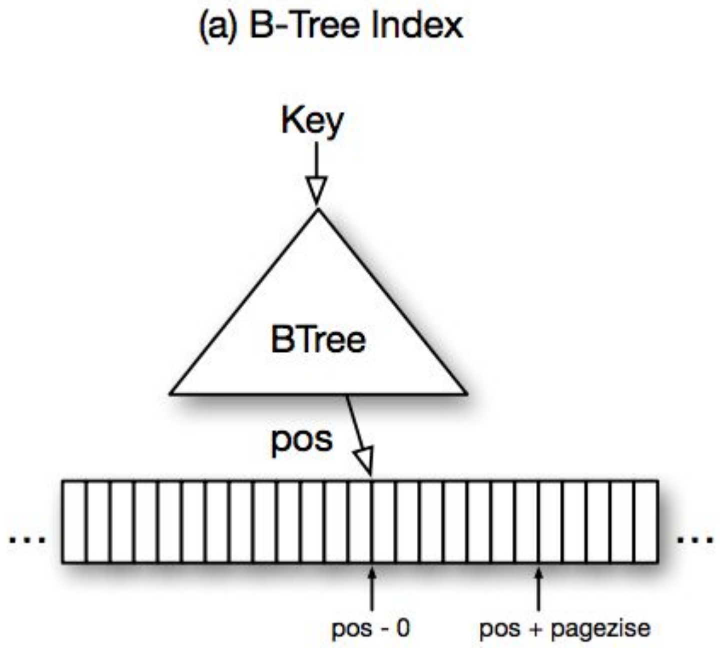
+19.3% faster vs. expert human for neural translation model

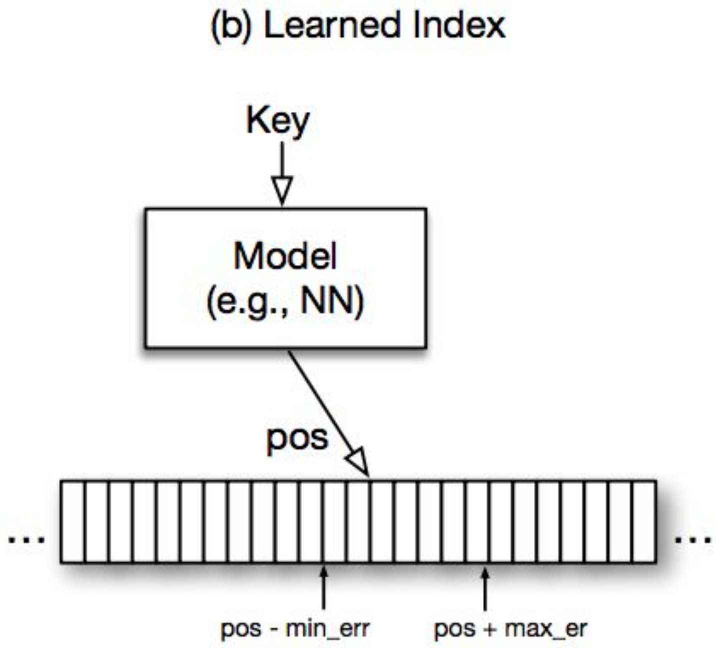
+19.7% faster vs. expert human for InceptionV3 image model

Device Placement Optimization with Reinforcement Learning,
Azalia Mirhoseini, Hieu Pham, Quoc Le, Mohammad Norouzi, Samy Bengio, Benoit Steiner, Yuefeng Zhou, Naveen Kumar, Rasmus Larsen, and Jeff Dean, ICML 2017, arxiv.org/abs/1706.04972

Learned Index Structures
not
Conventional Index Structures

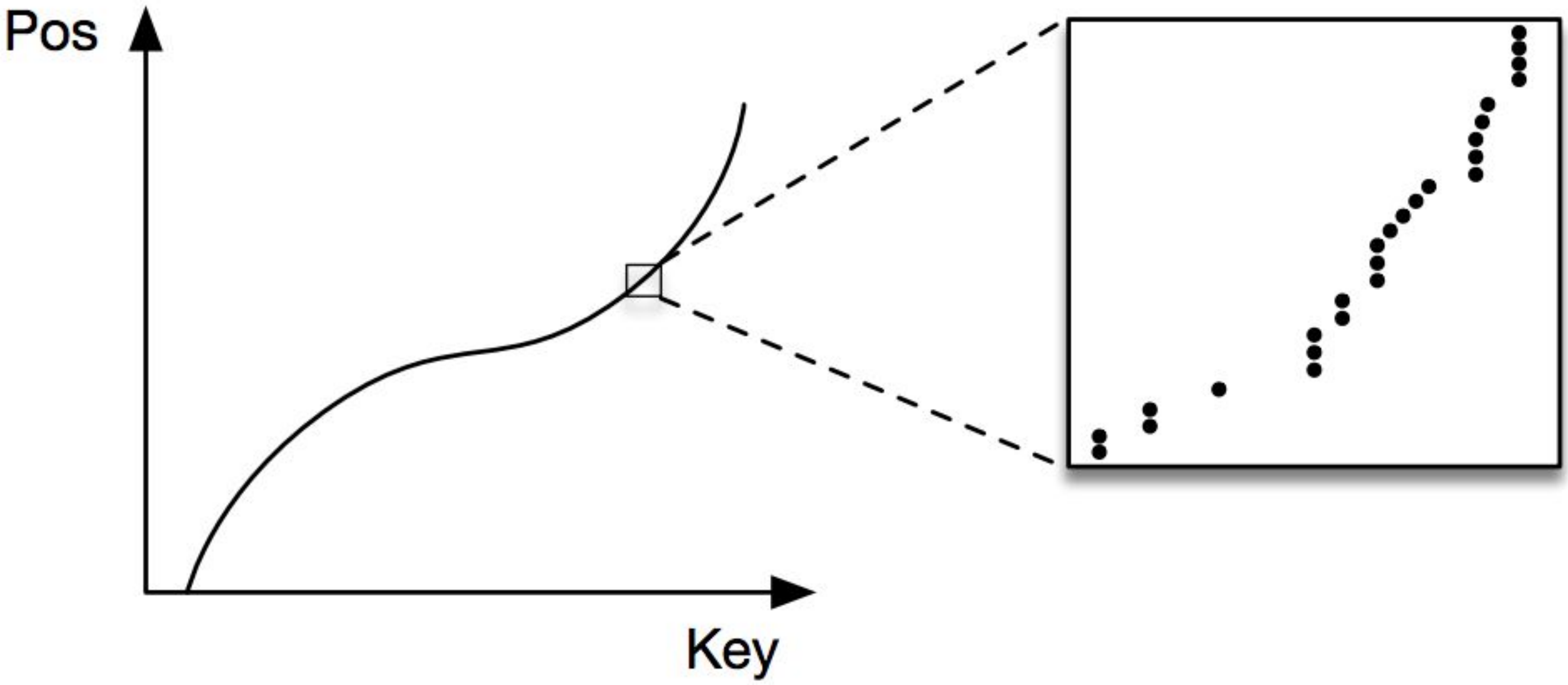
B-Trees are Models





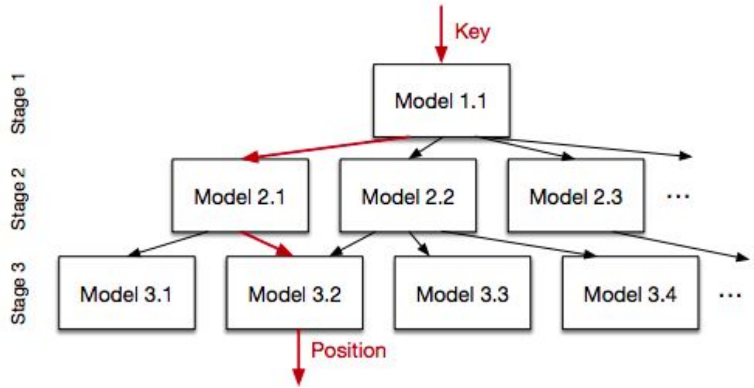
The Case for Learned Index Structures, Tim Kraska, Alex Beutel, Ed Chi, Jeffrey Dean & Neoklis Polyzotis, arxiv.org/abs/1712.01208

Indices as CDFs



The Case for Learned Index Structures, Tim Kraska, Alex Beutel, Ed Chi, Jeffrey Dean & Neoklis Polyzotis, arxiv.org/abs/1712.01208

Does it Work?

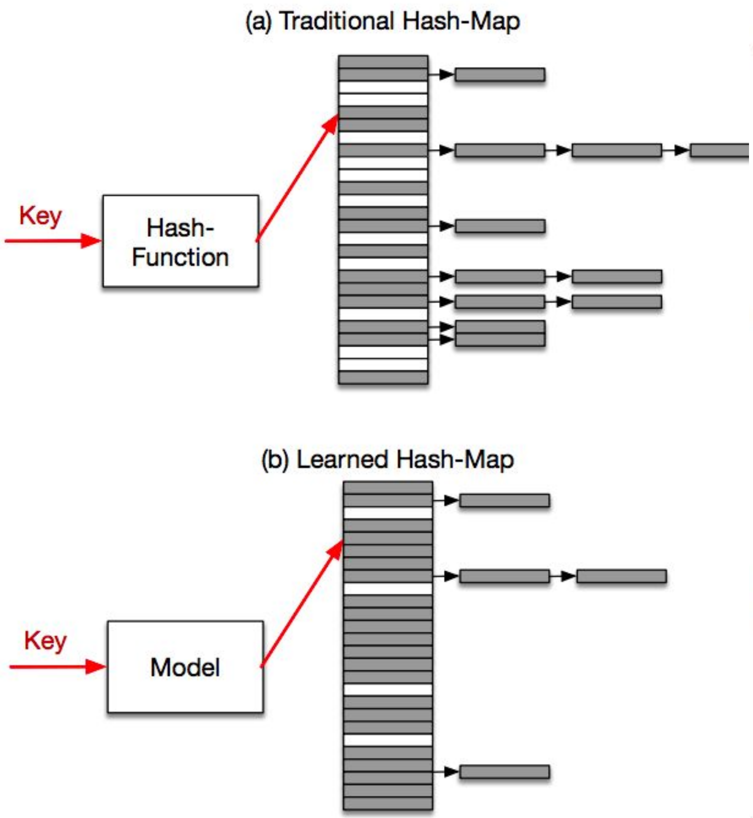


Index of 200M web service log records

Type	Config	Lookup time	Speedup vs. Btree	Size (MB)	Size vs. Btree
BTree	page size: 128	260 ns	1.0X	12.98 MB	1.0X
Learned index	2nd stage size: 10000	222 ns	1.17X	0.15 MB	0.01X
Learned index	2nd stage size: 50000	**162 ns**	**1.60X**	**0.76 MB**	**0.05X**
Learned index	2nd stage size: 100000	144 ns	1.67X	1.53 MB	0.12X
Learned index	2nd stage size: 200000	126 ns	2.06X	3.05 MB	0.23X

The Case for Learned Index Structures, Tim Kraska, Alex Beutel, Ed Chi, Jeffrey Dean & Neoklis Polyzotis, arxiv.org/abs/1712.01208

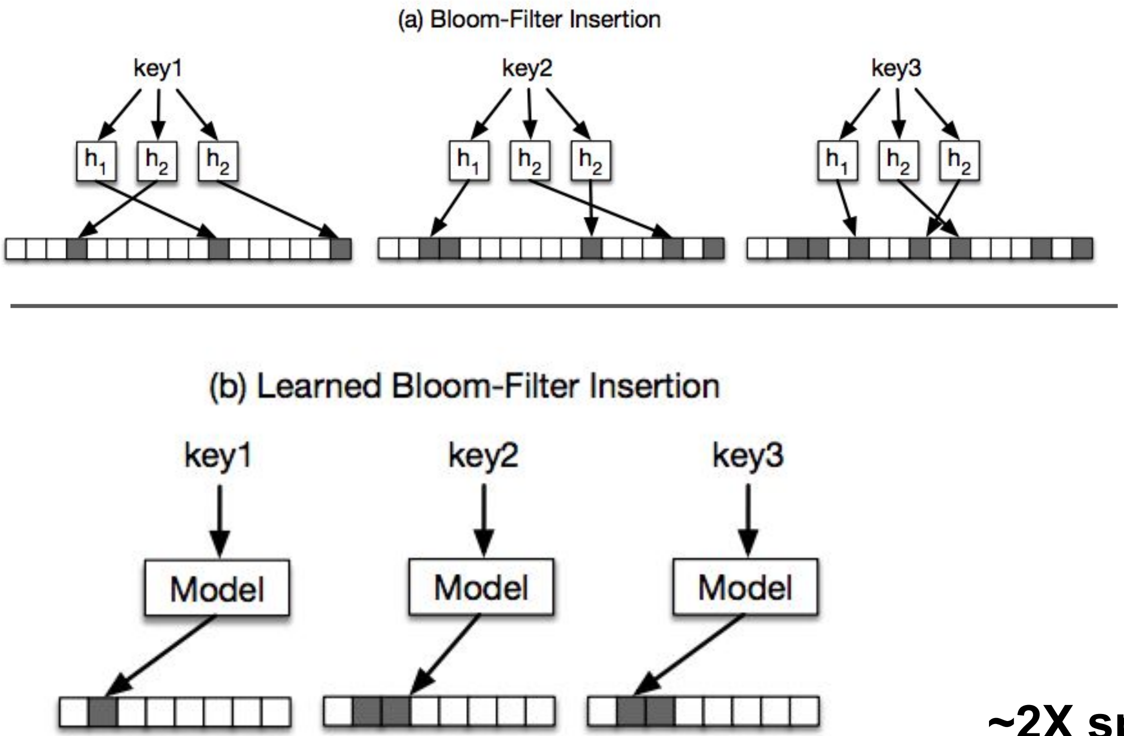
Hash Tables

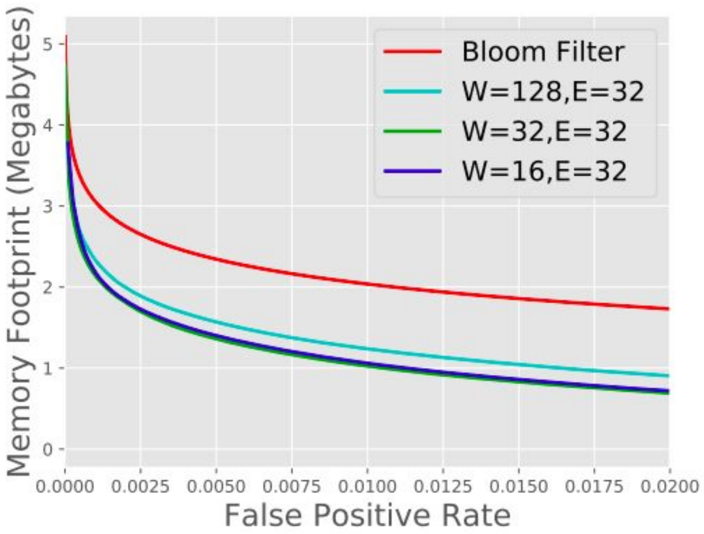


Dataset	Slots	Hash Type	Search Time (ns)	Empty Slots	Space Improvement
Map	75%	Model Hash	67	0.63GB (05%)	-20%
		Random Hash	52	0.80GB (25%)	
	100%	Model Hash	53	1.10GB (08%)	-27%
		Random Hash	48	1.50GB (35%)	
	125%	Model Hash	64	2.16GB (26%)	-6%
		Random Hash	49	2.31GB (43%)	
Web Log	75%	Model Hash	78	0.18GB (19%)	-78%
		Random Hash	53	0.84GB (25%)	
	100%	Model Hash	63	0.35GB (25%)	-78%
		Random Hash	50	1.58GB (35%)	
	125%	Model Hash	77	1.47GB (40%)	-39%
		Random Hash	50	2.43GB (43%)	
Log Normal	75%	Model Hash	79	0.63GB (20%)	-22%
		Random Hash	52	0.80GB (25%)	
	100%	Model Hash	66	1.10GB (26%)	-30%
		Random Hash	46	1.50GB (35%)	
	125%	Model Hash	77	2.16GB (41%)	-9%
		Random Hash	46	2.31GB (44%)	

The Case for Learned Index Structures, Tim Kraska, Alex Beutel, Ed Chi, Jeffrey Dean & Neoklis Polyzotis, arxiv.org/abs/1712.01208

Bloom Filters





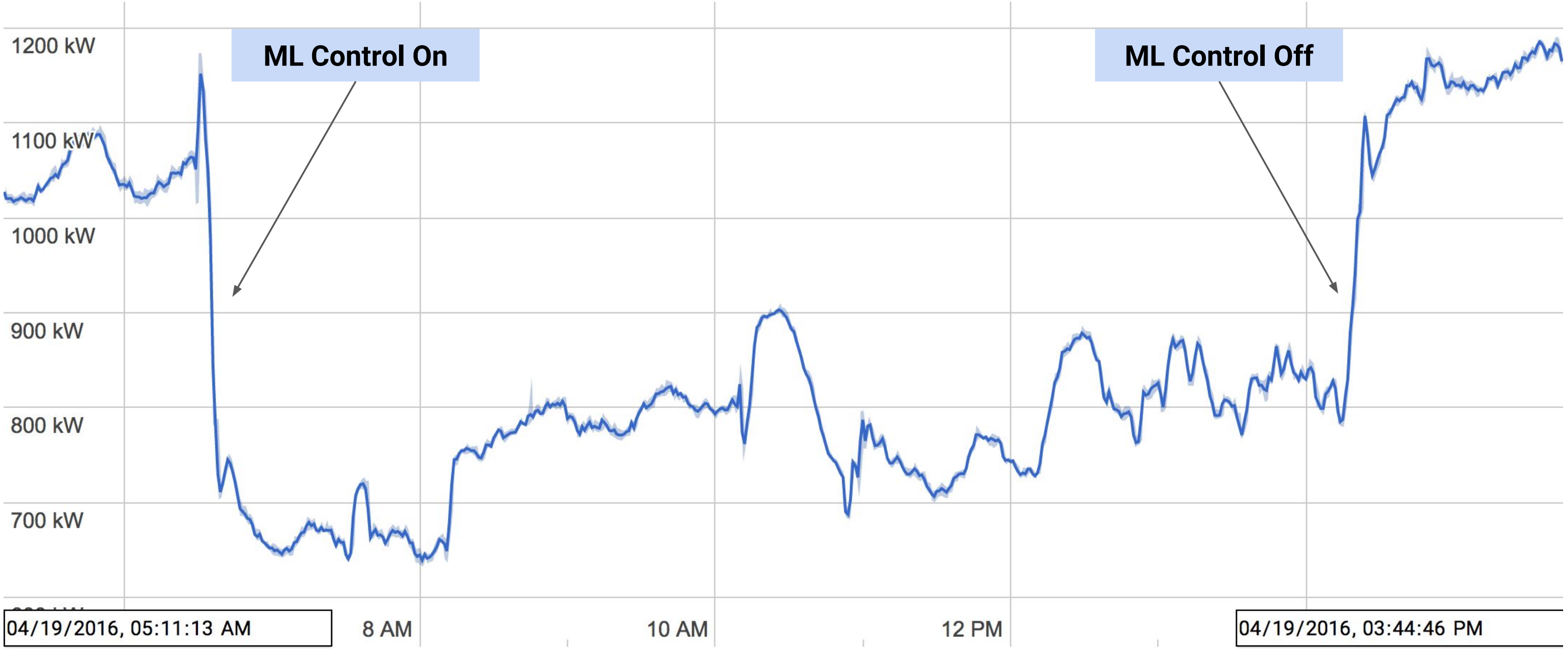
Model is simple RNN
W is number of units in RNN layer
E is width of character embedding

~2X space improvement over Bloom Filter at same false positive rate

The Case for Learned Index Structures, Tim Kraska, Alex Beutel, Ed Chi, Jeffrey Dean & Neoklis Polyzotis, arxiv.org/abs/1712.01208

Machine Learning for Improving Datacenter Efficiency

Machine Learning to Reduce Cooling Cost in Datacenters



Collaboration between DeepMind and Google Datacenter operations teams.
See https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/

Where Else Could We Use Learning?

Computer Systems are Filled With Heuristics

Compilers, Networking code, Operating Systems, …

Heuristics have to work well “in general case”

Generally don’t adapt to actual pattern of usage

Generally don’t take into account available context

Anywhere We're Using Heuristics To Make a Decision!

Compilers: instruction scheduling, register allocation, loop nest parallelization strategies, …

Networking: TCP window size decisions, backoff for retransmits, data compression, ...

Operating systems: process scheduling, buffer cache insertion/replacement, file system prefetching, …

Job scheduling systems: which tasks/VMs to co-locate on same machine, which tasks to pre-empt, ...

ASIC design: physical circuit layout, test case selection, …

Anywhere We've Punted to a User-Tunable Performance Option!

Many programs have huge numbers of tunable command-line flags, usually not changed from their defaults

```
--eventmanager_threads=16
--bigtable_scheduler_batch_size=8
--mapreduce_merge_memory=134217728
--lexicon_cache_size=1048576
--storage_server_rpc_freelist_size=128
...
```

Meta-learn everything

ML:

- learning placement decisions
- learning fast kernel implementations
- learning optimization update rules
- learning input preprocessing pipeline steps
- learning activation functions
- learning model architectures for specific device types, or that are fast for inference on mobile device X, learning which pre-trained components to reuse, …

Computer architecture/datacenter networking design:

- learning best design properties by exploring design space automatically (via simulator)

Keys for Success in These Settings

(1) Having a numeric metric to measure and optimize
(2) Having a clean interface to easily integrate learning into all of these kinds of systems

Current work: exploring APIs and implementations

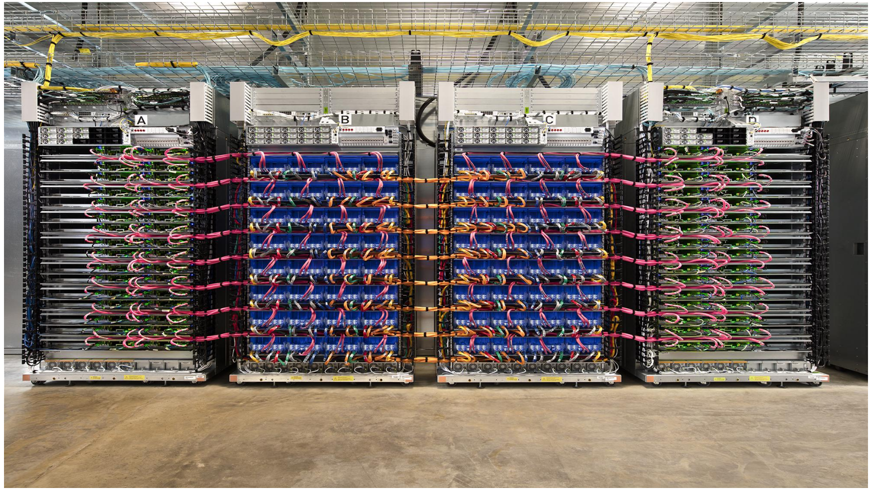
Basic ideas:

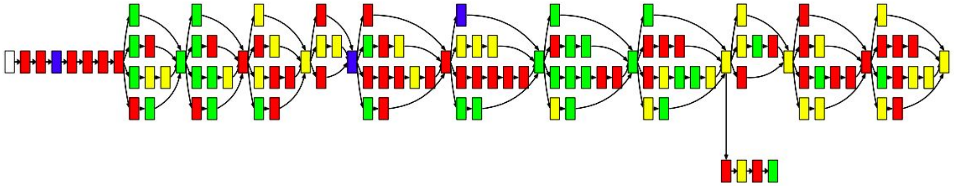
- Make a sequence of choices in some context
- Eventually get feedback about those choices
- Make this all work with very low overhead, even in distributed settings
- Support many implementations of core interfaces

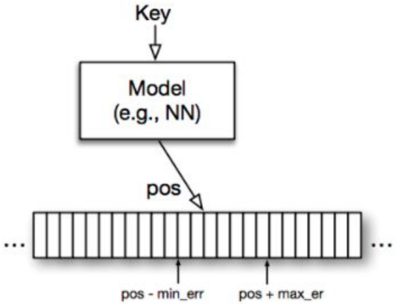
Conclusions

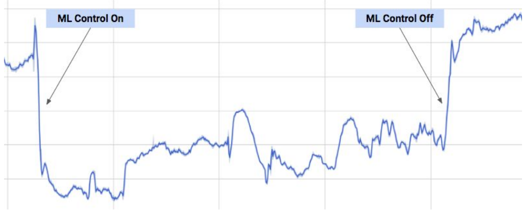
ML hardware is at its infancy. Even faster systems and wider deployment will lead to many more breakthroughs across a wide range of domains.

Learning in the core of all of our computer systems will make them better/more adaptive. There are many opportunities for this.









More info about our work at g.co/brain