SpiderCache: Semantic-Aware Caching Strategy for DNN Training

Zesong Wang
Wuhan National Laboratory for Optoelectronics
Huazhong University of Science and Technology
Wuhan, China
zs_wang@hust.edu.cn

Peng Fang*
Wuhan National Laboratory for Optoelectronics
Huazhong University of Science and Technology
Wuhan, China
fangpeng@hust.edu.cn

Fang Wang
Wuhan National Laboratory for Optoelectronics
Huazhong University of Science and Technology
Wuhan, China
wangfang@hust.edu.cn

Hong Jiang
Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, USA
hong.jiang@uta.edu

Yimin Lu
Wuhan National Laboratory for Optoelectronics
Huazhong University of Science and Technology
Wuhan, China
lymic@hust.edu.cn

Zhan Shi
Wuhan National Laboratory for Optoelectronics
Huazhong University of Science and Technology
Wuhan, China
zshi@hust.edu.cn

Dan Feng
Wuhan National Laboratory for Optoelectronics
Huazhong University of Science and Technology
Wuhan, China
dfeng@hust.edu.cn

Abstract

Deep neural network (DNN) training is both data-intensive and compute-intensive. As datasets grow, storing them entirely in memory becomes infeasible, making I/O a major bottleneck—often accounting for 30%-90% of total training time due to the widening gap between data loading and computation speed. While caching can mitigate I/O latency, traditional strategies fail under random sampling. Recent work shows that importance sampling can induce data locality, enabling more effective caching; however, loss-based importance ignores semantic attributes, limiting effectiveness in I/O-bound tasks.

We propose **SpiderCache**, a semantic-aware caching strategy for DNN training. It employs a graph-based importance sampling algorithm that captures semantic relationships to compute global sample importance, coupled with a dynamic cache manager that adapts to training stages and user needs. Experiments show SpiderCache boosts cache hit ratio by up to 8.5× (avg. 4.15×) and speeds up training by up to 2.33× (avg. 2.21×), while achieving the best accuracy and adaptability across diverse workloads.

CCS Concepts

• **Information systems → Data layout and storage**.

Keywords

I/O bottleneck, semantic-aware caching

*Corresponding Author

ACM Reference Format:
Zesong Wang, Peng Fang, Fang Wang, Hong Jiang, Yimin Lu, Zhan Shi, and Dan Feng. 2025. SpiderCache: Semantic-Aware Caching Strategy for DNN Training. In *54th International Conference on Parallel Processing (ICPP '25), September 08–11, 2025, San Diego, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3754598.3754672

1 Introduction

Deep neural networks (DNNs) have revolutionized various domains by mimicking the human brain's neural structure to process complex data patterns. They are key in image recognition [25][17], autonomous vehicles [5], facial recognition [36], machine translation [40], chatbot interfaces [12], and healthcare [13].

DNN training involves frequent data retrieval from I/O systems followed by intensive computation to update model parameters. Prior work has focused on improving computational efficiency via GPU memory optimization [19, 33], faster data communication [16], and compiler-level operator tuning [6, 34]. However, as datasets grow from terabytes [11, 31] to petabytes [1]—exceeding memory capacity—I/O becomes the main bottleneck. While accelerators like GPUs, FPGAs, and ASICs evolve rapidly, I/O optimization lags behind, severely limiting overall training efficiency [7, 22, 30]. In modern HPC-AI centers, datasets are often cloud-stored and remotely accessed by VMs [7, 10], further aggravating I/O delays. For image classification, I/O can account for 30%–90% of total training time [7, 22].

Caching is a common strategy to reduce I/O costs by storing frequently used items in memory. However, conventional policies like LRU and LFU are ineffective during DNN training, where random sampling disrupts data locality and hinders caching efficiency [46][30]. To address this, CoorDL [30] introduces MinIO, which prevents cache data replacement during training to improve the cache hit ratio.

Importance sampling (IS) is a technique that has been proposed to accelerate DNN training by dynamically adjusting the sampling probabilities based on each sample's contribution to the learning process, prioritizing those samples that have the most significant impact on model performance [20] [21]. Recent studies, such as SHADE [22] and iCache [7], show that IS induces data locality, benefiting caching, and have explored adapting caching policies based on IS to address the I/O problem.

However, commonly used IS algorithms are designed for computation-bound tasks. While reducing GPU processing time by omitting backpropagation for specific samples, these algorithms are ineffective in I/O-bound scenarios [7]. We have conducted an insightful analysis of this phenomenon and found that this ineffectiveness arises because these algorithms calculate importance based on loss, which isn't comparable across epochs and batches. Effective caching policies require comparing the global, not just the local or instantaneous importance of samples.

To address the above problem, we propose **SpiderCache**, a semantic-aware caching strategy for DNN training. Here, semantic-aware refers to understanding the intrinsic meaning of samples via their embeddings. Based on this, we first introduce a graph-based IS algorithm that calculates global importance scores by using the graph structure to analyze the samples' embeddings. Then, SpiderCache applies a semantic-aware caching mechanism that combines importance scores and sample similarity, using a two-layer structure to boost hit ratio. Moreover, as different cache sections impact training efficiency and accuracy differently, we develop an elastic cache manager that dynamically adjusts section ratios during training and accommodates user-specific performance goals.

Figure 1 illustrates SpiderCache's design objectives across three metrics. Compared to prior work, SpiderCache achieves superior performance across all metrics (Section 6). Inspired by its use of graph structures to capture global sample importance and its sensitivity to dynamic changes—akin to how a spider web responds to its environment—we name our approach **SpiderCache**.

In summary, this paper makes the following contributions.

- SpiderCache is the first approach to leverage the semantic information of samples for cache management in DNN training. Through a comprehensive analysis of common IS algorithms, we propose a novel graph-based IS algorithm tailored for I/O-bound scenarios.
- We design a semantic-aware cache mechanism, which uses both importance scores and sample similarity, featuring a two-layer structure designed to improve the total hit ratio.
- We developed an elastic cache manager that can not only adjust the ratio between cache sections in real time but also meet user-specific requirements for various performance metrics.
- Extensive evaluation results demonstrate that SpiderCache increases the cache hit ratio by up to 8.5× (avg. 4.15×) and speeds up training by up to 2.33× (avg. 2.21×) compared to the baseline, while also offering superior accuracy and elasticity across various workloads.

2 Background

2.1 I/O Characteristics in DNN Training

DNN training involves three stages—Data Loading, Preprocessing, and Computation (Figure 2). Each epoch processes the entire dataset in mini-batches. Data Loading fetches these from storage; Preprocessing handles decoding and collation; Computation performs forward and backward passes on accelerators like GPUs.

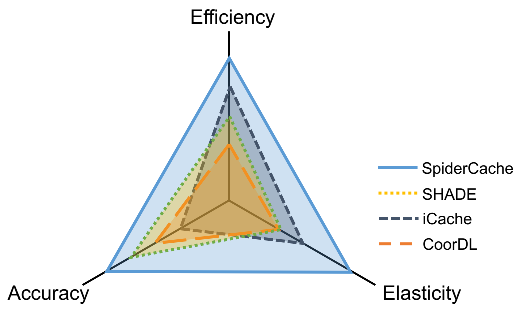

Figure 1: The Design Objective of SpiderCache

DNN training is both data- and computation-intensive [8], requiring substantial storage and compute resources. Data Loading and Computation dominate training time, while Preprocessing is typically lightweight [7]. DNN training datasets have grown from terabytes [31][11] to petabytes [1], far exceeding typical memory and some persistent storage. In recent years, accelerators such as GPUs, FPGAs, and ASICs have continued to advance rapidly in capability and computing speed. However, Data Loading has not kept pace in speed, causing accelerators like GPUs to remain in an idle waiting state. The I/O is becoming a critical performance bottleneck and significantly decreases overall training efficiency[22][30][7].

To reduce costs, deep learning often uses low-cost GPU Spot VMs [26][2], which are prone to termination and may cause data loss on local SSDs. Thus, datasets are usually stored in persistent cloud storage, accessed remotely during training [22][7]. This setup, common in HPC-AI centers [7][10], is widely adopted in recent studies [22][7], worsening the I/O bottleneck.

To verify this, we trained four widely used DNN models on a server with four A40 GPUs and SSD-based remote storage. As shown in Figure 3(a), Data Loading and Computation together account for over 95% of total training time. Notably, Data Loading alone consistently takes up more than 60%, marking it as a major performance bottleneck.

Caching is a common approach to mitigate I/O bottlenecks. However, DNN training often uses random sampling [46], which disrupts temporal and spatial locality [30], reducing caching efficiency [22][7]. Our analysis of LRU and LFU during ResNet18 training (Figure 3(b)), using cache sizes as a percentage of the dataset, confirms their poor performance in this context.

2.2 Importance Sampling

Importance Sampling (IS) is a widely used technique in statistics and machine learning to improve the efficiency of training by focusing computational resources on more important samples.

In DNN training, IS algorithms can prioritize samples based on metrics such as gradient magnitudes[21], model loss[20][27], or training specific model[44]. One common approach involves dynamically adjusting the sampling frequency based on the model's feedback. By focusing on more important samples—whether they

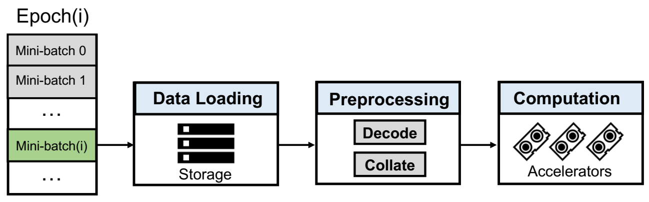

Figure 2: The Pipeline of DNN Training

are difficult to classify or have higher gradients—these algorithms optimize the use of computational resources and reduce training time.

Figure 4 depicts an image classification task where samples from groups (a), (b), and (c) are easily recognized and accurately classified by the model after a few training sessions. Conversely, samples from group (d) present a significant challenge and require more extensive training, being considered important samples.

Under importance sampling (IS), some samples are sampled more frequently than others, offering an opportunity for cache optimization. We compare the sample frequency across different epochs when training DNNs with importance sampling[20] and default sampling. As shown in Figure 5, with default sampling, each item is accessed once per epoch. In contrast, under IS, the sampling frequency of each item varies according to its importance score. Furthermore, the importance of samples varies across different epochs.

Recent works such as SHADE[22] and iCache[7] have observed this phenomenon and have begun to explore adapting cache policies based on IS to mitigate I/O issues.

3 Motivation

Motivation 1: Global importance score are needed.

Existing importance sampling algorithms [20, 21, 44] are designed for computation-bound tasks [7]. These methods reduce training time by decreasing computation, such as skipping backpropagation for certain samples to accelerate GPU processing. As a result, they only assess sample importance locally and instantaneously within the current batch to decide whether to skip training for those samples.

The most common approach is loss-based importance sampling (IS), where a sample's importance score is determined by the loss function, a performance metric typically calculated after a forward pass. The lower the loss, the better the samples are learned. Recent works, such as SHADE[22] and iCache[7], have adapted caching policies using loss-based IS to address the I/O problem.

In I/O-bound scenarios, effective cache policies aim to retain the most important samples in the cache. This requires a global comparison of sample importance across the entire dataset. As training progresses, importance sampling does not update every sample's score in each epoch, so global cache management needs to compare importance scores across broader time windows. However, the loss-based importance sampling (IS) algorithm fails to fully meet these requirements. It calculates an importance score that is typically only comparative within the current batch. Additionally, as shown in Figure 6(a), the variability of losses over time makes importance scores incomparable across broader training periods.

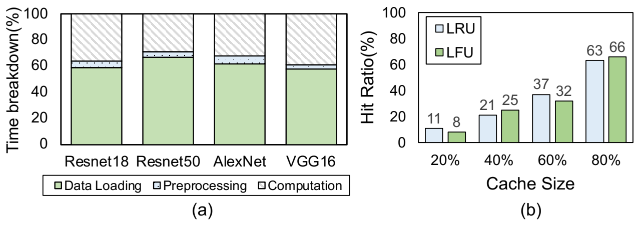

Figure 3: I/O Characteristics in DNN Training

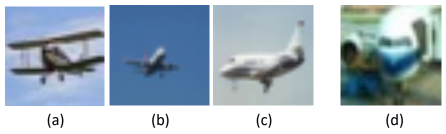

Figure 4: Classification of Airplane in Dataset CIFAR10

Motivation 2: The relationship between samples should be explored.

Current methods overlook sample relationships, resulting in suboptimal cache strategies. SHADE[22] caches samples solely based on their importance score. iCache[7] divides samples into H-samples (important) and L-samples (non-important). For H-samples, it caches by importance score; for L-samples, it randomly replaces items in the cache to improve hit rates. While this boosts the hit ratio, it significantly degrades the model's final accuracy (Figure 6(b)).

Neither SHADE[22] nor iCache[7] accounts for the relationships between samples, which leads to a suboptimal caching policy and potentially decreases the model's accuracy. This motivates the identification of these relationships to enable more rational replacements, thereby enhancing training efficiency and model performance. Therefore, the cache region should be divided into two sections: one for a caching strategy based on importance scores, and the other for one that considers sample relationships.

Motivation 3: Cache partition should be made elastic.

Both SHADE[22] and iCache[7] overlook the evolving distribution of importance scores during training. To analyze this, we tracked the standard deviation (std) of score changes throughout the training process.

Figure 6(c) shows the std of importance scores across four model training configurations first increases and then decreases, reflecting that the variance rises early in training and later converges. As the variance shrinks, importance scores become more uniform across samples, importance sampling assigns similar probabilities to most samples, thus lowering the proportion of "important" ones. In Motivation 2, we propose dividing the cache region into two sections. As the portion of important samples changes over time, the ratio between different cache sections should be adjusted dynamically.

4 Design

The SpiderCache design comprises three components: **Graph-based IS Algorithm**, **Semantic-aware Cache Mechanism**, and **Elastic Cache Manager**, addressing the three key challenges. Figure 7 shows the workflow. During Data Loading, data is first retrieved from the **Semantic-aware Cache**; on a miss, it's fetched from remote storage. In Data Processing, forward propagation computes embeddings and loss; loss is used for backpropagation, while embeddings feed into the **Graph-based IS Algorithm**, which uses

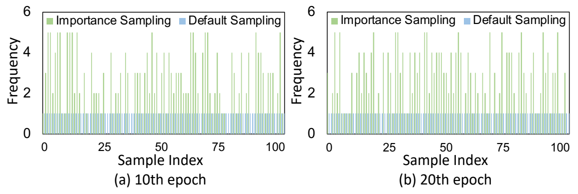

Figure 5: Sample Frequency on Different Epochs

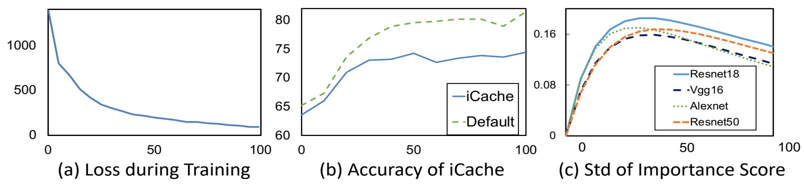

Figure 6: Training Observations over 100 Epochs

a graph to evaluate sample importance and update high-degree nodes into the homophily cache. Meanwhile, the **Elastic Cache Manager** in the Data Analyzing stage monitors model accuracy and score distribution to adjust cache space allocation dynamically.

4.1 Graph-based Importance Score Algorithm

Loss-based IS algorithms often capture only local, instantaneous importance, ignoring the global view. Inspired by the success of graph structures in modeling global relationships across domains [15][3], we propose a graph-based IS method that evaluates sample importance from a global perspective.

We construct a graph where each sample is a node. While loss indicates current learning status, it lacks intrinsic or semantic meaning. To capture this, we use embeddings from the feature extraction layer of the DNN, which provides feature-rich, high-level representations effective for tasks like clustering, similarity measurement, and transfer learning [4][29].

For instance, DNN training aims to cluster same-class embeddings closely and separate those of different classes. As illustrated in Figure 8, colors denote sample classes, showing intra-class clustering and inter-class separation. To capture global embedding relationships efficiently, we adopt the Approximate Nearest Neighbor (ANN) algorithm [18], which enables fast, accurate neighbor search in large datasets. Specifically, we use the HNSW library [28] for its fast index construction and support for dynamic sample updates.

With HNSW, we can efficiently compute the Euclidean distance between samples, where n represents the dimension of the embedding. As shown in Equation 1, the distance is calculated as:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{1}$$

Based on this distance, the similarity score, ranging from 0 to 1, is computed using an exponential decay function, as shown below:

$$sim\ (x, y) = e^{-\lambda * d(x,y)} \tag{2}$$

The similarity decreases as the distance increases and vice versa. This relationship is controlled by the hyperparameter λ, which adjusts the decay rate. When the similarity between two samples exceeds a threshold α, an edge is established between them. This relationship is defined as:

$$edge(x, y) = \begin{cases} 1 & if\ \ score(sim(x, y)) > \alpha \\ 0 & otherwise \end{cases} \tag{3}$$

The analysis is conducted on the entire dataset. For each node x, two values are computed: x_{same}, representing the number of neighboring nodes with the same class as x, and x_{other}, representing the number of neighboring nodes with a different class. The importance score of the sample is then defined as:

$$score(x) = \ln\left(\frac{1}{x_{same}} + \frac{x_{other}}{neigbormax} + 1\right) \tag{4}$$

The *neigbormax* is usually set to 500 in the HNSW default setting. Intuitively, $Part1$: $\frac{1}{x_{same}}$ emphasizing the rarity of intra-class neighbors, and $Part2$: $\frac{x_{other}}{neigbormax}$, highlighting the presence of inter-class neighbors. The logarithm function is applied to smooth the distribution of importance scores.

As depicted in Figure 8(b), We categorize trained samples into three states:

- **State 1: Well-classified samples.** High x_{same}, low x_{other}. Both $Part1$ and $Part2$ scores of Equation 4 are low, resulting in the lowest overall score.
- **State 2: Boundary samples.** High x_{same}, high x_{other}. $Part1$ score is low, and $Part2$ has a moderate score, resulting in a medium overall score.
- **State 2: Isolated samples.** Low x_{same},low x_{other}. $Part1$ score is high, and $Part2$ is low, resulting in a medium overall score.
- **State 3: Misclassified samples.** Low x_{same}, high x_{other}. Both $Part1$ and $Part2$ scores are high, yielding the highest overall score.

After calculation, we update the global importance scores corresponding to the processed samples, storing the node with the highest x_{same} with its neighbor lists for future cache updates. We then perform importance sampling based on these scores, using the biased sampling method torch.multinomial from PyTorch. Figure 10 (Line 16-23) exhibits the detailed process.

4.2 Semantic-aware Cache Mechanism

Previous approaches often result in suboptimal cache policies by ignoring relationships among samples (see Section 3). Moreover, DNN training datasets frequently contain many duplicate or highly similar samples [43], which generally have similar effects on model accuracy [37, 41]. Identifying and replacing them with similar counterparts in the cache can significantly reduce I/O overhead.

To address these issues, the graph structure (Section 4.1) captures sample relationships by connecting highly similar samples via edges (Equation 3). Moreover, high-degree nodes, which are often connected to many other nodes in the graph, generally represent a higher level of similarity to other nodes. By caching these high-degree nodes, we can replace a sample with its adjacent high-degree node during training, thereby reducing I/O overhead.

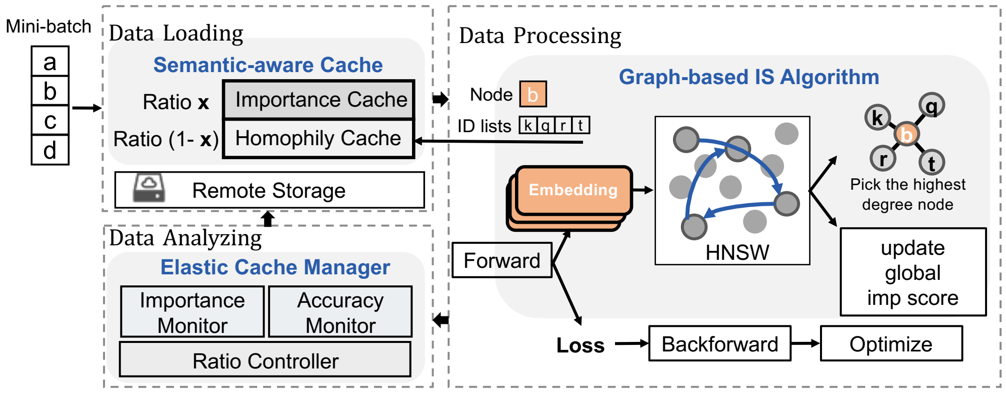

Figure 7: Framework of SpiderCache

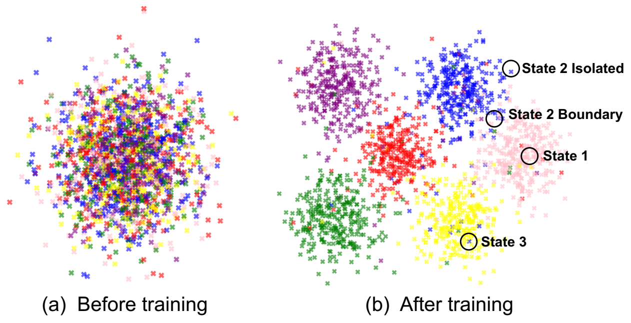

Figure 8: Embeddings in DNN Training

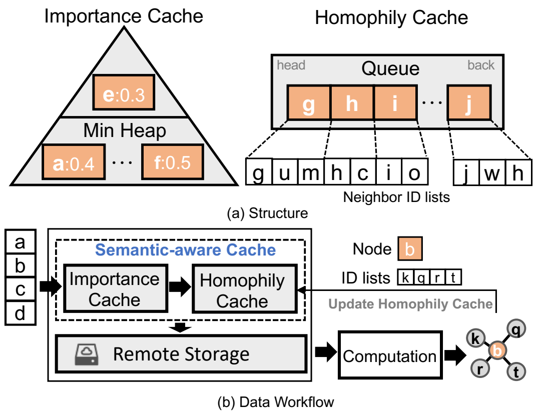

Figure 9: Semantic-aware Cache Mechanism

Inspired by this insight, we propose a Semantic-aware Cache Mechanism, as illustrated in Figure 9(a), which includes both an importance-driven cache (Importance Cache) and a homophily-replaceable cache (Homophily Cache).

1) Importance Cache: Eviction and prefetching are driven by sample importance scores. A min-heap manages the cache, evicting the least important samples when full. The Importance Cache prioritizes retaining critical samples identified by our graph-based IS algorithm.

2) Homophily Cache: Homophily in graph analysis refers to the tendency of nodes to connect with similar others [14]. The Homophily Cache stores high-degree nodes along with the IDs of their neighbors, which are listed in a neighbor ID list. If an incoming sample matches a neighbor, the corresponding high-degree node is fetched from the cache. This cache uses a FIFO (First-In-First-Out) update strategy, which ensures that all samples are regularly replaced, thereby fostering greater diversity in the training data.

The two caches are exclusive, with no data exchange or sharing between them. Figure 9(b) illustrates the data workflow of the Semantic-aware Cache. Samples are first searched in the Importance Cache; if not found, the Homophily Cache is queried. If both miss, the data is fetched from remote storage. After computation, the highest-degree node in the batch is used to update the Homophily Cache. Combined with Figure 9, there are four cases to illustrate how this cache mechanism work:

Case 1: Sample *a* (importance 0.4) hits the Importance Cache and is directly fetched.

Case 2: Sample *b* (importance 0.2) misses both the Importance Cache and the Homophily Cache. It is then fetched from remote storage. Since its score is lower than *e*'s (0.3) at the top of the Min Heap structure, no update happens to the Importance Cache.

Case 3: Sample *c* (importance 0.5) misses the Importance Cache but matches a neighbor in the Homophily Cache. Its similar node *h* from the Homophily Cache is fetched as a replacement.

Case 4: Sample *d* (importance 0.6) misses both caches and is fetched from remote storage. As its importance exceeds *e*'s (0.3), *e* is evicted and *d* is inserted into the Importance Cache.

The Importance Cache is updated only when a sample misses both caches and is fetched from remote storage. The Homophily Cache is updated after processing a batch. When samples *a*, *b*, *c*, and *d* are processed, the highest-degree node *b*, which was not previously in the Homophily Cache, is selected. *g* is evicted from the Homophily Cache, and *b* with its neighbor ID list is inserted. Details of the Semantic-aware Cache Mechanism are provided in Figure 10 (Line 4-13).

4.3 Elastic Cache Manager

Section 4.2 splits the cache into two: the Importance Cache, which stores samples by importance, and the Homophily Cache, which replaces samples with high-degree nodes. A fraction x of the cache is allocated to the Importance Cache, denoted as *imp-ratio*, leaving $(1 - x)$ for the Homophily Cache. Increasing *imp-ratio* helps preserve accuracy, while decreasing it may slightly reduce accuracy

Algorithm 1 SpiderCache: A Semantic-aware Caching Strategy for DNN Training

```
for each epoch in epochs do
    for each batch in batches do
        // 4.2 Semantic-aware Cache Mechanism
        for each index in batch do
            if importance_cache.contains(index) then        ▷ Check if the sample is in the Importance Cache
                item ⟵ importance_cache.get(index)
            else if neighbor_list.contains(index) then        ▷ Check if the sample is in the neighbor list of the Homophily Cache
                neighbor_index ⟵ neighbor_to_key(index), item ⟵ homophily_cache.get(neighbor_index)
            else
                item ⟵ remote_storage.get(index), importance_cache.update(item)        ▷ Fetch from remote storage
            end if
        end for
        (loss, embeddings) ⟵ Forward(items)
        // 4.1 Graph-based IS Algorithm
        ANN_index.update(indices, embeddings)        ▷ Update ANN with new embeddings
        for each (index, item) in zip(indices, items) do
            (score, neighbor_list) ⟵ compute_score(index), imp_score[index] ⟵ score        ▷ Update global imp_score
            if length(neighbor_list) > length(max_neighbor_list) then        ▷ Pick the highest degree node
                max_node, max_index, max_neighbor_list ⟵ item, index, neighbor_list
            end if
        end for
        homophily_cache.update_cache(max_node, max_neighbor_list)        ▷ Update Semantic Cache
        //4.3 Elastic Cache Manager
        imp_ratio ⟵ Control_ratio(imp_score, model_accuracy)        ▷ Adjust cache strategy based on performance
    end for
end for
```

Figure 10: Algorithm 1

but improve cache hit rates. This is because the Importance Cache retains original samples, whereas the Homophily Cache substitutes them with similar ones (see Section 6.5).

As shown in Figure 6(c), the standard deviation of importance scores first increases and then decreases, indicating that sample importance diverges early in training and later converges. This trend suggests a declining proportion of important samples (x) over time. As importance scores stabilize, the Importance Cache becomes less effective, and allocating more space to the Homophily Cache can improve cache hit rates.

To maintain model accuracy while maximizing the cache hit rate, we propose an Elastic Cache Manager that monitors training in real time and dynamically adjusts cache space allocation. In particular, it consists of three components Importance Monitor, Accuracy Monitor, and Ratio Controller.

1) Importance Monitor: This component is designed to monitor the slope of the standard deviation (σ) of importance scores. When the slope turns negative, indicating a decrease in the portion of important samples, an activation factor(β) is triggered. The formula to represent this operation is:

$$\beta = \begin{cases} 1 & if \ \frac{d\sigma}{dt} < 0 \\ 0 & otherwise \end{cases} \tag{5}$$

2) Accuracy Monitor: Given a series of accuracy measurements $\{a_t\}$, and considering the fluctuations in accuracy, we employ the Savitzky-Golay filter[35] to obtain the smoothed accuracy $\{\tilde{a}_t\}$. We then use the smoothed accuracy $\{\tilde{a}_t\}$ to calculate the latest average accuracy growth rate Δ_t:

$$\Delta_t = \frac{1}{m} \sum_{i=0}^{m-1} (\tilde{a}_{t-m+i+1} - \tilde{a}_{t-m+i}) \tag{6}$$

m denotes the size of the window, which is typically set to 5 in our experiments. The penalty factor u is introduced to regulate the rate of change in the important cache ratio. It is defined as:

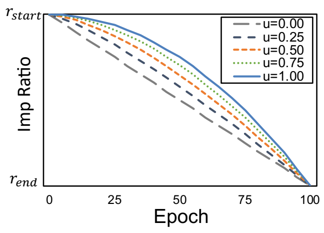

Figure 11: Ratio Change

$$u = \frac{\Delta_t}{\gamma + \Delta_t} \tag{7}$$

γ is a balancing factor that controls the impact of Δ_t on u. When Δ_t is large (indicating rapid growth), $u \longrightarrow 1$; when Δ_t is small (indicating growth stabilizes), $u \longrightarrow 0$.

3) Ratio Controller: Based on the activation factor β and penalty factor u, the update rule for the important cache ratio $imp\text{-}ratio(t)$ is:

$$imp\_ratio(t) = r_{start} - \beta \left(r_{start} - r_{end} \right) \left(\frac{t}{T} \right)^{1+u} \tag{8}$$

r_{start} and r_{end} represent the initial and final values of the *imp-ratio*. When the activation factor α is 0, the *imp-ratio* remains at r_{start}. As β reaches 1, the *imp-ratio* begins to change. t denotes the current training time, and T is the total training duration. The term $1 + u$ controls the rate at which the cache ratio changes over time.

Figure 11 exhibits the process of the *imp-ratio* changing from r_{start} to r_{end}. As u transitions from 1 to 0, the adjustment trend shifts from the upper line to the bottom. During the rapid growth phase of accuracy ($u \longrightarrow 1$), the adjustment slows down to maintain the model's accuracy. In the stabilized growth phase ($u \longrightarrow 0$), the adjustment speeds up, gradually decreasing the portion of the Importance Cache and increasing the Homophily Cache, improving the total cache hit ratio.

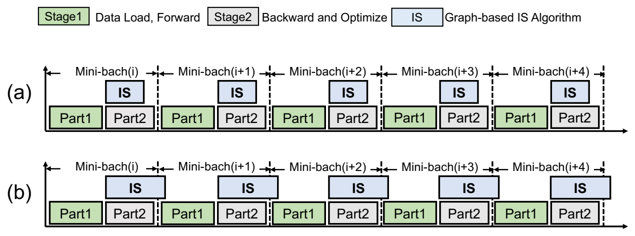

Figure 12: Pipeline of SpiderCache's DNNs Training

After extensive experimentation (discussed in Section 6.5), we recommend setting r_{start} and r_{end} to 90% and 80%, respectively. Additionally, We observe that the choice of ratio has varying impacts on different model metrics. To accommodate this, we have made it an adjustable parameter, enabling users to customize it based on their specific training requirements.

5 Overhead Analysis and Mitigation

SpiderCache is implemented in PyTorch2.7.0 by modifying DataLoader and Sampler, and uses Redis for in-memory caching, following SHADE[22]. This design enables easy integration into existing projects.

Considering the computational overhead introduced by our Graph-Based Importance Sampling (IS) Algorithm, we next analyze the time and space overheads and propose strategies to alleviate these overheads. Specifically, we segment the deep neural network (DNN) training process into three stages: Data Loader and Forward Pass (Stage1), Backward Pass and Optimization (Stage2), and IS computation using the Graph-Based IS Algorithm (IS, or Stage3), to better understand where these overheads occur.

In DNN training, Stage2 follows Stage1, while IS requires Stage1's embeddings as input. IS computation uses HNSW, whose runtime is mainly influenced by embedding dimension, not index size. Table 1 shows average per-mini-batch execution times across models and stages.

Table 1: Time Consumption Analysis.

Model	Stage1	Stage2	IS
Resnet18	42ms	35ms	16ms
Resnet50	48ms	37ms	18ms
AlexNet	62ms	33ms	35ms
Vgg16	56ms	28ms	31ms

To minimize computational overhead, we employ a pipelined parallelization strategy. Figure 12 shows the pipeline settings in SpiderCache. For most models like ResNet18, ResNet50, MobileNetV2, and Inception-v3, which require relatively shorter IS computation times, the IS stage overlaps with Part2, as shown in Figure 12(a). We also test models like AlexNet and VGG16, which have largest embedding dimensions among commonly used DNN models and thus longer IS computation times, we allow the IS stage to overlap with Part2 and the Part1 time of the next mini-batch, as shown in Figure 12(b).

This pipelined approach perfectly hides the time overhead of Graph-based IS computation without impacting the model accuracy and cache hit rate. This is because the Graph-based IS computation only updates the importance scores for the current batch, and any slight delay in updates does not affect the global computation.

During the execution of the Graph-based IS Algorithm, we leverage a graph structure for importance analysis; however, this graph is transient and not stored persistently. Specifically, to compute the global importance of each node, we only evaluate its degree (i.e., number of neighbors), which suffices to estimate the importance score. Neighbor IDs are temporarily stored only for the top-degree nodes within each mini-batch, while for all other samples, the graph structure is discarded immediately after scoring. Consequently, the main storage overhead comes from the Approximate Nearest Neighbor (ANN) index, rather than the graph itself.

To handle large-scale datasets efficiently, we adopt the HNSW algorithm in conjunction with quantization (Product Quantization) to minimize storage and ensure fast retrieval. Notably, such techniques are widely adopted by leading technology companies (e.g., Google, Meta, Amazon) in large-scale retrieval systems to optimize both storage and latency[39][9]. For instance, in the case of ImageNet-1K, which comprises 1.2 million images (138 GB of raw data), the HNSW-based ANN index requires only 134 MB, achieving a compression ratio of over 1000×. This trend generalizes across datasets, as shown in Table 2, where massive datasets such as LAION-400M and YFCC100M can also be indexed efficiently with storage budgets under tens of gigabytes.

Table 2: Storage efficiency of HNSW-based ANN indexing on various datasets.

Dataset	Image Count	Raw Size	Index Size	Compression Ratio
ImageNet-1K	~1.2M	~138 GB	~134 MB	~1029×
Open Images (V6)	~9M	~600 GB	~965 MB	~622×
ImageNet-21K	~14M	~1.3 TB	~1.5 GB	~870×
YFCC100M	~100M	~100 TB	~11.2 GB	~8928×
LAION-400M	~400M	~240 TB	~44.8 GB	~5357×
LAION-5B	~5B	~2.5 PB	~560 GB	~4464×

6 Evaluation

6.1 Experimental Setup

Environment. The experiments are conducted on four servers, each equipped with an NVIDIA V100 32GB GPU, and one server equipped with four NVIDIA A40 48GB GPUs specifically used to evaluate multi-GPU performance (detailed in Section 6.6). All servers are connected via a 10 Gbps Ethernet network. The implementation is compatible with Pytorch2.7.0, with CUDA 11.8 and cuDNN 8.0 for GPU acceleration. The training datasets are stored in an NFS file system located within the same datacenter, consistent with SHADE[22].

Datasets. We conducted experiments on three widely-used DNN training datasets: ImageNet [11], with 1.2 million images (138 GB) across 1,000 classes, a leading dataset in image recognition; CIFAR-10 [24], consisting of 50,000 images across 10 classes; and CIFAR-100 [24], also with 50,000 images but spanning 100 classes for finer classification.

Models. We evaluated our approach using four widely-adopted architectures: ResNet-18 [17], a compact 18-layer model, fast and resource-efficient; ResNet-50 [17], with 50 layers, offering deeper feature extraction for complex tasks; AlexNet [25], a pioneering eight-layer network balancing efficiency and performance; and

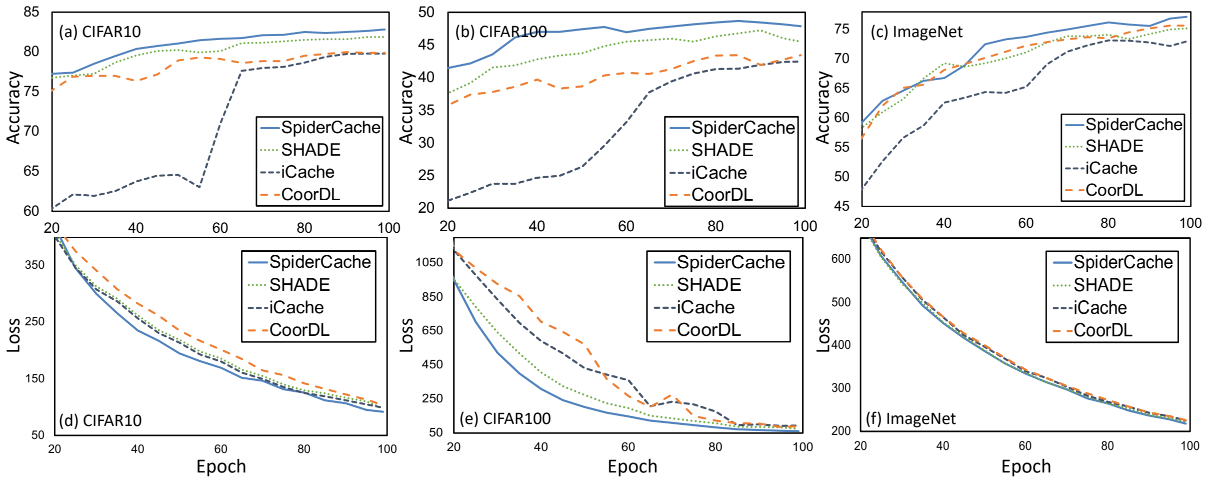

Figure 13: Comparison of IS Algorithm Performance in Terms of Accuracy and Loss

VGG-16 [38], a 16-layer model emphasizing depth with 3x3 convolution filters.

6.2 Effectiveness of Graph-based Importance Sampling Algorithm

In this section, we focus solely on comparing the effectiveness of the Importance Sampling (IS) algorithms, with all models having cache policies disabled to ensure a fair comparison.

SpiderCache represents our proposed graph-based IS algorithm, while **SHADE** refers to the loss-based IS algorithm proposed by SHADE. **iCache** represents the IS algorithm proposed in [20], a well-known computation-bound algorithm, adopted by iCache [7]. Finally, **CoorDL** represents the random sampling strategy utilized by CoorDL [30].

Since ResNet18 achieves the best performance on CIFAR10 and CIFAR100, and ResNet50 performs best on ImageNet, Figure 13 presents the experimental results obtained on CIFAR10 and CIFAR100 using ResNet18, and on ImageNet using ResNet50. Table 3 shows the Top-1 accuracy of different methods.

Table 3: Top-1 Accuracy(%)

Dataset	SpiderCache	SHADE	iCache	CoorDL
CIFAR-10	**81.8**	80.6	78.9	78.4
CIFAR-100	**45.7**	44.2	39.8	42.0
ImageNet	**75.2**	74.5	70.6	74.9

Key findings are as follows:

1) As shown in the accuracy comparison in Figure 13(a)-(c) and Table 3, SpiderCache achieved the highest accuracy across all three datasets. By utilizing the Importance Sampling (IS) algorithm, both SpiderCache and SHADE consistently outperformed random sampling (CoorDL). However, the computation-bound IS algorithm used by iCache primarily focuses on accelerating computations by skipping gradient updates for low-loss samples, resulting in the smallest accuracy improvement.

2) Loss reflects a model's learning status on the datasets, with lower loss indicating better model performance. As shown in Figure 13(d)-(f), SpiderCache demonstrates superior performance to others on CIFAR10 and CIFAR100. Given that CIFAR100 presents a more demanding classification task than CIFAR10 (with 10 times more classes), SpiderCache's advantages are even more pronounced on CIFAR100. ImageNet consists of 1.2 million images, provides a much larger number of samples for model training than the other two datasets. As a result, SpiderCache, SHADE, CoorDL, and iCache perform comparably in the loss metric.

6.3 Cache Hit Rate

We investigate cache hit rates under different cache sizes (10%, 25%, 50%, and 75% of the dataset size), and use an LRU-based caching policy with random sampling as the baseline. We compare SpiderCache, SHADE [22], iCache [7], and CoorDL [30]. For a more granular comparison, iCache was divided into iCache-imp (importance-cache-only) and full iCache (add random replacement), while SpiderCache was divided into SpiderCache-imp (importance-cache-only) and full SpiderCache (add homophily cache).

Figure 14 shows the average epoch cache hit rates for four models (ResNet18, ResNet50, AlexNet, and VGG16) on CIFAR10 under different cache sizes. The baseline uses PyTorch's default random sampling and LRU eviction. CoorDL[30] uses static caching, with hit rates proportional to cache size ratio. SHADE[22] improves hit rates with an importance-based strategy. iCache-imp[7], using a computation-bound IS algorithm, has lower hit rates than SHADE. Full iCache[7], with random replacement, achieves higher hit rates than SHADE. SpiderCache-imp, using a Graph-based IS algorithm for I/O-bound scenarios, outperforms CoorDL, SHADE, and iCache-imp. Combining homophily cache, SpiderCache achieves the highest hit rates, with up to 8.5× (average 4.15×) improvement over the baseline.

6.4 End-to-End Performance

In the end-to-end evaluation, we analyze the performance of four methods and the Baseline (LRU-based caching policy with random sampling) on Accuracy (model accuracy) and Efficiency (total training time), with all methods using a 20% cache size, a common setting in prior work[7][22][30], and training for 100 epochs. All

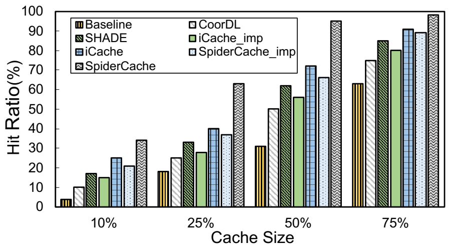

Figure 14: Average Epoch Cache Hit Ratio across four models on CIFAR10 under Different Cache Sizes

methods are evaluated with their full cache policies enabled. *Imp-ratio* refers to the portion of Importance Cache in the total cache, initially set to 90% and adjusted to 80% in the final evaluation.

Figure 15 presents the results of the best-performing models for each dataset: ResNet18 for CIFAR10/100 and ResNet50 for ImageNet. SpiderCache outperforms all baselines in both training efficiency and accuracy, owing to its carefully designed sampling and caching strategies. As shown in Tables 4 and 5, it achieves up to 2.33× (average 2.21×) speed-up over the baseline while maintaining the highest accuracy. SHADE reaches similar accuracy but is notably less efficient. iCache improves efficiency over SHADE and CoorDL via random replacement, but at the cost of accuracy. CoorDL and Baseline show the lowest efficiency due to reliance on random sampling.

Table 4: Total training time

Dataset	SpiderCache	SHADE	iCache	CoorDL	Baseline
CIFAR-10 (min)	**122**	171	160	199	284
CIFAR-100 (min)	**142**	199	175	213	314
ImageNet (hour)	**288**	380	361	429	611

Table 5: End-to-End Top-1 Accuracy(%)

Dataset	SpiderCache	SHADE	iCache	CoorDL	Baseline
CIFAR-10	**81.4**	80.6	72.8	78.4	78.3
CIFAR-100	**45.0**	44.2	37.7	42.2	42.0
ImageNet	**75.1**	74.3	67.5	74.4	74.3

To evaluate the impact of Homophily Cache (Section 4.2) on accuracy, we compare SpiderCache's performance between Table 3 (without Homophily Cache) and Table 5 (with Homophily Cache), showing that replacing highly similar samples has minimal impact on model performance.

6.5 Effectiveness of Elastic Cache Manager

To evaluate elastic caching, we compared SpiderCache's hit rates on CIFAR10 (ResNet18) under three strategies: *Imp-Ratio* 90%—a static 90:10 split between Importance and Homophily Caches; *Imp-Ratio* 90%-80%—a dynamic shift from 90:10 to 80:20; and *Imp-Ratio* 90%-50%—a dynamic shift from 90:10 to 50:50 during training.

Figure 16(a) shows that a static 90:10 ratio leads to a declining cache hit rate in later epochs as important samples decrease. A dynamic shift to 80:20 maintains stable hit rates by compensating with more Homophily Cache hits. Extending the shift to 50:50 further improves late-stage hit rates, demonstrating the effectiveness and necessity of elastic caching.

Table 6: End-to-end comparison under different *Imp-Ratio*

	90%	90%-80%	90%-50%
Top-1 Accuracy	81.63	81.44	78.87
Training time (min)	165	125	109

Figure 16(b) illustrates the accuracy under three strategies. With the same initial ratio, a lower *Imp-Ratio* shortens training time by improving cache hit rates through expanded Homophily Cache. However, this may slightly impact accuracy.

To quantify this, Table 6 reports the accuracy and training time for the three strategies. In SpiderCache, the *Imp-Ratio* is adjustable, allowing users to prioritize accuracy with a higher ratio or speed with a lower one. This flexibility supports customization based on specific data and training needs.

6.6 Multi-GPU Training

Figure 17 shows the per-epoch training time for ResNet18 on CIFAR10 with GPU counts ranging from 1 to 4. Compared to the baseline, the LRU-based caching policy, SpiderCache significantly reduces training time, with more pronounced decreases as the number of GPUs increases. This improvement arises from SpiderCache's ability to minimize I/O bottlenecks, thereby enhancing computational efficiency. Despite the acceleration achieved with multiple GPUs, there remains significant potential to further reduce overall computation time, primarily due to added overheads such as communication costs.

7 Related Work

In recent years, several approaches have been proposed to reduce storage I/O during DNN training. CoorDL[30] introduced the concept of data stalls—GPU idle time caused by slow I/O and preprocessing—and analyzed its impact. To mitigate I/O overhead, it proposed miniIO, a caching method that retains data without updates across epochs, leveraging full-dataset traversal in random sampling to maintain a stable cache hit rate.

More recently, some studies have aimed to improve system efficiency under importance sampling. The two most relevant works are iCACHE[7] and SHADE[22]. iCACHE categorizes samples into important and non-important groups, caching the former based on importance scores and randomly replacing the latter from the cache. However, its compute-bound sampling algorithm results in a low cache hit rate for important samples, while random replacement significantly harms accuracy. SHADE proposes a loss-based sampling method that ranks samples within each mini-batch using categorical cross-entropy, assigning a rank to each. Yet, it fails to effectively compare sample importance across mini-batches, and its cache strategy—based solely on importance scores—yields suboptimal performance.

A variety of works have been proposed to improve DNN training efficiency. DeepIO[46] uses entropy-aware batching to select informative mini-batches, reducing epochs to convergence. Hoard[32], Quiver[26], and FanStore[45] implement global caching across GPU clusters. DLFS[47] accelerates data access via direct hardware interaction. DIESEL[42] applies chunk-based shuffling and metadata

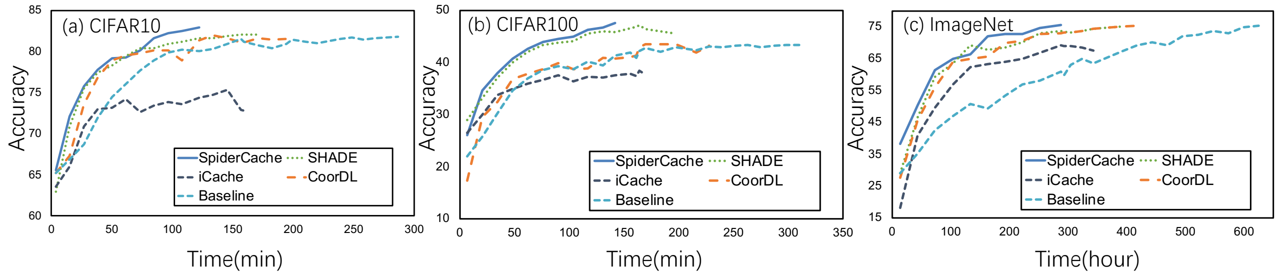

Figure 15: End-to-End Comparison

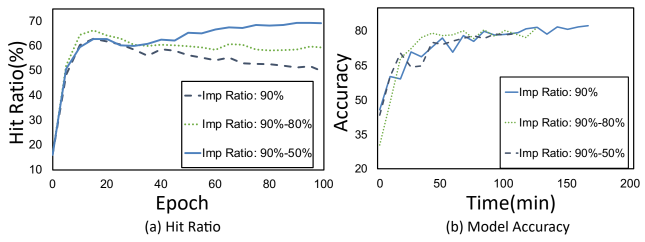

Figure 16: Comparison of Cache Hit Rates and Accuracy under Different Elastic Caching Strategies

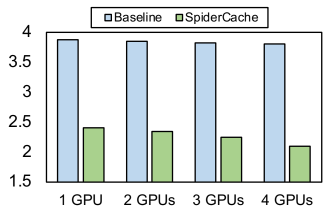

Figure 17: Per-epoch Training Time with Different GPU Counts

for task-level caching. ReFlex[23] enables remote flash access with local-like performance.

8 Conclusion

In this paper, we propose **SpiderCache**, a semantic-aware cache design for DNNs training. We introduce a global importance sampling algorithm and a semantic-aware caching mechanism to improve cache hit ratio. Additionally, we present an elastic cache manager that dynamically adjusts the cache size ratio between sections in real-time, based on the training state. The manager offers tunable parameters, allowing users to optimize for efficiency or accuracy. Extensive evaluation shows SpiderCache outperforms SHADE[22], iCACHE[7], and CoorDL[30] across efficiency, accuracy, and elasticity, providing a highly effective solution for DNN training optimization.

Acknowledgments

This work was supported by the National Key R&D Program of China (No. 2023YFB4502801), National Natural Science Foundation of China (Grant Nos. U22A2027 and 62402187), China Postdoctoral Science Foundation (Grant Nos. GZB20240243 and 2024M751009), and Postdoctoral Project of Hubei Province (Grant No. 2024HBB-HCXA024). We are grateful to our anonymous reviewers for their constructive comments and suggestions.

References

[1] Franklin Abodo, Robert Rittmuller, Brian Sumner, and Andrew Berthaume. 2018. Detecting work zones in shrp 2 nds videos using deep learning based computer vision. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 679–686.
[2] Stephen Balaban. 2015. Deep learning and face recognition: the state of the art. *Biometric and surveillance technology for human and activity identification XII* 9457 (2015), 68–75.
[3] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).
[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.
[5] Mariusz Bojarski. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).
[6] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 578–594.
[7] Weijian Chen, Shuibing He, Yaowen Xu, Xuechen Zhang, Siling Yang, Shuang Hu, Xian-He Sun, and Gang Chen. 2023. icache: An importance-sampling-informed cache for accelerating i/o-bound dnn model training. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 220–232.
[8] Xue-Wen Chen and Xiaotong Lin. 2014. Big data deep learning: challenges and perspectives. *IEEE access* 2 (2014), 514–525.
[9] Zhe Chen, Bolin Ding, and et al. 2021. SPANN: Highly-efficient Billion-scale Approximate Nearest Neighbor Search. In *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*. https://www.microsoft.com/en-us/research/publication/spann-highly-efficient-billion-scale-approximate-nearest-neighbor-search/
[10] Marco Dantas, Diogo Leitao, Cláudia Correia, Ricardo Macedo, Weijia Xu, and Joao Paulo. 2021. Monarch: Hierarchical storage management for deep learning frameworks. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 657–663.
[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.
[12] Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[13] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. 2017. Dermatologist-level classification of skin cancer with deep neural networks. *nature* 542, 7639 (2017), 115–118.
[14] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.
[15] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).
[16] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, and Roy Campbell. 2019. Tictac: Accelerating distributed deep learning with communication scheduling. *Proceedings of Machine Learning and Systems* 1 (2019), 418–430.
[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[18] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 604–613.
[19] Paras Jain, Xiangxi Mo, Ajay Jain, Harikaran Subbaraj, Rehan Sohail Durrani, Alexey Tumanov, Joseph Gonzalez, and Ion Stoica. 2018. Dynamic space-time

scheduling for gpu inference. *arXiv preprint arXiv:1901.00041* (2018), 1–8.

[20] Angela H Jiang, Daniel L-K Wong, Giulio Zhou, David G Andersen, Jeffrey Dean, Gregory R Ganger, Gauri Joshi, Michael Kaminksy, Michael Kozuch, Zachary C Lipton, et al. 2019. Accelerating deep learning by focusing on the biggest losers. *arXiv preprint arXiv:1910.00762* (2019).

[21] Tyler B Johnson and Carlos Guestrin. 2018. Training deep models faster with robust, approximate importance sampling. *Advances in Neural Information Processing Systems* 31 (2018).

[22] Redwan Ibne Seraj Khan, Ahmad Hossein Yazdani, Yuqi Fu, Arnab K Paul, Bo Ji, Xun Jian, Yue Cheng, and Ali R Butt. 2023. {SHADE}: Enable Fundamental Cacheability for Distributed Deep Learning Training. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*. 135–152.

[23] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. 2017. Reflex: Remote flash≈ local flash. *ACM SIGARCH Computer Architecture News* 45, 1 (2017), 345–359.

[24] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).

[26] Abhishek Vijaya Kumar and Muthian Sivathanu. 2020. Quiver: An informed storage cache for deep learning. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*. 283–296.

[27] Ilya Loshchilov and Frank Hutter. 2015. Online batch selection for faster training of neural networks. *arXiv preprint arXiv:1511.06343* (2015).

[28] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[29] Tomas Mikolov. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* 3781 (2013).

[30] Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. 2020. Analyzing and mitigating data stalls in DNN training. *arXiv preprint arXiv:2007.06775* (2020).

[31] Yosuke Oyama, Naoya Maruyama, Nikoli Dryden, Erin McCarthy, Peter Harrington, Jan Balewski, Satoshi Matsuoka, Peter Nugent, and Brian Van Essen. 2020. The case for strong scaling in deep learning: Training large 3d cnns with hybrid parallelism. *IEEE Transactions on Parallel and Distributed Systems* 32, 7 (2020), 1641–1652.

[32] Christian Pinto, Yiannis Gkoufas, Andrea Reale, Seetharami Seelam, and Steven Eliuk. 2018. Hoard: A distributed data caching system to accelerate deep learning training on the cloud. *arXiv preprint arXiv:1812.00669* (2018).

[33] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W Keckler. 2016. vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–13.

[34] Jared Roesch, Steven Lyubomirsky, Logan Weber, Josh Pollock, Marisa Kirisame, Tianqi Chen, and Zachary Tatlock. 2018. Relay: A new ir for machine learning frameworks. In *Proceedings of the 2nd ACM SIGPLAN international workshop on machine learning and programming languages*. 58–68.

[35] Abraham Savitzky and Marcel JE Golay. 1964. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry* 36, 8 (1964), 1627–1639.

[36] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 815–823.

[37] Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of big data* 6, 1 (2019), 1–48.

[38] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[39] Yifan Tan, Zheng Liu, Mingmin Ge, and et al. 2022. FINGER: Fast Inference for Graph-Based Approximate Nearest Neighbor Search. *Amazon Science* (2022). https://www.amazon.science/publications/finger-fast-inference-for-graph-based-approximate-nearest-neighbor-search

[40] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).

[41] Jason Wang, Luis Perez, et al. 2017. The effectiveness of data augmentation in image classification using deep learning. *Convolutional Neural Networks Vis. Recognit* 11, 2017 (2017), 1–8.

[42] Lipeng Wang, Songgao Ye, Baichen Yang, Youyou Lu, Hequan Zhang, Shengen Yan, and Qiong Luo. 2020. Diesel: A dataset-based distributed storage and caching system for large-scale deep learning training. In *Proceedings of the 49th International Conference on Parallel Processing*. 1–11.

[43] Suorong Yang, Weikang Xiao, Mengchen Zhang, Suhan Guo, Jian Zhao, and Furao Shen. 2022. Image data augmentation for deep learning: A survey. *arXiv preprint arXiv:2204.08610* (2022).

[44] Jiong Zhang, Hsiang-Fu Yu, and Inderjit S Dhillon. 2019. Autoassist: A framework to accelerate training of deep neural networks. *Advances in Neural Information Processing Systems* 32 (2019).

[45] Zhao Zhang, Lei Huang, J Gregory Pauloski, and Ian T Foster. 2020. Efficient I/O for neural network training with compressed data. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 409–418.

[46] Yue Zhu, Fahim Chowdhury, Huansong Fu, Adam Moody, Kathryn Mohror, Kento Sato, and Weikuan Yu. 2018. Entropy-aware I/O pipelining for large-scale deep learning on HPC systems. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 145–156.

[47] Yue Zhu, Weikuan Yu, Bing Jiao, Kathryn Mohror, Adam Moody, and Fahim Chowdhury. 2019. Efficient user-level storage disaggregation for deep learning. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 1–12.