National Institute of Technology Karnataka, Surathkal

Computational Sciences and Engineering

Course: Distributed Data Management (DDM)

Distributed Data Management Algorithms in Federated Data Management

---

Theory Assignment – 2

Submitted by: Stanzin Angmo
Roll No.: 252CS033
Semester: II
Academic Year: 2025–26

April 18, 2026

Contents

Abstract

Federated Data Management (FDM) integrates heterogeneous, autonomous data sources into a unified queryable system without physical data migration. This report surveys the key Distributed Data Management (DDM) algorithms that power federated systems—covering query decomposition, distributed commit protocols, gossip-based schema propagation, federated model aggregation, and cache-aware query optimization. We classify these algorithms by their primary concern (query-centric, consistency-centric, or integration-centric) and evaluate them across latency, consistency, fault tolerance, scalability, and privacy. Finally, we propose **AQUA-Fed** (Adaptive Query-routing and Update Aggregation for Federated systems), a novel hybrid algorithm that combines reinforcement-learning-based adaptive query routing with differential-privacy-aware result aggregation, designed specifically for privacy-sensitive federated environments such as cross-hospital health information exchanges.

Part I: Survey of DDM Algorithms in Federated Data Management

1. Introduction

Federated Data Management (FDM) provides an architectural paradigm to integrate diverse, autonomous data sources—relational databases, NoSQL stores, data lakes, REST APIs, and legacy systems—into a coherent, queryable whole *without* physical data migration . Unlike homogeneous distributed databases, federated systems must cope with heterogeneity at every level: data models, schemas, query languages, security policies, and transaction semantics.

At the heart of every federated system lies a set of **Distributed Data Management (DDM) algorithms** that handle:

- How global queries are decomposed and routed to the right sources.
- How cross-source transactions maintain consistency.
- How schema changes propagate across loosely coupled participants.
- How aggregated results are computed without exposing raw data.

This report identifies the key DDM algorithms relevant to the FDM domain, classifies them, evaluates their trade-offs, and proposes a novel hybrid algorithm.

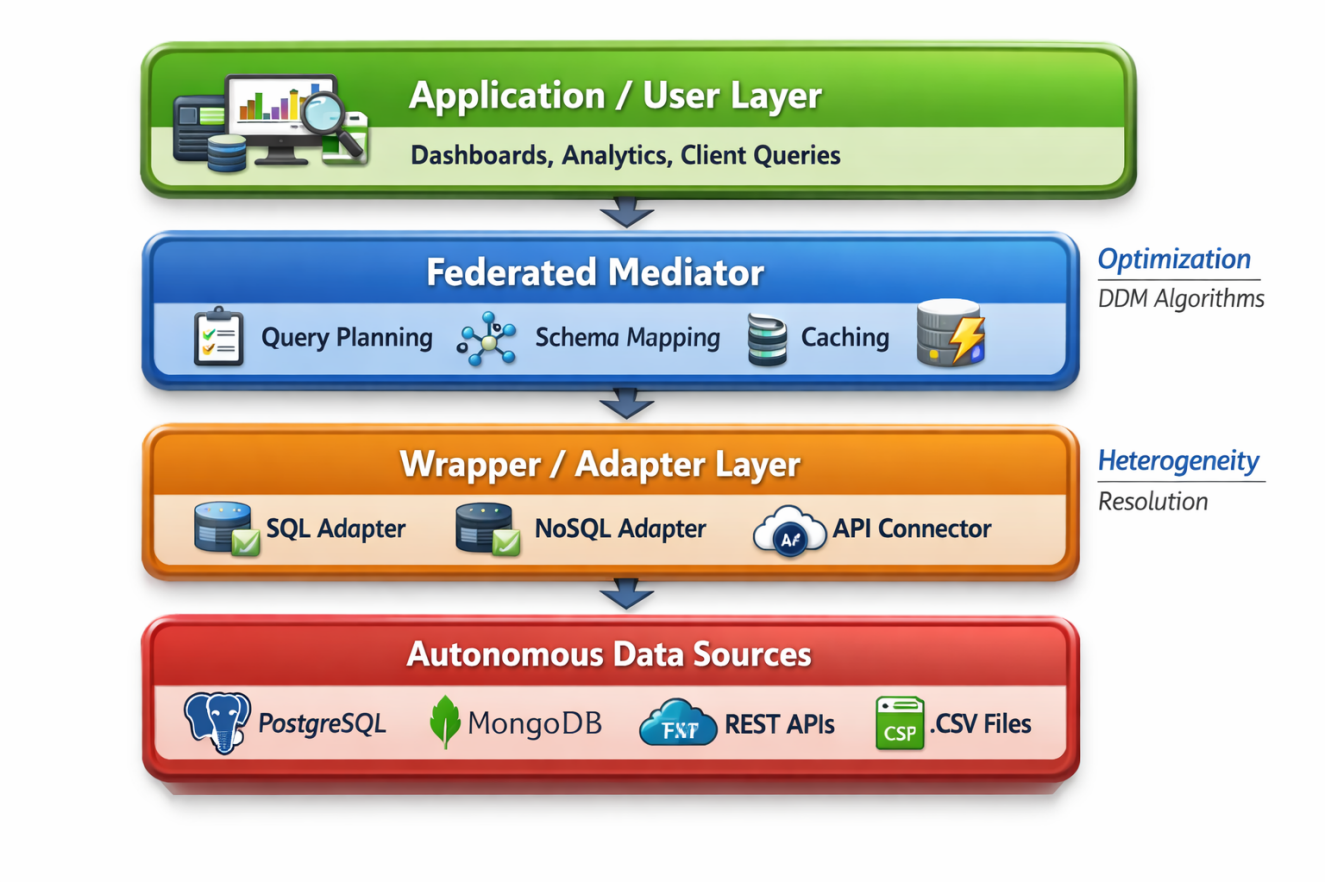


Figure 1: Federated Data Management (FDM) layered architecture showing mediator, adapters, and heterogeneous data sources.

2. Problem Definition & Taxonomy

2.1 The Federated Coordination Problem

Given a set of n autonomous data sources $\{S_1, S_2, \ldots, S_n\}$, each with its own local schema $\mathcal{L}_i$, the federated system must provide:

1. A **virtual global schema** $\mathcal{G}$ that unifies all $\mathcal{L}_i$.
2. A **query engine** that decomposes queries on $\mathcal{G}$ into sub-queries on each $\mathcal{L}_i$.
3. A **consistency protocol** for cross-source write operations.
4. A **result integrator** that merges sub-query results into a single answer.

Each of these components requires specialized DDM algorithms.

2.2 Algorithm Taxonomy

We classify DDM algorithms in federated systems into three families based on their *primary concern*.

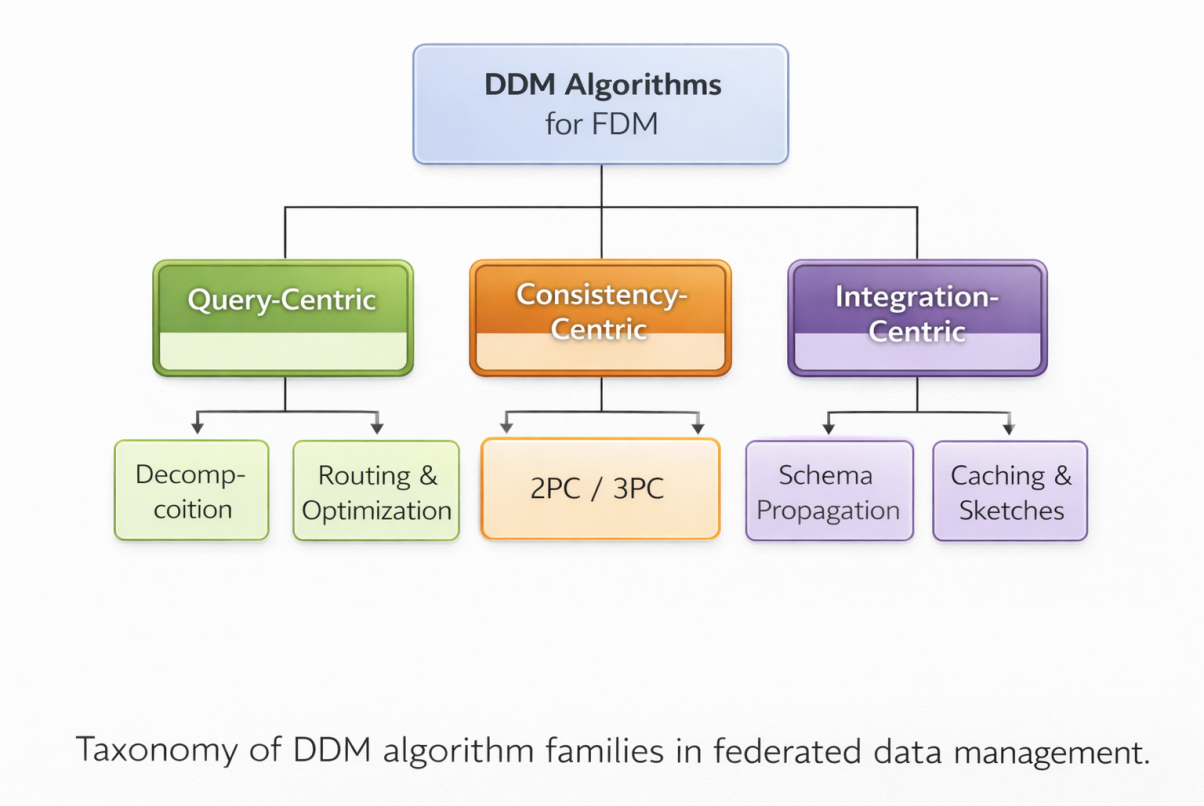


Figure 2: Taxonomy of DDM algorithm families in federated data management.

Table 1: Summary of DDM Algorithm Families for FDM

Family	**Primary Concern**	**Example Algorithms**
Query-Centric	How to split, route, and optimize global queries across sources	Query Decomposition, Cost-Based Routing, Bloom Filter Caching
Consistency-Centric	How to ensure atomicity and correctness of cross-source writes	Two-Phase Commit (2PC), Saga Pattern, Ticket-Based Ordering
Integration-Centric	How to maintain schema coherence and propagate updates	Gossip-Based Schema Propagation, Federated Averaging (FedAvg)

3. Classification of DDM Algorithms for FDM

Table 2 provides a bird's-eye view of all five algorithms surveyed in this report, their category, and the specific FDM challenge they address.

Table 2: Overview of Surveyed DDM Algorithms

#	**Algorithm**	**Category**	**FDM Challenge Addressed**
1	Federated Query Decomposition	Query-Centric	Global-to-local query splitting
2	Two-Phase Commit (2PC)	Consistency-Centric	Cross-source atomic transactions
3	Gossip Schema Propagation	Integration-Centric	Schema updates in loose federations
4	Federated Averaging (FedAvg)	Integration-Centric	Privacy-preserving model training
5	Bloom Filter Query Caching	Query-Centric	Reducing redundant source queries

- **Query-Centric** algorithms (1, 5) focus on the read path—making queries faster and smarter.
- **Consistency-Centric** algorithms (2) focus on the write path—ensuring correctness across sources.
- **Integration-Centric** algorithms (3, 4) focus on keeping the federation coherent—propagating schema updates and aggregating intelligence without exposing raw data.

4. Key DDM Algorithms in Federated Data Management

In this section, we provide detailed explanations of five foundational algorithms. Each subsection covers: the mechanism, a visual diagram, the mathematical basis (where applicable), and the specific FDM application.

4.1 Algorithm 1: Federated Query Decomposition & Routing

The most fundamental DDM operation in a federated system is **query decomposition**: transforming a query written against the global schema into a set of sub-queries targeting individual data sources .

4.1.1 Mechanism

1. **Schema Lookup:** The mediator consults the metadata catalog to determine which sources contain the required tables/attributes.
2. **Predicate Pushdown:** Selection conditions (`WHERE` clauses) are pushed to individual sources to reduce data transfer.
3. **Projection Pushdown:** Only the needed columns are requested from each source.
4. **Sub-query Generation:** The mediator generates native sub-queries for each target source.
5. **Result Merge:** Sub-query results are joined/unioned at the mediator.

4.1.2 Example

A federated SQL query joining `Customer` (PostgreSQL) with `Orders` (MongoDB):

```
SELECT c.name, o.total FROM Customer c JOIN Orders o ON c.id = o.cust_id WHERE o.total
> 1000
```

The mediator generates: (a) SQL sub-query to PostgreSQL for `Customer(id, name)`, and (b) aggregation pipeline to MongoDB for `Orders` where `total > 1000`. The cross-source join is performed at the mediator.

Table 3: Query Translation Examples Across Source Types

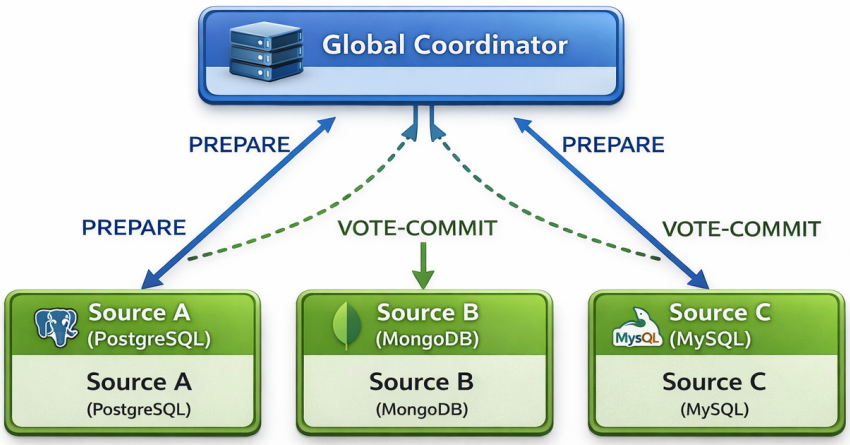
Source	Interface	Translated Sub-query
PostgreSQL	SQL	`SELECT id, name FROM customer`
MongoDB	Aggregation	`db.orders.aggregate([{$match:{total:{$gt:1000}}}])`
REST API	HTTP+JSON	`GET /api/orders?min_total=1000`
CSV/Parquet	File Reader	Scan with predicate on column `total`

4.2 Algorithm 2: Two-Phase Commit Protocol (2PC)

The **Two-Phase Commit (2PC)** protocol ensures *atomicity* across multiple autonomous data sources in a federated system [1]. When a federated transaction modifies data at multiple sites, 2PC ensures that either all sites commit or all abort.

4.2.1 Mechanism

1. **Phase 1 — Prepare (Voting):** The global coordinator sends `PREPARE` to all participating sources. Each source executes the transaction locally, acquires locks, and replies with `VOTE-COMMIT` or `VOTE-ABORT`.
2. **Phase 2 — Decision:** If all sources vote `COMMIT`, the coordinator broadcasts `GLOBAL-COMMIT`. If any source votes `ABORT`, it broadcasts `GLOBAL-ABORT`.



Two-Phase Commit (2PC) protocol in a federated system.

Figure 4: Two-Phase Commit (2PC) protocol in a federated system.

4.2.2 Properties

Table 4: Properties of the Two-Phase Commit Protocol

Property	**Description**
Atomicity	Guaranteed: all-or-nothing across all sources
Message Cost	$O(3n)$ messages for n participants
Blocking Risk	**Yes**—if coordinator fails after PREPARE, participants hold locks indefinitely
Autonomy Impact	High—local sources must cede commit control to the coordinator
Best For	Tightly coupled federations with OLTP workloads

4.2.3 FDM Limitation

In federated environments, 2PC violates source autonomy because local databases must defer their commit decision to the global coordinator. This tension has led to alternative patterns like the **Saga Pattern** [2], which decomposes a global transaction into a chain of local transactions with compensating rollback operations.

Table 5: 2PC vs. Saga Pattern in Federated Systems

Feature	**2PC**	**Saga**
Atomicity	Strong (all-or-nothing)	Eventual (via compensation)
Blocking	Yes (coordinator failure)	No
Autonomy	Low (centralized control)	High (local commits)
Latency	Higher (two rounds)	Lower (sequential local)
Complexity	Protocol-level	Application-level
Best For	Financial transactions	Long-running workflows

4.3 Algorithm 3: Gossip-Based Schema Propagation

In **loosely coupled** federations with no central administrator, schema changes at individual sources must be propagated to all participants. **Gossip protocols** provide an eventually consistent, decentralized mechanism for this [**?** **?**].

4.3.1 Mechanism

1. Each node maintains a local view of the federated schema catalog.
2. At random intervals, a node selects a random peer and exchanges its latest schema version vector.
3. Both nodes merge their catalogs, keeping the latest version of each schema entry.
4. Over $O(\log n)$ rounds, all n participants converge to a consistent view.

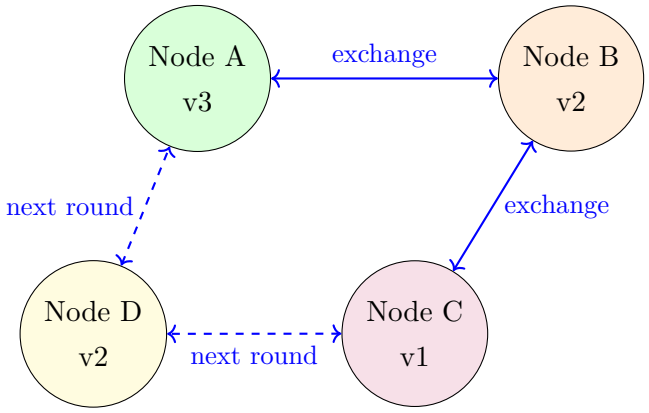


Figure 5: Gossip-based schema propagation in a loosely coupled federation.

4.3.2 Convergence

The gossip protocol guarantees **eventual consistency**: after $O(\log n)$ communication rounds, all nodes hold the same schema catalog with high probability. The convergence rate is exponential—each round doubles the number of informed nodes.

Table 6: Properties of Gossip-Based Schema Propagation

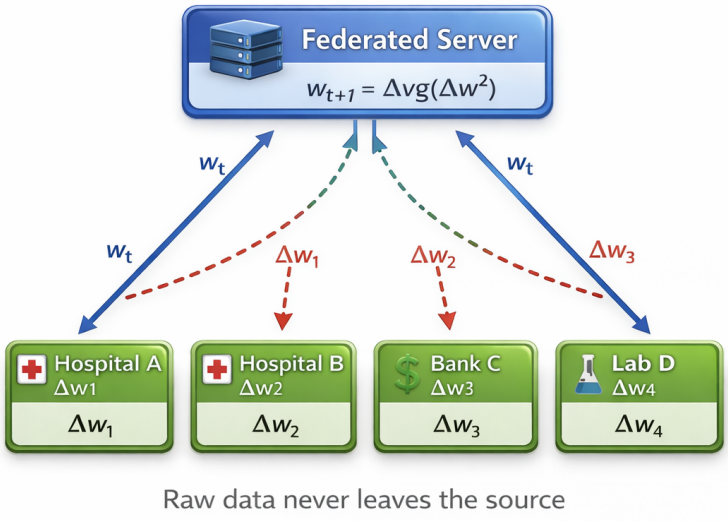
Property	**Value**
Consistency Model	Eventual consistency
Convergence Time	$O(\log n)$ rounds for n nodes
Fault Tolerance	**Very High**—no single point of failure
Message Overhead	$O(n \log n)$ total messages per update
Administration	None—fully decentralized
Best For	Loosely coupled federations, data mesh architectures

4.4 Algorithm 4: Federated Averaging (FedAvg)

Federated Averaging (FedAvg) is the foundational algorithm for *federated learning*—training a shared machine learning model across multiple data sources without centralizing the raw data.

4.4.1 Mechanism

1. The central server (mediator) initializes a global model w_0.
2. In each round t:
   (a) The server sends w_t to a subset of K participating sources.
   (b) Each source k trains w_t on its local data for E local epochs, producing w_t^k.
   (c) Sources send only the model update $\Delta w_t^k = w_t^k - w_t$ to the server.
   (d) The server computes the weighted average: $w_{t+1} = w_t + \sum_{k=1}^{K} \frac{n_k}{n} \Delta w_t^k$
3. Repeat until convergence.



Federated Averaging (FedAvg) process in a **federated system.**

Figure 6: Federated Averaging (FedAvg) process in a federated system where local updates are aggregated without sharing raw data.

Table 7: Properties of Federated Averaging

Property	**Value**
Data Movement	**None**—only model weights are transferred
Privacy	High—raw data stays at source
Communication Cost	Proportional to model size × number of rounds
Non-IID Handling	Moderate (struggles with highly heterogeneous data)
Best For	Cross-institutional ML (hospitals, banks), FDM analytics

4.4.2 FDM Application

FedAvg maps directly onto the mediator–wrapper architecture: the mediator plays the role of the federated server, and each wrapper acts as a local training node. This enables cross-hospital diagnostic model training where patient imaging data never leaves the hospital's secure environment .

4.5 Algorithm 5: Bloom Filter Query Caching

In a federated system, the mediator frequently queries multiple sources for the same entities across different user sessions. **Bloom filter–based caching** uses a probabilistic data structure at the mediator to quickly determine whether a source is likely to contain relevant data, thereby *avoiding unnecessary network round-trips* .

4.5.1 Mechanism

1. Each wrapper computes a Bloom filter B_i over the primary keys of its exported data.
2. The mediator stores all B_i in a compact cache.
3. When a query arrives for key k, the mediator checks $k \in B_i$ for each source.
4. If B_i returns `false`, source i is **definitely** skipped (no network call).
5. If B_i returns `true`, source i is queried (small false-positive rate $\approx 1\%$).

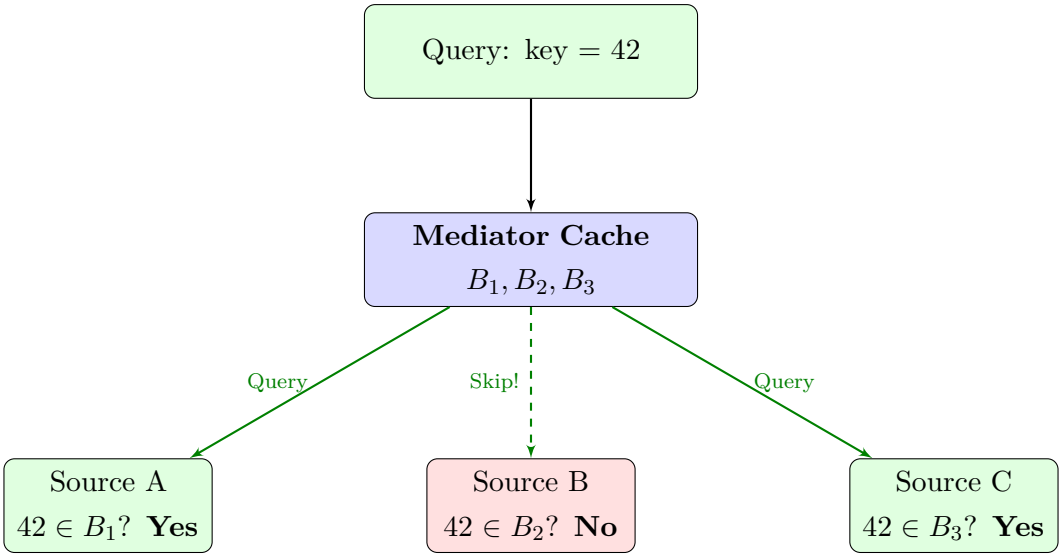


Figure 7: Bloom filter caching eliminates unnecessary source queries.

Table 8: Properties of Bloom Filter Query Caching

Property	**Value**
Space Cost	$\approx$ 10 bits per key (highly compact)
False Positive Rate	$\approx 1\%$ (tunable)
False Negative Rate	**0%**—never misses a relevant source
Network Savings	30–60% reduction in source queries
Update Cost	Wrappers periodically regenerate and push B_i
Best For	Key-lookup dominated workloads in federated systems

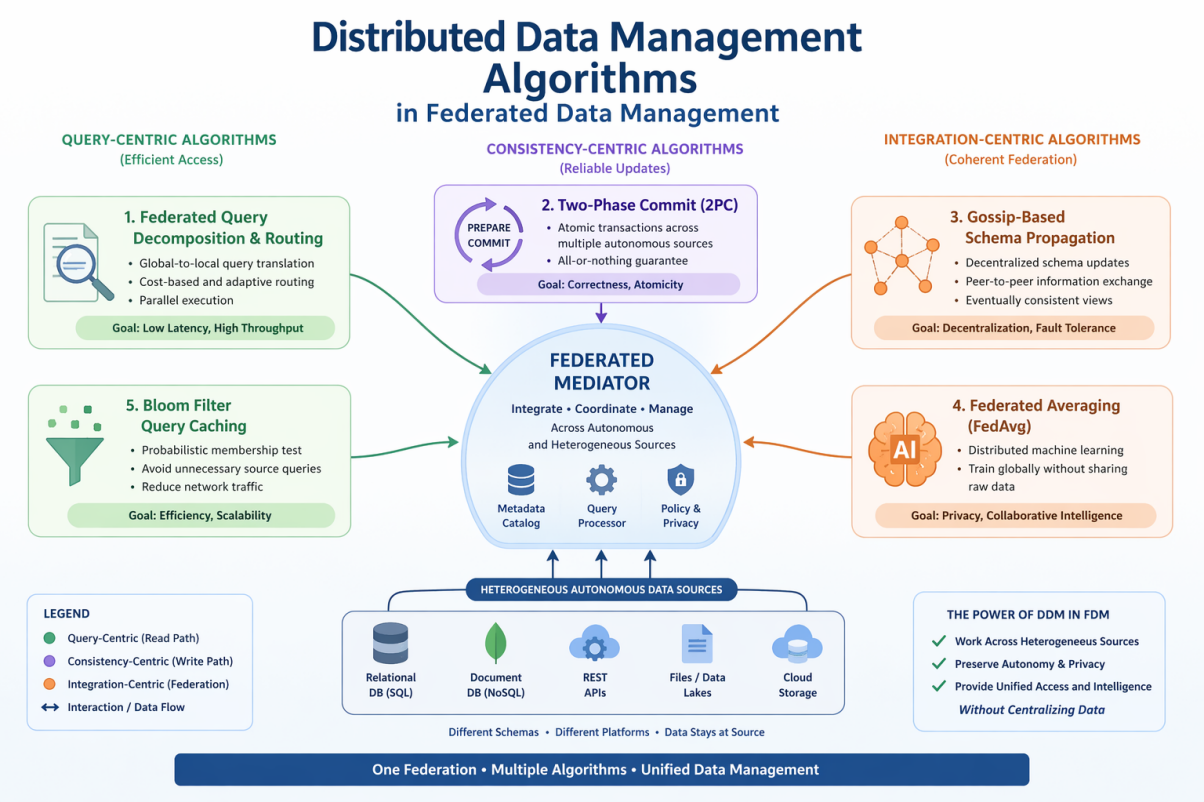


Figure 8: overall summary of Data disttributed management algorithms in federated data management.

5. Comparative Analysis

Table 9 provides a comprehensive comparison of all five surveyed algorithms across six critical dimensions relevant to federated data management.

Table 9: DDM Algorithm Comparison Matrix for Federated Data Management

Algorithm	**Latency**	**Consistency**	**Fault Tol.**	**Scalability**	**Privacy**
Query Decomp.	Medium	Exact	Low	High	Low
2PC	High	Strong	Low	Medium	Low
Gossip Schema	Low	Eventual	**Very High**	**Very High**	Medium
FedAvg	Medium	N/A	Medium	High	**High**
Bloom Caching	**Very Low**	N/A	Medium	High	Low

5.1 Key Trade-offs in Federated Systems

Table 10: Key Trade-offs Between DDM Algorithm Families

Trade-off	**Choose Left When...**	**Choose Right When...**
Consistency vs. **Availability**	Financial/transactional workloads requiring strong ACID guarantees	Analytical/read-heavy workloads tolerating staleness
Centralized vs. **Decentralized**	Tight coupling, stable topology, few sources	Loose coupling, dynamic membership, many sources
Exact vs. **Approximate**	Regulatory reports, billing systems	Real-time dashboards, anomaly detection
Privacy vs. **Performance**	Healthcare, cross-border data, GDPR	Internal analytics within a single org

5.2 When to Use Which Algorithm

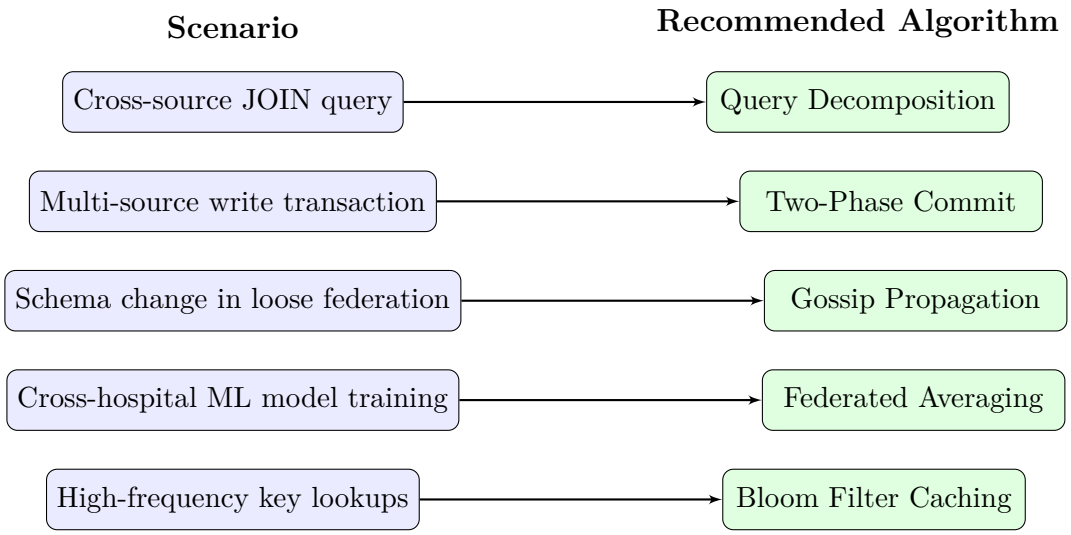


Figure 9: Algorithm selection guide: matching FDM scenarios to DDM algorithms.

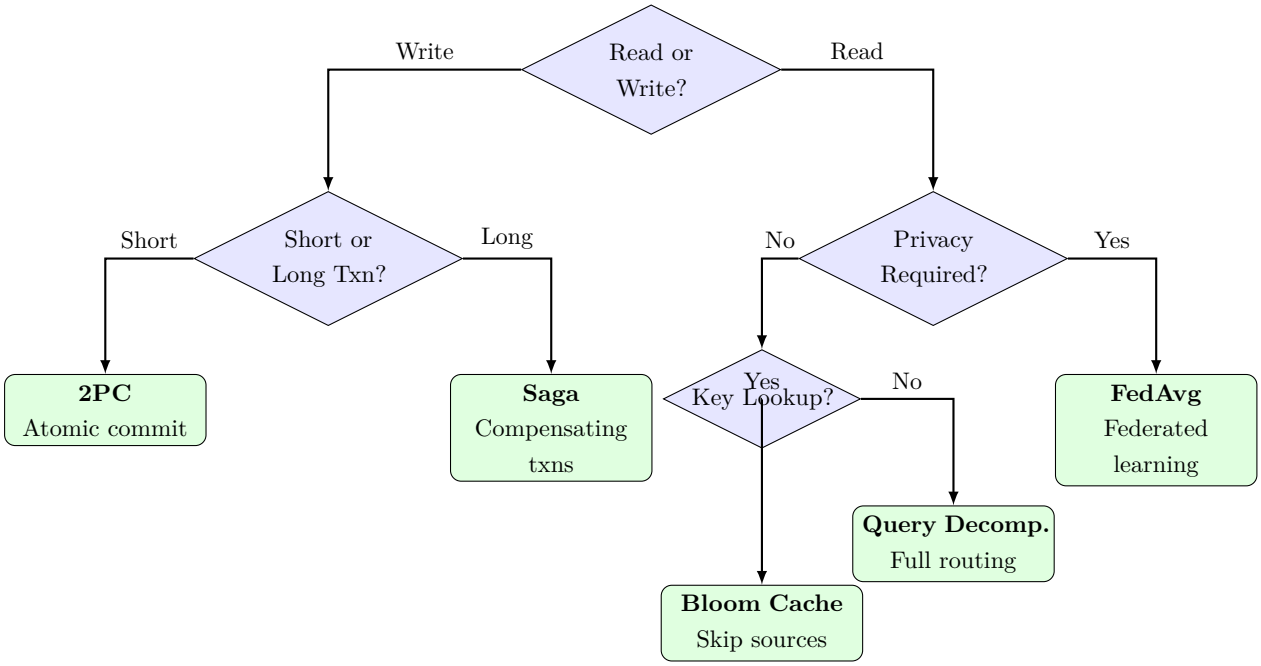


Figure 10: DDM algorithm selection flowchart for federated data management.

Analysis of Figure 10: The flowchart highlights that the choice of DDM algorithm in federated systems is driven by three key decisions:

1. **Workload type:** Is the operation a read (query) or a write (transaction)?
2. **Transaction duration:** For writes, is it a short ACID transaction or a long-running workflow?
3. **Privacy requirement:** For reads, does the data need to remain at the source?

This systematic approach prevents the common mistake of applying a single algorithm (e.g., 2PC everywhere) when the workload characteristics call for a fundamentally different approach.

Part II: Novel Algorithm — AQUA-Fed

6. AQUA-Fed: Adaptive Query-Routing and Update Aggregation

The algorithms surveyed in Part I each address one aspect of federated data management. However, real-world FDM systems face *simultaneous* challenges: sources have varying load (static routing creates hotspots), sources may go offline (rigid decomposition fails), and cross-source results involve sensitive data (privacy protection needed). No single existing algorithm handles all three.

We propose **AQUA-Fed** (Adaptive Query-routing and Update Aggregation for Federated systems), a hybrid algorithm with three integrated components:

1. **Adaptive RL-based query routing** — a Q-learning agent observes source load, latency, and data freshness to dynamically choose the best routing plan [**?**]. The agent updates its policy after every query using: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$, where the reward $r_t = -\text{latency}$.
2. **Gossip-based fallback** — when the health monitor detects a source failure (heartbeat timeout), neighboring wrappers gossip their cached partial results to reconstruct an approximate answer. This "Route-by-Default, Gossip-by-Exception" logic ensures graceful degradation.
3. **Differential privacy aggregation** — for sensitive queries, each wrapper adds calibrated Laplace noise $\tilde{v} = v + \text{Lap}(\Delta f/\varepsilon)$ before sending results to the mediator.

6.1 Architecture

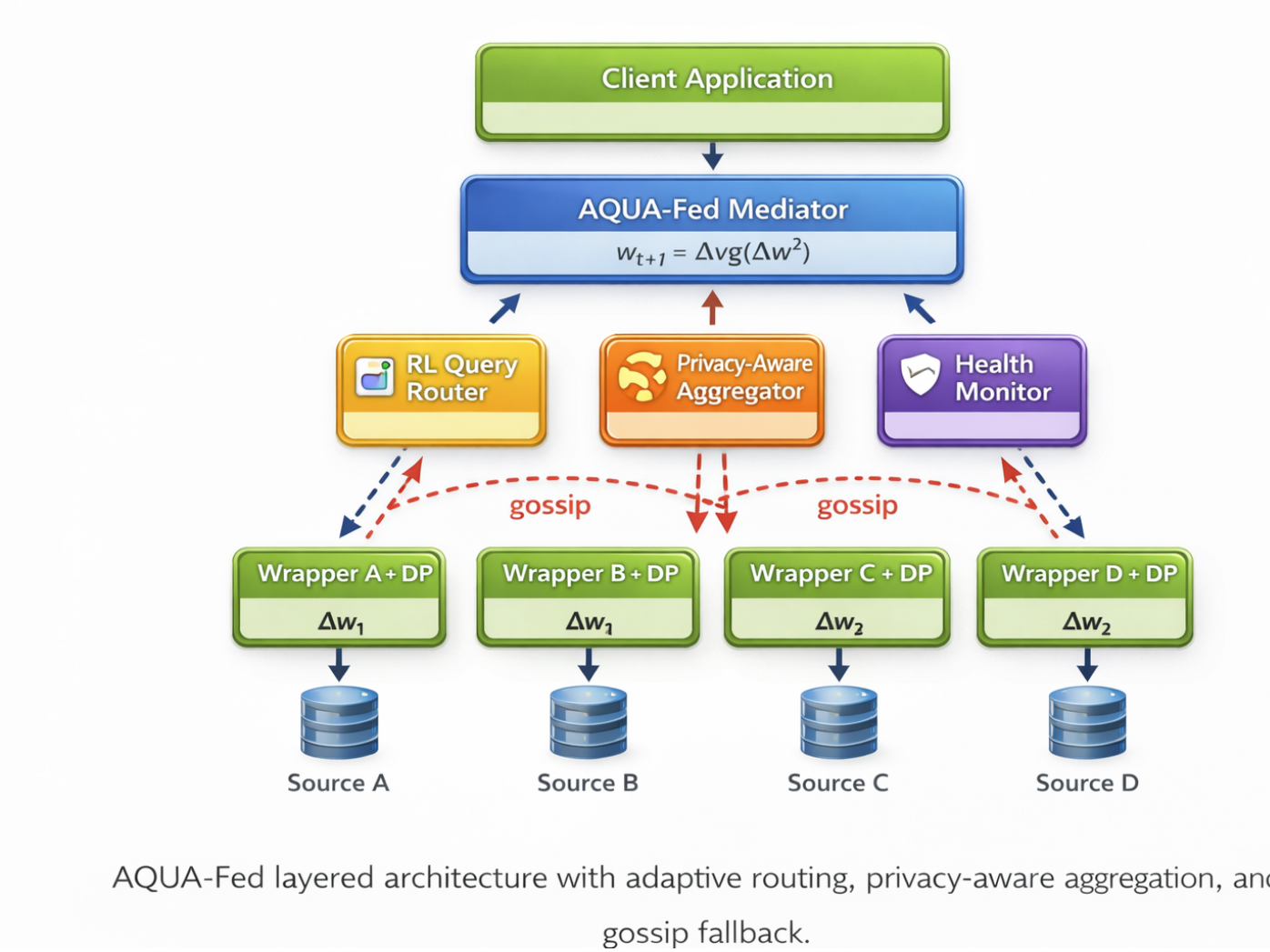


Figure 11: AQUA-Fed layered architecture with adaptive routing, privacy-aware aggregation, and gossip fallback.

6.2 Mechanism

AQUA-Fed operates in five sequential phases for every incoming global query:

1. **Decompose:** The global query Q is decomposed into sub-queries $\{q_1, \ldots, q_m\}$ using the federated schema mappings (identical to standard query decomposition).

2. **Observe & Route:** For each sub-query, the RL agent observes a state vector $s_t = (\text{load}_i, \text{latency}_i, \text{freshness}_i, \text{bloom}_i)$ and selects the optimal routing action a_t via its Q-table. This adaptively balances load across sources.

3. **Health Check & Fallback:** The health monitor checks source availability. If a target source is alive, the sub-query is routed normally. If it has failed (heartbeat timeout), neighboring wrappers activate gossip mode—exchanging cached partial

results to reconstruct an approximate answer.

4. **DP Noise Injection:** If the query is flagged as privacy-sensitive, each wrapper injects calibrated Laplace noise (ε-differential privacy) into its result *before* transmitting to the mediator.

5. **Merge & Learn:** The mediator merges all sub-results into the final answer. The RL agent then updates its Q-table using the observed latency as reward, continuously improving future routing decisions.

6.3 Properties

Table 11: Properties of AQUA-Fed Algorithm

Property	**Value**
Type	Hybrid (structured routing + gossip fallback + DP)
Routing Strategy	Adaptive via Q-learning; learns from real-time load and latency
Fault Tolerance	**High** — gossip fallback reconstructs approximate results on source failure
Privacy	ε-differential privacy at wrapper level; raw data never leaves source
Consistency	Approximate in fallback mode; exact in normal mode
Message Overhead	$O(n)$ steady-state; $O(n \log n)$ during gossip episodes
Cold Start	Requires 50–100 queries for RL convergence; uses static routing as bootstrap
Best For	Privacy-sensitive, multi-source federations with dynamic availability (e.g., cross-hospital HIE)

6.4 Algorithm Pseudocode

Algorithm 1 AQUA-Fed: Adaptive Query-Routing and Update Aggregation

Require: Global query Q, source set $\{S_1, \ldots, S_n\}$, privacy budget ε
Ensure: Privacy-protected merged result R

$\{q_1, \ldots, q_m\} \leftarrow \text{Decompose}(Q, \mathcal{G})$ {Phase 1: Decompose}
for each sub-query q_j **do**
  {Phase 2: Adaptive Route}
  $s_t \leftarrow \text{ObserveState}(\text{load}, \text{latency}, \text{freshness})$
  $a_t \leftarrow \arg\max_a Q(s_t, a)$
  if $\text{HealthMonitor}(S_{a_t}) = \text{ALIVE}$ **then**
    Route q_j to source S_{a_t}
  else
    $\text{result}_j \leftarrow \text{GossipReconstruct}(\text{neighbors}(S_{a_t}))$ {Gossip Fallback}
  end if
end for
for each result result_j **do**
  {Phase 3: DP Aggregation}
  if Q.privacyRequired **then**
    $\text{result}_j \leftarrow \text{result}_j + \text{Lap}(\Delta f / \varepsilon)$
  end if
end for
$R \leftarrow \text{Merge}(\text{result}_1, \ldots, \text{result}_m)$ {Phase 4: Merge}
Update RL: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$
return R

6.5 Comparison with Existing Algorithms

AQUA-Fed vs. Existing DDM Algorithms in Federated Systems

Feature	Static Decomp.	Pure Gossip	FedAvg	AQUA-Fed
Adaptive Routing	No	No	No	**Yes**
Load Balancing	No	Partial	No	**Yes**
Fault Tolerance	Low	High	Medium	**Yes**
Privacy Protection	No	Partial	No	**Yes (DP)**
Exact Results	Yes	Approx.	N/A	Approx.
Message Overhead	O(n)	O(n log n)	O(Kd)	O(n)*
	O(n)	O(Kd)	O(3n)	O(3n)

"O(n) in steady-state; O(n log n) during gossip fallback. K = participants, d = model dimension.

Figure 12: Comparison of AQUA-Fed with existing DDM algorithms in federated systems.

7. End-to-End Pipeline Mapping

The following diagram shows how the surveyed algorithms and AQUA-Fed map onto the layers of a complete federated data management pipeline.

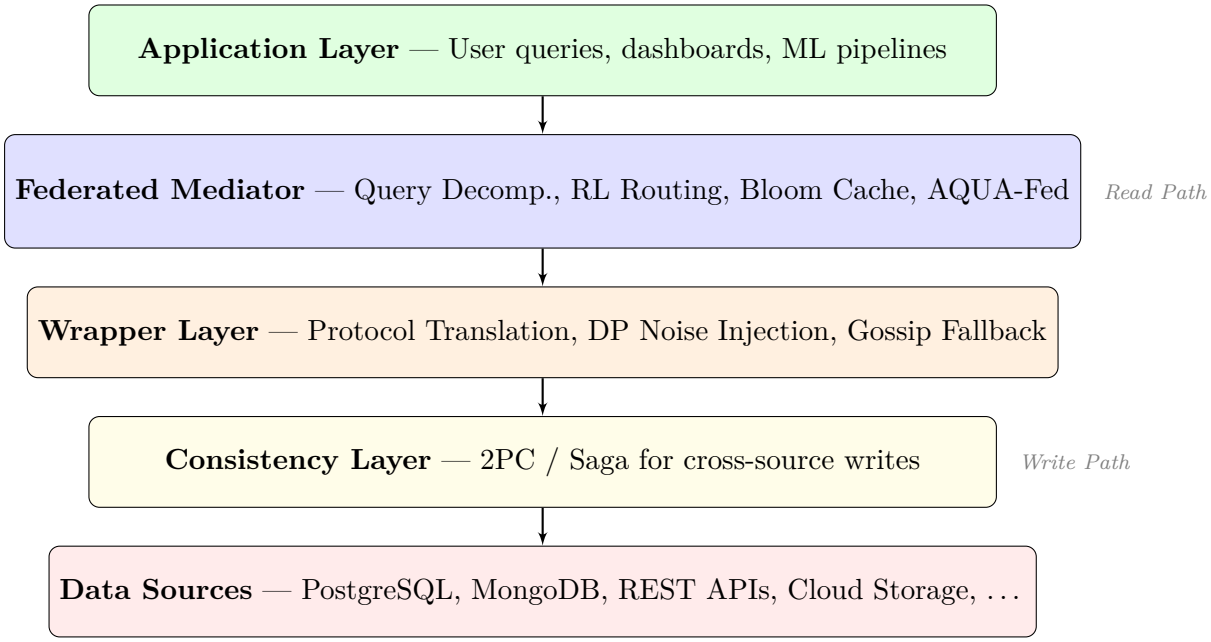


Figure 13: End-to-end FDM pipeline with DDM algorithm placement at each layer.

Table 12: Algorithm Placement Across FDM Pipeline Layers

Layer	**Algorithm(s)**	**Role**
Mediator	Query Decomposition	Split global queries into sub-queries
Mediator	RL Router (AQUA-Fed)	Adaptively route sub-queries based on load
Mediator	Bloom Filter Cache	Skip irrelevant sources
Wrapper	Gossip Propagation	Schema update dissemination
Wrapper	DP Noise (AQUA-Fed)	Privacy protection before aggregation
Wrapper	FedAvg	Distributed model training
Consistency	2PC / Saga	Cross-source write atomicity

8. Conclusion

This report has surveyed the landscape of Distributed Data Management algorithms as they apply to Federated Data Management systems. Key findings:

1. **No single algorithm suffices.** Federated systems require different DDM algorithms at different layers and for different workload types (reads vs. writes, exact vs. approximate, private vs. open).
2. **Query-centric algorithms** (decomposition, Bloom filter caching) optimize the read path by reducing unnecessary source queries and minimizing data transfer.
3. **Consistency-centric algorithms** (2PC, Saga) address the write path, balancing atomicity against autonomy and latency.
4. **Integration-centric algorithms** (gossip propagation, FedAvg) maintain the coherence and intelligence of the federation without centralizing raw data.
5. **AQUA-Fed**, our proposed hybrid algorithm, addresses the gap where existing algorithms fail individually: it provides adaptive routing under varying loads, graceful degradation under source failures, and built-in privacy protection for sensitive cross-source queries.

Table 13: Summary: Surveyed Algorithms and Their Primary Contribution

Algorithm	**Category**	**Key Contribution to FDM**
Query Decomposition	Query-Centric	Enables cross-source querying
Two-Phase Commit	Consistency-Centric	Atomic cross-source writes
Gossip Propagation	Integration-Centric	Decentralized schema updates
Federated Averaging	Integration-Centric	Privacy-preserving distributed ML
Bloom Filter Caching	Query-Centric	Reduces redundant source queries
AQUA-Fed (Novel)	**Hybrid**	**Adaptive routing + DP privacy + fault toleran**

The federated data management paradigm continues to grow in importance as organizations increasingly operate across multi-cloud, multi-institutional, and cross-border environments. Future work on AQUA-Fed includes: (1) extending the RL router to deep Q-networks for larger state spaces, (2) integrating secure multi-party computation alongside differential privacy, and (3) benchmarking against production federated engines like Trino and Denodo.

References

[1] S. Pandya *et al.*, "Federated learning for smart cities: A comprehensive survey," *Sustainable Energy Technologies and Assessments*, vol. 55, p. 102987, 2023.

[2] D. C. Nguyen *et al.*, "Federated learning for smart city applications: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2396–2431, 2021.

[3] O. Wahab *et al.*, "Federated learning: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2273–2298, 2021.

[4] T. Li *et al.*, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[5] Q. Yang *et al.*, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[6] V. Mothukuri *et al.*, "A survey on security and privacy of federated learning," *Future Generation Computer Systems*, vol. 115, pp. 619–640, 2021.

[7] T. Q. Dang *et al.*, "A review of federated learning for smart healthcare," *IEEE Reviews in Biomedical Engineering*, vol. 15, pp. 286–301, 2022.