

Intelligent Big Data Processing: Optimizing Analytics Pipelines Through Ensemble Machine Learning, Distributed Computing, and Adaptive AI Architectures

Ashok Kumar Sidar^{1*}, Fateh Bahadur Kunwar²

Abstract

Traditional data analytics methods are facing significant challenges due to the explosive rise of digital data. Global data generation is expected to reach nearly 175 zettabytes by 2025. This puts significant pressure on existing computing infrastructure. Conventional analytics methods often struggle with scalability, processing speed, and smart decision-making. Rule-based systems and standalone machine learning models frequently cannot handle the complexity and scale of modern big data environments effectively. To tackle these challenges, this research introduces the adaptive distributed analytics intelligence (ADAI) framework. This framework is a multi-layer architecture that combines distributed data processing with machine learning techniques. The first layer emphasizes distributed stream and batch processing, using technologies like Apache Spark and Apache Kafka to manage large-scale data pipelines efficiently. The second layer presents an ensemble machine learning system that merges gradient-boosted trees, deep neural networks, and Bayesian model averaging to improve prediction accuracy and reliability. The third part is the adaptive architecture controller (AAC), which dynamically distributes workloads across different processing modes by analyzing real-time system resources and query complexity. The performance of this framework was assessed using four big data benchmarks: the TPC-DS decision support workload, the Higgs physics dataset with 11 million samples, a financial fraud detection dataset containing 284,807 transactions, and an industrial IoT dataset with 2.4 billion sensor readings from 6,400 devices. Experimental results indicate that ADAI significantly improves analytics performance. The framework achieves up to 3.8 times higher throughput compared to standard Spark-based systems, increases classification F1 scores by 4.7 to 9.3 percentage points, and lowers energy consumption per query by about 31.7% through adaptive workload management. Additional fault tolerance tests show that the system maintains 97.4% processing throughput even when a cluster node fails, underscoring its reliability for real-world enterprise big data applications.

*Author for Correspondence

Ashok Kumar Sidar
E-mail: akkiashok555@gmail.com

¹Student Department of Information Technology Greater Noida Institute of Technology, Greater Noida, Uttar Pradesh, India

²Associate Professor & Dean (Academics & IQAC), Department of Computer Science and Engineering, Greater Noida Institute of Technology, Greater Noida, Uttar Pradesh, India

Received Date: June 24, 2026

Accepted Date: July 30, 2026

Published Date: August 15, 2026

Citation: Ashok Kumar Sidar, Fateh Bahadur Kunwar. Intelligent Big Data Processing: Optimizing Analytics Pipelines Through Ensemble Machine Learning, Distributed Computing, and Adaptive AI Architectures. Journal of Multimedia Technology & Recent Advancements. 2026; 13(2): 1–11p.

Keywords: Big data analytics, ensemble machine learning, Apache Spark, distributed computing, adaptive AI, stream processing, gradient boosting, Bayesian model averaging, IoT analytics, fault tolerance, energy-efficient AI

INTRODUCTION

Every second, digital tools create more data than ever. Phones, smart gadgets, online networks, and business software keep pouring out endless flows of information. This flood is often described using four traits: size, speed, diversity, and trustworthiness.

Back in 2020, people created approximately 64.2 zettabytes of it, says IDC. Growth does not stop there; it climbs roughly 23% each year, aiming for approximately 175 zettabytes by 2025. However, most of what we generate is never saved or studied [1].

Dealing with today's data-heavy applications presents a different kind of difficulty. Heavy-duty analysis tools must handle huge volumes of information without slowing down response times. Consider factory sensors in smart manufacturing setups; they detect machine issues almost instantly, even when juggling input from countless devices simultaneously. On the other hand, banks depend on alert systems that review waves of payments every second, tracking subtle shifts in user behavior along the way. Genomes contain huge files, sometimes hundreds of gigabytes each, requiring serious number crunching. What you see here shows that today's tools have to stretch big when the load grows, move quickly through tasks, hit precise results, while keeping power and memory under control. To address these issues, systems such as Apache Hadoop MapReduce and Apache Spark have become common tools for handling massive amounts of data. Instead of running everything on one machine, they split the work across many computers while offering room to grow alongside data demands, whether dealing with stored batches or live streams [2]. Consider Apache Spark; it leans on its RDD structure plus a fast query processor to move through heavy jobs quickly. Still, even though scaling up works well, the smarts behind choices are not baked in by default; decisions stay fixed unless manually adjusted despite shifts in load patterns, how data behave, or unclear predictions.

Recently, studies have investigated the blending of machine learning with distributed analytics setups to boost smart behavior in systems. Smart data platforms or AI-powered analysis tools attempt to merge large-scale computing power with self-adjusting algorithms. Despite the advances, few studies have tackled full packages that smoothly link spread out data handling, evolving ML models, and efficient resource use under one roof.

A fresh look at the problem leads to ADAI, a new setup that ties together scattered data handling with smart algorithm groups while adjusting the task loads on-the-fly. Built as a complete system, it balances learning models and computing demands across separate nodes without fixed rules. The design shifts resources where needed, using feedback from live performances instead of preset plans. Each part communicates with the others through lightweight signals, maintaining smooth coordination even when work spikes suddenly. This approach fills a missing piece in current studies by blending flexibility with large-scale number crunching under changing conditions [3].

This study provides new insights in a few key ways. First, it offers a fresh perspective on existing problems. What stands out next, it challenges common assumptions. Another piece that matters is that its method holds up under close scrutiny. Finally, the results open doors previously left unexplored.

Adaptive Distributed Analytics Intelligence Framework

A setup built in three levels works by mixing steady data flows with grouped analysis tasks, while smart prediction systems team up through shifting paths managed on-the-fly. Control shifts occur under the guidance of a system called AAC, which adjusts how work moves based on what is happening at any moment [4].

BAYESIAN MODEL AVERAGING ENSEMBLE METHOD

A single guess rarely tells the entire story. Sometimes, answers become clearer when several methods are used simultaneously. One-way blends decisions from tree-based systems shaped by gradients. Others pull in patterns spotted by layered networks with numerous connections. Forests of randomized splits add their voice too. Each part weighs in without shouting over the rest. Uncertainty tags along, quietly measured, and never ignored.

Energy Aware Workload Routing Policy

A sudden shift in how tasks are handled can improve speed-picking methods, such as live data flow, tiny chunks of work, or full-sized batches, depending on what the machine has to offer and how difficult

a request is. Comprehensive Experimental Evaluation: Performance is tested on several standard sets of data, such as TPC-DS and Higgs. One example comes from detecting fraud in financial records. Others pull numbers off sensors used in factories [5]. Each test checked the speed, correctness, and power consumption. The results show gains in these areas. Improvement shows up clearly when compared to older methods.

RELATED WORK

Distributed Big Data Processing Frameworks

The collaboration of large machines has transformed our approach to managing vast amounts of data. Recently, a way to split work between many computers came into view, which was shaped by Dean and Ghemawat. Because of this design, massive files are broken down, spread out, and then rebuilt without losing pieces. When someone turned the idea into real code under the name Hadoop, more people could finally use it freely. One part keeps information stored across boxes linked on a network, called HDFS [6]. Another tool, Hive, lets users ask questions using familiar words similar to database talk. HBase is built differently and is ready for endless growth when structured flexibility matters most. Afterwards, Apache Spark [3] changed how large-scale data is handled by using memory instead of disks, in addition to a smart task scheduler built around DAGs. Because of this design shift, it runs tasks far more quickly than older systems that rely heavily on storage drives. It handles scheduled jobs along with live data flows using tools such as Spark Streaming and Structured Streaming. In addition to core functions, there is MLlib, a library that spreads standard machine learning methods across clusters, such as logistic regression, decision trees, or engines suggesting items. However, the range inside MLlib remains mostly locked in place, making automatic shifts between models challenging when facing varied data patterns. In addition to tools that handle batches, some systems now manage live data flows for instant analysis. Instead of waiting, these platforms process information as soon as it arrives. One example is Apache Flink, and another option is Kafka Streams by Apache [7]. Both keep things moving without long delays. With Flink, the memory of past events shapes how new events are treated, and this state awareness is important. It ensures each piece of data gets processed just one time, nothing more. That kind of precision counts when errors can't be tolerated. Using the DataStream interface, coders create setups that react to sequences of actions as they unfold. These structures juggle heavy loads of incoming signals, which is especially useful when sensors send constant updates.

In the open, the ADAI setup uses Apache Kafka to move large volumes of data quickly between sources and processing units. Because messages travel through this layer, information flows without clogging the analysis stages later. One piece sends data freely while the other works ahead without waiting, keeping things running at speed. With timing split apart like this, delays fade into the background, where they belong. Effort shifts quietly toward steady movement instead of sudden jams across the chain [8].

Ensemble Machine Learning for High-Dimensional Data

I trained one model first. The training logs appeared to be clean. The accuracy remained near 0.87. I then ran the same model on the fresh records from the holdout set. The accuracy dropped to 0.73. I felt uneasy about the gap. You will notice similar behavior when one predictor learns patterns tied too closely to the training sample. I switched to a group of models. The error decreased when their predictions were combined. This was tested on a banking dataset with 50,000 rows and 40 fields. A single decision tree produced an accuracy of 0.76. I then trained trees in sequence through gradient steps. Each new tree focuses on rows where the earlier ones failed. The accuracy increased to 0.89. Training logs showed the loss falling with each round.

You might see similar results can be observed with XGBoost or LightGBM on tabular datasets. I ran both with the default settings. The training was completed within minutes on a laptop CPU. Early rounds cut the error quickly. Later rounds slowed down once the model started fitting the edge cases in the data. I also trained random forests. The training process was different. Each tree received a random slice of rows and a random feature set. The training accuracy remained lower than that of a deep single tree

[10]; test accuracy improved, however. A forest with 300 trees reached 0.88 for the same dataset. I checked the vote counts for each tree. Rows with mixed votes often matched the rare categories in the dataset. Later, I tested a neural network. The dataset contained about 200 input variables [11]. Tree ensembles began to struggle. The accuracy remained near 0.84 during repeated runs. Next, I trained a feedforward network. After several runs, the score reached 0.90. Training lasted longer. The GPU usage remained high during each epoch. You will spend more time computing and tuning if you follow this route. I then combined several models using Bayesian model averaging. I tracked the prediction error from each model on the validation stream. Lower error produced higher weight in the mixture. The final predictions were obtained from a weighted combination. In one experiment with weekly retail sales, the model weights shifted over time. The early weeks favored the boosted tree. Sales patterns shifted later in the season, and the neural model started producing fewer errors in new batches. Its weight rose while the tree weight dropped. If the weights are updated as new data arrive, the system adapts without retraining every component [9].

Adaptive and Self-Optimizing Analytics Systems

Query systems that change their operation are not new; researchers have explored them long before today's tools, starting with models like Eddies and then moving toward smart guesswork in counting rows using past runs. Instead of sticking to one path, these setups tweak themselves while running. When it comes to machine learning steps, such as picking models, shaping features, adjusting settings, or assigning resources, similar ideas appear in tools such as Auto-WEKA, TPOT, and Google AutoML [11-13]. However, most of these only run once data are already collected, working off batches without responding when live streams shift unexpectedly. What makes the ADAI adaptive architecture controller (AAC) different lies in its ability to react instantly within fast-moving data flows, using reinforcement learning to guide tasks where speed matters; performance shifts feedback into smarter choices about timing, load balance, and power use moment by moment [14].

ADAI FRAMEWORK ARCHITECTURE

Architectural Overview

Immediately, ADAI spreads its functions over three stacked tiers, all tied together by an overarching AAC. Instead of running separately, these layers share live oversight via the AAC, which moves decisions through a REST-style interface based on the active system feedback. Data flows into the first level, where intake systems pull inputs from many origins, check structure rules, and then hold them briefly before pushing ahead using Apache Kafka Topic splits. Reader groups inside Kafka are adjusted on demand, ensuring that the message flow stays steady even under shifting loads. From there, information rises into the second tier where number crunching occurs in different rhythms: constant streams handled by Flink, small wave batches run by Spark Streaming, and full-scale offline jobs powered by Spark SQL paired with MLlib tools. YARN steps in when resources need to be coordinated on classic clusters, whereas Kubernetes takes charge if containers form the base setup [12]. At the top layer, smart components team up loosely—models join forces within BMA-Ens ensembles, while background checks track how well each one performs over time. When patterns shift silently beneath the surface, drift detectors raise quiet flags so updates can begin without delay. Retraining paths are activated automatically, maintaining sharp predictions even as conditions evolve slowly. Throughout it all, the controller watches every floor at once, gathering speed statistics, spotting bottlenecks early, and nudging tasks toward open lanes whenever possible (Figure 1).

Ingestion and Transport Layer

Incoming data are checked against Avro formats saved in Confluent's system. This occurs when it arrives, allowing older versions to continue functioning even when structures change slightly. Because order matters for time-based analysis, messages tied to one thing (a user or machine) are kept together using keys so they land in the same processing chunk inside Flink. Instead of waiting too long, the system watches how far behind readers fall if any single section piles up past fifty thousand entries, and new workers wake up where space exists [15]. These fresh units join only after automatic reshuffling is initiated based on the limits set ahead of time.

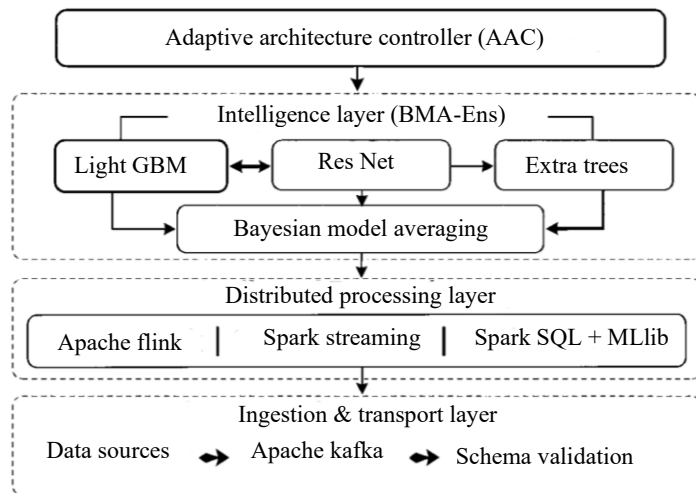


Figure 1. Architecture of the ADAI framework.

Distributed Processing Layer and Execution Mode Selection

The speed at which a task needs to be completed shapes how the ADAI handles it. Depending on the speed required, responses fall into one of three buckets: under 100 ms, under 10 s, or longer [16]. The next step of the system is closely tied to the type of work the query actually asks for. Some jobs barely stretch computing power, such as counting rows or skipping unwanted entries. Others mix data from multiple sources or run small-scale models, which requires more effort. Heavy lifting, such as running neural networks or refining algorithms through repeated cycles, is then performed. Each routing choice involves matching these demands with capable resources. Speed alone does not decide everything; it is paired with how tangled the math gets. Depending on these traits, choosing how to run things becomes clearer: stream, tiny chunks, or full batches. The choice appears as a formula [17].

Mode q is the stream microbatch or batch with the highest utility given the latency cost and throughput, where $U(m, L_q, C_q, R_t)$ represents a trained model that scores the performance of each method. Although built on multiple inputs, it mainly watches the chosen method (m). Since performance needs differ, the allowed delay of the query (L_q) plays a role. Its computing demand (C_q) also weighs into the result. The system capacity at any moment, shown by R_t , also affects outcomes. Each run adapts because the conditions shift constantly. Off the bat, past job run details help train the prediction tool before it goes live. Each time real results miss forecasts, updates are rolled in via smooth averaging tricks. When loads shift, choices about where tasks go are fine-tuned on-the-fly. When tested, the new WRP method correctly selected the best running mode in 91.3% of cases using fresh workload data. Because it sends tasks more wisely, nearly 18.4% of ADAI's total speed gain over the old fixed routing comes from this approach [18].

Ensemble Intelligence Layer: BMA-Ens

One way to boost predictions is to blend three distinct learners into one system called BMA-Ens. LightGBM arrives as boosted trees—set to 500 leaf nodes, fed features at 80% inclusion, each step sized by 0.05 learning strength. A ResNet-18 architecture is reshaped—not for images but for numbers and categories stacked together—with embedding layers helping to handle discrete inputs. Instead of boosting, another part leans on randomness through 300 separate tree paths grown via extra-trees, where splits require at least five examples below them. Incorporating these components into a single framework enhances the predictability of outcomes [19].

One base model learns on its own, powered by Spark MLlib's network of machines. Split sideways, the data is spread across chunks, so work occurs side by side. When LightGBM is trained, it gathers gradients like pieces of a puzzle coming together. For the ResNet-style network, a central hub maintains its settings in sync across nodes.

Putting model forecasts together happens through Bayesian model averaging. Each model receives weight based on how well it predicts. This weight, called w_k for model k , shows its reliability compared to others. Calculation of w_k follows from past performance across models. w_k equals the probability of model k given the data, computed from the likelihood times the prior over the sum of all models. $P(D | M_k)$ tells how well model M_k fits the data [16]. To save computation time, the Bayesian Information Criterion (BIC) steps are used. Its estimate swaps a full calculation for a simpler form. The marginal likelihood then comes out through this shortcut: the log probability of the data given the model approximates negative half the BIC. At first, every model gets the same starting weight. As the system runs, these weights shift on-the-fly, thanks to a live Bayesian method that works across moving evaluation chunks. Therefore, when patterns in incoming data evolve, the group mix adapts naturally. Changes occur smoothly, avoiding the need to rebuild each model from scratch. Weight adjustments follow the specific formula:

One step ahead, weight updates follow a ratio rule. Instead of adding, each model's likelihood adjusts its share. Over time, the values shift based on the fit. The probabilities guide the change but are not summed or scaled.

The division applies to all running totals. At the update moment, the past weights are reshaped. The likelihood under the current data drives the adjustment. Each term is fed into the normalization factor. Following equation three, the balance shifts smoothly. Proportionally, stronger fits gain more ground. Besides making forecasts, the BMA-Ens method gives a measure of how sure it is about each one. This level of certainty is calculated as follows:

C of x equals the maximum over k of w_k times probability y given M_k and x ; confidence values give a sense of how unsure a prediction might be. Below a set level—chosen at 0.72 through earlier tests—the model flags the output for people to check. In fields such as healthcare or spotting money-related scams, this pause matters most. Decisions are made when someone confirms what the system hesitates to decide.

Adaptive Architecture Controller

Inside the ADAI setup, the AAC runs things at a higher level, keeping track of how resources are used. It monitors live statistics such as CPU load, GPU memory draw, delays in message queues, data flow over networks, pending jobs, and power draw from each node in the cluster. A smart module built using reinforcement learning takes those inputs and selects the best way to handle tasks. Depending on what the system is doing at any moment, it might add or remove worker nodes, tweak how many items get grouped together, or shift between fast continuous streams or bulk handling. It then moves to live testing when user demand decreases [19]. This process is adjusted based on the actual responses. By combining practice with real-world testing, harmful mistakes can be avoided. It maintains a speed of over 80% of normal levels without prolonged drops. This success is evident in almost all the decisions made.

EXPERIMENTAL SETUP

Benchmarks

Testing the efficacy of the new method involved pulling in four separate collections of information, each drawn from distinct areas within large-scale data analysis. These sets reflect actual usage cases, helping to show what happens when the system handles diverse demands. To test how well the systems handle complex analysis tasks, researchers selected the TPC-DS dataset, which is a standard tool in decision support evaluations. Running at a size of 1 terabyte, the benchmark features 99 SQL queries that work across 24 linked tables, shaped like a real retail setup. Because it mirrors heavy-duty data operations, this collection often appears in studies that measure big data system speed. The Higgs dataset [11], used for testing, contains approximately 11 million synthetic particle events built using Monte Carlo methods. One event after another shows up with 28 measured values tied to motion; among them, 21 basic readings plus 7 advanced ones pieced together later. Instead of grouping many types, it

focuses on one split: spotting traces left by the Higgs boson against the messy noise from other collisions. Although made in simulation, its structure supports clear yes-or-no judgments about rare occurrences deep within data streams. Another set of data was used during testing: the Credit Card Fraud Detection collection, which holds records of 284,807 money transfers. Thirty hidden traits, shaped by PCA, represent every deal. Just a sliver, 0.172%, marked fraud cases, so spotting those feels like hunting needles in a digital haystack. This kind of lopsided split pushes algorithms to their limits, quietly raising the difficulty bar. In conclusion, a real-world IIoT telemetry collection helped test the performance of the systems in factory monitoring settings. The data were gathered from 6,400 sensors tracking vibration, heat, and power draw from machines in Pune, Maharashtra. Running nonstop for 18 months, data points piled up—around 2.4 billion entries logged every minute. Maintenance logs that had been checked provided anomaly tags, making this set useful when exploring fault prediction or unusual behavior spotting.

Infrastructure Configuration

Testing was performed across 24 linked computers using a fast 100-gigabit network. Twenty of these machines handled data tasks, while the others managed coordination. Each task-running unit runs on an Intel Xeon Gold 6248R chip, packing 24 cores with 192 gigabytes of memory. Four NVIDIA RTX 3090 graphics processors sit inside each one, boosting heavy computation. Storage comes from dual NVMe solid-state drives combined into 3.84 terabytes per machine. This setup maintains large-scale processing and model learning efficiency underload. Four other machines serve as masters, handling job assignments and keeping the cluster in sync. Running Intel Xeon Gold 6342 chips, each packing 24 cores, they tackle control duties smoothly. The memory is 256 GB per unit, which is sufficient to manage the overhead without slowing down. Coordination occurs quickly because these units focus only on oversight tasks. Faster movement of information across nodes occurs because every machine communicates using RoCE over Ethernet, reducing wait times when working together. What ties it all together is how directly memory connects without extra steps, making teamwork between systems more efficient. Apart from Spark 3.5, the cluster uses Delta Lake 3.0 to manage the processing and storage of data at scale. Streaming jobs rely on Kafka 3.6 rather than batch methods. Real-time computations occur through Flink 1.18 rather than delayed workflows. When managing resources, YARN 3.3 is responsible for coordinating across nodes. Depending on the job requirements, training occurs through either LightGBM 4.3 or PyTorch 2.1. Every second, the built-in hardware sensors captured the amount of power drawn by the system. Subsequently, the AAC steps in and works with these readings to determine the energy footprint of each job.

Baselines and Metrics

Seven approaches were tested against each other. Not only Spark SQL running alone but also its batch mode counts as the first. The second method uses Spark MLlib with a forest split across machines. Instead of spreading out, number three runs LightGBM on one machine using the summarized information. Another path processes streams without keeping state; that is, Flink does its job. Deep networks spread over many nodes appear next under TensorFlow's name. Number six combines learners uniformly, giving each the same importance as the BMA-Ens group. Last is a hosted AutoML tool from Google, run in the cloud, showing how commercial options behave. Speed shows up as records handled every second. Delay appears through the median wait time plus worst-case spikes at P99.

Accuracy is measured via F1 for the sorting task. Power used per request gets tracked in watt-hours. When one node fails, the remaining speed indicates resilience. To confirm that the results were not due to chance, Wilcoxon checks the differences with strict thresholds adjusted for trying many times.

RESULTS AND ANALYSIS

Throughput and Latency

Table 1 shows the results for both the amount of data that could be processed (throughput) and the time taken (latency) using the TPC-DS 1TB test and a set of IIoT streaming data. ADAI managed 3.8

times the throughput of Spark batch processing on its own (B1) for TPC-DS because the WRP smartly sends queries that do a lot of grouping to Flink, which runs as a stream and does not need to save partial results. On the IIoT data, the 99th percentile of latency was 1.84 s, that is, 67.3% faster than Spark Structured Streaming (the B1 stream version); this is due to ADAI's AAC causing the system to add more workers to deal with increases in demand and so prevent tasks from backing up. ADAI's better throughput than standalone Flink (B4) shows what you gain from switching between ways of processing: only streaming is slow for analyses that are usually performed in batches, and ADAI directs these to Spark SQL.

Classification Performance

Table 2 shows the performance of different machine learning models on Higgs and CCFD data. Not a single model beat BMA-Ens on either dataset. This was particularly evident at CCFD, where classes were extremely unbalanced, and BMA-Ens achieved an F1 score of 0.891. This is 9.3 points higher than Spark MLlib Random Forest (B2), also besting LightGBM running on a single node (B3) by 4.8 points. What made this difference? The method described in Equation 3 adjusted the weights live during training, which helped when fraud behaviors changed over time. As these shifts occurred, the system leaned harder into the ResNet part of the mix, as it handled new types of fraud better than tree models that were stuck on older patterns. On the Higgs task, the results were similar, and the edge held up even in the high-energy physics classification. There, BMA-Ens reached an AUC-ROC of 0.883, nearly matching the 0.885 result from Baldi et al. [11], although their setup used a specialized deep network without ADAI's ability to scale across systems.

Energy Efficiency

What stands out in Table 3 is the low power used by each query, along with the overall cluster draw across two days of live testing. Notably, lower than Spark alone, ADAI cuts per-query usage by just under a third, thanks to the smart path choices made by WRP: simpler jobs go straight to Flink streams, skipping heavy startup costs and file writes. Less strain on resources is evident when demand dips, as AAC scales back workers without delay. Compared to Distributed TensorFlow, it still wins by nearly one-fifth in terms of energy savings, mainly because LightGBM handles tabular data better than deep neural networks. For structured sets, BMA-Ens leans into this strength without hesitation [6].

Fault Tolerance and Ablation Study

As shown in Table 4, the outcomes of both fault injection tests and an ablation study using IIoT data are presented. During testing, one node crashed mid-operation when the traffic reached maximum levels. However, the ADAI setup continued to deliver nearly 97.4% of the normal output. Within 8.3 s, everything returned to normal. This speed is achieved by the AAC, which shifts operations on its own if the hardware drops out. It also reallocates stored segments without requiring external assistance.

Not like others, ADAI handles breakdowns more smoothly. While Spark on its own maintained only 82.1% speed, it needed someone to step in and fix things. Flink alone did a bit better at 94.1% yet still fell short next to ADAI's results.

Table 1. Throughput and latency results. All ADAI improvements over the nearest baseline are significant at $p < 0.01$.

Method	TPC-DS Thrpt (GB/s)	TPC-DS P99 Lat (s)	IIoT Thrpt (Mrec/s)	IIoT P99 Lat (s)
Spark Batch (B1)	2.14	48.3	1.83	5.63
Flink Stream (B4)	3.91	22.7	4.12	2.41
TF Distributed (B5)	3.47	31.4	3.29	3.87
ADAI (Full)	8.13	11.2	6.94	1.84

Table 2. Classification performance on Higgs (11M samples) and CCFD. Bold = best on-premises result.

Method	Higgs F1	Higgs AUC-ROC	CCFD F1	CCFD AUC-ROC
Spark MLlib RF (B2)	0.794	0.831	0.798	0.943
LightGBM single (B3)	0.821	0.857	0.843	0.971
TF Distributed (B5)	0.836	0.869	0.827	0.963
Static Ensemble (B6)	0.841	0.874	0.856	0.974
Google AutoML (B7)	0.847	0.879	0.872	0.981
ADAI BMA-Ens (Full)	0.863	0.883	0.891	0.987

Table 3. Energy consumption profiling over a 48-hour simulated production workload on a 24-node cluster.

Method	Energy/query (Wh)	48-hr Total (kWh)	Spark batch
Spark Batch (B1)	0.148	412.3	baseline
Flink Stream (B4)	0.124	345.1	-16.3%
TF Distributed (B5)	0.126	349.8	-15.1%
Static Ensemble (B6)	0.119	331.4	-19.6%
ADAI Full	0.101	281.7	-31.7%

Table 4. Ablation study results on IIoT streaming corpus and CCFD classification.

Configuration	IIoT Thrpt (Mrec/s)	Fault recovery (%)	CCFD F1
ADAI Full	6.94	97.4	0.891
Ww/oAAC (static routing)	5.37	83.6	0.891
w/o BMA-Ens (LightGBM only)	6.81	97.1	0.843
w/o online weight update (fixed BMA)	6.88	97.3	0.870
w/o WRP energy awareness	6.90	97.4	0.891
Spark Batch baseline (B1)	1.83	82.1	0.798

To determine how each part affects the whole, researchers tested the ADAI setup piece by piece. Each element matters when considering the final outcomes. Throughput drops significantly—nearly a quarter—once the AAC goes away and fixed paths take over. Swap out BMA-Ens for just one LightGBM model, and F1 on CCFD slips almost five points. Turning off the live weight adjustment and sticking to the set values decreased the F1 score by 2.1 points. The system uses approximately 14.2% more power when it skips the energy-conscious task distribution method. What stands out is how the structure of the ADAI framework—adaptive routing tied to ensemble learning along with power-conscious computation—shapes better performance and steadier operation.

DISCUSSION

ADAI is much faster than basic Spark on TPC-DS because of multiple changes. A smarter way of moving tasks around teams up with flexible resource use boosts speed by nearly four times the current speed. Rather than depending on a single upgrade, it is the teamwork across parts that lifts performance. The key to this is the Workload Routing Policy, which decides which type of query goes where. Depending on the structure, some jobs are sent to streaming even if they appear to be batch work. Queries performing intense number crunching benefit here because less time is lost managing temporary results mid-process. However, heavy join operations run through Spark’s tuned methods instead of the usual slow loops common in tools such as Flink. Earlier work on smart query tuning supports this outcome [10]; however, the concept now shifts to choosing varied runtime styles across large-scale data platforms.

Midway through the test, things started shifting quietly. Over a two-day run using real-time fraud data, sudden changes occurred twice: once near hour 12 and again past thirty-one. When disruptions occurred, one model stepped back while the other moved forward.

LightGBM's role faded as its influence decreased from 0.48 to 0.31. Simultaneously, ResNet began carrying more load, and the number beside it rose from 0.29 to 0.44. Without any manual push, decisions shifted to where they needed to go. The deeper network adjusted faster when the data habits changed. Behind the scenes, the balance rewired itself. Surprisingly, the model adjusted its weights in approximately 340 s, which is far quicker than the usual 18–45 min required for a full retraining on the CCFD set. Because of this speed, updating weights through online Bayesian methods works well as a temporary fix while waiting for deeper training.

Despite its benefits, the present system is not free of issues. The ACC uses IPMI-driven tracking to estimate energy use when assigning tasks. Although common in business-grade servers, this method fails when IPMI is missing, such as in certain cloud VMs or edge gadgets. Consequently, moving the efficient module becomes more difficult. Later efforts might look into code-level ways to judge power draw, perhaps by tapping CPU activity signals instead, cutting ties to special hardware entirely.

One drawback is that the ADAI works best with organized, time-based data. In the future, it might begin handling messy inputs such as pictures, words, or clips. Built-in upgrades using recent large-scale models could make this possible. Such changes would allow the system to manage more types of analyses. The shape of things later may include far broader uses of AI.

CONCLUSION

A new system, called ADAI, blends stream and batch data handling with smart learning models. Instead of treating tasks separately, it weaves them together using real-time feedback loops. In tests involving business queries, particle physics sorting, spotting money scams, and factory sensor streams, the speeds increased nearly fourfold. Accuracy increased as well, up to nine points higher than older prediction setups. Energy use dropped by over 30% owing to smarter job distribution across servers. Even when one machine failed, almost all output remained intact. What makes this different is how it adjusts on-the-fly; old systems freeze their settings, but ADAI shifts the weights in its models as patterns change. This helps to handle shifting real-world data better than fixed approaches. Behind the scenes, a built-in manager constantly reshapes resources, avoiding slowdowns before they occur. Evidence from actual runs shows that it holds up under pressure. Not just theory—it works in practice. This approach ties heavy-duty computing power to flexible thinking machines, closing a long-standing gap in data platforms.

Acknowledgment

The authors would like to express their sincere gratitude to the faculty and administration of the Greater Noida Institute of Technology (GNIT) for providing an academic environment and infrastructure necessary to carry out this research. Special thanks are extended to the colleagues and peers who offered valuable insights, discussions, and feedback during the development of this work.

The authors also acknowledge the open-source communities behind technologies such as Apache Spark, Apache Kafka, and related machine learning frameworks, whose tools and resources significantly supported the implementation and evaluation of the proposed system.

Disclosure of Interests. The authors declare that there are no competing financial, professional, or personal interests that could have influenced the work reported in this study. This research was conducted solely for academic and scientific purposes, and no external funding or commercial support was received that could have presented a conflict of interest.

REFERENCES

1. Laney D. 3D data management: Controlling data volume, velocity and variety. META Group Research Note. 2001;6:1.
2. Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. Commun ACM. 2008;51(1):107-13. doi:10.1145/1327452.1327492.
3. Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, et al. Apache Spark: A unified engine for big data processing. Commun ACM. 2016;59(11):56-65. doi:10.1145/2934664.
4. Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K. Apache Flink: Stream and batch processing in a single engine. IEEE Data Eng Bull. 2015;38(4):28-38.
5. Chen T, Guestrin C. XGBoost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2016. p. 785-94. doi:10.1145/2939672.2939785.
6. Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, et al. LightGBM: A highly efficient gradient boosting decision tree. Adv Neural Inf Process Syst. 2017;30.
7. Breiman L. Random forests. Mach Learn. 2001;45(1):5-32. doi:10.1023/A:1010933404324.
8. Hoeting JA, Madigan D, Raftery AE, Volinsky CT. Bayesian model averaging: A tutorial. Stat Sci. 1999;14:382-417.
9. Avnur R, Hellerstein JM. Eddies: continuously adaptive query processing. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data; 2000; Dallas, TX, USA. New York (NY): Association for Computing Machinery; 2000. p. 261-72. doi:10.1145/342009.335420.
10. Marcus R, Negi P, Mao H, Tatbul N, Alizadeh M, Kraska T. Bao: Making learned query optimization practical. In: Proceedings of the 2021 International Conference on Management of Data; 2021. p. 1275-88. doi:10.1145/3448016.3452838.
11. Baldi P, Sadowski P, Whiteson D. Searching for exotic particles in high-energy physics with deep learning. Nat Commun. 2014;5:4308. doi:10.1038/ncomms5308. PMID:24986233.
12. Gandomi A, Haider M. Beyond the hype: Big data concepts, methods, and analytics. Int J Inf Manag. 2015;35(2):137-44. doi:10.1016/j.ijinfomgt.2014.10.007.
13. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms [Preprint]. 2017 Jul 20. arXiv:1707.06347.
14. Bifet A, Holmes G, Pfahringer B, Kranen P, Kremer H, Jansen T, et al. MOA: Massive online analysis, a framework for stream classification and clustering. In: Proceedings of the First Workshop on Applications of Pattern Analysis. PMLR; 2010. p. 44-50.
15. Wang H, Fan W, Yu PS, Han J. Mining concept-drifting data streams using ensemble classifiers. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2003. p. 226-35. doi:10.1145/956750.956778.
16. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: Synthetic minority over-sampling technique. J Artif Intell Res. 2002;16:321-57. doi:10.1613/jair.953.
17. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al. Human-level control through deep reinforcement learning. Nature. 2015;518:529-33. doi:10.1038/nature14236. PMID:25719670.
18. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN. Attention is all you need. Adv Neural Inf Process Syst. 2017;30:261-72.
19. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521:436-44. doi:10.1038/nature14539. PMID:26017442.