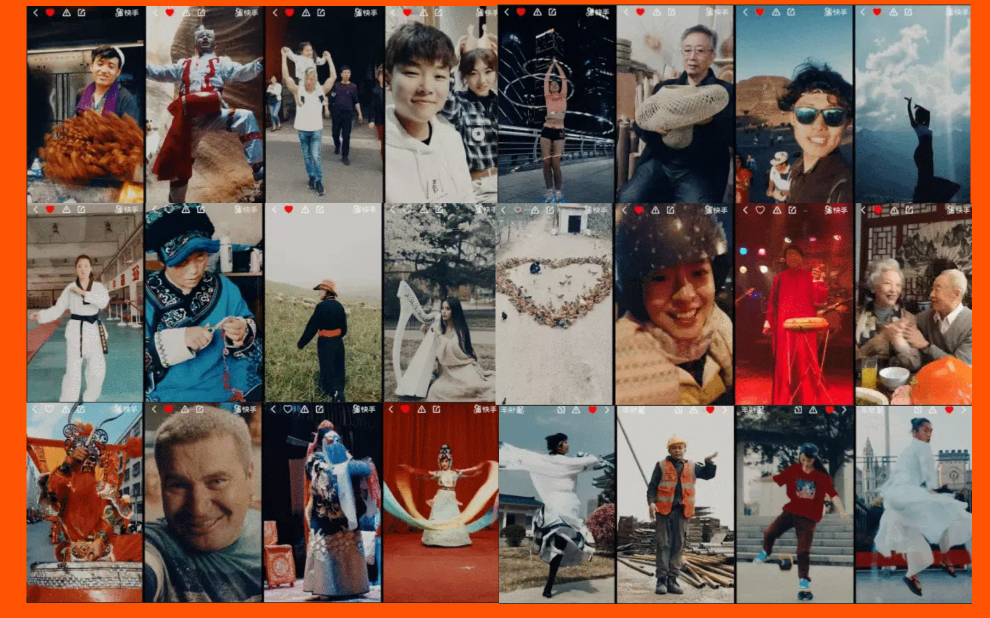
Memory-Efficient Continual Learning for Large-Scale Real-Time Recommendations

Minhui Xie*, Kai Ren*, Youyou Lu, Guangxu Yang, Qingxing Xu, Bihai Wu, Jiazhen Lin, Hongbo Ao, Wanhong Xu, and Jiwu Shu

Tsinghua University **Kuaishou Inc.**

Recommendation System in Kuaishou



Recommendations

Large-Scale Continual Learning Scenario

Large-Scale Learning

> 700 million users

Over **20 billion** videos in the warehouse

The backend model contains **tens of billions** of parameters.

Continual Learning & Real-Time Serving

10 million fresh UGC **per day**

2 million new training samples **per second**

Never end learning.

Typical DNN Model Architecture for Recommendation (I)

Continuous Features

Numeric columns

Age

videos watched

Categorical Features

Sparse lists of **ids** with extreme high dimensions

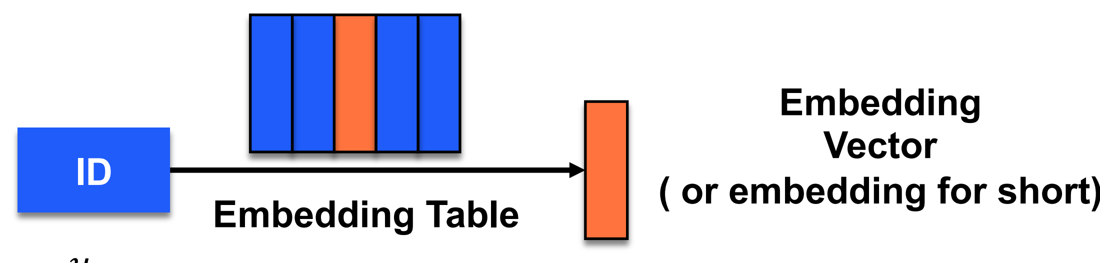
User ID $[u_2]$

Like Video IDs $[v_2, v_4]$

User Embedding Table → User Embedding Vector u_2

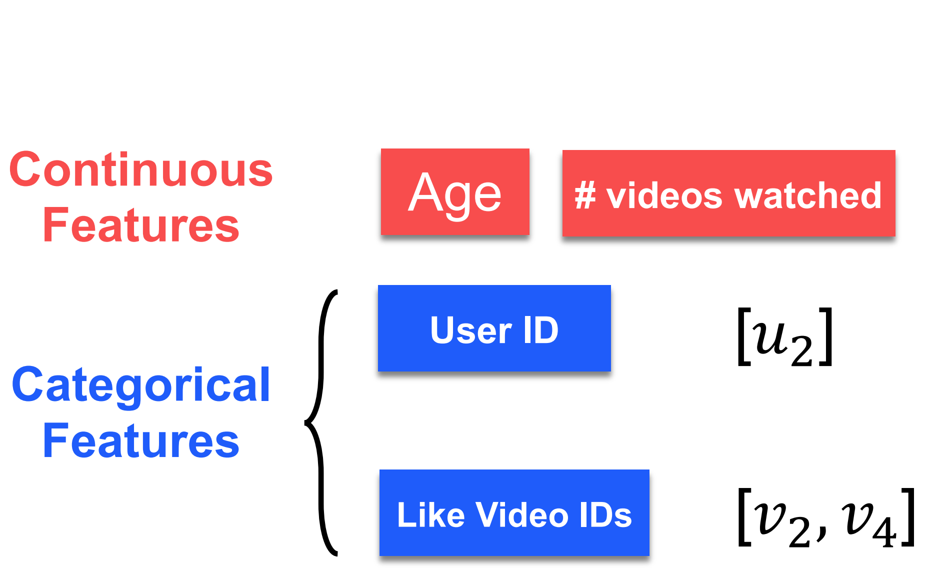
Video Embedding Table → Like Video Embedding Vector v_2 v_4

Embedding Lookup



Embedding Vector
(or embedding for short)

Typical DNN Model Architecture for Recommendation (II)



Labe

Fully-C cted

Fully-C cted

Pooling

Pooling

Fully-Connected

Fully-Connected

Embedding Tables

User Embedding T

Video Embedding Tb

$[u_2]$

$[v_2, v_4]$

Age

videos watched

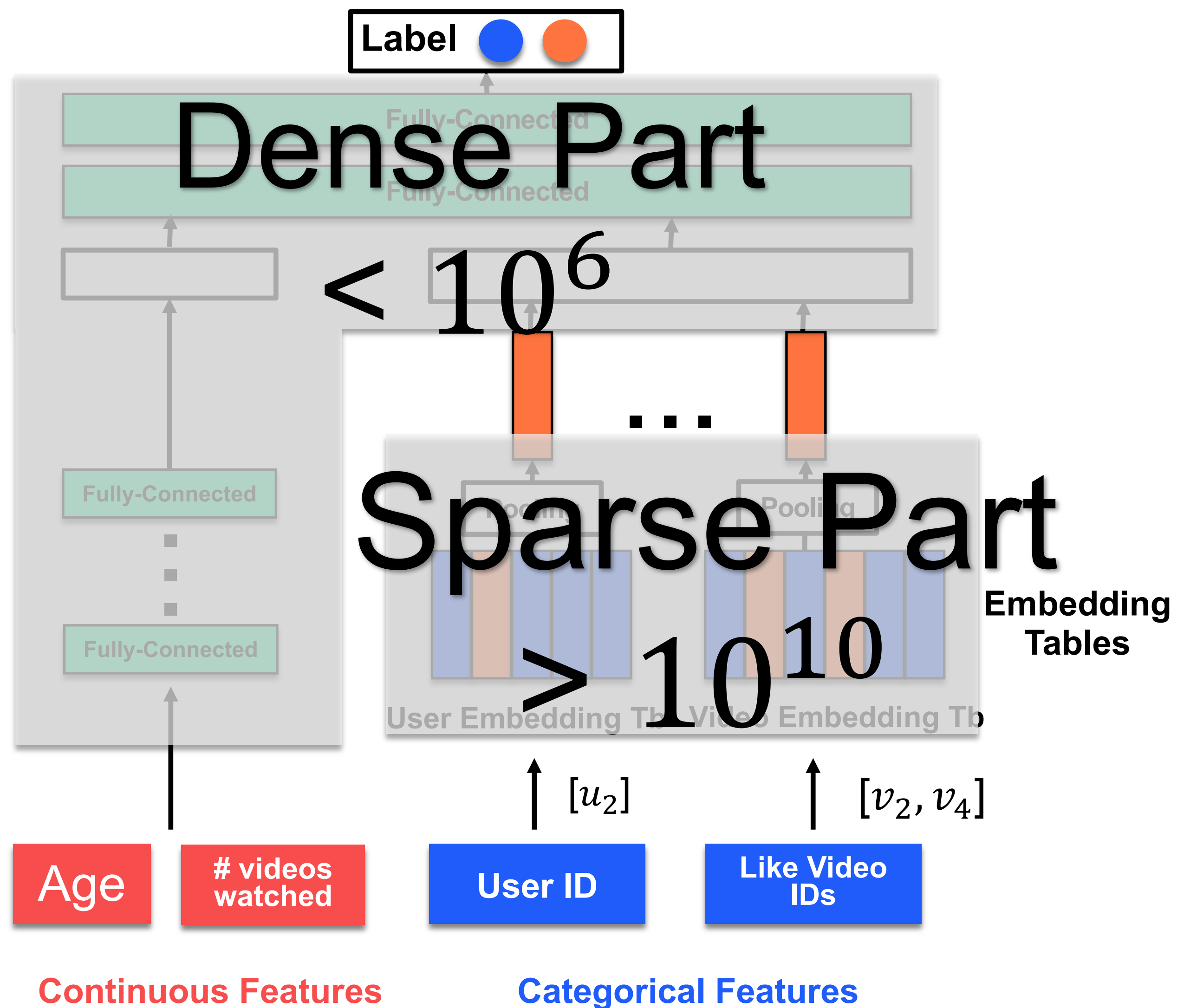
User ID

Like Video IDs

Continuous Features

Categorical Features

Our Models



Hash trick & Hash collision (I)

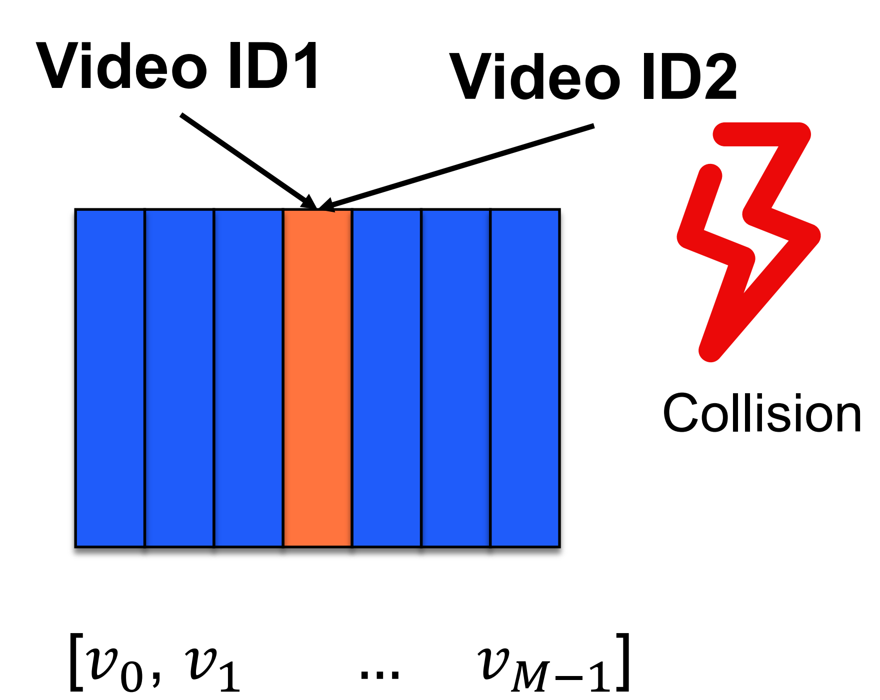
ID space >> embedding tb size

Hash trick

Hash(id) % M

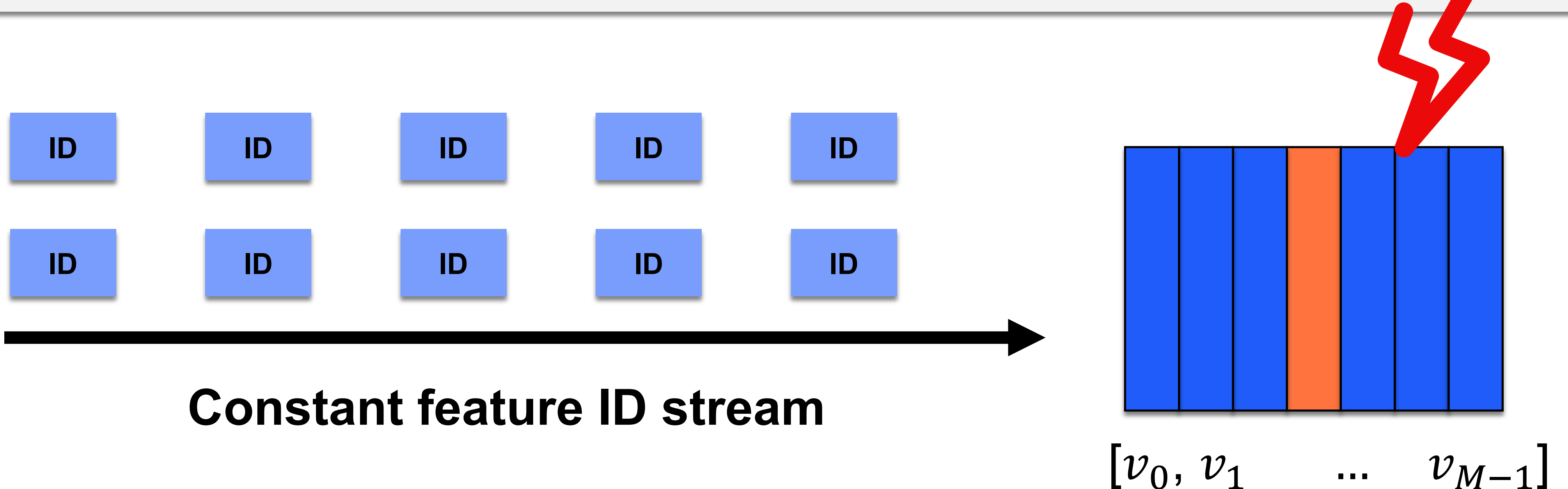
$[v_0, v_1 \ \dots \ v_{M-1}]$

Hash collision



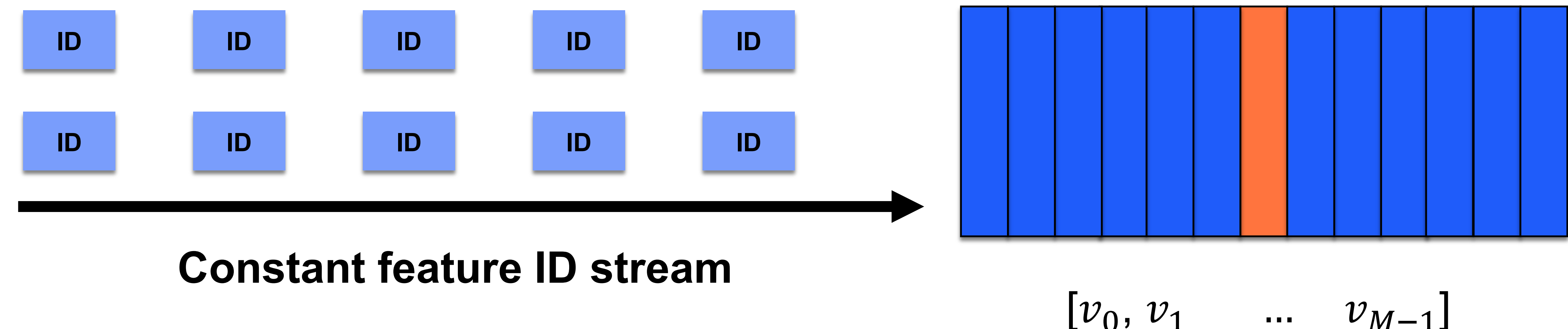
Hash(**VID1**) = Hash(**VID2**) mod M

Hash trick & Hash collision (II)

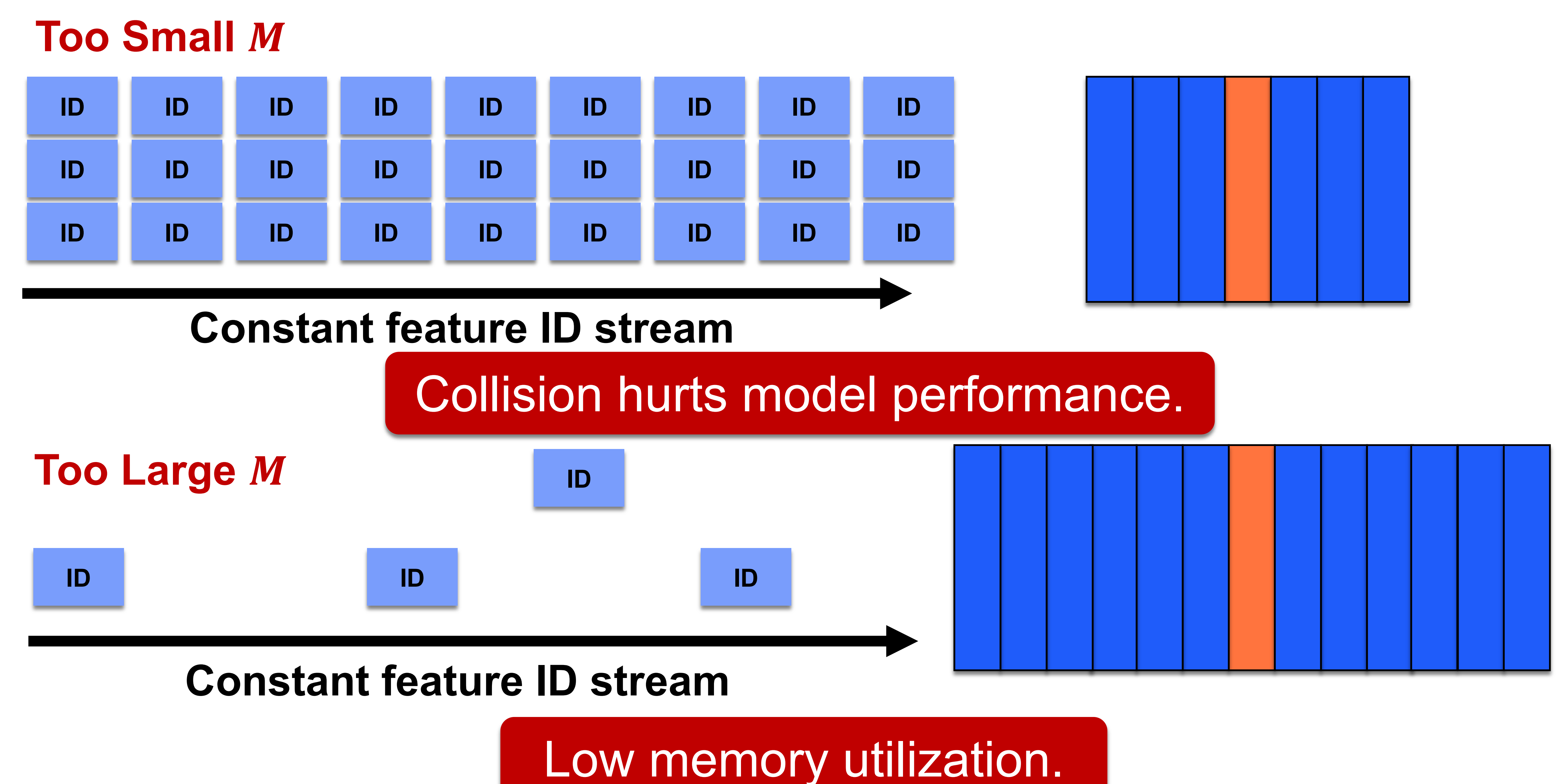


Hash trick & Hash collision (III)

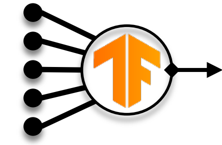
A naïve approach: **Increase M**



Hash trick & Hash collision (IV)



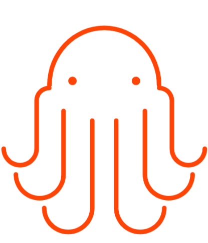
Facing the Large-Scale Continual-Learning Challenge

- **Our server resources are always limited.**
- **Extremely high memory pressure to both the training systems and inference systems**
  - Huge models 
  - Constant streams of data
- **Existing systems (e.g. TensorFlow)**
  - Low memory utilization under the circumstance of large-scale continual learning.
  - Can’t train and serve real-time with giant rec-models.

Problem

How to make large-scale continual learning memory-efficient?

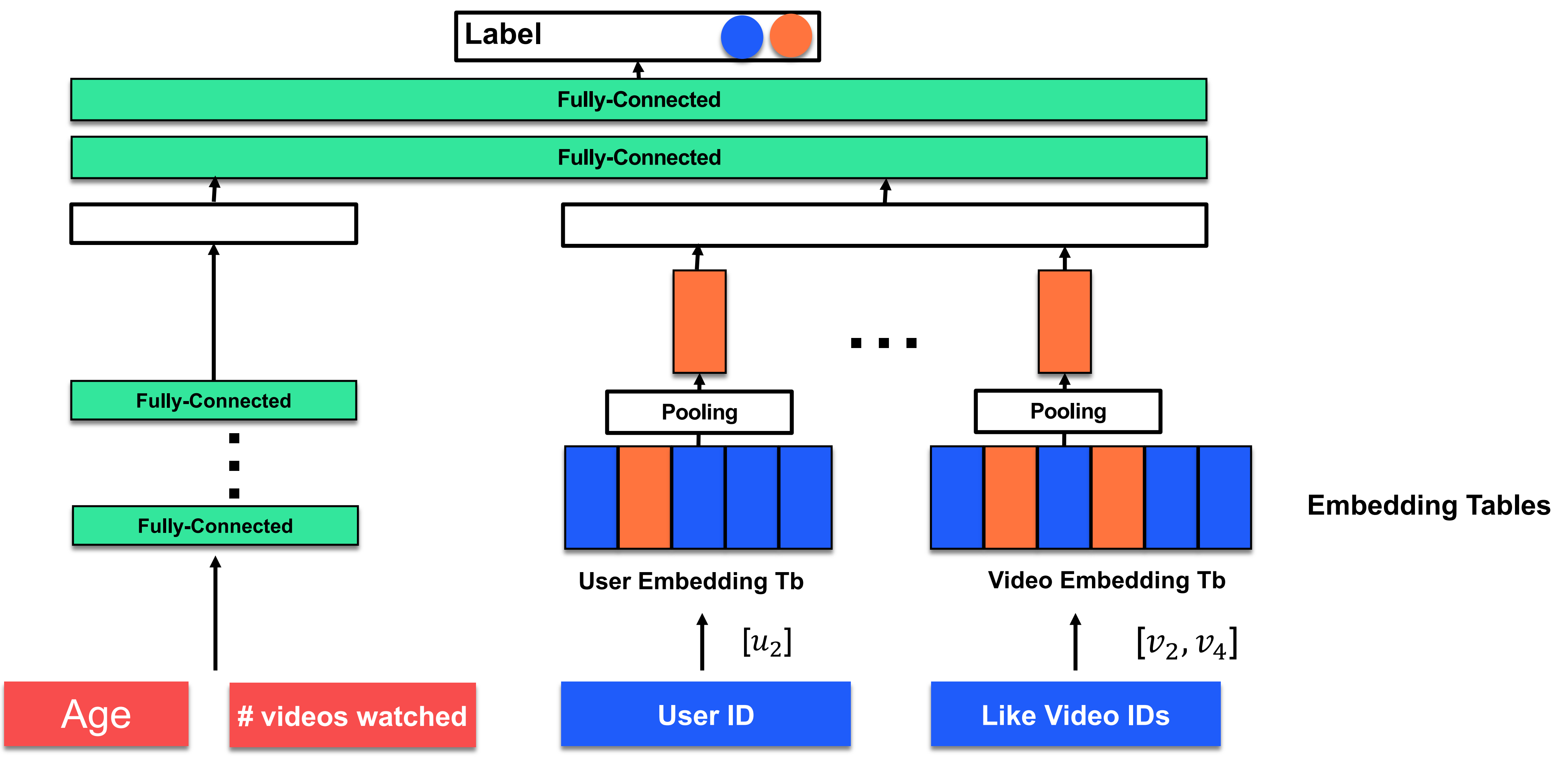
Kraken: Memory Efficient Continual Learning for Large-Scale Real-Time Recommendations



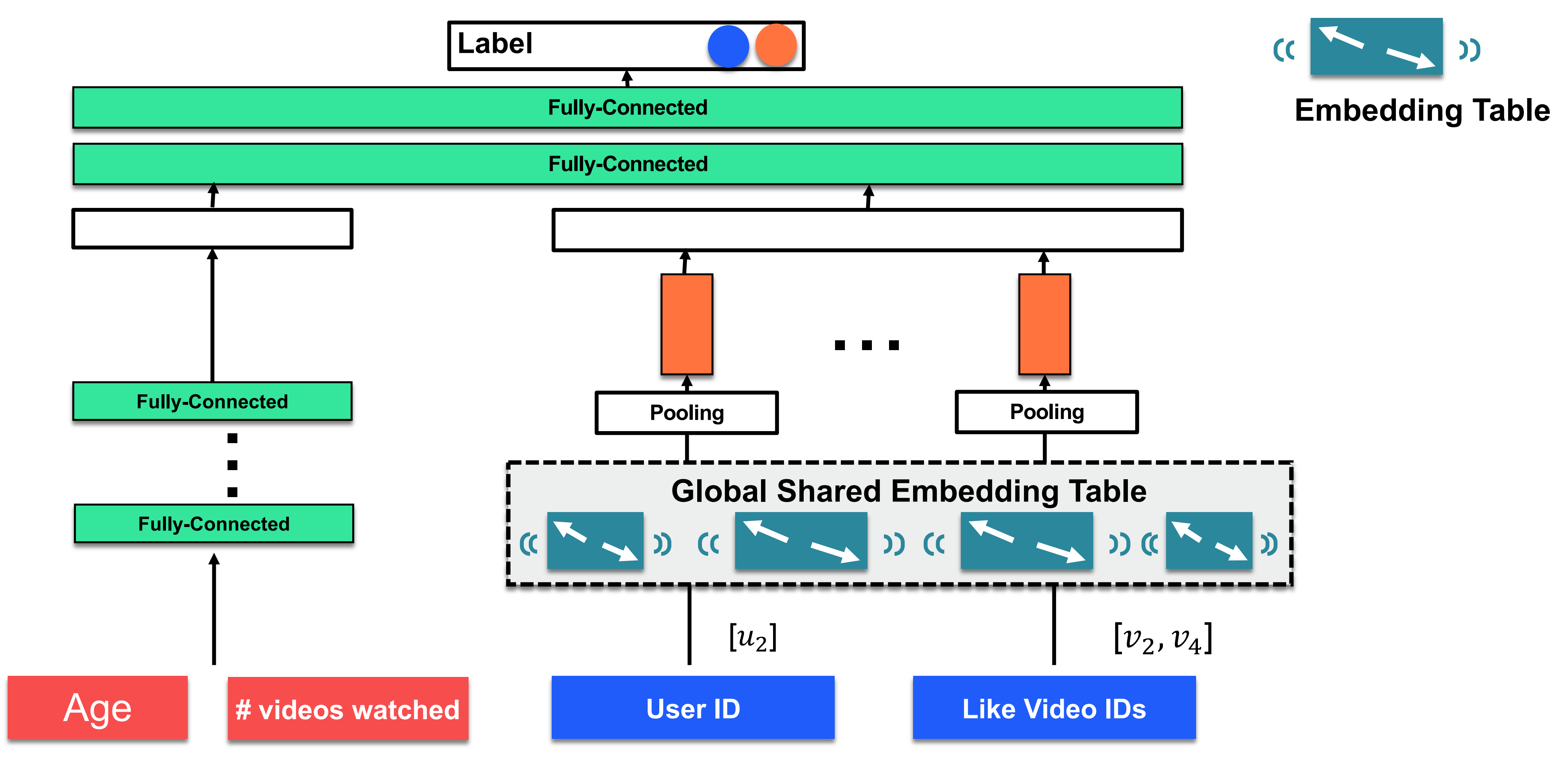
Kraken Overview

- **For both training and serving**
  - Global Shared Embedding Table (GSET).
- **For training**
  - Sparsity-aware training framework.
- **For serving**
  - Efficient continuous deployment and real-time serving.

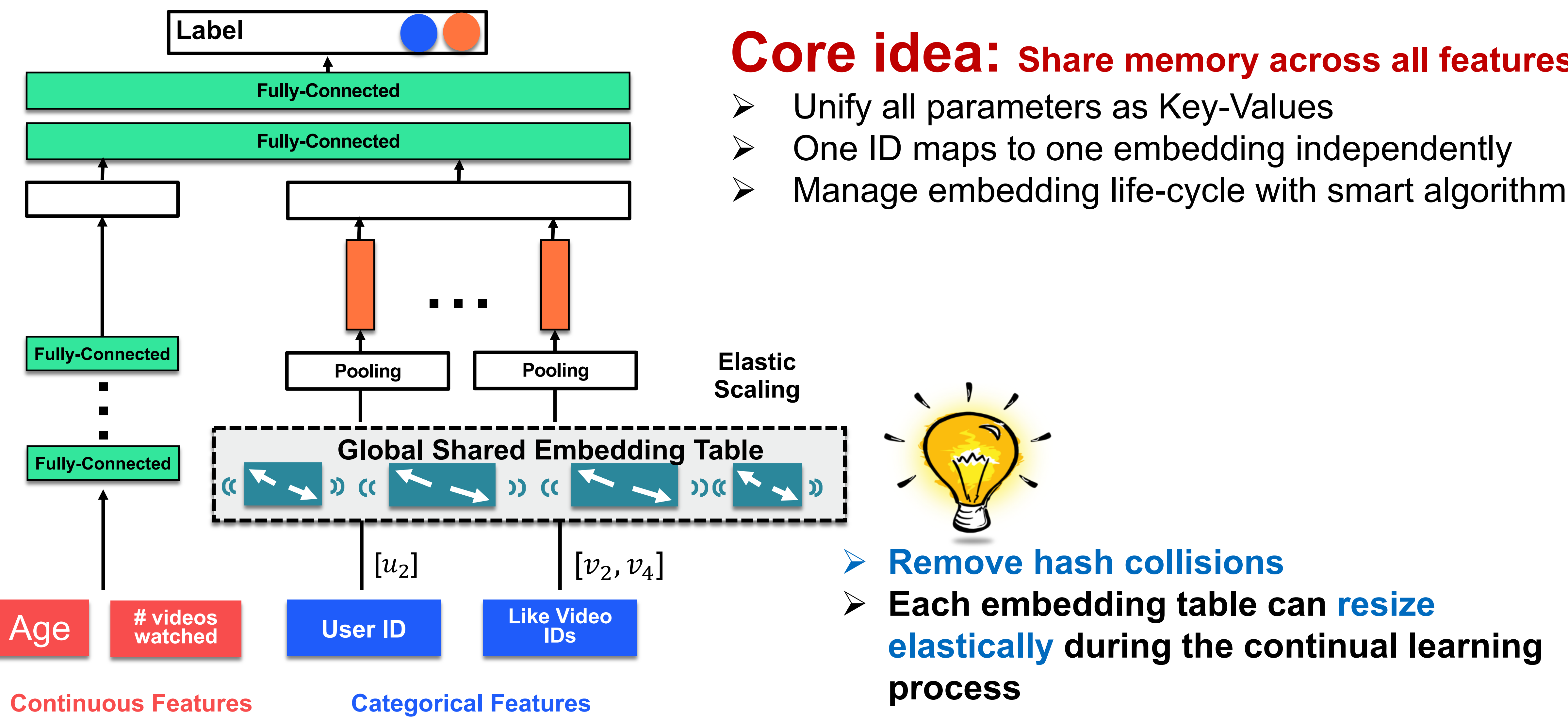
Global Shared Embedding Table (GSET)



Global Shared Embedding Table (GSET)



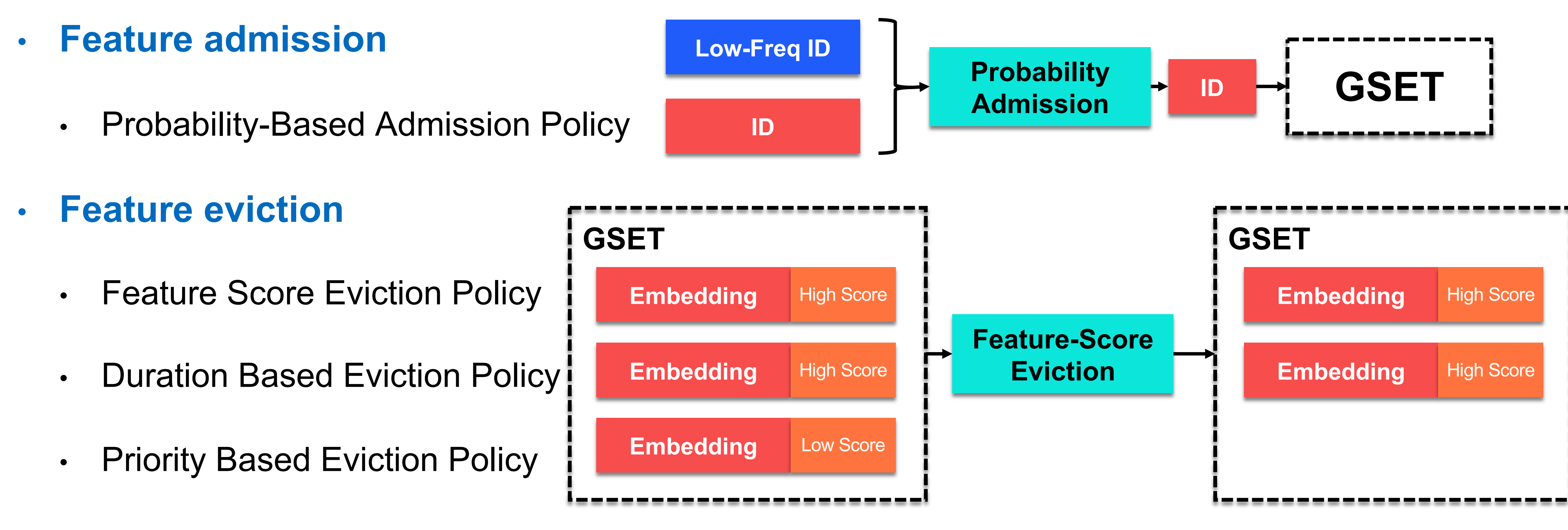
Global Shared Embedding Table (GSET)



GSET: Smart Entry Replacement Algorithms

- Based on **our observations of production**, Kraken supports different policies for ML engineers to customize with their domain knowledge:

- **Feature admission**
  - Probability-Based Admission Policy
- **Feature eviction**
  - Feature Score Eviction Policy
  - Duration Based Eviction Policy
  - Priority Based Eviction Policy



MORE INFO IN PAPER

Kraken Overview

- ~~**For both training and serving**~~
  - ~~Global Shared Embedding Table (GSET).~~
- **For training**
  - Sparsity-aware training framework.
- **For serving**
  - Efficient continuous deployment and real-time serving.

Sparsity-Aware Training Framework

- Embedding compress techniques like hash trick save memory at the cost of accuracy. Kraken sets its sights on the **optimizer state parameters (OSPs)**.

- Different optimizers require different amount of **OSPs**.

Optimizers	Memory Requirement (OSPs)	Adaptive?
SGD	0x	×
AdaGrad	1x	√
Adam	2x	√

Motivation for Sparsity-Aware Training Framework (I)

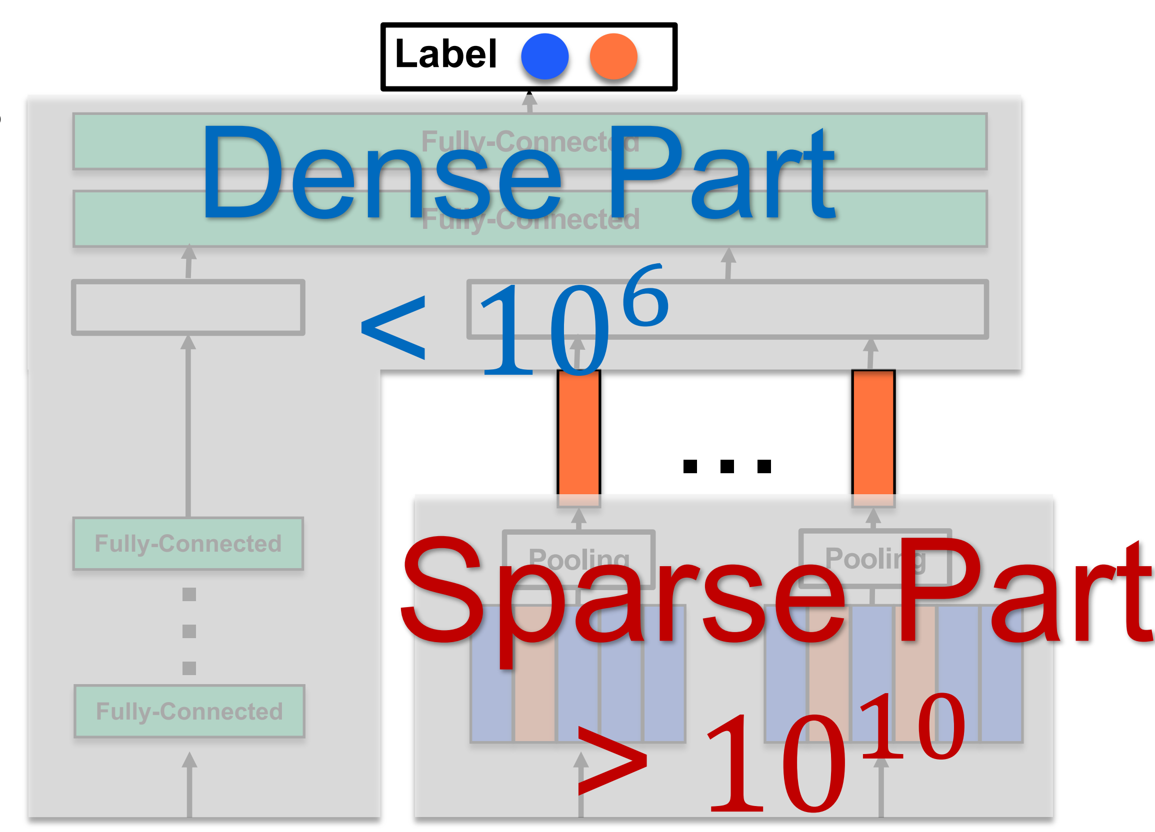
Motivation for Sparsity-Aware Training Framework (II)

AdaGrad 1x

Sparsity-Aware Training Framework

- For the **sparse part** [>10TB]
  - **Adaptive optimizers** with fewer OSPs
  - The closer you get to zero,

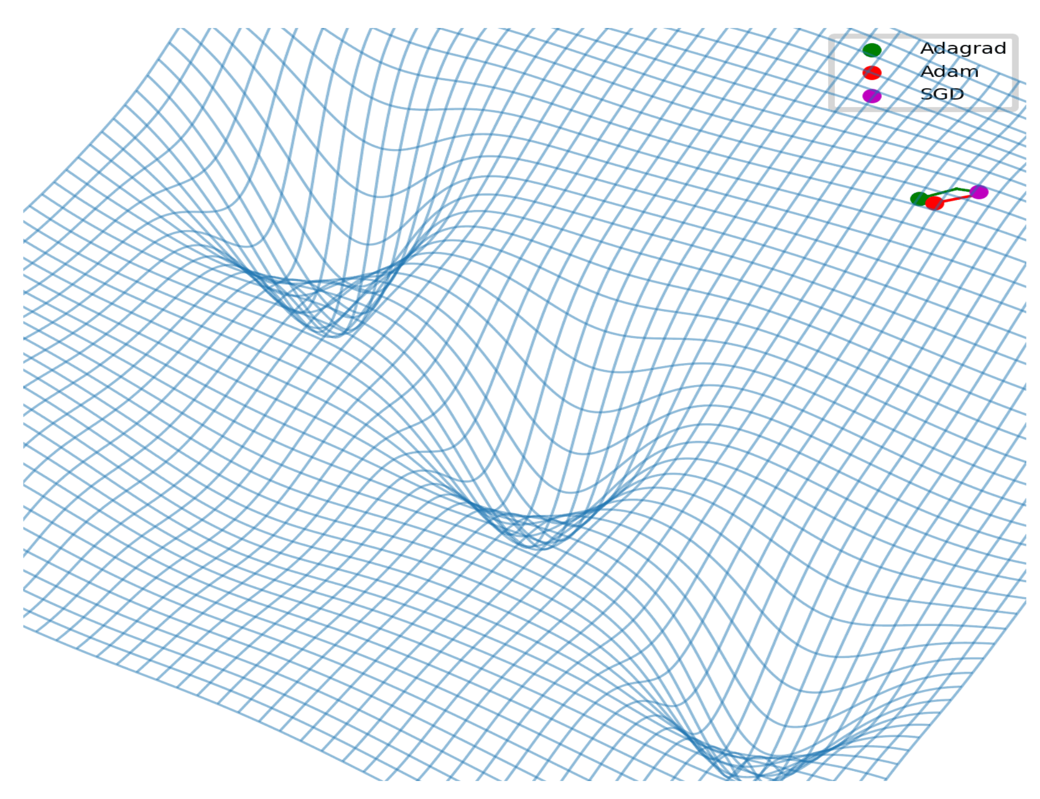
    the more memory you save
- For the **dense part** [<100MB]
  - Adam for better performance
  - It is tolerable in spite of 2x OSPs



Motivation for Sparsity-Aware Training Framework (III)

SGD 0x

Adaptive Optimizers Make Better



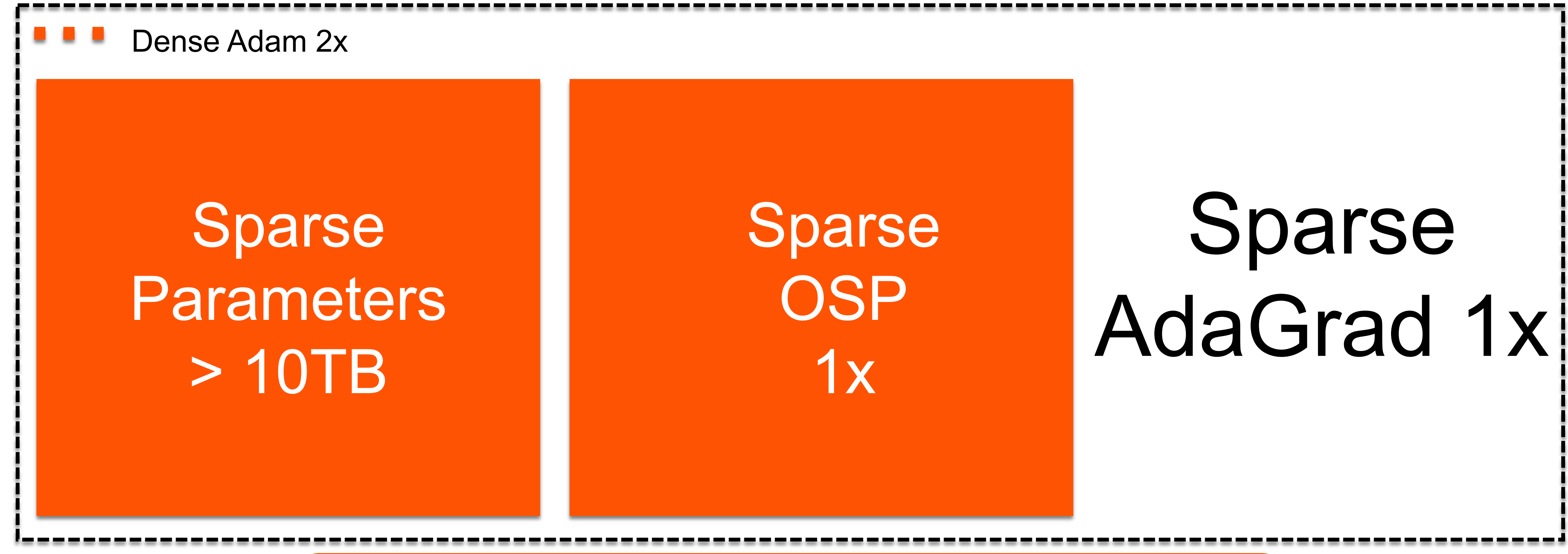
Small learning rate.

Adagrad
Adam
SGD

Big learning rate.

Code from https://github.com/Jaewan-Yun/optimizer-visualization

Adam for the Dense Part
AdaGrad for the Sparse Part



Is that the limit?
Can we save more memory resources?

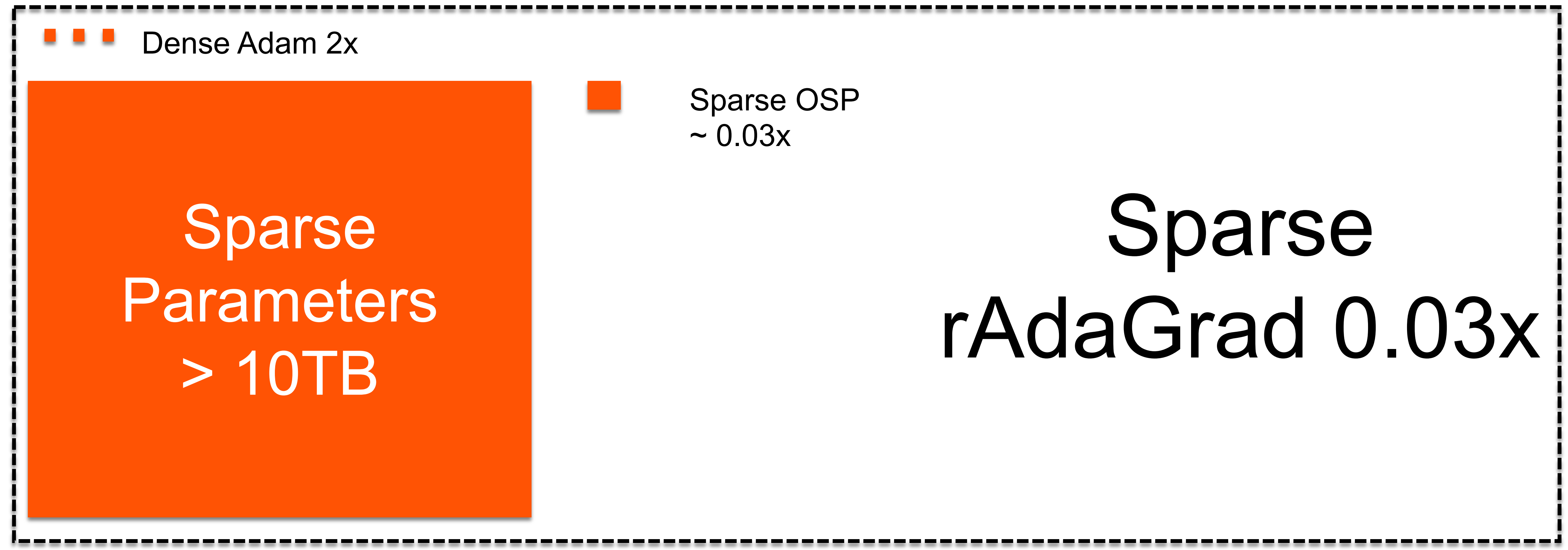
Sparsity-Aware Training Framework

- **rAdaGrad**
  - An adaptive optimizer extremely suitable for sparse parameters.
  - Storing **only one float** for each embedding (usually 32-64 floats).

$$w_{t+1} = w_t - \alpha \frac{g_t}{\sum_{\tau=1}^{t} \left|\left|g_t\right|\right|_2^2 * \mathbf{1}}$$

MORE INFO IN PAPER

Adam for the Dense Part
rAdaGrad for the Sparse Part

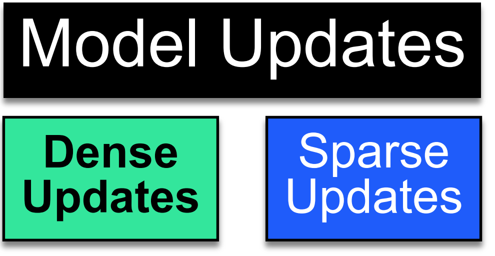


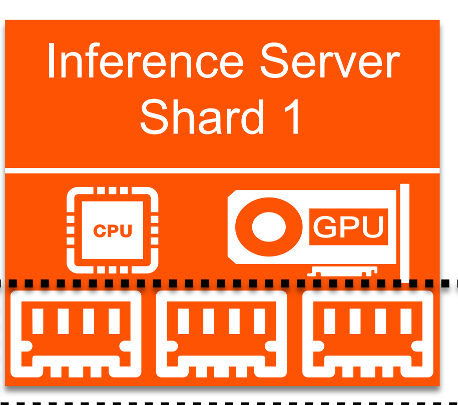
SGD-like memory resources, but great performance

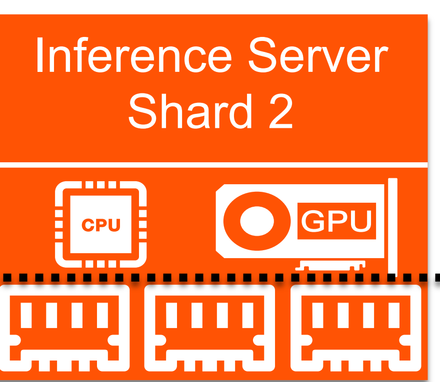
Kraken Overview

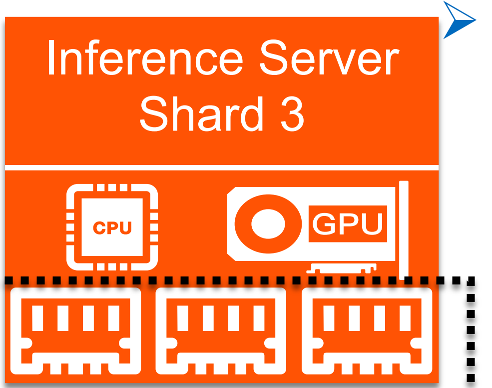
- ~~**For both training and serving**~~
  - ~~Global Shared Embedding Table (GSET).~~
- ~~**For training**~~
  - ~~Sparsity-aware training framework.~~
- **For serving**
  - Efficient continuous deployment and real-time serving.

A Naïve Method: Co-Located Deployment





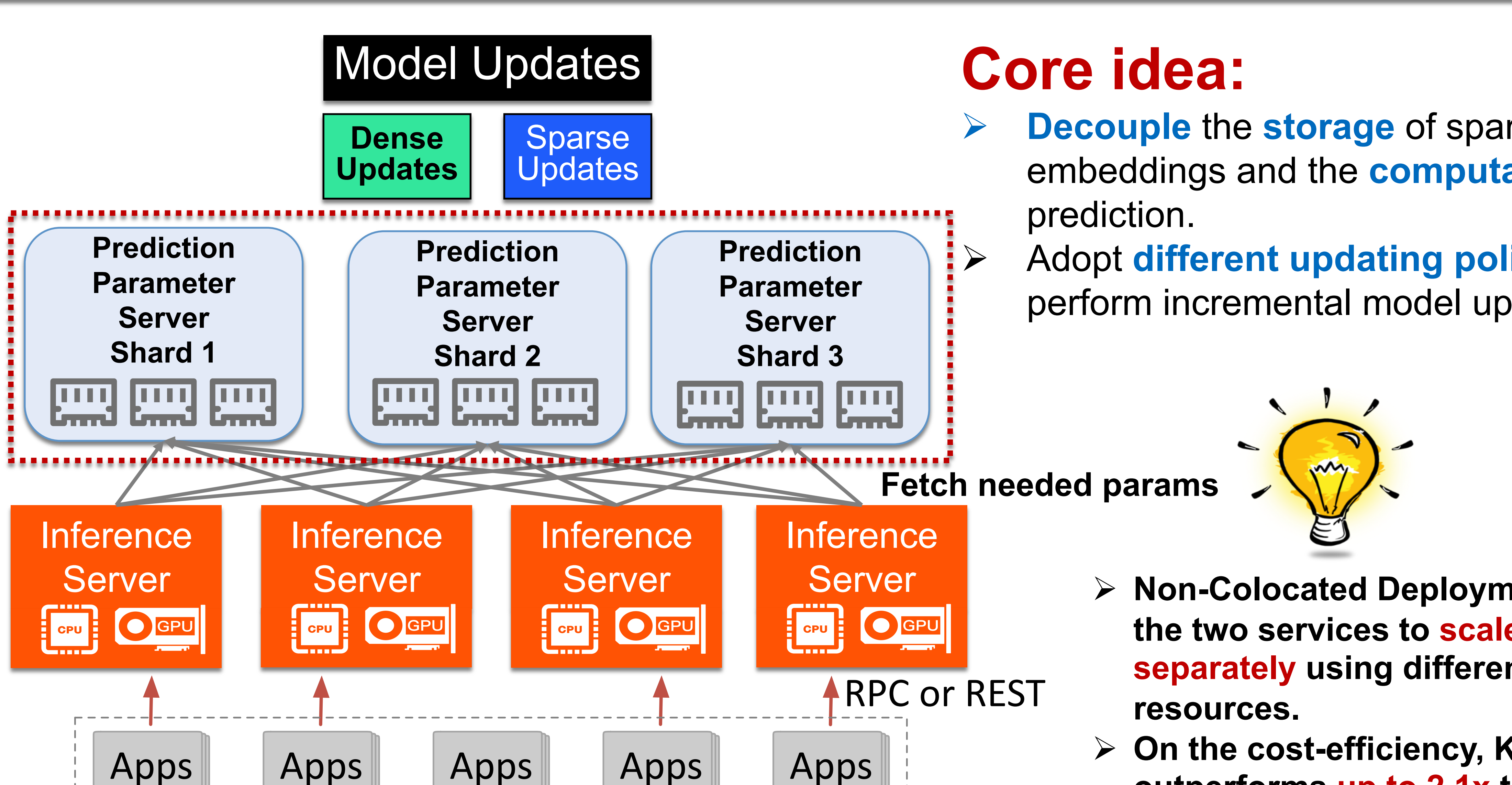




Drawbacks:

- **Introduce High CapEx** because every inference server requires high capability DRAM to store a part of sparse parameters
- **Waste NIC bandwidth & CPU** for constant model updates

Non-Colocated Deployment: Efficient for Real-Time Serving



Core idea:

- **Decouple** the **storage** of sparse embeddings and the **computation** of prediction.
- Adopt **different updating policies** to perform incremental model updates.

- **Non-Colocated Deployment allows the two services to scale up separately using different hardware resources.**
- **On the cost-efficiency, Kraken outperforms up to 2.1x than baseline.**

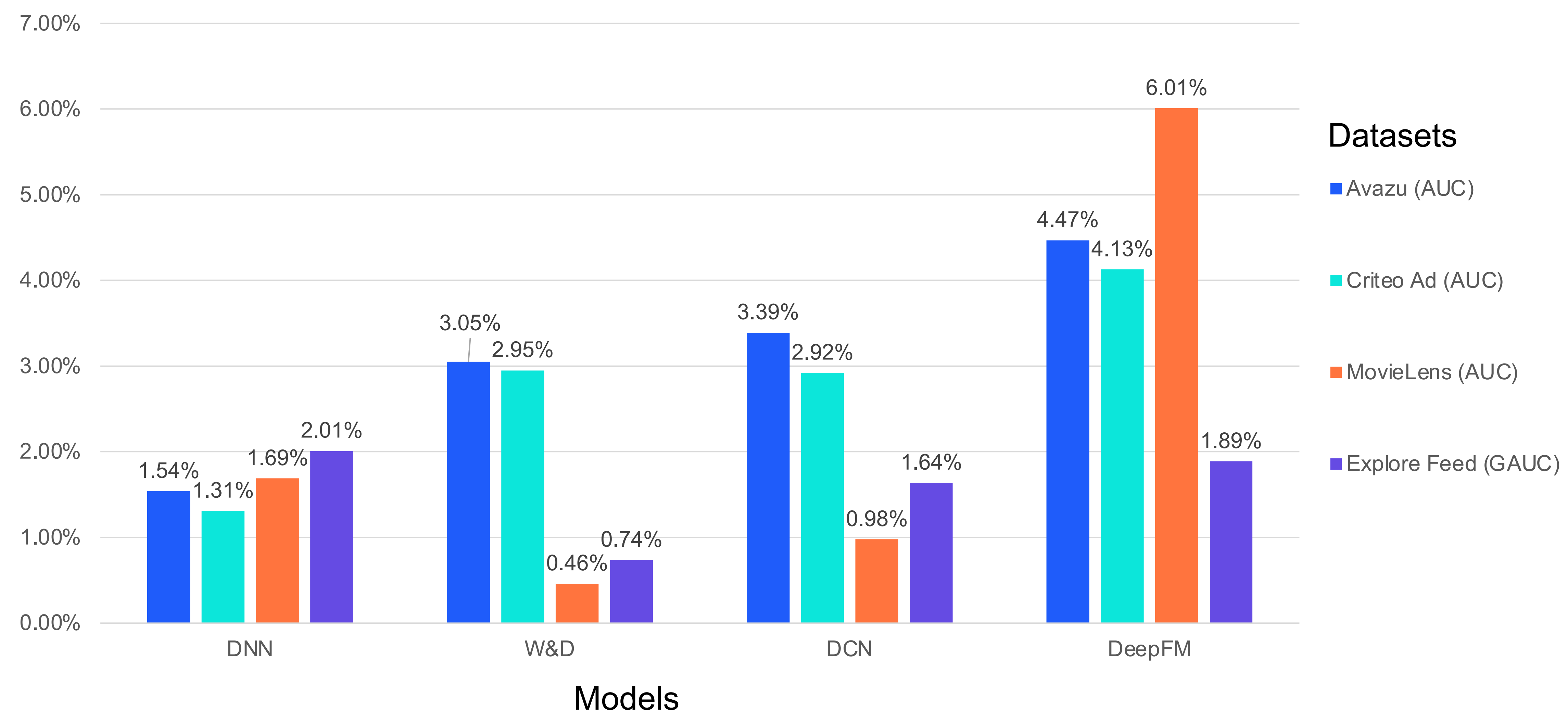
Evaluation

- **Dataset**
  - 3 public & 2 production datasets
  - Learn in an online learning manner
- **Four industrial models**
  - DNN、Wide and Deep、DeepFM、 Deep Cross Network
- **Metric:** AUC & Group AUC (GAUC)*
- **Baseline:** TensorFlow with default embedding tables and Adam optimizer
- **Kraken:** with GSET and sparsity-aware training optimizer

Datasets		# Sparse IDs	# Samples	# Parameters
Public Datasets	Criteo Ad	33M	45M	0.5B
	MovieLens	0.3M	25M	2M
	Avazu CTR	49M	40M	0.8B
Production Datasets	Explore Feed	45M	50M	0.5B
	Follow Feed	1.3B	10B	50B

* H. Zhu, J. Jin, C. Tan, F. Pan, Y. Zeng, H. Li, and K. Gai, "Optimized cost per click in taobao display advertising," in Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ser. KDD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 2191–2200. [Online]. Available: https://doi.org/10.1145/3097983.3098134

Overall Performance Improvement with the same memory (enough to hold 60% of all IDs' embeddings)



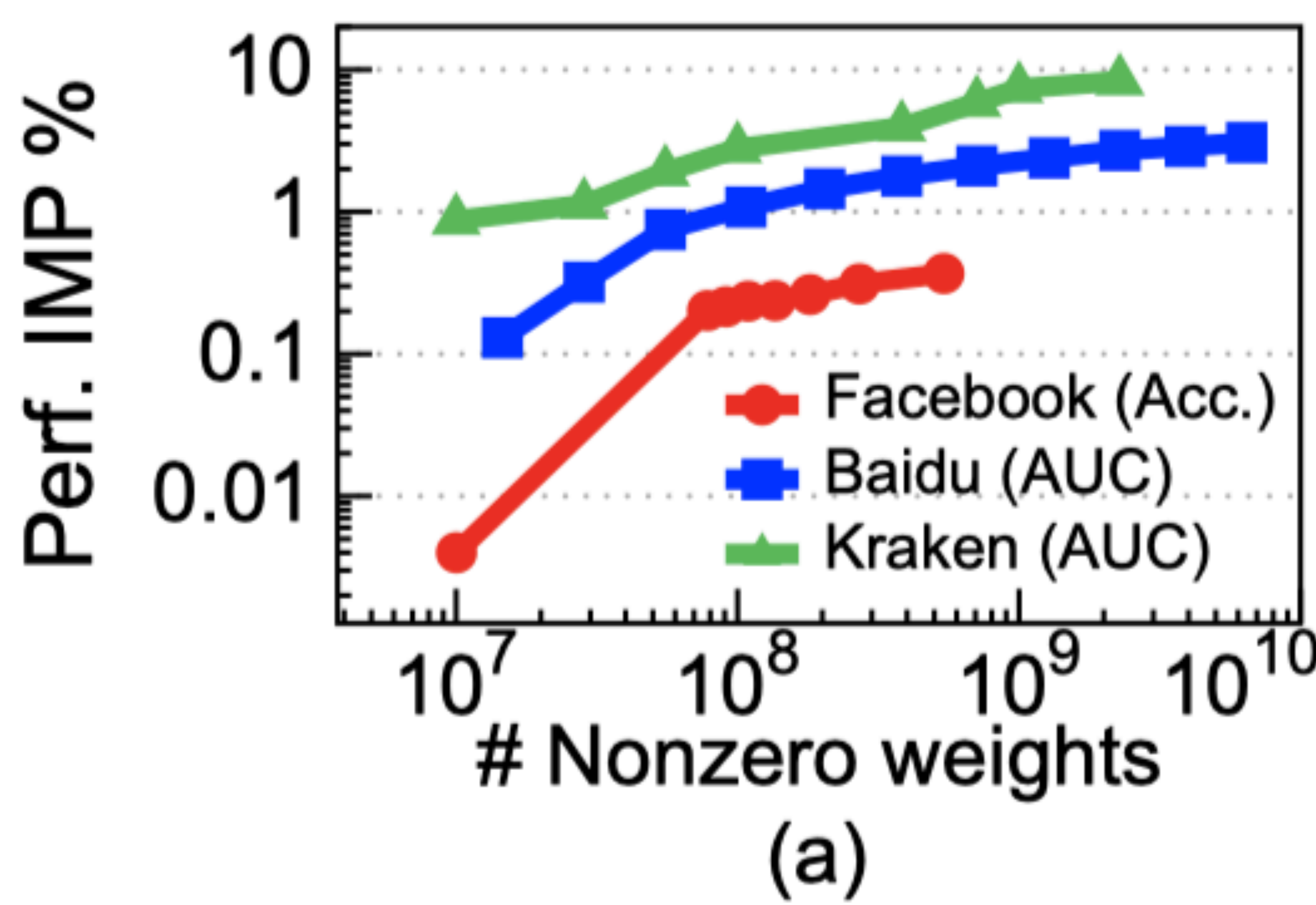
Kraken benefits performance consistently on different datasets and models

Conclusion

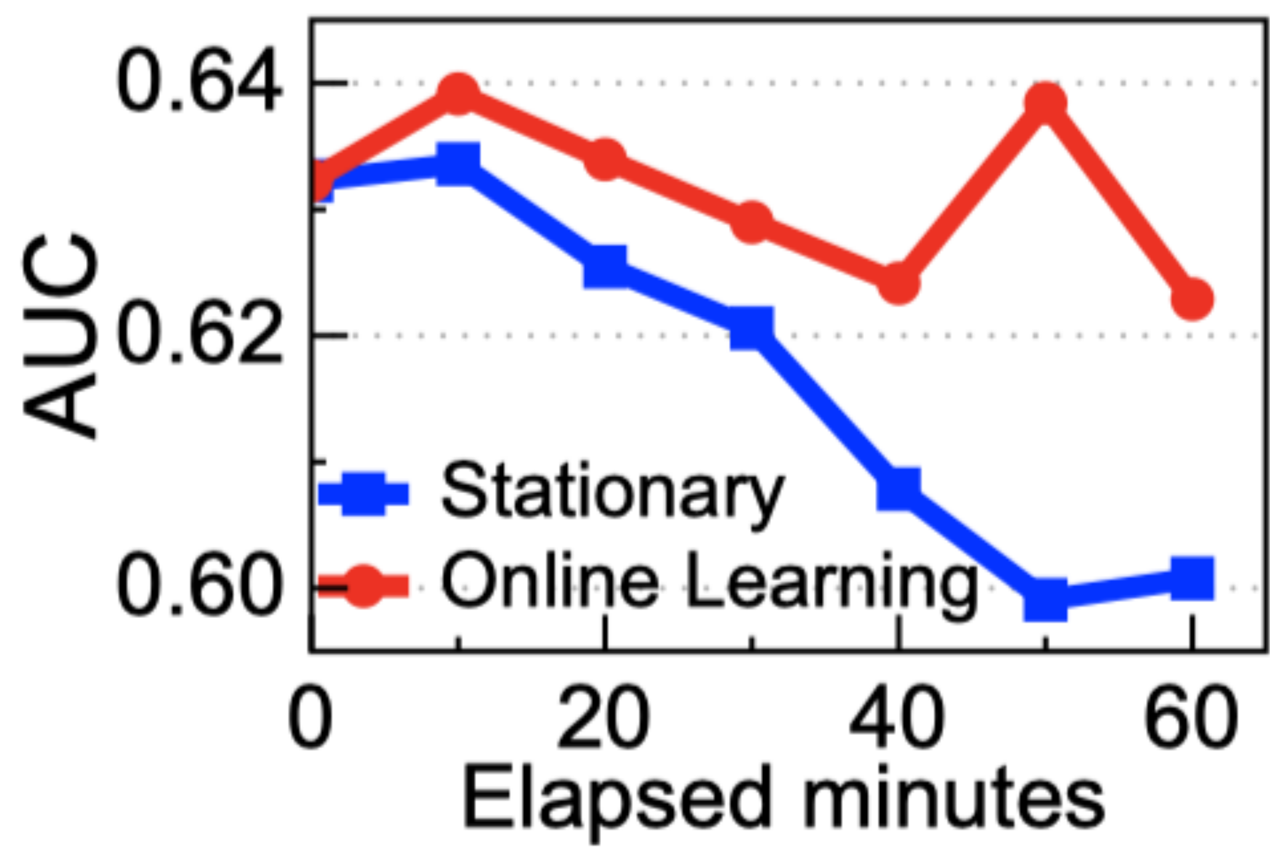
- An **in-production** continual learning system for **large-scale recommendation** with
  - **A Memory-Efficient Design**
    - **Share memory** among traditional embedding tables
    - **Distinguish** the dense part and sparse part in continual training
  - **Enabling Real-Time Recommendation**
    - **Decouple** the storage and computation of models for real-time serving

Thank you!

Large models make better



Online Model V.S. Stationary Model



GSET under different memory budgets

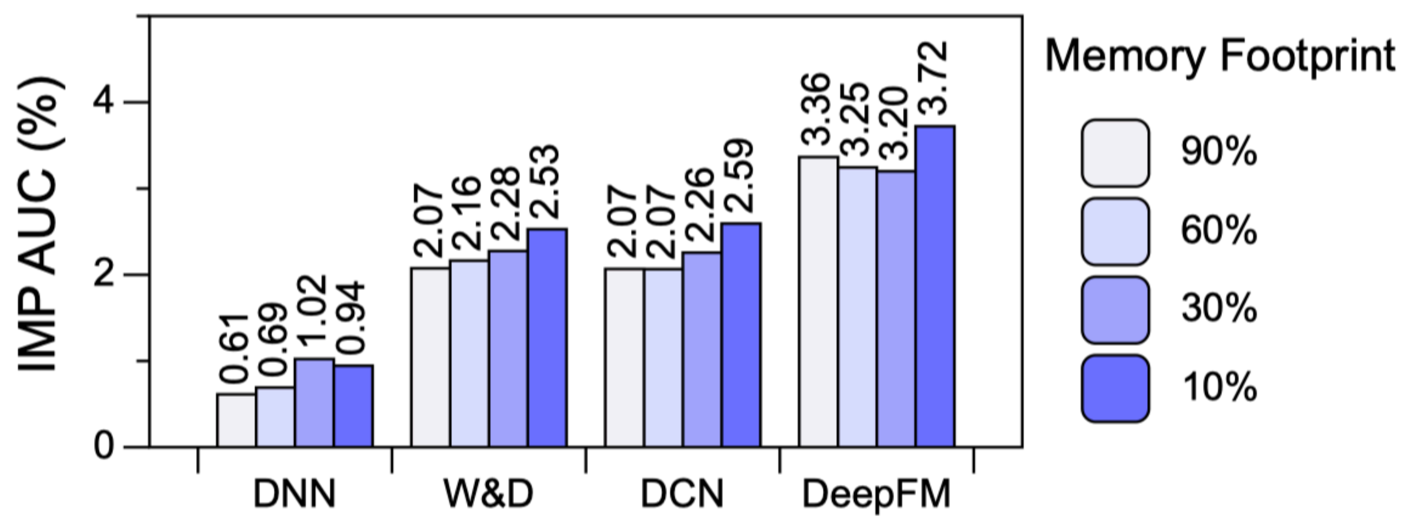


Fig. 8: The AUC improvement of four models in TensorFlow and Kraken under several memory footprints on the Criteo dataset. The percentage represents the corresponding proportion of all original features that memory can hold at most.

Feature admission probabilities

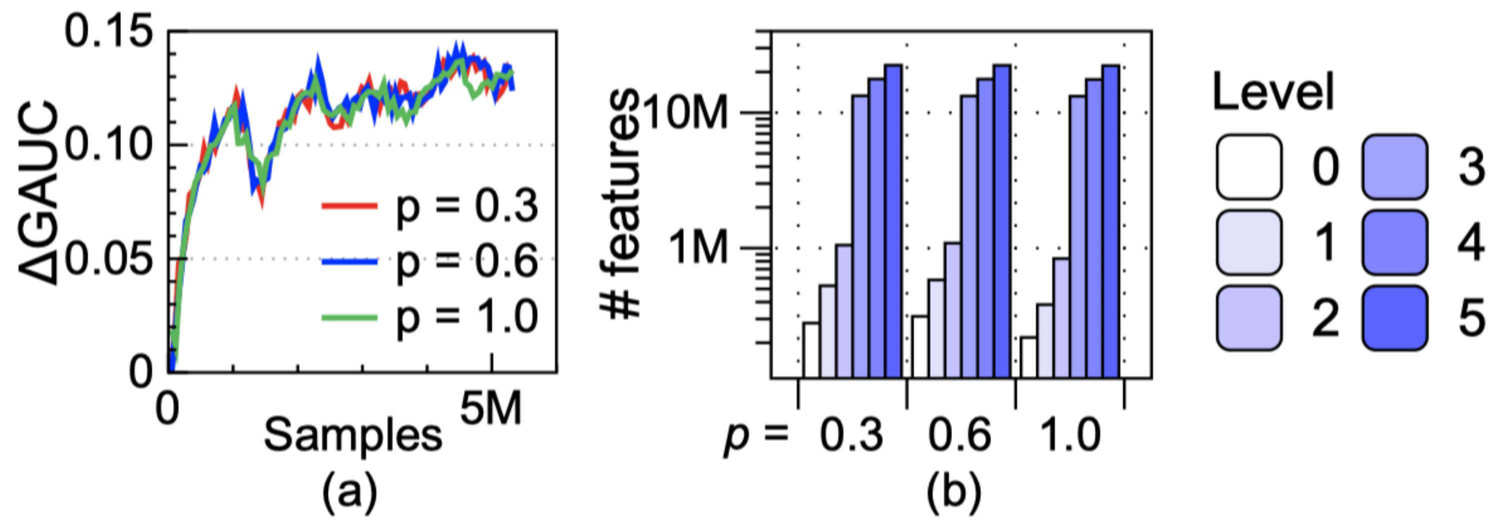


Fig. 9: With different probabilities of feature admission p, (a) shows the relative GAUC of Kraken and (b) shows the number of different frequency-levels of features in the last training-hour. Level i counts the number of features whose frequency is between 2^i to 2^{i+1}.

Different Eviction Policy

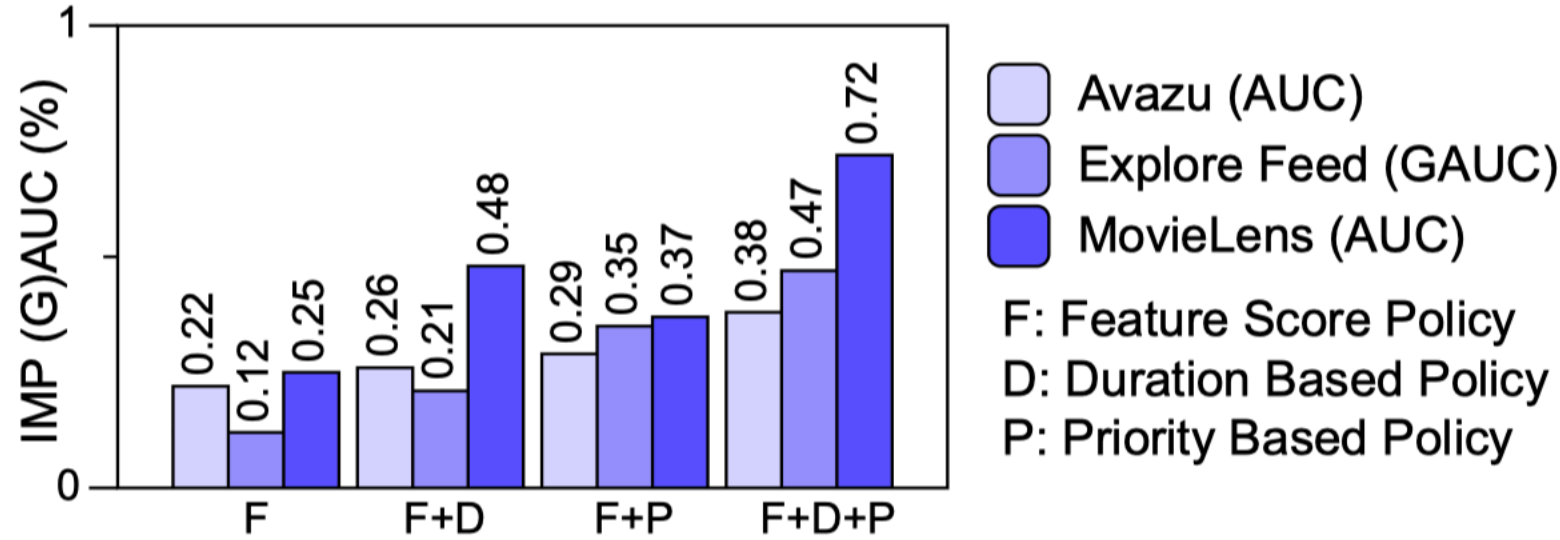


Fig. 10: Contribution of different eviction policies to the model performance. The improved AUC over the raw LFU are shown.

Evaluation of Hybird Optimizer

	Dense Opt	Sparse Opt	Memory Usage	Criteo				MovieLens				Avazu			
				DNN	W&D	DeepFM	DCN	DNN	W&D	DeepFM	DCN	DNN	W&D	DeepFM	DCN
Vanilla Optimizer	SGD		1x	0.7979	0.7896	0.7986	0.7908	0.7760	0.7760	0.7979	0.8019	0.7434	0.7436	0.7502	0.7573
	AdaGrad		2x	0.8001	0.7899	0.8016	0.7992	0.8062	0.8062	0.8061	0.8062	0.7727	0.7795	0.7815	0.7799
	Adam		3x	0.8066	0.7893	0.7956	0.7955	0.8102	0.8112	0.8153	0.8147	0.7559	0.7623	0.7638	0.7631
Hybrid Optimizer	Adam	AdaGrad	$\tilde{2}$x	0.8048	0.8005	0.8057	0.8044	0.8177	0.8184	0.8198	0.8191	0.7734	0.7786	0.7803	0.7807
	Adam	SGD	$\tilde{1}$x	0.7974	**0.7988**	0.8038	0.8026	0.7974	0.8018	0.8045	0.8140	0.7487	0.7646	0.7665	0.7638
	Adam	rAdaGrad	$\tilde{1}$x	**0.8010**	0.7907	**0.8048**	**0.8048**	**0.8132**	**0.8132**	**0.8178**	**0.8153**	**0.7653**	**0.7779**	**0.7800**	**0.7772**
AUC IMP % with the same memory w.r.t vanilla optimizer			1x	0.38	1.17	0.78	1.77	4.79	4.79	2.49	1.67	2.95	4.61	3.97	2.63
			2x	0.59	1.34	0.51	0.65	1.43	1.51	1.70	1.60	0.09	-0.12	-0.15	0.10

TABLE IV: Comparisons of Vanilla and Hybrid Optimizer performances on different datasets and models. The last two rows listed here are to clarify the improved AUC of Hybrid Optimizer respect to Vanilla Optimizer with the same memory usage.

Non-Colocated Deployment

	# of servers with / without large memory	Throughput (QPS)	Total Rent ($ per month)		Ratio	
			AWS	Alibaba	AWS	Alibaba
Baseline	400 / 0	30,325	1,041,408	666,750	29.12	45.48
Kraken	16 / 384	35,726	802,529	372,512	**37.79**	**95.91**

TABLE V: Kraken (Non-Colocated Deployment) shows better cost-effectiveness (around 1.3× to 2.1×) than baseline (Co-Located Deployment). Ratio=1000*Throughput/Total Rent.