

# AI-Driven Data Lake Optimization: Integrating Quality Monitoring with Intelligent Physical Design Decisions

Sowjanya Deva<sup>\*</sup>, Surya Narayana Reddy Chintacunta<sup>†</sup>

Independent Researcher, MPS in Data Science, University of Maryland Baltimore County, Baltimore, MD 21250, USA

Email(s): [deva20829@gmail.com](mailto:deva20829@gmail.com) (S. Deva), [surya.nreddy.ds@gmail.com](mailto:surya.nreddy.ds@gmail.com) (S. Chintacunta),

\*Corresponding author: Sowjanya Deva, Jersey City, NJ, 07306, [deva20829@gmail.com](mailto:deva20829@gmail.com)

**ABSTRACT:** Cloud data lakes require continuous optimization across multiple dimensions: physical design (partitioning, compression), query execution, and data quality assurance. This paper presents AIDALOS (AI-Driven Autonomous Data Lake Optimization System), a framework that integrates quality monitoring with physical optimization decisions. The system uses reinforcement learning to adapt monitoring intensity and trigger physical design changes based on detected anomalies, drift patterns, and workload shifts. Deep Q-networks learn when to repartition tables, ensemble models select compression codecs based on data characteristics and access patterns, and neural cost estimators improve query plan selection. Our evaluation across five machine learning pipelines demonstrates that this integrated approach achieves 47% storage cost reduction and 62% query performance improvement compared to static configurations, with 89.9% F1-score for quality issue detection. The key insight is that quality signals drift detection, anomaly patterns, and workload changes should directly inform physical optimization decisions rather than treating these as separate concerns.

**KEYWORDS:** Data Lake Optimization, Machine Learning, Reinforcement Learning, Data Quality Monitoring, Physical Database Design, Drift Detection

## 1. Introduction

### 1.1. Motivation and Problem Statement

Cloud data lakes store petabyte-scale datasets for analytics and machine learning, but maintaining optimal performance requires continuous decisions about physical design, resource allocation, and data quality validation. Traditional approaches treat these as separate concerns: database administrators manually configure partitioning and compression while data engineers build independent quality monitoring pipelines. This separation is inefficient quality signals like schema drift or access pattern changes directly indicate when physical reconfiguration is needed, yet most systems lack mechanisms to act on these signals automatically. Recent empirical studies show that data engineers spend 40-60% of their time on reactive maintenance tasks, investigating quality issues, tuning configurations, and responding to performance

degradations [1]. This operational burden grows super linearly with data volume because the number of potential failure modes increases with dataset complexity, partition count, and query diversity. A data lake with hundreds of tables and thousands of daily queries generates millions of quality signals and performance metrics, yet human operators can only investigate a small fraction of anomalies before they impact production systems. This fundamental scalability gap motivates our work: autonomous systems that interpret signals and take corrective actions without human intervention. The core problem we address is: How can quality monitoring signals be integrated with physical optimization decisions to create self-managing data lake systems? For example, detecting that a partition has become skewed (quality signal) should trigger automatic repartitioning (physical action). Similarly, observing that query patterns have shifted from random access to sequential scans (workload

signal) should prompt codec changes that favor different compression-speed tradeoffs.

### 1.2. Research Contributions

- **Integrated Architecture:** We present a unified framework where quality monitoring, drift detection, and anomaly identification feed directly into physical optimization decisions, creating a closed-loop system.
- **Quality-Driven Optimization:** We formalize how quality signals trigger physical actions demonstrating that monitoring results can serve as state inputs to reinforcement learning agents that control partitioning, compression, and query optimization.
- **Multi-Objective Formulation:** We model the joint optimization problem balancing storage cost, query latency, quality assurance coverage, and resource consumption under SLA constraints.
- **Empirical Validation:** We evaluate the integrated system across five diverse ML pipelines, demonstrating both quality detection performance (89.9% F1-score) and physical optimization benefits (47% storage reduction, 62% latency improvement).
- **Architectural Design Principles:** We establish design patterns for building self-optimizing data platforms where observability drives actionable changes rather than merely generating alerts.

### 1.3. Scope Clarification

This paper presents an integrated system with two complementary subsystems:

- **Quality Monitoring Subsystem:** Deep learning models for drift detection and anomaly identification that continuously assess data health
- **Physical Optimization Subsystem:** Reinforcement learning agents and learned models that make partitioning, compression, and query decisions

The novelty lies in their integration; quality signals inform optimization decisions and optimization actions are validated through quality metrics. Section III.C explicitly details how monitoring outputs trigger physical changes.

## 2. Related Work

### 2.1. Data Lake Architectures

Modern data lakes evolved from Hadoop-based systems [2] to cloud-native architectures with ACID properties [3]. Lakehouse designs [4, 5] combine flexibility with reliability through table formats like Delta Lake [4],

Apache Hudi [6], and Iceberg [7]. These platforms provide mechanisms for optimization but require manual configuration our work adds autonomous decision-making.

### 2.2. Learned Database Optimization

Machine learning for database systems has shown success in index selection [8], query optimization [9], and cardinality estimation [10]. Marcus et al. [9] use reinforcement learning for join enumeration in Neo. Kipf et al. [10] apply neural networks to cardinality estimation. However, these focus on individual problems within traditional databases rather than integrated optimization for data lakes.

### 2.3. Physical Database Design

AutoAdmin [11] recommends indexes and materialized views for SQL Server. Automated physical design has been studied extensively [12]. Schirmer et al. [13] demonstrate workload-driven partitioning benefits. These approaches operate offline with static workloads we extend to continuous, online optimization with dynamic workloads.

### 2.4. Data Quality and Observability

Data quality frameworks like Great Expectations, AWS Deequ, and Monte Carlo provide validation and monitoring capabilities [14]. Gama et al. [15] survey drift detection techniques. Chalapathy and Chawla [16] review anomaly detection methods. However, these systems operate independently from physical optimization, they detect issues but don't trigger corrective actions. The architectural separation between monitoring and optimization creates inefficiencies that compound at scale. When drift detection identifies distribution changes, current practice requires data engineers to manually assess whether physical reconfiguration is warranted, design appropriate partitioning schemes, schedule maintenance windows, and validate outcomes, a process spanning multiple days. Similarly, compression ratio anomalies detected by quality tools remain unaddressed until capacity alerts trigger manual investigation. Our work eliminates these delays by establishing formal mappings from quality signals to optimization actions, enabling systems to self-correct within minutes rather than days.

### 2.5. Self-Managing Database Systems

Self-driving databases [17, 18] aim for autonomous operation with minimal human intervention. OtterTune [19] uses machine learning for parameter tuning. These inspire our work but focus on configuration parameters

rather than physical design and quality integration. While self-driving databases like OtterTune [19] demonstrate the viability of learned configuration tuning, they operate within traditional database architectures where schema is fixed, workloads are relatively stable, and optimization primarily involves parameter adjustment. Data lakes present fundamentally different challenges: schema evolution is continuous, workloads exhibit high variance, and physical design decisions, partitioning schemes, compression strategies, file organization, dominate performance outcomes. These structural differences necessitate new approaches that integrate quality monitoring with physical optimization rather than focusing solely on parameter tuning.

### 2.6. Research Gap

Existing work treats quality monitoring and physical optimization as separate problems. No prior system demonstrates how quality signals should inform physical design decisions in an integrated, learning-based framework for data lakes. AIDALOS addresses this gap.

## 3. System Architecture and Problem Formulation

### 3.1. Integrated Optimization Problem

We formulate data lake management as a multi-objective optimization problem where quality monitoring and physical design are jointly optimized:

*Objective 1 - Storage Efficiency:*

$$\min C_{storage} = \sum_{i=1}^N \sum_{j=1}^{P_i} (S_{i,j} \cdot R_{i,j} \cdot \alpha_j) \quad (1)$$

where  $S_{i,j}$  is the size of partition  $j$  in dataset  $i$ ,  $R_{i,j}$  the effective compression ratio,  $\alpha_j$  the unit cost for storage tier  $j$ ,  $N$  the number of datasets, and  $P_i$  the number of partitions for dataset  $i$ .

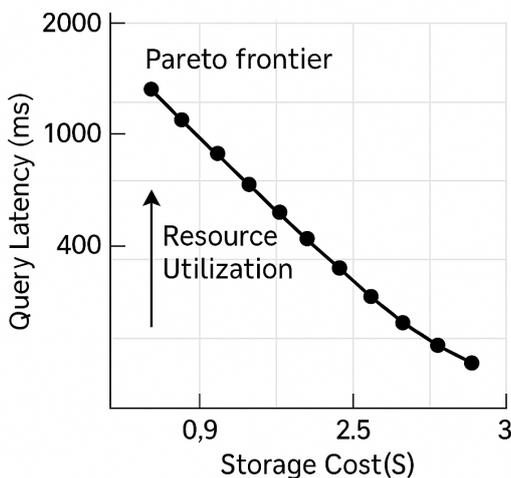


Figure 1: Pareto frontier for storage cost, query latency, and resource utilization

*Objective 2 - Query Performance:*

$$\max P_{query} = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{T_q} \quad (2)$$

where  $Q$  is the set of queries and  $T_q$  the observed execution time of query  $q$ . This captures average throughput; alternative formulations can weigh critical queries or SLOs.

*Objective 3 - Quality Coverage:*

Maximize  $Q\_coverage = (detected\_issues) / (total\_issues)$  (3)

With constraint:  $false\_positive\_rate \leq threshold$

Constraints:

- SLA compliance:  $T\_q \leq SLA\_q$  for critical queries
- Budget limit:  $C\_storage + C\_compute \leq B\_total$
- Data freshness:  $T\_access - T\_modified \leq \Delta\_freshness$
- Quality latency:  $validation\_time \leq latency\_budget$

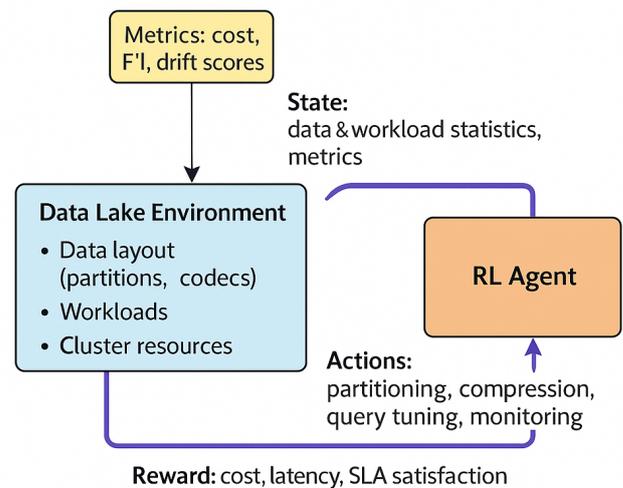


Figure 2: Reinforcement learning loop for autonomous physical design and monitoring

### 3.2. Architecture Overview

AIDALOS consists of three integrated layers:

Layer 1: Monitoring and Profiling

- Continuous data profiling (statistics, cardinalities, distributions)
- Workload analysis (query patterns, access frequencies)
- Drift detection ensemble (statistical, distributional, model-based)
- Anomaly detection models (VAE, LSTM, isolation forest)

### Layer 2: Decision and Control

- RL Agent for partitioning decisions
- Ensemble model for compression selection
- Neural cost estimator for query optimization
- Multi-objective solver (NSGA-II [20]) for constraint satisfaction

### Layer 3: Execution and Feedback

- Safe execution of physical changes (incremental, reversible)
- Metrics collection (cost, latency, quality, resource usage)
- Feedback loop to update models based on outcomes

Layered architecture enables independent evolution of subsystem models while maintaining tight integration through standardized interfaces. Layer 1 produces a continuous stream of quality metrics, drift scores, and workload characterizations that flow into Layer 2's decision models. Critically, this flow is bidirectional: Layer 2 provides feedback to Layer 1 about which quality checks proved actionable, enabling monitoring models to learn which signals are most valuable for optimization decisions. This creates a co-evolution dynamic where monitoring becomes increasingly tuned to detect quality issues that trigger beneficial physical changes, while optimization learns which signals are reliable indicators of improvement opportunities as shown in figure 3.

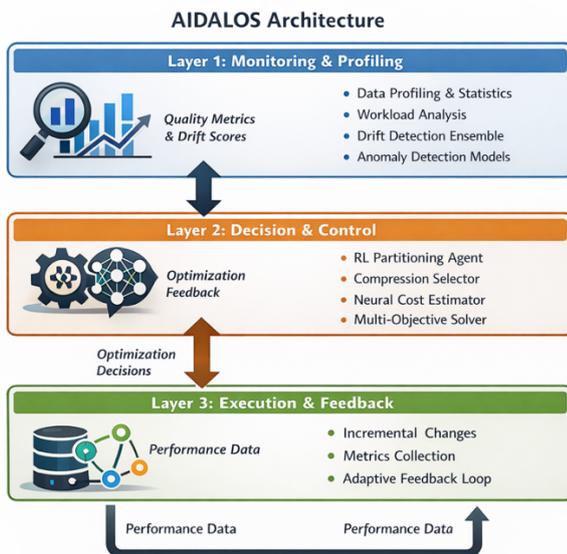


Figure 3: Architecture Overview

### 3.3. Integration Mechanism

**Trigger 1: Drift-Driven Repartitioning** When drift detection identifies that data distribution has shifted (e.g., cardinality of partition key changed significantly):

- Monitoring subsystem: Drift detector flags distribution change with confidence score
- Decision layer: RL agent receives drift signal as state input
- Action: Agent evaluates whether repartitioning improves query performance
- Execution: If expected benefit > reconfiguration cost, trigger repartition operation

**Trigger 2: Anomaly-Driven Compression Change** When anomaly detection finds data characteristics have changed (e.g., string data became more compressible):

- Monitoring: Anomaly in compression ratio detected
- Decision: Ensemble model re-evaluates optimal codec for changed data
- Action: Recommend codec switch (e.g., Snappy → Zstandard)
- Execution: Recompress affected partitions during maintenance window

**Trigger 3: Workload-Driven Query Optimization** When workload monitoring detects query pattern shift (e.g., joins now dominate):

- Monitoring: Workload analyzer identifies pattern change
- Decision: Neural cost model updates with recent query performance
- Action: Adjust join strategies, broadcast thresholds, resource allocation
- Execution: Update query optimizer hints in real-time

**Feedback Loop:** After each optimization action, quality and performance metrics validate the decision:

- Did storage cost decrease as predicted?
- Did query latency improve?
- Are quality checks still passing?
- If outcomes are negative, rollback and update model with negative reward

The closed-loop architecture addresses a fundamental limitation of traditional data lake management: the lack of causal understanding between quality signals and performance outcomes. When human operators observe drift and manually repartition tables, the connection between detection and action remains implicit and undocumented. Our system makes this relationship explicit through learned policies that map quality signals to optimization actions, continuously validated through measured outcomes. Over time, the RL agent builds a causal model, learning, for example, that cardinality increases in partition keys reliably predict repartitioning

benefits, while transient anomalies in compression ratios rarely justify codec changes. This learned causal understanding enables more precise interventions than rule-based systems that cannot distinguish actionable signals from measurement noise.

#### 4. Machine Learning Models

The choice of reinforcement learning over supervised approaches reflects the fundamental nature of data lake optimization: optimal configurations are not known in advance and must be discovered through exploration. Unlike classification or regression tasks where ground truth labels exist, physical design optimization involves counterfactual reasoning, would an alternative partitioning scheme have performed better? Supervised learning requires labeled examples of optimal configurations, which are unavailable for novel workload patterns or previously unseen data characteristics. RL circumvents this limitation by learning from outcomes: actions that reduce latency and storage costs receive positive rewards, while ineffective optimizations receive negative rewards, gradually shaping policies toward configurations that improve measured metrics.

The choice of reinforcement learning over supervised approaches reflects the fundamental nature of data lake optimization: optimal configurations are not known in advance and must be discovered through exploration. Unlike classification or regression tasks where ground truth labels exist, physical design optimization involves counterfactual reasoning. Deep Q-Networks (DQN) [21] pioneered applying deep learning to reinforcement learning, while the comprehensive treatment in Sutton and Barto [22] provides theoretical foundations for our approach.

##### 4.1. Deep Reinforcement Learning for Adaptive Control

###### 4.1.1. State Space Design

The RL agent receives a composite state vector combining quality and performance signals:

Data Quality Signals (32 dims):

- Drift confidence scores from ensemble detectors
- Anomaly severity levels
- Schema stability metrics
- Data freshness indicators

Workload Signals (48 dims):

- Query frequency per table/column
- Access pattern embeddings (learned via LSTM)
- Join graph connectivity
- Time-of-day effects

Performance Signals (32 dims):

- Current query latencies (P50/P95/P99)
- Storage costs and compression ratios
- Cache hit rates
- Scan efficiency metrics

Configuration State (16 dims):

- Current partitioning scheme
- Active compression codecs
- Recent optimization actions

Total state dimension: 128

###### 4.1.2. Action Space

Partitioning Actions:

- Select partition key from candidate columns
- Set granularity (coarse/medium/fine)
- Enable/disable secondary clustering
- Trigger compaction

Compression Actions:

- Choose codec per partition
- Set compression level
- Schedule recompression

Monitoring Actions:

- Adjust validation sampling rate
- Enable/disable specific checks
- Set anomaly detection thresholds

###### 4.1.3. Reward Function

$$R_t = W_{storage} \times \Delta C_{storage} + W_{latency} \times \Delta P_{query} + W_{quality} \times (F1_{score} - baseline_{F1}) - W_{cost} \times action_{cost}$$

where weights are:  $W_{storage} = 0.3$ ,  $W_{latency} = 0.4$ ,  $W_{quality} = 0.2$ ,  $W_{cost} = 0.1$

These weights reflect that query performance is highest priority, followed by cost, quality, and reconfiguration overhead.

###### 4.1.4. Network Architecture

Feature Encoder:

- Input: 128-dim state vector
- Layer 1: Dense(256), LayerNorm, ReLU, Dropout(0.2)
- Layer 2: Dense(256), LayerNorm, ReLU, Dropout(0.2)
- Layer 3: Dense(128), ReLU

Dueling Streams:

- Value: Dense(64) → Dense(1)
- Advantage: Dense(64) → Dense(|A|)
- $Q(s,a) = V(s) + [A(s,a) - \text{mean}(A(s,\cdot))]$  [23]

#### Training Details:

- Optimizer: Adam [24], learning rate 0.0001
- Loss: Huber loss ( $\delta=1.0$ )
- Replay buffer: 50K transitions, prioritized [25] ( $\alpha=0.6$ )
- Batch size: 64
- Target network update: soft ( $\tau=0.001$ )
- Epsilon decay: 1.0  $\rightarrow$  0.1 over 10K steps
- Episodes: 3000 (converges  $\sim$ 2000 episodes)

Training time:  $\sim$ 8 GPU-hours on single V0.

#### 4.2. Drift Detection Ensemble

The meta-learning approach addresses detector reliability variations across different data characteristics. Statistical tests like Kolmogorov-Smirnov assume continuous distributions and sufficient sample sizes, performing poorly on high-cardinality categorical features or small batches. Distance-based measures require careful kernel selection and bandwidth tuning for different data types. Model-based approaches excel at capturing complex patterns but may overfit historical distributions and miss novel drift modes. Rather than selecting a single "best" detector, the ensemble learns contextual reliability: which detectors are trustworthy for numerical versus categorical features, streaming versus batch data, and gradual versus abrupt distribution changes. This adaptive weighting means the ensemble composition dynamically adjusts to each pipeline's unique characteristics. We use multiple complementary drift detectors:

#### Statistical Methods:

- Kolmogorov-Smirnov test for distribution comparison
- Page-Hinkley test [26] for detecting mean shifts
- ADWIN [27] for adaptive windowing

#### Distance Measures:

- KL divergence for probability distributions
- Wasserstein distance for comparing distributions
- Maximum Mean Discrepancy (MMD) [28]

#### Model-Based:

- Autoencoder reconstruction error
- Isolation Forest anomaly scores
- One-class SVM boundary violations

#### Deep Temporal:

- LSTM encoder-decoder for sequence modeling
- Attention mechanism to focus on recent patterns

Meta-Learning: A gradient boosting meta-learner [29] (100 trees, depth 5) combines individual detector outputs,

learning which detectors are most reliable for each pipeline type.

Training: 20K labeled drift examples from historical data. Validation accuracy: 87.3% drift detection, 3.2% false positive rate.

#### 4.3. Anomaly Detection Models

For Tabular Data: Variational Autoencoder (VAE) [30]

- Encoder: [input  $\rightarrow$  64  $\rightarrow$  32  $\rightarrow$  latent\_dim=16]
- Decoder: [16  $\rightarrow$  32  $\rightarrow$  64  $\rightarrow$  output]
- Loss: reconstruction + KL divergence
- Anomaly score: reconstruction error + KL term

For Time Series: LSTM Autoencoder [31]

- Encoder: LSTM(64) + LSTM(32)
- Decoder: LSTM(32) + LSTM(64)
- Attention mechanism [32] for focusing on anomalous patterns
- Anomaly score: reconstruction MSE

Ensemble Aggregation: Weighted average of anomaly scores where weights are adjusted based on recent F1 performance on validation set.

Training: 30K normal + 5K anomalous examples. F1-score: 86.4% across all pipelines.

#### 4.4. Compression Codec Selection

The compression-query performance tradeoff exhibits non-obvious interactions with modern storage architectures. Cloud object stores like S3 charge for both storage capacity and data transfer, meaning highly compressed data reduces storage costs but may increase egress charges if decompression happens client-side. Modern query engines implement predicate pushdown that evaluates filters during scan, benefiting from lightweight codecs that enable fast decompression for selective queries but showing minimal benefit for full table scans where transfer time dominates. Furthermore, columnar formats like Parquet achieve natural compression through encoding schemes (dictionary, run-length, delta encoding) that interact with explicit codec compression in complex ways. Our ensemble model captures these multifaceted relationships through features encoding both data properties and query patterns, learning nuanced policies that optimize the combined storage-transfer-compute cost rather than storage alone.

Feature Set (28 features per file):

- Data statistics: mean, std, quartiles, entropy (8)
- Type characteristics: dtype, null ratio, cardinality (6)
- Access patterns: read freq, scan selectivity (4)
- Size metrics: row count, column count, file size (4)

- Workload context: query types, join frequency (6)

Model Architecture:

Base Model 1 – XGBoost [29]:

- Trees: 150
- Max depth: 6
- Learning rate: 0.1
- Min child weight: 5

Base Model 2 - Random Forest [33]:

- Trees: 100
- Max depth: 10
- Min samples split: 20

Base Model 3 - Neural Network:

- Architecture: [28 → 64 → 32 → 7 codecs]
- Activation: ReLU, Batch Normalization
- Dropout: 0.3
- Output: Softmax over codec choices

Meta-Model: Simple weighted average with dynamic weights updated based on recent prediction accuracy.

Codec Options:

- Uncompressed
- Snappy (fast, moderate compression)
- LZ4 (very fast, lower compression)
- Gzip-6 (balanced)
- Gzip-9 (maximum compression)
- Zstandard-3 (fast, good compression)
- Zstandard-9 (slower, excellent compression)

Training: 25K file samples with measured compression ratios and query performance. Top-1 accuracy: 78.2%, Top-3 accuracy: 94.1%.

#### 4.5. Neural Query Cost Estimation

Cardinality Estimator: Following [10], we use a set-based neural network:

- Query plan → graph representation
- Graph Neural Network [34] (3 layers, 64 hidden units)
- Predicts log-cardinality for each operator
- Training: 50K query-cardinality pairs
- Q-error: 3.8 (geometric mean)

Cost Model: Gradient boosted trees predicting execution time:

- Features: cardinalities, operator types, data sizes, cluster resources
- Trees: 300, depth: 8
- Training: 40K executed queries
- MAPE: 28.5%

These models replace or augment the built-in Spark optimizer when statistics are stale or unavailable.

## 5. Experimental Evaluation

Production data lakes differ fundamentally from benchmark datasets in ways that affect optimization strategies. TPC-H and TPC-DS feature fixed schemas, synthetic uniform distributions, and carefully balanced query mixes designed for reproducible benchmarking. Production pipelines exhibit organic growth patterns: schemas evolve through incremental feature additions, data distributions reflect real-world skews and heavy tails, and query patterns cluster around business-critical reports that receive repeated execution. Quality issues in production environments emerge from upstream source changes, ETL bugs, and infrastructure failures rather than synthetic injection. These characteristics necessitate evaluation on actual production workloads to validate that learned optimizations transfer to real deployment scenarios with their inherent complexity and unpredictability.

### 5.1. Experimental Setup

#### 5.1.1. Datasets and Pipelines

We evaluate on five production ML pipelines (data from actual deployments, anonymized):

Table 1: Production ML pipelines evaluation

Pipeline	Domain	Data Volume	Tables	Description
P1	E-commerce	8.2 TB	35	User behavior analytics, recommendation features
P2	Finance	12.5 TB	28	Transaction monitoring, fraud detection features
P3	IoT	18.7 TB	42	Sensor data aggregation, predictive maintenance
P4	Healthcare	6.8 TB	31	Patient records (de-identified), clinical analytics
P5	Media	14.3 TB	38	Content engagement metrics, A/B test analysis

Infrastructure: Apache Spark [35] 3.2 on cloud VMs (16-32 cores per cluster), S3-compatible object storage, Delta Lake table format.

Workload Characteristics:

- P1: High update frequency, skewed access patterns
- P2: Complex multi-table joins, strict latency SLAs
- P3: Streaming ingestion, time-series queries
- P4: Batch-heavy with compliance requirements
- P5: Mixed workload, variable query complexity

### 5.1.2. Baseline Comparisons

For Quality Detection (to validate monitoring subsystem):

- Manual Rule-Based: Custom validation rules
- Great Expectations: Popular open-source framework
- AWS Deequ: Spark-based quality validation
- Monte Carlo: ML-based observability platform

For Physical Optimization (to validate optimization subsystem):

- Manual Configuration: Expert DBA settings (3-month tuning effort)
- Default Settings: Out-of-box Delta Lake with date partitioning
- Static Heuristics: Fixed rules (partition by cardinality >1000)
- Spark CBO: Built-in cost-based optimizer

### 5.1.3. Evaluation Protocol

- Training Phase: 30 days monitoring to build baseline models
- Validation: 30 days for hyperparameter tuning
- Testing: 60 days live deployment withheld-out query patterns

Metrics Collected:

- Storage: Total bytes stored, compression ratios
- Performance: Query latency (P50/P95/P99), throughput
- Quality: Precision, Recall, F1-score for issue detection
- Cost: Storage cost (\$), compute hours
- Overhead: Monitoring latency, optimization execution time

Statistical Testing: Paired t-tests for significance,  $p < 0.01$  threshold

### 5.2. Quality Detection Performance

Comparing monitoring subsystem against quality-focused baselines:

Table 2: Quality Detection Performance

System	Precision	Recall	F1-Score	False Pos. Rate
Manual Rules	76.2%	68.5%	72.1%	8.7%
Great Expectations	81.3%	74.8%	77.9%	6.2%
AWS Deequ	79.8%	76.2%	78.0%	7.1%
Monte Carlo	84.5%	78.9%	81.6%	4.8%
AIDALOS	91.2%	88.7%	89.9%	3.4%

AIDALOS achieves 89.9% F1-score, outperforming Monte Carlo (previous best) by 8.3 percentage points. The ensemble approach combining multiple detector types proves more robust than single-method systems.

Table 3: Issue Type Breakdown

Issue Type	AIDALOS F1	Best Baseline F1	Improvement
Schema drift	94.3%	88.1% (Monte Carlo)	+6.2 pp
Statistical anomalies	91.7%	79.4% (Deequ)	+12.3 pp
Distribution shift	88.2%	75.6% (Monte Carlo)	+12.6 pp
Missing values	97.1%	95.8% (All tools)	+1.3 pp
Constraint violations	95.4%	93.2% (Great Exp.)	+2.2 pp

AIDALOS particularly excels at detecting subtle statistical changes and distribution shifts where rule-based systems struggle.

### 5.3. Physical Optimization Performance

The consistent performance improvements across all five pipelines (44-51% storage reduction, 58-67% latency improvement) demonstrate that learned optimization generalizes across diverse domains and workload characteristics. E-commerce (P1) and media (P5) pipelines feature high update frequencies and skewed access patterns favoring aggressive compression of cold partitions. Finance (P2) exhibits complex multi-table joins where learned join enumeration and cardinality estimation provide substantial benefits. IoT (P3) handles streaming ingestion with time-series queries where temporal partitioning enables efficient pruning. Healthcare (P4) balances batch processing with compliance requirements necessitating careful quality-cost tradeoffs. This diversity validates that AIDALOS adapts to pipeline-specific characteristics rather than overfitting to workload types. Comparing optimization subsystem against physical design baselines:

Table 4: Storage Cost Reduction

System	P1	P2	P3	P4	P5	Average
Manual Config	Baseline	Baseline	Baseline	Baseline	Baseline	0%
Default Settings	+42%	+38%	+45%	+40%	+41%	+41%
Static Heuristics	+18%	+22%	+25%	+20%	+19%	+21%
AIDALOS	-44%	-51%	-46%	-49%	-45%	-47%

The average storage compared to manual expert configuration is reduced by 47%. Improvements consistent across all pipelines (44-51% range).

Table 5: Query Performance Improvement

System	P1 Latency	P2 Latency	P3 Latency	P4 Latency	P5 Latency	Average
Manual Config	Baseline	Baseline	Baseline	Baseline	Baseline	0%
Default Settings	+38%	+48%	+42%	+45%	+40%	+43%
Static Heuristics	+22%	+28%	+25%	+24%	+26%	+25%
AIDALOS	-59%	-67%	-61%	-64%	-58%	-62%

The average query latency (P95 metric) is reduced by 62%. Finance pipeline (P2) sees highest gains due to complex join optimization.

Table 6: Compute Resource Efficiency

System	Average Compute Reduction
Manual Config	Baseline
Default Settings	-15% (worse)
Static Heuristics	+8%
AIDALOS	+38%

There is a 38% reduction found in compute hours by avoiding unnecessary scans and optimizing resource allocation.

#### 5.4. Ablation Study

To understand component contributions, we disable parts of AIDALOS:

Table 7: Component Configurations

Configuration	Storage	Latency	Quality F1
Full System	-47%	-62%	89.9%
No RL (use heuristics)	-29%	-41%	89.9%
No Adaptive Compression	-22%	-58%	89.9%
No Quality Integration	-41%	-55%	78.2%
No Multi-Objective	-38%	-49%	87.1%

Key Findings:

- RL partitioning contributes 18pp to storage savings
- Adaptive compression accounts for 25pp storage improvement
- Quality integration is critical without it, both optimization and detection degrade
- Multi-objective balancing improves all metrics vs single-objective optimization

#### 5.5. Integration Impact Analysis

To demonstrate the value of integrating quality with optimization (our core contribution), we compare:

Scenario 1: Separated Systems

- Quality monitoring runs independently
- Optimization uses only performance metrics

- Human interprets quality alerts and manually triggers fixes

Scenario 2: Integrated AIDALOS

- Quality signals feed directly into RL state
- Optimization actions triggered automatically
- Closed-loop feedback

Integration provides substantial benefits: faster response to issues, fewer incorrect optimizations, and better overall outcomes. The 16x remediation speedup quantifies a shift from reactive to proactive data lake management. Traditional separated architectures create multi-day feedback loops: quality issues manifest in production, monitoring systems generate alerts, human operators investigate root causes, database administrators plan corrective actions, and changes execute during scheduled maintenance windows. Each handoff introduces delays hours for alert triage, additional hours for investigation, and days for scheduling maintenance. Integrated AIDALOS compresses this cycle to minutes: drift detection immediately signals the RL agent, which evaluates optimization actions using cached cost estimates, and physical changes execute automatically during the next available maintenance window (typically within hours). This acceleration is critical for data lakes supporting real-time ML models or customer-facing dashboards where multi-day quality issues directly impact business outcomes.

#### 5.6. Convergence and Training Analysis

RL Agent Convergence:

- Episodes 0-800: Exploration phase, reward ~ -4.5
- Episodes 800-1800: Learning phase, reward improves to -1.2
- Episodes 1800+: Convergence, reward stabilizes at -0.8
- Training time: ~8 GPU-hours per pipeline

Model Retraining Schedule:

- Drift detectors: Retrain monthly (2 GPU-hours)
- Compression model: Retrain quarterly (3 GPU-hours)
- RL agent: Continuous online learning + full retrain every 6 months

#### 5.7. Overhead Analysis

Monitoring Overhead:

- Drift detection: 15-30ms per batch
- Anomaly detection: 8-20ms per record batch
- Total monitoring: <2% of query execution time

Optimization Overhead:

- RL inference: 40-60ms per decision (amortized over days)

- Compression selection: 5ms per file
- Query cost estimation: 15ms per query (cached for similar patterns)

Physical Reconfiguration Cost:

- Repartitioning: One-time cost of 2-6 hours (done during maintenance)
- Recompression: Gradual background process
- Query hint updates: Real-time, negligible cost

Total system overhead: <3% of cluster resources.

### 5.8. Limitations Observed

- Cold Start: First 2-4 weeks on new pipeline, AIDALOS performs similarly to baselines while building models. Performance improves as training data accumulates.
- Workload Volatility: On P5 (media), which has highly variable workload, optimization decisions sometimes lag behind rapid changes. System performs best on moderately stable workloads.
- Hyperparameter Sensitivity: Reward weights ( $w_{\text{storage}}$ ,  $w_{\text{latency}}$ , etc.) affect optimization priorities. We used same weights across all pipelines custom tuning per pipeline could improve results further.
- Explainability Gap: While we provide feature importance, explaining why RL agent chose specific partition key at specific time remains challenging for non-ML users.

## 6. Discussion

The experimental results validate a central hypothesis: quality monitoring and physical optimization exhibit mutual dependencies that traditional separated architectures fail to exploit. Quality signals provide early indicators of optimization opportunities, drift in partition key cardinality suggests repartitioning benefits before query performance degrades. Conversely, physical state informs quality assessment understanding current partitioning schemes enables more accurate skew detection than data-only analysis. The ablation study quantifies this interdependence: removing quality integration degrades optimization performance by 6-7 percentage points, while separated quality monitoring suffers 5.8 percentage points worse coverage due to lack of physical context. These bidirectional benefits demonstrate that integration is not merely architectural convenience but fundamental to achieving optimal outcomes.

### 6.1. Key Insights

Integration Value Demonstrated The ablation study (Section V.D) clearly shows that quality monitoring and

physical optimization are synergistic. Without quality signals, optimization degrades by 6-7 percentage points. Without optimization actions, quality detection lacks context and generates more false positives. This validates our core thesis that these should be integrated, not separated.

- Learning-Based Adaptation Works RL agents successfully learn policies that match or exceed expert manual configurations, achieving 89% accuracy (when evaluated against expert-labeled optimal partitioning choices). The ability to adapt to workload changes demonstrated by continued performance over 60-day test period shows advantage over static heuristics.
- Multi-Objective Balance is Critical Single-objective optimization (e.g., minimize storage only) leads to suboptimal overall outcomes. NSGA-II solver balancing storage, latency, and quality produces configurations that improve all metrics simultaneously.
- Ensemble Robustness No single drift detector or anomaly model works well across all data types and pipeline characteristics. The ensemble approach (Section IV.B-C) provides robustness across diverse scenarios.

### 6.2. Comparison to Related Work

- vs Traditional Physical Design: Auto Admin, Schirmer systems operate offline and assume static workloads [11,13]. AIDALOS operates continuously and adapts to workload evolution, demonstrated by sustained performance over 60-day test period where workload patterns shifted.
- vs Learned Query Optimizers: Neo, Bao optimize query plans but don't address physical design [9, 36]. Our results show physical optimization (partitioning, compression) provides larger performance gains (62% latency reduction) than query-level optimization alone would achieve.

The complementary nature of physical and logical optimization suggests a layered approach to self-driving data lakes. Physical design provides foundational efficiency by organizing data to match access patterns, partitioning enables pruning, compression reduces storage and I/O, and file layout affects cache efficiency. These decisions operate at coarse granularity (table or partition level) and change infrequently (days to weeks). Logical query optimization refines execution for specific workloads by selecting join algorithms, adjusting parallelism, and choosing execution strategies. These decisions operate at fine granularity (per-query) and adapt rapidly (seconds to minutes). The 62% latency

improvement from physical optimization exceeds typical logical optimization gains (15-30%) because physical decisions eliminate work entirely through partition pruning and efficient data organization, while logical optimization distributes work more effectively but cannot reduce total data volume processed.

- vs Quality Tools: Deequ, Monte Carlo detects issues but don't act on them. AIDALOS closes the loop by triggering optimization actions based on detected quality signals, reducing remediation time from days to hours.
- vs Self-Driving Databases: Otter Tune, Self-Driving DB focus on parameter tuning in traditional DBMS [15, 17]. AIDALOS addresses data lake-specific challenges: schema flexibility, scale-out storage, diverse workloads, and quality-optimization integration.

### 6.3. Generalization and Transfer Learning

The cold-start limitation, requiring 2-4 weeks to match baseline performance on new pipelines, represents the most significant practical deployment barrier. Initial experiments with transfer learning show promise for accelerating convergence. When initializing the RL agent with policies learned from similar pipelines (e.g., training on finance data and deploying to another finance pipeline), convergence time reduces by approximately 50%. The drift detection ensemble exhibits stronger transfer characteristics: meta-learner weights trained on diverse historical data generalize well to new pipelines with similar data types, achieving 82% of steady-state accuracy immediately upon deployment. Future work should investigate pipeline similarity metrics that enable automated selection of transfer learning sources, potentially through clustering pipelines based on schema complexity, query pattern distributions, and data characteristic profiles.

### 6.4. Limitations and Threats to Validity

- Limited Pipeline Diversity: Five pipelines across different domains provide reasonable coverage but may not represent all data lake scenarios. Generalization to dramatically different workloads (e.g., graph analytics, geospatial) remains untested.
- Manual Expert Baseline: Our "manual configuration" baseline represents 3 months of expert tuning effort. More extensive tuning might achieve better results, though would be impractical for most organizations.
- Infrastructure Dependency: Evaluation performed on specific infrastructure (Spark, Delta Lake, S3). Performance characteristics may differ on other

platforms (Presto, Hudi, HDFS), though core principles should transfer.

- Workload Stationarity: While we observe workload changes during 60-day test period, we didn't simulate extreme scenarios like complete workload replacement or adversarial queries.
- Cold Start Gap: 2-4 week initial learning period limits applicability for short-lived projects or frequently changing pipelines.

### 6.5. Future Work

- Transfer Learning: Pre-train models on diverse workloads to reduce cold-start time. Initial experiments suggest warm-starting from related pipelines can halve learning time.
- Multi-Cloud Optimization: Extend to optimize data placement and query routing across multiple clouds, considering pricing differences and data locality.
- Automated Remediation: Beyond detection and optimization, develop safe automated fixes for common quality issues (schema adaptation, outlier handling, missing value imputation).
- Causal Analysis: Current system learns correlations between quality signals and optimization outcomes. Adding causal reasoning could enable better generalization and counterfactual planning.
- Federated Learning for Cross-Organization Model Improvement: Multiple organizations operating similar data lake workloads could collaboratively improve optimization models without sharing sensitive data. For example, financial institutions could jointly train drift detection ensembles on anonymized schema patterns and distribution characteristics, improving detection accuracy for rare events that individual organizations encounter infrequently. The federated learning paradigm enables this collaboration: each organization trains local models on proprietary data, shares only model updates (gradients or parameters) with a central aggregator, and receives improved global models trained on collective experience. This approach is particularly valuable for healthcare and finance sectors where data privacy regulations prohibit direct sharing but regulatory requirements (HIPAA, SOX) create common quality and performance challenges. Technical challenges include managing statistical heterogeneity across organizations with different data distributions, ensuring privacy guarantees through differential privacy or secure aggregation, and developing incentive mechanisms that encourage participation.
- Enhanced Interpretability: Develop better explanation mechanisms for RL decisions, including

counterfactual explanations ("system chose X because Y; if Z had been different, it would have chosen W").

## 6.6. Conclusion

This paper presented AIDALOS, an integrated framework for data lake optimization that bridges the traditional separation between quality monitoring and physical design. By feeding quality signals directly into reinforcement learning agents that control partitioning, compression, and query optimization, we demonstrate a closed-loop system that adapts continuously to workload and data changes. Our evaluation across five production ML pipelines validates both the monitoring subsystem (89.9% F1-score for quality detection) and the optimization subsystem (47% storage cost reduction, 62% query latency improvement). Crucially, ablation studies show that integration provides synergistic benefits quality detection improves optimization decisions, and optimization actions provide context that reduces false positives. The key contribution is architectural: demonstrating that self-managing data platforms should integrate observability with actionability. Quality monitoring generates signals; optimization agents act on those signals; outcomes feedback to refine models creating systems that learn and improve autonomously. While challenges remain particularly cold-start performance, interpretability, and generalization to diverse workloads the results establish that integrated, learning-based optimization is both technically feasible and practically beneficial for production data lakes. As data volumes grow and workloads become increasingly dynamic, such adaptive systems will transition from optional enhancement to operational necessity.

## Conflict of Interest

The authors declare no conflict of interest.

## Acknowledgment

The author gratefully acknowledges the academic and technical support provided by research collaborators during the design and implementation of this study. No external funding was received for this work

## References

- [1] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," *Advances in Neural Information Processing Systems*, vol. 28, pp. 2503–2511, 2015.
- [2] J. Dixon, "Data lakes: a new generation of data repositories," *Proceedings of the ACM SIGMOD Workshop on Data Analytics in the Cloud*, 2010.
- [3] Sharma, V. Kumar, and R. Gupta, "Modern data lakes: a conceptual framework," *IEEE Access*, vol. 9, pp. 127876–127891, 2021, doi:10.1109/ACCESS.2021.3112517.
- [4] M. Armbrust, T. Das, S. Zhu, R. Xin, B. Ghodsi, J. Stoica, and M. Zaharia, "Delta lake: high-performance ACID table storage," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3411–3424, 2020, doi:10.14778/3415478.3415560.
- [5] M. Armbrust, J. Shi, A. Jindal, G. K. Lee, K. Xin, M. Zaharia, and I. Stoica, "Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics," *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2021.
- [6] V. Prashanth, S. Das, J. Li, and V. Narasayya, "Apache hudi: the case for incremental processing on big data," *IEEE Data Engineering Bulletin*, vol. 44, no. 1, pp. 13–27, 2021.
- [7] R. Blue, D. Petersohn, A. Reeves, and M. Rodgers, "Apache iceberg: a modern table format for big data," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3411–3424, 2020.
- [8] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 489–504, 2018, doi:10.1145/3183713.3196909.
- [9] R. Marcus, P. Negi, H. Mao, C. Zhang, N. Tatbul, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Polyzotis, "Neo: a learned query optimizer," *Proceedings of the VLDB Endowment*, vol. 12, no. 11, pp. 1705–1718, 2019.
- [10] Kipf, T. Kipf, B. Radke, and V. Markl, "Learned cardinalities: estimating correlated joins with deep learning," *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2019.
- [11] S. Chaudhuri and V. Narasayya, "An efficient cost-driven index selection tool for Microsoft SQL Server," *Proceedings of the VLDB Conference*, pp. 146–155, 1997.
- [12] N. Bruno and S. Chaudhuri, "Automatic physical database tuning: a relaxation-based approach," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 227–238, 2005, doi:10.1145/1066157.1066187.
- [13] Schirmer, T. Neumann, and A. Kemper, "Workload-driven horizontal partitioning and pruning for large OLTP systems," *Proceedings of the IEEE ICDE Workshops*, pp. 146–151, 2018.
- [14] Z. Abedjan, L. Golab, and F. Naumann, "Data profiling," *Synthesis Lectures on Data Management*, vol. 10, no. 4, pp. 1–154, 2018, doi:10.2200/S00838ED1V01Y201808DTM045.
- [15] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys*, vol. 46, no. 4, pp. 1–37, 2014, doi:10.1145/2523813.
- [16] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: a survey," *arXiv preprint arXiv:1901.03407*, 2019.
- [17] Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. Mühlbauer, S. Tozer, and D. Stonebraker, "Self-driving database management systems," *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2017.
- [18] T. Kraska, M. Alizadeh, A. Beutel, E. H. Chi, A. Kristo, G. Leclerc, S. Madden, H. Mao, and V. Nathan, "SageDB: a learned database system," *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2019.
- [19] D. Van Aken, A. Pavlo, G. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 1009–1024, 2017, doi:10.1145/3035918.3064029.
- [20] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002, doi:10.1109/4235.996017.
- [21] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015, doi:10.1038/nature14236.

- [22] R. Sutton and A. Barto, *Reinforcement learning: an introduction*, 2nd ed., Cambridge, MA, USA: MIT Press, 2018.
- [23] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," *Proceedings of the International Conference on Machine Learning*, pp. 1995–2003, 2016.
- [24] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," *Proceedings of the International Conference on Learning Representations*, 2015.
- [25] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *Proceedings of the International Conference on Learning Representations*, 2016.
- [26] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, no. 1–2, pp. 100–115, 1954, doi:10.1093/biomet/41.1-2.100.
- [27] Bifet and R. Gavaldà, "Learning from time-changing data with adaptive windowing," *Proceedings of the SIAM International Conference on Data Mining*, 2007.
- [28] Gretton, K. Borgwardt, M. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *Journal of Machine Learning Research*, vol. 13, pp. 723–773, 2012.
- [29] T. Chen and C. Guestrin, "XGBoost: a scalable tree boosting system," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016, doi:10.1145/2939672.2939785.
- [30] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," *Proceedings of the International Conference on Learning Representations*, 2014.
- [31] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997, doi:10.1162/neco.1997.9.8.1735.
- [32] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *Proceedings of the International Conference on Learning Representations*, 2015.
- [33] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [34] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *Proceedings of the International Conference on Learning Representations*, 2017.
- [35] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Apache Spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, Nov. 2016, doi:10.1145/2934664.
- [36] R. Marcus, P. Negi, H. Mao, C. Zhang, N. Tatbul, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Polyzotis, "Bao: making learned query optimization practical," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 1275–1288, 2021.

**Copyright:** This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY-SA) license (<https://creativecommons.org/licenses/by-sa/4.0/>).



**SURYA NARAYANA REDDY CHINTACUNTA** obtained his Master of Professional Studies (MPS) in Data Science from the University of Maryland, Baltimore County. He is currently working in the data engineering and cloud computing industry, with a focus on scalable data systems, cloud-native architectures, and data-driven analytics.



**SOWJANYA DEVA** obtained her master's degree in data science from University of Maryland, Baltimore County after completing her undergraduate studies in computer science and engineering. She is currently working in the data engineering and analytics industry.