Slipstream: Semantic-Based Training Acceleration for Recommendation Models

Yassaman Ebrahimzadeh Maboud
Electrical and Computer Engineering
The University of British Columbia
Vancouver, Canada
yassaman@ece.ubc.ca

Muhammad Adnan
Electrical and Computer Engineering
The University of British Columbia
Vancouver, Canada
adnan@ece.ubc.ca

Divya Mahajan
Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia
divya.mahajan@gatech.edu

Prashant J. Nair
Electrical and Computer Engineering
The University of British Columbia
Vancouver, Canada
prashantnair@ece.ubc.ca

***Abstract*—Recommendation models play a crucial role in delivering accurate and tailored user experiences. However, training such models poses significant challenges regarding resource utilization and performance. Prior research has proposed an approach that categorizes embeddings into popular and non-popular classes to reduce the training time for recommendation models. We observe that, even among the popular embeddings, certain embeddings undergo rapid training and exhibit minimal subsequent variation, resulting in saturation. Consequently, updates to these embeddings become redundant.**

This paper presents Slipstream, a software framework that identifies stale embeddings on the fly and skips their updates to enhance performance. Our experiments demonstrate Slipstream's ability to maintain accuracy while effectively discarding updates to non-varying embeddings. This capability enables Slipstream to achieve substantial speedup, optimize CPU-GPU bandwidth usage, and eliminate unnecessary memory access. SlipStream showcases training time reductions of 2×, 2.4×, 1.2×, and 1.175× across real-world datasets and configurations, compared to Baseline XDL, Intel-optimized DRLM, FAE, and Hotline, respectively.

***Index Terms*—Recommendation Models, Machine Learning, Training Systems, Memory Systems**

I. Introduction

Recommendation systems are critical in online advertising and e-commerce [1]. These systems combine computationally heavy neural networks with memory-intensive embedding tables. Accessing embedding tables creates significant performance bottlenecks, limiting training throughput. Prior research has explored embedding popularity, compression, and sparsity to enhance efficiency [2], [3]. Our study, however, focuses on leveraging the invariability of large embedding sets.

Deep Learning Recommendation Models (DLRM) use embedding tables to store sparse item and user features [4], [5], while a bottom Multi-Layer Perceptron (MLP) processes dense features. These outputs feed into a feature interaction layer connected to a top MLP for Click-Through Rate (CTR) prediction. Embedding tables can exceed hundreds of gigabytes [5], making DLRM training memory-bound.

In real-world scenarios, embedding access is highly skewed, with certain 'hot embeddings' accessed far more frequently.

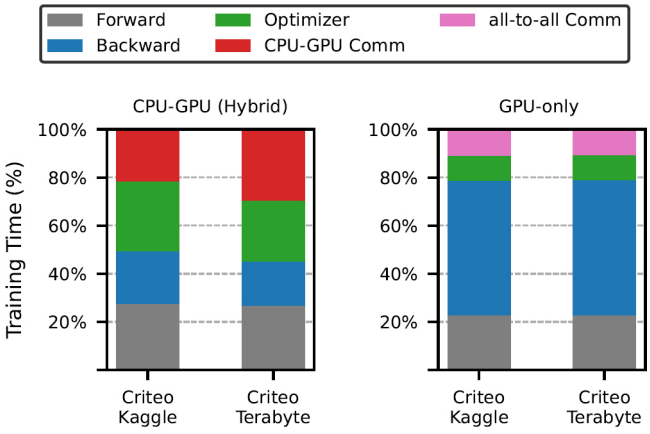

Fig. 1. Training time analysis for Deep Learning Recommendation Models (DLRM) on a single node with four NVIDIA V100 GPUs reveals differing bottlenecks in hybrid CPU-GPU and GPU-only modes. In the hybrid mode, the forward pass, optimizer, and CPU-GPU communication dominate. In the GPU-only mode, the forward pass, backward pass, and inter-GPU communication are the primary bottlenecks. These variations underscore the need for a versatile software framework to optimize DLRM training across configurations.

This skew reduces training time but leads to minimal updates after the embeddings converge, creating unnecessary computations and data movement. We propose Slipstream, a framework that optimizes training by selectively updating embeddings based on data awareness, eliminating redundant computations, and minimizing unnecessary updates.

Slipstream is a co-designed algorithm and software framework that dynamically identifies fully trained embeddings with minimal performance overhead. It then focuses on training only the remaining embeddings without affecting baseline accuracy. To this end, Slipstream comprises three key runtime components, which are instantiated one after another to enhance the training performance and accuracy.

1) **Snapshot Block**: Slipstream employs the 'Snapshot' Block

978-3-9826741-0-0/DATE25/© 2025 EDAA

to identify frequently accessed or 'hot' embeddings that train rapidly. It periodically captures snapshots of these embeddings during runtime to track their training dynamics. Even though embedding tables in real-world datasets can reach hundreds of gigabytes, the size of hot embeddings is relatively small, usually a few hundred megabytes. This allows multiple snapshots to be stored on GPUs without significant memory overhead.

2) **Sampling Block**: Determining when an embedding is fully trained—when its values stabilize within a certain threshold—can be challenging and computationally expensive. Iterating over snapshots with different thresholds can increase overheads by up to 500×. Slipstream addresses this using input sampling to efficiently estimate the embeddings with low variation, reducing overheads by nearly 1000×.

3) **Input Classifier Block**: After identifying stable embeddings, Slipstream improves training efficiency by skipping inputs associated with these embeddings for the remainder of the process, reducing communication and computation overheads.

This is achieved through an input classifier block, which filters inputs to focus on high-variation embeddings. Slipstream enables dynamic, performance-optimized training by leveraging popularity-related semantics in recommender models.

We demonstrate that Slipstream maintains training accuracy while significantly reducing training time. Slipstream, on average, reduces the training time by 2×, 2.4×, 20%, and 18% compared to commercial XDL [6], Intel-Optimized DLRM [7], FAE [2], and Hotline [8] baselines, respectively.

II. Background and Motivation

A. Training Setup

Training DLRMs typically uses a combination of CPUs and GPUs in two distributed modes: hybrid CPU-GPU or GPU-only. In hybrid mode, embeddings are stored on the CPU, while neural networks run on GPUs in a data-parallel manner. All model parameters are stored in GPU memory in GPU-only mode, requiring GPU scaling to handle the model size.

1) Hybrid CPU-GPU Mode: This mode stores embedding tables in CPU memory, allowing larger models without requiring additional GPUs. However, it faces throughput limitations due to significant data transfers and the slower bandwidth of CPU memory. As shown in Figure 1, 70% of the training time is spent on embedding lookups, updates, and the communication of embeddings and gradients over PCIe.

2) GPU-only Mode: In this mode, embeddings are distributed across multiple GPUs (model-parallel) while the neural network remains data-parallel. This requires scaling GPUs to match the size of the embedding tables. It also involves significant all-to-all communication to transfer embeddings and gradients between devices and nodes. Figure 1 shows that in a 4-node, 4-GPU setup, NVLink provides 2400 Gbps for intra-node communication, while Infiniband offers only 100 Gbps for inter-node transfers, making it the bottleneck. Infiniband overheads consume around 50% of total training time.

B. Motivation: Data-Aware Embedding Updates

This paper aims to enable embedding-variation-aware training at runtime, based on three key observations: the high cost of embedding operations, the dynamic frequency of accesses, and the changing update patterns of embeddings over time.

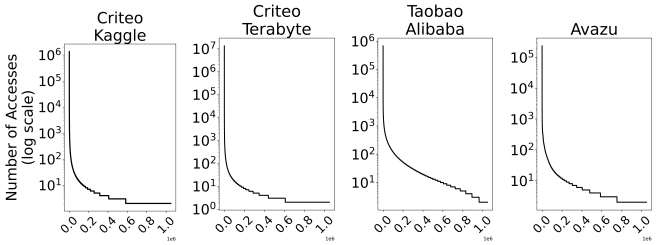

Fig. 2. Access frequency to the largest embedding table during a single training epoch. This skewed access categorizes embeddings into 'hot' and 'cold.' The x-axis shows embedding indices in millions. Typically, 'hot' entries are accessed over 100× compared to 'cold' entries.

1) Breakdown of Training Time: Figure 1 shows the training time distribution for two real-world models and datasets in hybrid CPU-GPU and GPU-only modes. Embedding operations, including lookups, updates, and CPU-GPU communication, can account for up to 75% of training time in hybrid mode. In GPU-only mode, all-to-all communication consumes up to 50% of training time. These observations show the need for optimizations to manage embedding operations efficiently and reduce training time.

2) 'Hot' Embeddings and Skewed Access Patterns: Recommendation models often exhibit skewed access patterns. As shown in Figure 2, a small subset of 'hot' embeddings can be accessed over 100× more frequently than 'cold' embeddings, driven by the popularity of specific users or items.

3) Embedding Value Saturation: Due to their higher access rates, 'hot' embeddings converge faster during training. Figure 3 demonstrates that these embeddings often plateau early in training, with minimal updates after the initial iterations. This suggests that continuing to update these embeddings provides little benefit, making them candidates for optimization.

III. Design: The Slipstream Framework

The Slipstream framework provides two key performance benefits: (1) it eliminates compute and memory operations for low-variability embeddings, and (2) it improves CPU-GPU bandwidth utilization, reducing communication and synchronization overhead. Key notations used in the framework are summarized in Table I.

A. Efficient Snapshots with 'Hot' Embeddings

Slipstream leverages the distinction between 'hot' and 'cold' embeddings based on access frequency to create embedding snapshots. Let N represent the total number of snapshots. The 'hot' embeddings, denoted by $\mathbf{E}_{\text{hot}}$, are much smaller than the full embedding table $\mathbf{E}$. For instance, in the terabyte dataset, $|\mathbf{E}_{\text{hot}}|$ is only 500MB but handles about 75% of total inputs, while the full embedding table is 63GB – over 128× larger.

Slipstream employs a module similar to FAE [2] to identify 'hot' embeddings [1]. The parameter λ defines 'hot' embeddings

[1]Slipstream's objective and approach differ from FAE [2], which lacks a mechanism for skipping stale embeddings.

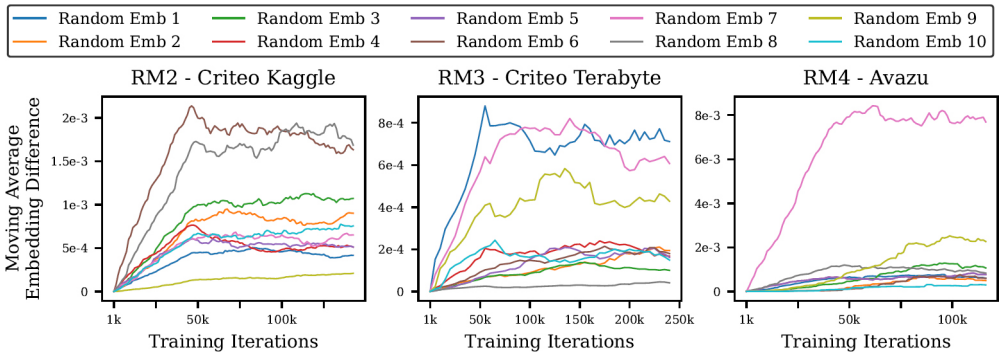

Fig. 3. The temporal difference in values for *ten randomly selected* 'hot' embeddings for RM2 (Criteo Kaggle), RM3 (Criteo Terabyte), and RM4 (Avazu) recommendation models. As 'hot' embeddings account for a significant fraction of accesses, they tend to saturate quickly – in under 25% of the training iterations. This experiment uses DLRM [4] for the training process.

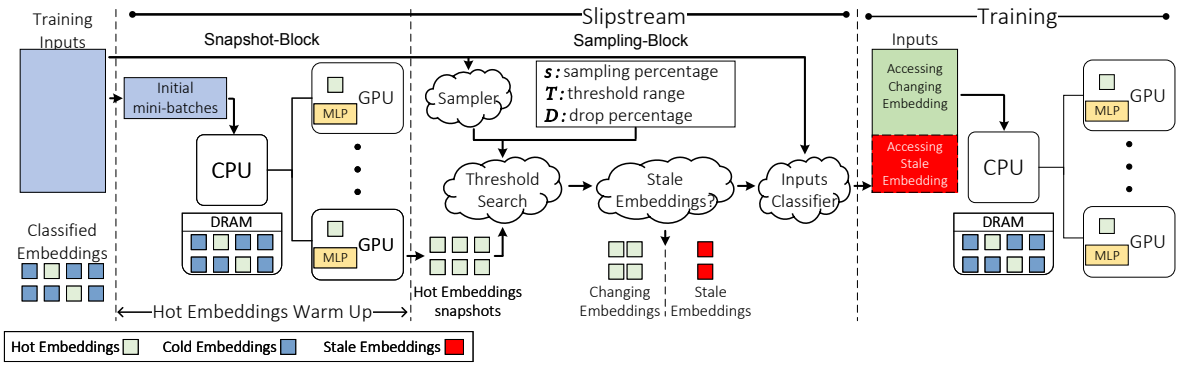

Fig. 4. The flow for Slipstream framework. It consists of three phases. The first phase warms up the embeddings, akin to baseline training. The second phase samples inputs and determines the threshold for identifying stale embeddings. In the third phase, training continues only on the varying embeddings while inputs to stale embeddings are dropped.

TABLE I
TABLE OF NOTATIONS OF *Key* TERMS

Notation	Description
$\mathbf{E}$	All embedding tables
$\mathbf{E}_{\text{hot}}$	Hot embedding table
$\hat{\mathbf{E}}^n_{\text{hot}}$	nth snapshot of hot embedding table
$\mathbf{E}_{\text{stale}}$	Binary set with stale hot embeddings equal to 0
$\hat{\mathbf{e}}^n_{\text{hot}}(i)$	ith entry of nth snapshot of hot embedding table
N	Total number of snapshots
λ	Ratio of accesses for an embedding entry to be hot
T	Threshold to determine stale embeddings
I	Training inputs dataset
I_{hot}	Hot inputs set
I_{cold}	Cold inputs set
s	Sampling percentage from I_{hot}
$\mathcal{S}$	Sampled input set from I_{hot}
D	Input drop-percentage (%) from $\mathcal{S}$
$\bar{D}$	Average input drop-percentage from I_{hot}
α	Number of stale features for an input to be skipped
sd	Standard Deviation
CI	Confidence Interval

as those with access frequency ≥ 1 in every 10,000 accesses when $\lambda = 10^{-5}$. This separates the training dataset I into 'hot' and 'cold' sets: $I = I_{\text{hot}} + I_{\text{cold}}$.

1) Warmup Period: Figure 4 illustrates the Slipstream workflow. The snapshot block is initiated after a *warmup* period, during which most embeddings are accessed at least once. The warmup period, typically 2K iterations (1%-2% of total iterations), is empirically determined to ensure sufficient input coverage. It is adjustable as a hyperparameter. After the warmup, Slipstream uses *low-footprint* temporal profiles in $\hat{\mathbf{E}}^n_{\text{hot}}$ to guide further training.

2) Embedding Snapshots: Slipstream then creates N temporal snapshots, denoted $\hat{\mathbf{E}}^n_{\text{hot}}$, for the hot embedding table. Each snapshot captures embedding values at a specific time, where n ranges from 1 to N. The snapshot size $|\hat{\mathbf{E}}^n_{\text{hot}}|$ is significantly smaller than $|\mathbf{E}|$ and can be adjusted to fit within GPU memory.

B. Challenge: Identifying the Skip Threshold

Slipstream aims to find the threshold T for skipping stale embeddings—those no longer needing updates. This is done by comparing 'hot' embeddings ($\hat{\mathbf{E}}_{\text{hot}}$) in snapshot n with snapshot $n-1$, represented as $\Delta\hat{\mathbf{E}}_{\text{hot}} = \hat{\mathbf{E}}^n_{\text{hot}} - \hat{\mathbf{E}}^{n-1}_{\text{hot}}$.

Figure 5 shows that this process introduces significant overhead, as Slipstream must iterate over all 'hot' embeddings for various T values. For RM2 (Criteo Kaggle), the overhead exceeds 30×; larger datasets like RM3 (Criteo Terabyte) can exceed 500× the baseline training time.

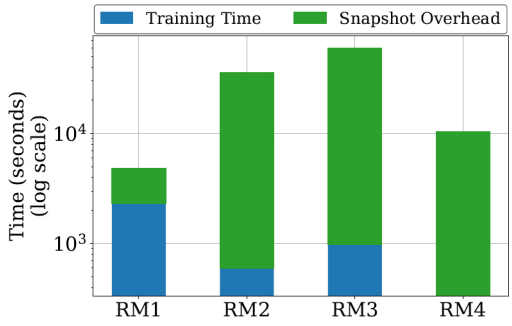

Fig. 5. The overhead for creating temporal snapshots of 'hot' embedding entries after each training minibatch is shown on a logarithmic scale. This overhead can be 2× to 500× higher than the baseline training time. The increase depends on the snapshot instances and the thresholds (T) used for comparison. Even if the 'hot' embedding entries are a small fraction of the total embeddings, iterating over them consumes significant performance overheads.

C. Solution: Sampling for Threshold Finding

1) Insight: Slipstream minimizes the overhead of finding the threshold T by leveraging the insight that a precise T is unnecessary. Instead, it converges on T with a certain confidence level through input sampling. Rather than iterating over the entire embedding table, Slipstream samples a subset of hot inputs I_{hot}, iterating only over the sampled set.

2) Input Sampling Block: The input sampling block selects s inputs from I_{hot}, forming the sampled set $\mathcal{S}$, where $|\mathcal{S}| = s$. These sampled inputs are used for threshold-based comparison of embeddings between snapshots.

The comparison between two snapshots, $\hat{\mathbf{e}}^n_{\text{hot}}$ and $\hat{\mathbf{e}}^{n-1}_{\text{hot}}$, is defined by threshold T, yielding a binary set $\mathbf{E}_{\text{stale}}$ indicating stale (0) or varying (1) embeddings:

$$\mathbf{E}_{\text{stale}} = \delta(\hat{\mathbf{e}}^n_{\text{hot}}, \hat{\mathbf{e}}^{n-1}_{\text{hot}}, T) = \begin{cases} 1, & \text{if } \|\hat{\mathbf{e}}^n_{\text{hot}} - \hat{\mathbf{e}}^{n-1}_{\text{hot}}\| > T \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Here, $\|\cdot\|$ is a distance metric (Slipstream uses Euclidean), and $\delta(\cdot)$ determines if the embeddings have changed significantly (1) or not (0) based on T.

3) Dropping Inputs to Stale Embeddings: By iterating over the sampled inputs $\mathcal{S}$, Slipstream calculates the *fraction of inputs* that can be dropped using threshold T, denoted as D in Equation 2.

$$D = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \delta(s, T) \quad (2)$$

The *average number* of dropped hot inputs, $\bar{D}$, is $\bar{D} = D \times I_{\text{hot}}$. The standard deviation sd of dropped inputs is calculated using Equation 3.

$$sd = \sqrt{\frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \left(\delta(s, T) - \bar{D}\right)^2} \quad (3)$$

4) Confidence in Estimating Dropped Inputs: We use the "Student's t-interval" to establish the confidence interval (CI) for $\bar{D}$, shown in Equation 4.

$$CI_{100\times(1-\alpha)} = \bar{D} \pm t_{\frac{\alpha}{2}} \times \sqrt{\left(\frac{I - |\mathcal{S}|}{I}\right) \times \left(\frac{sd^2}{m}\right)} \quad (4)$$

Here, $t_{\frac{\alpha}{2}}$ is the critical value of the t-distribution, with m input samples. For 99.9% confidence, $t_{\frac{\alpha}{2}} \approx 3.340$.

5) Rapidly Finding Threshold: Slipstream employs a binary search to determine the optimal threshold T, using only the sampled inputs $\mathcal{S}$. This reduces computational overheads significantly, as shown in Figure 6, with savings up to 1000× compared to processing all 'hot' embeddings.

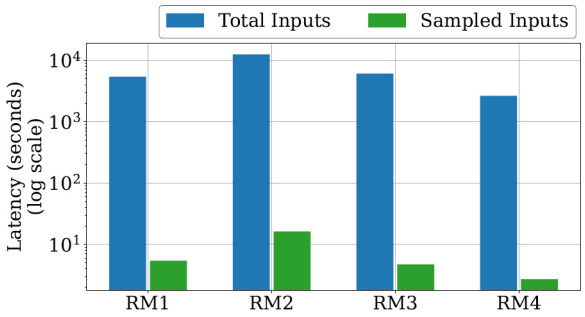

Fig. 6. The latency savings due to sampling for four real-world datasets. We observe at least a 1000× lower latency as the sampling block selects 0.1% of hot inputs (I_{hot}).

D. Input Classifier Block

The input classifier block separates inputs accessing varying embeddings from those accessing stale embeddings. For each input $i \in I_{\text{hot}}$, the embedding vectors from consecutive snapshots are compared. Let $\hat{\mathbf{E}}^n_{\text{hot}}(i)$ represent the embedding matrix for input i at the n-th snapshot. If the Euclidean distance between snapshots n and $n-1$ exceeds threshold T, the embedding $\mathbf{E}_{\text{hot}}(i)$ is marked as varying, as shown in Equation 5.

$$\left\|\mathbf{E}^n_{\text{hot}}(i) - \mathbf{E}^{n-1}_{\text{hot}}(i)\right\|_2 > T \Rightarrow \mathbf{E}_{\text{hot}}(i) \text{ is varying} \quad (5)$$

Inputs are then classified into two subsets based on the marked embeddings, with α determining the number of stale features to skip. Inputs accessing varying embeddings ($I^{\text{vary}}_{\text{hot}}$) are defined in Equation 6.

$$I^{\text{vary}}_{\text{hot}}(\alpha) = \{i(\alpha) \in I_{\text{hot}} \mid \mathbf{E}_{\text{hot}}(i) \text{ is varying}\} \quad (6)$$

Inputs accessing stale embeddings ($I^{\text{stale}}_{\text{hot}}$) are defined in Equation 7.

$$I^{\text{stale}}_{\text{hot}}(\alpha) = \{i(\alpha) \in I_{\text{hot}} \mid \mathbf{E}_{\text{hot}}(i) \text{ is stale}\} \quad (7)$$

$$I_{\text{hot}} = I^{\text{vary}}_{\text{hot}} + \cancel{I^{\text{stale}}_{\text{hot}}} \quad (8)$$

The input classifier block incurs a one-time overhead by classifying inputs in a single pass. In subsequent training, only inputs in $I^{\text{vary}}_{\text{hot}}$ are used, as they access embedding entries with significant variations.

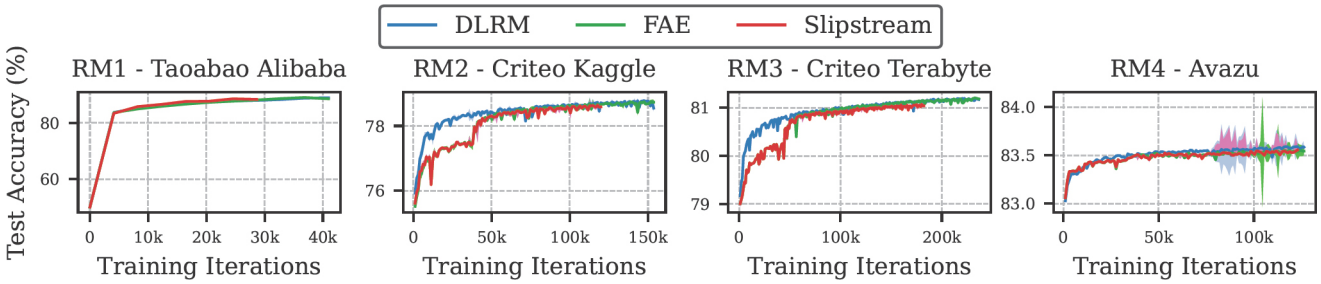

Fig. 7. Testing Accuracy across four real-world datasets and commonly used recommender models across multiple runs. The testing accuracy is measured across training iterations. Notably, Slipstream consistently follows the accuracy curve of DLRM and FAE baselines. Overall, Slipstream incurs a maximum accuracy loss of 0.04%.

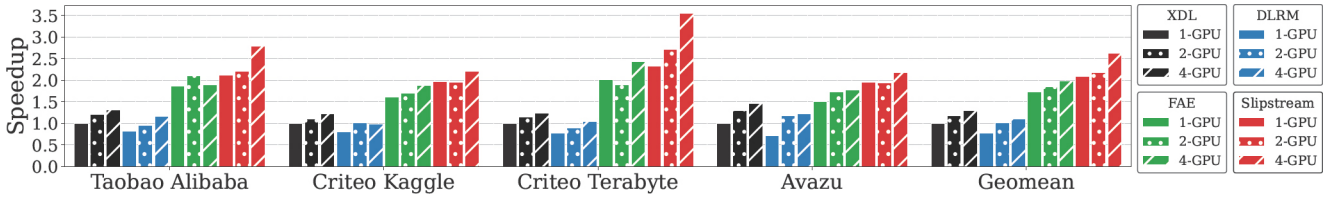

Fig. 8. The performance of Slipstream compared to the prior state-of-the-art for Criteo Kaggle, Taobao Alibaba, Criteo Terabyte, and Avazu datasets. All values are normalized to XDL 1-GPU. On average, Slipstream provides 2.5× training time speedup compared to an aggressive XDL baseline. It also consistently maintains a high speedup across all datasets and models.

IV. Evaluation

A. Experimental Setup and Models

We tested four recommendation models (RM1, RM2, RM3, RM4) on publicly available datasets to evaluate Slipstream's performance. These models vary in size and complexity, with dense features ranging from 1 to 13 and sparse features from 3 to 26. We used an open-source Deep Learning Recommendation Model (DLRM) [4] and a Time-Based Sequence Model (TBSM) [9] for training.

The hyperparameter λ, borrowed from the FAE framework [2], determines the size of hot embeddings on the GPU. Slipstream sets $\lambda = 10^{-6}$ for RM1 and $\lambda = 10^{-7}$ for RM2, RM3, and RM4, balancing the size of hot embeddings and the percentage of hot inputs. For instance, $\lambda = 10^{-6}$ classifies embeddings with more than 1 in a million accesses as hot.

1) Datasets: We evaluate Slipstream using four public datasets: Taobao [10] (user behavior from Taobao), Criteo Kaggle [11] (CTR prediction for display ads), Criteo Terabyte [12] (large-scale click logs used in MLPerf), and Avazu [13] (CTR prediction competition dataset).

2) Baselines: We compare Slipstream against four baselines: (1) XDL [6], (2) DLRM and TBSM [4], [9], (3) FAE [2], and (4) Hotline [8]. All hybrid baselines run in CPU-GPU hybrid mode, with embeddings on the CPU and neural networks on the GPU. We also conduct GPU-only experiments to showcase Slipstream's advantages.

3) Hardware and Software Setup: DLRM and TBSM are configured with PyTorch-1.8.1 [14] and Python-3.8, using NCCL [15] for GPU-GPU communication over NVLink [16]. XDL-1.0 [6] is run with TensorFlow-1.2 [17].

TABLE II
Training System Specifications

Device	Architecture	Memory	Storage
CPU	Intel Xeon Silver 4116 (2.1GHz)	192 GB DDR4 (76.8GB/s)	1.9 TB NVMe SSD
GPU	Nvidia Tesla V100 (1.2GHz)	16 GB HBM-2.0 (900GB/s)	-

Table II details our evaluation hardware, where GPUs communicate via NVLink-2.0 and a 16x PCIe Gen3 bus, similar to the FAE [2] and Hotline [8] setups.

B. Results and Analysis

1) Accuracy: Figure 7 presents the testing accuracy of Slipstream compared to the DLRM and FAE baselines. The baselines achieve accuracies of 78.5%, 89%, 81%, and 83.5% for the Criteo Kaggle, Taobao, Criteo Terabyte, and Avazu datasets, respectively. Slipstream matches these accuracies with a negligible loss of up to 0.04%, caused by occasional misidentification of stale embeddings due to its sampling-based approach. These results show that skipping inputs for saturated embeddings maintains robust accuracy while delivering significant performance improvements. Slipstream initially shows an accuracy boost for the Criteo datasets but converges with the baseline accuracy over time, preserving training fidelity.

2) Performance Improvement: Slipstream improves performance by skipping non-contributing inputs during training. Figure 8 shows a consistent 2.5× speedup in training time compared to the aggressive XDL baseline across datasets and models. On 4-GPUs, weak scaling experiments show that Slipstream's speedup increases with the number of GPUs. However, the benefits compared to FAE for the sequence-based Taobao dataset are less pronounced due to added overhead.

Overall, Slipstream reduces bandwidth overheads in collective operations like scatter and gather, enhancing training efficiency.

3) Evaluation Metrics: Table III summarizes the numeric evaluation metrics, including Area Under Curve (AUC), Testing Accuracy, and Binary Cross Entropy (BCE) loss. Slipstream maintains these metrics compared to the baseline DLRM, demonstrating that its optimizations enhance training efficiency.

TABLE III
EVALUATION METRICS: SLIPSTREAM VERSUS DLRM

Metric	RM2 - Criteo Kaggle		RM3 - Criteo Terabyte		RM4 - Avazu	
	DLRM	Slipstream	DLRM	Slipstream	DLRM	Slipstream
AUC	0.80	0.797	0.791	0.787	0.764	0.757
Accuracy (%)	78.63	78.59	81.16	81.13	83.59	83.56
BCE Loss	0.456	0.458	0.421	0.422	0.387	0.389

C. Scaling Mini-Batch Size

Figure 9 shows Slipstream's speedup relative to the 4-GPU XDL baseline across the Kaggle, Taobao Alibaba, Criteo Terabyte, and Avazu datasets. Time-based models like Taobao Alibaba scale better due to a smaller dataset with more 'hot' embeddings, which fit on the GPU and train faster. This results in higher speedup gains for Taobao Alibaba.

Slipstream's scalability comes from reducing communication overhead. As the number of GPUs increases, collective communication (e.g., all-to-all) outweighs weak scaling benefits. Larger mini-batches also demand more bandwidth, impacting CPU-GPU communication.

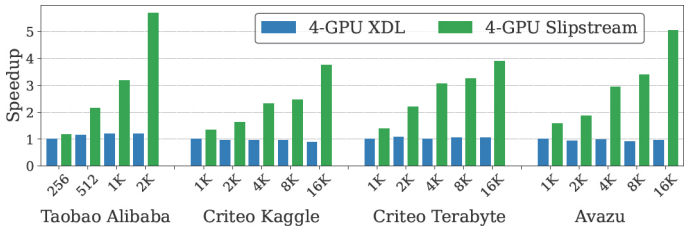

Fig. 9. Mini-batch size scaling for 4-GPU execution. Slipstream showcases a minimum speedup of 3.8× considering all models and datasets as we increase the mini-batch size.

D. Online Training

We ran Slipstream in an online scenario by dividing the training dataset into two segments: *segment 1* (initial days) and *segment 2* (final days), with consistent test data from the last day. This was tested on the Criteo Kaggle and Terabyte datasets, though it can be extended to others. As shown in Figure 10, Slipstream accelerates online training by up to 2.1×.

V. RELATED WORK

1. Optimized Embedding Placement: Prior work addresses embedding placement to reduce memory footprint and improve performance. FAE [2] places frequently accessed embeddings on GPUs and the rest in CPU memory. Other works [5], [18] explore strategies in heterogeneous systems.

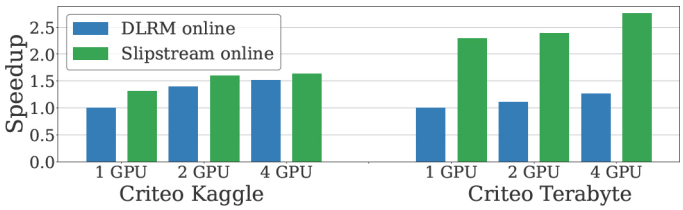

Fig. 10. Training performance for two largest datasets, Kaggle and Terabyte, for online training.

2. Efficient Recommender Model Training: Efforts like [19] increase throughput through near-memory processing and spatial/temporal reuse. Skewed access patterns have been used to redesign embedding structures [20], [21], develop compression [22], and accelerate training [3], [23], [24]. Hotline [8] uses an FPGA-based accelerator for pipelining inputs.

3. Machine Learning Accelerators: Several accelerators [8], [25], [26] focus on deep learning compute or collaborative filtering models. Slipstream speeds up training without specialized hardware but could benefit from these accelerators.

4. Feature Interactions for Improved Convergence: Methods like DLRM [4], DeepFM [27], and DCN [28] enhance feature interactions. Slipstream complements these by skipping stale embeddings and optimizing training efficiency without changing model architecture.

5. Early Stopping for DNNs: Early stopping has been explored in image recognition and NLP [29]–[31]. Slipstream introduces a new approach, skipping stale inputs to apply early stopping at a granular level, a technique not previously explored in recommender systems.

6. Mitigating Communication Overhead: All-to-all GPU communication limits DLRM training. Prior work [32] proposes prefetching based on minibatch patterns, while [33] uses a unified cache for hot embeddings. Slipstream accelerates training by detecting and discarding saturated embeddings.

VI. CONCLUSIONS

The growing size of recommendation models has increased communication and computational overhead during training. This paper introduces Slipstream, a runtime framework that skips fully trained, stale embeddings with low variability based on the intuition that they no longer contribute to model quality. Our experiments show Slipstream reduces training time by 2×, 2.4×, 1.2×, and 1.175× compared to XDL, Intel-Optimized DLRM, FAE, and Hotline baselines, respectively.

ACKNOWLEDGEMENTS

This project is part of the *STAR Lab* at The University of British Columbia (UBC). We thank the Advanced Research Computing Center team at UBC [34]. We also thank the anonymous reviewers from DATE 2025 for their invaluable feedback. This project was supported by Intel Transformation Server Architecture (TSA) and the Natural Sciences and Engineering Research Council of Canada (NSERC) [funding reference number RGPIN-2019-05059] Grants.

REFERENCES

[1] C. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, "Machine learning at facebook: Understanding inference at the edge," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2019, pp. 331–344.

[2] M. Adnan, Y. E. Maboud, D. Mahajan, and P. J. Nair, "Accelerating recommendation system training by leveraging popular choices," *Proc. VLDB Endow.*, vol. 15, no. 1, p. 127–140, sep 2021. [Online]. Available: https://doi.org/10.14778/3485450.3485462

[3] G. Sethi, B. Acun, N. Agarwal, C. Kozyrakis, C. Trippel, and C.-J. Wu, "Recshard: Statistical feature-based memory optimization for industry-scale neural recommendation," 2022.

[4] M. Naumov, D. Mudigere, H. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy, "Deep Learning Recommendation Model for Personalization and Recommendation Systems," *CoRR*, vol. abs/1906.00091, 2019.

[5] W. Zhao, D. Xie, R. Jia, Y. Qian, R. Ding, M. Sun, and P. Li, "Distributed hierarchical gpu parameter server for massive scale deep learning ads systems," 2020.

[6] B. Jiang, C. Deng, H. Yi, Z. Hu, G. Zhou, Y. Zheng, S. Huang, X. Guo, D. Wang, Y. Song, L. Zhao, Z. Wang, P. Sun, Y. Zhang, D. Zhang, J. Li, J. Xu, X. Zhu, and K. Gai, "XDL: An Industrial Deep Learning Framework for High-Dimensional Sparse Data," ser. DLP-KDD '19. New York, NY, USA: Association for Computing Machinery, 2019.

[7] D. Kalamkar, E. Georganas, S. Srinivasan, J. Chen, M. Shiryaev, and A. Heinecke, "Optimizing Deep Learning Recommender Systems Training on CPU Cluster Architectures," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20. IEEE Press, 2020.

[8] M. Adnan, Y. E. Maboud, D. Mahajan, and P. J. Nair, "Heterogeneous acceleration pipeline for recommendation system training," ser. ISCA '24. IEEE Press, 2024. [Online]. Available: https://doi.org/10.1109/ISCA59077.2024.00081

[9] T. Ishkhanov, M. Naumov, X. Chen, Y. Zhu, Y. Zhong, A. G. Azzolini, C. Sun, F. Jiang, A. Malevich, and L. Xiong, "Time-based Sequence Model for Personalization and Recommendation Systems," *CoRR*, vol. abs/2008.11922, 2020.

[10] Alibaba. User Behavior Data from Taobao for Recommendation. https://tianchi.aliyun.com/dataset/dataDetail?dataId=649userId=1.

[11] CriteoLabs. Criteo Display Ad Challenge. https://www.kaggle.com/c/criteo-display-ad-challenge.

[12] ——. Terabyte Click Logs. https://labs.criteo.com/2013/12/download-terabyte-click-logs.

[13] Kaggle. Avazu mobile ads CTR. https://www.kaggle.com/c/avazu-ctr-prediction.

[14] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," 2017.

[15] Nvidia, "NVIDIA Collective Communications Library (NCCL)," https://docs.nvidia.com/deeplearning/nccl/index.html.

[16] ——. Nvlink. https://www.nvidia.com/en-us/data-center/nvlink/.

[17] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," 2015. [Online]. Available: http://download.tensorflow.org/paper/whitepaper2015.pdf

[18] B. Acun, M. Murphy, X. Wang, J. Nie, C.-J. Wu, and K. Hazelwood, "Understanding Training Efficiency of Deep Learning Recommendation Models at Scale," 2020.

[19] L. Ke, U. Gupta, B. Y. Cho, D. Brooks, V. Chandra, U. Diril, A. Firoozshahian, K. Hazelwood, B. Jia, H. S. Lee, M. Li, B. Maher, D. Mudigere, M. Naumov, M. Schatz, M. Smelyanskiy, X. Wang, B. Reagen, C. Wu, M. Hempstead, and X. Zhang, "RecNMP: Accelerating Personalized Recommendation with Near-Memory Processing," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 790–803.

[20] A. Ginart, M. Naumov, D. Mudigere, J. Yang, and J. Zou, "Mixed dimension embeddings with application to memory-efficient recommendation systems," *CoRR*, vol. abs/1909.11810, 2019. [Online]. Available: https://arxiv.org/abs/1909.11810

[21] H.-J. M. Shi, D. Mudigere, M. Naumov, and J. Yang, *Compositional Embeddings Using Complementary Partitions for Memory-Efficient Recommendation Systems*. New York, NY, USA: Association for Computing Machinery, 2020, p. 165–175.

[22] C. Yin, B. Acun, X. Liu, and C.-J. Wu, "TT-Rec: Tensor Train Compression for Deep Learning Recommendation Models," 2021.

[23] S. Agarwal, Z. Zhang, and S. Venkataraman, "Bagpipe: Accelerating deep recommendation model training," 2022. [Online]. Available: https://arxiv.org/abs/2202.12429

[24] D. Mudigere, Y. Hao, J. Huang, Z. Jia, A. Tulloch, S. Sridharan, X. Liu, M. Ozdal, J. Nie, J. Park, L. Luo, J. A. Yang, L. Gao, D. Ivchenko, A. Basant, Y. Hu, J. Yang, E. K. Ardestani, X. Wang, R. Komuravelli, C.-H. Chu, S. Yilmaz, H. Li, J. Qian, Z. Feng, Y. Ma, J. Yang, E. Wen, H. Li, L. Yang, C. Sun, W. Zhao, D. Melts, K. Dhulipala, K. Kishore, T. Graf, A. Eisenman, K. K. Matam, A. Gangidi, G. J. Chen, M. Krishnan, A. Nayak, K. Nair, B. Muthiah, M. khorashadi, P. Bhattacharya, P. Lapukhov, M. Naumov, A. Mathews, L. Qiao, M. Smelyanskiy, B. Jia, and V. Rao, "Software-hardware co-design for fast and scalable training of deep learning recommendation models," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 993–1011. [Online]. Available: https://doi.org/10.1145/3470496.3533727

[25] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1–12.

[26] J. Park, H. Sharma, D. Mahajan, J. K. Kim, P. Olds, and Hadi Esmaeilzadeh, "Scale-Out Acceleration for Machine Learnng," Oct. 2017.

[27] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: a factorization-machine based neural network for ctr prediction," *arXiv preprint arXiv:1703.04247*, 2017.

[28] R. Wang, R. Shivanna, D. Cheng, S. Jain, D. Lin, L. Hong, and E. Chi, "Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems," in *Proceedings of the web conference 2021*, 2021, pp. 1785–1797.

[29] R. Heckel and F. F. Yilmaz, "Early stopping in deep networks: Double descent and how to eliminate it," *arXiv preprint arXiv:2007.10099*, 2020.

[30] Y. Bai, E. Yang, B. Han, Y. Yang, J. Li, Y. Mao, G. Niu, and T. Liu, "Understanding and improving early stopping for learning with noisy labels," *Advances in Neural Information Processing Systems*, vol. 34, pp. 24 392–24 403, 2021.

[31] Z. Ji, J. Li, and M. Telgarsky, "Early-stopped neural networks are consistent," *Advances in Neural Information Processing Systems*, vol. 34, pp. 1805–1817, 2021.

[32] K. Balasubramanian, A. Alshabanah, J. D. Choe, and M. Annavaram, "cdlrm: Look ahead caching for scalable training of recommendation models," in *Proceedings of the 15th ACM Conference on Recommender Systems*, ser. RecSys '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 263–272.

[33] X. Song, Y. Zhang, R. Chen, and H. Chen, "Ugache: A unified gpu cache for embedding-based deep learning," in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 627–641. [Online]. Available: https://doi.org/10.1145/3600006.3613169

[34] "UBC Advanced Research Computing, "UBC ARC Sockeye." UBC Advanced Research Computing, 2019, doi: 10.14288/SOCKEYE."