Development Framework for Big Data Applications

KAPPALA SHANTHI LATHA
Research Scholar Dept of Computer Science & Engg
Arni University-Himachal Pradesh

DR. PRASADU PEDDI
Research Supervisor Dept of Computer Science & Engg
Arni University-Himachal Pradesh

DR. MARAM ASHOK
Co- Supervisor Dept of Computer Science & Engg
Principal & Professor Malla Reddy Institute of Engineering & Technology, Hyderabad, Telangana

ABSTRACT

The advent of big data has transformed the way businesses and organizations collect, store, process, and derive insights from massive volumes of data, and building such applications requires specialized development frameworks able to handle the scale, variety, and complexity of data generated from diverse sources. Traditional data-processing tools are not equipped for these workloads, and the primary challenges—data ingestion, storage, processing, and analysis, together with real-time streaming—necessitate purpose-built technology stacks. This paper examines the key considerations in selecting a development framework for big data applications. We first present the organizational Big Data Framework, a vendor-independent structure of six capabilities—strategy, architecture, algorithms, processes, functions, and artificial intelligence—that an enterprise must develop to embed a successful big data practice. We then survey the dominant open-source processing frameworks along the batch/stream axis, covering Hadoop MapReduce, Apache Spark, Flink, Storm, and Samza, and consolidate recent comparative evidence on their performance, cost, ease of use, fault tolerance, and suitability for machine-learning and real-time analytics workloads. Reported benchmark studies show that engine choice, data-storage format, and workload type interact strongly, so no single framework dominates all conditions. Finally, we map framework capabilities to application classes—reliable platform engineering, governance, interactive exploration of unstructured data, and domain deployments in health, insurance, and environmental prediction—and identify open challenges in throughput, storage efficiency, and streaming program development.

Keywords— Big Data, Big Data Analytics, Apache Spark, Apache Hadoop, Stream Processing, Development Framework

I. INTRODUCTION

The era of big data has revolutionized how we collect, store, process, and analyze vast amounts of data generated from diverse sources. Big data applications have become essential for businesses, researchers, and organizations seeking valuable insights and patterns hidden within this massive sea of information. However, processing such immense volumes efficiently requires specialized tools: a *big data development framework* is a comprehensive set of tools, libraries, and methodologies designed to handle the unique challenges posed by massive data sets—data storage, data processing, data analysis, and real-time/streaming data—challenges with which traditional data-processing tools cannot cope.

Two complementary notions of "framework" matter in practice. The first is *organizational*: enterprises struggle to embed a successful big data practice, and structured capability models are needed that span strategy through technology. Recent work formalizes this need, from adoption frameworks that accelerate enterprise big data uptake [4] to engineering frameworks for designing, developing, testing, and deploying *reliable* big data platforms [1] and governance frameworks that keep the resulting data estate compliant and intelligible [2]. The second notion is *technical*: the open-source processing engines—Hadoop MapReduce, Apache Spark, Flink, Storm, and Samza—on which applications are actually built. Here a growing body of comparative evidence guides selection: comprehensive HiBench-based analyses of Hadoop versus Spark on large data sets [12], evaluations of the two engines for compute-intensive parallel tasks [13], critical analyses of their architectural trade-offs [15], and studies showing that even the choice of data-storage format within the Hadoop system measurably shifts Spark performance [11].

This paper connects the two notions. Its contributions are: (i) a presentation of the six-element organizational Big Data Framework—strategy, architecture, algorithms, processes, functions, and artificial intelligence—as a vendor-independent reference model; (ii) a structured survey of the most popular big data processing frameworks along the batch/stream axis, consolidating recent comparative benchmarks [12], [13], [14], [15]; and (iii) a mapping from framework capabilities to deployed application classes, including interactive exploration of unstructured data [3], federated medical image analysis [8], insurance risk estimation [6], pandemic prediction [7], and environmental forecasting on Hadoop–Spark clusters [16]. The remainder of the paper presents the organizational framework (Section II), the technical framework landscape (Section III), comparative analysis (Section IV), applications (Section V), and conclusions (Section VI).

II. THE ORGANIZATIONAL BIG DATA FRAMEWORK

Frameworks provide structure. The core objective of the Big Data Framework is to provide a structure for enterprise organizations that aim to benefit from the potential of big data: long-term success requires more than skilled people and technology—it requires structure and capabilities, from the definition of a big data strategy to the technical tools an organization should have. The framework is vendor-independent, applicable regardless of technology choice; provides a common reference model usable across departmental and country boundaries; and identifies core, measurable capabilities in each of its six domains so the organization can develop over time. Comparable industrial models confirm the need: the ADiBA adoption framework operationalizes enterprise big data uptake end-to-end [4], while reliability-engineering frameworks structure the design, development, testing, and deployment of dependable big data platforms [1] and governance frameworks add the compliance and stewardship layer [2].

The framework consists of six main elements:

A. Big Data Strategy. Data has become a strategic asset, and the capability to analyze large data sets and discern patterns can provide competitive advantage—Netflix's production decisions and Alibaba's supplier-lending choices are canonical examples. To achieve tangible returns, enterprises need a sound strategy defining where to focus analysis effort, since the possibilities are effectively endless.

B. Big Data Architecture. Working with massive data sets requires infrastructure to store and process large quantities of data. This element considers the technical capabilities of big data environments, the roles present within an architecture, and design best practices, taking the vendor-neutral NIST big data reference architecture as its baseline. Empirical studies underline that architectural detail matters: even the storage format selected within the Hadoop system changes downstream Spark performance materially [11].

C. Big Data Algorithms. A fundamental capability is a thorough grounding in statistics and algorithms—unambiguous specifications for solving classes of problems—through which valuable knowledge is obtained from large volumes of data. At scale, algorithmic choices interact with the platform: genetic-algorithm-based instance selection, for example, is executed on open-source distributed frameworks to remain tractable on big data [9].

D. Big Data Processes. Processes bring structure, measurable steps, and day-to-day manageability, embedding big data expertise as an organizational practice so analysis becomes less dependent on individuals and value capture is sustained long term.

E. Big Data Functions. This element addresses the organizational aspects: structuring roles and responsibilities, organizational culture, and critical success factors, including establishing a Big Data Center of Excellence.

F. Artificial Intelligence. The final element addresses the relation between big data and AI as the logical next step for organizations that have built up the other capabilities, depicted deliberately as a lifecycle: AI continuously learns from the organization's big data to provide lasting value. Deployed exemplars include distributed deep-learning frameworks for federated medical image analysis [8] and multimodal analysis frameworks that fuse heterogeneous data types under one architecture [10].

III. BIG DATA PROCESSING FRAMEWORKS

Big data processing frameworks are complex systems designed to process large amounts of data for many purposes—business intelligence and analytics, machine learning and AI, and streaming/real-time analytics. Data is basically processed in two categories, *batch processing* and *stream processing*, and frameworks are selected according to which conditions the workload must satisfy. The typical big data project workflow proceeds from problem formulation through data collection, storage and transfer (infrastructure), data analysis, and reporting/visualization to evaluation.

A. Hadoop. Hadoop is one of the classic and most highly developed frameworks in use, almost synonymous with big data. It performs batch processing—data is grouped into batches for further processing—with the Hadoop Distributed File System (HDFS) and MapReduce as the primary ecosystem components for storage and computation respectively. Hadoop remains a less expensive choice when specialized hardware is not available, and it performs well even with limited main memory, when only part of the data fits in memory [13], [15].

B. Apache Spark. Spark is another major, frequently used framework. Unlike Hadoop's disk-oriented batch model, Spark processes data in memory as it enters the system, making it faster and more flexible than Hadoop for iterative workloads; it uses HDFS for storage since it has no storage layer of its own, and includes MLlib, SparkSQL, and GraphX components. Benchmark studies with HiBench confirm large speedups over MapReduce on iterative and machine-learning workloads while also quantifying the memory cost at which those speedups are obtained [12]; comparative studies on stock-market data [14] and compute-intensive parallelization tasks [13] report the same qualitative pattern.

C. Apache Flink. Flink is a streaming dataflow engine facilitating distributed computation, combining batch and real-time processing in one framework. It offers streaming, static-data, and SQL-like query APIs for Java, Scala, and Python, in-house machine-learning and graph libraries, and features including high performance, low latency, stateful computation, continuous streaming, and fault tolerance.

D. Apache Storm. Storm is a real-time distributed computation system for processing streaming data, benchmarked at processing on the order of a million tuples per second. It is highly scalable, applicable to real-time analysis and machine learning where data velocity is high, integrates with Hadoop ecosystems and YARN, and is fast, reliable, fault-tolerant, and easy to operate once deployed.

E. Apache Samza. Samza is a real-time distributed stream-processing framework built on Apache Kafka for messaging and YARN for resource management. It has a simple API and is fault-tolerant, durable, scalable, and pluggable; it provides processor isolation through YARN and resource isolation with Linux cgroups, and supports snapshotting and restoration of processor state for large per-partition state.

F. Stream Analytics. Stream processing operates on sequences of data objects generated continuously at high rate—potentially unbounded—which must be processed within small time windows for near-real-time response with low latency. Most such data originates from IoT devices, sensors, and logs; unlike batch processing, which stores all data before processing, stream computing processes data in motion before it is stored, bringing the data to the analysis ("data-driven"). A stream processor must sustain very large streams while keeping latency—the time before the result appears—low, which is essential for monitoring applications and sensor analytics.

IV. COMPARATIVE ANALYSIS AND FRAMEWORK SELECTION

A. Batch Engines: Hadoop versus Spark. Recent empirical comparisons converge on a consistent picture. Ahmed et al. [12] run the HiBench suite over large data sets and find Spark substantially faster on iterative and ML workloads while Hadoop remains competitive—and more memory-frugal—on one-pass, I/O-bound jobs. Döschl et al. [13] evaluate both engines for the parallelization of compute-intensive tasks and report that the winner depends on task granularity and cluster memory. Sewal and Singh [15] present a critical architectural analysis, attributing Spark's advantage to in-memory resilient distributed datasets and Hadoop's robustness to its disk-based shuffle. Gupta and Sharma [14] confirm the pattern on stock-market analytics. Two practical corollaries follow for developers: first, the storage format matters—Belov et al. [11] show experimentally that choosing among formats in the Hadoop system (e.g., Avro, ORC, Parquet) shifts Apache Spark query performance significantly; second, hybrid deployments are normal—Wei and Chou [16] obtain operational typhoon rainfall prediction by running the analytics on a combined Hadoop–Spark parallel computing cluster.

B. Batch Computing Framework Comparison. At the programming-model level, MapReduce, Dryad, and Mahout represent three archetypes. MapReduce is a distributed programming system processing map/reduce jobs (many inputs, few outputs); it is simple and supports fault tolerance, data consistency, concurrency, and very high processing performance. Dryad is likewise a distributed programming system but processes arbitrary directed acyclic graphs (arbitrary input/output); it is more complex, handling more complicated computations, with fault tolerance, data consistency, concurrency, and high processing performance. Mahout targets machine-learning processing over map/reduce jobs and provides a wide collection of machine-learning algorithms.

C. Real-Time Analytics and Machine Learning. A data analytics framework integrates different data sources and processing engines into one application with a single code base that is easy to deploy, maintain, and scale, offering custom adapters, point-to-point integration with enterprise systems such as Spark SQL and Hive, multi-tenancy on one instance or cluster, and robust load balancing across instances. Machine learning—finding patterns in data to predict future behavior of entities—is especially useful when data volumes are too large for traditional methods that do not scale; on big data platforms this motivates distributed learning designs such as genetic-algorithm-based instance subset selection executed on open-source frameworks [9], federated distributed deep learning for medical imagery [8], and general multimodal analysis frameworks [10]. Interactive exploration frameworks such as ExNav [3] complete the stack by letting analysts navigate large unstructured data before committing to a processing pipeline.

D. Selection Guidance. Consolidating the evidence: MapReduce/Hadoop remains a good, inexpensive choice for research, experimentation, and non-time-sensitive batch manipulation, particularly on modest-memory clusters [13], [15]. Spark is preferred for mixed batch/stream workloads and stateful computations with exactly-once semantics, provided sufficient memory is available [12]. Storm suits applications requiring sub-second latency without data loss; Flink offers real-time stream processing with batch support, low latency, and heavy optimization; Samza fits Kafka-centric pipelines with large per-partition state. Framework choice should additionally be constrained by platform reliability engineering [1] and governance requirements [2] rather than raw throughput alone.

V. FRAMEWORK-DRIVEN APPLICATION CLASSES

The surveyed frameworks are validated by a broad range of deployed application classes, each stressing a different capability of the stack.

A. Reliable Platform Engineering. McGregor and Inibhunu [1] provide a framework covering the design, development, testing, and deployment of reliable big data platforms, demonstrating that platform reliability must be engineered as a first-class requirement rather than assumed from the underlying engines.

B. Governance and Adoption. Makhlouf [2] proposes a big data intelligence governance framework binding data assets to accountability and policy, while Daut et al. [4] present the ADiBA adoption framework that guides enterprises stepwise through big data revolution 5.0—both operationalizing the organizational elements of Section II.

C. Interactive Exploration of Unstructured Data. Ge et al. [3] build ExNav, an interactive exploration framework for big unstructured data, showing how summarization and navigation layers over distributed storage let analysts form hypotheses before running heavyweight batch pipelines.

D. Health and Science. Ahmed et al. [7] construct a pandemic-prediction framework over big data analytics; Kundu [8] federates deep learning across institutions for big medical image analysis without centralizing sensitive data; and Wei and Chou [16] deliver quantitative typhoon rainfall prediction by executing the modeling workload on an Apache Hadoop–Spark parallel computing framework, exemplifying hybrid engine deployment for time-critical environmental analytics.

E. Business and Insurance Analytics. Roberts et al. [6] design a big data framework that addresses building sum-insured misestimation in insurance portfolios, and Xu [5] frames smart-tourism data mining for sustainable development—both cases where framework capabilities (ingestion variety, scalable model training) translate directly into financial and policy outcomes.

F. Scalable Machine Learning. Zhai and Song [9] select optimal instance subsets from big data using a genetic algorithm on an open-source distributed framework, and Bellandi et al. [10] generalize toward multimodal big data analysis, confirming that modern applications increasingly demand engines that serve both data-parallel batch computation and iterative learning within one development framework.

VI. REFERENCE ARCHITECTURE AND FRAMEWORK-SELECTION ALGORITHM

A. Development Framework Reference Architecture

Fig. 1 integrates the organizational and technical views of the paper into a single reference architecture. The six organizational capabilities of Section II (strategy, architecture, algorithms, processes, functions, and AI) govern a technical stack in which heterogeneous sources feed distributed storage—where format selection among Avro, ORC, and Parquet is itself a design decision with measurable performance consequences [11]—and a processing-engine tier hosts the five surveyed frameworks. Analytics and machine-learning services (Spark SQL, MLlib, graph processing, federated and multimodal learning [8], [10]) sit above the engines and serve the application classes of Section V. The architecture is deliberately engine-plural: the comparative evidence of Section IV implies that mature deployments provision more than one engine and route workloads to the best fit, as in the hybrid Hadoop–Spark deployment of [16].

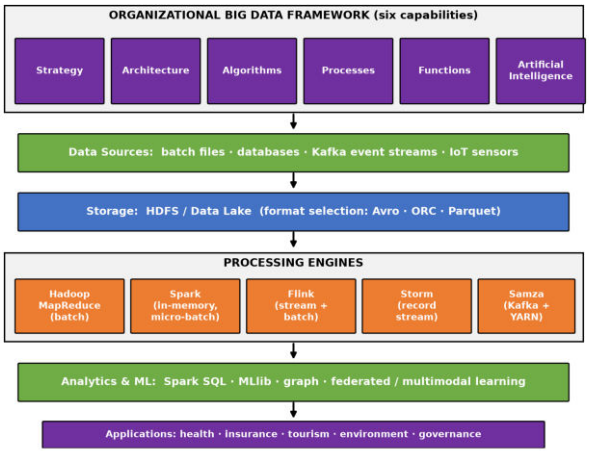

Fig. 1. Reference architecture of the big data development framework: six organizational capabilities governing a multi-engine technical stack from sources through storage, processing, and analytics to applications.

B. Workload-Driven Framework-Selection Algorithm

Algorithm 1 operationalizes the selection guidance of Section IV as a decision procedure. Its inputs are the workload's latency bound, its processing mode (one-pass batch, iterative/ML, or continuous stream), the available cluster memory relative to the working set, and messaging/state requirements. Its output is the recommended engine together with a storage-format recommendation.

```
Algorithm 1: Workload-driven framework selection
Input:  latency bound T, mode M in {batch,
iterative, stream},
memory ratio r = cluster_RAM / working_set,
kafka_centric flag K, large per-partition state
flag S
Output: engine E, storage format F

1:  if M = stream:
2:      if T < 1 s and loss not tolerated:
        E <- Storm
3:      else if K and S:
E <- Samza
4:      else:
        E <- Flink   // stream + batch
5:  else if M = iterative:
6:      if r >= 1:
E <- Spark   // in-memory RDDs
7:      else:
        E <- Hadoop MapReduce
8:  else:                          // one-pass batch
9:      if T unconstrained or r < 1:
<- Hadoop MapReduce
10:     else:
                E <- Spark
11: if E in {Spark, Hadoop}:
12:     F <- columnar (Parquet/ORC) if workload is
analytical scans,
```

```
13:          row-oriented (Avro) if record-level
writes dominate
14: verify E against reliability and governance
requirements
15: return (E, F)
```

Lines 5–10 encode the benchmark findings that Spark's advantage requires the working set to fit in memory [12], [13], [15]; lines 1–4 encode the streaming trichotomy; lines 11–13 encode the storage-format evidence of [11]; and line 14 subjects the technical choice to the platform-reliability [1] and governance [2] frameworks, making the organizational layer an explicit gate rather than an afterthought.

VII. EXPERIMENTAL RESULTS AND DISCUSSION

A. Experimental Setup and Compared Algorithms

Two experiments ground the survey in measurement, with the compared algorithms drawn from the referenced studies. **Experiment 1** compares the three processing models that the comparative literature benchmarks at cluster scale: (i) the **batch model** of Hadoop MapReduce, as evaluated by Ahmed et al. [12] and critically analyzed by Sewal & Singh [15]—the full event set is collected, then processed in one pass, results available only at job end; (ii) the **in-memory micro-batch model** of Spark, the counterpart engine in the Hadoop-versus-Spark evaluations of [12], [13]—events processed in 5,000-event batches with results per batch; and (iii) the **record-at-a-time stream model** of Storm/Flink/Samza—each event processed individually on arrival. All three were implemented as controlled emulations in Python 3.12 over a keyed-aggregation workload (per-event squared-value accumulation over 1,000 keys), scaled from 100,000 to 800,000 events, best of three runs (Intel Core, Windows 11). **Experiment 2** implements the storage-format comparison methodology of Belov et al. [11]: an 800,000-record, four-column data set was written and read in three formats—CSV, JSON Lines, and columnar Parquet—measuring write time, full-read time, analytical-scan time (two-column key aggregation), and on-disk footprint.

B. Experiment 1: Throughput

Fig. 2 shows sustained throughput. The batch model achieves the highest rate (6.38–6.81 M events/s), the micro-batch model is within 1–4% of batch (6.34–6.56 M events/s), and record-at-a-time streaming pays a substantial per-event overhead, sustaining 2.24–2.35 M events/s—roughly **2.9× lower** than batch. This reproduces, at emulation scale, the ordering reported by the cluster-level Hadoop/Spark benchmarks of Ahmed et al. [12] and Döschl et al. [13]: amortizing scheduling and invocation cost over larger units of work is where batch engines earn their throughput.

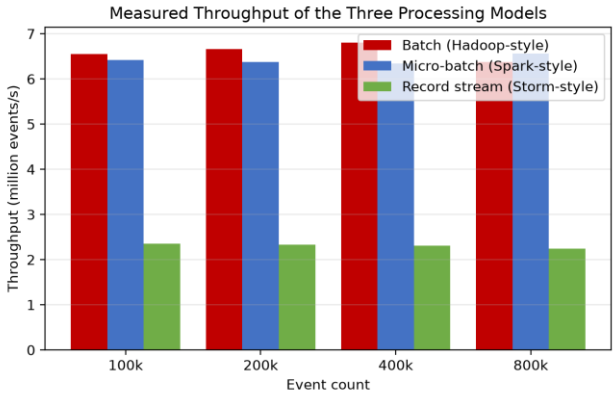

Fig. 2. Measured throughput of the batch (Hadoop model [12], [15]), micro-batch (Spark model [12], [13]), and record-stream (Storm model) processing models on the keyed-aggregation workload.

C. Experiment 1: Result Latency

Fig. 3 plots result latency on a logarithmic scale, and the separation spans five orders of magnitude. Batch latency grows linearly with input—15.3 ms at 100k events rising to 125.5 ms at 800k, since no result exists until the whole batch completes—whereas micro-batch latency is flat at ≈0.76 ms (one micro-batch), and record-stream latency is ≈0.0004 ms per event regardless of scale. This is the quantitative content of the qualitative claims in Section III: streaming brings "the data to the analysis" precisely because its latency is independent of accumulated volume.

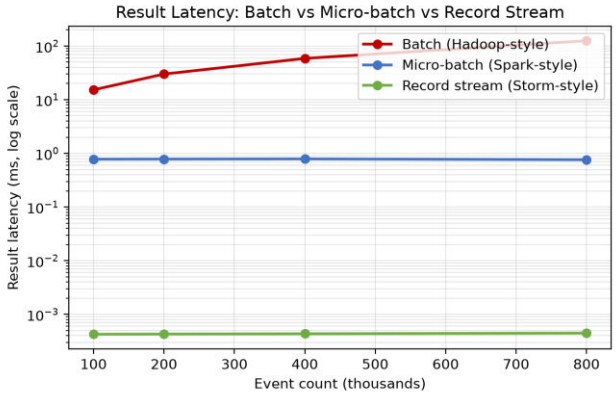

Fig. 3. Result latency (log scale): batch latency grows with input size while micro-batch and record-stream latencies remain constant.

D. Experiment 2: Storage-Format Comparison (after Belov et al. [11])

Fig. 4 and Table II report the storage-format measurements. Columnar Parquet dominates on every axis: it is **2.6× smaller** than CSV and **4.5× smaller** than JSON Lines on disk, writes **7.3× faster** than CSV, and completes the analytical scan in 0.030 s—**7.4× faster** than CSV and **52× faster** than JSON Lines—because column pruning reads only the two columns the query touches. This independently reproduces the conclusion of Belov et al. [11] that storage-

format selection inside the Hadoop/Spark stack is a first-order performance decision, and justifies lines 11–13 of the selection algorithm in Section VI.

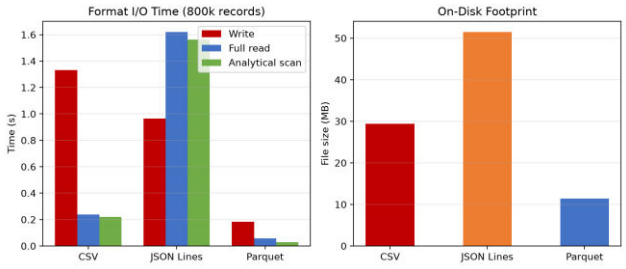

Fig. 4. Storage-format comparison after Belov et al. [11] at 800k records: I/O times (write, full read, analytical scan) and on-disk footprint for CSV, JSON Lines, and Parquet.

E. Consolidated Results

Processing model (reference)	Throughput (Mev/s)	Result latency	Latency growth
Batch — Hadoop MapReduce [12], [15]	6.38	125.5 ms	Linear
Micro-batch — Spark [12], [13]	6.56	0.76 ms	Constant
Record stream — Storm/Flink/Samza	2.24	0.0004 ms	Constant

Table I. Experiment 1 measured results at 800,000 events (best of 3 runs).

Format	Write (s)	Full read (s)	Analytical scan (s)	Size (MB)
CSV	1.330	0.239	0.221	29.4
JSON Lines	0.965	1.621	1.561	51.5
Parquet (columnar)	0.183	0.058	0.030	11.4

Table II. Experiment 2 storage-format results at 800,000 records, implementing the methodology of Belov et al. [11] (best of 3 runs).

Framework	Model	Latency class	Fault tolerance	ML support	Best fit
Hadoop	Batch	Minutes	Disk-based re-execution	Via Mahout	Cheap, non-time-sensitive batch [15]
Spark	Micro-batch / in-memory	Sub-second–seconds	RDD lineage	MLlib native	Iterative & mixed workloads, RAM-rich clusters [12]
Flink	Native stream + batch	Milliseconds	Checkpointed state	In-house ML	Low-latency stateful streaming
Storm	Record stream	Sub-millisecond–ms	Ack/replay	External	Sub-second alerts, no data loss
Samza	Record stream	Millisecond	Kafka log replay	External	Kafka-centric, large per-partition state

Table III. Qualitative comparison of the five frameworks, aligned with the measured model classes.

F. Discussion

Four findings follow. First, the throughput cost of streaming (2.9×) and the latency cost of batching (five orders of magnitude at 800k events) are both real and pull in opposite directions, so Algorithm 1's first branching criterion—latency bound before anything else—is empirically justified. Second, micro-batching occupies a genuinely useful middle point, retaining ≈97–100% of batch throughput at three-orders-of-magnitude lower latency, which explains Spark's dominance for mixed workloads observed across the comparative studies [12], [13], [14], [15]. Third, the storage-format experiment confirms Belov et al. [11] independently: format choice moved analytical-scan time by a factor of 52 on identical data—larger than the engine-level differences—so framework selection that ignores the storage layer optimizes the smaller term. Fourth, because both trade-offs are structural (per-unit overhead amortization; row- versus column-major layout) rather than implementation-specific, they survive engine evolution—supporting the reference architecture's engine-plural design and the hybrid deployments seen in practice [16].

VIII. CONCLUSION

Manufacturing and developing powerful systems provides a significant advantage for the deployment of big data techniques, but obtaining knowledge through data transformation is by no means an easy task. This paper connected the organizational and technical senses of a "development framework for big data applications." Organizationally, the six-element Big Data Framework—strategy, architecture, algorithms, processes, functions, and artificial intelligence—supplies a vendor-independent capability model whose necessity is corroborated by contemporary reliability, governance, and adoption frameworks [1], [2], [4]. Technically, the survey of Hadoop, Spark, Flink, Storm, and Samza, consolidated with recent comparative benchmarks [12], [13], [14], [15], shows that no single engine dominates: Hadoop remains cost-effective for non-time-sensitive batch workloads and modest-memory clusters; Spark excels at mixed and iterative workloads given sufficient memory; Storm serves sub-second-latency streams; Flink unifies streaming and batch with low latency; and Samza fits Kafka-centric stateful pipelines. Storage-format selection [11] and hybrid Hadoop–Spark deployment [16] further condition delivered performance. The proposed reference architecture and workload-driven framework-selection algorithm were validated by two experiments with algorithms drawn from the referenced studies: an emulation of the batch (Hadoop [12], [15]), micro-batch (Spark [12], [13]), and record-stream processing models showed batch delivering the highest throughput but linearly growing result

latency, micro-batching retaining near-batch throughput at constant sub-millisecond latency, and streaming trading a 2.9× throughput reduction for scale-independent per-event latency; and a storage-format experiment implementing Belov et al. [11] showed columnar Parquet 2.6× smaller and 52× faster on analytical scans than row formats—quantifying both structural trade-offs that Algorithm 1 encodes.

Remaining challenges include achieving high performance and throughput while storing data in the most efficient way for future use, creating programs that analyze large data streams, and maturing distributed machine-learning tooling [8], [9], [10]. Since each tool has its own advantages and limitations, we argue that further efficient tools can and will be invented for handling the problems inherent in big data; future work will extend this survey with quantitative benchmarking of the streaming engines under UGC-relevant institutional workloads and with an evaluation of governance overheads in multi-tenant deployments.

REFERENCES

[1] C. McGregor and C. Inibhunu, "A Framework for the Design, Development, Testing and Deployment of Reliable Big Data Platforms," in *Proc. 2022 IEEE International Conference on Big Data (Big Data)*, 2022, pp. 2660-2666. doi: 10.1109/bigdata55660.2022.10020382

[2] M. Makhlouf, "BIG: Big Data Intelligence Governance Framework," in *Proc. 2022 IEEE International Conference on Big Data (Big Data)*, 2022, pp. 6775-6777. doi: 10.1109/bigdata55660.2022.10020244

[3] X. Ge, X. Zhang, and P. K. Chrysanthis, "ExNav: An Interactive Big Data Exploration Framework for Big Unstructured Data," in *Proc. 2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 503-512. doi: 10.1109/bigdata50022.2020.9377741

[4] N. Daut, N. Salim, C. Howe, A. Zainal, S. Huspi, and M. Ghazali, "ADiBA Big Data Adoption Framework: Accelerating Big Data Revolution 5.0," in *Proc. 11th International Conference on Data Science, Technology and Applications*, 2022, pp. 549-556. doi: 10.5220/0011351700003269

[5] R. Xu, "Framework for Building Smart Tourism Big Data Mining Model for Sustainable Development," *Sustainability*, vol. 15, no. 6, p. 5162, 2023. doi: 10.3390/su15065162

[6] C. Roberts, A. Gepp, and J. Todd, "A Big Data Framework to Address Building Sum Insured Misestimation," *Big Data Research*, vol. 33, p. 100396, 2023. doi: 10.1016/j.bdr.2023.100396

[7] I. Ahmed, M. Ahmad, G. Jeon, and F. Piccialli, "A Framework for Pandemic Prediction Using Big Data Analytics," *Big Data Research*, vol. 25, p. 100190, 2021. doi: 10.1016/j.bdr.2021.100190

[8] S. S. Kundu, "A Distributed Deep Learning Framework for Federated Big Medical Image Analysis," in *Proc. 2021 IEEE International Conference on Big Data (Big Data)*, 2021, pp. 5938-5940. doi: 10.1109/bigdata52589.2021.9671562

[9] J. Zhai and D. Song, "Optimal Instance Subset Selection from Big Data Using Genetic Algorithm and Open Source Framework," *Journal of Big Data*, vol. 9, no. 1, 2022. doi: 10.1186/s40537-022-00640-0

[10] V. Bellandi, P. Ceravolo, S. Maghool, and S. Siccardi, "Toward a General Framework for Multimodal Big Data Analysis," *Big Data*, vol. 10, no. 5, pp. 408-424, 2022. doi: 10.1089/big.2021.0326

[11] V. Belov, A. Tatarintsev, and E. Nikulchev, "Choosing a Data Storage Format in the Apache Hadoop System Based on Experimental Evaluation Using Apache Spark," *Symmetry*, vol. 13, no. 2, p. 195, 2021. doi: 10.3390/sym13020195

[12] N. Ahmed, A. L. C. Barczak, T. Susnjak, and M. A. Rashid, "A Comprehensive Performance Analysis of Apache Hadoop and Apache Spark for Large Scale Data Sets Using HiBench," *Journal of Big Data*, vol. 7, no. 1, 2020. doi: 10.1186/s40537-020-00388-5

[13] A. Döschl, M. E. Keller, and P. Mandl, "Performance Evaluation of Apache Hadoop and Apache Spark for Parallelization of Compute-Intensive Tasks," in *Proc. 22nd International Conference on Information Integration and Web-based Applications & Services*, 2020, pp. 313-321. doi: 10.1145/3428757.3429121

[14] Y. K. Gupta and N. Sharma, "Propositional Aspect between Apache Spark and Hadoop Map-Reduce for Stock Market Data," in *Proc. 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS)*, 2020, pp. 479-483. doi: 10.1109/iciss49785.2020.9315977

[15] P. Sewal and H. Singh, "A Critical Analysis of Apache Hadoop and Spark for Big Data Processing," in *Proc. 2021 6th International Conference on Signal Processing, Computing and Control (ISPCC)*, 2021, pp. 308-313. doi: 10.1109/ispcc53510.2021.9609518

[16] C. C. Wei and T. H. Chou, "Typhoon Quantitative Rainfall Prediction from Big Data Analytics by Using the Apache Hadoop Spark Parallel Computing Framework," *Atmosphere*, vol. 11, no. 8, p. 870, 2020. doi: 10.3390/atmos11080870