# Monolith: Real Time Recommendation System With Collisionless Embedding Table

Zhuoran Liu et al., ByteDance Inc.

*arXiv preprint.*

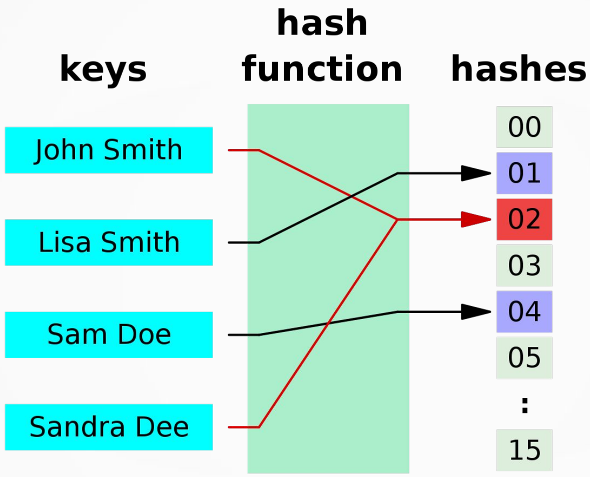
# Introduction

- An engineering-heavy paper from ByteDance, creator of TikTok
- Powers the Infamous(?) recommendation algorithms of TikTok
- Architecture is **currently being used live** in BytePlus Recommend

Key Contributions

- Unveils industrial details
- Open source release
- Decision making procedures through experiments in industrial settings

# Preliminaries : Hashing



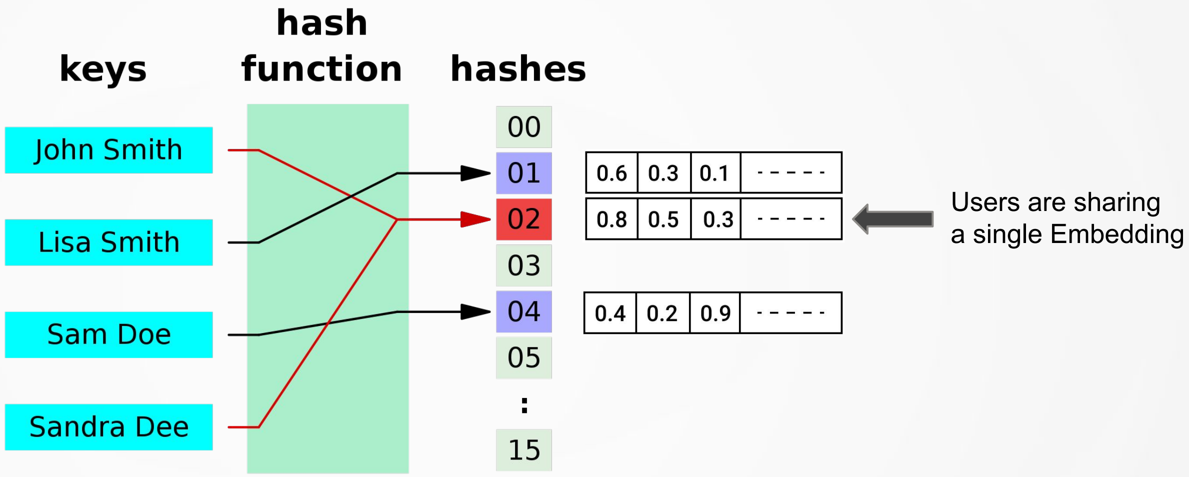

An example hash function:

$$h(x) = x \bmod 15$$

- A **hash function** maps input values of infinite range to finite-length **buckets.**
- Therefore sometimes more than one value is stored in a single bucket.
- This is **hash collision.**

[1] Image source : https://en.wikipedia.org/wiki/Hash_function

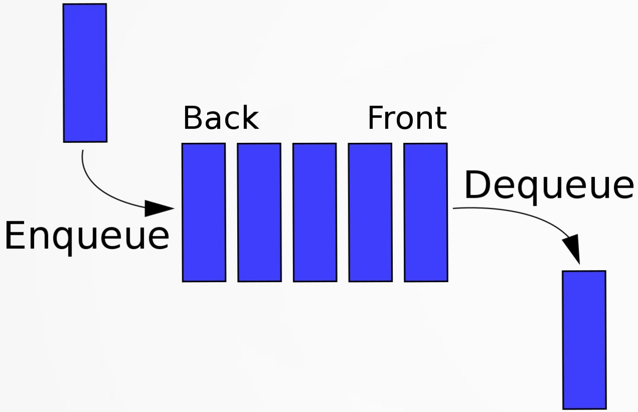
# Preliminaries : Hashing in RecSys

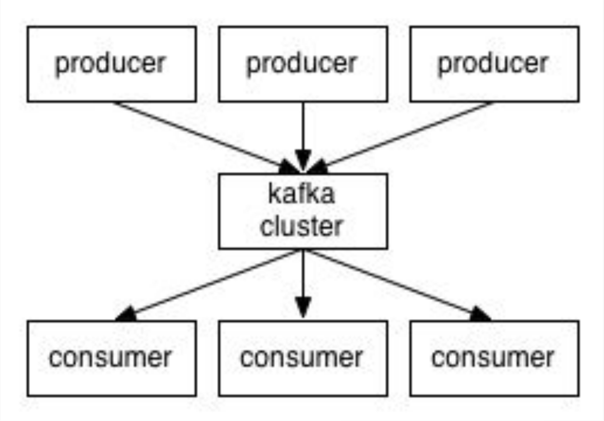



- The number of user/item embeddings are theoretically infinite
- Hash tables automatically expand via **rehashing** according to **load factor**
- … At the cost of slight data corruption.

[1] Image source : https://en.wikipedia.org/wiki/Hash_function

# Preliminaries : Apache Kafka & Flink




- A queue is a FIFO(First-In-First-Out) data structure
- **Apache Kafka** enables multiple users to share a single queue deployed online!
- **Apache Flink** enables complex tasks to be performed during the pipeline!

[1] Image source : https://en.wikipedia.org/wiki/Queue_(abstract_data_type)
[2] Image source : https://kafka.apache.org/08/documentation.html#introduction

# Preliminaries : FM & DeepFM

| | Feature vector x | | | | | | | | | | | | | | | | | | | | | Target y | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | User | | | | Movie | | | | | Other Movies rated | | | | | Time | Last Movie rated | | | | | | |
| | A | B | C | ... | TI | NH | SW | ST | ... | TI | NH | SW | ST | ... | Time | TI | NH | SW | ST | ... | | |
| $x^{(1)}$ | 1 | 0 | 0 | ... | 1 | 0 | 0 | 0 | ... | 0.3 | 0.3 | 0.3 | 0 | ... | 13 | 0 | 0 | 0 | 0 | ... | 5 | $y^{(1)}$ |
| $x^{(2)}$ | 1 | 0 | 0 | ... | 0 | 1 | 0 | 0 | ... | 0.3 | 0.3 | 0.3 | 0 | ... | 14 | 1 | 0 | 0 | 0 | ... | 3 | $y^{(2)}$ |
| $x^{(3)}$ | 1 | 0 | 0 | ... | 0 | 0 | 1 | 0 | ... | 0.3 | 0.3 | 0.3 | 0 | ... | 16 | 0 | 1 | 0 | 0 | ... | 1 | $y^{(2)}$ |
| $x^{(4)}$ | 0 | 1 | 0 | ... | 0 | 0 | 1 | 0 | ... | 0 | 0 | 0.5 | 0.5 | ... | 5 | 0 | 0 | 0 | 0 | ... | 4 | $y^{(3)}$ |
| $x^{(5)}$ | 0 | 1 | 0 | ... | 0 | 0 | 0 | 1 | ... | 0 | 0 | 0.5 | 0.5 | ... | 8 | 0 | 0 | 1 | 0 | ... | 5 | $y^{(4)}$ |
| $x^{(6)}$ | 0 | 0 | 1 | ... | 1 | 0 | 0 | 0 | ... | 0.5 | 0 | 0.5 | 0 | ... | 9 | 0 | 0 | 0 | 0 | ... | 1 | $y^{(5)}$ |
| $x^{(7)}$ | 0 | 0 | 1 | ... | 0 | 0 | 1 | 0 | ... | 0.5 | 0 | 0.5 | 0 | ... | 12 | 1 | 0 | 0 | 0 | ... | 5 | $y^{(6)}$ |

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j$$

- Generalized frameworks for regression, classification, and ranking tasks
- Takes into account multi-hop interactions between features
- Recommendation tasks can be cast into 0-1 classification tasks via negative sampling

[1] Image and equation source : Rendle, Steffen. “Factorization machines.” 2010 IEEE International conference on data mining. IEEE, 2010.

# The Monolith Architecture

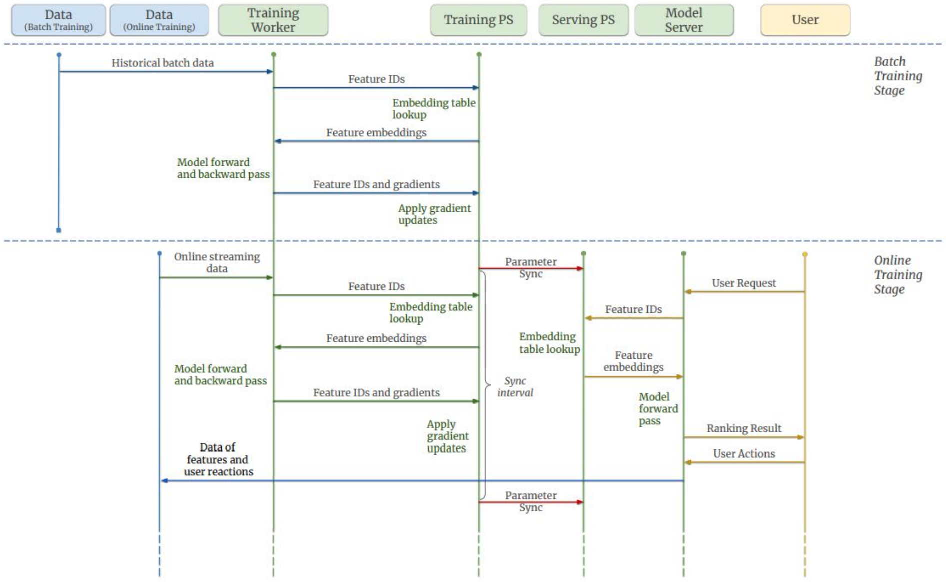


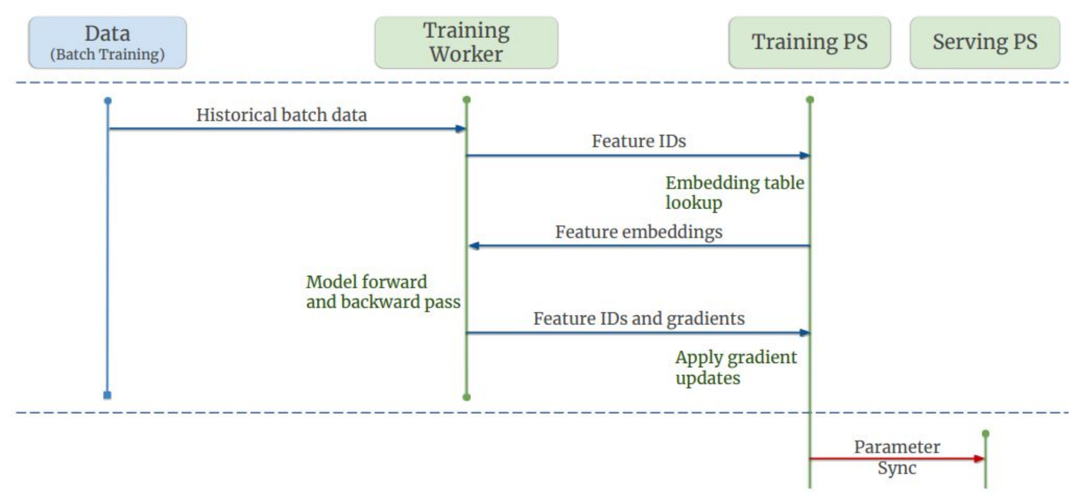

**Figure 1: Monolith Online Training Architecture.**

[1] Image source : Liu, Zhuoran, et al. "Monolith: Real Time Recommendation System With Collisionless Embedding Table." arXiv preprint arXiv:2209.07663 (2022).

# The Monolith Architecture

- **Training Parameter Server**
  - Distributed computing is a must for large-scale machine learning tasks
  - Multiple workers each train with small batches of data and *needs simultaneous access to parameters*
  - In a distributed ML task, each machine is either a parameter server or a worker node
- **Serving Parameter Server**
  - Holds parameters for the final recommendation model for users
  - Parameters must be periodically synchronized with the training PS
- **Training Worker**
  - Runs forward passes, calculates gradients
  - Sends back gradients to PS for parameter updates
- **Model Server**
  - Inference worker for the final recommendation model for users

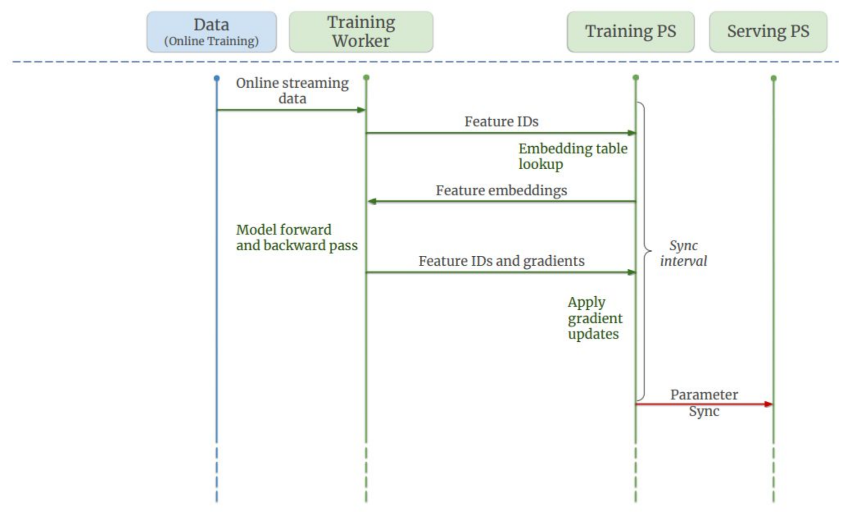
# Batch Training with Monolith




- Training data is stored in HDFS
- Trains like your plain old everyday ML task, but for **only one epoch**
- Sends parameters to Serving PS when training is complete

[1] Image source : Liu, Zhuoran, et al. "Monolith: Real Time Recommendation System With Collisionless Embedding Table." arXiv preprint arXiv:2209.07663 (2022).

# Online Training with Monolith




- Same procedure as batch training, but uses **stream data** from **live user interactions**
- **Apache Kafka** is used for streaming interaction data
- Sends parameters to Serving PS once every few iterations

[1] Image source : Liu, Zhuoran, et al. "Monolith: Real Time Recommendation System With Collisionless Embedding Table." arXiv preprint arXiv:2209.07663 (2022).

# Online Training with Monolith

```
1:
1488844,3,2005-09-06
822109,5,2005-05-13
885013,4,2005-10-19
30878,4,2005-12-26
823519,3,2004-05-03
893988,3,2005-11-17
124105,4,2004-08-05
1248029,3,2004-04-22
1842128,4,2004-05-09
```

+

| | | | |
|---|---|---|---|
| 0.6 | 0.3 | 0.1 | - - - - - |
| 0.8 | 0.5 | 0.3 | - - - - - |
| 0.6 | 0.3 | 0.1 | - - - - - |
| 0.4 | 0.2 | 0.9 | - - - - - |

=

Feature vector **x** / Target y

| | User: A | B | C | ... | Movie: TI | NH | SW | ST | ... | Other Movies rated: TI | NH | SW | ST | ... | Time | Last Movie rated: TI | NH | SW | ST | ... | Target y | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x^{(1)}$ | 1 | 0 | 0 | ... | 1 | 0 | 0 | 0 | ... | 0.3 | 0.3 | 0.3 | 0 | ... | 13 | 0 | 0 | 0 | 0 | ... | 5 | $y^{(1)}$ |
| $x^{(2)}$ | 1 | 0 | 0 | ... | 0 | 1 | 0 | 0 | ... | 0.3 | 0.3 | 0.3 | 0 | ... | 14 | 1 | 0 | 0 | 0 | ... | 3 | $y^{(2)}$ |
| $x^{(3)}$ | 1 | 0 | 0 | ... | 0 | 0 | 1 | 0 | ... | 0.3 | 0.3 | 0.3 | 0 | ... | 16 | 0 | 1 | 0 | 0 | ... | 1 | $y^{(2)}$ |
| $x^{(4)}$ | 0 | 1 | 0 | ... | 0 | 0 | 1 | 0 | ... | 0 | 0 | 0.5 | 0.5 | ... | 5 | 0 | 0 | 0 | 0 | ... | 4 | $y^{(3)}$ |
| $x^{(5)}$ | 0 | 1 | 0 | ... | 0 | 0 | 0 | 1 | ... | 0 | 0 | 0.5 | 0.5 | ... | 8 | 0 | 0 | 1 | 0 | ... | 5 | $y^{(4)}$ |
| $x^{(6)}$ | 0 | 0 | 1 | ... | 1 | 0 | 0 | 0 | ... | 0.5 | 0 | 0.5 | 0 | ... | 9 | 0 | 0 | 0 | 0 | ... | 1 | $y^{(5)}$ |
| $x^{(7)}$ | 0 | 0 | 1 | ... | 0 | 0 | 1 | 0 | ... | 0.5 | 0 | 0.5 | 0 | ... | 12 | 1 | 0 | 0 | 0 | ... | 5 | $y^{(6)}$ |

- Two Kafka queues are used: Log Kafka & Feature Kafka
- **Online Joiner** consumes these queues to generate training examples
- Negative sampling is conducted during the process

[1] Data snapshot from : https://grouplens.org/datasets/movielens/
[1] Image and equation source : Rendle, Steffen. “Factorization machines.” 2010 IEEE International conference on data mining. IEEE, 2010.

# Online Training with Monolith

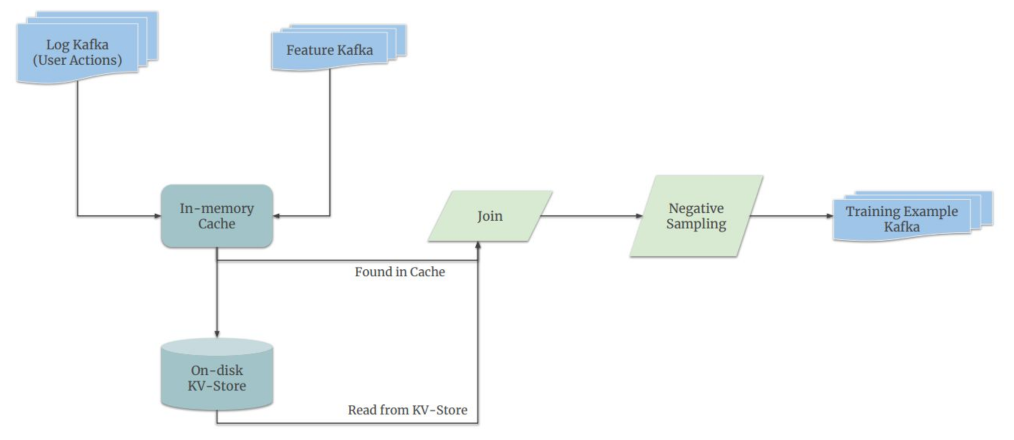



**Figure 5: Online Joiner.**

- Input two streams -> Outputs one stream
- **Apache Flink** is used for complex joining and negative sampling procedures
- **Caching** is always nice for memory management

[1] Image source : Liu, Zhuoran, et al. "Monolith: Real Time Recommendation System With Collisionless Embedding Table." arXiv preprint arXiv:2209.07663 (2022).

# Online Training with Monolith

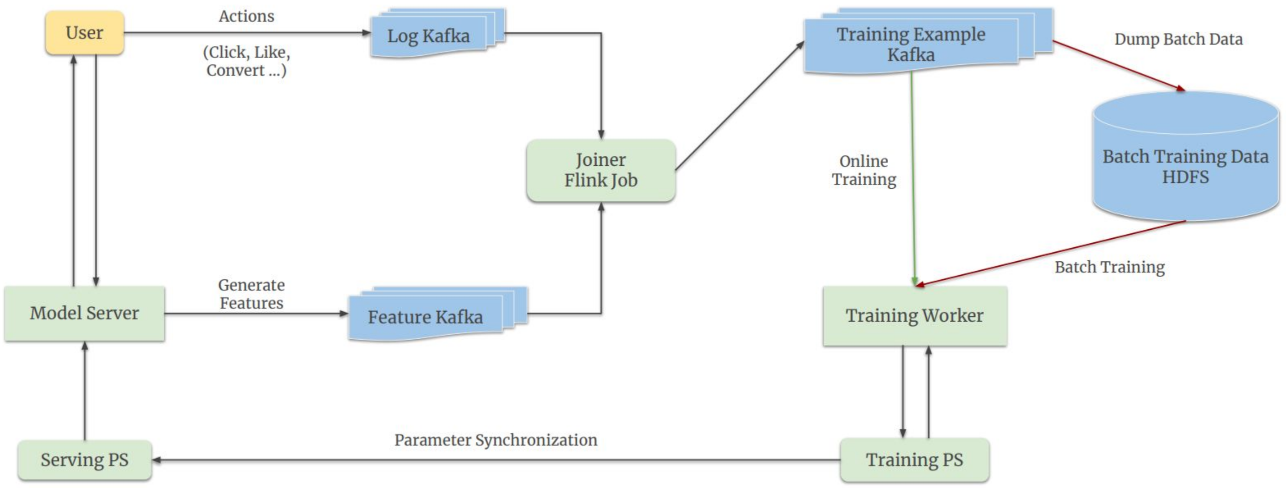
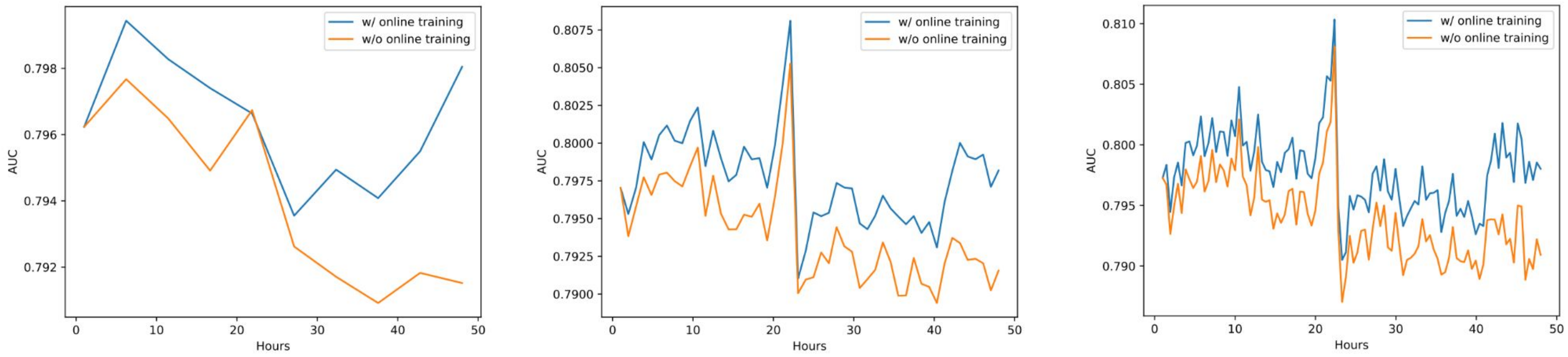



**Figure 4: Streaming Engine.**
*The information feedback loop from [User → Model Server → Training Worker → Model Server → User] would spend a long time when taking the Batch Training path, while the Online Training will close the loop more instantly.*

[1] Image source : Liu, Zhuoran, et al. "Monolith: Real Time Recommendation System With Collisionless Embedding Table." arXiv preprint arXiv:2209.07663 (2022).

# Experiments : Parameter Sync



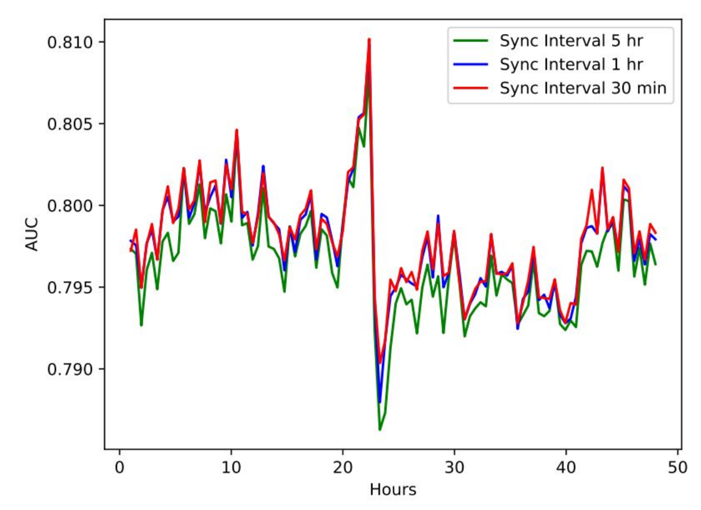

(a) Online training with 5 hrs sync interval (b) Online training with 1 hr sync interval (c) Online training with 30 min sync interval

**RQ1.** Does online learning really improve recommendation performance?

**A1.** Yes it does.

[1] Image source : Liu, Zhuoran, et al. "Monolith: Real Time Recommendation System With Collisionless Embedding Table." arXiv preprint arXiv:2209.07663 (2022).

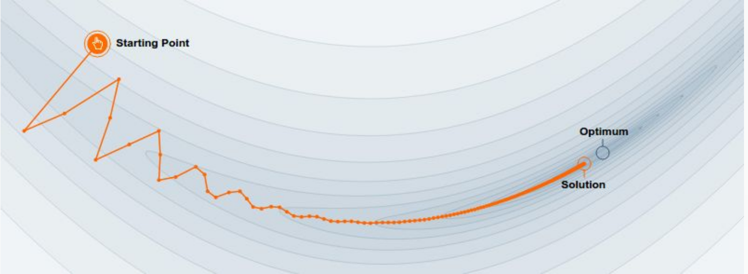
# Engineering Details : Parameter Sync




- According to experiment results, the more often you sync parameters the better.
- But parameter synchronization is not free when your model is **terabytes** large.
- We need to take the trade-off into account.

[1] Image source : Liu, Zhuoran, et al. "Monolith: Real Time Recommendation System With Collisionless Embedding Table." arXiv preprint arXiv:2209.07663 (2022).

# Engineering Details : Parameter Sync




- **Facts.**
  - Sparse embeddings are dominant among the model parameters.
  - During a short time window, only a small subset of sparse embeddings are updated.
  - Values of dense embeddings change much slower than sparse embeddings.
- **Derivations.**
  - Update costs of sparse embeddings are cheap.
    - Update as often as possible; up to once per minute
  - Dense embeddings don't need to be updated as often.
    - Once per day is enough, preferrably at midnight or dawn.

[1] Image source : https://www.luigifreda.com/2017/04/04/optimization-momentum-really-works/

# Experiments : Hash Collision

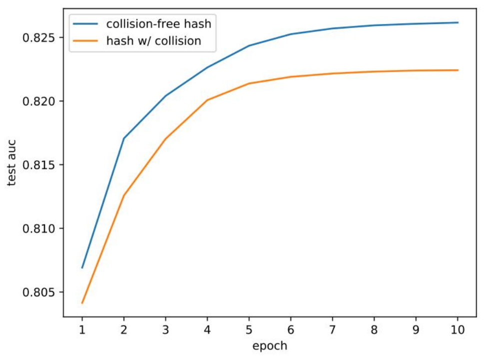



**Figure 7: Effect of Embedding Collision On DeepFM, MovieLens**

| | User IDs | Movie IDs |
|---|---|---|
| # Before Hashing | 162541 | 59047 |
| # After Hashing | 149970 | 57361 |
| Collision rate | 7.73% | 2.86% |

**RQ2.** Does hash collision degrade recommendation quality?

**A2.** Yes it does.

[1] Image source : Liu, Zhuoran, et al. "Monolith: Real Time Recommendation System With Collisionless Embedding Table." arXiv preprint arXiv:2209.07663 (2022).

# Engineering Details : Hash Collision

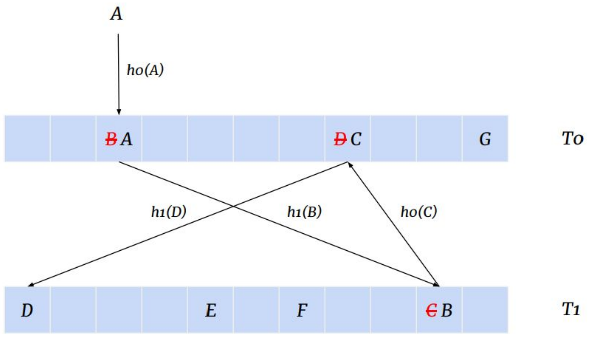



**Figure 3: Cuckoo HashMap.**

- **Cuckoo Hashmap** is one of the collision-free hashing techniques
- Uses two hashmaps and two different hash functions
- When data is inserted into a pre-occupied bucket, old data is kicked out to the other side

[1] Image source : Liu, Zhuoran, et al. "Monolith: Real Time Recommendation System With Collisionless Embedding Table." arXiv preprint arXiv:2209.07663 (2022).

# Engineering Details : Parameter Sync

- More memory is required for collision-free hashing

- **Facts.**
  - User/Item IDs are long-tail distributed.
  - Stale IDs don’t contribute to the recommendation quality as much.
    - Reasons: deleted accounts, user deleted the app, old trends, etc.

- **Derivations.**
  - Don’t remember every single ID.
    - Only remember IDs with high occurences;
    - After applying a probabilistic filter(e.g. 75% chance to remember)

# More Experiments : Parameter Sync

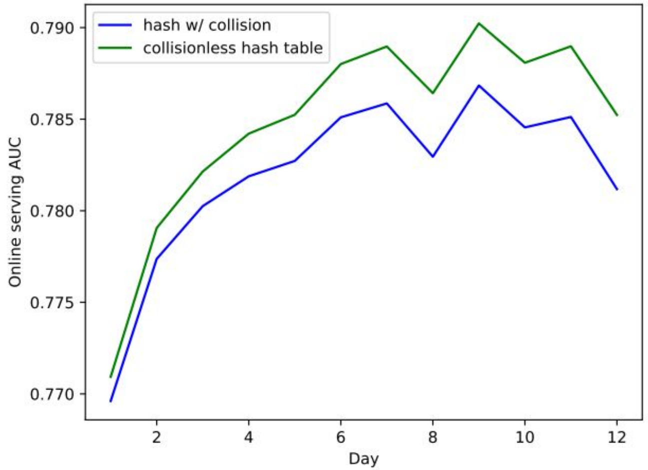



**Figure 8: Effect of Embedding Collision On A Recommendation Model In Production**

*We measure performance of this recommendation model by online serving AUC, which is fluctuating across different days due to concept-drift.*

[1] Image source : Liu, Zhuoran, et al. "Monolith: Real Time Recommendation System With Collisionless Embedding Table." arXiv preprint arXiv:2209.07663 (2022).

# Insights : Fault Tolerance

- Monolith automatically restarts upon failure
- Parameter updates are lost!
- Obvious solution is to take snapshots periodically
- …but even so, some part of the data is lost.

Also, taking snapshots of a model that weights multiple terabytes is EXPENSIVE.

**RQ3.** How often should we take snapshots?

# Insights : Fault Tolerance

*“Suppose a model’s parameters are sharded across 1000 PS, and they snapshot every day. Given 0.01% failure rate, one of them will go down every 10 days and we lose all updates on this PS for 1 day. Assuming a DAU of 15 Million and an even distribution of user IDs on each PS, we lose 1 day’s feedback from 15000 users every 10 days. This is acceptable…”*

[1] Quote from: Liu, Zhuoran, et al. "Monolith: Real Time Recommendation System With Collisionless Embedding Table." arXiv preprint arXiv:2209.07663 (2022).

# Conclusions

**Architectures**

- Distributed computing is a must for large-scale ML tasks.
- Use parameter servers!
- Separate workers/parameter servers for training/inference.

**Experiments**

- Online learning improves recommendation performance.
- Sync parameters as often as possible.
- Hash collision degrades recommendation performance.
- Don’t keep track of every single User/Item embeddings, just the important onces.
- It’s ok to take snapshots a lot less frequently than you’d think.

# Next Time?

**Nonuniform Negative Sampling and Log Odds Correction with Rare Events Data**

**HaiYing Wang**
Department of Statistics
University of Connecticut
haiying.wang@uconn.edu

**Aonan Zhang**
ByteDance Inc.
aonan.zhang@bytedance.com

**Chong Wang**
ByteDance Inc.
chong.wang@bytedance.com

# Discussions