EVStore: Storage and Caching Capabilities for Scaling Embedding Tables in Deep Recommendation Systems

Daniar H. Kurniawan, Ruipu Wang,
Kahfi Zulkifli, Fandi Wiranata,
John Bent, Ymir Vigfusson, Haryadi S. Gunawi


VANCOUVER


TOP 10
VANCOUVER
10:01


Vancouver &
Vancouver Island
SIX-DAY
ITINERARY
37:50


VANCOUVER
TIPS
23:54
RECOMMENDED


TOP 10
PLACES
10:31
RECOMMENDED

Deep Recommendation Systems (DRS)

- **Usage:**
  Personalized **advertisements**, **movies**, **news**, **product recommendation**, etc.

- **Impact:** 30% 40% 60% 75%

What's inside?

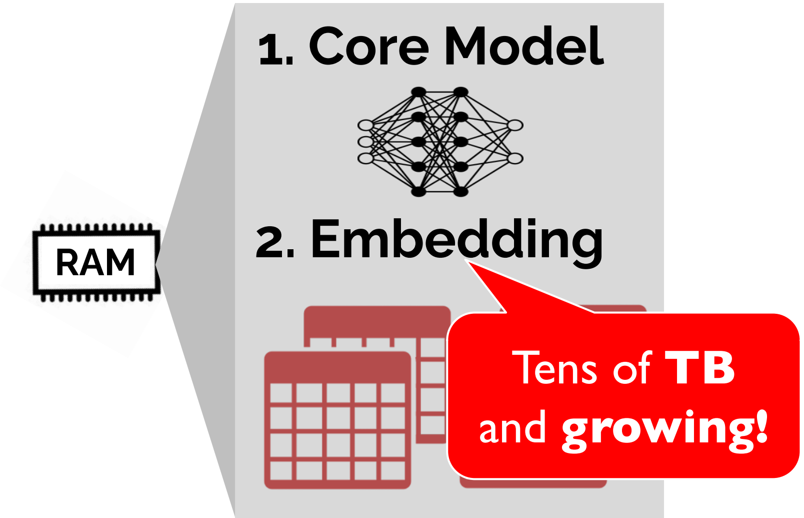
I watch 

???

- Database
- ML Models
- **Embedding Table**

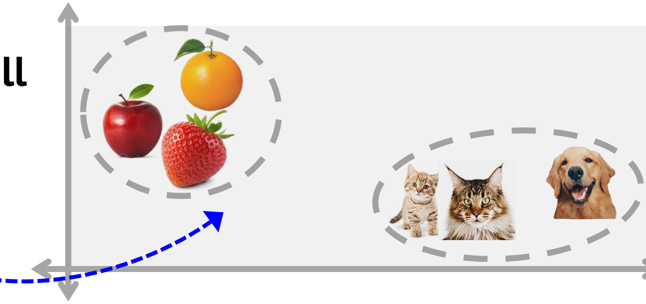
Similar videos are suddenly being **recommended** to me

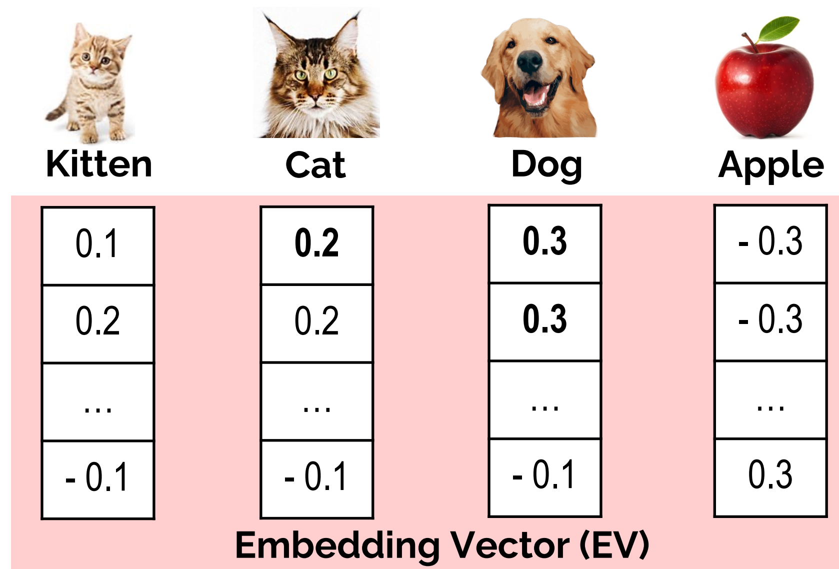
DRS model deployment



- Low latency
- Poor scalability
- Very expensive

Similar objects will appear **closer** in the vector space

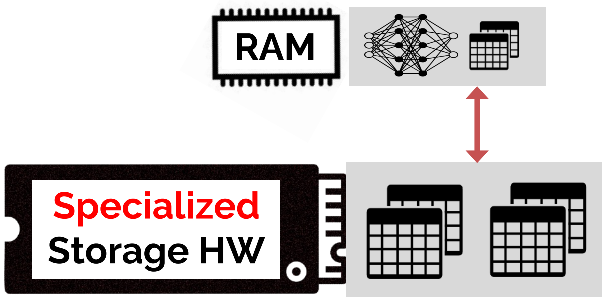




DRS is up against a scaling wall!

Embedding-lookup = ~40% of inference latency

Existing solution:

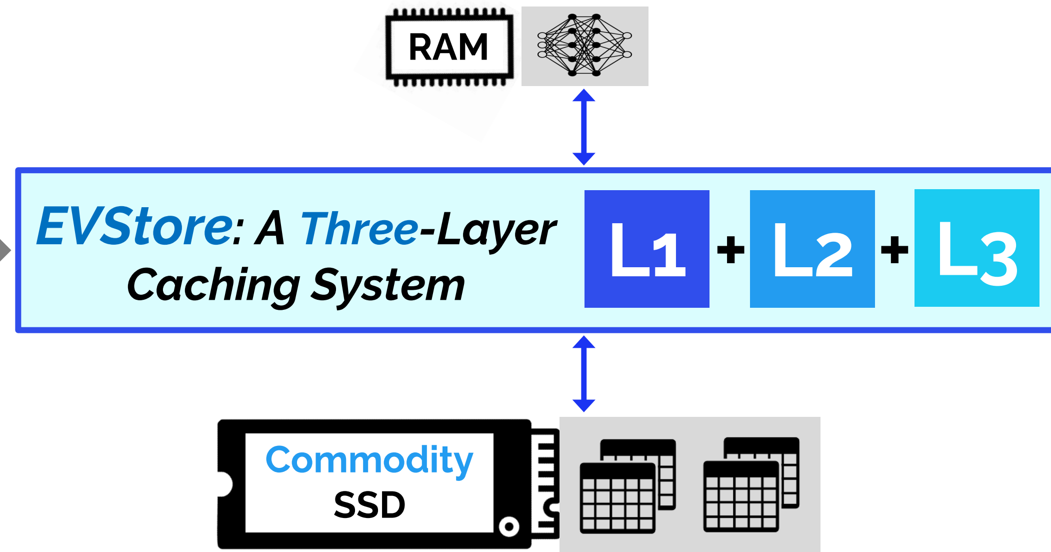


Off-load the embedding table to specialized storage

but...

- Require HW Modification
- Complex to use

Ours:



94% Lighter memory

23% Faster inference

4x Model collocation

EVStore Overview

Existing algorithms (e.g., LRU, Clock, LIRS, LeCAR) are not effective!

***EVCache:** Optimized for **multi-table** lookups*

L1

Let's cache the **smaller precision**!

EVMix: mixed-precision caching

L1 L2

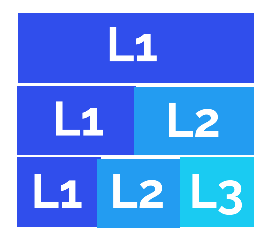
Let's enable **approximation**!

***EVProx:** Approximate Embedding*

L1 L2 L3

The Story of EVStore

- **Goal**: **Scalable and performant** DRS deployment on **commodity** backend
- Design Principles
  - **Simple** policies
  - **Modular** for easy integration
- Limitation: Stale cache problem → flush the cache
- EVStore Approach/Techniques
  - + **EVCache :** Groupability-aware **caching** →
  - + **EVMix : Mix**-precision caching →
  - + **EVProx :** Embedding **approx**imation →



94%	23%	4x
Lighter memory	**Faster inference**	**Model collocation**

- ❑ Background & Motivation
- ❑ EVStore Overview
- ❑ EVStore: Design
  - **EVCache**: Groupability-aware caching
  - **EVMix**: Mixed-precision caching
  - **EVProx**: Embedding approximation
- ❑ EVStore Evaluation
- ❑ Overhead
- ❑ Summary

EVStore$_1$: Groupability-aware caching (EVCache)

EVCache : Extract **groupability relationships** between items to improve **perfect-hit**

L1

lookup(A$_1$, B$_4$, C$_6$, . . Z$_9$)

All hit = PERFECT hit

	Recency	Frequency	Groupability
Other algorithms	✓	✓	✗
EVCache (*ours*)	✓	✓	✓

Optimized to improve the perfect hit rate

Groupability Scoring (1/3)

The **1st** Lookup

A1 B1 C1 D1 E1

✖ ✖ ✖ ✖ ✖

#hit = 0

Cache State (**prior** lookup)	Cache State (**after** lookup)	
EMPTY	A1 → 0	**Newly inserted**
	B1 → 0	**Newly inserted**
	C1 → 0	**Newly inserted**
	D1 → 0	**Newly inserted**
	E1 → 0	**Newly inserted**

Groupability Scoring (2/3)

The 2nd Lookup

A1 B1 C1 **D2** **E2**

#hit = 3

Cache State (**prior** lookup)	Cache State (**after** lookup)	
A1 → 0	A1 → 3	Updated
B1 → 0	B1 → 3	Updated
C1 → 0	C1 → 3	Updated
D1 → 0	D1 → 0	
E1 → 0	E1 → 0	
	D2 → 3	**Newly inserted**
	E2 → 3	**Newly inserted**

Groupability Scoring (3/3)

The 3rd Lookup

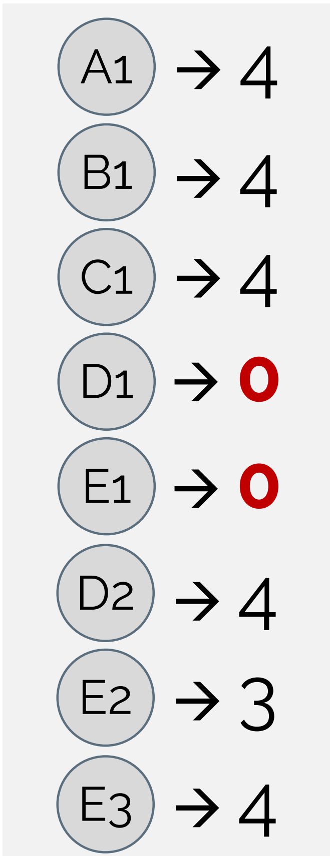
A1	B1	C1	D2	**E3**
✓	✓	✓	✓	✗

#hit = 4

Cache State (**prior** lookup)	Cache State (**after** lookup)	
A1 → 3	A1 → 4	Updated
B1 → 3	B1 → 4	Updated
C1 → 3	C1 → 4	Updated
D1 → 0	D1 → 0	
E1 → 0	E1 → 0	
D2 → 3	D2 → 4	Updated
E2 → 3	E2 → 3	
	E3 → 4	**Newly inserted**

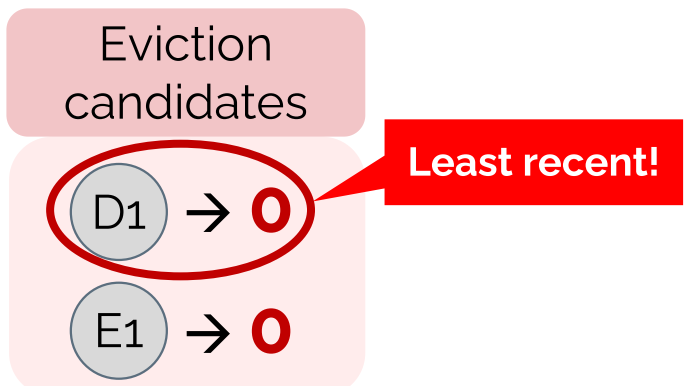
EVCache Eviction

Prior Eviction

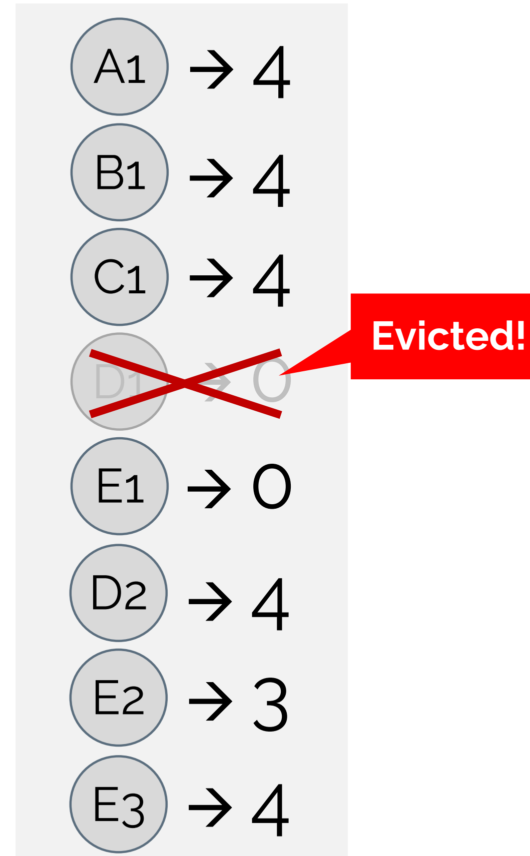


Eviction algorithm:

1. Check **Groupability**
2. Check **Recency**



After Eviction

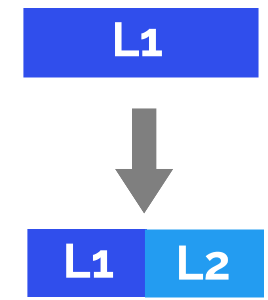


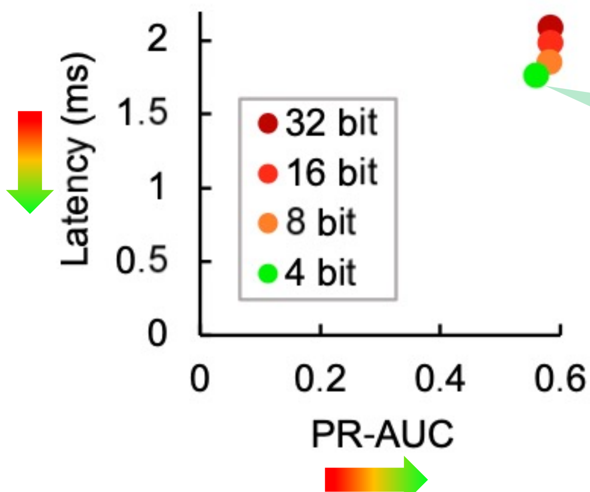
EVStore$_2$: Mixed-precision caching (EVMix)

EVCache : Extract **groupability relationships** between items to improve perfect-hit

+

EVMix : Split the cache into **two** layers, with each layer storing **different precision**





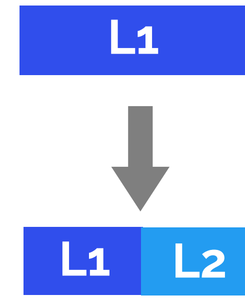
Lower precision = **faster inference** latency

EVStore$_2$: Mixed-precision caching (EVMix)

EVCache : Extract **groupability relationships** between items to improve perfect-hit

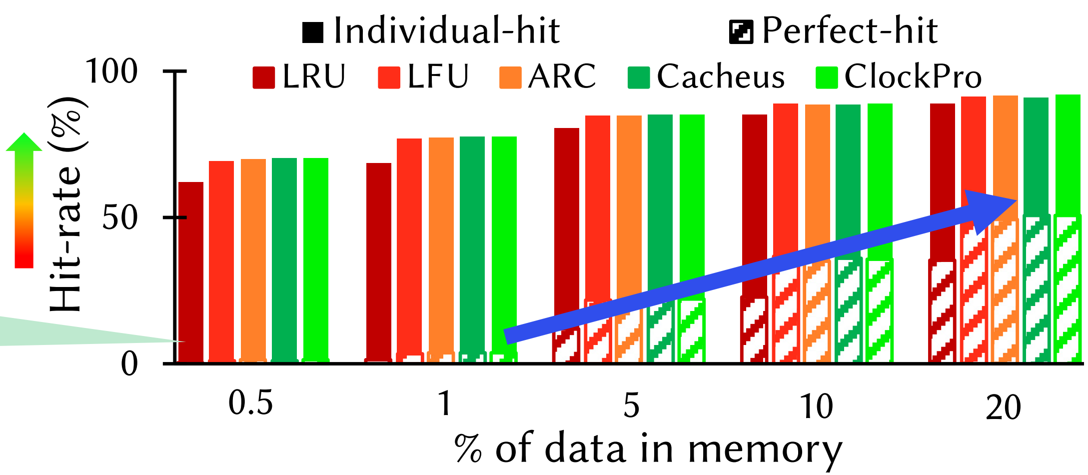
+

EVMix : Split the cache into **two** layers, with each layer storing **different precision**



Lower precision → Cache **more key-value** pairs

The **more data** being cached, the **higher** perfect-hit

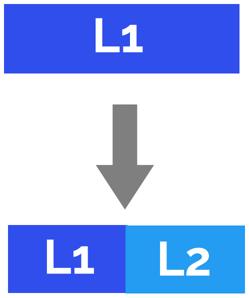


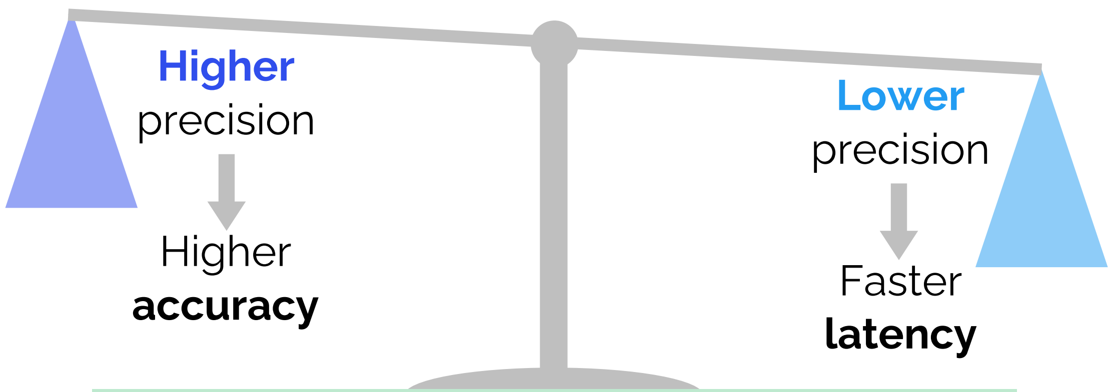
EVStore$_2$: Mixed-precision caching (EVMix)

EVCache : Extract **groupability relationships** between items to improve perfect-hit

+

EVMix : Split the cache into **two** layers, with each layer storing **different precision**





Enable **accuracy vs latency** tradeoff

EVStore$_3$: Embedding value approximation (EVProx)

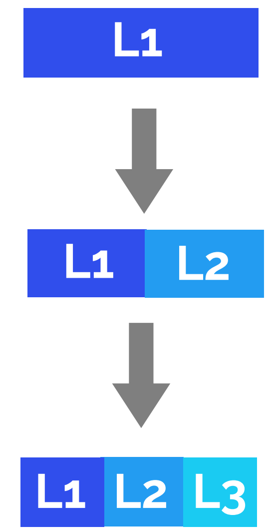
EVCache : Extract **groupability relationships** between items to improve perfect-hit

+

EVMix : Split the cache into **two** layers, with each layer storing **different precision**

+

EVProx : Enable **key-to-key** caching to find an **approximately similar** embedding value



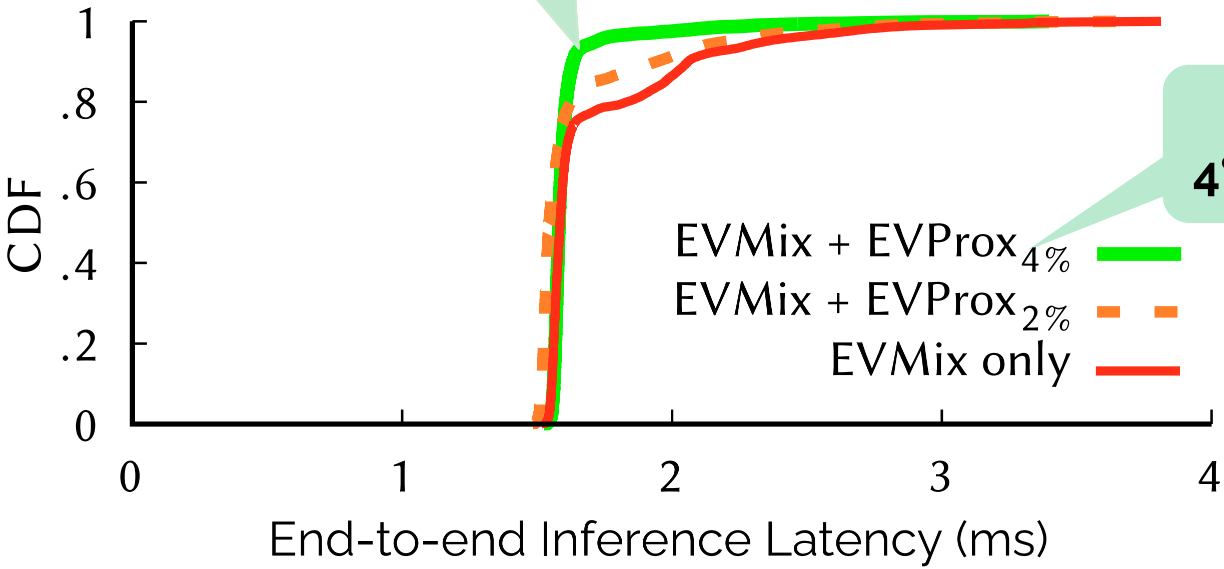
Surrogate keys preprocessing :
1. Run **similarity** analysis
2. Analyze key's **popularity**

Euclidian and **Cosine** dist.

Surrogate key → The **most popular** key out of N-most **similar** keys

The Effectiveness of EVProx

The tail is shifted **from p80 to p98**



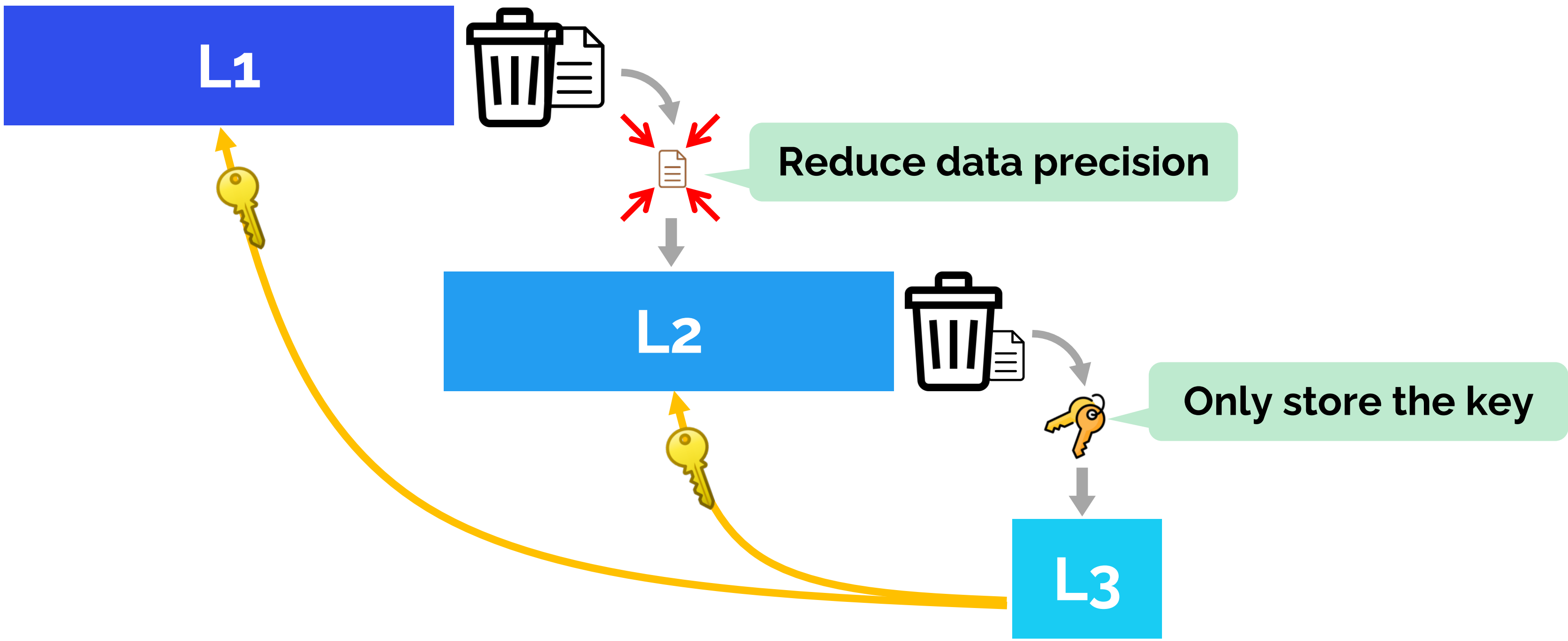
EVProx only uses **4%** of the cache size

"Small but powerful"

EVStore: Data Life Cycle

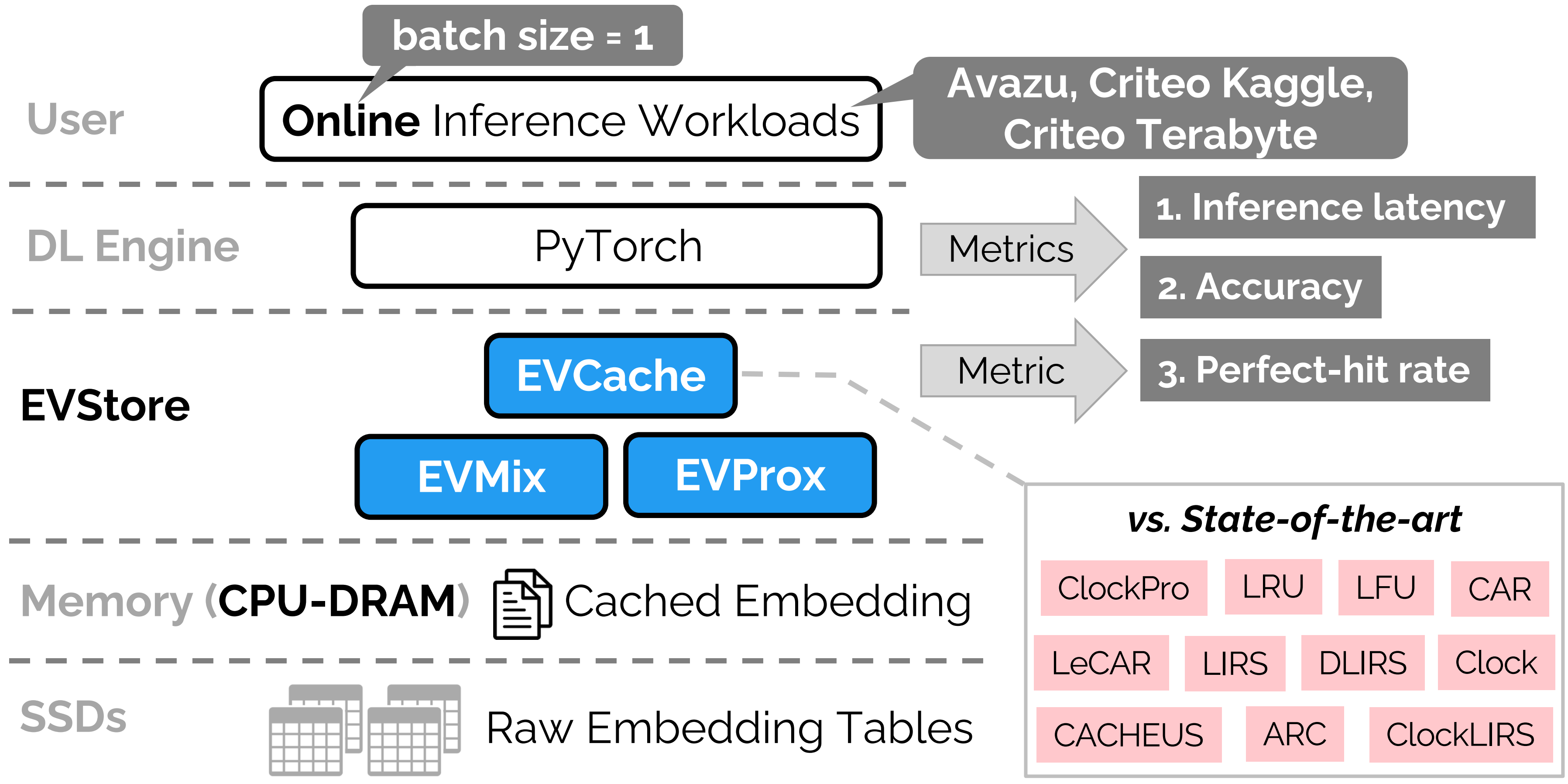
Goal : ❑ **No duplicate** items

❑ **Minimize data trashing**

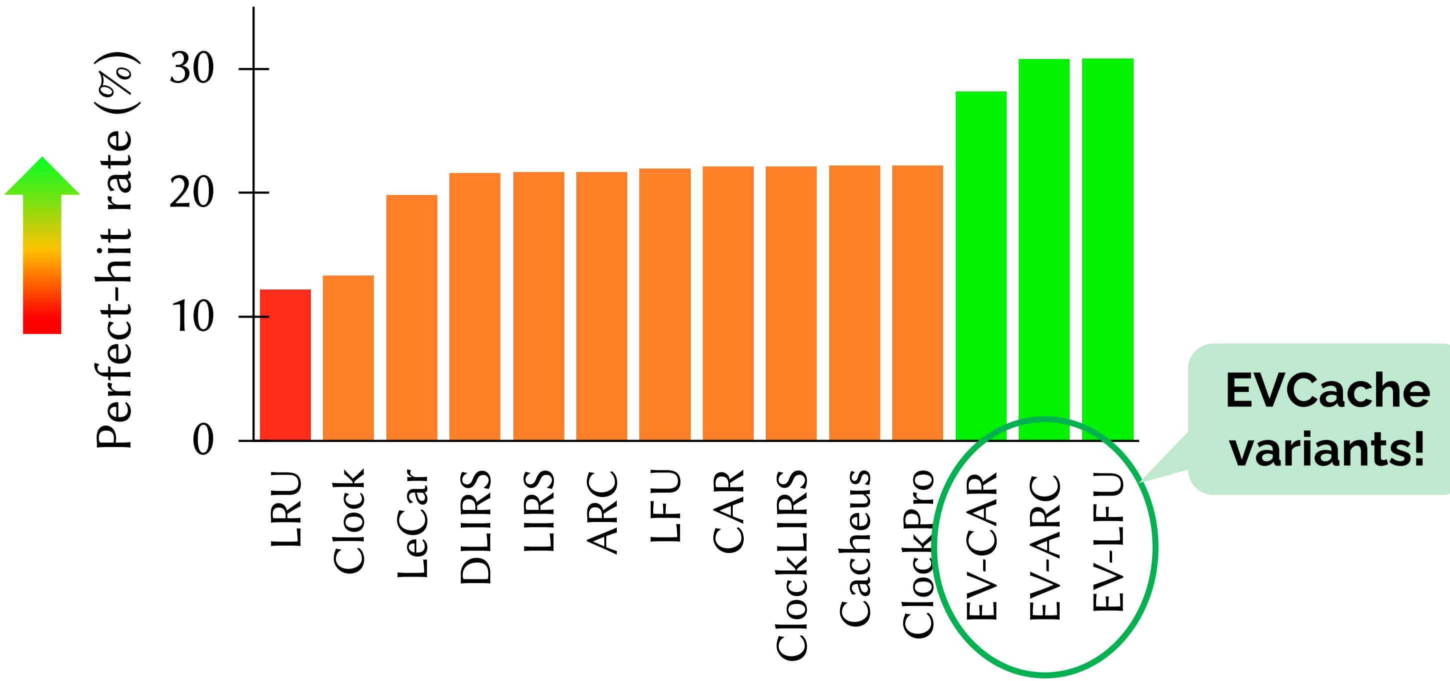


- Background & Motivation
- EVStore Overview
- EVStore Design
  - **EVCache**: Groupability-aware caching
  - **EVMix**: Mixed-precision caching
  - **EVProx**: Embedding approximation
- **EVStore Evaluation**
- **Overhead**
- **Summary**

EVStore Stack and Evaluation Setup

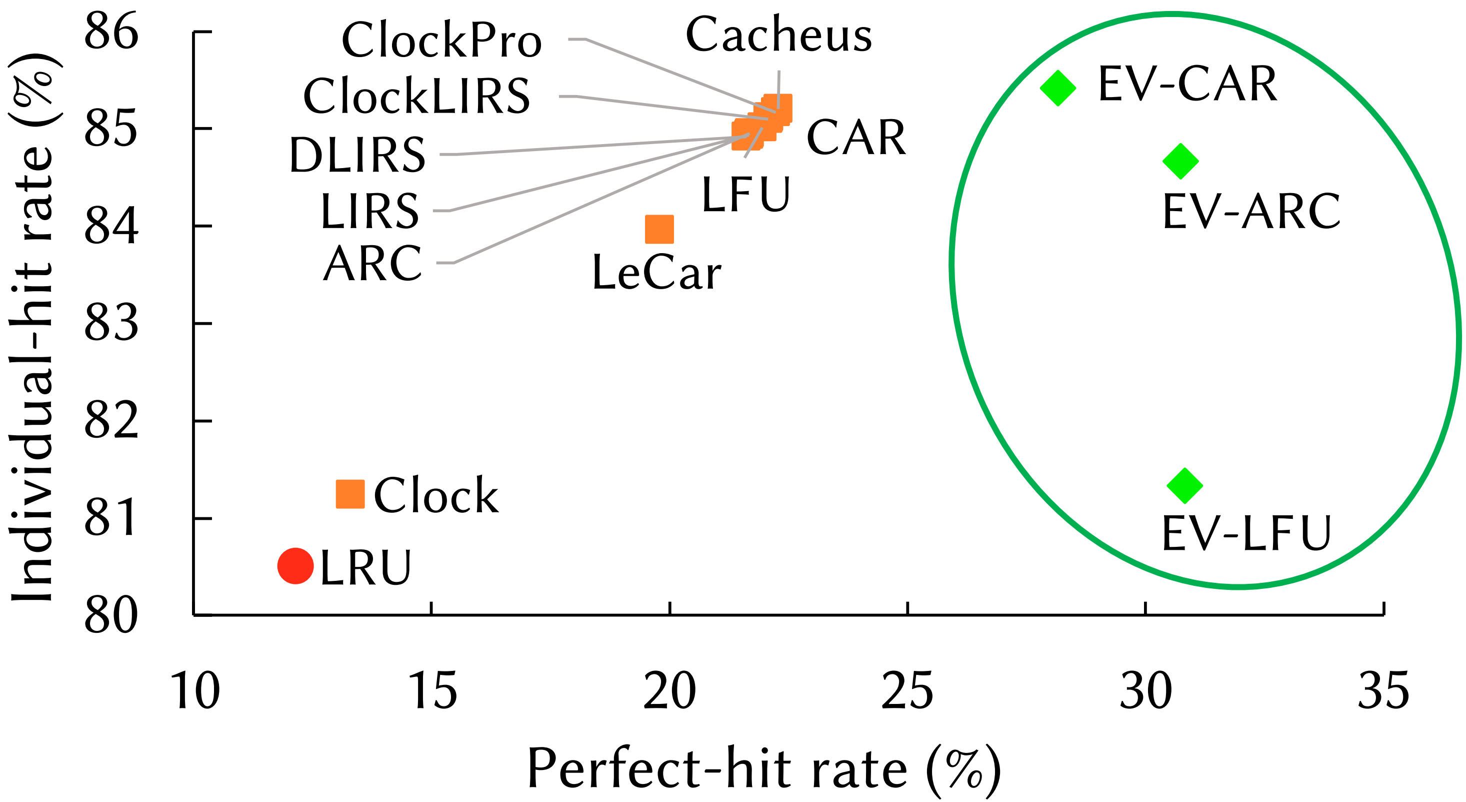


EVCache vs State-of-the-art

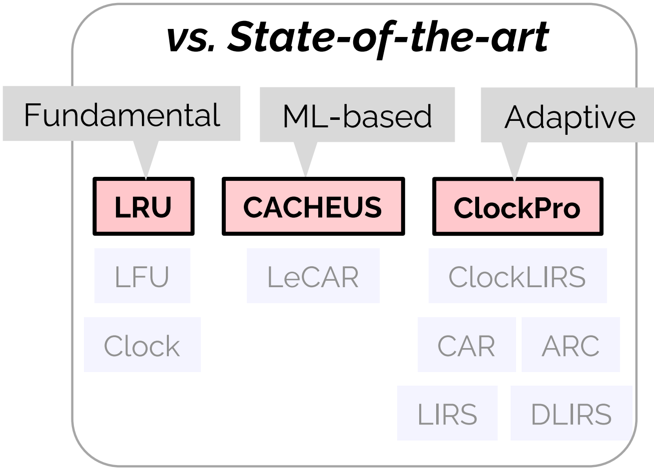


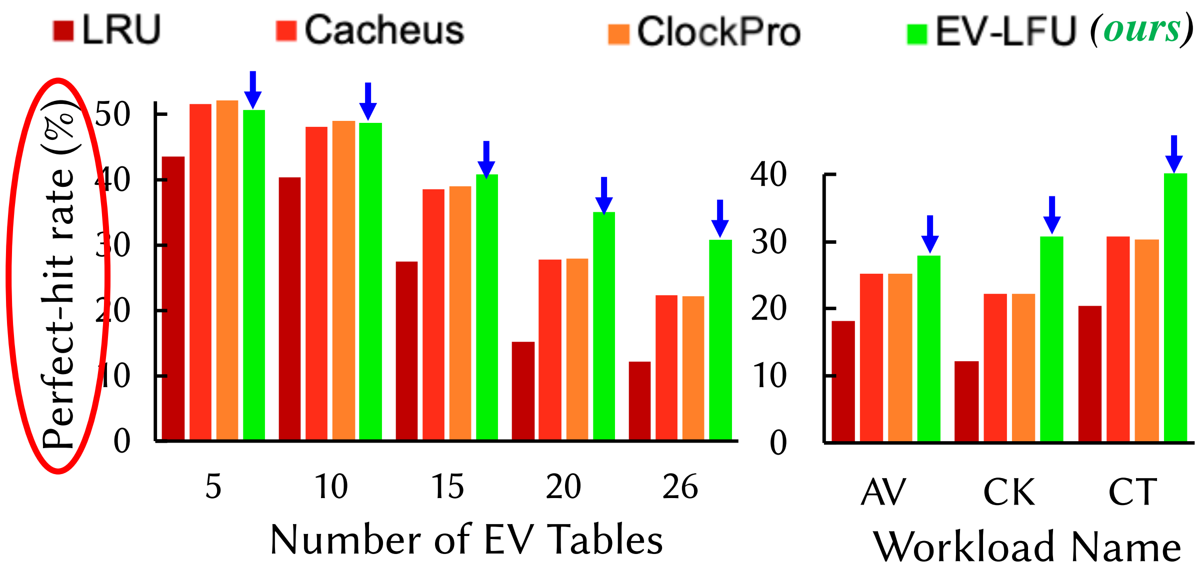
EVCache variants get up to 10% higher perfect hit rate!

Perfect-hit vs Individual-hit



EVCache on Various Workloads

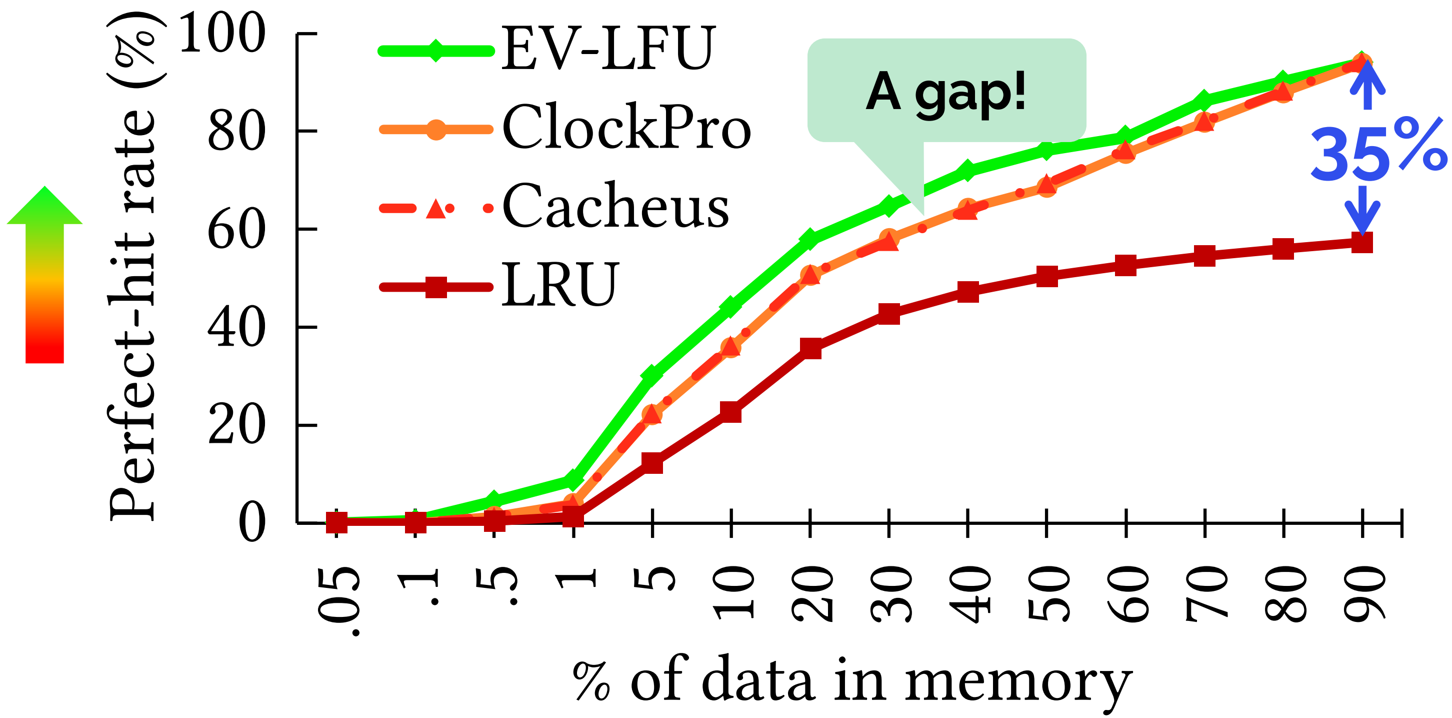




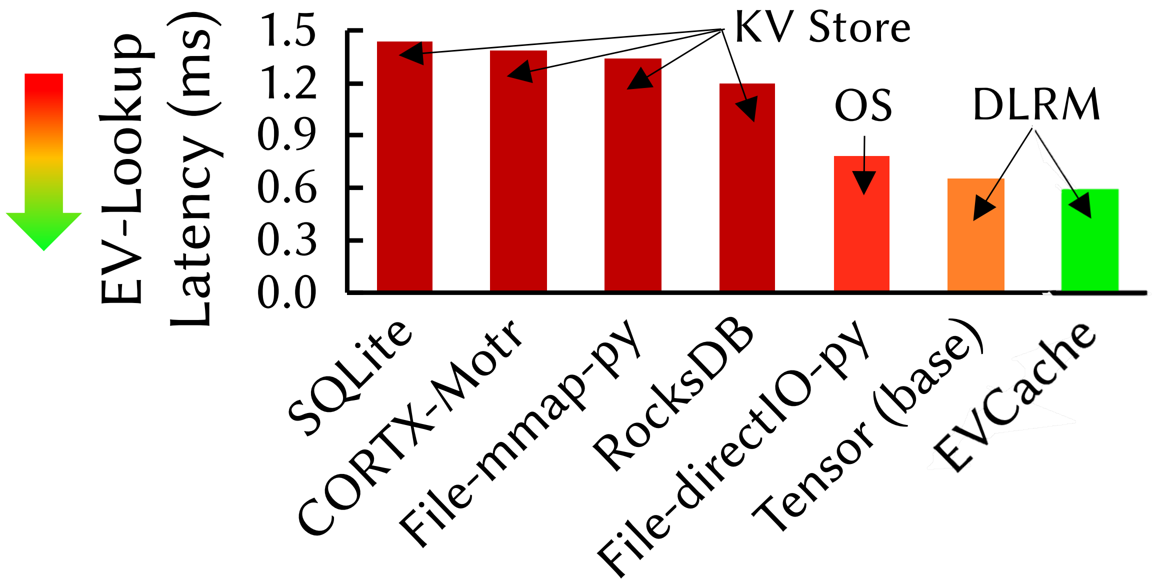
EV-LFU shows steeper benefit as the number of EV tables grows!

EV-LFU outperform others on various datasets!

Perfect-hit rate on various cache sizes



Where to put the caching layer?



External caching layer is **NOT** effective!

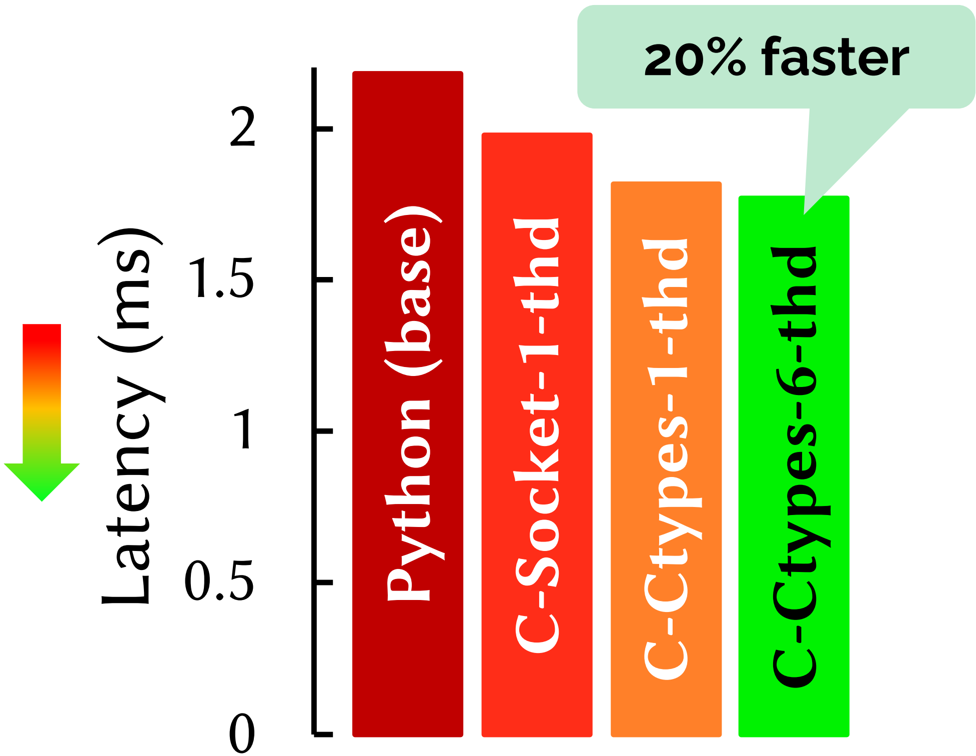
An optimum place to **deploy EVCache** is **in the DLRM!**

How to implement EVMix?

Goal: *Find the most performant Implementation of EVMix layers*

Setup

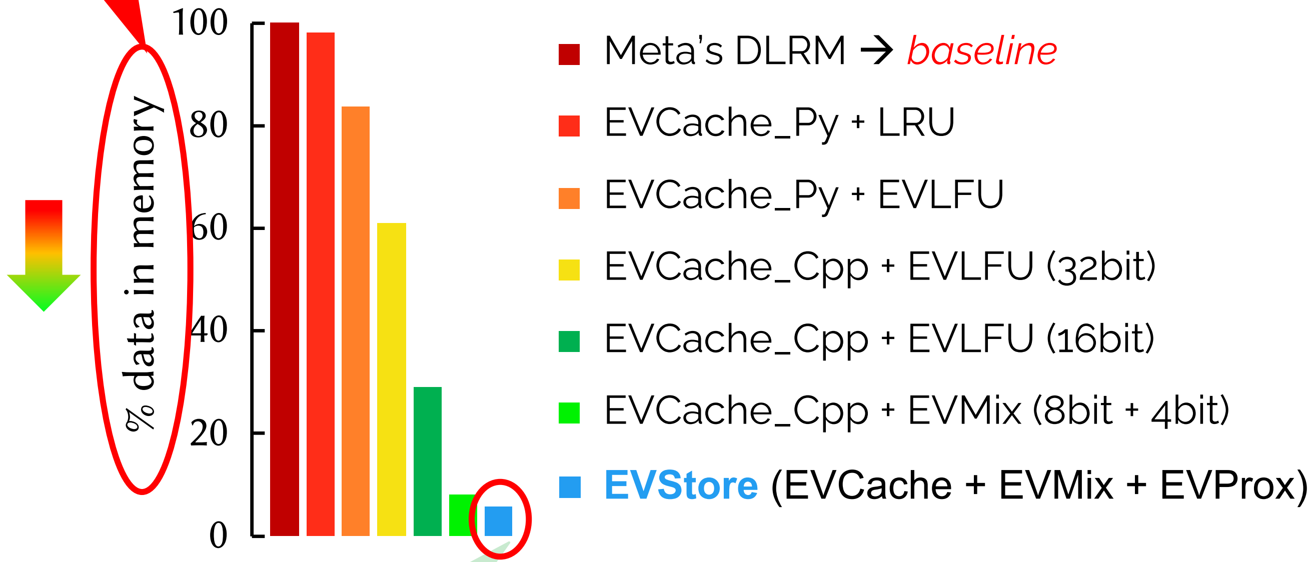
- ***Baseline:*** *Python-based caching layer within the DLRM*
- ***Precision****: 32bit*



EVStore Orthogonal Evaluation

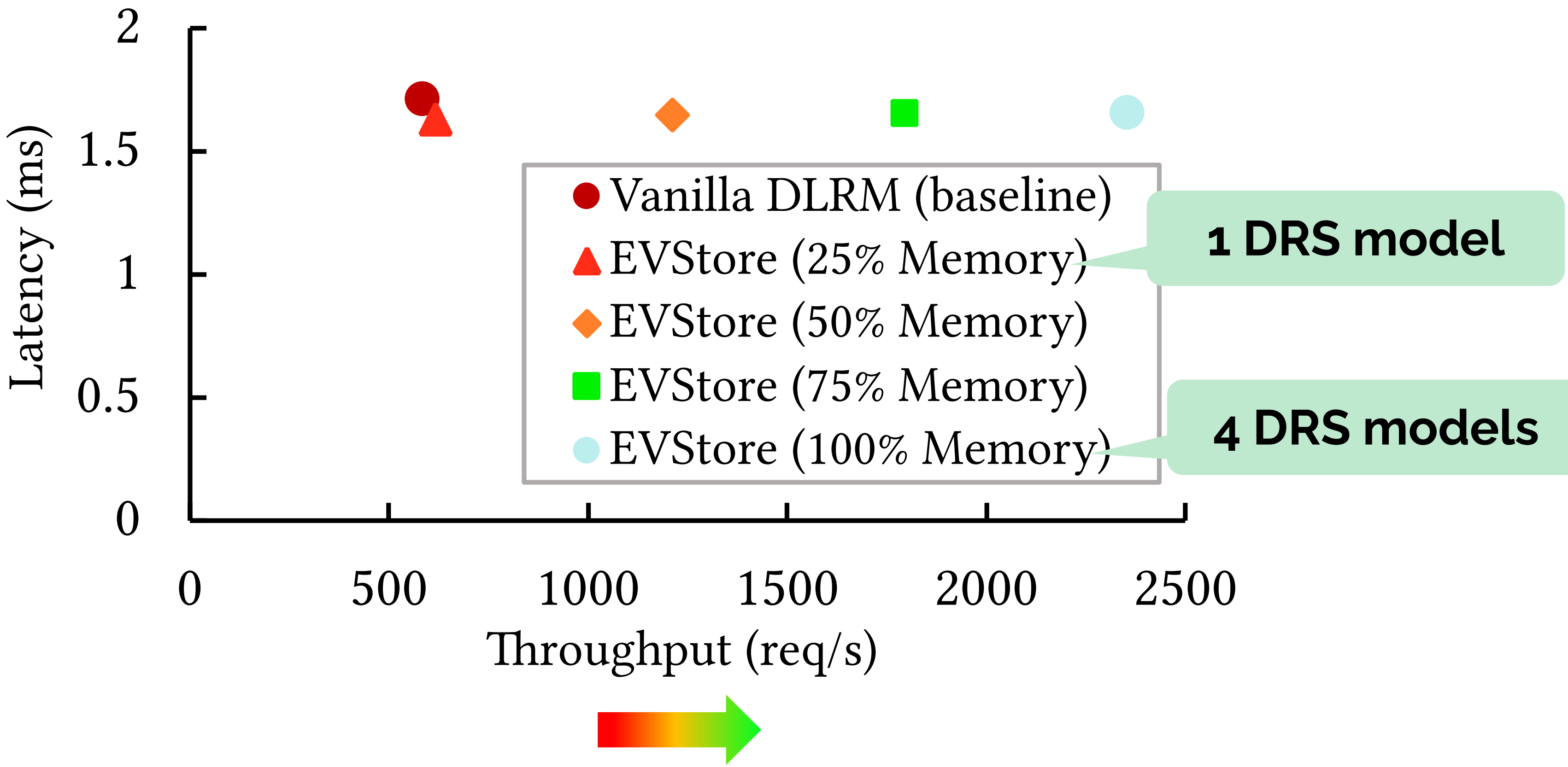
SLA : Inference Latency **< 2 ms**
Accuracy drop **< 0.2%**

Memory Usage!



Fully optimized **EVStore reduces** the **memory usage** by up to **94%**

Collocating Multiple DRS



More in the Paper!

- The importance of perfect hits
- EVCache variants implementation
- EVMix's bit-coding optimization
- More evaluation results
  - Trade-off between latency and accuracy
  - …

EVStore: Storage and Caching Capabilities for Scaling Embedding Tables in Deep Recommendation Systems

Daniar H. Kurniawan
University of Chicago
Chicago, IL, USA
daniar@uchicago.edu

Ruipu Wang
Beijing University of Technology
Beijing, China
rexwoodrp@gmail.com

Kahfi S. Zulkifli
Fandi A. Wiranata
Bandung Institute of Technology
Bandung, Jawa Barat, Indonesia
{sbhnkahfi,fandi.z.w}@gmail.com

John Bent
Seagate Technology
Fremont, CA, USA
john.bent@seagate.com

Ymir Vigfusson
Emory University
Atlanta, GA, USA
ymir@mathcs.emory.edu

Haryadi S. Gunawi
University of Chicago
Chicago, IL, USA
haryadi@cs.uchicago.edu

ABSTRACT

Modern recommendation systems, primarily driven by deep-learning models, depend on fast model inferences to be useful. To tackle the sparsity in the input space, particularly for categorical variables, such inferences are made by storing increasingly large embedding vector (EV) tables in memory. A core challenge is that the inference operation has an all-or-nothing property: each inference requires multiple EV table lookups, but if any memory access is slow, the whole inference request is slow. In our paper, we design, implement and evaluate EVStore, a 3-layer EV table lookup system that harnesses both structural regularity in inference operations and domain-specific approximations to provide optimized caching, yielding up to 23% and 27% reduction on the average and p90 latency while quadrupling throughput at 0.2% loss in accuracy. Finally, we show that at a minor cost of accuracy, EVStore can reduce the Deep Recommendation System (DRS) memory usage by up to 94%, yielding potentially enormous savings for these costly, pervasive systems.

CCS CONCEPTS

• **Information systems → Novelty in information retrieval**; • **Computer systems organization → n-tier architectures**; **Secondary storage organization**; **Pipeline computing**; *Real-time system architecture*; • **Computing methodologies →** *Neural networks.*

KEYWORDS

Recommendation Systems; Deep learning; Caching systems; Inference systems; Performance

ACM ISBN 978-1-4503-9916-6/23/03…$15.00
https://doi.org/10.1145/3575693.3575718

ACM Reference Format:
Daniar H. Kurniawan, Ruipu Wang, Kahfi S. Zulkifli, Fandi A. Wiranata, John Bent, Ymir Vigfusson, and Haryadi S. Gunawi. 2023. EVStore: Storage and Caching Capabilities for Scaling Embedding Tables in Deep Recommendation Systems. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '23), March 25–29, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3575693.3575718

1 INTRODUCTION

Recommendation systems are used prominently across modern online services to help people make decisions. They capture user behavior and preferences to display personalized advertisements [29, 30], rank news [10, 24], and recommend products [69]. The impact of recommendation systems on user engagement is tremendous. Recent studies show that a significant amount of content—30% of all traffic on Amazon's website, 60% of the videos on YouTube, and 75% of the viewed movies on Netflix came from suggestions made by recommendation algorithms [7, 8, 62, 74].

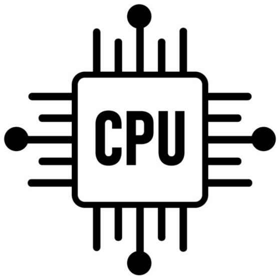
In the age of Deep Learning, Deep Recommendation Systems (DRSs) are widely used to deliver high-quality recommendations [30, 78], but tackling *categorical ("sparse") input features* is their Achilles' heel. Modern DRSs, such as Facebook's post recommendation systems [30], often contain hundreds or thousands of categorical features (*e.g.*, users, posts, or pages), each of which can contain millions or even tens of billions of possible categories. To make the complexity of the deep neural network (DNN) tractable, sparse categorical data is usually converted to ("dense") vectors of numbers before being fed to the model. The most popular conversion is via *embedding vector tables*, or "**EV tables**" for short (§2).

By reducing the DNN complexity, EV tables sacrifice space for faster computation, and thus require significant memory. Consequently, the space management of EV tables becomes challenging: many real-world EV tables contain billions of embedding vectors [31, 69] that require tens of TBs of memory capacity. Such DRAM-heavy architectures account for significant operational costs for DRS users measured in millions of dollars—nearly 80% of all AI-related deployment in Facebook's data centers in 2020 directly supported DRSs [30]. Additionally, industry's insatiable appetite for improved recommendation accuracy is driving the rapid growth of

EVStore Overhead

1. **Memory**
   - No duplicate value
   - The embedding value is stored as it is
   - Storing groupability-score ➡ **Negligible** overhead (<1%)

2. **Computation**
   - **Offline** overhead ➡ mapping surrogate keys
   - **Online** overhead (**negligible**)
     - Converting low precision to full precision
     - Managing the cache eviction
     - Decoding stream of data from C++ to Python side



3. **Storage (SSD)**
   - Storing multiple precisions of the same value ➡ **2x** Overhead

EVStore Summary

Storage and Caching Capabilities for Scaling Embedding Tables in Deep Recommendation System

- **EVCache**: groupability-aware caching
- **EVMix**: mixed-precision caching
- **EVProx**: embedding value approximation

L1 + L2 + L3

94%	23%	4x
Lighter memory	**Faster inference**	**Model collocation**

EVStore code: https://github.com/**ucare-uchicago/ev-store-dlrm**

License GPL v3 Platform x86-64

```
EVStore -- Groupability-aware caching systems for DRS
```

This repository contains the implementation code for paper:

EVSTORE: Storage and Caching Capabilities for Scaling Embedding Tables in Deep Recommendation Systems

Thank you!

I'm on the job market. <daniar@uchicago.edu>