MONGOOSE:

A Learnable LSH Framework for Efficient Neural Network Training

Beidi Chen, Zichang Liu, Binghui Peng, Zhaozhuo Xu, Jonathan Lingjie Li, Tri Dao, Zhao Song, Anshumali Shrivastava, Christopher Ré

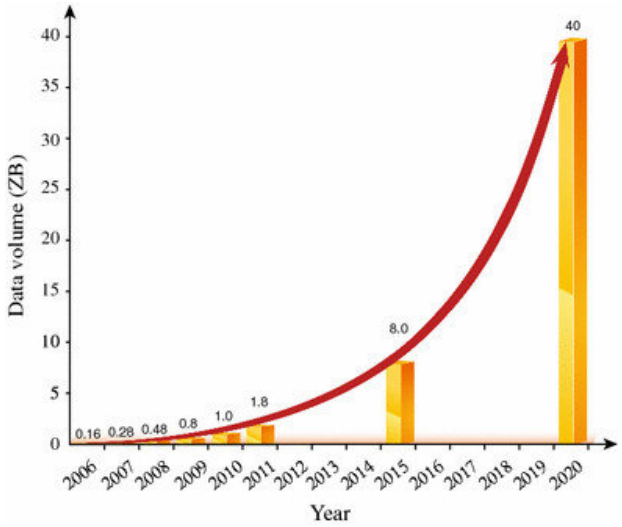
Stanford University RICE COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK PRINCETON UNIVERSITY

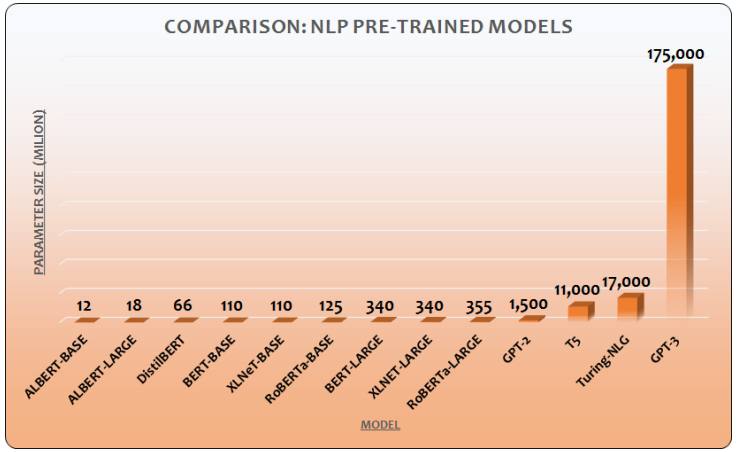
Neural Network (NN) Training Bottlenecks

Exponentially Growing Data Volume

Larger Model Size

More Resources





Training large-scale models imposes challenges on computational and memory resources.

middle plot by Kirk Borne on Twitter

One Approach: Locality Sensitive Hashing (LSH)

Bottlenecks we focus on: giant matrix multiplication in linear layers preceding a softmax

- e.g. attention layer, large output layer

Existing approaches: approx. matrix multiplication with LSH to trade accuracy for efficiency

- SLIDE (Chen et al. 2020)
- Reformer (Kitaev et al. 2020)

Challenges: LSH is inefficient because it introduces high overhead in practice

- not widely used for its bad accuracy-efficiency trade-off
- naively applying LSH → either 20x slow down or 10 point accuracy drop for NN training

Goal: we'll show how you can improve efficiency-accuracy tradeoffs for LSH to achieve up to 20x speed-up and 4x reduction in memory usage with no accuracy drop.

Outline

Part 1 Background

LSH and its overhead for approx. matrix multiplications in NN training
Data-dependent hashing can’t reduce the overhead

Part 2 Observation

Opportunity for overhead reduction:
NN weights and their LSH hash codes change slowly

Part 3 MONGOOSE

A scheduler for LSH updates + Learnable LSH hash functions

Part 4 Application

Extreme classification and language modeling tasks

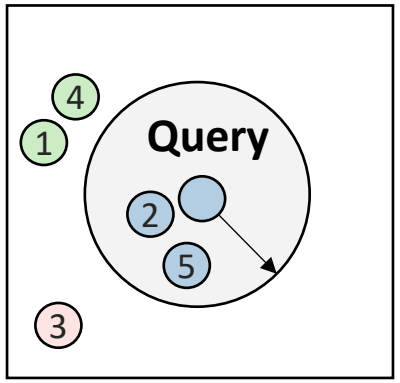
Locality Sensitive Hashing for Near-Neighbor Search

Hashing: Function **(Randomized)** h that maps a given data point ($x \in R^D$) to an integer key $h : R^D \mapsto \{0, 1, 2, \dots, N\}$. $h(x)$ serves as a discrete fingerprint.

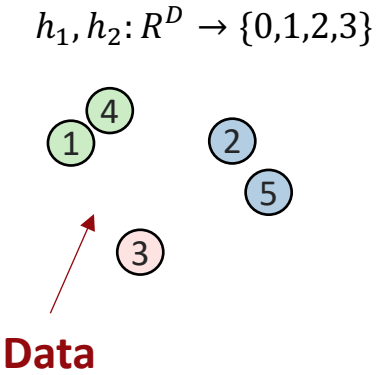
Locality Sensitive Property:

- If $Sim(x, y)$ is high, then $\Pr(h(x) = h(y))$ is high
- If $Sim(x, y)$ is low, then $\Pr(h(x) = h(y))$ is low

Near-neighbor Search

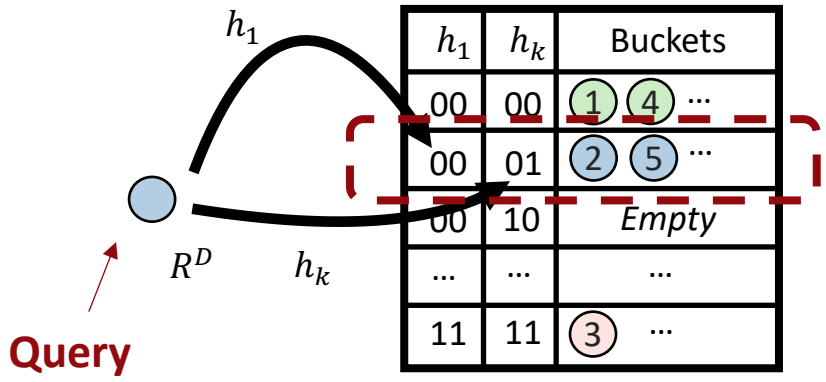


1 Preprocessing Step



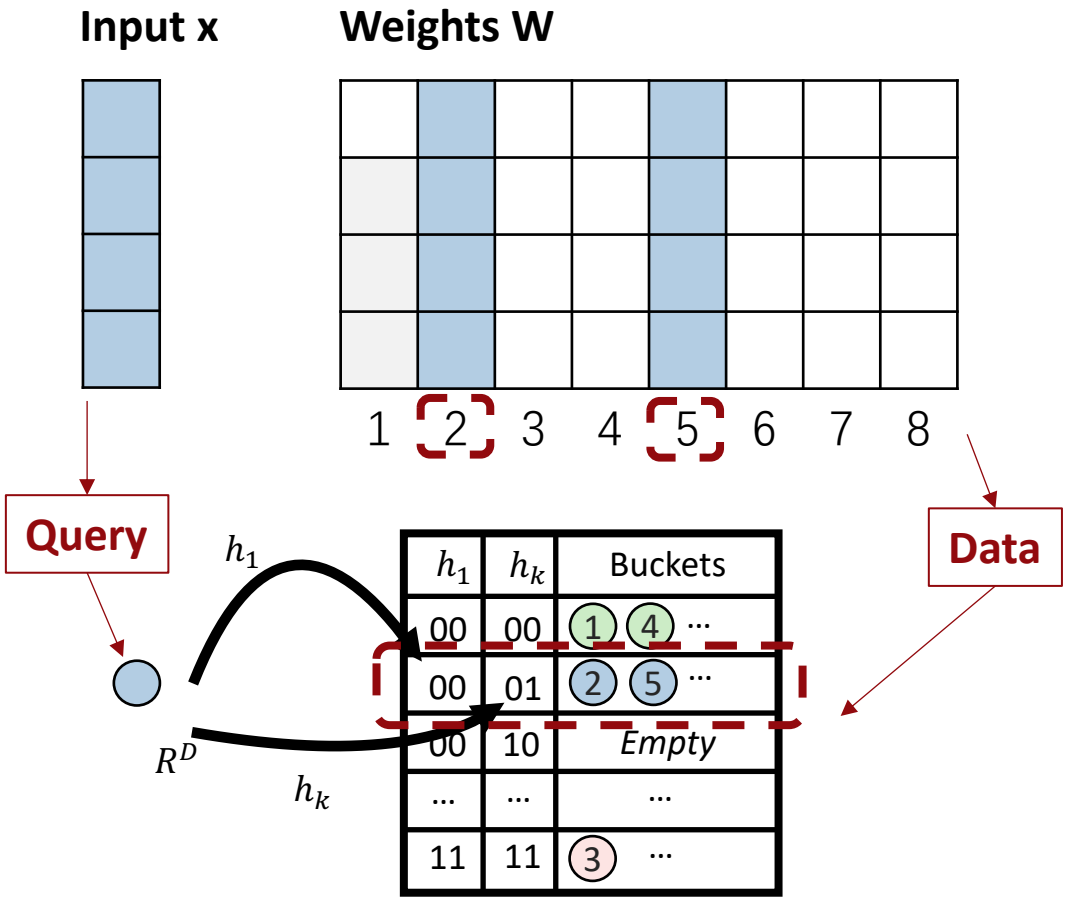
h_1	h_k	Buckets
00	00	1 4 ···
00	01	2 5 ···
00	10	*Empty*
···	···	···
11	11	3 ···

2 Query Step

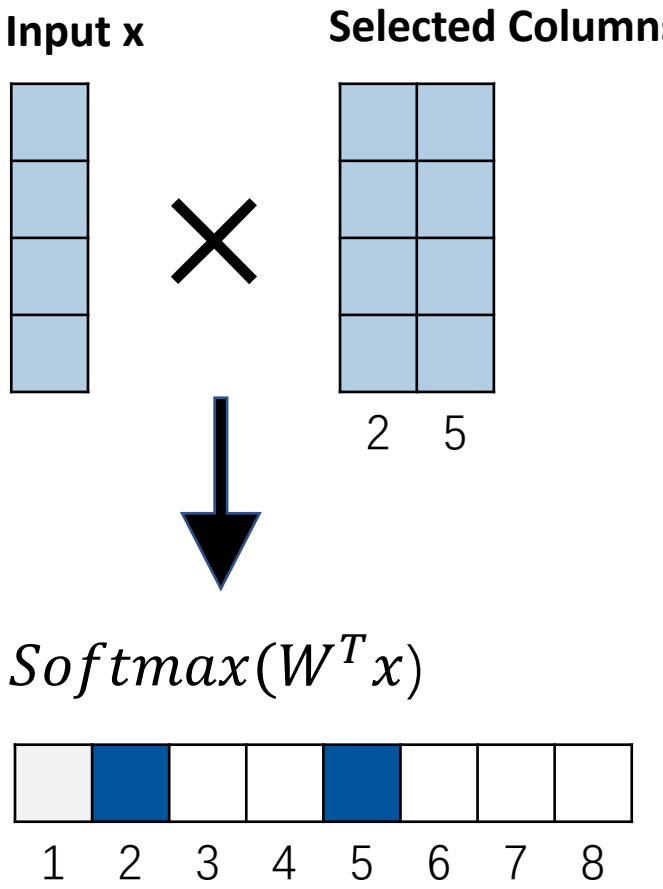


LSH for Approx. Matrix Multiplication

1 Near Neighbor Search

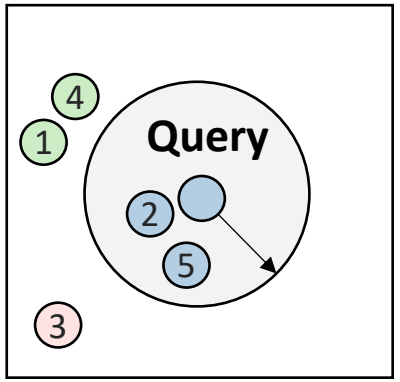


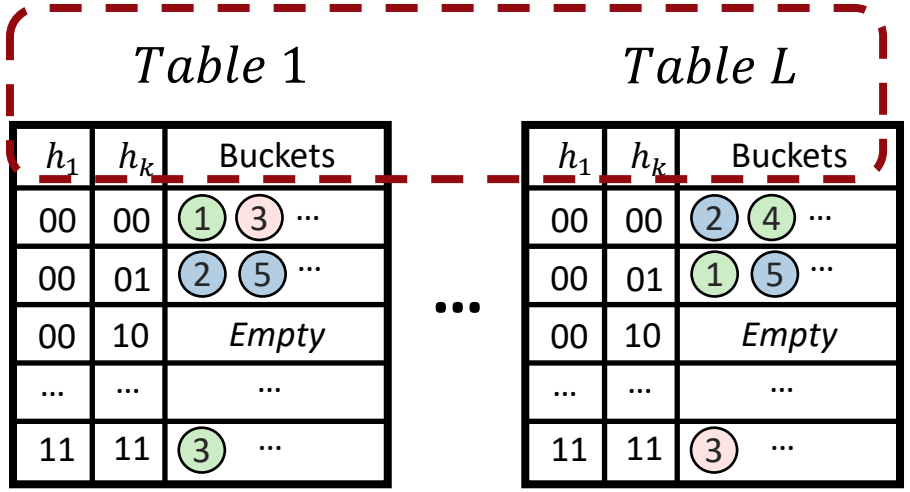
2 Approx. Matrix Multiplication



Practical Issues of LSH in NNS: Query Overhead

Space Partitions

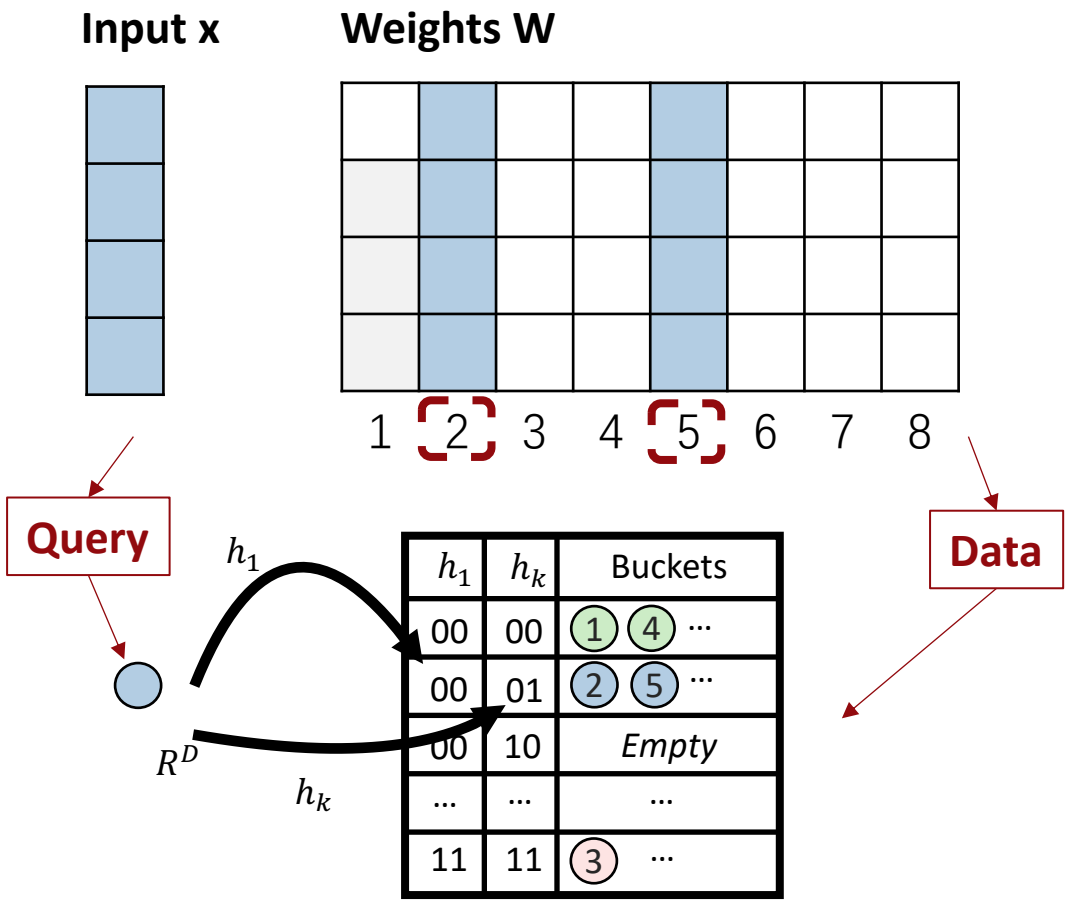




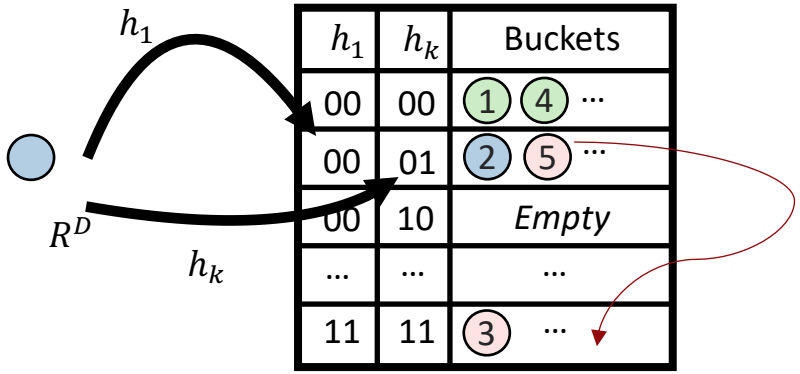
- LSH is data independent → random space partitions independent of data
- Needs lots of hash functions, tables and distance computations for good near-neighbor quality
- Buckets can be quite heavy, due to poor randomness or unfavorable data distributions

Is data-dependent hashing a straightforward solution? → No!

Issues in NN Training: Update Overhead



- Parameters are evolving during training
- LSH needs to update accordingly
- Otherwise large drop in accuracy



Even worse … data-dependent hashing increases update overhead.

Challenges and Goals

Challenges

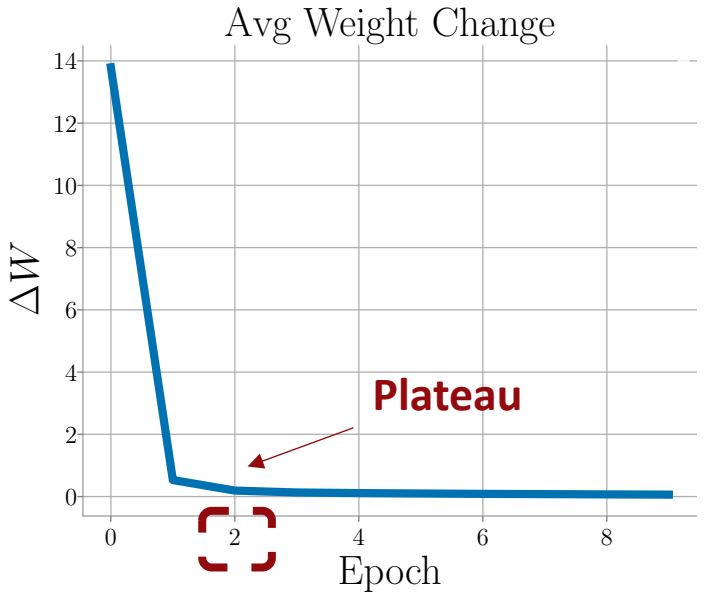
- LSH has been mainly studied in a static data setting; less understanding in NN training
- LSH has poor accuracy-efficiency trade-off (query overhead) when approximating matrix multiplication
- LSH has high update overhead when model parameters in NN evolve

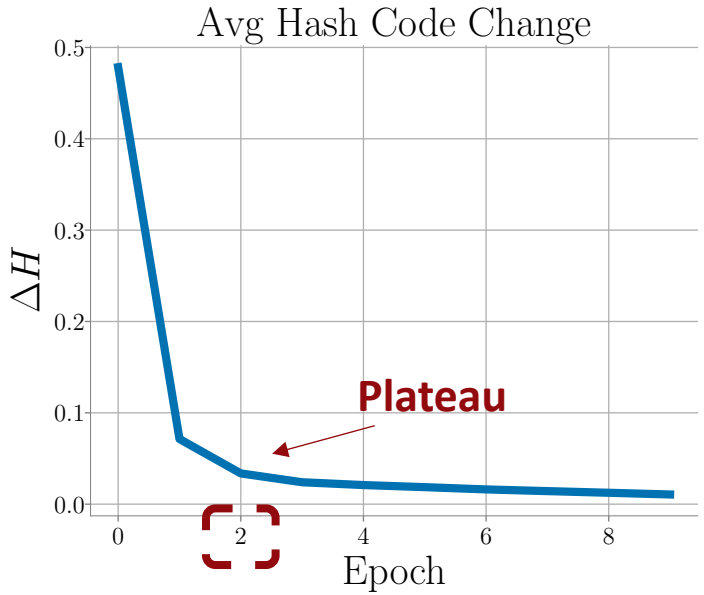
Goals

- Have deeper understanding of parameter update dynamics in NN training
- Have faster query time while maintaining high accuracy for approximate matrix multiplication
- Perform low-cost updates to account for evolving parameters

Approach: MONGOOSE, A Learnable LSH Framework for Efficient NN Training

Observation: NN Weights Change Slowly





- Changes are big at the beginning and plateau early
- On avg, only 1-5% of the hash codes change every epoch (e.g. MLP, Transformers)

Main Insight: Most weights don't change enough to trigger LSH updates

Benefits and Issues

Benefits

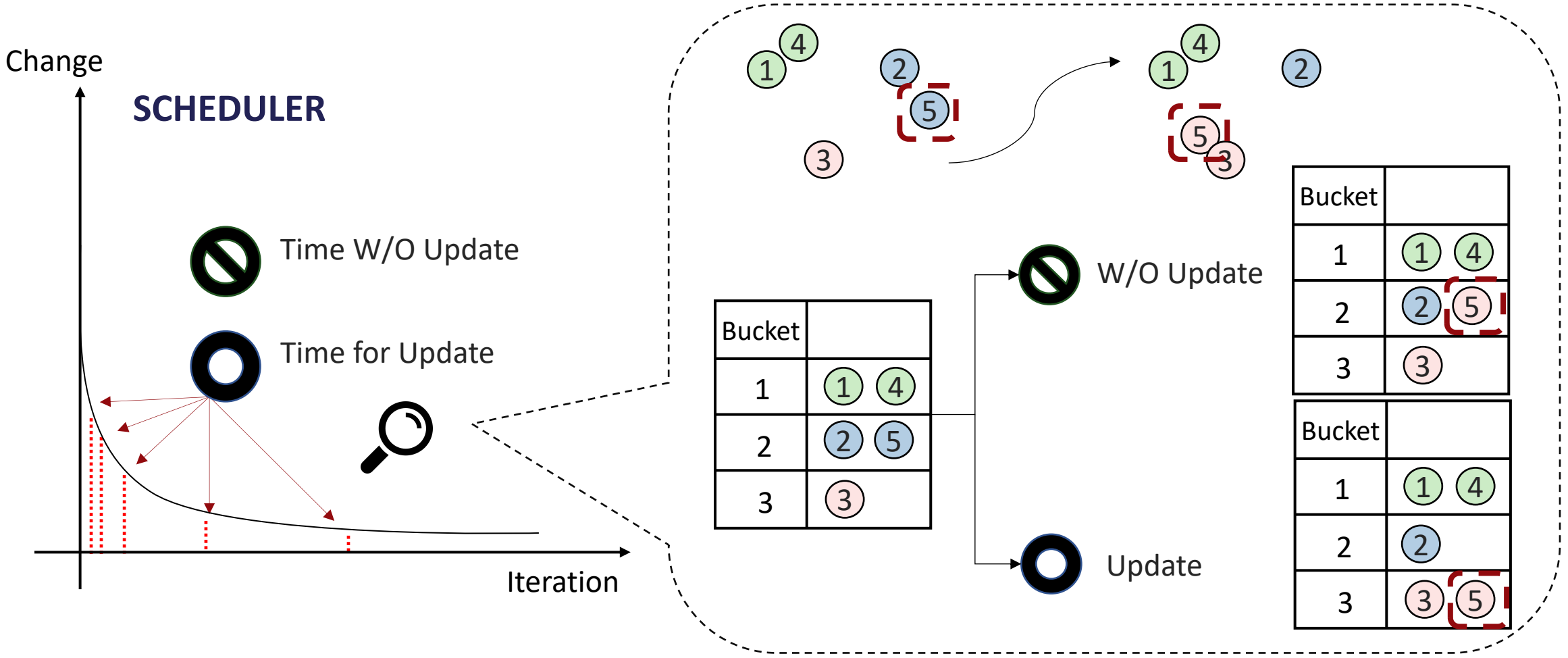
- With update oracle: 100× reduction on LSH update frequency
- Lower update overhead: an opportunity to use data-dependent hash functions to achieve faster query time

Issues

- How to schedule the updates without oracle or computing the hash codes?
- How to learn data-dependent hash functions without compromising on the update time?

An algorithm for scheduling efficient LSH updates + learning parameterized LSH hash functions

Scheduler: Update or No Update



Intuition: Maintain low-cost structure to automatically decide update or not.

Guarantees

Condition 1: NN weights change slowly over time.

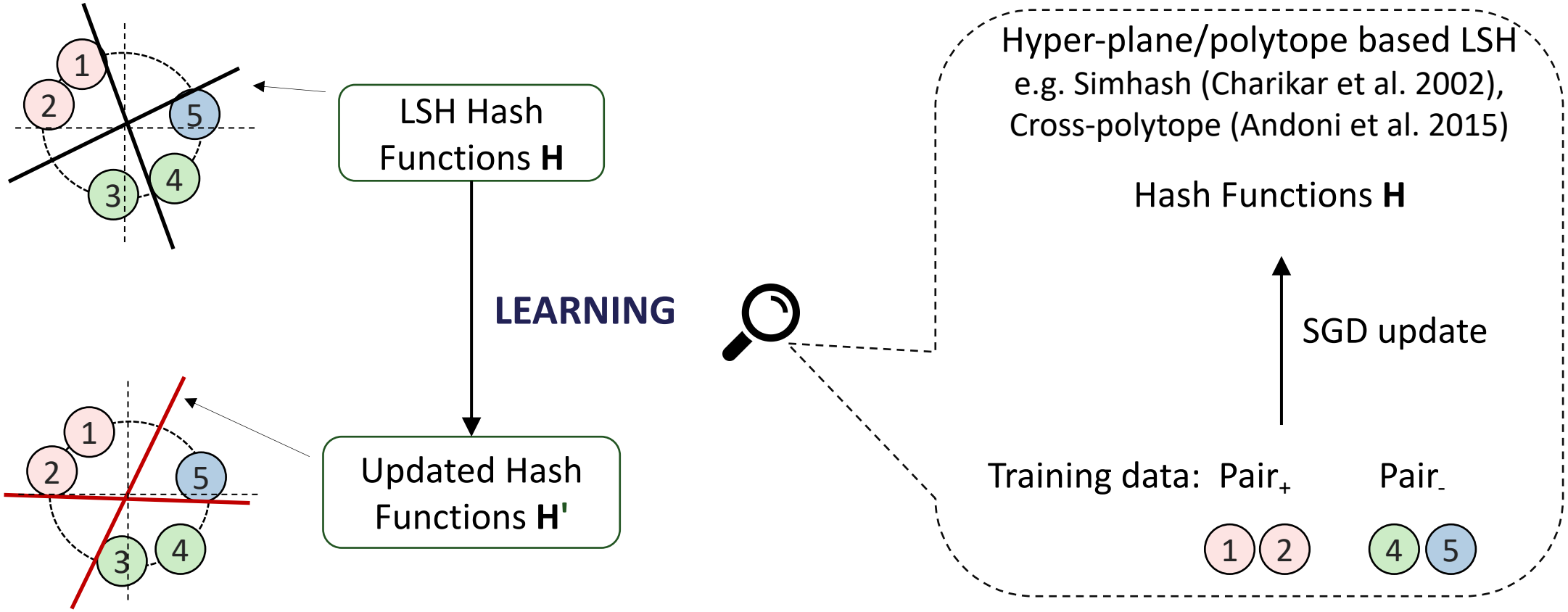
Key Observation

Condition 2: The avg time to update N data points at once is faster than update them sequentially.

Parallelism

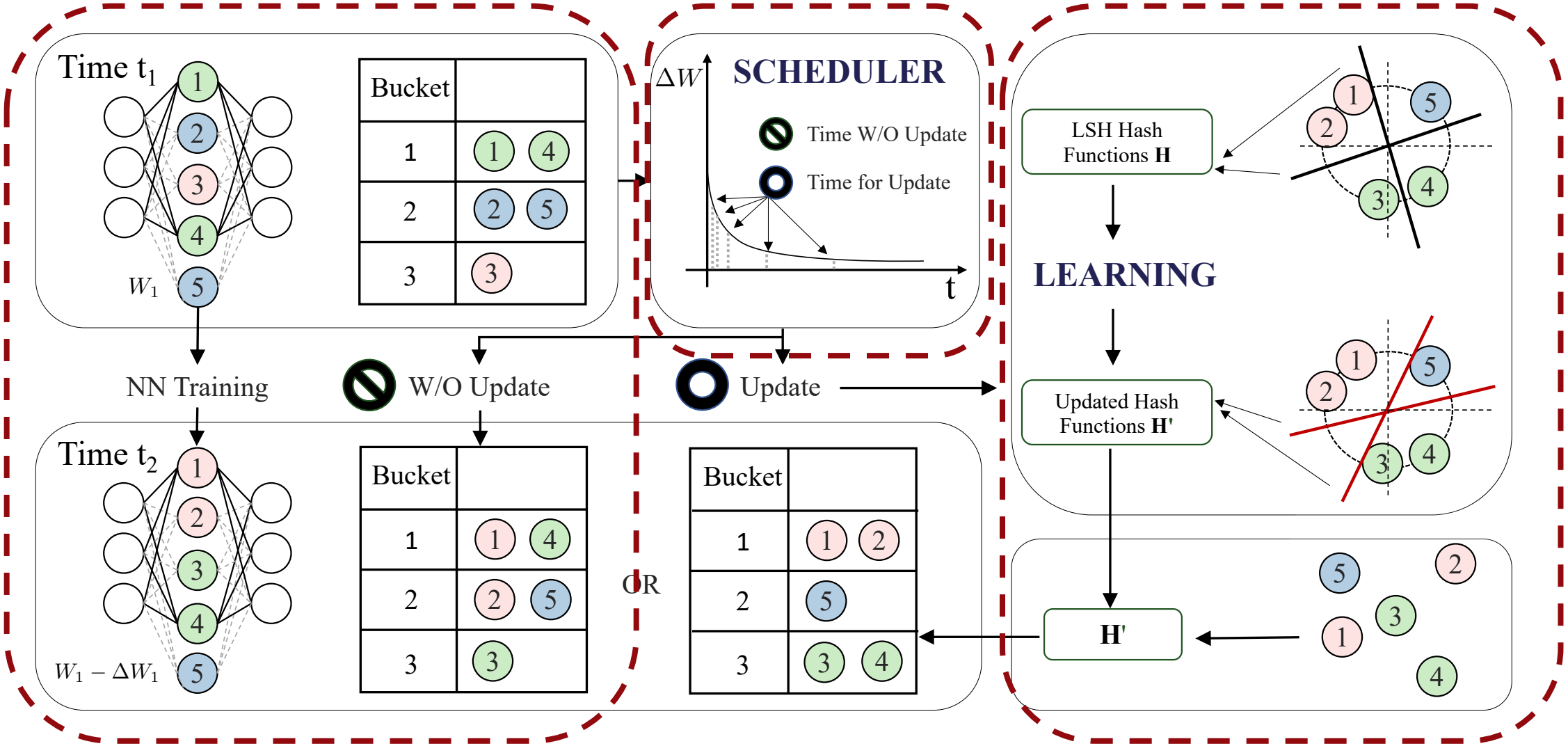
We theoretically show that our scheduler is always faster (amortized) than the naïve sequential updating strategy while not hurting the accuracy.

Learnable LSH (Low-cost)



Intuition: Perform low-cost supervised learning on LSH hash functions.

MONGOOSE Workflow



Task 1: Extreme Classification

Dataset: Wiki-325K
Baseline: SLIDE (Chen et al. 2020) → Vanilla LSH
(All the numbers higher the better)

Model	Precision@1	Time (reduction)	Memory (reduction)
MLP	0.501	1	1
MLP w/ SLIDE	0.438	1.4 x	1.7 x
MLP w/ MONGOOSE	**0.519**	**20 x**	**4 x**

MONGOOSE is more time & memory efficient, more accurate than baselines.

Task 2: Language Modeling

Dataset: Copy and Enwik8 LM
Baseline: Reformer (Kitaev et al. 2020) → Vanilla LSH
(All the numbers lower the better)

Model	Copy Task (Loss)	Enwik8 (Perplexity)
Reformer	1.25	2.65
MONGOOSE	**0.75**	**2.59**

MONGOOSE is more accurate than Reformer under the same time/memory constraint.

Conclusion and Future Directions

Conclusion

Observe slowly changing NN training patterns → allow for a data-dependent LSH framework

Design a LSH update scheduler to cope with parameter changes → reduce LSH update overhead

Adapt LSH hash functions to data → reduce LSH query overhead

MONGOOSE achieves better accuracy-efficiency trade-off of LSH for more efficient NN training

Future Directions

Are there more efficient data structures for Near Neighbor Search in NN training?

Are there new NN models that can exploit LSH/NNS in approx. large matrix multiplication?

*A **full** meal makes a **slow mongoose**, and if wanted all his strength and **quickness** ready, he must keep himself **thin**.*

— Rikki-tikki-tavi from The Jungle Books

THANKS

Poster #3 May 3 5-7pm PDT

Contact: beidic@stanford.edu

Paper URL