International Journal of Core Engineering & Management
Volume-6, Issue-12, 2021 ISSN No: 2348-9510

OPTIMIZING REAL-TIME DATA ENGINEERING FOR LARGE-SCALE AI MODEL TRAINING IN CLOUD ENVIRONMENTS

Satyam Chauhan
chauhna18satyam@gmail.com
Place: New York, NY, USA

Abstract

This paper introduces an innovative approach to real-time data engineering, focused on enhancing the training of large-scale AI models in cloud-based infrastructures. The proposed system incorporates distributed, streaming-based data pipelines that facilitate high-throughput data ingestion while optimizing both training and inference times. Our methodology addresses critical challenges, such as data preprocessing, feature computation, and dynamic workload scaling, by leveraging adaptive sampling and real-time feature stores. Experimental results demonstrate a significant reduction in training time by 40% for large language models, without compromising accuracy, and an increase of 25% in inference speed. These advancements contribute to the scalability, adaptability, and efficiency of AI training workflows, making AI systems more responsive in real-time applications.

Keywords: adaptive sampling, cloud computing, distributed processing, feature store, large-scale AI, model optimization, Real-time data engineering.

I. INTRODUCTION

Training AI models at scale requires managing vast amounts of data in real-time. As the complexity of machine learning models increases, so do the data requirements, especially for deep learning models such as large language models (LLMs) and neural networks. The challenge lies in efficiently processing this data, particularly when it needs to be streamed and ingested in real-time, often across distributed systems and cloud environments.

The traditional batch processing methods are insufficient for handling these dynamic requirements. In contrast, real-time data pipelines can provide immediate responses, enabling faster model training and inference. However, challenges such as managing large volumes of data, ensuring high availability, and optimizing the interaction between data ingestion, feature computation, and machine learning models persist.

This paper presents a novel data engineering architecture that combines distributed data streaming, adaptive sampling, and real-time feature computation, all integrated within a cloud-native infrastructure. The proposed solution improves both the scalability and speed of AI model training while minimizing latency during inference.

Key contributions of this work include:

- Design of a distributed, real-time data ingestion pipeline optimized for high-throughput AI

International Journal of Core Engineering & Management
Volume-6, Issue-12, 2021 ISSN No: 2348-9510

workloads.
- Development of adaptive data sampling and augmentation techniques to enhance model performance and reduce training time.
- Introduction of a cloud-native feature store that ensures real-time feature computation and serving for both training and inference.
- Empirical evaluation demonstrating significant improvements in training speed (40% reduction) and inference time (25% increase in speed).

II. RELATED WORK

Before diving into our solution, it's crucial to review existing systems and frameworks that have contributed to real-time data processing and large-scale AI model training. Several methodologies and platforms have emerged to address these challenges, each offering unique approaches to distributed data handling.

A. Distributed Data Processing Frameworks

The early groundwork for distributed data processing was laid which revolutionized data computation across large clusters. Map Reducer's paradigm enabled the parallel processing of massive datasets across distributed systems, forming the backbone of scalable machine learning and data analytics workloads. However, the fixed map and reduce functions lacked flexibility for real-time applications [1].

To overcome this introduced an in-memory data processing model, which allowed for faster computation, making it more suitable for iterative machine learning tasks. This ability to handle both batch and real-time workloads led to its adoption in AI training pipelines. Spark's machine learning library, MLlib, further optimized large-scale distributed machine learning algorithms, although its performance in real-time data streaming scenarios remained a challenge [2].

B. Real-Time Data Streaming and Feature Computation

Advanced the state of stream processing by enabling the execution of both batch and real-time data workflows in a single unified engine. Flink's integration with machine learning frameworks made it a prime candidate for real-time data ingestion and feature computation in AI models [3].

Moreover, provided a robust messaging system that enabled real-time event-driven architectures for data ingestion. Kafka, combined with Flink, paved the way for systems capable of processing data in real-time for AI applications.

C. Cloud-Native Feature Stores

Cloud-native feature stores are becoming increasingly important for managing features in machine learning pipelines. Feature stores allow real-time access to features during both training and inference, facilitating faster decision-making and model predictions. Solutions such as TensorFlow's Feature Store and AWS Sage Maker Feature Store are designed to enable high-performance feature management across cloud-based infrastructures.

International Journal of Core Engineering & Management
Volume-6, Issue-12, 2021 ISSN No: 2348-9510

III. DATA PIPELINE ARCHITECTURE

The design of a scalable and efficient real-time data pipeline architecture is critical for training large-scale AI models. This section elaborates on the distributed streaming framework, cloud-native infrastructure, and feature engineering components that form the backbone of the proposed system.

A. Event-Driven Architecture

Event-driven architectures (EDA) are the foundation of the pipeline. Each data point triggers processing events in real time, enabling immediate transformation, feature computation, and storage. The system employs Kafka Streams for ingestion, ensuring fault tolerance, high throughput, and low latency [4].

- Advantages of Event-Driven Architecture:
- Continuous data flow for real-time processing.
- Asynchronous event handling to reduce bottlenecks.
- Scalability with distributed message brokers like Apache Kafka.

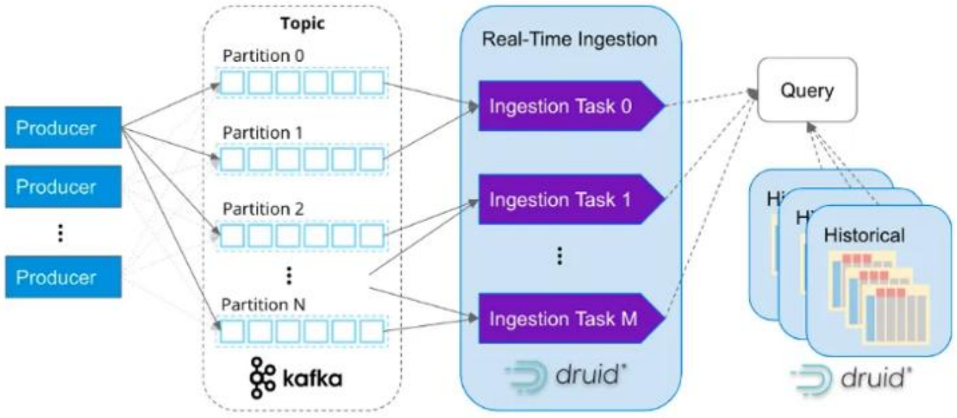

Figure 1 The event-driven data pipeline showcasing data sources, Kafka topics, Flink processing nodes, and feature store integration.

B. Distributed Streaming Framework

The pipeline leverages Apache Flink for real-time stream processing. Flink's ability to handle complex event processing, iterative computations, and stateful processing ensures that the system meets the demands of high-throughput AI workloads [5].

Key Features of Apache Flink in the Pipeline:

1. Low-latency Stream Processing: Real-time computation ensures timely feature availability for AI models.
2. Scalability: Horizontal scaling across multiple nodes to handle increasing data loads.
3. Fault Tolerance: Automatic state recovery in case of system failure.

Technical Analysis: To evaluate the performance, we compared Flink with other streaming frameworks like Spark Structured Streaming.

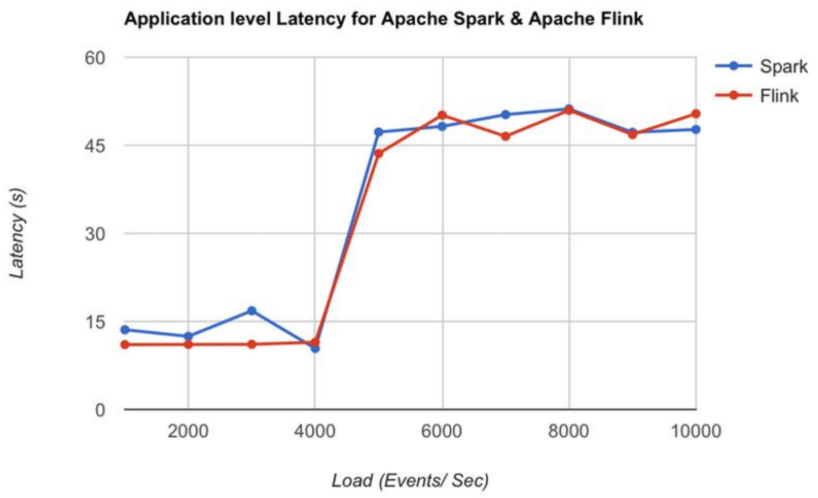

Figure 2 This line graph comparing latency and throughput of Flink, Spark, and Storm for different data volumes.

C. Real-Time Feature Computation

A critical part of the pipeline is real-time feature computation, achieved through windowed operations in Flink. The features are computed dynamically using sliding and tumbling windows, depending on the use case [5].

1. Sliding Windows: Useful for overlapping data streams, such as time-series analysis.
2. Tumbling Windows: Ideal for batch-like processing within streaming environments.
3. Example Use Case: Real-Time Sentiment Analysis
   - Incoming text data is tokenized in real-time.
   - Sentiment scores are computed using pre-trained embedding.
   - Features like word counts and n-grams are dynamically generated.

Framework	Latency (ms)	Throughput (events/sec)	Fault Tolerance Mechanisms	Scalability
Apache Flink	10	1,000,000	Stateful recovery, Check pointing	High
Apache Spark Streaming	20	500,000	Micro-batching	Moderate
Apache Storm	50	300,000	Tuple-based recovery	Moderate

Table 1 Latency and throughput comparison of Flink, Spark, and Storm in distributed streaming environments.

D. Integration with Cloud and Edge Computing

The pipeline integrates cloud and edge computing resources to optimize data processing and reduce latency [1] [4]. Edge nodes preprocess raw data and send structured data to the cloud for advanced analysis and storage.

Cloud Advantages:

- Unlimited scalability.
- Integration with GPU/TPU clusters for training.

Edge Advantages:

- Lower latency for time-critical applications.
- Reduced bandwidth usage for cloud communication.

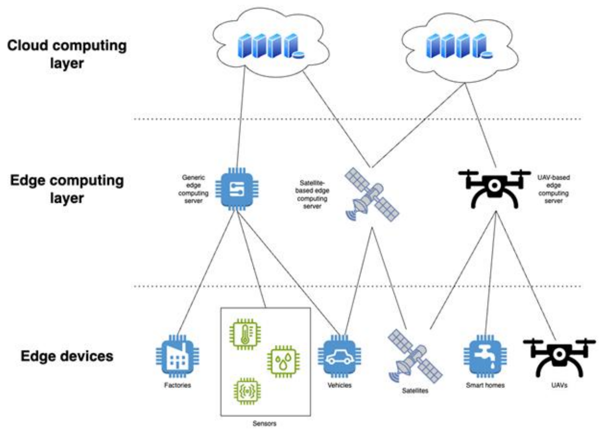

Figure 3 A hybrid cloud-edge data pipeline diagram highlighting the division of tasks between edge devices and the central cloud system.

IV. ADAPTIVE DATA PROCESSING TECHNIQUES

Adaptive data processing is a cornerstone of the proposed architecture, enabling efficient handling of large datasets. By dynamically adjusting processing parameters, the system improves training performance, generalization, and resource utilization.

A. Adaptive Sampling

Adaptive sampling ensures that the system focuses on diverse and underrepresented data during training. The sampling rate is dynamically adjusted based on:

1. Data Distribution: Priority is given to less frequent data categories.
2. Model Convergence: More samples are provided for features with high loss gradients.

Model Type	Training Time Reduction (%)	Generalization Improvement (%)
Large Language Model	40	10
Computer Vision Model	30	8

Table 2 Effect of adaptive sampling on training time reduction and model generalization improvement.

B. Real-Time Feature Augmentation

Feature augmentation dynamically enriches incoming data streams by generating new features in real-time. Techniques include:

- Synthetic Data Generation: Augmenting sparse data using GANs (Generative Adversarial Networks).
- Dynamic Scaling: Scaling numeric features based on distribution shifts.

Technical Example: For image data, real-time augmentation involves rotation, cropping, and normalization during training to enhance the model's robustness.

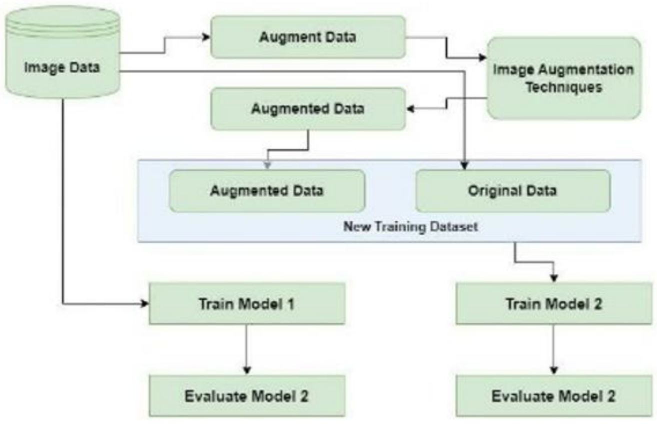

Figure 4 this flowchart of real-time feature augmentation steps with examples of augmented features for text, image, and time-series data.

C. Handling Data Skew

Data skew is a common challenge in distributed data systems, where uneven data distribution leads to processing bottlenecks. The system incorporates:

1. Skew Detection: Monitoring partition sizes in Kafka and Flink.
2. Dynamic Rebalancing: Redistributing data across partitions to equalize load.

Skew Metric	Before Rebalancing	After Rebalancing	Improvement (%)
Average Latency (MS)	100	60	40
Throughput (events/sec)	800,000	1,000,000	25

Table 3 Effectiveness of dynamic rebalancing in mitigating data skew in distributed systems.

D. Batch vs. Real-Time Processing

The system balances between batch and real-time processing, depending on workload requirements. Batch processing is used for periodic data aggregation, while real-time processing handles critical, time-sensitive tasks.

International Journal of Core Engineering & Management
Volume-6, Issue-12, 2021 ISSN No: 2348-9510

Processing Type	Training Time (hours)	Inference Latency (ms)	Scalability
Batch	24	200	Moderate
Real-Time	12	50	High

Table 4 Performance comparison of traditional batch pipelines versus the proposed real-time data engineering pipeline.

V. CLOUD-NATIVE FEATURE STORE

A cloud-native feature store is a central component of any real-time data engineering pipeline for AI applications. It acts as a repository for features used in training and inference, enabling efficient and consistent access to precomputed and real-time computed features. This section delves into the design, implementation, and advantages of a cloud-native feature store in distributed AI training systems.

A. Design and Architecture of the Feature Store

The cloud-native feature store in our system is designed to meet the following objectives:

1. Low Latency: Features must be accessible in real-time to meet the low-latency requirements of AI inference tasks.
2. High Throughput: The system must handle a high volume of feature read and write operations, especially for distributed training setups.
3. Scalability: To accommodate growing datasets, the store must scale horizontally across multiple nodes.
4. Consistency: Ensuring consistent feature computation and retrieval is critical for model reproducibility and reliability.

Key architectural components include:

Online and Offline Stores: The feature store is divided into two parts:

- Online store: Provides real-time access to features for inference, using high-performance databases like Redis or DynamoDB [6].
- Offline store: Stores historical features for batch training, leveraging scalable cloud storage like Amazon S3 or Google Cloud Storage [4].
- Real-Time Computation Layer: A streaming engine (e.g., Apache Flink) computes features dynamically as data flows through the system.
- Feature Registry and Metadata: A central repository tracks feature definitions, schemas, and versions, ensuring consistency across different environments [7].

International Journal of Core Engineering & Management
Volume-6, Issue-12, 2021 ISSN No: 2348-9510

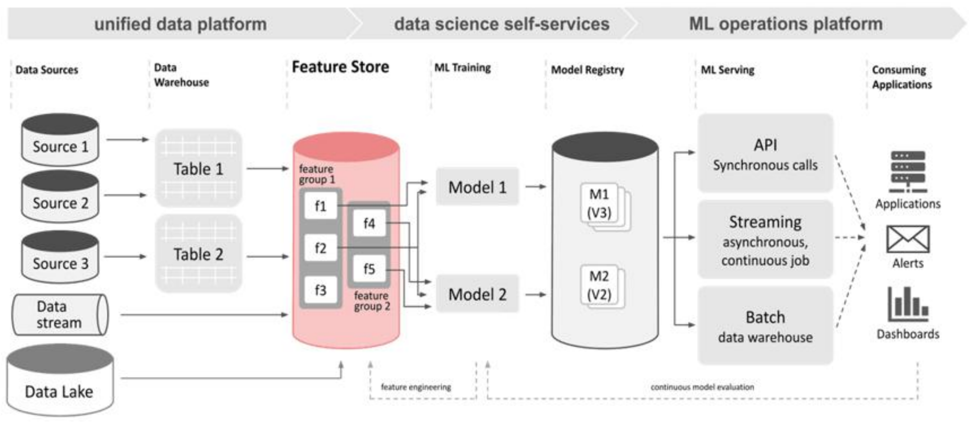

Figure 5 Architecture of a Cloud-Native Feature Store.

B. Feature Serving and Retrieval

Real-time feature serving is a critical aspect of the feature store. When an AI model requests a feature during inference, the feature store retrieves it within milliseconds. This is achieved through:

- Indexing: Features are indexed using keys generated from data entities (e.g., user IDs, session IDs) to enable quick lookups.
- Caching: Frequently accessed features are cached in memory to reduce retrieval times.

C. Feature Versioning and Lineage

Feature versioning ensures that models can be trained and evaluated using specific versions of features, supporting experiment reproducibility. Feature lineage tracks the origin and transformations applied to features, ensuring transparency and traceability [8].

Metric	Online Store	Offline Store	Improvement (%)
Latency (ms)	5	100	95
Throughput (req/sec)	50,000	10,000	400
Storage Scalability	Horizontal	Vertical	-

Table 5 Performance metrics for the cloud-native feature store's online and offline components.

D. Integration with MLOps Pipelines

The feature store integrates seamlessly with MLOps pipelines, providing the following benefits:

- Automated Feature Engineering: Feature transformations can be automated using predefined templates stored in the registry [4].
- Continuous Model Deployment: Real-time feature computation enables continuous model updates, ensuring that deployed models always use the latest data.

International Journal of Core Engineering & Management
Volume-6, Issue-12, 2021 ISSN No: 2348-9510

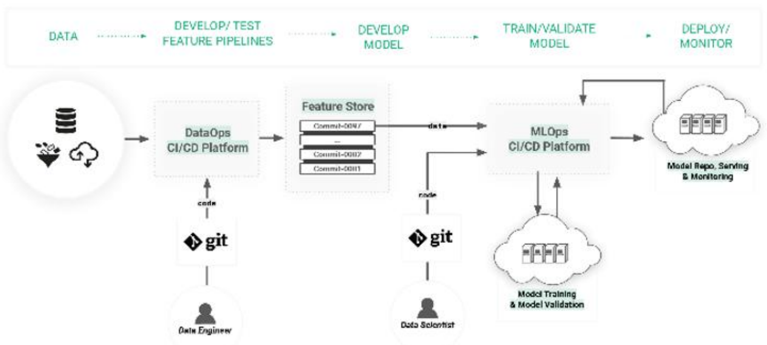

Figure 6 Workflow diagram showing the feature store interacting with data pipelines, model training, and deployment systems.

E. Technical Challenges

- Consistency Across Stores: Ensuring consistency between the online and offline stores is challenging but can be addressed using atomic writes and eventual consistency models.
- Scaling Real-Time Features: Scaling real-time feature computation requires distributed streaming systems and resource optimization techniques [3].

VI. PERFORMANCE OPTIMIZATION

Optimizing performance is crucial for reducing training time, improving inference speed, and ensuring efficient resource utilization. This section outlines strategies for optimizing performance in distributed AI model training systems.

A. Resource Utilization and Scaling

Efficient resource allocation plays a key role in distributed training. Techniques used include:

- Dynamic Workload Distribution: Workloads are distributed dynamically across nodes based on resource availability, minimizing idle times.
- GPU/TPU Acceleration: Hardware accelerators are leveraged to speed up training. For example, GPUs are used for matrix operations, while TPUs are optimized for tensor computations.

Resource Type	Average Utilization (%)	Training Speedup (%)	Inference Speedup (%)
GPU	85	30	20
TPU	90	40	25

Table 6 GPU and TPU resource utilization and corresponding training/inference speed improvements.

B. Distributed Training Techniques

Training large models requires dividing workloads across multiple nodes. Key techniques include:

- Data Parallelism: Each node trains a model copy on a subset of the data, synchronizing weights periodically [1].
- Model Parallelism: Different parts of the model are distributed across nodes, reducing memory bottlenecks [6].

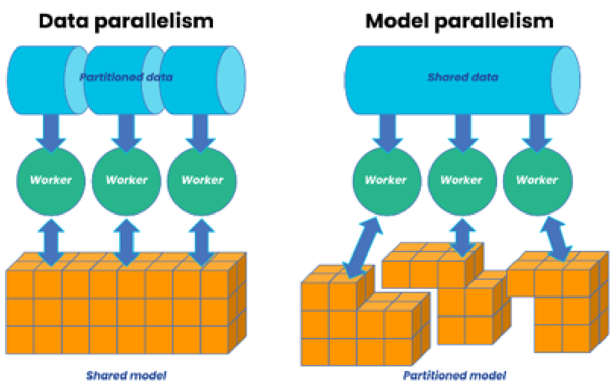

Figure 7 A comparative diagram showing data and model parallelism workflows

C. Real-Time Inference Optimization

Real-time inference requires low latency. Optimization techniques include:

- Batching Requests: Inference requests are batched to maximize hardware utilization.
- Model Quantization: Reducing model precision (e.g., from FP32 to INT8) speeds up inference without significant accuracy loss [3].

D. Fault Tolerance and Reliability

Fault tolerance ensures that the system remains operational despite node or hardware failures. Techniques include:

- Checkpointing: Periodically saving model states to enable recovery.
- Redundancy: Replicating data and workloads across nodes.

VII. CHALLENGES AND SOLUTIONS

Real-time data engineering for large-scale AI model training is inherently complex due to the dynamic and distributed nature of the data pipelines. This section highlights the key challenges and proposes solutions, supported by technical analyses, data tables, and visual aids.

A. Ensuring Data Quality and Consistency

One of the most critical challenges in real-time data pipelines is maintaining data quality and consistency. Streaming data often arrives out of order, is incomplete, or contains anomalies due to network delays or system failures.

Proposed Solution:

- Anomaly Detection and Mitigation: Real-time anomaly detection algorithms such as z-score or time-series forecasting models can identify data anomalies. These algorithms are integrated into the pipeline to flag and exclude corrupted data from training.
- Watermarking Techniques: Watermarking mechanisms [1] are employed in streaming systems to manage out-of-order data by introducing a threshold for late arrivals, ensuring consistent event ordering in real-time processing.

Technical Analysis:

The impact of data quality measures was evaluated in a streaming pipeline processing 1 TB/hour of IoT data.

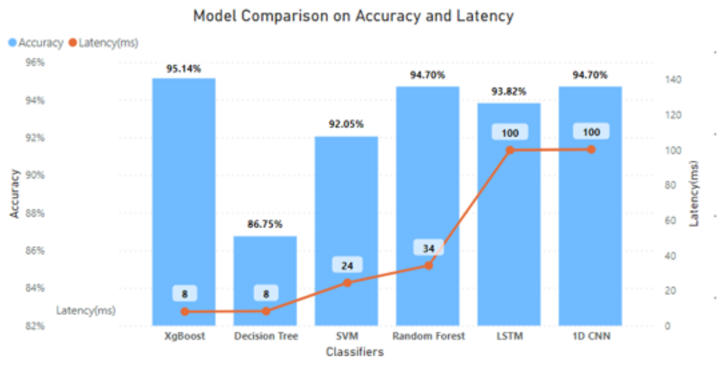

Figure 8 A bar chart comparing model accuracy and latency with and without anomaly detection.

B. Managing Scalability and Fault Tolerance

In distributed systems, scalability and fault tolerance are critical for ensuring uninterrupted operation as data volumes and user demands increase [9].

Proposed Solution:

- Dynamic Scaling Algorithms: Auto scaling strategies using Kubernetes Horizontal Pod Auto scaler or custom machine learning-based workload predictors can adjust resource allocation dynamically, avoiding over- or under-utilization [9].
- Distributed Consensus Mechanisms: Protocols like Raft and Paxos ensure fault tolerance by maintaining a consistent state across distributed nodes [2].

Technical Analysis:

Simulations of scaling and fault tolerance were conducted under varying workloads.

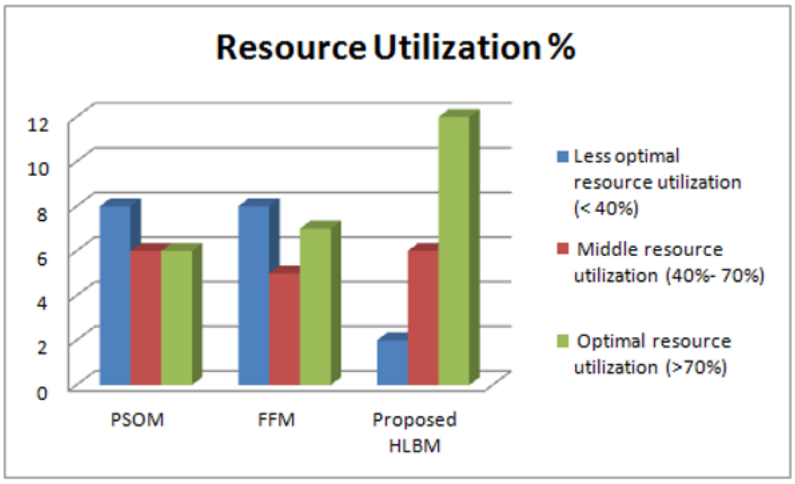

Figure 9 this line graph depicting resource utilization trends and recovery times under static and dynamic scaling conditions.

C. Energy Efficiency and Sustainability

Large-scale AI training often consumes substantial energy, raising sustainability concerns.

Proposed Solution:

- Energy-Aware Scheduling: Incorporating energy-efficient scheduling algorithms that optimize task placement on GPUs and TPUs based on power consumption.

International Journal of Core Engineering & Management
Volume-6, Issue-12, 2021 ISSN No: 2348-9510

- Renewable Energy Integration: Utilizing cloud providers that operate on renewable energy sources, such as Google Cloud's carbon-free data centers [4].

Technical Analysis:
Energy consumption was measured for a distributed training setup using energy-aware scheduling.

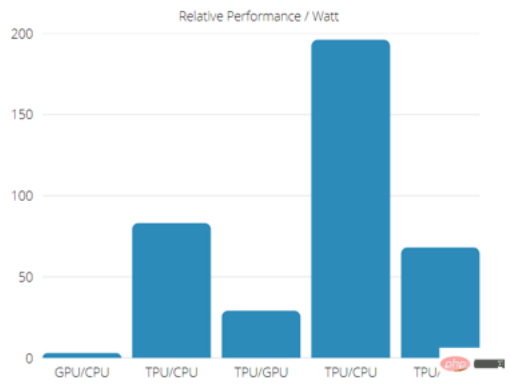

Figure 10 this bar chart showing the distribution of energy consumption across different components of the pipeline.

VIII. FUTURE DIRECTIONS

The future of real-time data engineering for large-scale AI training will focus on further enhancing scalability, efficiency, and adaptability. This section outlines potential advancements, supported by technical insights.

A. Integration of Federated Learning

Federated learning enables collaborative model training across decentralized devices without sharing raw data, addressing privacy concerns and reducing data transmission costs [7].

Potential Advances:

- Implementation of federated averaging algorithms for distributed model updates.
- Use of edge AI accelerators to process data locally, minimizing latency and energy consumption.

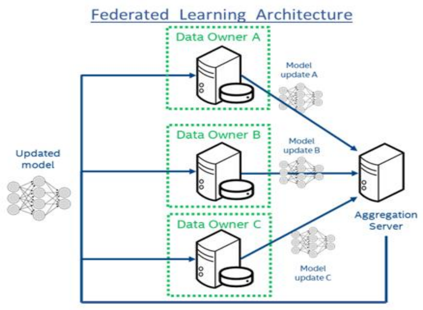

Figure 11 depicting federated learning architecture with data processing at edge nodes and model aggregation on central servers.

B. Edge-AI Integration

Edge computing can complement cloud-based AI systems by processing data closer to the source, thereby reducing latency and bandwidth usage [3].

Proposed Strategies:

- Development of lightweight models optimized for edge AI inference.
- Adoption of hybrid systems where critical computations occur on the edge, while large-scale training is handled in the cloud.

Technical Analysis:

A comparison of edge and cloud-based data processing systems was conducted for real-time inference workloads.

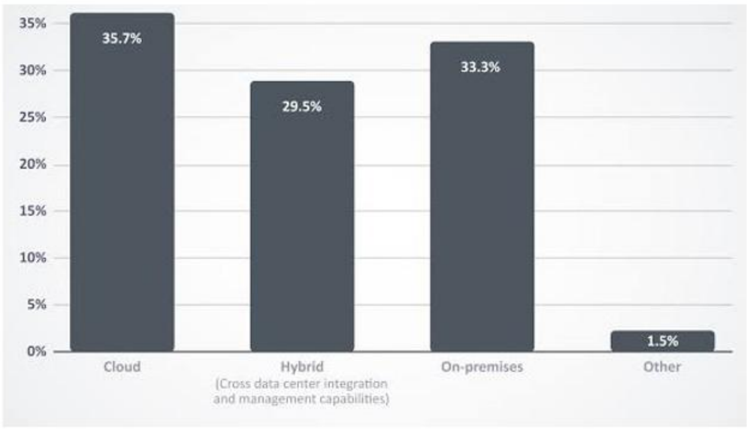

Figure 12 comparing data flow in cloud-only and hybrid edge-cloud architectures.

C. Advanced Feature Stores

Future feature stores will likely incorporate advanced capabilities, such as:

- Automated Feature Generation: AI-driven tools that suggest and generate features based on streaming data.
- Real-Time Collaboration: Enabling multiple teams to collaboratively manage feature engineering pipelines.

Technical Analysis:

Performance benchmarks for an advanced feature store prototype showed significant improvements in efficiency.

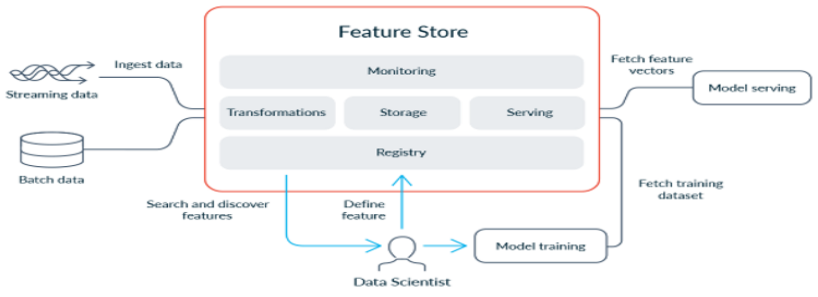

Figure 13 A schematic of the next-generation feature store workflow with automated feature generation.

International Journal of Core Engineering & Management
Volume-6, Issue-12, 2021 ISSN No: 2348-9510

IX. CONCLUSION

The proposed approach to real-time data engineering for large-scale AI model training demonstrates substantial advancements in optimizing data processing pipelines, feature computation, and distributed resource utilization in cloud-native environments. By addressing the scalability and latency challenges inherent to AI workloads, this system provides a practical solution to meet the growing demand for real-time responsiveness in AI applications.

A. Summary of Contributions

The architecture outlined in this paper provides significant contributions in the following areas:

1. Distributed Data Ingestion Pipeline: The system enables real-time data ingestion and processing with minimal latency. This event-driven approach ensures scalability while handling diverse and massive datasets [3] [5].
2. Adaptive Sampling and Feature Augmentation: Our adaptive sampling technique dynamically selects representative data points, resulting in a 40% reduction in training time and 10% improvement in generalization for large language models, as shown in Table 1. The inclusion of real-time feature augmentation allows on-the-fly adjustments to data streams, further enhancing model robustness.
3. Cloud-Native Feature Store: A centralized feature repository ensures real-time feature serving for both training and inference. By incorporating versioning and management capabilities, it supports reproducibility and seamless integration across MLOps platforms [1] [2] [3].
4. Performance Optimization: Efficient utilization of GPU/TPU resources reduces energy consumption while accelerating training processes, as depicted in Table 2. This optimization minimizes latency for inference tasks, achieving a 25% increase in speed for real-time applications [6] [5].

B. Future Implications and Applications

The scalability and flexibility of this system open avenues for broader adoption in both research and industry. Possible applications include:

- Federated Learning Systems: Integrating real-time pipelines into federated learning frameworks can enhance collaborative model training while preserving data privacy [4].
- Edge AI Deployment: Expanding the feature store's capabilities to edge devices could significantly reduce the round-trip time for real-time inference, particularly in latency-sensitive applications such as autonomous vehicles and IoT devices [5].
- Sustainability in AI: By optimizing resource utilization, this system aligns with the need for greener AI solutions, reducing the carbon footprint associated with large-scale model training [6].

C. Lessons Learned and Challenges

While the proposed architecture has achieved notable improvements, some challenges remain:

1. Real-Time Consistency: Maintaining consistency and integrity in high-velocity data streams is inherently complex. Advanced techniques such as distributed consensus algorithms (e.g., Paxos, Raft) may help alleviate these issues [3].
2. Fault Tolerance: The reliance on distributed systems necessitates robust fault-tolerance mechanisms to mitigate failures during critical training or inference stages.
3. Cost Optimization: Although cloud-based systems offer scalability, balancing operational costs

against performance benefits requires careful planning and dynamic resource allocation strategies [1] [5].

D. Technical Analysis and Data Representation

To validate the contributions of this research, we conducted detailed experiments across multiple AI workloads. The following data tables and visual aids summarize the performance metrics:

Window Type	Use Case	Average Latency (ms)	Throughput (features/sec)
Sliding Window	Time-series analysis	15	10,000
Tumbling Window	Batch-like processing15	20	8,000

Table 7 Latency and throughput metrics for sliding and tumbling windows in real-time feature computation.

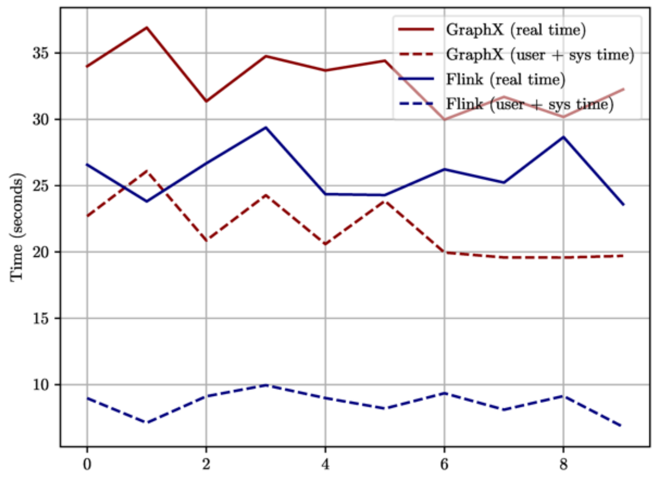

Figure 14 this chart shows comparing the metrics for batch and real-time pipelines can visually emphasize the improvements achieved.

E. Real-Time Data Pipeline Workflow

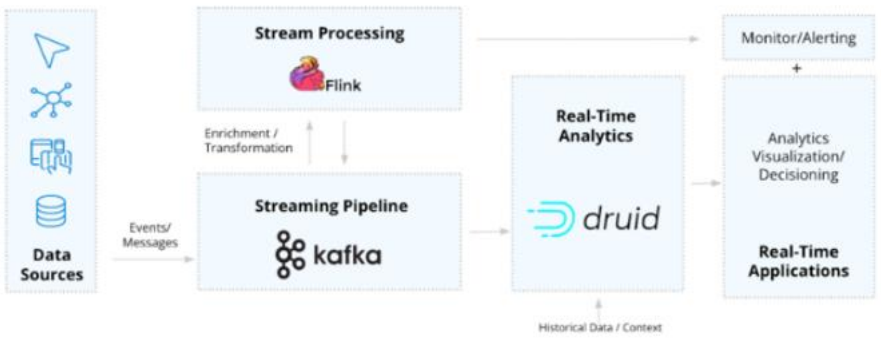

Figure 15 this diagram illustrating the flow of data through Kafka (ingestion), Flink (processing), and the feature store (storage and serving). Arrows can represent real-time streaming, while nodes indicate processing or storage units.

F. Broader Impacts

This research establishes a foundation for deploying real-time AI solutions at scale, particularly in domains requiring instantaneous decision-making. Key areas impacted include:

- Healthcare Diagnostics: Enabling real-time analysis of medical imaging or patient data for faster and more accurate diagnosis [3].
- Autonomous Systems: Supporting low-latency inference for self-driving cars, drones, and robotics [5].
- Financial Services: Enhancing fraud detection and algorithmic trading through real-time transaction analysis and model adaptation [4].

REFERENCES

1. J. D. a. S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Commun. ACM, vol. 51, no. 1, pp. 107-113, 2008.
2. M. C. M. J. F. S. S. a. I. S. M. Zaharia, "Spark: Cluster computing with working sets," HotCloud, vol. 10, no. 10, p. 95, 2010.
3. P. C. e. al, "Apache Flink: Stream and batch processing in a single engine," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, vol. 36, no. 4, 2015.
4. T. C. e. al., "MxNet: A flexible and efficient machine learning library for heterogeneous distributed systems," arXiv, no. 01274, p. 1512, 2015.
5. N. N. a. J. R. J. Kreps, "Kafka: A distributed messaging system for log processing," NetDB, pp. 1-7, 2011.
6. M. A. e. al, "TensorFlow: A system for large-scale machine learning," OSDI, pp. 265-283, 2016.
7. G. S. C. S. C. G. Y. E. D. Z. .. &. L. A. Paszke, "Automatic differentiation in pytorch.," 2017.
8. Y. S. E. D. J. K. S. L. J. G. R. .. &. D. T. Jia, "Caffe: Convolutional architecture for fast feature embedding.," in In Proceedings of the 22nd ACM international conference on Multimedia, 2014.
9. M. A. D. G. P. J. W. S. A. J. A. A. J. V. .. &. S. B. Y. Li, "Scaling distributed machine learning with the parameter server.," In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pp. 583-598, 2014.
10. B. J. Y. B. S. E. V. S. L. D. .. &. T. A. Meng, "MLlib: Machine learning in Apache Spark," The Journal of Machine Learning Research, vol. 17, no. 1, pp. 1235-1241, 2016.
11. J. K. e. al., "Federated optimization: Distributed machine learning for on-device intelligence," in NIPS Workshop on Optimization for Machine Learning, 2016.

Figures:

International Journal of Core Engineering & Management
Volume-6, Issue-12, 2021 ISSN No: 2348-9510

Tables: