Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Event-Driven Document Processing: MongoDB, Flink, and AI Schema Evolution

Jyothish Sreedharan

Independent Researcher, USA

ARTICLE INFO

Received: 24 Feb 2026

Accepted: 02 March 2026

ABSTRACT

Document-based data systems have changed how organizations handle data by allowing flexible structures without fixed schemas, however processing large volumes of documents presents challenges where schemas change unpredictably, document structures vary widely, and relationships between data elements shift over time. This paper proposes a conceptual framework that combines MongoDB as an event store, Apache Flink for stream processing on Kubernetes, and AI-powered schema management, where the framework envisions MongoDB storing documents in sharded clusters that handle high read and write volumes. As documents change by adding new fields, changing nesting depth, or shifting meaning, traditional pipelines fail to maintain consistency, therefore the proposed system includes an AI layer designed to learn document structures, identify relationships, and detect changes automatically. The design incorporates large language models to process unstructured data and extract entities, while embedding models would match new fields to known patterns, and Apache Flink on Kubernetes would process document changes in real time, normalizing and enriching data as it flows. The system integrates with lakehouse platforms through filtering and metadata optimization. This paper presents a conceptual architectural framework with proposed design patterns based entirely on publicly available research, open-source technologies, and theoretical analysis, where no proprietary data, production systems, or organizational deployments are referenced. Empirical validation through benchmark testing using publicly available datasets including TPC-H benchmark data, Yelp Open Dataset, GitHub Archive, Intel Berkeley Research Lab sensor data, and synthetic healthcare data from Synthea, along with AI model accuracy evaluation and simulated deployment metrics are identified as critical future work. Specific performance improvements, cost-benefit analyses, and model selection criteria require experimental validation before production deployment using these public domain datasets. The framework provides detailed implementation guidance including model selection criteria, training methodologies, cost-latency trade-offs, and comprehensive experimental design for validation using publicly accessible benchmark datasets to ensure complete independence from any specific organization or employer, maintaining transparency and reproducibility through exclusive reliance on open-source tools and public data sources. This paper presents a conceptual architectural framework with proposed design patterns. Empirical validation through benchmark testing, AI model accuracy evaluation, and production deployment metrics are identified as critical future work. Specific performance improvements, cost-benefit analyses, and model selection criteria require experimental

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

validation before production deployment. The framework provides detailed implementation guidance, including model selection criteria, training methodologies, cost-latency trade-offs, and a comprehensive experimental design for validation.

Keywords: Event-Driven Architecture, Document-Native Data Sourcing, Ai-Enabled Schema Evolution, Apache Flink Stream Processing, Mongodb Sharding

1. Introduction

Document-based databases have become popular because they can handle different data structures without requiring predefined schemas. Unlike traditional relational databases that enforce strict structures, document databases let organizations manage diverse data sources that change unpredictably. MongoDB demonstrates this flexibility through its BSON format, which stands for Binary JSON, where each document in a collection can have a different structure. The WiredTiger storage engine uses document-level locking and compression to reduce storage by up to sixty percent while maintaining fast operations through sophisticated caching mechanisms. These technical capabilities enable organizations to ingest diverse document types rapidly while maintaining operational efficiency at scale. Contemporary data ecosystems demand systems capable of handling documents undergoing structural changes throughout their lifecycles, acquiring new fields, altering nesting depth, and experiencing semantic shifts as business requirements evolve.

However, this flexibility creates challenges at a large scale. Schemas drift without bounds, meaning that document structures can change in unpredictable ways over time. Document shapes become inconsistent, where different documents in the same collection may have vastly different structures. Nesting grows arbitrarily deep, creating complex hierarchies that are difficult to parse and transform. Relationships between data elements become volatile, where connections that existed yesterday may no longer be valid today. Modern event-driven systems need pipelines that process document changes at high throughput while maintaining consistency across distributed systems. Traditional extract-transform-load approaches cannot handle the speed and variability of modern data, where documents change structure throughout their lifetime. Stream processing frameworks address these needs through continuous computation that processes unlimited datasets with low latency, moving away from batch processing that artificially groups continuous data. Without adaptive mechanisms, conventional pipelines degrade over time, causing analytical inconsistencies where downstream systems receive incompatible data, integration failures where components cannot communicate, and computational inefficiencies that worsen as datasets mature.

This paper proposes a conceptual framework that addresses these challenges by combining MongoDB as a scalable event store, Apache Flink on Kubernetes for distributed stream processing, and AI-driven schema management. The architecture aims to shift from reactive to proactive data management, where machine learning models would analyze document structures, identify relationships, and automatically adapt transformation rules. Apache Flink's stream processing supports event-time semantics and stateful operations, processing data based on timestamps in the events rather than when they arrive. This ensures correct results even when events arrive out of order or experience network delays. This approach treats document processing as an intelligent, self-adjusting system rather than a fixed sequence of transformations. The significance extends beyond technical implementation to address fundamental limitations in current data engineering, where manual schema management consumes substantial engineering effort and cannot scale to handle growing numbers of diverse data sources.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Component	Characteristic	Capability
Document Model	BSON-based flexible schema	Dynamic structure without predefined constraints
Storage Engine	WiredTiger with compression	Document-level concurrency control
Schema Flexibility	Multi-structured content support	Accommodation of evolving data patterns
Stream Processing	Continuous computation model	Unbounded dataset processing with minimal latency
Event-Time Semantics	Timestamp-based processing	Correct results despite out-of-order arrival
Batch vs Stream	Departure from batch boundaries	Natural handling of continuous data flows

Table 1: MongoDB Architecture and Stream Processing Foundations [1, 2]

2. Architectural Foundations and System Design

The proposed architecture uses MongoDB's distributed document model as the primary event store, taking advantage of horizontal sharding and replica sets to achieve high throughput for data ingestion. MongoDB's sharding works through three components that work together to distribute data and coordinate operations. Shard servers store data subsets, where each shard holds a portion of the overall dataset based on a shard key that determines how documents are distributed. Config servers maintain cluster information, including which documents live on which shards and how the cluster is configured. Mongos query routers direct operations to the right shards based on shard keys, acting as the entry point for all client operations. The system uses eventual consistency models to provide reliability for event sourcing while handling write-heavy workloads from document changes.

Sharding uses either range-based partitioning or hashed sharding to distribute data across the cluster. Range-based partitioning keeps documents with similar key values on the same shard, which makes range queries efficient because all the relevant data is colocated. For example, if documents are sharded by date, queries for a specific date range only need to access one or a few shards rather than scanning the entire cluster. Hashed sharding distributes documents evenly across shards by computing hash values of the shard key, which prevents hotspots where one shard receives disproportionate traffic but sacrifices some range query optimization. Sharded clusters spread documents across multiple nodes using configurable shard keys, enabling linear scaling as data volumes grow and providing natural boundaries for downstream processing.

The event-driven approach organizes the system around document change events, which include inserts when new documents are created, updates when existing documents are modified, and deletes when documents are removed. These events are captured through MongoDB's change streams, which are a mechanism that monitors the database for changes and emits notifications. This transforms the database from a passive storage system into an active participant in data flow, sending detailed event notifications that include both the changed document and metadata describing the change. The metadata includes information like the type of change, when it occurred, and what fields were affected. Change streams use MongoDB's operation log, which is an internal record of all database operations, to provide ordered and resumable event sequences with delivery guarantees suitable for building reliable distributed systems. The event stream serves as the main connection between storage and processing, separating ingestion speed from transformation capacity through persistent event queues that can buffer events during processing delays.

Apache Flink, deployed in a Kubernetes environment, forms the stream processing layer that would consume document change events and execute real-time transformations. Flink's dataflow model is

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

based on directed graphs of stateful operators, where each operator performs a specific transformation and can maintain state between events. This provides the foundation for implementing complex document normalization, enrichment, and restructuring. Flink achieves low latency through its pipelined execution, where operators process records immediately upon arrival rather than collecting micro-batches and processing them together. This means that as soon as an event arrives, it begins flowing through the transformation pipeline without waiting for other events. This approach contrasts with micro-batch architectures used by some competing frameworks that typically show latencies from hundreds of milliseconds to several seconds because they must accumulate a batch of events before processing begins.

The framework's exactly-once processing semantics would ensure document transformations remain consistent even during failures. This is achieved through distributed snapshots and two-phase commit protocols that use lightweight asynchronous barriers flowing through the data stream. When a failure occurs, Flink can restore the state from the most recent snapshot and replay events from that point forward, ensuring that each event is processed exactly once, neither lost nor duplicated. Kubernetes integration would enable dynamic resource allocation, letting the system scale processing capacity in response to traffic bursts while maintaining workload isolation through namespaces and resource quotas. Namespaces provide logical separation between different workloads, while resource quotas limit how much CPU, memory, and storage each namespace can consume.

The design emphasizes loose coupling between components through well-defined interfaces and event contracts. Document change events follow standardized formats that encode versioned schemas, time metadata, and origin information needed for downstream consumers to interpret changes correctly. Versioned schemas allow the system to evolve event formats over time while maintaining backward compatibility with older consumers. Time metadata includes timestamps indicating when events occurred and when they were processed, enabling accurate event-time processing. Origin information tracks where events came from, which is essential for debugging and tracing data lineage. This separation allows independent evolution of storage, processing, and consumption layers while maintaining end-to-end consistency through schema registries that enforce compatibility rules across component boundaries. The resulting architecture achieves the flexibility needed to handle diverse document types while maintaining the operational characteristics required for production deployment at scale.

Architecture Layer	Implementation Strategy	Performance Characteristic
Shard Servers	Data subset storage	Horizontal scalability
Config Servers	Cluster metadata management	Routing information maintenance
Query Routers	Client operation direction	Shard key-based distribution
Range Partitioning	Adjacent value colocation	Efficient range query support
Hashed Sharding	Uniform distribution via hash	Hotspot prevention
Pipelined Execution	Immediate record processing	Millisecond-level latency achievement
Asynchronous Barriers	Lightweight checkpoint coordination	Exactly-once semantics without overhead

Table 2: Distributed Architecture Components [3, 4]

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

3. AI-Enabled Schema Evolution and Semantic Understanding

Traditional document processing pipelines assume static or slowly changing schemas, relying on manually created transformation rules that become misaligned with actual document structures as source systems evolve. Engineers write code that expects documents to have specific fields in specific formats, but when source systems add new fields, change data types, or restructure hierarchies, these transformation rules break. This brittleness causes failures in dynamic environments where document shapes vary widely, introducing fields with new meanings, restructuring hierarchies, or establishing previously unknown relationships between entities. For example, an e-commerce system might initially store customer addresses as flat fields but later restructure them into nested objects containing street, city, and postal code components. Traditional pipelines would fail when encountering this new structure because they expect the old flat format.

Automated data preparation techniques have emerged as solutions, using machine learning to perform data cleaning, transformation, and feature engineering without manual work. Research shows that data scientists reportedly spend up to eighty percent of their time on data preparation rather than analysis, which represents a significant inefficiency. Automated approaches can detect anomalies like outliers or inconsistent formats, handle missing values through imputation or other strategies, and standardize formats to ensure consistency. Enterprise deployments of automated data preparation systems demonstrate significant improvements in data quality while reducing manual effort by approximately sixty to seventy percent. The proposed framework addresses this through an AI-enabled schema evolution layer that would continuously learn document structures, identify semantic relationships, and automatically adapt transformation logic to maintain consistency across diverse document populations.

3.1 Proposed AI Components and Model Selection

The framework proposes three AI components working together to handle schema evolution. Each component addresses a specific aspect of the schema management problem and would need to be carefully selected and configured based on deployment requirements.

The first component uses a large language model for structural analysis and semantic interpretation of document content. Large language models have demonstrated remarkable capabilities in understanding context, extracting entities, and identifying relationships in unstructured text. When applied to document schemas, these models can analyze field names like "cust_addr_line_1" and understand that it represents the first line of a customer address, even if they have never seen this exact field name before. Candidate models for implementation include GPT-4 or Claude-3.5 for complex semantic understanding of field names and values, fine-tuned BERT models for domain-specific entity extraction with lower latency, and open-source alternatives like Llama-3 or Mistral for cost-sensitive deployments where inference costs need to be minimized.

Model selection would depend on multiple criteria that need to be balanced against each other. Inference latency represents how quickly the model can process a document, with a target of less than one hundred milliseconds per document to avoid becoming a bottleneck in the processing pipeline. Accuracy requirements focus on entity extraction precision, targeting greater than ninety percent to ensure the system correctly identifies most entities without excessive false positives. Cost constraints require balancing accuracy against inference cost, where more capable models like GPT-4 provide higher accuracy but cost significantly more per inference. Deployment options include using cloud APIs where the model provider hosts the model and charges per inference, versus self-hosted deployment, where the organization runs the model on its own infrastructure and pays for compute resources rather than per-inference costs.

The expected capabilities of the large language model component include extracting entities from field names and values, where entities might be customers, products, transactions, or other business

objects. The model would identify attributes and relationships, understanding that certain fields describe properties of entities while others represent connections between entities. The model would categorize unfamiliar structures based on semantic similarity, meaning when it encounters a new field structure, it would compare it to known structures and classify it accordingly. For example, if the system has seen various address formats before, it could recognize a new address format as semantically similar even if the exact field names differ.

The second component uses embedding models for similarity matching, which provides the mathematical foundation for comparing document structures. Embedding models convert text into numerical vectors in high-dimensional space, where semantically similar items have vectors that are close together, while dissimilar items have vectors that are far apart. This allows the system to perform nearest-neighbor searches to find structurally similar elements across different document collections. Candidate embedding models include OpenAI text-embedding-3, which produces vectors with fifteen hundred thirty-six dimensions and provides high accuracy, Sentence-BERT as an open-source alternative producing seven hundred sixty-eight dimension vectors with faster inference, and custom Skip-gram models that can be trained on domain-specific data, producing three hundred to six hundred dimension vectors.

The Skip-gram model architecture, which is part of the Word2Vec family of models, processes training corpora containing billions of words to learn distributed vector representations. These representations capture sophisticated semantic and syntactic relationships through simple vector arithmetic. The canonical example demonstrates that the vector for "King" minus the vector for "Man" plus the vector for "Woman" approximately equals the vector for "Queen", showing that the model has learned gender relationships. When applied to document schemas, similar arithmetic can identify that "customer_name" and "client_name" are semantically equivalent, or that "order_total" and "invoice_amount" represent similar concepts in different systems.

The proposed similarity matching process would work through several steps. First, the system converts field names and sample values to embeddings by passing them through the embedding model. For a field like "customer_email", the system would create a vector representation that captures its semantic meaning. Second, it performs nearest-neighbor search in a vector database, which is a specialized database optimized for finding similar vectors quickly. Third, it identifies fields with cosine similarity greater than zero point seven as semantically related, where cosine similarity is a measure of how similar two vectors are based on the angle between them rather than their magnitude. A similarity of zero point seven represents a threshold where fields are similar enough to likely represent the same concept. Fourth, it applies transformation strategies from similar historical mappings, meaning if the system previously learned how to transform "cust_email" to a standard format, it would apply the same transformation to "customer_email" based on their semantic similarity.

The third component uses reinforcement learning for adaptation decisions, where the system must decide whether observed schema changes represent genuine evolution requiring adaptation or transient anomalies that should be ignored. The schema evolution layer would maintain a versioned catalog of discovered document structures, tracking how schemas change over time and maintaining relationships between structural variants. This historical view enables sophisticated drift detection algorithms that distinguish between meaningful structural evolution and temporary anomalies. Machine learning classifiers would analyze the frequency of changes, asking whether this change occurs consistently or sporadically. They would examine the size of changes, determining whether many documents show the change or just a few. They would assess the consistency of changes, checking whether the change appears across related document types or only in isolated cases.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

3.2 Training Methodology

The training methodology for these AI components requires careful planning and execution. Data collection would gather historical document samples across various schema versions, providing examples of how schemas have evolved in actual production systems. This historical data serves as training examples showing the system what genuine schema evolution looks like. Human-labeled examples of correct entity extraction would provide ground truth for training the entity extraction models, where experts annotate documents identifying entities, attributes, and relationships. Validated field mappings from production systems would show the system which fields from different source systems represent the same concepts, teaching it to recognize semantic equivalence.

The training approach would begin with an initial supervised learning phase using these labeled examples. During this phase, the models learn to replicate the decisions that human experts made in the training data. An active learning loop would then allow human experts to validate edge cases, where the system identifies situations where it has low confidence in its predictions and asks humans for guidance. This approach maximizes the value of expensive human expert time by focusing it on the most challenging cases rather than having experts label routine examples. Continuous model refinement based on production feedback would allow the models to improve over time as they process real production data and receive feedback on their decisions.

Evaluation metrics establish concrete targets for assessing model performance. Schema inference precision targets greater than eighty-five percent, meaning that when the model identifies an entity or relationship, it should be correct at least eighty-five percent of the time. Schema inference recall targets greater than eighty percent, meaning the model should successfully identify at least eighty percent of the entities and relationships that actually exist in the documents. Drift detection accuracy targets greater than ninety percent, meaning the system should correctly classify schema changes as either genuine evolution or transient anomalies at least ninety percent of the time. The false positive rate for drift detection targets less than five percent, meaning the system should avoid incorrectly flagging normal variations as significant schema changes more than five percent of the time. These metrics provide quantitative goals for model development and enable comparison between different model architectures and training approaches.

3.3 Cost and Latency Trade-offs

Understanding the cost and latency trade-offs is essential for practical deployment decisions. Different AI approaches impose different costs in terms of both money and time, and the optimal choice depends on specific deployment requirements and constraints.

LLM inference costs vary significantly between different deployment options. Using GPT-4 API would cost approximately one to three cents per thousand tokens, where a token roughly corresponds to a word or word fragment. For document schema analysis, a typical document might consume five hundred to one thousand tokens, including the document structure, field names, and sample values, resulting in costs of approximately half a cent to three cents per document. This provides highly accurate results but becomes expensive when processing millions of documents. Claude-3.5 API costs approximately zero point three to one point five cents per thousand tokens, providing a balanced option with good accuracy at a lower cost than GPT-4. Self-hosted Llama-3 has infrastructure cost only, where the organization pays for servers to run the model but not for each inference. This can provide the lowest per-document cost at scale because the infrastructure costs are amortized across many inferences, though it requires expertise to deploy and maintain.

The decision framework for choosing between these options depends on document characteristics and processing requirements. High-value complex documents where errors would be costly should use advanced LLMs like GPT-4 to maximize accuracy even at a higher cost. Standard documents with known patterns can use rule-based approaches that cost nearly nothing because they require only

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

simple code execution. Medium complexity documents that fall between these extremes would benefit from fine-tuned, smaller models that balance accuracy and cost. Batch processing scenarios where results are not needed immediately allow acceptable latency for more thorough analysis using slower but more accurate models. Real-time processing, where results must be available within milliseconds, requires cached patterns and lightweight models that can execute quickly even if they provide slightly lower accuracy.

Latency impact varies dramatically between different approaches and directly affects system throughput and responsiveness. Rule-based transformation takes less than one millisecond per document because it involves only simple code execution, checking predefined patterns, and applying transformations. Embedding similarity search takes five to ten milliseconds per document because it requires computing the embedding vector and searching the vector database for nearest neighbors. LLM inference via API takes two hundred to five hundred milliseconds per document due to network latency and model inference time, where the request must travel over the network to the model provider, the model must process the request, and the response must travel back. Self-hosted LLM inference takes fifty to one hundred fifty milliseconds per document, faster than API-based approaches because network latency is eliminated, but still slower than simpler approaches because large language models are computationally intensive.

The recommended hybrid approach addresses these trade-offs by using different strategies for different documents based on their characteristics. Rule-based approaches would handle eighty percent of documents that follow known patterns, providing near-zero latency and cost. AI models would be applied to twenty percent of documents that exhibit novel or ambiguous structures where rule-based approaches would fail. Complex documents would be queued for batch AI processing when real-time constraints exist, allowing the system to maintain responsive performance for routine documents while still handling complex cases accurately. This hybrid approach balances cost, latency, and accuracy by using the most appropriate technique for each situation.

Evolution Component	Machine Learning Technique	Operational Benefit
Data Preparation	Neural networks and ensemble methods	Anomaly detection and format standardization
Transformation Automation	Algorithm-driven cleaning	Manual effort reduction
Entity Extraction	Natural language understanding	Field name and value interpretation
Semantic Mapping	Skip-gram model architecture	High-dimensional vector space representation
Vector Arithmetic	Distributed representations	Syntactic and semantic relationship capture
Field Matching	Nearest-neighbor searches	Automatic transformation inference

Table 3: AI-Driven Schema Evolution Mechanisms [5, 6]

4. High-Throughput Stream Processing and Document Normalization

The stream processing layer, implemented through Apache Flink operators deployed across Kubernetes pods, would transform raw document change events into normalized and enriched representations suitable for downstream analytics. This transformation must operate at extreme scale,

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

handling millions of events per second while maintaining consistent semantics and acceptable latency characteristics. Flink's operator-based programming model enables building complex transformation pipelines from reusable processing units, where each unit encapsulates specific normalization logic while maintaining state needed for context-aware transformations. The state might include information about previously seen document structures, cached reference data, or intermediate results that need to be combined across multiple events.

Document normalization would address structural differences through a multi-stage transformation process that progressively reshapes documents into standard forms. Initial operators parse incoming documents, validating that they contain well-formed data structures and extracting metadata necessary for routing decisions. The metadata might include information about which source system produced the document, what type of entity it represents, or which version of the schema it follows. Subsequent stages apply learned transformation rules to restructure documents into standard forms, which might involve flattening nested hierarchies where deeply nested structures are converted into flatter representations that are easier to query. It might involve expanding arrays into separate records where a single document containing an array of items is split into multiple documents, one for each item. Or it might involve consolidating related fields into complex structures where multiple flat fields are grouped into nested objects based on semantic understanding that they represent components of a larger concept.

Data preparation for analytical systems includes several critical activities that improve data quality and usability. Data cleaning removes inconsistencies and errors such as duplicate records, invalid values, or conflicting information. Transformation converts data into suitable formats, such as standardizing date formats, normalizing text to consistent casing, or converting units of measurement. Integration combines data from multiple sources, resolving conflicts when different sources provide different values for the same entity. Reduction manages dataset size while preserving information content through techniques like aggregation, sampling, or dimensionality reduction. Studies show that proper data preparation can improve model accuracy by twenty to thirty percent because cleaner, more consistent data enables algorithms to learn more accurate patterns. It can also reduce training time by forty to fifty percent because well-prepared data requires fewer iterations to converge to good models.

This normalization process does not impose a single rigid schema but rather adapts transformation strategies to document types, recognizing that different document families require distinct standard representations optimized for their specific analytical uses. For example, transactional documents might be normalized to focus on monetary amounts, timestamps, and parties involved, while product catalog documents might be normalized to emphasize hierarchical categorization, attributes, and relationships. This flexibility ensures that normalization enhances rather than degrades the usefulness of data for its intended purposes.

Enrichment operators augment normalized documents with derived attributes, reference data, and contextual information not present in the original change events. Derived attributes are calculated values based on existing fields, such as computing age from birthdate, categorizing transactions by size, or deriving geographic regions from postal codes. Reference data comes from external sources like lookup tables that provide additional information about entities mentioned in documents, such as expanding product codes to full product descriptions or augmenting customer records with demographic information. Contextual information might include market conditions at the time a transaction occurred, weather data for location-based events, or historical trends relevant to interpreting current values.

These transformations leverage external data sources, including reference tables stored in databases or cached in memory, graph databases encoding entity relationships that can be traversed to find connected information, and machine learning models that compute derived features from document

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

content. The enrichment process occurs incrementally as documents flow through the pipeline rather than requiring all enrichment to complete before passing documents downstream. Operators cache frequently accessed reference data in memory to minimize latency, avoiding repeated database queries for the same information. Flink's asynchronous I/O mechanisms prevent blocking on external lookups, allowing operators to initiate multiple lookups concurrently and continue processing other events while waiting for responses. This incremental approach to enrichment is essential for maintaining throughput because it avoids sequential dependency chains that would otherwise limit parallelism, where each enrichment step must complete before the next can begin.

Kubernetes orchestration would provide the infrastructure foundation for high-throughput processing through dynamic resource scaling and efficient workload distribution. Horizontal pod autoscaling would automatically adjust the number of Flink task manager replicas in response to incoming event rates, ensuring processing capacity matches ingestion speed without manual intervention. The Horizontal Pod Autoscaler runs a control loop at intervals of fifteen seconds, regularly checking metrics and making scaling decisions. It calculates desired replica counts using the formula where desired replicas equals the ceiling of current replicas multiplied by current metric value divided by desired metric value. For example, if there are currently ten replicas running at eighty percent CPU utilization but the target is sixty percent, the autoscaler calculates that the ceiling of ten times eighty divided by sixty equals the ceiling of thirteen point three, which equals fourteen replicas, so it would scale up to fourteen replicas.

The system includes a built-in stabilization window, preventing rapid changes that could cause thrashing, where the system repeatedly scales up and down. Scale-down operations require all recommendations to consistently suggest a reduction for three hundred seconds before execution, meaning the system must observe five consecutive measurements showing that fewer replicas are sufficient before actually reducing capacity. This prevents premature scale-down that would immediately need to be reversed. Pod affinity rules optimize the placement of related processing tasks to minimize inter-pod network latency, ensuring that operators who frequently communicate are placed on the same physical nodes when possible. Anti-affinity constraints ensure fault tolerance by distributing replicas across availability zones, preventing a single zone failure from taking down all replicas of a critical component.

Resource quotas and quality-of-service classes prioritize critical pipeline components during resource contention. Resource quotas limit the total amount of CPU, memory, and storage that a namespace can consume, preventing any single workload from monopolizing cluster resources. Quality-of-service classes determine which pods get evicted first when nodes run low on resources. Guaranteed quality-of-service pods that have explicit resource requests and limits matching are never evicted if they stay within their limits. Burstable pods that have resource requests but can use more if available get evicted only if no best-effort pods remain. Best-effort pods that have no resource requirements get evicted first during resource pressure. Critical pipeline components are configured with guaranteed quality-of-service to maintain stable end-to-end latency even under degraded conditions.

The autoscaler supports scaling based on multiple metrics simultaneously, including CPU utilization, measuring what percentage of allocated CPU is being used, memory consumption tracking resident memory usage, and custom application-specific metrics like event queue depth or processing lag. The system selects the recommendation that produces the highest replica count to ensure adequate capacity, meaning if CPU metrics suggest ten replicas but queue depth suggests twelve replicas, the system scales to twelve. This conservative approach ensures that all constraints are satisfied rather than risking underprovisioning. This tight integration between Flink's distributed runtime and Kubernetes' orchestration capabilities creates a resilient processing environment capable of sustaining high throughput while handling failures gracefully through automatic replica replacement and stateful recovery.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Processing Aspect	Implementation Method	Efficiency Gain
Data Cleaning	Multi-stage transformation	Inconsistency and error removal
Format Conversion	Learned transformation rules	Suitable format generation
Source Integration	Incremental enrichment	Multi-source data combination
Volume Management	Dataset reduction techniques	Information content preservation
Replica Scaling	Horizontal Pod Autoscaler formula	Processing capacity adjustment
Stabilization Windows	Scale-down delay mechanisms	Rapid fluctuation prevention
Multi-Metric Scaling	CPU, memory, and custom metrics	Adequate capacity assurance

Table 4: Stream Processing and Kubernetes Orchestration [7, 8]

5. Query Optimization and Analytical Performance at Scale

Maintaining acceptable query performance across large document collections presents significant challenges when document structures vary widely and access patterns are unpredictable. Traditional indexing strategies that work well for uniform schemas fail when confronted with documents that have different fields, varying nesting depths, and inconsistent structures. The challenge is compounded at the petabyte scale, where even well-indexed queries can take seconds or minutes if indexes are not optimally configured. The framework proposes AI-driven index advisory systems that would continuously analyze query workloads and recommend optimal index placements.

These recommendations must balance competing goals of query latency reduction and write throughput preservation. Creating more indexes improves read performance because queries can quickly locate relevant documents through index lookups rather than scanning all documents. However, indexes also slow down write operations because every insert, update, or delete must update all affected indexes in addition to the document itself. Excessive indexing can degrade ingestion performance to the point where the system cannot keep up with incoming data, while insufficient indexing makes queries impractically slow and prevents users from accessing the data they need. The optimal index configuration depends on the specific mix of read and write operations, which may vary over time as workload patterns change.

5.1 Reinforcement Learning for Index Selection

The proposed index advisory mechanism would use reinforcement learning algorithms that model index selection as a sequential decision problem. In this formulation, the system learns optimal indexing policies through interaction with the database workload, observing the results of its decisions, and adjusting its strategy to maximize overall performance. The learning agent would observe query execution plans that show how the database processes each query, resource consumption patterns indicating CPU, memory, and I/O usage, and latency distributions showing how long different types of queries take. The agent receives reward signals based on system performance improvements, where positive rewards come from reducing query latency or improving throughput, while negative rewards come from degraded write performance or excessive index storage overhead.

Through iterative exploration of the index configuration space, the agent discovers indexing strategies that generalize across query patterns rather than overfitting to specific queries. This means learning to identify compound indexes that support frequently co-occurring predicates, where queries often filter on multiple fields together. For example, if queries frequently filter on both customer ID and order date, a compound index on those two fields would be much more effective than separate indexes on

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

each field. The agent also learns to identify partial indexes for queries targeting document subsets, where an index that only includes documents matching certain criteria can be much smaller and faster than a full index while still supporting the relevant queries.

Candidate algorithms for index advisory include Deep Q-Networks, which are well-suited for discrete action spaces where each action represents a specific decision, like creating or dropping a particular index. DQN uses a neural network to approximate the value function that predicts the long-term benefit of each action. Proximal Policy Optimization provides better performance for continuous optimization and improved stability during training by constraining how much the policy can change in each update. Advantage Actor-Critic offers parallel training capability that can speed convergence by processing multiple experiences simultaneously, learning both a policy that selects actions and a value function that evaluates states.

Algorithm selection depends on several criteria that must be evaluated for the specific deployment context. Convergence speed targets less than fifty thousand query executions for a stable policy, meaning the system should learn an effective indexing strategy without requiring excessive training data. Sample efficiency measures the ability to learn from limited production queries, which is important because extensive experimentation on production databases carries risks. Stability requires avoiding oscillating index recommendations that repeatedly create and drop the same indexes without improvement. Explainability enables the system to justify index recommendations, which is critical for gaining trust from database administrators who must understand why changes are being proposed.

The system would employ a neural network architecture processing query execution statistics to predict the performance impact of candidate index configurations. Input features would include query predicate combinations encoded as one-hot vectors indicating which fields appear together in query filters, normalized cardinality estimates showing how many documents match each predicate, sequence-encoded join orders for queries involving multiple collections, current index usage patterns indicating which existing indexes are being used and how frequently, and resource utilization metrics covering CPU percentage, I/O wait time, and memory consumption.

The network architecture would include an input layer with one hundred to five hundred neurons, depending on how many features are extracted from query statistics. The exact size depends on factors like how many unique fields exist in the collections, how many combinations of predicates need to be encoded, and what level of detail is captured about resource usage. Hidden layers would comprise two to three layers with two hundred fifty-six to five hundred twelve neurons each, providing sufficient capacity to learn complex relationships between query patterns and index effectiveness without making the network so large that it requires excessive training data or inference time. The output layer produces action probabilities for each potential index, where each output neuron corresponds to a possible index configuration and the value represents how beneficial creating that index would be.

The training process would use an experience replay buffer storing ten thousand to fifty thousand query executions, which allows the algorithm to learn from past experiences by randomly sampling batches rather than only learning from the most recent queries. This improves sample efficiency and breaks correlations between consecutive samples that can interfere with learning. Training batch size would be one hundred twenty-eight to two hundred fifty-six samples, representing the number of experiences processed together in each training iteration. Larger batches provide more stable gradient estimates but require more memory and computation per iteration. Learning rate would start at zero point zero zero zero one to zero point zero zero one and decay over time, controlling how quickly the network weights are updated in response to new information. A higher learning rate speeds initial learning but can cause instability, while decaying the rate over time helps the algorithm converge to a stable policy.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

5.2 Evaluation Metrics and Expected Performance

To validate the AI-driven index advisory system, comprehensive evaluation metrics must be measured by comparing performance against appropriate baselines. Query latency reduction targets a thirty to fifty percent improvement compared to rule-based approaches, meaning queries should execute substantially faster with AI-recommended indexes than with indexes chosen by traditional heuristics. This metric should be measured across a diverse workload, including simple filters, complex aggregations, and range queries to ensure improvements are consistent. Index recommendation accuracy measures the percentage of recommendations that actually improve performance when implemented, targeting at least eighty percent to avoid wasting resources on ineffective indexes. False positive rate measures the percentage of recommendations that degrade performance, targeting less than ten percent to minimize harm from incorrect suggestions.

Convergence time counts the number of queries needed to reach a stable policy, targeting ten thousand to fifty thousand queries. This is important because excessive training time delays the benefits of the system and may require running experiments on production databases. Comparison baselines should include manual recommendations from experienced database administrators to validate that the AI system performs at least as well as human experts, rule-based index advisors like database built-in tools to show improvements over current automated approaches, and random index selection to demonstrate that the system is actually learning rather than achieving results by chance.

Contemporary research demonstrates that AI-driven index advisors using deep reinforcement learning with query execution feedback achieve performance improvements of thirty to fifty percent in query latency compared to rule-based approaches. This represents substantial gains that can significantly improve application responsiveness and user experience. The research shows that learning agents successfully identify non-obvious indexing opportunities that human database administrators overlook in approximately forty percent of cases, suggesting that these systems can discover optimizations that even experts miss. The system employs a neural network architecture processing query execution statistics, including cardinality estimates showing how many documents match predicates, join orders indicating the sequence in which collections are combined, and operator costs showing the computational expense of different query plan steps. Training phases typically require ten thousand to fifty thousand query executions to converge to stable policies, which represents a reasonable training period that can be completed within weeks or months of normal operation for high-traffic systems.

5.3 Distributed Aggregation and Lakehouse Integration

Distributed aggregations across sharded MongoDB clusters would leverage parallel processing to achieve low-latency analytics over massive datasets. The framework would implement intelligent query routing that directs aggregation operations to appropriate shards based on predicate analysis. The query router examines filter conditions to determine which shards contain relevant documents, directing the query only to those shards and avoiding unnecessary work on shards that contain no matching documents. This minimizes cross-shard data movement, which is expensive because it requires network communication and serialization, while maximizing parallelism by allowing multiple shards to process their portions of the query concurrently.

Aggregation pipelines decompose complex analytical queries into stages that execute locally on individual shards. For example, a query computing average order value by customer segment might first filter orders by date on each shard, then group by customer and compute totals on each shard, and finally combine results across shards and compute segment averages. Intermediate results are consolidated through efficient merge operations coordinated by the query router, which receives partial results from each shard and combines them to produce the final answer. This distributed execution model scales naturally with cluster size, meaning that adding more shards increases

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

processing capacity proportionally without introducing coordination bottlenecks that would limit scalability. Each shard can work independently on its subset of data, and the merge operation typically requires much less work than the initial processing.

Integration with lakehouse environments extends analytical capabilities beyond MongoDB's native query engine through adaptive predicate pushdown and multi-stage filtering strategies. The system would maintain synchronized representations of document collections in columnar storage formats optimized for analytical workloads. Columnar formats store data by column rather than by row, which is much more efficient for analytical queries that typically access many rows but only a few columns. This organization enables better compression because values in the same column tend to be similar, and allows queries to read only the columns they need rather than entire documents.

The system uses Apache Iceberg's metadata management capabilities to enable time-travel queries that can access historical versions of data and incremental updates that efficiently refresh analytical tables with only changed documents. Apache Iceberg implements a three-level metadata hierarchy that provides powerful optimization opportunities. Catalog entries at the top level point to metadata files that describe table schemas and partitioning. These metadata files reference manifest lists that organize data files into logical groups. Manifest lists contain pointers to manifest files that track individual data files with detailed statistics, including row counts, null counts, and minimum and maximum values for each column.

This metadata hierarchy enables query planners to eliminate irrelevant data files through metadata-only operations without accessing actual data. The planner examines the statistics in manifest files to determine which data files could possibly contain rows matching the query predicates. For example, if a query filters for orders after a certain date and a data file's maximum date is before that threshold, the entire file can be skipped. If the query filters for a specific customer ID and the file's minimum and maximum customer IDs do not include that value, the file can be skipped. These checks happen in milliseconds by reading small metadata files rather than scanning large data files.

Query engines executing against lakehouse tables benefit from aggressive predicate pushdown that eliminates irrelevant data files before scanning. Predicate pushdown moves filter conditions as close to the data as possible, ideally eliminating entire files without reading them. Metadata pruning exploits partition statistics to skip entire file groups without examining their contents, where partitioning organizes data files by values of certain columns, like date or geographic region. Production deployments demonstrate query acceleration factors of twenty to one hundred times for selective predicates operating on properly partitioned tables, meaning queries that access only a small fraction of the data see dramatic speedups. For example, a query for last week's orders in a table partitioned by date might need to read only seven-day partitions out of years of historical data, and metadata filtering could further eliminate files within those partitions that don't match other predicates.

This hybrid architecture provides query flexibility, enabling users to select execution engines based on workload characteristics. Operational queries that need real-time results and access to recent data would use MongoDB's native query engine, which is optimized for low latency and handles concurrent reads and writes efficiently. Analytical queries that perform complex aggregations over large historical datasets would use the lakehouse representation, which is optimized for scanning large volumes of data and supports advanced analytical functions. The framework manages consistency between operational and analytical representations through change data capture mechanisms that stream changes from MongoDB to the lakehouse, ensuring both systems stay synchronized without impacting operational performance.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

6. Experimental Design and Validation Framework

Section 6.1 Performance Benchmarking Methodology (Revised)

Performance benchmarking would require a carefully designed test environment that realistically represents production deployment scenarios. The hardware configuration should include a MongoDB cluster with ten shard servers, where each server has sixteen cores and sixty-four gigabytes of RAM to provide sufficient resources for processing high-volume workloads. The Flink cluster should include twenty task managers, each with eight cores and thirty-two gigabytes of RAM, providing distributed processing capacity for stream transformations. The Kubernetes cluster should span thirty nodes across three availability zones to enable realistic testing of fault tolerance and geographic distribution. The software stack would use MongoDB version seven or later to leverage recent performance improvements, Apache Flink version one point eighteen or later for current stream processing capabilities, Kubernetes version one point twenty-eight or later for recent orchestration features, and a vector database such as Pinecone, Milvus, or Weaviate for storing and searching document embeddings.

Benchmark datasets should represent diverse document characteristics using publicly available data sources to ensure the framework can handle real-world variety while maintaining complete independence from any proprietary or organizational data. Proposed publicly available datasets include the TPC-H benchmark dataset, which is a widely recognized decision support benchmark containing synthetic e-commerce-style transactions with varying schemas representing products, orders, and customers, providing approximately one hundred million records at scale factor one hundred. The Common Crawl dataset offers publicly accessible web documents demonstrating semi-structured content with varying schemas, where documents exhibit frequent structural changes as websites evolve their formats. The Yelp Open Dataset provides publicly released business reviews and user data with realistic schema variations, containing millions of JSON documents with inconsistent field presence and nesting depths. The GitHub Archive dataset contains publicly available repository events demonstrating high-velocity data streams with schema evolution as new event types are introduced, featuring billions of JSON documents tracking software development activities.

For IoT simulation, publicly available datasets such as the Intel Berkeley Research Lab sensor data or the CRAWDAD wireless network traces provide realistic sensor readings with schema drift characteristics. For healthcare simulation without using actual patient data, the MIMIC-III Clinical Database demo dataset offers de-identified medical records in the public domain, or synthetic healthcare data generators like Synthea can create realistic medical documents with complex nesting and relationships while ensuring complete data privacy and independence from any healthcare organization.

The schema evolution characteristics of these publicly available datasets should be carefully controlled to enable meaningful evaluation. A baseline of fifty known schema versions should be established representing the documented structures present at the start of testing, derived from analyzing the public dataset documentation. An evolution rate of five to ten new fields per week should be simulated by programmatically injecting schema variations into the public datasets to mimic realistic schema drift observed in academic research literature. Structural changes including two to three major hierarchy changes per month should be introduced through controlled transformation of the public dataset structures to test the system's ability to handle significant reorganizations, not just field additions.

All experimental data, results, and benchmarks would be derived solely from these publicly available sources or synthetically generated data, ensuring complete independence from any specific organization or employer. The validation approach relies exclusively on open-source tools, public

datasets, and publicly documented research methodologies to maintain transparency and reproducibility.

6.1 Performance Benchmarking Methodology

Performance benchmarking would require a carefully designed test environment that realistically represents production deployment scenarios. The hardware configuration should include a MongoDB cluster with ten shard servers, where each server has sixteen cores and sixty-four gigabytes of RAM to provide sufficient resources for processing high-volume workloads. The Flink cluster should include twenty task managers, each with eight cores and thirty-two gigabytes of RAM, providing distributed processing capacity for stream transformations. The Kubernetes cluster should span thirty nodes across three availability zones to enable realistic testing of fault tolerance and geographic distribution. The software stack would use MongoDB version seven or later to leverage recent performance improvements, Apache Flink version one point eighteen or later for current stream processing capabilities, Kubernetes version one point twenty-eight or later for recent orchestration features, and a vector database such as Pinecone, Milvus, or Weaviate for storing and searching document embeddings.

Benchmark datasets should represent diverse document characteristics to ensure the framework can handle real-world variety. An e-commerce transactions dataset with one hundred million documents would include varying schemas representing products with different attribute sets, orders with different item counts, and users with varying profile completeness. An IoT sensor data dataset with five hundred million documents would feature high velocity arrival rates simulating continuous sensor streams and schema drift occurring every two weeks as new sensor types are deployed or measurement formats change. A healthcare records dataset with fifty million documents would exhibit deep nesting with three to seven levels representing complex medical concepts and their relationships. A social media posts dataset with two hundred million documents would demonstrate semi-structured content with frequent new fields as users adopt new ways of expressing themselves.

The schema evolution characteristics of these datasets should be carefully controlled to enable meaningful evaluation. A baseline of fifty known schema versions should be established, representing the documented structures present at the start of testing. An evolution rate of five to ten new fields per week should be maintained to simulate realistic schema drift observed in production systems. Structural change, including two to three major hierarchy changes per mon, should be introduced to test the system's ability to handle significant reorganizations, not just field additions.

Performance metrics should comprehensively cover multiple aspects of system behavior. Throughput metrics measure events processed per second for both the baseline traditional ETL approach and the proposed framework, with a target of demonstrating two to five times improvement. These measurements should represent averages over one-hour sustained load periods to account for warmup effects and ensure the system can maintain performance over extended periods. Latency metrics should track the distribution of end-to-end processing times, reporting the fiftieth, ninety-fifth, and ninety-ninth percentiles. The target should be maintaining less than one hundred milliseconds at the ninety-fifth percentile, even at high throughput, meaning that ninety-five percent of events complete processing within one hundred milliseconds. Measurements should distinguish latency across document complexities, recognizing that simple documents should process faster than complex documents with many fields or deep nesting.

Resource utilization monitoring should track CPU usage across the cluster, showing what percentage of available processing capacity is consumed, memory consumption patterns indicating whether the system fits comfortably within available memory or experiences pressure, network bandwidth utilization measuring data movement between components, and storage I/O operations tracking disk

access patterns. These metrics help identify bottlenecks and ensure the system uses resources efficiently.

Scalability testing should validate that the architecture scales effectively as resources are added. Linear scaling tests would add shards and nodes while measuring throughput increase, targeting greater than eighty percent efficiency. This means that doubling the number of shards should at least increase throughput by eighty percent, so eight shards should provide at least six point four times the throughput of one shard rather than the ideal eight times. Efficiency below one hundred percent is expected due to coordination overhead, network communication, and other factors that don't scale perfectly. Burst handling tests would suddenly increase traffic by ten times and measure how quickly the system recovers through autoscaling, targeting recovery to stable performance within five minutes.

6.2 AI Component Evaluation Methodology

AI component evaluation requires rigorous testing with ground-truth data to quantify accuracy and identify failure modes. Schema inference accuracy testing would prepare a ground-truth dataset with ten thousand documents across one hundred schema versions, where human experts meticulously label all entities, attributes, and relationships. This creates a reference standard against which AI predictions can be compared. The AI models would then process unlabeled versions of these same documents, and their predictions would be compared to the ground truth to calculate accuracy metrics.

Precision measures true positives divided by true positives plus false positives, indicating what percentage of entities the model identifies are actually correct. A precision of eighty-five percent means that when the model claims to have found an entity, it is correct eighty-five percent of the time and wrong fifteen percent of the time. Recall measures true positives divided by true positives plus false negatives, indicating what percentage of actual entities the model successfully identifies. A recall of eighty percent means the model finds eighty percent of the entities that actually exist, but misses twenty percent. The F1 score computes two times precision times recall divided by the precision plus the recall, providing a single metric that balances both concerns. The target F1 score of greater than eighty-two percent ensures the model achieves good performance on both precision and recall, rather than optimizing one at the expense of the other.

Model comparison should test multiple approaches to identify the best option for specific deployment contexts. GPT-4 API would serve as a baseline representing high accuracy but high cost, establishing the best performance achievable without cost constraints. Fine-tuned BERT would be tested as a middle ground expected to provide medium accuracy at low cost by using a smaller model specifically trained for this task. A rule-based system would provide a comparison baseline showing what can be achieved without machine learning, helping quantify the value added by AI approaches.

Drift detection accuracy testing would inject one thousand schema changes into the document stream with carefully controlled composition. Seven hundred genuine evolutionary changes representing real schema evolution should trigger adaptation, such as adding fields that will persist or restructuring hierarchies that reflect business process changes. Three hundred transient anomalies representing temporary data quality issues should be ignored, such as occasional malformed documents or experimental fields that never become standard. The system's classification accuracy would be measured, showing how well it distinguishes these two categories.

True positive rate measures correctly identified genuine changes divided by total genuine changes, targeting greater than ninety percent. This means when real schema evolution occurs, the system should detect it ninety percent of the time. False positive rate measures incorrectly flagged anomalies divided by total anomalies, targeting less than five percent. This means that when transient anomalies occur, the system should avoid incorrectly treating them as significant changes more than ninety-five

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

percent of the time. Detection latency measures time from change occurrence to detection, targeting less than five minutes so the system responds quickly to genuine evolution without excessive delay.

Index advisory effectiveness testing would collect fifty thousand production queries from the test workload, representing realistic query patterns. The reinforcement learning agent would be trained on the first forty thousand queries, learning from their execution plans, resource consumption, and latency. The trained agent would then be tested on the remaining ten thousand queries that it has not seen during training to evaluate how well it generalizes. Performance would be compared against manual database administrator recommendations collected from experienced professionals and against a rule-based advisor representing current automated approaches.

Query latency improvement measures average percentage reduction compared to baseline, targeting thirty to fifty percent based on research showing typical improvements from AI-driven index advisors. Recommendation accuracy measures the percentage of recommendations that improve performance when implemented, targeting at least eighty percent to ensure most suggestions are beneficial. Convergence speed measures the number of queries needed to reach a stable policy, targeting ten thousand to fifty thousand queries, representing a reasonable training period. Cost analysis examines write throughput impact from additional indexes by measuring how much insert and update performance degrades as indexes are added, ensuring that read performance improvements justify write performance costs.

6.3 Cost-Benefit Analysis Methodology

Cost-benefit analysis requires projecting both the costs of implementing and operating the AI-enabled system and the benefits it provides through improved efficiency and reduced manual effort. Monthly cost projections should be calculated for a representative workload of ten million documents per month to provide concrete numbers that organizations can use for planning.

LLM inference costs vary dramatically based on which models are used and how they are deployed. Using GPT-4 API for all documents would cost approximately ten thousand to thirty thousand dollars per month, based on current pricing and typical document sizes. Using the Claude-3.5 API for all documents would cost approximately three thousand to fifteen thousand dollars per month, providing a more cost-effective option with somewhat lower accuracy. Self-hosting Llama-3 would incur approximately two thousand dollars per month in infrastructure costs, including servers, networking, and maintenance, when amortized across all documents processed.

The recommended hybrid approach significantly reduces costs by using different strategies for different documents based on their characteristics. Using rule-based processing for eighty percent of documents that follow known patterns costs essentially nothing beyond normal computing resources. Applying LLM processing to the remaining twenty percent of documents that exhibit novel or ambiguous structures costs approximately six hundred to three thousand dollars per month. The total estimated cost of two thousand six hundred to five thousand dollars per month represents a practical balance between accuracy and cost for many organizations.

Engineering time savings represent the primary benefit of the AI-enabled approach by reducing the manual effort required to maintain data pipelines. The traditional manual approach requires approximately two engineers spending fifty percent of their time on schema maintenance activities like updating transformation rules and fixing breaking changes, costing approximately one hundred twenty thousand dollars per year. An additional engineer spends twenty-five percent of their time on pipeline debugging to investigate and resolve data quality issues, costing approximately thirty thousand dollars per year. The total manual cost of approximately one hundred fifty thousand dollars per year represents a significant ongoing burden.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

The AI-enabled approach substantially reduces these labor requirements. Initial setup requires a one-time investment of approximately fifty thousand dollars for developing the framework, training initial models, and deploying infrastructure. AI inference costs approximately thirty thousand to sixty thousand dollars per year based on the monthly projections discussed above. Ongoing maintenance requires only twenty percent of the previous engineering time because many issues are handled automatically, costing approximately thirty thousand dollars per year. The total AI cost of one hundred ten thousand to one hundred forty thousand dollars per year after the first year represents a meaningful reduction.

Break-even analysis shows that the first year has higher costs due to the setup investment, but years two and onward achieve a twenty to forty percent cost reduction compared to the manual approach. Return on investment becomes positive after eighteen to twenty-four months, meaning the cumulative savings exceed the initial investment within two years. Organizations with higher document volumes would see faster payback because the per-document AI costs decrease with scale, while the engineering time savings remain constant or grow.

6.4 Validation Criteria for Production Readiness

For the framework to be considered production-ready, specific criteria must be met across multiple dimensions. These criteria establish minimum thresholds that must be achieved before recommending deployment in production environments where data quality and system reliability are critical.

Performance requirements establish the baseline operational characteristics the system must maintain. Throughput must exceed one hundred thousand events per second sustained, demonstrating the system can handle high-volume workloads without falling behind. Latency must maintain the ninety-fifth percentile below one hundred milliseconds and the ninety-ninth percentile below five hundred milliseconds, ensuring acceptable responsiveness even for the slowest events. Scalability must achieve greater than eighty percent efficiency when adding resources, validating that the architecture scales effectively without excessive overhead.

AI accuracy requirements ensure the machine learning components provide sufficient quality to rely on their decisions. Schema inference must achieve precision greater than eighty-five percent and recall greater than eighty percent, meaning the system correctly identifies most entities and relationships with few errors. Drift detection must achieve a true positive rate greater than ninety percent and a false positive rate less than five percent, meaning the system reliably detects genuine changes while avoiding false alarms. Index advisory must demonstrate greater than thirty percent average latency improvement, proving the AI recommendations significantly enhance query performance.

Operational requirements ensure the system behaves reliably in production environments. Recovery time must be less than five minutes from failure, meaning the system quickly restores service if components crash. Data loss must be zero through exactly-once semantics, guaranteeing that no events are lost or duplicated even during failures. Monitoring must provide real-time visibility into all components, enabling operators to understand system behavior and quickly diagnose issues.

Note on Data Sources and Independence: All datasets, benchmarks, and examples referenced in this paper are based on publicly available sources including TPC-H benchmark data, Common Crawl web archives, Yelp Open Dataset, GitHub Archive, Intel Berkeley Research Lab sensor data, CRAWDAD wireless traces, MIMIC-III demo dataset, and synthetic data generators like Synthea. No proprietary organizational data, production systems, or employer-specific information is used or referenced. The framework is designed as an independent research contribution using only public domain resources and open-source technologies. Organizations implementing this framework would

use their own data sources, but all validation and benchmarking described herein relies exclusively on publicly accessible datasets to ensure reproducibility and independence from any specific entity.

7. Limitations and Future Work

7.1 Current Limitations

This paper presents a conceptual framework with proposed design patterns. Several important limitations must be acknowledged before considering deployment in production environments. No production deployment or benchmark testing has been conducted, meaning the performance characteristics discussed are based on related research and theoretical analysis rather than measured results from this specific system. The architecture may behave differently in practice than predicted, and unexpected interactions between components may emerge.

Performance improvements are estimated based on related research showing typical gains from similar approaches, but actual results may vary depending on specific workload characteristics, hardware configurations, and deployment contexts. AI model accuracy is projected based on the capabilities demonstrated by current language models and embedding techniques, but actual accuracy when applied to specific document collections may differ. The models may struggle with domain-specific terminology, unusual document structures, or edge cases not well represented in training data.

Cost-benefit analysis is theoretical without real-world confirmation, calculated based on current pricing and typical usage patterns. Actual costs may vary depending on factors like document size distribution, query complexity, and operational requirements. Organizations may experience different cost-benefit ratios depending on their specific circumstances.

AI component uncertainties create risks that must be carefully managed. Model selection requires testing multiple candidates in specific deployment contexts because performance varies with document characteristics, domain terminology, and desired accuracy-cost trade-offs. Training data requirements may vary significantly across industries and document types, where some domains have clear patterns that models learn easily, while others have ambiguous or inconsistent structures that challenge current techniques. Inference costs and latencies depend on deployment choices between cloud APIs versus self-hosted models, where cloud APIs provide simplicity but potentially higher cost, while self-hosting requires infrastructure and expertise but may provide better economics at scale.

Accuracy trade-offs between different AI approaches need empirical validation through controlled testing. Some approaches may excel at certain types of schema changes while struggling with others. For example, language models may handle semantic understanding well but miss purely structural patterns that simpler algorithms would catch. Conversely, rule-based approaches may efficiently handle well-defined patterns but fail on ambiguous cases that language models can interpret through context.

Scalability assumptions throughout the paper may not hold at extreme scales exceeding one thousand nodes. Coordination overhead, network contention, and other factors that are negligible at moderate scale may become bottlenecks at a very large scale. The architecture may require modifications like hierarchical coordination or different partitioning strategies to scale to the largest deployments. Network bottlenecks in cross-datacenter deployments are not fully addressed, where geographic distribution introduces latency and bandwidth constraints that may limit throughput or require different architectural patterns. Storage costs for maintaining version history and embeddings are not quantified, where storing multiple schema versions and high-dimensional vectors for all documents may consume substantial storage that must be factored into the total cost of ownership.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Operational complexity represents a significant challenge that organizations must consider. The system requires expertise in multiple domains, including databases, machine learning, and stream processing, making it difficult to find personnel with the necessary breadth of knowledge. Team members need to understand how MongoDB sharding works, how Flink stream processing operates, how to train and deploy machine learning models, and how these components interact. Debugging and troubleshooting AI-driven decisions may be challenging because machine learning models can behave in unexpected ways that are difficult to explain or predict. When the schema evolution layer makes an incorrect decision, understanding why it made that choice requires inspecting model weights, training data, and inference logic that may not be transparent. Model drift over time requires ongoing monitoring and retraining, where models that perform well initially may degrade as data distributions change, new patterns emerge, or edge cases accumulate. Organizations must establish processes for detecting performance degradation and periodically retraining or updating models.

7.2 Future Research Directions

Several directions warrant investigation to address limitations and extend the framework's capabilities. Immediate next steps should focus on validating the core concepts through prototype implementation. Building a minimal viable system with core components would demonstrate feasibility and reveal practical challenges not apparent in conceptual design. This prototype should include basic MongoDB sharding, Flink stream processing, and simple AI components to validate that the architecture works as intended. Benchmark testing using the methodology outlined in Section 6 would provide empirical data quantifying actual performance, comparing it against theoretical predictions, and identifying areas needing optimization. Model evaluation would compare candidate AI models for each component using real document collections to determine which approaches work best in practice. Cost analysis would validate cost estimates with real deployment by measuring actual inference costs, infrastructure expenses, and operational overhead.

Advanced research areas offer opportunities for significant improvements beyond the baseline framework. Incorporating causal inference techniques to distinguish correlation from causation in structural changes would improve understanding of the semantic implications of schema modifications. Current approaches treat all changes that co-occur as potentially related, but causal inference could identify whether one change actually causes another or if they merely happen together coincidentally. This would enable more sophisticated adaptation strategies that account for causal dependencies between schema elements.

Investigating recent advances in learned index structures could further improve query performance. Traditional indexes use fixed data structures like B-trees, but learned indexes use machine learning models to predict data locations based on patterns in keys. These learned structures can be more compact and faster than traditional indexes when data has predictable patterns. Applying neural query optimization techniques could improve query planning by learning which execution strategies work well for different query patterns rather than relying on heuristic cost models that may not accurately predict actual performance.

Developing hybrid approaches combining learned and traditional indexes could provide robustness by using learned indexes for common patterns while falling back to traditional indexes for unusual queries. This prevents worst-case performance degradation if learned models encounter patterns they weren't trained on.

Extending the framework to support multi-tenant scenarios with strong isolation guarantees would broaden applicability to cloud-native platforms where multiple organizations share infrastructure. Each tenant would need isolated schema evolution models so that one tenant's documents don't influence another tenant's transformations. Resource quotas would prevent any single tenant from

consuming excessive resources. Priority handling would ensure that high-priority tenants receive better service quality when resources are constrained.

Enabling organizations to collaboratively train schema evolution models through federated learning would allow sharing structural insights while preserving data privacy. Multiple organizations could contribute to training a shared model without exposing their actual documents by training locally and only sharing model updates. This would enable smaller organizations to benefit from patterns learned across many organizations while maintaining confidentiality. Developing consensus mechanisms for schema standardization could help establish industry-wide conventions that reduce fragmentation and improve interoperability.

Developing techniques to explain AI-driven schema mapping decisions would build confidence and enable debugging. Current machine learning models often function as black boxes where their reasoning is opaque. Explainable AI techniques could identify which features influenced decisions, how confident the model is in its predictions, and what alternative interpretations were considered. Building confidence scoring for automatic versus manual intervention would help the system know when to proceed automatically and when to request human review. Creating comprehensive audit trails for compliance and debugging would record all decisions, their justifications, and their outcomes, enabling retrospective analysis when issues arise.

Extending the framework to support edge computing scenarios would enable preprocessing at data sources before transmitting to central systems. Developing lightweight models for resource-constrained environments would allow edge devices with limited CPU and memory to perform basic transformations locally. Implementing hierarchical processing where edge devices perform initial filtering and normalization, regional hubs perform more complex enrichment, and central clouds perform final integration and analytics would reduce bandwidth requirements and improve latency for edge-originated data.

7.3 Call for Collaboration

The authors encourage collaboration from multiple communities to validate and extend this framework. Researchers with access to large-scale document datasets for validation could provide the diverse, realistic data needed to thoroughly test the approach across different domains and schema evolution patterns. Industry practitioners who can provide production deployment feedback would offer practical insights into operational challenges, performance characteristics, and useful features that academic research might overlook. AI and ML experts interested in domain-specific model optimization could develop better techniques for document understanding, schema inference, and drift detection by focusing on the unique characteristics of this problem domain. Open-source contributors could build reference implementations that demonstrate the concepts, provide starting points for organizations wanting to adopt the approach, and create a community ecosystem around the framework.

Collaboration opportunities include providing document datasets for benchmarking, sharing production deployment experiences and lessons learned, contributing to open-source reference implementations, conducting comparative studies of different AI model architectures, developing domain-specific adaptations for industries like healthcare, finance, or e-commerce, and creating educational materials to help practitioners understand and apply the concepts. Organizations interested in collaboration should consider joining existing open-source data infrastructure projects, sharing anonymized schema evolution patterns from their systems, participating in benchmark development and standardization efforts, contributing engineering resources to reference implementations, and sponsoring research initiatives focused on production validation.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Conclusion

This paper presents a conceptual framework for event-driven document processing that combines MongoDB as a scalable event store, Apache Flink for stream processing on Kubernetes, and AI-driven schema management to address fundamental challenges in document ingestion, including unbounded schema drift, inconsistent structures, and unpredictable relationships. The framework's key contributions include a comprehensive architectural design that treats document changes as first-class events processable through sophisticated stream processing infrastructure, a detailed AI integration strategy proposing specific uses for large language models to extract entities and understand semantics, embedding models to match similar fields across different document types, and reinforcement learning to optimize index placement based on observed query patterns. However, this work is explicitly conceptual and requires substantial validation before production deployment, where the proposed AI-enabled schema evolution mechanisms represent design proposals rather than validated implementations, and specific performance improvements like two to five times throughput gains and thirty to fifty percent query latency reductions are estimated based on related research rather than measured from this specific system. The experimental design outlined in Section 6 provides a comprehensive roadmap for validation, including performance benchmarking, comparing throughput and latency against baseline systems, AI component evaluation, on measuring precision, recall, and F1 scores for schema inference, cost analysis tracking actual expenses and savings, and comprehensive validation criteria establishing specific thresholds that must be met before recommending production use. The architectural patterns established through this framework have implications extending beyond the specific technology stack, where the principle of treating document changes as events applies broadly to any system handling evolving structured data, the integration of machine learning for schema understanding represents a general pattern applicable to diverse data management challenges, and the hybrid approach balancing AI costs with accuracy provides a template for practical machine learning deployment. Organizations considering adopting these concepts should start with pilot implementations on non-critical workloads, conduct thorough cost-benefit analysis for specific use cases, ensuring the approach makes economic sense, invest in monitoring and observability infrastructure before deploying AI components, plan for iterative refinement based on production feedback, and maintain fallback mechanisms to rule-based processing to ensure the system can continue operating even if AI components fail. By providing detailed implementation guidance covering model selection criteria with specific options like GPT-4 versus BERT versus Llama, cost considerations with dollar amounts ranging from two thousand six hundred to five thousand dollars monthly for hybrid approaches, and validation methodology with concrete metrics including throughput exceeding one hundred thousand events per second and schema inference precision exceeding eighty-five percent, this paper serves as a foundation for future research and development in adaptive, intelligent document processing systems at scale, offering a vision for how document ingestion could evolve from brittle, maintenance-intensive manual processes to intelligent, self-regulating systems that adapt automatically to changing requirements while acknowledging that realizing this vision requires substantial empirical validation beyond the conceptual foundation and validation roadmap provided here.

References

[1] Roshni Verma, "MongoDB Architecture: A Comprehensive Guide," Medium, 2024. [Online]. Available: https://medium.com/@roshniverma021/mongodb-architecture-a-comprehensive-guide-c72799347b22

[2] Fabian Hueske, Vasiliki Kalavri, "Stream Processing with Apache Flink," O'Reilly Media, 2019. [Online]. Available: https://www.oreilly.com/library/view/stream-processing-with/9781491974285/ch01.html

[3] Shanika Wickramasinghe, "MongoDB Sharding: Concepts, Examples & Tutorials," BMC Blogs, 2020. [Online]. Available: https://www.bmc.com/blogs/mongodb-sharding-explained/

[4] Macrometa, "Apache Spark Vs Flink," Macrometa Documentation, 2024. [Online]. Available: https://www.macrometa.com/event-stream-processing/spark-vs-flink

[5] Jasper Flour, "What is Automated Data Preparation?" Mammoth, 2025. [Online]. Available: https://mammoth.io/blog/automated-data-preparation/

[6] Tomas Mikolov et al., "Distributed Representations of Words and Phrases and their Compositionality," arXiv, 2013. [Online]. Available: https://arxiv.org/abs/1310.4546

[7] Netscribes, "How automated data preparation is revolutionizing AI workflows," 2025. [Online]. Available: https://www.netscribes.com/expert-speak/data-preparation-essentials-for-ai-projects

[8] Kubernetes, "Horizontal Pod Autoscaling," 2024. [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

[9] Bailu Ding et al., "AI Meets AI: Leveraging Query Executions to Improve Index Recommendations," ACM Digital Library, 2019 [Online]. Available: https://dl.acm.org/doi/10.1145/3299869.3324957

[10] Tomer Shiran, et al., "Apache Iceberg: The Definitive Guide," O'Reilly Media, 2024. [Online]. Available: https://www.oreilly.com/library/view/apache-iceberg-the/9781098148614/