Optimal Distributed Filesystem and Database for Multimodal ML Training

Kastan Day

Grainger College of Engineering, University of Illinois at Urbana-Champaign,
Center for AI Innovation, National Center for Supercomputing Applications, UIUC.

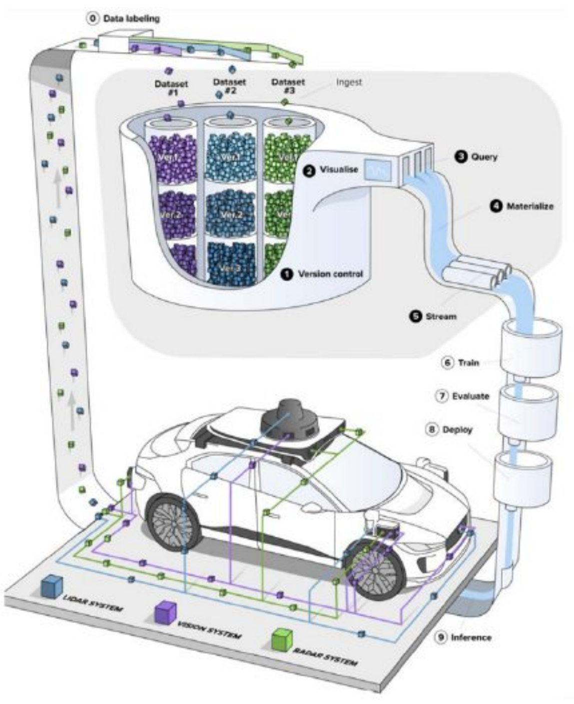


Figure 1 What filesystem and database are suitable for multimodal ML data collection, wearhousing and training? Many competitors exist, but none addresses the full scope of challenges for multimodal, distributed, low-latency, high IOPS, and cost-efficient data stores. Figure from [1].

Abstract

Data, not model architecture or compute, is the differentiator in machine learning capabilities. New ML systems are increasingly multimodal, and scaling from text to images to video to 3D point clouds will put ever-greater pressure on storage systems. Today, there is no silver bullet for both long-term data storage, and ultra-low latency caching for ML training, and every major ML innovation is accompanied by a bespoke and costly data engineering effort. This work evaluates the cloud data solutions for complex non-tabular and n-dimensional vector datatypes used in machine learning. ML-first databases like Deep Lake paired with a cloud object store like S3 offer the best developer experience and lowest cost available today; an ideal choice for academics.

Keywords: ML databases, distributed filesystems, Multimodal training, Non-tabular relational databases.

1 Introduction

It’s a very exciting time in databases. The NoSQL movement came and went, columnar databases are complementary with RDBMS and there’s a growing wave of unstructured data used in AI applications that is putting huge strains on data warehouses (for structured SQL data and semi-structured JSON) and data lakes (for unstructured audio/video/vector data). However, a major gap exists for non-tabular data, especially n-dimensional vector data visualized in figure 2, historically used in geospatial applications, but which is now commonly used to store machine learning (ML) embeddings and training data. In essence, scientific datasets that don’t fit into SQL tables or DataFrames are not well served by any database solution in the cloud today. Historically, and even today, this data is stored in HDFS filesystems, with flexible file formats like HDF, NetCDF and HDF5. These formats are widely loved for optimized random access reads from extremely flexible data structures, especially supporting n-dimensional arrays. However, these formats are incompatible with the only practical cloud storage solution for big data: object storage, as evaluated later in this work via AWS S3. Object storage largely destroys what made these data formats so great: efficient random access, leading to large cloud egress bills.

1.1 Scientific Data is Breaking S3

The problem is as urgent as ever. NASA has accumulated an EarthData Worldview archive of almost 22 petabytes (PB) which is expected to grow to 247 PB by 2025 at roughly 114 TB per day. Most of this data is from satellites that produce HDF records that will be uploaded to S3 to increase data access to researchers. However, HDF files were never designed for the cloud. The metadata that makes random access so efficient is strewn throughout the file, which was efficiently handled by the filesystem, HDFS, but is incompatible with object storage, forcing users to download the entirety of large files just to read small internal elements. NASA realized this, too, citing “cloud egress fees” as the number one concern that could make “scientific data become less available to researchers” [2]. Machine learning projects face the same concerns.

The best so called “cloud-native” alternatives to HDF are Parquet and Cloud-Optimized GeoTIFF, but each has compromises. The problem remains that n-dimensional vectors aren’t well suited to be stored in Parquet tables, or as a stack of ‘images’ in a GeoTIFF. It’s possible to make these data fit, and that idea is evaluated below, but is dramatically inefficient for the

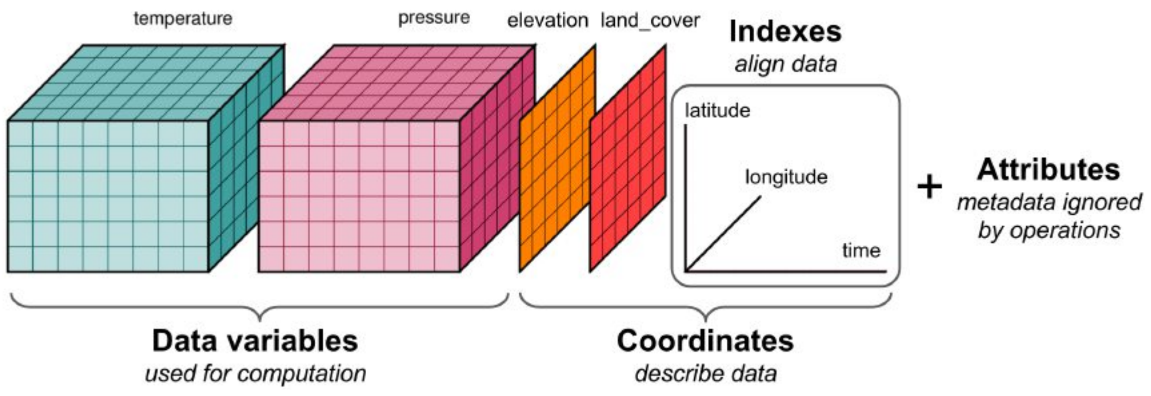


Figure 1.1 N-dimensional vectors as visualized in the popular Xarray library [2], showing a single datapoint with mixed-size tensors, search indices and JSON-like metadata.

expected data access pattern, misses critical opportunities for compression and altogether ruins the benefit of optimized storage in the first place.

1.2 Training Data Diversity Necessitates New Databases

Mirroring the scientific world, ML training datasets build complex n-dimensional tensor datatypes mixed together with many forms of audio, video, pdf together. Most large datasets today rely on one-off massive data engineering efforts from large tech companies. Best practices are nascent and evolving, especially for academic research groups. This work address the needs of ML academics building custom datasets easily and cheaply, while supporting multi-cloud and on-prem for both data and compute.

Large Language Models (LLMs) are starved for data. Deepmind's Chinchilla paper [3] famously proposed compute optimal LLMs should be trained on 20 tokens of text per model parameter, many times the data size of all prior works. Pushing data sizes even large, FAIR's LLaMA paper [4] pushed this rule of thumb even further by showing that "small" 7B models continue to learn after 1T tokens (142x tokens to params). This data size is manageable when our productionized datasets of 1.4T tokens is only a few TB of data on disk, but becomes entirely unmanageable when multi-modal models require image, video, vectors and even 3D point clouds [5]. For example, replicating CLIP requires 5B images and 110 TB [6], [7]. Self-supervised learning [8] and simulation data [9], [10] will drive the explosion in data sizes, making data engineering the central engineering challenge in SOTA ML systems. Therefore, more than ever, we need scalable databases specifically for parallel and distributed ML training.

The winds of the scaling laws have shifted back from prioritizing unimaginably large models to emphasizing unimaginably large and high-quality data. Open-source models are still a long way away from utilizing all the publicly available data on the internet, let alone all the private data. Pre-training on a curated collection of the best public datasets and fine-tuning on proprietary data will be an effective strategy. AI projects should pay close attention to data quality infrastructure.

2 Related Work

This work considers the tradeoffs when working with structured (e.g., tabular) vs semi-structured (e.g., JSON) vs unstructured (e.g., photo/video) vs n-dimensional matrices (e.g., Tensors and GeoTIFF). Tabular data problems are well serviced by solutions like DataBricks & Spark or RedShift and Sage Maker or Google BigTable. But in unstructured data there's a tension between "files on disk" creating a data swamp, and "files in database" in a relational database being the wrong tool for the job.

2.1 Data Wearhouse and Data Lakes

Data warehouses and data lakes attempted to solve the problems of (1) separate data silos with no single source of truth and (2) unifying data types from structured SQL to semi-structured JSON and even unstructured media files into a single query-able database. The first generation of data lakes relied on 'files on disk' use of distributed file systems, most prominently HDFS [11] and, starting in 2015 onwards from cloud providers AWS S3 [12] and GCS[13]. Cloud providers offered greater durability (>11 nines) and potentially automatic cost savings via offload to archival storage e.g., AWS Glacier. However, these completely unorganized object stores quickly devolved into "data swamps" that were costly to manage due to high data egress fees and the inherent difficulty of querying unstructured data like photos and videos.

Second generation data "Lakehouses" addressed the need for organization by simply combining a two-tier architecture of SQL-style databases with links to media in object storage, hence the name Lakehouse for combining structured warehouses with unstructured lakes, as shown in figure 2.1. These systems typically offer more flexibility and performance in their RDBMS system by building on open-source, structured formats such as Parquet and OCR [79, 6], which enabled formats like Delta, Iceberg, and Hudi [14]–[16] with features like ACID transactions, time travel, and schema evolutions. These offer the clean organization and optimized query performance of an RDBMs with moderately more flexibility via evolving schemas and columnar-oriented scale-out performance. These systems are typified by Snowflake, BigQuery, Redshift, Clickhouse and Databricks.

However, second generation Lakehouses suffer from the same limitations as traditional RDBMs: they can't store binary data like media, or high-dimensional matrices that are used as machine learning (ML) embeddings. Reading unstructured data is inefficient or impossible with SQL queries, forcing data scientists to export results into 3rd party tools for even the most basic analysis. Moreover, Lakehouses offer little integration with the ML ecosystem often requiring expensive data ETL pipelines to convert SQL queries into intermediate formats like NumPy matrices or TensorFlow Tensors before the data can be loaded via PyTorch or TensorFlow Dataloaders.

For example, Databricks's LakeHouse [14] primary innovation is the DataBricks File System (DBFS) [17] that uses S3 as a high-performance storage system for Spark DataFrames [18]. The filesystem is heavily tuned for caching and query optimizations to reach RDBMs-like performance using much cheaper object storage instead of hosted SQL systems. Furthermore, it co-locates structured and unstructured data in the same S3 storage, helping to break down silos and keep data fresh. Ideally programmers shouldn't have to laboriously move data from cold to hot storage, and this exemplifies that philosophy. Unfortunately, Databricks focuses on optimizing SQL Business Intelligence-style queries on cheap S3 storage and does not innovate in support for complex datatypes like n-dimensional tensors or integrate with the ML ecosystem.

Apache Spark is the go-to processing engine for data science and ML at scale, but it is only optimized for tabular data. I used it during my time at Apple, and respect it deeply but I see so many better options on the horizon, and finding the best combination of them to displace Spark is the purpose of this project. Spark, and the RDBMs it often attaches to, are optimized

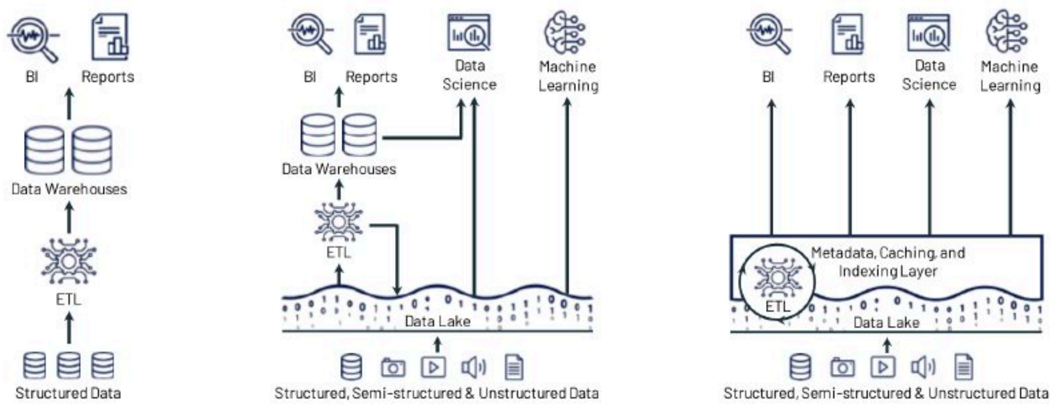


Figure 2.1 The three generations of data warehouses. (left) first generation systems supported only tabular data, (middle) then second-generation systems added links to unstructured media stored separately, and (right) future systems aim to flexibly support all data types in a single cheap lake with high-performance queries. Figure from [12].

for SQL query performance, but analytical queries don't lend themselves to building datasets with complex compute-intensive processing. It's also missing the ability to stream the query result as training data directly to remote GPU nodes. It's missing the ability to resume streaming from the middle of an SQL query, such as when one wants to resume training from a checkpoint without iterative over all previously seen training data. Spark excels in tabular data but falls short of other solutions for ML data preprocessing and training. Future LakeHouse systems should aim to address these key limitations.

2.2 Generations of Databases

The evolution of databases can be broadly categorized into three key generations: Relational Database Management Systems (RDBMS), NoSQL, and Data LakeHouses. This progression has been driven by the need to accommodate diverse data types, structures, and use cases, as well as to address the increasing complexity and scale of data storage and management.

The first generation, RDBMS, emerged to facilitate transactional processing and enforce data consistency through the ACID (Atomicity, Consistency, Isolation, Durability) properties. RDBMSs primarily utilize SQL, a structured query language, to manage and manipulate data. Despite their robustness, RDBMSs faced limitations in handling large-scale, distributed data storage and processing, which led to the advent of the second generation, NoSQL.

NoSQL databases were designed to address the scalability and flexibility challenges of RDBMSs, particularly in the context of big data and web applications. NoSQL encompasses various types of databases, including columnar and key-value stores, as well as NewSQL databases that combine the benefits of both RDBMS and NoSQL. Instead of adhering to the ACID properties, NoSQL databases follow the BASE (Basically Available, Soft state, Eventual consistency) properties, which prioritize availability and partition tolerance over strict consistency.

The third generation Data LakeHouses emerged as a response to the limitations of both RDBMS and NoSQL in handling diverse data types and structures, such as media, large vectors, and high-dimensional data. Data LakeHouses leverage optimized indices for metadata-filtered cosine similarity queries and support vector databases, which are better suited for these complex data types. But they are still in their infancy, and intense competition between cloud providers for the lucrative job of storing all of a company's data is driving rapid innovation.

2.3 Networked Filesystem Evolution

High performance networked filesystems started with NFS, then the needs of supercomputing demanded parallel filesystems, bringing about GBFS, LustreFS, and CephFS. Yet these early filesystems were designed for spinning HDDs and large file sizes, and rely on separate POSIX metadata stores that keep file descriptors separately from block storage. Finally, third generation filesystems are designed for all-SSD devices and handle the Lots of Small Files (LOSF) problem without metadata bottlenecks. Early examples of these include WekaFS and Vast Data but only Weka was evaluated since all others required enterprise contracts instead of self-service public cloud deployments.

WekaFS is a ML-first filesystem provider, offering a SaaS filesystem under the sky-computing philosophy: it's one-click deployable on AWS, Azure, GCP and OCI providers and is closed-source deployable to on-prem clusters [19]–[21].

What sets WekaFS apart is distributed data and patented metadata stores to support any IO pattern, including LOSF which is common in ML and scientific HPC workloads. Linear scaling to unlimited random IOPS at sub-250 microsecond latency is a dream come true, especially when you consider that the majority of the data is stored on HDDs. Other benefits like supporting single files up to 4 petabytes and 14 exabytes in a single namespace, make it suitable to the astounding diversity of ML workloads.

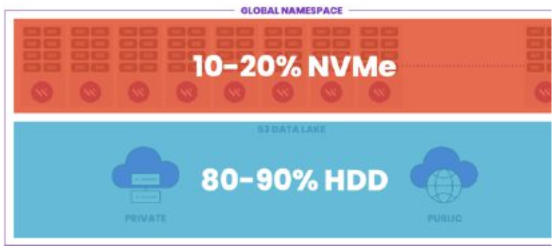


Figure 2.2: WekaFS' adaptive SSD cache on top of heterogenous on-prem and cloud (S3) storage, figure from [21].

Traditional HPC filesystems like Lustre, GlusterFS, GPFS and HDFS break down under modern workloads. The root of the issue is metadata management, where dedicated file metadata servers can't scale and swap metadata from DRAM to disk at a tremendous rate. This exact issue plagues that ML training performance of the NCSA's new Delta supercomputer, where small file IO falls in the 10s of Mbps of throughput, where large file IO sees 10s of Gbps. This enormous operational problem initially spurred my interest in this field.

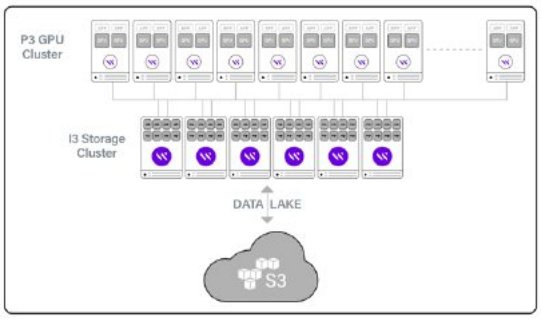


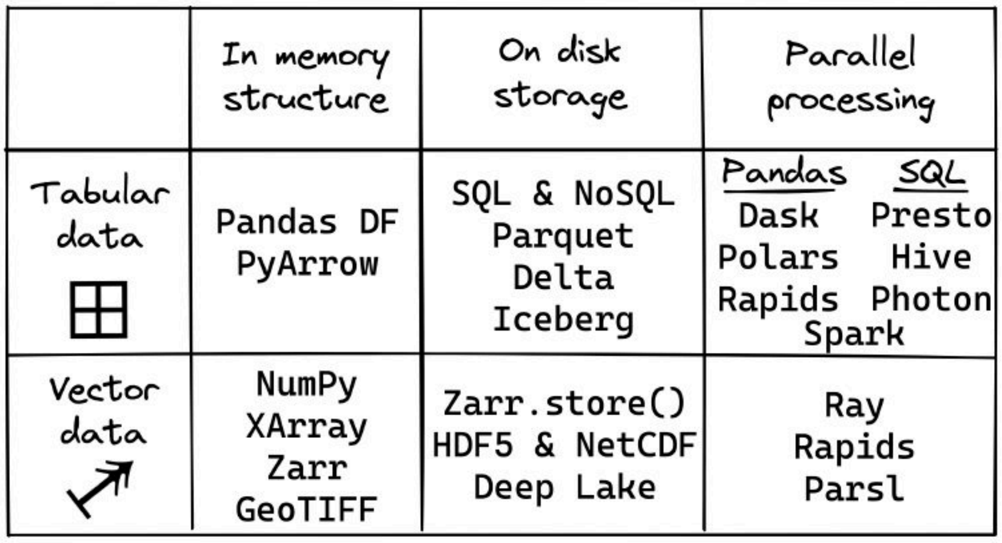
Figure 2.3: WekaFS as a caching layer to seamlessly make S3 sufficient for world class distributed ML training throughput [21].

Another key usage pattern for WekaFS is as a wrapper on top of S3, as shown in figures 2.2 and 2.3, making it easy to spin up an intelligent SSD cache in front of long-term S3 storage. This can be used to alleviate Dataloader bottlenecks in ML training if data is being streamed from S3, Weka can serve as a transparent no-code, drop-in caching layer for instant performance improvement. Any no-code improvements are invaluable to replace expensive engineering time in the already overly complex ML tooling landscape.

2.4 Python Native In-Memory Structures and Files Formats

Python has excellent in-memory data structures for vector and unstructured data, like NumPy, Xarray and Zarr as detailed in figure 2.4, but severely lacks on-disk persistent file formats that support cloud object storage or relational query features. After evaluation against my requirements for ML data tooling, enumerated in the following section, all candidates fail the test. The most notable tool by far is Apache Parquet columnar data format, and the complementary Apache Arrow in-memory layout, which is ideal for text and tabular data, and which feels like an ideal solution that solves text-storage problems. Despite this success, Parquet and Arrow don't work for media or large vector storage due to its lack of support for n-dimensional arrays. In contrast, Deep Lake is optimized for large, dynamically shaped tensor data, enabling modern ML and geospatial workloads.

Between these tools Zarr and HDF5 certainly could solve the problem with high performance but have difficult to use APIs and fall into the 'files on disk' paradigm without support for database features. Zarr is more modern than HDF5, and even has experimental



	In memory structure	On disk storage	Parallel processing
Tabular data ⊞	Pandas DF PyArrow	SQL & NoSQL Parquet Delta Iceberg	Pandas SQL Dask Presto Polars Hive Rapids Photon Spark
Vector data ↗	NumPy XArray Zarr GeoTIFF	Zarr.store() HDF5 & NetCDF Deep Lake	Ray Rapids Parsl

Figure 2.4 Overview of data formats in Python for tabular and vector data.

support for cloud object storage, making it just about perfect if one can sacrifice relational features in favor of very high performance, partial I/O, hierarchical organization, and arbitrary metadata. On the downside, Zarr can require significant tuning to achieve good performance, has less-than-perfect support for ragged arrays and cloud object storage, and a highly complex API that is an unappealing developer experience in comparison to NumPy, XArray or Deep Lake. It also lacks database features like query-ability and ML-specific features like streaming to Dataloaders which pushed me to keep searching for a better tool for my problem. Enter ML-first database solutions.

2.5 ML-first databases

In this category, on-disk data layout is the core determinant of performance and design flexibility of the database to meet the requirements detailed below. This work compares three leading ML databases Deep Lake [22], DuckDB [23] and LanceDB [24] as well as other popular cloud-native Dataloaders like WebDataset [27], FFCV [28], MosaicML Streaming [29], and Deep Lake Streaming [22].

The essential criteria for selecting a suitable database for machine learning data preprocessing and training applications used in this work are:

- Truly distributed read and write capabilities, allowing for mutable databases with in-place updates, with ACID or near-ACID guarantees.
- Columnar $O(1)$ indexing to enable efficient random sample retrieval and, ideally, vector-based similarity search.
- On-the-fly compression, incorporating both sample-based compression (e.g., JPEG, MP4) and chunk-based compression (e.g., LZ4 and ZSTD).
- Support for ragged arrays, n-dimensional vectors with significantly different shapes, while avoiding padding with wasted space.
- “Nice to haves” include (git-like) data checkpointing and redundancy, which may be provided by the filesystem.

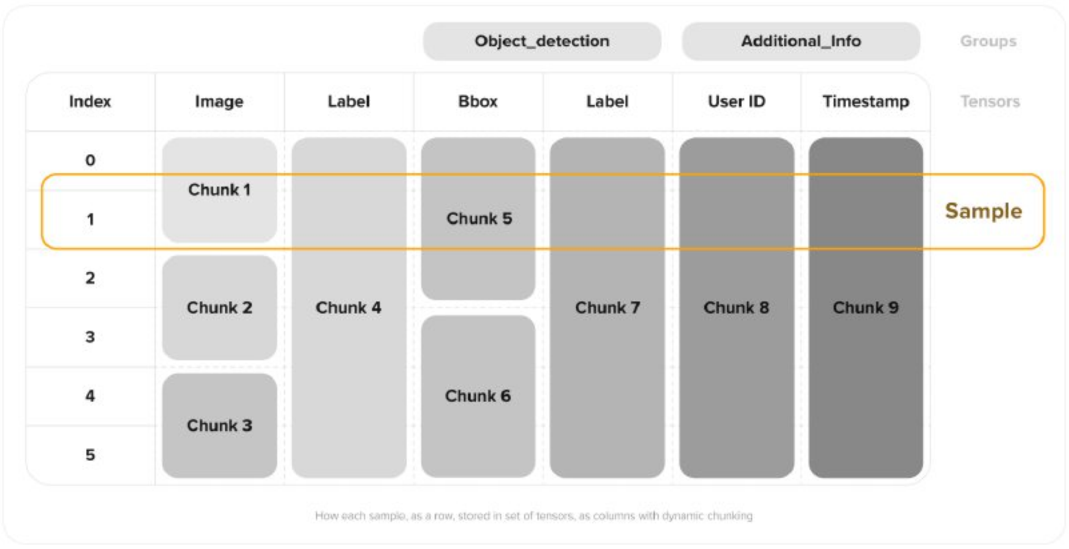
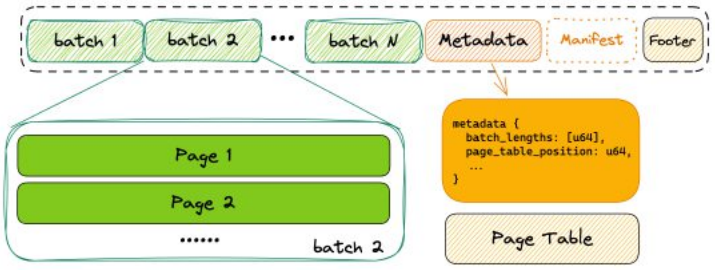


Figure 2.5 Deep Lake data layout on disk. The use of variable chunk sizes for chunk-based compression fundamentally enables raged arrays, or storing matrices of different shapes, efficiently on disk in a single database. Furthermore, it enables each modality, such as images and audio, to use domain-specific compression algorithms like JPEG, MP3, MP4, Point-Cloud LZ4, which is essential for big data ML. Figure from [17].

Based on these requirements, Deep Lake, the Data Lake for Deep Learning [22], is the most capable, stable and user friendly data lake solution for deep learning applications. Deep Lake offers optimal compression and infinite flexibility due to its unique data layout on disk with both columnar indexing and flexible chunking, which is further explained in Figure 2.5. Deep lake offers explicitly multi-modal data storage and many of the other required features, with the notable exception of distributed write operations, and instead only supports multi-threaded but single-node writes, as shown in figure 2.7. This may be overcome by combining it with a distributed filesystem like WekaFS for a potentially world-class distributed multimodal data solution. This is currently the leading option I see in the ecosystem.

The most capable competitor to Deep Lake is the FiftyOne database whose founders have published over 250 papers during 30 years of academic experience [25]. Their platform is focuses on heavily on images and 3D point clouds for industry CV and ML workflows. Although appearing very similar to Deep Lake I found that it lacked the parallel and distributed processing capabilities I required and lacked some of the streaming features, like streaming from S3 directly to GPU nodes, that make big data workflows much simpler in multi-cloud environments. Data streaming offers incredible flexibility and cost savings because one can keep data wherever it's cheapest and spin up GPU nodes wherever they're cheapest at any given moment by quickly and reproducibly streaming training data to the GPUs. Another



```
metadata {
  batch_lengths: [u64],
  page_table_position: u64,
  ...
}
```

Figure 2.6 LanceDB's file structure uses a unique batch-native-layout that may accelerate ML apps during streaming, but introduces write inefficiency when constructing complicated batch. Figure from [25].

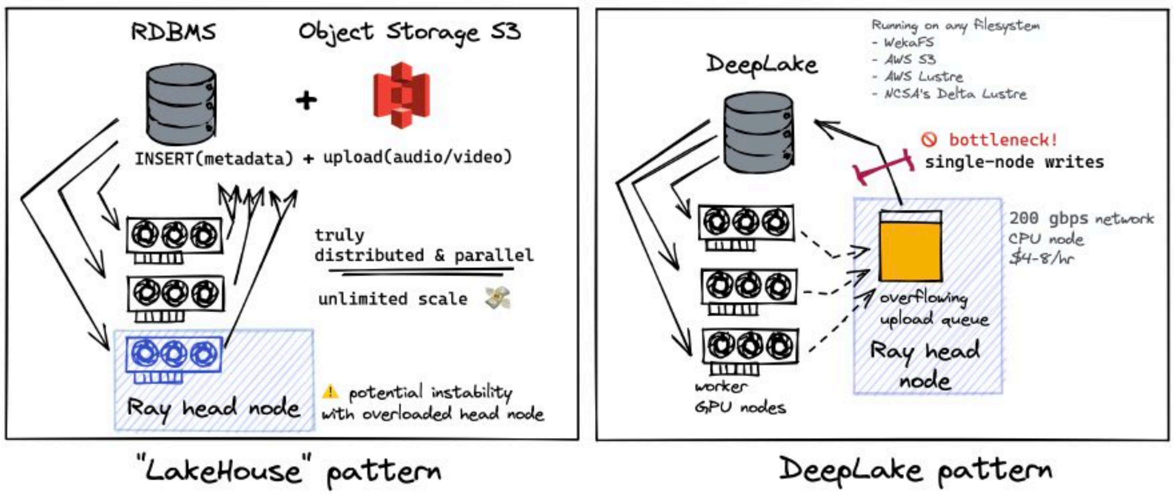


Figure 2.7 The LakeHouse pattern (left) enables truly distributed, horizontally scalable performance, but misses all the ML-focused benefits of a single Deep Lake database. However, Deep Lake currently uses write locks to enforce transactional safety, meaning it's capable of multi-threaded writes, but not multi-node writes, posing a bottleneck for truly parallel data ingest.

competitor, LanceDB [24] as shown in figure 2.6, is a young startup in this space, but focuses more on the ML inference rather than data preprocessing and ML training, with strong support for cost-efficient serverless vector retrieval and similarity search. Finally, DuckDB [23] is an established product in vector databases, supporting SQL's Online Analytical Processing (OLAP) of multi-dimensional arrays of data, which again works perfectly for ML inference, but like LanceDB, is not appropriate for concurrent writes or distributed data preprocessing.

Uber Petastorm [19] and Ray Data [20] offer distributed data primitives like map, shuffle and sort that are suitable for ETL-like uses but do not provide persistent data structures for storage let alone database query features. Like most ETL libraries, they lack any proprietary database object and instead use Apache Arrow and Parquet columnar data format for efficient storage. Both systems are used in industry today, where Petastorm is in used Databricks ML offerings and Ray Data is a popular stand-alone library in use at IBM Research in the Foundation Model initiative and at Google in the Alpa parallel model training project. I used Ray and Ray Data to implement all parallelism used in this work, paired with Deep Lake as the persistent datastore and high-performance streaming PyTorch Dataloader.

3 Methods

To determine the best filesystem and database combination for ML data preprocessing and training, I employ both synthetic benchmarks using the industry standard `fio` Unix utility and real-world experiments using a common vector-based ML training dataset. All benchmarking code is available on GitHub [26].

To ensure I was fairly benchmarking the filesystems, I interviewed JD Maloney, Sr. HPC Storage Engineer and designer of Delta's storage infrastructure, about the tradeoffs of different filesystems and how I may avoid common bottlenecks. JD explained the three generations of filesystems, and the tradeoffs that led him to choose LustreFS for the Delta supercomputer. He explained how supercomputer designers estimate filesystem needs and provision metadata servers to support the anticipated workloads. On Delta, the primary "hot" storage system uses 14 NVMe SSDs just for metadata lookup, achieving 80k to 90k file creates per second, which is 2x the metadata performance of Blue Waters. However, my workload was naïvely reading and writing millions of small files, leading to large bottlenecks in metadata operations. Despite recording up to 12 Gbps of network throughput, that was a small fraction of the expected the 100 Gbps networking. With this information about metadata server bottlenecks, I transformed my dataset from LOSF to one Deep Lake database thereby eliminating the metadata bottleneck and allowing me to reach 75+ Gbps network throughput with fully saturated GPUs. This was nice proof that a good database can compensate for a poor filesystem, or vice versa and as a result the workload selected for experimentation will stress both raw synthetic performance and realistic ML performance.

4 Experiments

The experiments compare industry standard parallel filesystem Lustre, initially released in 2003, against modern and closed source Weka.io, founded in 2013. I additionally compare against Databricks' implementation of the Databricks File System on AWS S3 and AWS S3 itself. Particular focus is paid to write performance, small-file IO, and sustained workloads where SSDs can become bottlenecked. In all experiments the parallel database Deep Lake is used, due to its outstanding performance and feature set, and although a comparison to Databricks is made, a direct comparison to Spark is impractical since it doesn't support n-dimensional Tensor datatypes.

4.1 Experimental setup

All experiments are conducted on the same environment in AWS, comparing AWS's highest performing filesystem AWS FSx Lustre, against Weka, where, despite the home field advantage, it falls behind in both maximum performance and total cost of ownership.

Weka leverages AWS's enhanced networking (ENA) detailed in Appendix A. Moreover, all instances support AWS Elastic Fiber Adapter (EFA) which enables technologies like NCCL, and RDMA which are critical to distributed ML training [27]. EFA bypass the CPU and DRAM, enabling direct NIC to GPU communication, eliminating a crucial bottleneck in large model training. It paves the way for GPUs to directly communicate (via RDMA) at PCIe 5 speeds and beyond, taking another step towards the goal of enabling large clusters of GPUs to act more and more as a single device.

However, in extremely high IOPs scenarios, those approaching 10 to 20 million IOPs on a single host, the Linux kernel becomes the bottleneck. Fortunately, Weka and AWS support Intel Data Plane Development Kit [28], which is also included in the Linux kernel used by Ubuntu, to bypass the Linux kernel TCP/IP implementation, promising higher bandwidth and packet-per-second (pps) performance. One AWS performance watchdog team managed a 60% performance improvement even on 10 Gbps ethernet (3.8 Gbps at 551k pps on TCP/IP Linux kernel vs 6.1 Gbps at 884k pps on DPDK) showing that bypassing the kernel can be productive even at modest network speeds [29].

Weka's filesystem requires a minimum of 6 storage servers, AWS i3en-category devices that are optimized for SSD IO. It uses a 4+2 replication policy, meaning the cluster can sustain 2 failed servers before experiencing data loss. Subsequently, one or more client servers connect to the storage cluster and run a user's target application, such as ML training. It is possible to run jobs directly on the storage servers, but it's not recommended as it will break the assumptions Weka engineers made by causing CPU resource contention. Exact experimental details can be found in appendix A.

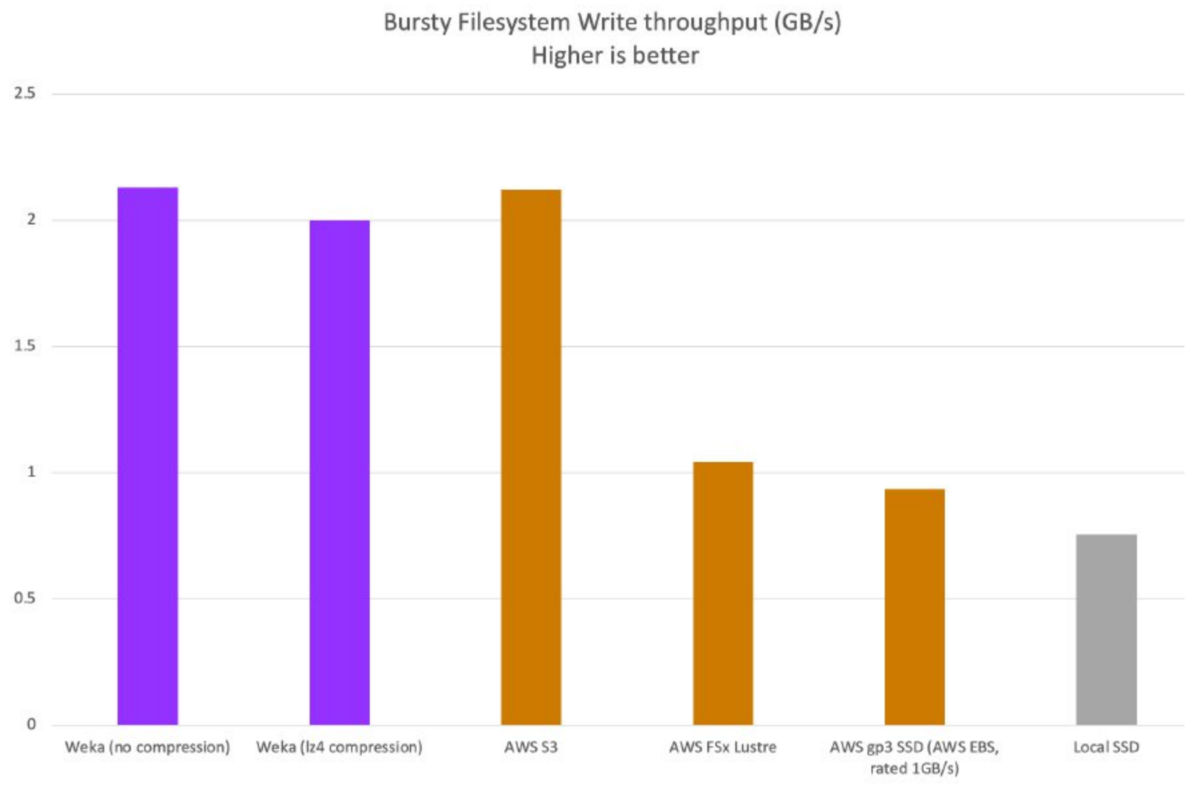


Figure 4.1: WekaFS is tied with S3 for the highest write throughput, holding other variables constant, showing Deep Lake can write to both filesystems equally efficiently and both saturated the node's CPU. Enabling filesystem-level compression does not have a meaningful impact on throughput, suggesting performance is network bandwidth constrained, since the CPUs have free cycles to perform compression.

In figure 4.1, each run writes 380 GB of data, made up of 100k fp32 NumPy vectors of shape (1024, 1024). This dimension is most common shape of input to (multimodal) transformers and most large language models. The vectors are constructed in parallel, across all but 1 of the available threads, leaving room for the Weka driver, and uploaded to the database. Weka completed the task in 4 minutes while AWS Lustre required 6.25, or 78% more wall-clock time.

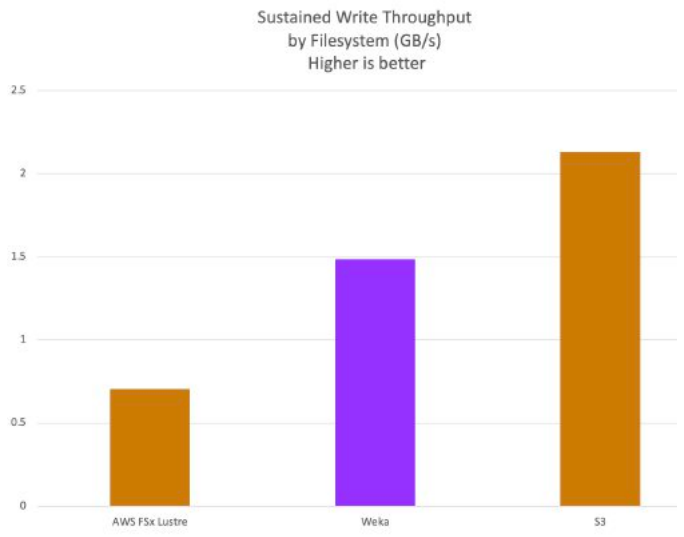


Figure 4.2: In sustained writes, AWS Lustre's performance decays rapidly, resulting in 18.5 minutes vs 8.8 min for Weka to complete the job, or 210% more wall-clock time. However, S3 experiences no performance drop during sustained writes leading to better performance than Weka.

In figure 4.2 and 4.3, the weaknesses of AWS Lustre are exasperated by decaying performance over longer, sustained writes. With double the data size, 200k vectors, AWS's Lustre throughput began at 2.2 GB/s (of the

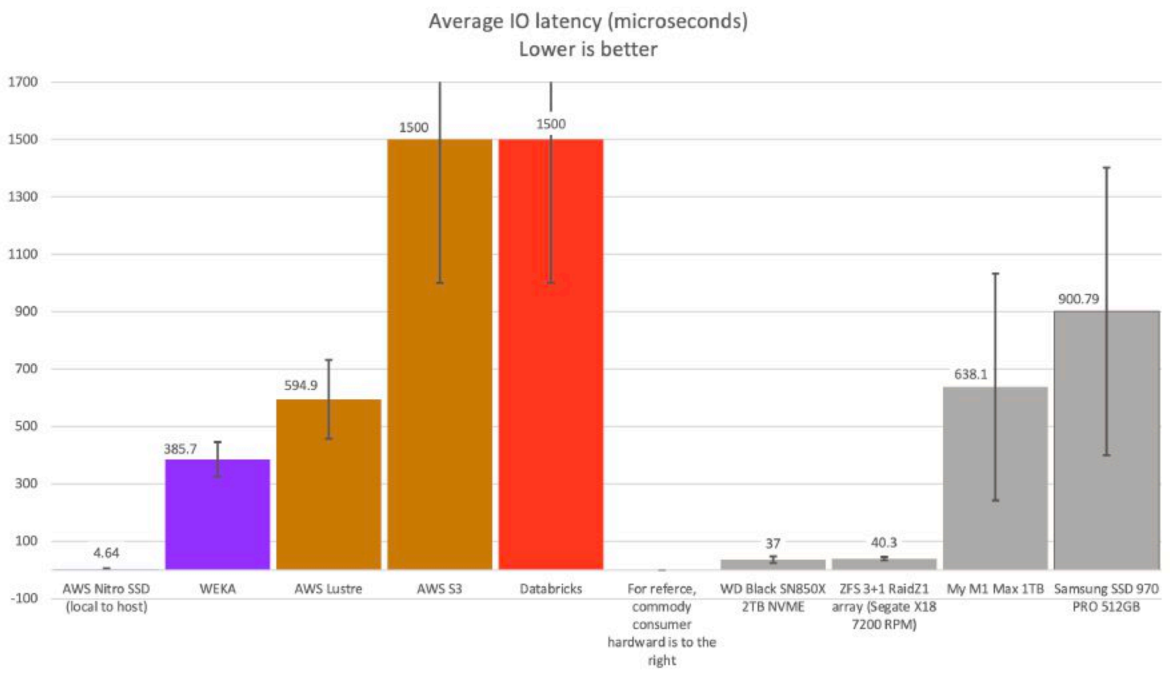


Figure 4.3: IO latency is crucial when constructing a dataset with random file reads, and Weka has both 54.2% lower latency and 228% lower average deviation in latency than AWS Lustre (60 us vs 136.9 us for AWS Lustre). Notably, AWS's in-house SSDs have extremely fast response times, 10x that of WD's fastest consumer drive, the SN850X. Predictably, Databricks built on S3, and S3 itself, had by far the worst latency, which is the only major compromise in S3 performance.

quoted 2.88 GB/s) and performance steadily dropped to 400 MB/s, and to a steady state of 800 MB/s, with a minimum speed of 683 MB/s. IO latency was a similar story, with Weka showing off 54.2% lower latency access.

4.2 Weka Surpasses Lustre in All Evaluations

Weka benefits from up to 2x faster throughput that does not degrade on continuous writes, up to 54% lower latency, 200% lower volatility latency. This evaluation shows Weka offers twice the performance of AWS Lustre for the same price. However, it still cannot beat the price to performance ratio of S3, especially when using Deep Lake to make the most of S3 storage performance. A detailed price to performance comparison against all filesystems evaluated is made in figure 4.4.

Weka is unique in the filesystem landscape for its unique partitioning of metadata, making is scale linearly with cluster size and excellent at small file IO, like images and audio clips, that are so common in multi-modal AI model training. Weka is best suited for jobs that require LOSF, extremely low predictable latency, or jobs that receive distributed writes from many clients at once, otherwise S3 may be sufficient. Weka helps prevents AWS lock-in by easily migrating to any Cloud provider, or a hybrid cloud and on-prem approach. It goes so far as to allow users to specify different object storage compliant backends for *individual datapoints* lending users ultimate multi-cloud flexibility and minimized egress costs.

4.3 Database Ingest and Egress/Streaming Performance

Specifically addressing the needs of fast, distributed PyTorch Dataloaders are WebDataset [25], FFCV [26], MosaicML Streaming [27], and Deep Lake Streaming. Prior work on Deep Lake shows faster performance on equivalent AWS hardware versus the other

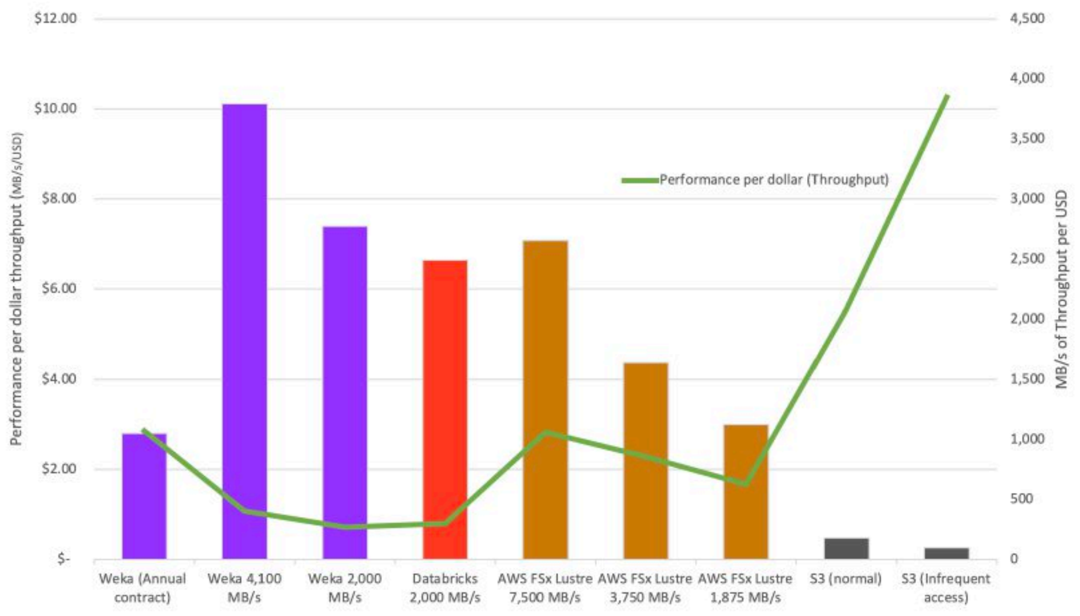


Figure 4.4 Price (vertical bars) vs performance (line) comparison, the vertical bars represent the USD cost per hour to run a solution (lower is better), and the green line represents the performance per dollar in MB/s of throughput (higher is better). Weka offers the highest absolute performance but S3's price advantage with high throughput offers unmatched value.

works tested as detailed in figure 4.5. Tests were run on an AWS c5.9xlarge machine. Based on MosaicML's collaboration with WebDataset prior to launching this product, one would expect their offerings to be identical to WebDataset. Here, Deep Lake is benchmarked against the fastest cloud-native ML-first streaming solutions, most notably WebDataset the previous SOTA, for which ingest and streaming are the only job requirements, making Deep Lake's superior database write and network streaming performance all the more impressive.

4.4 Technique tradeoffs

The benefit of using cloud tools is the wonderful simplicity they can sometimes enable. In this case, massive dataset processing, with 100Gbps networking offers such improved developer productivity and time-to-market advantages over an on-prem, self-hosted solution, that it's a compelling offer.

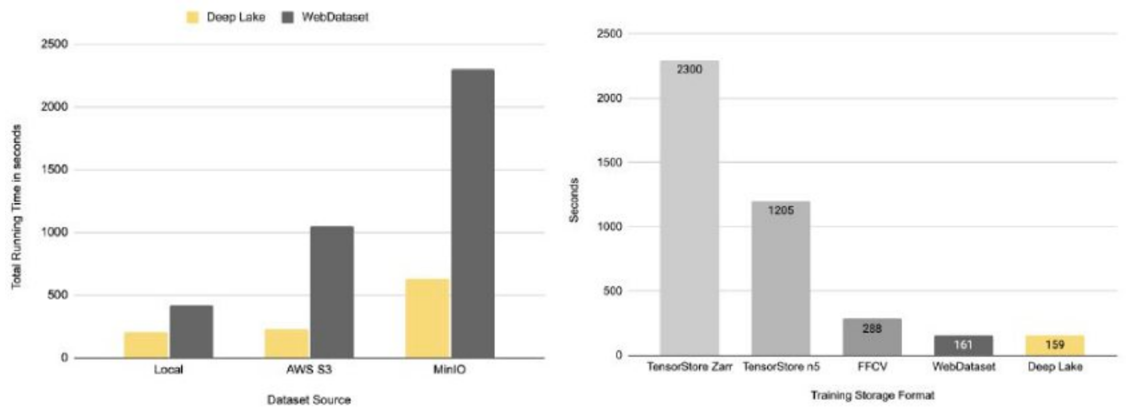


Figure 4.5 (Left) Streaming performance of Deep Lake (in yellow) vs WebDataset and (right) a benchmark of ingestion speed measured by the time to write 10,000 images to the database. Lower is better in both cases. Figure from [17].

However, even with the roughly 2X savings Weka provides over Lustre, or 5X savings it provides on an annual contract, it's still expensive for storage, as shown in figure 4.4. Therefore, workloads must have a compelling reason to migrate to Weka or FSx Lustre when S3 is significantly cheaper. For the most economical multi-modal large model training, it is best to collect and pre-process data locally, on-prem. Then when it is ready for a ML training run, upload the cleaned files to S3 and put Weka in front as a caching layer. Weka will never experience a cache miss due to the predictable nature of PyTorch Dataloaders, for which it is optimized. For on-prem, buying HDDs for on-prem deployments pays for themselves in about 2 months versus S3 (with average drive life of 3 years, offering an 18x cost savings). The same 18x cost savings is roughly true for SSD as well, after on-the-fly compression benefits are factored in. Still, the cost of on-prem infrastructure and dedicated personnel to manage it should never be taken lightly, so both on-prem and cloud solutions offer compelling advantages.

4.5 Future directions

There are still no industry standard methods for data preprocessing and training data serving for machine learning. Data LakeHouses are still not flexible enough for complex datatypes, like n-dimensional vectors, 3D point clouds and media. ML-first databases are still not truly distributed and horizontally scalable and therefore leave room for a single bottleneck and point of failure. Formats like Zarr and HDF5 are powerful, but have a difficult developer experience, painful limitations when used in the cloud and lack integrations with the machine learning toolset. Hopefully innovation in cloud storage will soon ameliorate this pain point.

Another promising innovation are vector databases starting with FAISS [30] and continuing with QDrant [31], Pinecone [32], and Weaviate [33]. Boasting the highest throughput and lowest latency performance is QDrant, even beating Redis and Elasticsearch [34]. Yet, these databases are built for low latency vector similarity search and do not support other database features like ragged arrays, compression, or streaming so it's unclear if they could compete directly with Deep Lake, Zarr and HDF5 for storing complex tensor data.

5 Discussion

Deep Lake achieved 93.9% of the synthetic performance recorded legendary filesystem benchmarking tool `fio` (2.16 GB/s of the 2.3 GB/s achieved with `fio`), showing it can completely saturate a filesystem and leave little room for improvement while still doing meaningful work. ML-first databases like Deep Lake, FiftyOne and DuckDB paired with a cloud object store storage backend offer the best developer experience available today. The benefit of having media and tensor data *inside* the database is that users can visualize their samples, with bounding boxes and other metadata already overlaid on the images to verify data quality, a crucially important observable during research. Furthermore, users can query the data in a SQL-style format, called Tensor Query Language (TQL), and even apply NumPy-style vector slicing on top of the results to prep training runs with, for example, different image resolutions. Finally, keeping the data together enables hyper-efficient streaming to remote GPU nodes with look-ahead caching to make sure the GPUs are always utilized. Together with the excellent developer experience, these ML-first innovations uniquely position Deep Lake as a perfect solution for academics, and if they added truly distributed writes, for industry as well. The dream of wonderful developer experience, with simplicity and scalability, might finally be a reality.

References

[1] "Deep Lake White Paper." https://www.deeplake.ai/whitepaper (accessed May 08, 2023).

[2] S. Sharwood and A. Editor, "NASA to launch 247 petabytes of data into AWS – but forgot about eye-watering cloudy egress costs before lift-off." https://www.theregister.com/2020/03/19/nasa_cloud_data_migration_mess/ (accessed May 07, 2023).

[3] J. Hoffmann *et al.*, "Training Compute-Optimal Large Language Models." arXiv, Mar. 29, 2022. doi: 10.48550/arXiv.2203.15556.

[4] H. Touvron *et al.*, "LLaMA: Open and Efficient Foundation Language Models".

[5] C. Zhou *et al.*, "A Comprehensive Survey on Pretrained Foundation Models: A History from BERT to ChatGPT." arXiv, Feb. 18, 2023. doi: 10.48550/arXiv.2302.09419.

[6] C. Schuhmann *et al.*, "LAION-400M: Open Dataset of CLIP-Filtered 400 Million Image-Text Pairs." arXiv, Nov. 03, 2021. Accessed: Feb. 26, 2023. [Online]. Available: http://arxiv.org/abs/2111.02114

[7] C. Schuhmann *et al.*, "LAION-5B: An open large-scale dataset for training next generation image-text models." arXiv, Oct. 15, 2022. doi: 10.48550/arXiv.2210.08402.

[8] "Self-supervised learning: The dark matter of intelligence." https://ai.facebook.com/blog/self-supervised-learning-the-dark-matter-of-intelligence/ (accessed Feb. 26, 2023).

[9] "Develop on NVIDIA Omniverse Platform," *NVIDIA Developer*, Apr. 29, 2020. https://developer.nvidia.com/nvidia-omniverse-platform (accessed Feb. 26, 2023).

[10] E. Wood *et al.*, "Fake It Till You Make It: Face analysis in the wild using synthetic data alone." arXiv, Oct. 05, 2021. doi: 10.48550/arXiv.2109.15102.

[11] "The Hadoop Distributed File System | IEEE Conference Publication | IEEE Xplore." https://ieeexplore.ieee.org/document/5496972 (accessed May 07, 2023).

[12] "Cloud Object Storage – Amazon S3 – Amazon Web Services," *Amazon Web Services, Inc.* https://aws.amazon.com/s3/ (accessed May 07, 2023).

[13] "Google Cloud Storage," *Google Cloud*. https://cloud.google.com/storage (accessed May 07, 2023).

[14] "Hello from Apache Hudi | Apache Hudi - https://hudi.apache.org/." https://hudi.apache.org/ (accessed May 07, 2023).

[15] "Apache Iceberg - https://iceberg.apache.org/." https://iceberg.apache.org/ (accessed May 07, 2023).

[16] A. Mesbah, A. Rice, E. Johnston, N. Glorioso, and E. Aftandilian, "DeepDelta: learning to repair compilation errors," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, in ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, Aug. 2019, pp. 925–936. doi: 10.1145/3338906.3340455.

[17] "What is the Databricks File System (DBFS)?" https://docs.databricks.com (accessed May 07, 2023).

[18] "Apache Spark^{TM} - Unified Engine for large-scale data analytics - https://spark.apache.org/." https://spark.apache.org/ (accessed Feb. 26, 2023).

[19] D. Ali, "Solving the Challenge of Lots of Small Files (LOSF)," *WEKA*. https://www.weka.io/resources/solution-briefs/solving-the-challenge-of-lots-of-small-files-losf (accessed Feb. 26, 2023).

[20] WEKA, "WEKA Architectural Whitepaper," *WEKA*. https://www.weka.io/resources/white-papers/wekaio-architectural-whitepaper (accessed Feb. 26, 2023).

[21] M. Sharma, "WEKA^{TM} on AWS," *WEKA*. https://www.weka.io/resources/datasheets/wekafstm-on-aws (accessed Feb. 26, 2023).

[22] S. Hambardzumyan *et al.*, "Deep Lake: a Lakehouse for Deep Learning." arXiv, Dec. 13, 2022. doi: 10.48550/arXiv.2209.10785.

[23] "An in-process SQL OLAP database management system," *DuckDB*. https://duckdb.org/ (accessed May 01, 2023).

[24] "lancedb/lancedb." Lance DB, May 01, 2023. Accessed: May 01, 2023. [Online]. Available: https://github.com/lancedb/lancedb

[25] B. Moore, "Introducing FiftyOne: A Tool for Rapid Data & Model Experimentation," *Voxel51*, Sep. 12, 2020. https://medium.com/voxel51/introducing-fiftyone-a-tool-for-rapid-data-model-experimentation-73c85b8406e1 (accessed May 01, 2023).

[26] K. Day, "fastest-filesystem-and-databse-for-multimodal-ML." Apr. 03, 2023. Accessed: Apr. 27, 2023. [Online]. Available: https://github.com/KastanDay/fastest-filesystem-and-databse-for-multimodal-ML

[27] "Elastic Fabric Adapter — Amazon Web Services," *Amazon Web Services, Inc.* https://aws.amazon.com/hpc/efa/ (accessed Apr. 02, 2023).

[28] "DPDK Boosts Packet Processing, Performance, and Throughput," *Intel.* https://www.intel.com/content/www/us/en/communications/data-plane-development-kit.html (accessed Apr. 02, 2023).

[29] T. Edwards, "(22) Getting Started with DPDK on AWS EC2 | LinkedIn - https://www.linkedin.com/pulse/getting-started-dpdk-aws-ec2-thomas-edwards/," 2019. https://www.linkedin.com/pulse/getting-started-dpdk-aws-ec2-thomas-edwards/ (accessed Apr. 02, 2023).
[30] "Faiss." Meta Research, May 09, 2023. Accessed: May 08, 2023. [Online]. Available: https://github.com/facebookresearch/faiss
[31] "Qdrant - Vector Database." https://qdrant.tech/ (accessed May 08, 2023).
[32] "Pinecone - Vector Database for Vector Search," *Pinecone*. https://www.pinecone.io/ (accessed May 08, 2023).
[33] "Weaviate - Vector Database." https://weaviate.io/ (accessed May 08, 2023).
[34] "Vector Database Benchmarks - Qdrant." https://qdrant.tech/benchmarks/ (accessed May 08, 2023).
[35] "libfabric." OpenFabrics Interfaces Working Group, Mar. 23, 2023. Accessed: Apr. 02, 2023. [Online]. Available: https://github.com/ofiwg/libfabric
[36] "Lambda GPU Cloud | Reserved Instances." https://lambdalabs.com/service/gpu-cloud/reserved (accessed Apr. 02, 2023).
[37] *Lambda's Machine Learning Infrastructure Playbook and Best Practices*, (Feb. 23, 2022). Accessed: Apr. 02, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=3EnIW0EZkr4

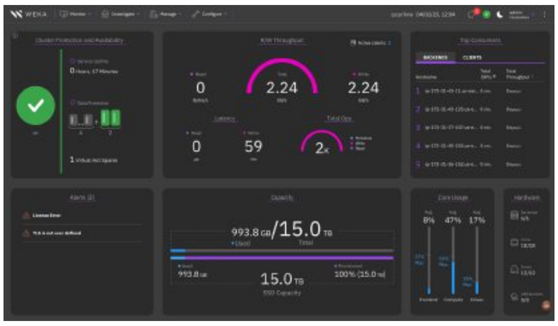
Appendix A. Reproduceable Weka Performance Benchmark on AWS

AWS compute infrastructure.
File servers: 6x AWS i3en.2xlarge.
i3en instances are optimized for high IO, dense, NVMe SSD workloads, with 25 to 100Gbps ethernet and addition AWS EBS networking for a separate inter-node communication layer, which is very useful in Weka's distributed architecture.
Client: 1x c5n-9xlarge. c5n-class instances, where 'n' is for networking, offer a highly converged network stack and extraordinarily fast local "Nitro" SSDs that, like most of the stack, are made in-house by AWS. Additionally, three network technologies enable remote storage to approach local speeds:

ENA: Elastic Network Adapter.
EFA: Elastic Fabric Adapter.
DPDK: Data Plane Development Kit.

It was confirmed that ENA and EFA/DPDK were active using these commands for ENA `$ ethtool -i eth0` documentation, and DPDK networking `$ weka cluster processes`.

ENA is built on the open source libfabric, already widely adopted in MPI-type in HPC environments [35].



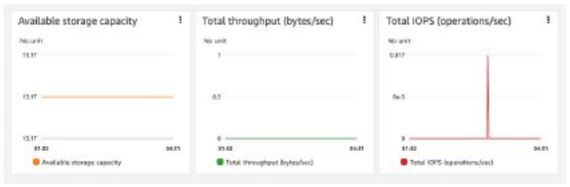
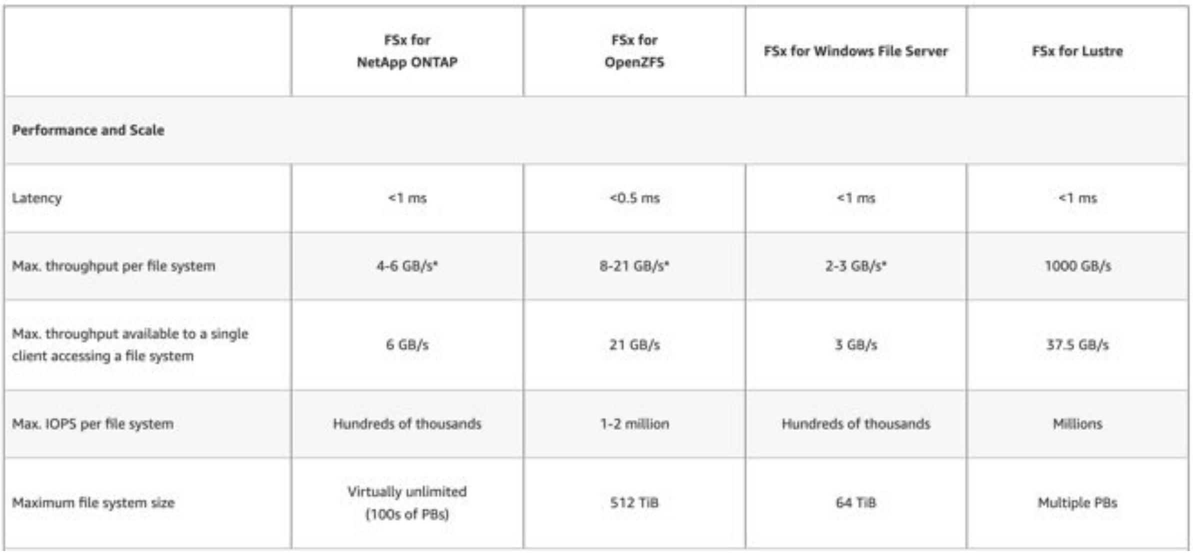


Figure A.1: [Top] the Weka dashboard is exceptionally well designed and provides sufficiently detailed information, like historical throughput and system load. I appreciated how well it surfaced potential bottlenecks (like CPU vs disk utilization). [Bottom] AWS offers very little observability, and one is left to simply trust but verify their performance guarantees, with no ability to diagnose bottlenecks.



	FSx for NetApp ONTAP	FSx for OpenZFS	FSx for Windows File Server	FSx for Lustre
Performance and Scale				
Latency	<1 ms	<0.5 ms	<1 ms	<1 ms
Max. throughput per file system	4-6 GB/s*	8-21 GB/s*	2-3 GB/s*	1000 GB/s
Max. throughput available to a single client accessing a file system	6 GB/s	21 GB/s	3 GB/s	37.5 GB/s
Max. IOPS per file system	Hundreds of thousands	1-2 million	Hundreds of thousands	Millions
Maximum file system size	Virtually unlimited (100s of PBs)	512 TiB	64 TiB	Multiple PBs

Figure A.2 AWS's own comparison of their offered high-performance filesystems. FSx for Lustre has the highest throughput to a single client, and was therefore chosen for these performance tests [36].

Notable in the AI-first training world, many startups are using this niche to undermine AWS's dominance. A remarkable example of this is Lambda Labs, which offers 3200 Gbps networking (or 400 Gbps of bandwidth *per GPU*). This greatly surpasses the 400 Gbps AWS offers with Nvidia GPUs, and is still 4x the maximum AWS offers on their custom "Tranium" AI accelerators in `trn1.32xlarge` instances ("Lambda GPU Cloud"). Interestingly, Lambda Labs also uses Weka as their filesystem provider, and the CEO is quoted as saying "when you're doing large-scale, InfiniBand driven, storage the folks at WEKA have a really phenomenal storage platform for that. Lambda Labs just uses Weka and FreeNAS (Lambda's ML Infrastructure and Best Practices, 2022).