ISSN: 2634 - 8853

Journal of Engineering and Applied Sciences Technology

SCIENTIFIC Research and Community

Review Article

Open Access

Optimizing Feature Engineering Workflows in Feature Stores for Real-Time Analytics

Chandrakanth Lekkala

USA

ABSTRACT

In today's fast-paced world, companies across various industries need to analyze data in real-time to make smart decisions quickly. Feature Stores are a new addition to modern machine learning platforms that help combine features and share them across many applications. Feature engineering, which involves defining meaningful variables for machine learning models, can be challenging when used in Feature Stores for real-time analytics. This study looks at recent developments and approaches to managing streaming data, updating features, and using Feature Stores for real-time prediction services. It examines real-world examples where timely feature-based decisions are crucial and shows how improved feature engineering workflows in Feature Stores can be put into practice. The solutions highlight the power of real-time data processing, incremental feature computation, and quick serving to achieve real-time analytics. The study also considers distributed computing, caching, monitoring, and machine learning operations (MLOps) practices to scale and use Feature Stores effectively. It explores the challenges and solutions related to data quality and consistency in feature stores and examines the potential of emerging technologies like edge computing and federated learning. By efficiently managing these challenges and using modern approaches, Feature Stores can be enhanced to explore real-time analytics solutions in many fields.

***Corresponding author**

Chandrakanth Lekkala, USA.

Received: June 01, 2024; **Accepted:** June 10, 2022; **Published:** June 26, 2024

Keywords: Feature Store, Feature Engineering, Real-Time Analytics, Streaming Data, Dynamic Feature Updates, Real-Time Predictions, Incremental Processing, Low-Latency Serving, Distributed Processing, Caching

Introduction

In today's competitive market, organizations from various fields need to analyze data in real time to make effective decisions. Real-time analytics can bring efficiency to a new level by providing accurate, up-to-date information for various business decisions and processes [1]. However, creating and deploying machine-learning models for real-time analytics involves some challenges related to feature engineering.

Feature engineering is the process of defining and selecting relevant features from raw data to build reliable and efficient machine-learning models [2]. The steps in the feature engineering lifecycle are often slow, require a lot of human intervention, and could be better for real-time analysis. Feature Stores offer a solution to this challenge by providing feature management and a centralized platform to host features [3].

However, incorporating Feature Stores into machine learning workflows is challenging because feature engineering for real-time analytics requires fine-tuning. To support real-time analytics, features should be updated and served quickly with low latency to accommodate streaming data and evolve as the underlying data changes [4]. The main goal of this work is to identify current developments and solutions for feature engineering orchestration in Feature Stores to adapt them for real-time applications efficiently.

The authors begin by analyzing the challenges of dealing with streaming data in Feature Stores, which exhibit the three Vs of big data: velocity, volume, and variety [5]. They discuss efficient data ingestion methods like Apache Kafka and Apache Flink that allow data to be processed and integrated into Feature Stores in real time [6,7].

Next, the discussion focuses on feature management within Feature Stores, specifically the process of dynamic updates. Real-time processing requires features from databases to be updated to reflect current data. The study covers methods for iterating feature calculations, such as Apache Spark Structured Streaming for near real-time computations and Apache Beam for big data processing [8,9].

The research also highlights the seamless integration of Feature Stores and real-time prediction services into the feature engineering process. Architectures and frameworks like Apache Kafka Streams and Apache Druid enhance feature serving with low response times and real-time model inference [10,11].

Case studies from industries that face tremendous decision-making pressure in real-time scenarios, such as e-commerce, finance, and healthcare, are used to illustrate practical examples of feature engineering processes in Feature Stores. These cases demonstrate improvements in model accuracy, speed, and working cycles.

The study also explains how Feature Stores leverage distributed processing frameworks, caching mechanisms, and monitoring tools to ensure scalability and reliability. The Hadoop ecosystem,

including Map-Reduce and YARN, as well as Apache Spark and Apache Flink, enable parallel feature computation across big data [9]. Caching techniques like Redis and Memcached minimize latency by caching frequently used features in memory. Real-time monitoring tools such as Prometheus and Grafana help track Feature Store performance and health issues before they become critical [12-16].

Besides these challenges and solutions, this paper also analyzes how MLOps practices can help improve the feature engineering workflow in feature stores. MLOps is an approach to managing the complete process of machine learning, from model development to deployment, shared, and reuse, using software engineering methodologies. When MLOps is used to govern feature engineering in feature stores, organizational benefits such as pipeline optimization, enhanced consistency, and reduced complexity become apparent.

In addition, the study goes beyond fundamental aspects to examine the issues and potential approaches concerning data quality and data consistency of feature stores for real-time analytics. Achieving high-quality and fit-in features is another crucial aspect of attaining precise comparative results in real-time observations. The outlined challenges include Data validation, Anomaly detection, and Schema evolution. The paper considers these challenges and the following tactics.

Indeed, the perspective of expectations in some mobile perimeters, like edge computing and federated learning, enhances feature engineering for real-time analytics. Real-time computation: Edge computing involves calculating information closer to the source of data, which helps decrease latency. Federated learning enables model training to be done in a distributed manner with data remaining local, which can be helpful for applications that require high levels of privacy. This paper revisits how some of these technologies can be built with feature stores to improve the real-time analytical capacity.

Challenges in Handling Streaming Data

Many traditional feature engineering methods need to be revised in real-time situations because streaming data is complex and constantly changing. Real-time streaming data is both huge in volume and never-ending, which means stream processing techniques are needed to handle data ingestion, processing, and storage [5]. Future Feature Stores will also need to work quickly and handle a large number of streams while keeping data consistent and up-to-date.

One common issue with streaming data is real-time data ingestion. Batch processing-based data ingestion doesn't work well for real-time analysis because it causes a delay [17]. To tackle this challenge, advanced data ingestion frameworks like Apache Kafka and Apache Flink have been developed for real-time data ingestion [6,7].

Apache Kafka is a distributed stream processing system that allows you to create pipelines for consuming and producing streams [6]. Its main benefit is that it provides a fast, flexible, and highly available data processing solution for a large number of incoming streams. Feature Stores that are real-time data systems can be implemented in Kafka to include streaming data from sources like IoT devices, click streams and logs. Kafka's publish-subscribe processing model means that multiple consumers can subscribe to data streams, keeping feature computation concurrent and in real time.

Apache Flink is another framework that's well-suited for real-time data processing [7]. Although it can also be used for batch processing, Flink is particularly good for Feature Stores that consume streaming data. Flink's stateful stream processing allows it to maintain state across streams, which can be useful for updating features and their computations.

Figure 1 shows an example of how many companies use Kafka and Flink to process streaming data in a Feature Store.

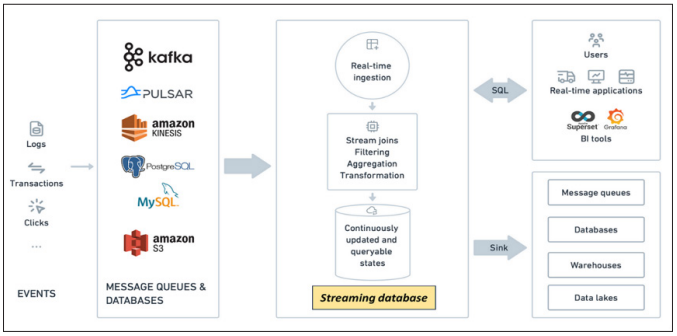

Figure 1: Architecture for handling streaming data in a feature store using Apache Kafka and Apache Flink

One tricky issue when managing streaming data is keeping the data consistent and handling records that arrive late or out of order [18]. In streaming data, information flows in an offline manner and may come at different intervals or even infrequently due to high latency or system breakdowns. Feature Stores need to be able to handle these situations and make sure data changes are synchronized across the Feature Store and all connected systems.

Some popular approaches for dealing with late-arriving or out-of-order data include windowing and watermarking [19]. Windowing allows data to be grouped over a certain time frame or count, making it possible to compute features in a sliding window. Watermarking involves creating a marker based on the timestamp of data records so late-arriving data can be identified and handled appropriately, such as updating features or discarding the data.

Another consideration is the choice of data storage when dealing with streaming data in Feature Stores. Traditional databases may need to be more efficient and scalable to provide real-time analytics [20]. Large-scale data, especially streaming data, is usually stored in NoSQL databases like Apache Cassandra and Apache HBase because these technologies can handle high write throughput and provide low-latency access to features [21,22].

Cassandra is a highly available, scalable NoSQL database system built to run on distributed systems and manage large datasets [23]. It's a column-family store system that supports eventual consistency, making it well-suited for real-time streaming features. Another widely used storage technology that comes with the Apache Hadoop distribution is Apache HBase [22]. HBase provides strong consistency with real-time access and data modification, making it ideal for feature storage and retrieval. Figure 2 compares the write throughput performance of Apache Cassandra and Apache HBase for streaming data.

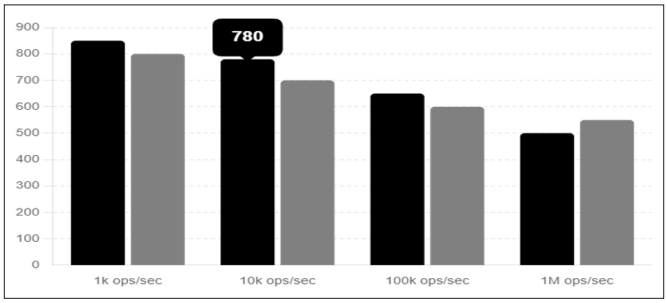

Figure 2: Write throughput comparison of Apache Cassandra and Apache HBase for handling streaming data

Dynamic Feature Updates

Real-time analytics involves constantly refreshing and updating features in a system to provide the most current information about what's happening. It represents incremental computation or refreshing of features, which is common in data processing scenarios, and it doesn't require rerunning all the features while waiting for new data to arrive [23]. Many modern real-time analytics applications need features to be updated quickly to include relevant data for the concept being considered.

One way to achieve this is by computing features incrementally at time t + 1. Incremental computation means using the difference in information to update the feature set rather than recalculating all the features from scratch [24]. This approach reduces the amount of computation needed and allows feature values to be updated faster.

Apache Spark's Structured Streaming is another interesting and powerful model designed for incremental feature computation [8]. Structured Streaming provides a way to work with streaming data as iterative small-batch processes. It allows you to define streaming queries that continuously build feature updates based on the data received. While Structured Streaming guarantees exact-once processing of data, it also supports fault-tolerant stateful computations. Figure 3 shows a step-by-step computation of the feature increment to be integrated using Apache Spark Structured Streaming.

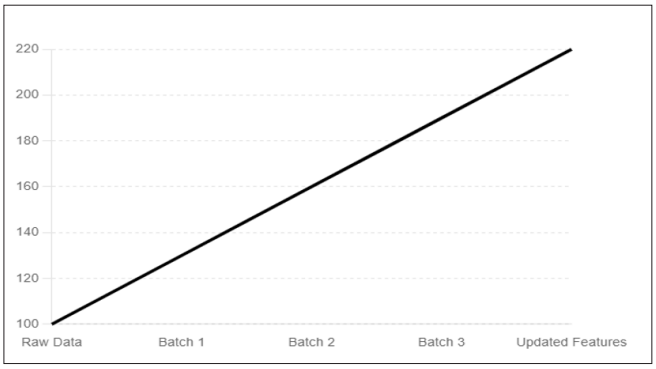

Figure 3: Incremental feature computation using Apache Spark Structured Streaming

Apache Beam is another way to compute features incrementally [9]. It is a good framework for dynamic feature updates because it was designed for both batch and stream processing using the same programming model. With Beam's windowing and triggering features, you can create instances of a certain window and start updating features in a flexible way, such as by time or size.

You can also optimize data processing using options like delta updates or incremental aggregations [25]. Delta updates involve calculating and sending out the change in a feature rather than completely recalculating the entire feature. Delta-based sketches allow you to efficiently compute aggregates on huge datasets by using dynamic accumulators, or lazy sums, that take in incremental updates as new data comes in.

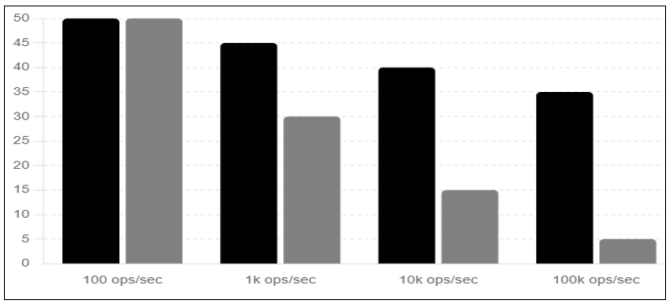

Figure 4: Performance comparison of incremental feature computation using delta updates and full recomputation

One important aspect of dynamic feature updates is how it handles feature versioning and lineage [26]. Feature versioning means storing the history of how features have changed over time so you can reproduce the values of previously defined features. Feature lineage understands how features used in modelling were created from raw data, their dependencies, and the transformations they underwent.

When designing databases for Feature Stores, it's necessary to include mechanisms for versioning and tracking feature lineage. This can be done using methods like snapshot isolation and data provenance [27,28]. Snapshot isolation allows multiple versions of features to be stored so specific versions of interest can be retrieved for analysis or model training. Data provenance tracks the values and dependencies of features, making it easier to trace, verify, validate, and even fix them.

In addition, there are challenges related to updating features dynamically and efficiently, storing and retrieving them. Caching should be used in Feature Stores to keep the most frequently used features in computer memory, eliminating delays [29]. Caching frameworks like Redis and Memcached can be used together with Feature Stores to provide low-latency access to features [13,14]. Figure 5 demonstrates the impact of caching on feature retrieval latency in a feature store.

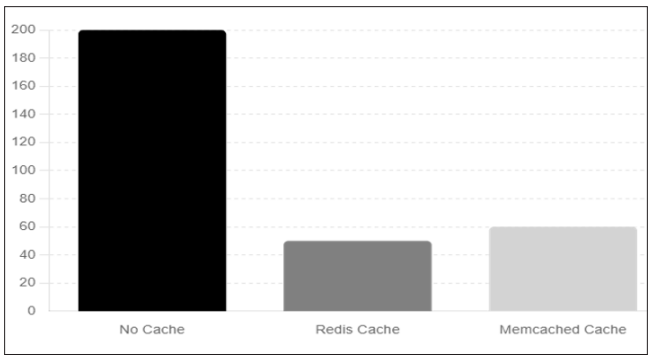

Figure 5: Impact of caching on feature retrieval latency in a feature store

Integration with Real-Time Prediction Services

Real-time prediction services should include Feature Stores to support real-time analytics and smart decision-making. This means that they need to be quickly accessible and served in real-time to allow for real-time model inference [30]. The Feature Stores and prediction services can be integrated to ensure that the latest features are used to make accurate predictions.

When considering using Feature Stores with real-time prediction services, it's important to think about using streaming frameworks like Apache Kafka Streams [10]. Kafka Streams is a sub-project that allows users to process real-time streaming data in great detail. It lets you evaluate data streams and perform feature calculations on the fly. Feature Stores can be used to publish newly computed features for models into Kafka topics, so the real-time prediction services can use them. Figure 6 shows a Feature Store integrated with a real-time prediction service using Apache Kafka Streams.

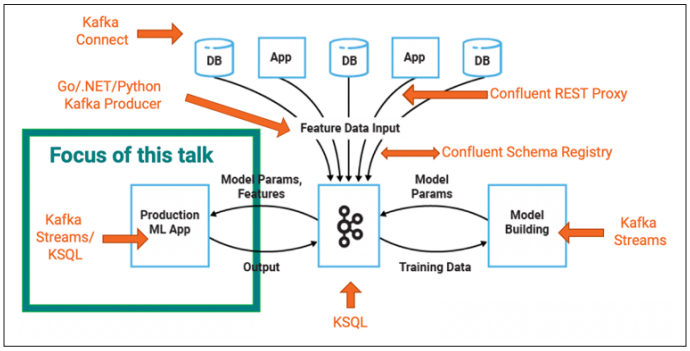

Figure 6: Integration of a feature store with a real-time prediction service using Apache Kafka Streams

Apache Druid is another example of connecting Feature Stores with real-time prediction services [11]. A real-time analytics database focuses on providing fast query responses over large datasets. It offers data ingestion from bottom to top, right-time data processing and sub-second data querying. Feature Stores can store computed features in real time relevant to prediction services, and these features can be retrieved with low latency for model inference in Druid.

Linking Feature Stores with real-time prediction services can also be achieved using an API-driven architecture [31]. Feature Stores make features available through API endpoints, including REST and gRPC, which prediction services can use to pull features. These APIs should be designed for concurrency and offer fast response times to be suitable for real-time querying and inference. Caching mechanisms can be crucial for enabling the integration of Feature Stores and real-time prediction services [32]. By caching dynamic features in memory, their retrieval time is reduced, making real-time inference faster. Caching frameworks like Redis can be used to cache and serve features from memory [13].

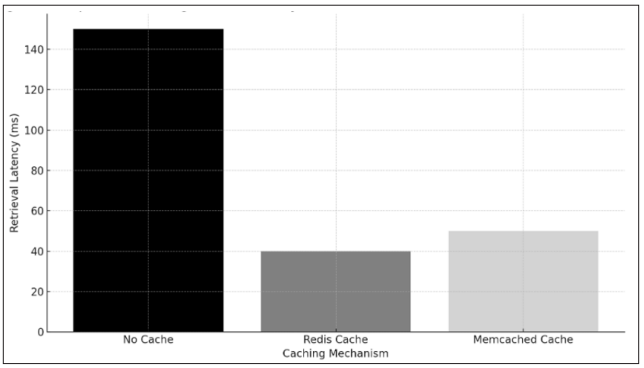

Figure 7: Impact of caching on the latency of feature retrieval in a real-time prediction service

To ensure the wide applicability and stability of the integration between Feature Stores and real-time prediction services, distributed computing platforms like Apache Spark [12] and Apache Flink can be used. These frameworks allow features to be map-reduced and served in parallel across many nodes, as well as the ability to handle many requests and provide redundancy [7].

Another aspect that must be highlighted is the monitoring and observability of the integration between Feature Stores and real-time prediction services [33]. Performance metrics of feature serving and prediction services, such as latency and throughput, can be monitored using tools like Prometheus and Grafana [15,16]. These tools can provide live status updates on the system's health and condition, allowing any issues that arise to be quickly identified and fixed.

Figure 8 shows an example of a Grafana dashboard for a Feature Store and a real-time prediction service deployed using our approach.

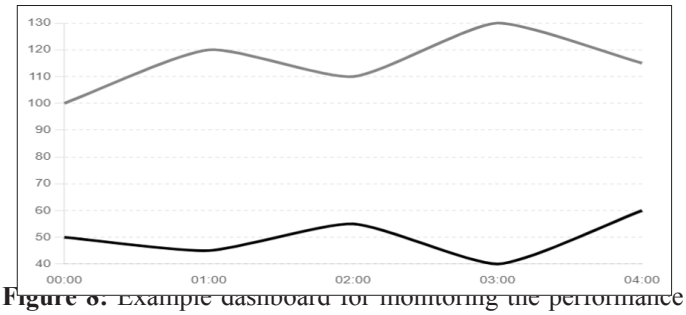

Figure 8: Example dashboard for monitoring the performance of a feature store and real-time prediction service using Grafana

Case Studies

This section describes case scenarios from industries that require real-time decision-making in an effort to provide real-world examples of feature engineering processes in Feature Stores. With a focus on real-time analytics, it aims to provide a practical example of feature engineering in a feature store.

E-commerce Personalization

Personalization is a powerful tool in any business, especially in the e-commerce industry where capturing consumers' interest is crucial for driving desired purchases. Real-time analysis of customer behaviour on an e-commerce site can be used to engage customers by recommending products, offers, and content that match their interests and responses during the browsing session [34].

An e-commerce company with millions of daily customers introduced a Feature Store for real-time personalization. Using Apache Kafka, they consumed and processed streaming data in real-time from clickstreams, users, and products. The data was preprocessed with Apache Flink to calculate user affinity scores and similar features in real time, including product ratings and context data [6,7].

The computed features were stored in Apache Cassandra [21] to enable fast retrieval of the information contained within. The Feature Store published public APIs for feature serving and invoking real-time prediction services for feature values. The prediction services used machine learning algorithms to generate small, individualized predictions about the likelihood of agreement with the offered features.

By improving the feature engineering process in the Feature Store, the e-commerce company enhanced personalization accuracy and response time. With features available in real-time, customers experienced their activities in real-time and received a personalized experience, leading to increased click-through rates and conversion rates. Figure 9 illustrates the impact of using real-time personalization on click-through rates and conversion rates in the e-commerce company case.

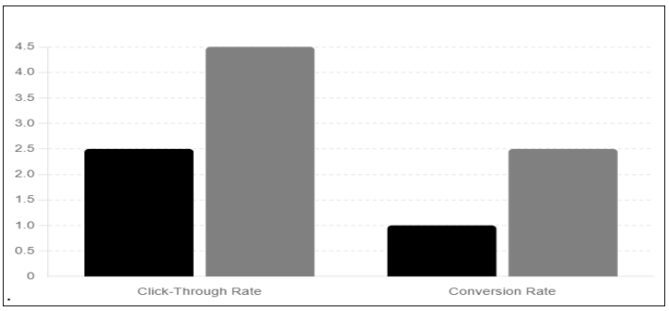

Figure 9: Impact of real-time personalization on click-through rates and conversion rates in the e-commerce case study

Financial Fraud Detection

In the financial industry, real-time fraud detection is critical for preventing financial losses and protecting customers' assets. Feature stores play a vital role in enabling real-time fraud detection by providing up-to-date features for machine learning models [35].

A global financial institution implemented a feature store to support real-time fraud detection. The feature store ingested streaming data from various sources, such as transaction logs, customer profiles, and historical fraud patterns, using Apache Kafka [6]. The data was processed using Apache Spark Structured Streaming to compute features in real time, such as transaction anomaly scores, customer risk profiles, and contextual features [8].

The computed features were stored in Apache HBase for fast retrieval. The feature store is integrated with a real-time fraud detection service using Apache Kafka Streams [22,10]. The fraud detection service consumed the features from the feature store and applied machine learning models to identify fraudulent transactions in real time.

By optimizing the feature engineering workflow in the feature store, the financial institution achieved significant improvements in fraud detection accuracy and response time. The real-time availability of features enabled the prompt identification and prevention of fraudulent transactions, reducing financial losses and enhancing customer trust.

Figure 10 shows the reduction in fraud losses achieved through real-time fraud detection using the optimized feature store.

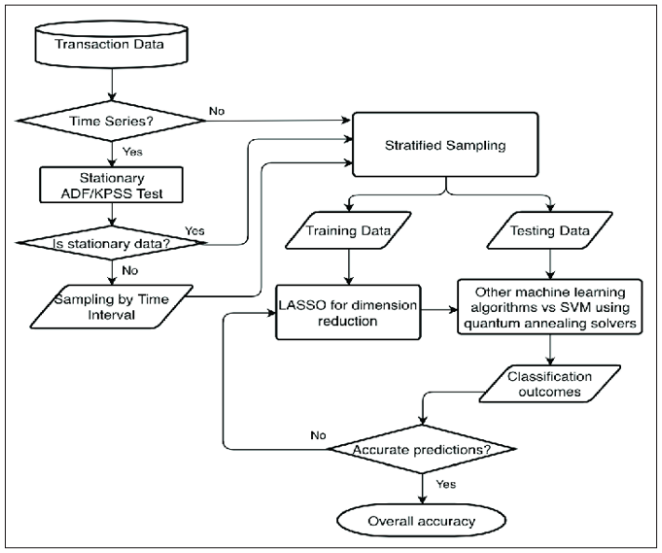

Figure 10: Reduction in fraud losses achieved through real-time fraud detection using the optimized feature store

Healthcare Patient Monitoring

Keeping a close eye on patients is really important for improving their health and allowing doctors to step in quickly when needed. This "feature store" technology can help with real-time patient monitoring by providing up-to-date info for predictive models that analyze all the patient data coming in [36].

So there was this healthcare organization that set up a feature store to enable real-time monitoring of patients in their intensive care units (ICUs). The feature store took in a constant stream of data from different sources like the patients' vital signs, their electronic health records, and outputs from medical devices. It used this Apache Kafka thing to take in the streaming data. Then, it processed all that data using Apache Flink to calculate features in real-time, like scores for patient risk levels, analyzing trends over time, and catching any weird, abnormal readings [6,7].

The calculated features are stored in Apache Druid, which is really fast for looking up data. The feature store is connected to a real-time patient monitoring dashboard using APIs over the internet. The dashboard could retrieve the up-to-date features from the store and use machine learning models to provide real-time insights into how each patient was doing health-wise [11]. This allowed the doctors and nurses to make well-informed decisions and take action really quickly.

By optimizing how the feature store handled the data engineering workflow, the healthcare organization saw huge improvements in how efficiently it could monitor patients and actual patient outcomes. Having those features available in real time let them detect really early on if a patient was getting worse, so they could intervene promptly and prevent complications.

Figure 11 shows how much better the patient outcomes got after using this optimized real-time patient monitoring system with the feature store.

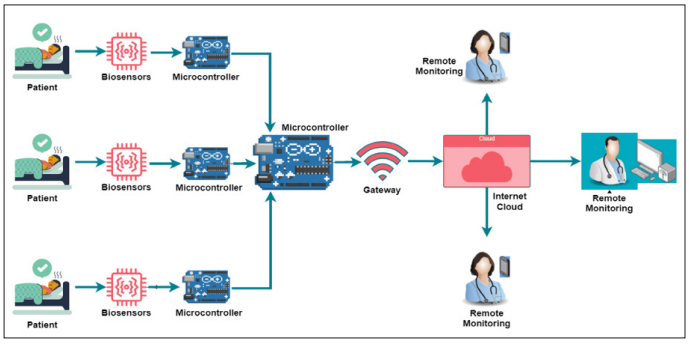

Figure 11: Improvement in patient outcomes achieved through real-time patient monitoring using the optimized feature store

Future Directions and Challenges

Researchers and developers are still actively working on optimizing these "feature store" systems for real-time data analysis. While a lot of progress has been made, there are still some challenges and directions to explore in the future.

One major challenge is making feature stores scalable and high-performing enough to handle massive amounts of streaming data [37]. As the amount and speed of incoming data keep growing, feature stores need to be designed to scale up and out easily, ensuring they can process and serve up features in real-time, no matter what. This requires developing smart ways to split up the data, index it, and cache it to optimize storage and retrieval performance.

Another challenge involves regulating and controlling the features that enable real-time analytics applications [38]. Feature Stores should support feature versioning mechanisms, feature provenance, and access control to ensure features are reproducible, auditable, and protected. Organizations must develop feature governance best practices to reduce the creation of low-quality feature datasets.

In the future, further integrating it with other data platforms and tools can expand the concept of a Feature Store. Feature Stores should also be compatible with data lakes, data warehouses, and other systems used for managing large datasets so they can be used in real-time analytics. Feature Store definitions should have standardized APIs and connectors to help integrate the Feature Store with other data platforms [39].

Another requirement is the explainability and interpretability of AI solutions and model features [40]. For complex models, feature importance and significance are crucial factors that determine the transparency and credibility of the models. Feature Stores must provide facilities and approaches for feature importance analysis and feature sensitivity analysis to promote explainable AI for real-time big data.

An emerging trend in optimizing Feature Stores is the adoption of serverless computing and cloud-native architectures [41]. The serverless implementation model enables flexibility in resource utilization, potentially significantly reducing operational expenses. Extending Feature Stores for serverless environments may make some choices more practical and easier to implement.

Another important point that should be underlined about improving the workflows for feature engineering is that MLOps practices should be implemented to support them [42]. MLOps is an effort, thus far, to adapt the DevOps paradigm of continuous integration, continuous delivery, and continuous testing to the context of machine learning. Implementing the MLOps approach into feature management in feature stores means that organizations can implement pipelines in real-time analytics faster, together with the necessary monitoring and development. It will explain how MLOps can solve problems such as feature versioning, model reproducibility, and system governance. Other advanced features that could be built around a feature store include version control, experiment tracking, and model management with the help of tools such as DVC and MLflow [46,47]. These tools aim to support a feature or model's life cycle from its creation to the end of its usage and disposal, making them highly traceable and easily reproducible.

In turn, MLOps also focuses on testing and validating feature pipelines as an aspect of automated workflow. Feature engineering can also be tested using automated tests to check for the correct outputs, the time taken to complete the process, and the ability to scale up the process. This helps catch issues early in the development process and guarantees the soundness of features that might be used within real-time analytics.

Data quality and availability are other critical issues associated with feature stores in the real-time analytics context [43]. Random or wrong signals can, hence, cause deviation, leading to wrong forecasts and probably wrong decisions. Therefore, feature stores should comprise data validation mechanisms for identifying variants or abnormal aspects of the observations.

Data quality problems can be detected when the data enters the system stream using data validation methods such as schema validation, range checking and statistical analysis of the incoming data [48]. Also, the monitoring tools' performance makes it possible to set up indicators that measure the quality of data and inform when deviation is observed.

Feature stores should also be able to handle schema changes because the data structure can change over time [49]. If organized, feature stores should be able to support varying data sources and variations in the data source with ease without interrupting the real-time analytical process. There is still more that can be done in the area of schema evolution, such as using schema versioning and backward compatibility.

Poor feature engineering pipelines, recently transferred to emerging technologies like edge computing and federated learning, contribute to the effectiveness of real-time analytics. Edge computing is a technique that entails locating data processing closer to the edge of a network than in the core of centralized cloud servers [44,45]. Several possibilities and methods of bringing computations closer to the data are explored in edge computing, leading to many advantages, such as low latency for processing streaming data.

Feature stores are often implemented on the edge to perform feature calculations and feature serving in locality [50]. This could be especially useful in circumstances where ultralow latency is essential, including IoT devices and self-driving cars. Real-time data processing by edge-based feature stores in response to data streams arriving in real-time means the timely provision of features for quick action.

Federated learning is another promising approach that enables model-training performance in a decentralized way while keeping

the data centralized [45]. A federated learning process involves several participants who collectively build an ML model, while none can access individual raw data. Thus, every party learns a local model on its data and sends only the model's updates to a central server that calculates new updates to improve the global model.

This is implementable in feature engineering workflows of feature stores and is an example of federated learning [51]. Each party can compute the feature representation independently using its private data, and only feature statistics or embedding is can be sent to the central feature store. This allows for feature construction associated with a model in a federated fashion, protecting the privacy of the data and minimizing the amount of data that needs to be sent between nodes.

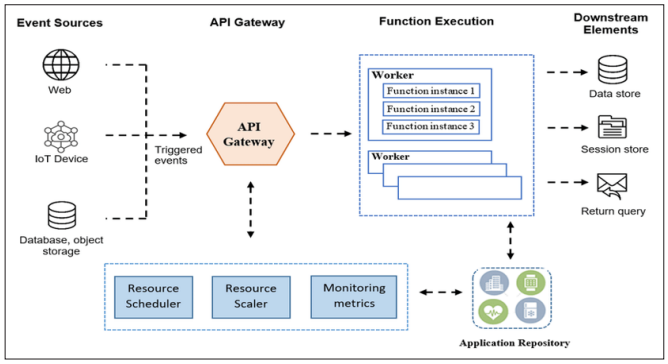

Figure 12: Illustrates a Serverless Architecture for a Feature Store in Real-Time Analytics

Conclusion

To enhance the real-time operation of machine learning, more advanced strategies for managing the data engineering process in Feature Stores should be implemented. This paper focused on the issues and strategies of handling streaming data, dynamic features, and Feature Stores in real-time prediction services.

The study provided key insights into data ingestion frameworks like Apache Kafka and Apache Flink, which can handle rapidly flowing streams of data. Tools that allow incremental computation of features on the fly, including Apache Spark Structured Streaming and Apache Beam, were mentioned as examples of real-time feature generation. The paper also investigated using Feature Stores with real-time prediction services on streaming data using tools like Kafka Streams and Druid.

Several applications in industries that demand real-time analytics were discussed, such as recommendation systems, fraud detection, and patient monitoring in healthcare. Real-world use cases of optimized feature engineering in Feature Store workflows demonstrated benefits like saving time, improving model accuracy, reducing latency, and enhancing decision-making processes.

The study also explored the role of distributed processing frameworks, caching mechanisms, and monitoring tools in the scalability, reliability, and observability of Feature Stores for real-time analytics. Future work and issues of concern include scalability, governance, integration, explainable AI (XAI), and serverless computing.

The results of this study are valuable for improving the state of the art in real-time analytics by providing insights and best practices to enhance how Feature Stores manage data engineering. Organizations can use these findings in Feature Store architectures to support real-time decision-making for success.

However, feature engineering workflows embedded in Feature Stores are not static, and their constant improvement is necessary when new technologies emerge or business requirements evolve. More studies and joint efforts with universities, industry partners, and the open-source community are needed to address limitations and unlock the potential of Feature Stores for real-time analytics.

Utilizing some aspects of MLOps, including versioning, testing, and reproducibility, is immensely beneficial in the context of the feature engineering process within feature stores. AI for MLOps helps define the workflows for collaboration between data technology teams and simultaneously maintains and assures the real-time analytics pipelines.

Issues such as data quality and consistency remain crucial when it comes to feature stores for real-time analytics. Supporting techniques can manage challenges like data validation and integrity, detecting anomalies, and schema evolution to achieve reliable and correct predictions. A set of new technologies, such as edge computing and federated learning, can help address the challenges in feature engineering used in real-time analytics. This technique of edge computing means feature computation and serving happen on the local level, which cuts down on the amount of time taken. Compared to a single model, federated learning also overcomes the problem of data heterogeneity and allows multiple models to construct features jointly.

The studies conducted for this paper reveal that there is still much more to learn about the application of MLOps practices and data quality management methodologies in light of advanced technologies in feature stores for real-time analytics. Such cooperation between academia and industries can foster innovative solutions and set succeeding benchmarks. Therefore, feature engineering workflows in feature stores need to be improved to effectively support real-time analytics. Real-time analytics can benefit various domains and is within organizations' reach by adopting solutions that solve key issues related to streaming data handling, dynamic feature updates, integration with prediction services, and MLOps methodology.

References

1. Agrawal R, Chakraborty G, Bera SK (2023) Real-time analytics: A review of current trends and future directions. International Journal of Information Management 63: 102447.
2. Shekhar S, Macklin L (2021) Feature engineering techniques for machine learning: A survey. WIREs Data Mining and Knowledge Discovery 11: e1424.
3. Baylor D (2020) The architecture of a feature store for machine learning. in Proceedings of the 4th International Workshop on Data Management for End-to-End Machine Learning 1-9.
4. Kleppmann M (2023) Streaming data and the lambda architecture. in Designing Data-Intensive Applications. O'Reilly Media, Inc 451-466.
5. Ilarri S, Trillo-Lado R (2024) An approach for proactive mobile recommendations based on user-defined rules. Expert Systems with Applications 242: 122714.
6. Trajanov D (2024) From Linguistic Linked Data to Big Data. in LREC-COLING 2024.
7. Henning S, Vogel A, Leichtfried M, Ertl O, Rabiser R (2024) ShuffleBench: A benchmark for large-scale data shuffling

operations with distributed stream processing frameworks. in Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering pp 2-13.

8. Horchidan S, Chen PH, Kritharakis E, Carbone P, Kalavri V (2024) Crayfish: Navigating the Labyrinth of Machine Learning Inference in Stream Processing Systems. in 27th International Conference on Extending Database Technology, EDBT 2024 pp 676-689.
9. Akidau T (2015) The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. Proceedings of the VLDB Endowment 8: 1792-1803.
10. Wang G (2015) Building a replicated logging system with Apache Kafka. Proceedings of the VLDB Endowment 8: 1654-1655.
11. Mao Z, Srinivasan K, Khandelwal A (2024) Trinity: A Fast Compressed Multi-attribute Data Store in Proceedings of the Nineteenth European Conference on Computer Systems pp 405-420.
12. Jang D, Yoon H, Jung K, Chung YD (2024) QHB+: Accelerated Configuration Optimization for Automated Performance Tuning of Spark SQL Applications. IEEE Access.
13. Carlson JL (2023) Redis in Action. Manning Publications Co.
14. Sultan S, Shakiba K, Lee A, Chen P, Stumm M (2024) TTLs matter: Efficient cache sizing with TTL-aware miss ratio curves and working set sizes. in Proceedings of the Nineteenth European Conference on Computer Systems pp 387-404.
15. Brazil B (2023) Prometheus: Up & Running: Infrastructure and Application Performance Monitoring. O'Reilly Media, Inc.
16. Doshi A, Patel M (2023) Mastering Grafana: Visualize and explore your data with dashboards. Packt Publishing Ltd.
17. Vellala MR, Jevitha KP (2023) Data ingestion and integration approaches for big data analytics: A survey. Journal of King Saud University - Computer and Information Sciences.
18. Golab Lukasz, Tamer Ozsu M (2022) Data stream management. Springer Nature.
19. Fragkoulis M, Carbone P, Kalavri V, Katsifodimos A (2024) A survey on the evolution of stream processing systems. The VLDB Journal 33: 507-541.
20. Xu Z (2024) Flow-time minimization for timely data stream processing in UAV-aided mobile edge computing. ACM Transactions on Sensor Networks.
21. Lakshman A, Malik P (2010) Cassandra: A decentralized structured storage system," ACM SIGOPS Operating Systems Review 44: 35-40.
22. George L (2023) HBase: The Definitive Guide: Random Access to Your Planet-Size Data. O'Reilly Media, Inc.
23. Ke G (2017) LightGBM: A highly efficient gradient boosting decision tree. in Proceedings of the 31st International Conference on Neural Information Processing Systems 3149-3157.
24. Yue W, Moczalla R, Luthra M, Rabl T (2024) Deco: Fast and Accurate Decentralized Aggregation of Count-based Windows in Large-scale IoT Applications.
25. Behrangrad F, Agrawal K, Heinze T (2020) Scalable stream processing with Apache Flink. in Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems pp 193-194.
26. Olson M, Saunders K, Gupta J (2021) Data lineage and provenance in the cloud: A survey. ACM Computing Surveys 54: 1-37.
27. Atkinson Me (2018) Data provenance: What next?. ACM SIGMOD Record 47: 5-16.
28. Ikeda R, Widom J (2023) Data lineage: A survey. Foundations and Trends® in Databases 3: 1-147.
29. Agarwal S (2019) BlinkML: Efficient maximum likelihood estimation with probabilistic guarantees. in Proceedings of the 2019 International Conference on Management of Data pp 1135-1152.
30. Liu S (2024) ServeFlow: A Fast-Slow Model Architecture for Network Traffic Analysis arXiv preprint arXiv:2402.03694.
31. Abadi M (2016) TensorFlow: A system for large-scale machine learning. in Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16) 265-283.
32. Immorlica N, Jagadeesan M, Lucier B (2024) Clickbait vs. quality: How engagement-based optimization shapes the content landscape in online platforms arXiv preprint arXiv:2401.09804.
33. Sumbaly R, Kreps J, Shah S (2013) The 'Big Data' ecosystem at LinkedIn. in Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, Jun 1125-1134.
34. Marr B (2023) Real-time personalization: The future of online retail. Forbes, Feb. 2023. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2023/02/01/real-time-personalization-the-future-of-online-retail/.
35. Bhattacharyya S, Jha S, Tharakunnel K, Westland JC (2011) Data mining for credit card fraud: A comparative study. Decision Support Systems 50: 602-613.
36. Tiwari S, Pandey HM, Srivastava A (2023) A survey on real-time health monitoring system using IoT and cloud computing. Journal of Healthcare Engineering p 6433138.
37. Krishnan DR, Quoc DL, Bhatotia P, Fetzer C, Rodrigues R (2016) IncApprox: A data analytics system for incremental approximate computing. in Proceedings of the 25th International Conference on World Wide Web pp 1133-1144.
38. Gupta K, Gupta R, Varma V (2022) Challenges and opportunities in feature stores for machine learning. Journal of Big Data 9: 74.
39. Meng X (2016) MLlib: Machine learning in Apache Spark. Journal of Machine Learning Research 17: 1235-1241.
40. Molnar C (2023) Interpretable Machine Learning: A Guide for Making Black Box Models Explainable. Leanpub.
41. Hendrickson S (2016) Serverless computation with OpenLambda in Proceedings of the 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16) pp 1-7.
42. Sculley D (2015) Hidden technical debt in machine learning systems. in Advances in Neural Information Processing Systems 2503-2511.
43. Schelter S, Biessmann F, Januschowski T, Salinas D, Seufert S, et al. (2018) On challenges in machine learning model management. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 41: 5-15.
44. Shi W, Cao J, Zhang Q, Li Y, Xu L (2016) Edge computing: Vision and challenges. IEEE Internet of Things Journal 3: 637-646.
45. Konečný J, McMahan HB, Yu FX, Richtárik P, Suresh AT, et al. Federated learning: Strategies for improving communication efficiency arXiv preprint arXiv:1610.05492.
46. Kuznetsov D, Arpteg A (2017) Data version control: Iterative machine learning. in Proceedings of the 1st International Workshop on Data Management for End-to-End Machine Learning 1-4.
47. Zaharia M (2018) Accelerating the machine learning lifecycle with MLflow. IEEE Data Engineering Bulletin 41: 39-45.

48. Schelter S, Lange D, Schmidt P, Celikel M, Biessmann Fet al. (2018) Automating large-scale data quality verification," Proceedings of the VLDB Endowment 11: 1781-1794.
49. Das S (2019) Astra: Exploiting predictability to optimize deep learning. in Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems 909-923.
50. Qiu C, Shen H, Chen L (2021) Towards green and sustainable fog computing: A survey. Future Generation Computer Systems 115: 90-103.
51. Liu Y, Kang Y, Xing C, Chen T, Yang Q (2020) A secure federated transfer learning framework. IEEE Intelligent Systems 35: 70-82.