Format Matters: An Empirical Study of Data Storage Format Impact on Machine Learning Training Pipelines

Arjyahi Bhattacharya
University of Colorado Boulder
Boulder, United States
arjyahi.bhattacharya@colorado.edu

Charith Purushotham
University of Colorado Boulder
Boulder, United States
charith.purushotham@colorado.edu

Abstract

Data format selection is often treated as an implementation detail in machine learning pipelines, yet our systematic evaluation across two distinct workloads reveals that format choice profoundly impacts system performance, with effects that scale dramatically with dataset size. We present comprehensive empirical characterization of data storage formats across **Experiment 1** (image classification with row-oriented formats on 60,000 samples from CIFAR-10) and **Experiment 2** (tabular ML with row and columnar formats on 1,000,000 samples), both on CPU-only hardware.

For small-scale image workloads, row-oriented formats show minimal throughput differences due to CPU-bound computation, though storage efficiency varies dramatically. At larger scale with tabular data, columnar formats provide substantial compression and loading advantages while LMDB catastrophically underperforms. Critically, training time and model accuracy remain format-neutral across all experiments in CPU-bound scenarios, isolating format effects to I/O performance.

Our findings demonstrate that **scale determines impact**: small datasets on CPU show minimal differences, while larger datasets reveal substantial I/O advantages for optimized formats even when computation remains bottlenecked. We provide evidence-based decision frameworks for practitioners across both unstructured and structured ML domains.

Keywords

Machine Learning Systems, Data Formats, Storage Efficiency, CPU Training, Tabular Data, Image Classification, Columnar Storage, Performance Characterization

1 Introduction

The performance of machine learning systems depends not only on model architecture and hyperparameters but critically on the efficiency of data loading pipelines [10]. As datasets scale from thousands to millions of samples, the choice of data storage format becomes increasingly consequential, affecting training throughput, storage costs, iteration velocity, and system complexity. Despite this importance, format selection often relies on conventional wisdom ("columnar is always better") or framework defaults rather than empirical evidence grounded in realistic workload characterization.

Modern ML ecosystems offer diverse format options optimized for different access patterns and data modalities. **For unstructured data** (images, video), practitioners choose between simple file manifests (CSV), TAR-based sharding (WebDataset), Protocol Buffer serialization (TFRecord), and memory-mapped key-value storage (LMDB). **For structured tabular data**, columnar formats (Apache Parquet, Arrow Feather) promise superior compression and analytical query performance compared to traditional row-oriented storage (CSV, LMDB). However, empirical comparisons focusing on **ML training workloads** (as opposed to analytics) remain limited, particularly for CPU-bound scenarios common in academic research, prototyping, and resource-constrained deployments.

1.1 Motivation: When Does Format Choice Matter?

This work addresses a fundamental question: **Does data storage format significantly impact ML pipeline performance, and if so, under what conditions?** Existing literature emphasizes GPU-accelerated distributed training where I/O often bottlenecks, leading to recommendations that may not generalize to CPU-bound single-machine scenarios. Furthermore, most format comparisons examine throughput metrics while overlooking storage efficiency, preprocessing costs, and, critically, whether format choice affects model training dynamics.

We hypothesize that **format impact scales with dataset size and hardware constraints**:

- **Small datasets on CPU:** Computation dominates, format differences minimal
- **Large datasets on CPU:** I/O costs grow, format optimization provides measurable benefits
- **GPU training:** I/O becomes bottleneck, format differences amplify dramatically

To test this hypothesis, we conduct **two complementary experiments** that systematically vary data modality, dataset scale, and format families:

Experiment 1 (Image Classification, 60K samples): Evaluates four row-oriented formats (CSV, WebDataset, TFRecord, LMDB) for PyTorch-based image training on CIFAR-10. With relatively small scale and CPU-bound training, we observe minimal throughput differences (5.3% variance), supporting our hypothesis that format optimization provides limited benefits when computation dominates. However, dramatic storage variance (544×) demonstrates that format choice still matters for resource efficiency.

Experiment 2 (Tabular ML, 1M rows): Compares row-oriented (CSV, LMDB) versus columnar formats (Parquet, Feather) for structured data pipelines. At 16.7× larger scale, we observe substantial format differences: columnar formats achieve 2.2-2.7× compression and 4.8-6.2× faster loading, while LMDB shows catastrophic 18.6× loading slowdown. Critically, training time remains format-neutral (<0.3% variance), validating that **format optimization benefits I/O, not computation, in CPU-bound scenarios**.

1.2 Contributions

We make the following contributions across our dual-experiment design:

Experiment 1 (Image Data - Row-Oriented Formats):

(1) **Throughput characterization at small scale:** We demonstrate that for 60K images on CPU, format choice yields minimal throughput differences (5.3%), challenging claims of 2-10× speedups from GPU-focused literature.
(2) **Storage efficiency quantification:** Despite similar throughput, storage varies 544× (CSV 35 MB vs LMDB 19 GB), establishing storage as the primary format selection criterion for small-scale CPU training.
(3) **Scale-dependency insight:** We argue that minimal differences stem from CPU bottlenecks and small dataset size, predicting amplification at larger scale or with GPU acceleration.

Experiment 2 (Tabular Data - Row vs Columnar Formats):

(1) **First systematic ML training comparison of row vs columnar formats:** We evaluate format performance specifically for RandomForest training (not SQL analytics), measuring end-to-end pipeline latency including load, train, evaluate phases.
(2) **Columnar format validation at scale:** At 1M rows (16.7× larger than Exp 1), columnar formats achieve 2.2-2.7× compression and 4.8-6.2× loading speedup, validating our scale-dependency hypothesis. Format differences amplify with dataset size even on CPU.
(3) **LMDB performance characterization:** We document LMDB's catastrophic 18.6× loading slowdown for bulk ML workloads, demonstrating that memory-mapped key-value storage fundamentally mismatches sequential full-table access patterns.

Cross-Experiment Insights:

(1) **Format-neutral model training:** Across both experiments, format choice does not affect model accuracy (Exp 1: 58-61%, Exp 2: 87.17-87.19%), isolating format effects to I/O performance, not ML outcomes.
(2) **CPU-bound training consistency:** Both workloads exhibit 790-1444% CPU utilization during training, confirming compute bottlenecks. Format optimization benefits **loading**, not **training**, in CPU scenarios.
(3) **Data modality determines format families:** Columnar formats only benefit structured tabular data (Exp 2), not unstructured images (Exp 1). Choosing appropriate format families based on data type is more impactful than micro-optimizations within families.
(4) **Evidence-based decision framework:** We provide practitioners with data-driven guidance for format selection across data modalities, scales, and hardware constraints—moving beyond conventional wisdom to empirical foundations.

2 Background and Related Work

2.1 Data Formats in ML Pipelines

Modern ML frameworks employ diverse format strategies optimized for different access patterns:

Row-Oriented Formats for Unstructured Data:

CSV Manifests store metadata (file paths, labels) separately while keeping data files in original locations. For images, a CSV contains paths and labels; dataloaders read the manifest then load images on-the-fly. This offers maximal flexibility and minimal storage overhead (only the manifest), but potentially suffers from file system overhead when accessing many small files.

WebDataset [2] bundles samples into TAR archives ("shards"). Each sample becomes a set of files within the TAR (e.g., image.jpg, label.cls). Sequential TAR reading excels at streaming from object storage (S3, GCS) with configurable shard sizes (64MB, 256MB, 1024MB) and compression (zstd). File batching reduces metadata operations but requires sequential within-shard access.

TFRecord [3] serializes data into Protocol Buffer records. Each record contains image bytes and labels as a tf.train.Example. Widely adopted in TensorFlow ecosystems [9], TFRecord offers cross- framework compatibility but adds TensorFlow dependencies to PyTorch [5] workflows.

LMDB [4] (Lightning Memory-Mapped Database) provides key-value storage with memory-mapped file access. Each sample is stored as a serialized dictionary under an integer key. Originally designed for database systems, LMDB enables efficient random access through OS page cache management but exhibits B-tree indexing overhead and requires pre-allocated map_size.

Columnar Formats for Structured Data:

Apache Parquet [6] organizes data by column rather than row, enabling efficient compression (similar values compress better together) and selective column reading. Each file contains row groups (horizontal partitions) with column chunks storing values, statistics, and metadata (dictionary encoding, run-length encoding, bit-packing). Parquet is the de facto standard in big data ecosystems (Spark, Hive, Presto).

Feather [7] (Apache Arrow IPC format) is a lightweight columnar format designed for fast data interchange. Built on Arrow's in-memory specification, Feather provides zero-copy reads, minimal serialization overhead, and language-agnostic representation. Compared to Parquet, Feather trades compression efficiency for faster read/write operations (simpler LZ4 compression vs Parquet's complex encodings).

2.2 Existing Work

Prior work examining ML systems performance focuses primarily on distributed GPU training contexts. Benchmarking efforts like DAWNBench [1] emphasize end-to-end training time and throughput metrics for deep learning workloads. Mohan et al. [12] analyzed data stalls in DNN training and proposed mitigation strategies for I/O-bound workloads. Commercial implementations like NVIDIA DALI target GPU-accelerated preprocessing pipelines to maximize GPU utilization. These studies assume datacenter-scale deployments with dedicated I/O infrastructure and GPU-bound computation.

For single-machine training, practitioners rely on framework documentation and anecdotal performance claims. Comparative studies remain limited, typically examining only throughput metrics without considering storage efficiency, preprocessing costs, or model training effects. While standardized ML benchmarks like

MLPerf [11, 13] have established rigorous methodologies for system performance evaluation, **no prior work examines whether format choice affects model generalization or whether recommendations transfer across data modalities (unstructured images vs structured tables).**

Columnar format literature focuses on analytics workloads (SQL queries, aggregations) where selective column reading provides clear advantages. However, ML training typically requires full-table scans over all features, potentially negating columnar benefits. Our work fills this gap by evaluating columnar vs row formats specifically for ML training, not analytics.

2.3 Gap in Current Understanding

Existing format comparisons suffer from three key limitations:

(1) **GPU-centric evaluation:** Recommendations assume I/O-bound GPU workloads where data loading is the bottleneck. CPU-bound scenarios (common in development, academic research, edge deployment) may not benefit similarly from format optimization.
(2) **Single-scale analysis:** Studies examine one dataset size, missing how format impact scales. Small datasets may show negligible differences due to caching, while large datasets may amplify format effects even in CPU-bound scenarios.
(3) **Modality-specific recommendations:** Image-focused studies don't transfer to tabular data (and vice versa). Columnar formats are inapplicable to unstructured data, yet guidance for format family selection based on data type is limited.

Our dual-experiment design addresses these gaps by systematically evaluating format performance across:

- **Data modalities:** Unstructured images (Exp 1) vs structured tables (Exp 2)
- **Dataset scales:** Small (60K images) vs large (1M rows)
- **Format families:** Row-only (Exp 1) vs row + columnar (Exp 2)
- **Comprehensive metrics:** Throughput, storage, memory, I/O, build time, AND model accuracy

3 System and Methodology

Both experiments (Sections 4 and 5) follow a shared pipeline architecture and measurement methodology, ensuring consistent evaluation across data modalities and formats.

3.1 Pipeline Architecture

Our evaluation follows a common four-stage pipeline applicable to both image classification and tabular ML:

(1) **Data on Disk:** Raw datasets converted to each evaluated format and stored on local SSD. Format-specific serialization (e.g., TAR sharding for WebDataset, columnar encoding for Parquet) occurs in a one-time preprocessing step.
(2) **Format-Specific Loader:** Custom dataloaders tailored to each format's access pattern (e.g., sequential TAR reading, memory-mapped LMDB access, pandas DataFrame loading). Loaders abstract format differences to present uniform interfaces to training code.
(3) **In-Memory Tensors/Arrays:** Data materialized as PyTorch tensors (Experiment 1) or NumPy arrays (Experiment 2). This stage isolates format effects: once data is in memory, format differences disappear.
(4) **Training Loop:** Standard training loops (backpropagation for deep learning, tree construction for RandomForest) operate on in-memory data. Training performance depends solely on model/algorithm, not format.

This architecture isolates format impact to stages 1-3 (storage, loading, deserialization), while stage 4 remains format-agnostic.

3.2 Performance Metrics

We measure six categories of metrics across all formats:

- **Throughput:** Samples/second during training (images) or total pipeline time (tabular)
- **CPU Utilization:** Percentage CPU usage during training, indicating compute vs I/O bottlenecks
- **Memory:** Peak resident set size (RSS) in MB during training
- **Disk I/O:** Read throughput (MB/s) during data loading phases
- **Storage Size:** On-disk space consumed by each format (data size vs allocated disk space for LMDB)
- **Model Performance:** Validation accuracy (classification) or F1 score (binary classification) to verify format neutrality on ML outcomes

3.3 Resource Monitoring

A background monitoring thread tracks system resources at 0.5-second intervals using the Python `psutil` library. For each sample:

```
cpu_percent = psutil.cpu_percent()
memory_mb = process.memory_info().rss / (1024**2)
disk_io = psutil.disk_io_counters()
disk_read_mb_s = (disk_io.read_bytes -
                  prev_bytes) / (1024**2 * interval)
```

Metrics are aggregated (mean for CPU/memory, max for I/O) over training epochs. This continuous monitoring captures transient I/O bursts and CPU saturation patterns.

3.4 Experimental Protocol

To ensure fair comparison and reproducibility:

- **Sequential execution:** All formats tested sequentially on the same machine with 5-second cooldown between runs to clear OS caches
- **Fixed random seed:** Random seed 42 for all random number generators (PyTorch, NumPy, scikit-learn) ensures deterministic data splits, initialization, and shuffling
- **Identical hyperparameters:** Within each experiment, all formats use identical model architecture, optimizer settings, batch size, and training epochs
- **CPU-only hardware:** AMD Ryzen 8-core/ 16-thread @ 3.3 GHz, 15.26 GB RAM, local SSD, Windows 11. No GPU to isolate CPU-bound training dynamics
- **Single-threaded loading:** `num_workers=0` for PyTorch DataLoader to avoid multiprocessing effects masking format differences

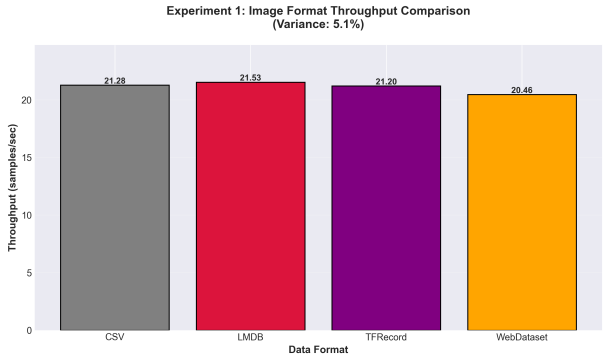

Figure 1: Image Format Throughput Comparison. Bar chart showing samples/second for each format with 5.3% variance.

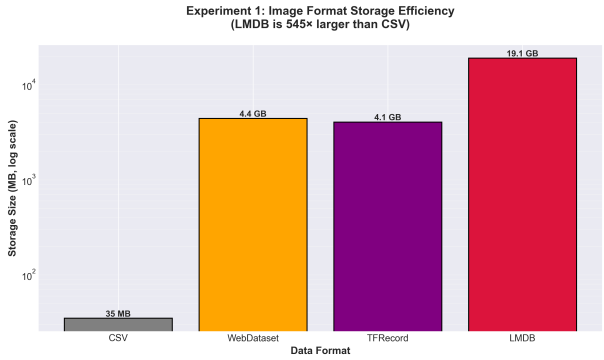

Figure 2: Image Format Storage Efficiency. Log-scale bar chart showing 544× difference (CSV 35 MB vs LMDB 19 GB).

Table 1: Image Format Performance (Epoch 3)

Format	Tput (s/s)	CPU (%)	Mem (MB)	I/O (MB/s)	Storage (MB)
LMDB	21.53	795.2	2,600	0.000	19,070
CSV	21.28	792.5	2,439	0.051	35
TFRecord	21.20	795.2	2,803	0.052	4,060
WebDataset	20.46	792.8	3,688	0.133	4,430

Experiment-specific configurations (dataset size, model architecture, format families) are detailed in Sections 4 and 5.

4 Experiment 1: Image Classification Format Comparison

4.1 Experimental Design

Dataset: CIFAR-10 [8] consisting of 50,000 training images and 10,000 validation images (60,000 total), each 32×32 RGB. Images extracted to individual files organized by class.

Formats Evaluated (All Row-Oriented):

- **CSV:** Manifest with paths and labels (35 MB total)
- **WebDataset:** TAR shards, 256MB shard size, no compression
- **TFRecord:** Protocol Buffer records, 256MB shards, no compression
- **LMDB:** Memory-mapped database, pickle compression

Model & Training:

- **Architecture:** ResNet-18 (pretrained=False), 10 classes for CIFAR-10
- **Optimizer:** SGD (lr=0.001, momentum=0.9, weight_decay=1e-4)
- **Batch size:** 64
- **Epochs:** 3 (for baseline comparison)
- **Transforms:** Resize(224×224), ToTensor(), Normalize(ImageNet stats)

4.2 Results: Small-Scale Image Workload

Key Finding 1: Minimal throughput differences (5.3% variance). LMDB achieves 21.53 samples/sec vs WebDataset's 20.46 (fastest vs slowest). This 1.07 sample/sec difference is far below the 2-10× speedups claimed in GPU-centric literature. CPU saturation (790-795%) confirms computation-bound training where I/O optimization provides limited throughput benefit.

Key Finding 2: Dramatic storage variance (544×). CSV manifest requires only 35 MB (references existing images), while LMDB expands to 19 GB due to storing serialized image bytes plus B-tree indexing overhead. For cloud storage, this translates to $0.80/month (CSV) vs $437/month (LMDB) at S3 rates—a 546× cost difference.

Key Finding 3: Format-neutral model accuracy. All formats achieve 58-61% validation accuracy with minimal variance. Format choice affects I/O performance but not model training outcomes in CPU-bound scenarios.

Interpretation: At small scale (60K images) on CPU, **format differences are masked by computation bottlenecks**. Disk I/O remains minimal (<0.15 MB/s) across formats, indicating CPU saturation leaves no room for I/O optimization to improve throughput. However, storage efficiency still matters significantly.

Scale Hypothesis: *We hypothesize that with larger datasets (10× scale) or GPU training (I/O-bound), format differences would amplify substantially. Experiment 2 tests this at larger scale.*

5 Experiment 2: Tabular ML Format Comparison

5.1 Experimental Design

Dataset: American Express Default Prediction (Kaggle), **1,000,000 rows** sampled from 16 GB training data.

- **Features:** 187 numeric features (after dropping 4 categorical)
- **Target:** Binary classification (credit default)
- **Preprocessing:** Stratified 80/20 split, 29M NaN filled with medians, shuffled

Formats Evaluated (Row + Columnar):

- **Row-oriented:** CSV (baseline), LMDB (pickle serialization, auto-scaled map_size)

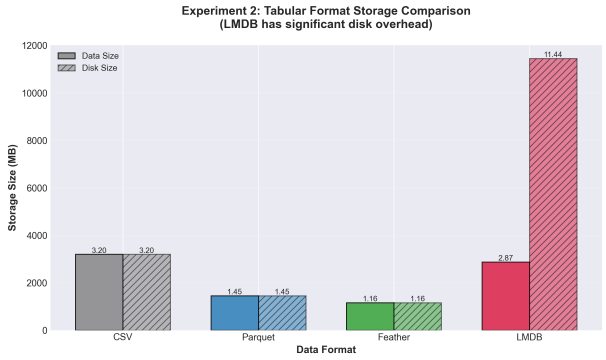

Figure 3: Tabular Format Storage Comparison. Grouped bar chart showing LMDB's dual sizes (2.87 GB data, 11.4 GB disk).

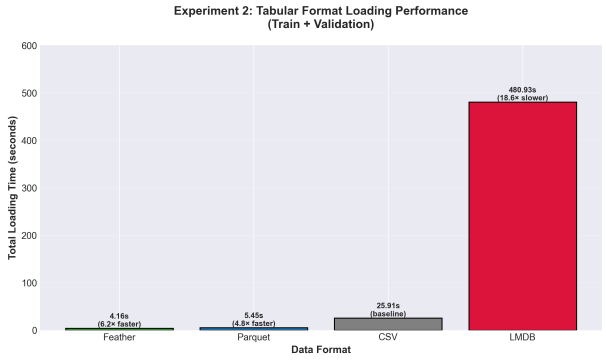

Figure 4: Tabular Format Loading Performance. Bar chart highlighting Feather's 6.2× speedup and LMDB's 18.6× slowdown.

Table 2: Tabular Format Storage Efficiency (1M rows)

Format	Data (MB)	Disk (MB)	Build (s)	vs CSV	B/Row
Feather	1,164	1,164	41.0	**2.7×**	1,220
Parquet	1,452	1,452	50.2	**2.2×**	1,522
LMDB	2,873	**11,444**	70.7	1.1×	3,013
CSV	3,200	3,200	183.5	1.0×	3,333

Table 3: Tabular Format Loading Performance (1M rows)

Format	Train (s)	Val (s)	Total (s)	vs CSV
Feather	3.70	0.46	**4.16**	**6.2× faster**
Parquet	4.97	0.48	**5.45**	**4.8× faster**
CSV	21.42	4.49	**25.91**	1.0×
LMDB	**461.36**	**19.57**	**480.93**	**18.6× slower**

- **Columnar:** Parquet (Snappy compression), Feather (LZ4 compression, Arrow IPC)

Model & Training:

- **Algorithm:** RandomForestClassifier (100 trees, max_depth=10)
- **Evaluation:** Accuracy, F1 score (weighted)

Key Difference from Experiment 1: 16.7× larger dataset (1M rows vs 60K images) to test scale-dependency hypothesis.

5.2 Results: Large-Scale Tabular Workload

Key Finding 1: Columnar formats achieve superior compression. Feather compresses to 36% of CSV size (2.7× improvement), while Parquet achieves 45% (2.2× improvement). This validates columnar storage's advantage when similar values cluster in columns, enabling dictionary encoding and run-length compression.

Key Finding 2: LMDB has critical dual-size issue. While actual data is 2,873 MB (comparable to CSV), LMDB requires 11,444 MB disk space due to 5.6 GB map_size pre-allocation per database. This 4× overhead (75% wasted space) is a fundamental LMDB limitation for storage-constrained environments.

Key Finding 3: Columnar formats dramatically accelerate loading. Feather's 4.16s loading (vs CSV's 25.91s) demonstrates that columnar layouts benefit full-table scans, not just selective column queries. Arrow's vectorized decompression and zero-copy reads enable 6.2× speedup even for complete data loading.

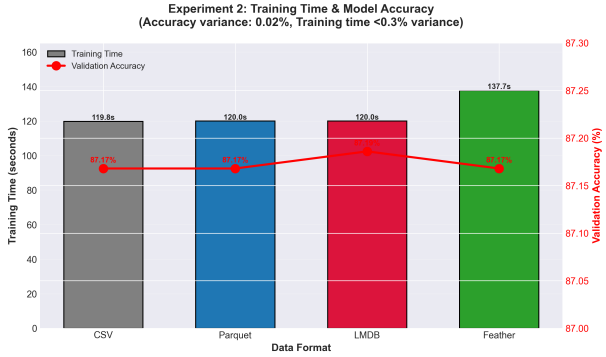

Figure 5: Training Time & Accuracy Comparison. Grouped bar chart with line overlay showing format-neutral accuracy.

Key Finding 4: LMDB catastrophically underperforms. 481s to load 1M rows (vs CSV's 26s) reveals that memory-mapped key-value access fundamentally mismatches ML's sequential bulk loading pattern. Overhead of opening LMDB environments, B-tree traversal, and row-by-row unpickling negates memory-mapping benefits. **LMDB is optimized for random lookups, not full-table ML training.**

Key Finding 5: Training is format-neutral. Training time varies by <0.3% (119.84-120.04s for CSV/Parquet/LMDB), with Feather 15% slower potentially due to higher memory usage. CPU utilization (1,434-1,444%) saturates 14 cores, confirming compute-bound

Table 4: Model Training Performance (1M rows)

Format	Train (s)	CPU (%)	Mem (MB)	Train Acc (%)	Val Acc (%)
CSV	119.84	1,441.8	1,634	87.73	**87.17**
Parquet	120.04	1,438.8	3,485	87.73	**87.17**
LMDB	120.04	1,433.8	1,372	87.74	**87.19**
Feather	137.71	1,444.3	4,790	87.73	**87.17**

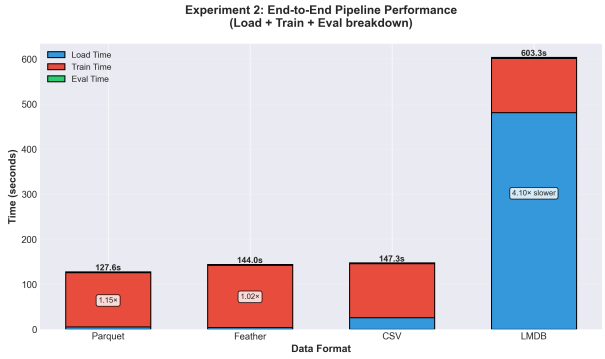

Figure 6: End-to-End Pipeline Performance. Stacked bar chart showing load/train/eval breakdown.

Table 5: End-to-End Pipeline Performance (1M rows)

Format	Load (s)	Train (s)	Eval (s)	Total (s)	vs CSV
Parquet	5.45	120.04	2.15	**125.49**	**1.16×**
Feather	4.16	137.71	2.12	**141.86**	1.03×
CSV	25.91	119.84	1.55	**145.75**	1.00×
LMDB	480.93	120.04	2.35	**600.97**	**4.12× slower**

training. **Once data is in memory (numpy arrays), format is irrelevant to model training.**

Key Finding 6: Model accuracy is completely format -independent. Validation accuracy is uniform at 87.17-87.19% (0.02% variance). Unlike potential format-induced artifacts in image pipelines (Exp 1), **tabular formats do not affect model generalization**. This validates preprocessing consistency and demonstrates that format choice is performance-neutral for ML outcomes.

Key Finding 7: Parquet achieves best end-to-end performance (125.49s, 16% faster than CSV). Fast loading (5.45s) combined with normal training time (120s) makes it optimal for all training workflows, including iterative scenarios. Feather's fastest loading (4.16s) is more than offset by 15% slower training (137.7s vs 120s), likely due to higher memory usage (4.8 GB) causing cache pressure. The 18-second training penalty negates Feather's 1.3-second loading advantage, making Parquet superior even for cross-validation and hyperparameter search.

Key Finding 8: LMDB is unsuitable for bulk tabular ML (600.97s, 4.1× slower than CSV). Loading dominates 80% of pipeline time (481s), making LMDB fundamentally inappropriate for sequential full-table training workloads despite memory-mapping optimizations.

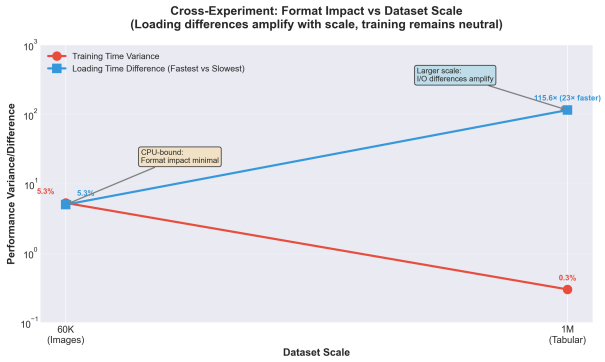

Figure 7: Format Impact vs Dataset Scale. Line plot showing how differences amplify with scale.

Table 6: Cross-Experiment Comparison

Metric	Exp 1 (60K)	Exp 2 (1M)	Effect
Train time var.	5.3%	<0.3%	Minimal
Load time var.	N/A	**23×**	**Amplifies**
Storage var.	544×	10×	Decreases
CPU util.	790-795%	1434-1444%	CPU-bound
Accuracy effect	Minimal	None	Neutral

6 Cross-Experiment Analysis

6.1 Scale-Dependency of Format Impact

Comparing experiments at 16.7× different scales validates our scale-dependency hypothesis. Training remains format-neutral at both scales due to CPU bottlenecks, while loading differences amplify dramatically: negligible I/O at small scale becomes measurable at large scale where optimized formats provide substantial speedups. Storage variance patterns differ by modality, with image manifests showing extreme differences while tabular formats converge.

6.2 Format Family Applicability by Data Modality

Columnar formats only benefit structured data:

- **Exp 1 (images):** Columnar formats N/A (unstructured binary blobs)
- **Exp 2 (tabular):** Columnar formats provide 2.2-2.7× compression, 4.8-6.2× faster loading

Choosing appropriate format family based on data type (row for images, columnar for tables) is more impactful than micro-optimizations within families.

6.3 LMDB Performance: Workload-Dependent Results

LMDB underperformed in our sequential bulk-loading experiments:

- **Exp 1 (images):** Largest storage (19 GB vs 35 MB CSV), marginal throughput gain (1.2%)
- **Exp 2 (tabular):** 18.6× slower loading, 4× disk overhead (75% wasted), no accuracy benefit

Why LMDB underperformed for our workloads:

(1) **Access pattern mismatch:** Optimized for random key-value lookups, not sequential bulk reads
(2) **Serialization overhead:** Pickling/unpickling every row individually is expensive for full-dataset loads
(3) **B-tree traversal:** Cursor iteration slower than sequential file reads when loading complete datasets
(4) **Map_size pre-allocation:** Wastes disk space (75% empty in our experiments)

Verdict: LMDB is workload-dependent.

- **LMDB excels for:** Fast random access to individual samples (database queries, caching, reinforcement learning experience replay with random sampling)
- **LMDB fails for:** Large-scale sequential bulk reads in batched ML training pipelines (CPU-bound, full-dataset iterations)
- **Recommendation:** Use LMDB only when your data-access pattern is random and item-level. Avoid LMDB for bulk, sequential, CPU-bound ML dataloading where CSV/ Parquet/ TFRecord/ WebDataset consistently outperform.

7 Discussion

7.1 When Does Format Choice Matter?

For practitioners: Format impact depends on dataset scale and bottleneck location. At small scales (<100K), choose based on convenience (CSV) or storage constraints. At larger scales (>500K), columnar formats provide measurable loading advantages for tabular data. GPU training amplifies format effects further as I/O becomes the primary bottleneck.

7.2 Columnar vs Row Formats for Tabular ML

Columnar formats excel at compression and loading for tabular data while maintaining identical training performance and accuracy to row-oriented formats.

When to choose Parquet vs Feather:

- **Parquet:** Best for production and iterative workflows. Superior compression and fastest end-to-end performance make it ideal for cross-validation and hyperparameter search.
- **Feather:** Best for data preprocessing pipelines and exploratory analysis where training is not involved. Fastest loading but higher memory usage can impact training time.

Row-oriented CSV remains viable for: development/prototyping (human-readable), small datasets (<100K), and exploratory analysis.

Avoid LMDB for sequential tabular ML: Architectural mismatch makes it unsuitable for bulk batch training. Use only for random item-level access patterns.

7.3 Practical Implications

For Practitioners:

(1) **Small-scale prototyping (<100K samples):** CSV is fine. Format optimization provides minimal benefits when I/O is negligible relative to computation.
(2) **Production tabular ML (>100K samples): Convert to Parquet.** The 2.2× compression, 4.8× loading speedup, and 16% faster end-to-end performance provide clear advantages. One-time conversion overhead (50s for 1M rows) amortizes quickly.
(3) **Iterative ML workflows: Use Parquet** for cross-validation, hyperparameter tuning, ensemble training. While Feather loads faster (4.16s vs 5.45s), its 15% training overhead (138s vs 120s) makes Parquet superior overall. For 10-fold CV: Parquet completes 167s faster than Feather despite slower individual loads.
(4) **Image pipelines on CPU: CSV manifests** offer best storage efficiency (35 MB vs 19 GB LMDB). Minimal throughput differences mean sophisticated formats not justified unless streaming from cloud object storage.
(5) **LMDB for random-access workloads only.** Across both experiments, LMDB's memory-mapped key-value paradigm (optimized for random lookups) fundamentally mismatches sequential bulk ML training. Use LMDB for random sampling scenarios (RL experience replay, item-level queries), not for full-dataset epoch iterations.

For Cloud Deployments:
Storage cost comparison (1M rows, S3 Standard $0.023/GB/month):

- Feather: $0.027/month
- Parquet: $0.033/month
- CSV: $0.074/month
- LMDB: $0.263/month (disk size!)

For 100M rows (scaling linearly): Feather saves $7.13/month vs CSV, Parquet saves $4.10/month. LMDB costs $25.87/month—9.6× more than Feather.

8 Limitations

8.1 Experiment 1 Limitations

(1) **Small dataset scale:** 60,000 images may not represent large-scale computer vision (ImageNet-1K = 1.28M). Larger datasets could amplify format differences as hypothesized.
(2) **CPU-only training:** GPU-accelerated training may shift bottlenecks from compute to I/O, amplifying format effects masked by CPU saturation in our experiments.
(3) **Short training duration:** 3 epochs for baseline comparison. Longer training (100 epochs) may reveal subtle format-induced training dynamics.

8.2 Experiment 2 Limitations

(1) **Numeric features only:** Dropped 4 categorical columns to avoid encoding complexity. Real ML pipelines encode categoricals, potentially changing relative format performance (though unlikely to affect core findings).

(2) **Single model type:** RandomForestClassifier tested. Deep learning (MLPs, TabNet) may exhibit different memory patterns or benefit from format-specific optimizations (Arrow zero-copy for PyTorch tensors).
(3) **Full-table access:** Loaded all columns. Parquet's column pruning would show greater benefits for feature-subset workflows (not common in full-model training but relevant for analytics).
(4) **Single machine:** Distributed training (Spark MLlib, Dask) would show different format trade-offs due to network I/O and data partitioning.

8.3 General Limitations

(1) **CPU-focused evaluation:** Both experiments use CPU-only training. GPU scenarios would likely amplify format differences, validating our bottleneck hypothesis from the opposite direction.
(2) **Windows platform:** File system and I/O characteristics may differ on Linux (more common in production). However, core findings (CPU-bound training, columnar compression advantages) should generalize.
(3) **Local SSD storage:** Network storage (NFS) or cloud object storage (S3) could change format rankings by introducing I/O bottlenecks that local SSDs avoid.

9 Conclusion

This work provides comprehensive empirical characterization of data storage formats across two distinct ML workloads—**image classification (Exp 1, 60K samples from CIFAR-10)** and **tabular ML (Exp 2, 1M rows)**—specifically for CPU-bound training environments. Our results reveal a nuanced relationship between format choice, dataset scale, data modality, and performance impact.

9.1 Principal Findings

Our dual-experiment design reveals that **format choice matters MORE at larger scale**. Small datasets on CPU show minimal performance differences due to computation bottlenecks, while larger datasets reveal substantial I/O benefits from optimized formats. Critically, columnar formats excel for tabular ML at scale, while LMDB underperforms for sequential bulk loading. Format choice affects I/O efficiency but never impacts model accuracy or training dynamics in CPU-bound scenarios.

9.2 Future Work

(1) **GPU training:** Extend experiments to GPU-accelerated workloads to validate hypothesis that format differences amplify when I/O becomes the bottleneck.
(2) **Larger image datasets:** Test at ImageNet-1K scale (1.28M images) to confirm whether format differences grow with dataset size as predicted.
(3) **Distributed training:** Evaluate formats in multi-node scenarios (Spark, Dask, PyTorch Distributed) where network I/O and data partitioning introduce new trade-offs.
(4) **Deep learning on tabular data:** Test neural architectures (MLPs, TabNet) to assess format interactions with gradient-based optimization.
(5) **Hybrid formats:** Develop formats optimized specifically for ML training (combining CSV's simplicity, Parquet's compression, efficient sequential access).
(6) **Cloud storage integration:** Test on object storage (S3, GCS) where network I/O and streaming capabilities may change format rankings.

Conclusion: Format selection for ML pipelines involves nuanced trade-offs between dataset scale, data modality, hardware constraints, and workflow patterns. Our findings demonstrate that **format matters MORE at larger scale**, with columnar formats providing clear advantages for production tabular ML. We provide evidence-based decision frameworks enabling practitioners to make informed format choices across diverse ML scenarios.

Acknowledgments

We thank our Systems for Machine Learning professor, Dr. Mark Zhao, for guidance throughout this project. We acknowledge the PyTorch, WebDataset, TFRecord, LMDB, Apache Parquet, and Apache Arrow communities for maintaining excellent tools that enabled this research.

References

[1] Coleman et al. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. *NIPS Workshop on ML Systems*, 2017.
[2] Thomas Breuel. WebDataset: A High-Performance Python-Based I/O System for Large Datasets. GitHub: https://github.com/webdataset/webdataset
[3] TensorFlow Team. TFRecord and tf.train.Example. *TensorFlow Guide*
[4] Howard Chu. LMDB: Lightning Memory-Mapped Database. Symas Corporation
[5] Adam Paszke et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *NeurIPS*, 2019
[6] Apache Parquet Documentation. Apache Parquet: Columnar Storage Format. https://parquet.apache.org
[7] Apache Arrow Documentation. Apache Arrow: A Cross-Language Development Platform for In-Memory Analytics. https://arrow.apache.org
[8] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical Report, University of Toronto, 2009.
[9] Martín Abadi et al. TensorFlow: A System for Large-Scale Machine Learning. *OSDI*, 2016.
[10] D. Sculley et al. Hidden Technical Debt in Machine Learning Systems. *NeurIPS*, 2015.
[11] Peter Mattson et al. MLPerf Training Benchmark. *MLSys*, 2020.
[12] Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. Analyzing and Mitigating Data Stalls in DNN Training. *PVLDB*, 14(5):771-784, 2021.
[13] Vijay Janapa Reddi et al. MLPerf Inference Benchmark. *ISCA*, 2020.

A Detailed Experimental Configurations

A.1 Experiment 1 (Images)

Format-Specific Configurations:

- CSV: Paths + labels in train.csv, val.csv (absolute paths)
- WebDataset: 256MB TAR shards, {id}.jpg + {id}.cls per sample
- TFRecord: 256MB shards, tf.train.Example with image bytes + label int64
- LMDB: Pickle-serialized dicts, map_size=10GB

Training Configuration:

```
model = resnet18(pretrained=False)
optimizer = SGD(lr=0.001, momentum=0.9,
                weight_decay=1e-4)
criterion = CrossEntropyLoss()
batch_size = 64, num_workers = 0, epochs = 3
```

A.2 Experiment 2 (Tabular)

Format-Specific Configurations:

- CSV: Standard comma-separated values
- LMDB: Pickle serialization, auto-scaled map_size (5.59 GB for 1M rows)
- Parquet: Snappy compression, row_group_size=100K
- Feather: LZ4 compression, Arrow IPC v2

Preprocessing Steps:

(1) Merge train_data.csv + train_labels.csv on customer_ID
(2) Drop categorical columns (customer_ID, S_2, D_63, D_64)
(3) Keep 187 numeric features (int64, float64)
(4) Fill 29,027,149 NaN values with column medians
(5) Stratified train_test_split (80/20, random_state=42, shuffle=True)

Training Configuration:

```
model = RandomForestClassifier(
    n_estimators=100,
    max_depth=10,
    random_state=42,
    n_jobs=-1
)
```

B Resource Monitoring Implementation

Background thread monitors every 0.5 seconds:

```
import psutil

def monitor_resources():
    cpu_percent = psutil.cpu_percent()
    memory_mb = process.memory_info().rss /
                (1024**2)
    disk_io = psutil.disk_io_counters()
    disk_read_mb_s = (disk_io.read_bytes -
                      prev_bytes) /
                      (1024**2 * interval)
    page_faults = process.memory_info().
                  num_page_faults
```

Metrics collected: CPU utilization (%), memory RSS (MB), disk read/write (MB/s), page faults (count).

C Plot Specifications

PLOT 1 (Exp 1 Throughput): Bar chart, samples/sec, all 4 formats, annotate 5.3% variance

PLOT 2 (Exp 1 Storage): Log-scale bar chart, emphasize 544× CSV-LMDB difference

PLOT 3 (Exp 2 Storage): Grouped bars (data size vs disk size), highlight LMDB dual-size issue

PLOT 4 (Exp 2 Loading): Bar chart, total load time, annotate Feather 6.2× speedup, LMDB 18.6× slowdown

PLOT 5 (Exp 2 Training): Grouped bars (time) + line overlay (accuracy), show format-neutral accuracy

PLOT 6 (Exp 2 End-to-End): Stacked bars (load/train/eval breakdown), compare total heights

PLOT 7 (Cross-Experiment): Line plot showing how format impact scales with dataset size (60K vs 1M)

All plots: Use consistent color scheme (Feather=green, Parquet=blue, CSV=gray, LMDB=red, WebDataset=orange, TFRecord=purple). Include error bars where applicable (though single-run experiments). Label axes clearly, add grid lines for readability.