Real-time Event Joining in Practice With Kafka and Flink

Srijan Saket*
srijanskt@gmail.com
ShareChat
Seattle, USA

Vivek Chandela*
vivekchandela@sharechat.co
ShareChat
Bangalore, India

Md. Danish Kalim
danish@sharechat.co
ShareChat
Bangalore, India

ABSTRACT

Historically, machine learning training pipelines have predominantly relied on batch training models, retraining models every few hours. However, industrial practitioners have proved that real-time training can lead to a more adaptive and personalized user experience. The transition from batch to real-time is full of tradeoffs to get the benefits of accuracy and freshness while keeping the costs low and having a predictable, maintainable system.

Our work characterizes migrating to a streaming pipeline for a machine learning model using Apache Kafka and Flink. We demonstrate how to transition from Google Pub/Sub to Kafka to handle incoming real-time events and leverage Flink for streaming joins using RocksDB and checkpointing. We also address challenges such as managing causal dependencies between events, balancing event time versus processing time, and ensuring exactly-once versus at-least-once delivery guarantees, among other issues. Furthermore, we showcase how we improved scalability by using topic partitioning in Kafka, reduced event throughput by **85%** through the use of Avro schema and compression, decreased costs by **40%**, and implemented a separate pipeline to ensure data correctness. Our findings provide valuable insights into the tradeoffs and complexities of real-time systems, enabling better-informed decisions tailored to specific requirements for building effective streaming systems that enhance user satisfaction.

CCS CONCEPTS

• **Computer systems organization** → *Data flow architectures.*

KEYWORDS

event streaming; system design; cost optimisation; short video

ACM Reference Format:
Srijan Saket*, Vivek Chandela*, and Md. Danish Kalim. 2024. Real-time Event Joining in Practice With Kafka and Flink. In *Proceedings of the 4th International Workshop on Online and Adaptive Recommender Systems (OARS 2024), Held in conjunction with CIKM-2024, October 25, 2024, Boise, ID, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3627673.3679083

OARS 2024, October 25, 2024, Boise, ID, USA

ACM ISBN 979-8-4007-0436-9/24/10…$15.00
https://doi.org/10.1145/3627673.3679083

1 INTRODUCTION

In the modern information technology era, users generate large quantities of data. Platforms collect and process this amassed data to derive meaningful insights that can enhance their system. Social media platforms, in particular, generate enormous volumes of data from various user activities [18, 20, 24]. Short-form video (SFV) platforms are especially prominent, offering an immersive viewing experience that attracts substantial user attention. Due to the nature of their offerings, SFV platforms cater to the passive preferences of users, which are inherently dynamic and frequently changing. The quality of users' lean-back experience (i.e., users passively consume information) relies heavily on the system's ability to capture user feedback and adapt recommendations in real-time using user feedback.

Stream processing is a well-known and extensively studied field. Work in this domain has focused on

(1) Utilising real-time processing framework like Flink to prepare data from machine learning systems [14, 16, 22, 23, 27].
(2) Employing real-time framework for real-time model updates such as Monolith [21], Solma [10] and others [1, 7, 13].

This paper aims to use a state-of-the-art stream processing framework to implement practical and industry-scale real-time data systems that will enable real-time model building. We will discuss the challenges in building these systems and strategies to overcome them. Key challenges in building industry-scale real-time data systems include:

(1) Stream Integration: When we deal with data streams containing features and labels, using a connector to merge them into machine learning systems is essential [4, 6]. Sometimes, combining labels from different streams into a single stream for multi-task modeling setups is necessary [8, 19].
(2) Out-of-order Events: Managing delays in data streams generated by events is vital in system configurations. The system should be able to process events regardless of their timing to ensure the accuracy of training data. For instance, creating training data where a user has viewed and liked an item but has not shared it would require assigning a value of 1 to the like label and 0 to the share label.
(3) Load Handling: User engagement with content on the platform over time drives the model's load. User behaviors can change, leading to fluctuations like increased usage in the evening compared to during the day, which highlights the importance of efficiently managing back pressure [9].
(4) Concurrent Updates: With the increased volume of data, updates can occur concurrently for a critical element, such

*Equal Contribution

as users or items. The system must address conflicts arising from these simultaneous updates to maintain training stability.

Considering all the above challenges, we built two systems. Both approaches differ in the choice of components, complexity, and maintenance:

(1) **Approach 1**: In the setup described in Fig 1a, PubSub [3] is used as the messaging queue. Order of events is preserved (in a time window) by using Memcache [5], also used for real-time label joining and preparing training samples for downstream training jobs. Further, Redis is used for distributed locking to avoid concurrent updates on the same ID.
(2) **Approach 2**: The setup in Figure 1b uses Kafka [15] as the messaging queue, with its benefits discussed in subsequent sections. By integrating Apache Flink with Kafka, we maintain event order and prevent concurrent updates. Although this approach involves fewer components than the initial setup, we will demonstrate why it is superior through comparison.

2 METHODOLOGY

2.1 Data Context & User Signals

The study was conducted on ShareChat's data, which is a multilingual social media platform that delivers content in over 18 languages, with a user base exceeding 180 million monthly active users. The application generates two broad categories of events: views and engagements. The system generates view events whenever it shows content to a user. These are high-volume events with a peak throughput of *>100 MB/sec*. Conversely, engagements are user signals comprised of implicit signals (such as video play and skip) and explicit signals (including click, like, and share). However, the event processing pipeline does not distinguish between them. The volume of these events is significantly lower compared to views. We used Field-aware Factorization Machines (FFM) [11, 12] to learn 32-dimensional embedding for each signal. Real-time training retrieves the previous state of these embeddings from a NoSQL database and updates them using real-time user signals from messaging queues.

2.2 Generating Training Samples

The system generates real-time training samples by joining a view event with its corresponding engagement label, if available [25, 29]. In this use case, it considers a view with an engagement event as a positive label and a view without engagement as a negative label. For example, if there is a like event corresponding to a view but no share, comment, or favorite, the label for the like will be 1 and 0 for the others. These events are received through different PubSub topics, as we have separate models to predict embeddings for each user signal, as illustrated in Figure 1a.

2.3 Key Challenges

2.3.1 Handling out-of-order events. The sequence of events is crucial for making meaningful updates; otherwise, we risk incorporating incorrect information in the future. Since the system processes views and engagements in separate queues, we cannot determine whether a view precedes engagements or vice versa [2a]. One approach is to delay view events by a few minutes by holding them in Pub/Sub without acknowledging receipt while temporarily storing engagements in a distributed key-value store like Memcache, with a TTL longer than the delay. However, delaying view events in Pub/Sub by withholding acknowledgment is inefficient.

2.3.2 Pod Contention. If two pods update the same user embedding concurrently with different training samples, it can lead to incorrect outcomes due to the lack of order preservation. To prevent this, we need to lock the user ID during updates as described in [2b]. Acquiring a distributed lock on the user ID via Redis prevents concurrent updates, thereby resolving the issue. However, this adds an extra component to the system, increasing costs.

2.3.3 Inflating Queue Size. Pub/Sub lacks extensive support for data retention, message replay, and message ordering compared to log-based queues. Once a message is acked in Pub/Sub, it is gone forever. So, the only way to delay view events is by unacking them, which inflates the queue size [2c] and our cloud bills.

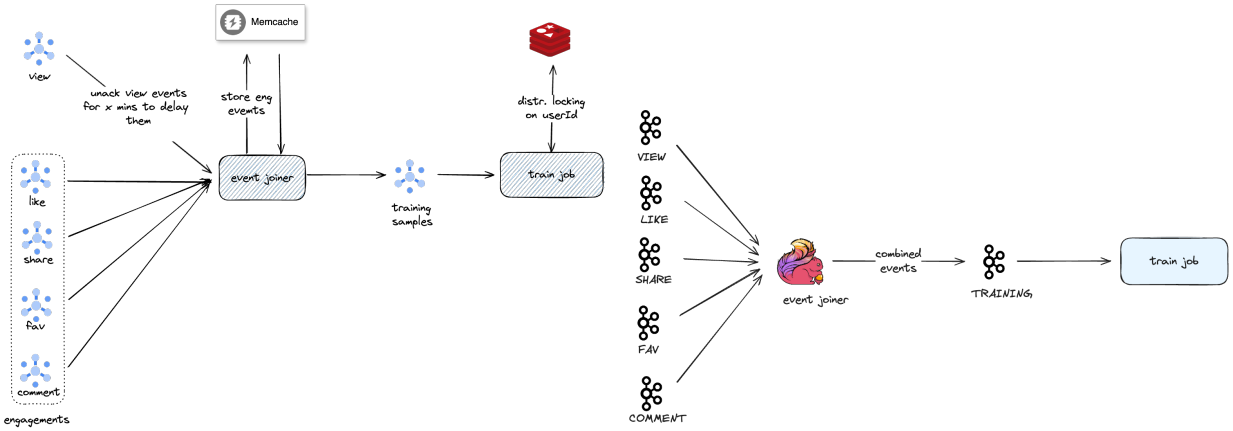
2.4 Making "Better" System Choices

Having established the problem statement in the previous sections, let us explore how we can make choices to create a more reliable system with the same outcomes while keeping costs under control.

2.4.1 Replace PubSub with Kafka. A good starting point is replacing Google PubSub with Apache Kafka. Kafka is a log-based event-streaming platform that provides message ordering, the ability to replay messages, and longer data retention. Apache Flink has excellent support for Kafka. Also, self-hosted Kafka proved cheaper than GCP's managed PubSub offering.

2.4.2 Use Flink. We then rely on Apache Flink [14], an open-source stream processing framework, as the real-time event joiner. Its rich feature set includes internal state management, keyed streams, timers, and many more. Leveraging Flink's support for the state backend, particularly RocksDB, allowed us to eliminate the need for Memcache. Some beneficial properties of Flink in the proposed solution are:

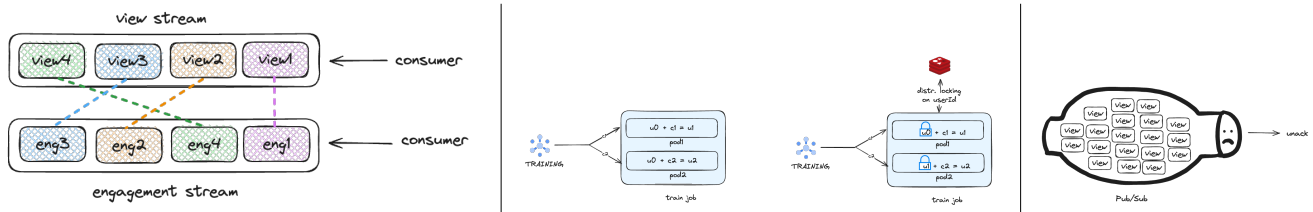
(1) Graceful Backpressure Handling: Backpressure refers to a system receiving data faster than it can process, often during a temporary load spike. Flink handles backpressure gracefully without any sophisticated mechanism.
(2) State: Flink enables operators to retain information across multiple events using state. Flink provides various state primitives like ValueState for single values, MapState for key-value pairs, and many more. A TTL can be assigned to any state as needed.
(3) Low-level APIs: Flink supports both high-level and low-level APIs. We chose low-level APIs like `KeyedProcessFunction` and `KeyedCoProcessFunction` for joining view and engagement events, as they give more control over each event.
(4) Checkpointing: The central part of Flink's fault tolerance mechanism is drawing consistent snapshots of all the states in timers and stateful operators. The system can fall back to the latest snapshot in case of failure.



(a) System to join incoming real-time events with labels to prepare training samples; using caching components while maintaining the order of events

(b) Modified system to join incoming real-time events with labels to prepare training samples; using Kafka and Flink overcoming the shortcomings of previous approach

Figure 1: System comparison of Approach 1 vs. Approach 2



(a) Out-of-order events

(b) Concurrent updates on an ID

(c) Inflating queue size

Figure 2: Key Challenges in Event Streaming Pipelines

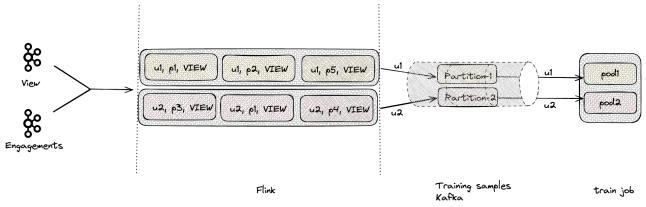


Figure 3: How pod contention can be solved using intrinsic properties of Kafka and Flink

2.4.3 Remove Redis. Finally, we can eliminate Redis using Keyed Streams in Flink and Topic partitioning in Kafka. In the example shown in Fig 3, Flink processes events for users *u*1 and *u2* across distinct keyed streams. Flink directs these streams to separate partitions in Kafka, and the pods of the consumer job subsequently consume them. The crucial insight here lies in pre-allocating the partitions in Kafka and the pods in the training job. Without this pre-allocation, events for the same user could be assigned to different partitions during rebalancing, affecting the training. As seen, *partition1* and *pod1* will process all interactions by *user1* while *partition2* and *pod2* will process all interactions by *user2*.

2.4.4 Other Optimisations. Data compression is an effective strategy to reduce throughput [2, 26]. We further optimized our pipeline by transitioning from JSON to Avro schema and employing **LZ4** compression, resulting in an 85% reduction in the throughput.

2.5 Analysing the Trade-Offs

2.5.1 At-least Once Delivery. While exactly-once delivery is ideal for accuracy, it is complex to implement, and the additional complexity does not offset the gains. We implement a de-duplication layer in the training job, effectively making it an idempotent sink. So, at-least-once delivery in addition to an idempotent sink is a much more practical choice for us.

2.5.2 Event & Processing Time. We assess the tradeoffs between utilizing event time and processing time. Event time is when an event occurred, while processing time is when Flink starts processing the event. Their lag can be significant due to network delays and asynchronous environments, such as data transmission via message queues. The example in Fig 4 shows the tradeoffs between event and processing time. Event time leads to more accurate results, but implementing it is more challenging, and we must deal with "late events".

2.5.3 Using Watermarks. Let us say we have tumbling windows of size 1 min each. In Fig 4, window 1 misses event 2, and window 4

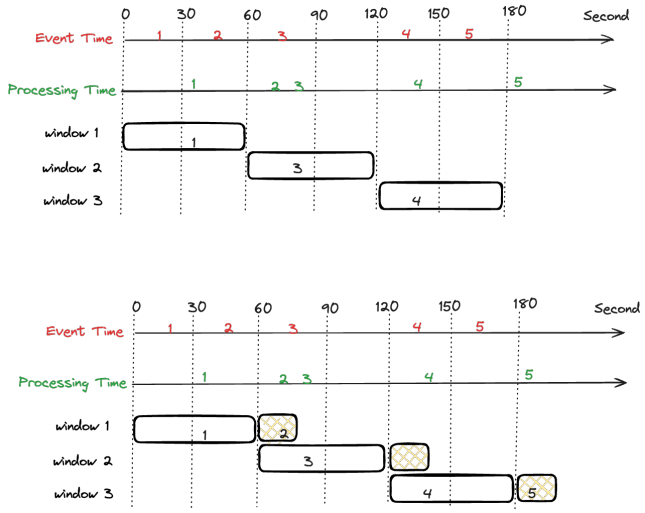


Figure 4: Demonstration of watermark in event processing.

misses event 5. So, how do we solve this? We can use a watermark [17, 28] to extend our windows by an additional 15 sec. However, watermarks only assist with slightly delayed events, not those with significant delays. It is a tradeoff between latency and accuracy: extending the window boosts accuracy by capturing late events but adds latency t to the system, and vice versa.

3 RESULTS & OBSERVATIONS

Components	Approach 1	Approach 2	Relative Cost Savings (2 vs 1)
Messaging System	PubSub	Kafka	**55%**
Event Joiner	Golang Job + Memcache	Flink	**52%**
Redis	✓	-	**100%**
Schema	Json	Avro	**85%**
Compression	-	LZ4	

Table 1: Comparison of Approaches and Cost Savings

3.1 Cost Comparison

We compare the cost of the two systems described in Fig 1 by breaking it at the component level.

3.1.1 PubSub vs. Kafka. We compare the cost difference between two setups at a peak throughput of *200 MBPS*, with a message retention period of 3 days for both, translating to around 10TiB of data daily. The Pub/Sub configuration uses a single topic with a single subscription, while the Kafka setup includes seven brokers. Currently, the Kafka setup is regional, but switching to a zonal setup could reduce costs by approximately 25% due to lower inter-zone egress charges. However, this change would also result in reduced availability. For a similar setup, we observe the cost of Kafka as **55%** lesser than PubSub. We can extrapolate the numbers linearly as there are five such topics, adjusting the throughput.

3.1.2 Flink vs. Consumer Job. The event joiner in Fig 1a is a job written in Golang that consumes the events and joins them with real-time labels via Memcache. The Flink consumer does the same task in Fig 1b. A GCS bucket is required to store checkpoints for the Flink job. The combined cost of Flink is **52%** lesser than the Golang job.

3.1.3 Redis and Memcache. The cost of Memcache goes away with Flink. As described in Section 2.4.3, Flink solves pod contention using intrinsic properties. So, with Flink in place, Redis's cost also goes away.

3.1.4 Data Compression. As mentioned in 2.4.4, using Avro schema and LZ4 compression resulted in an 85% reduction in throughput, leading to similar savings in data ingestion cost.

3.2 Performance or Latency

Both the systems can be scaled horizontally to adjust according to the incoming traffic. The consumers in Approach 1 scale according to the number of unacked messages or oldest unacked message age in PubSub, along with CPU & memory utilization. Flink also allows dynamic adjustment of resources based on workload, helping to optimize performance and cost. However, it requires re-partitioning in the Kafka topic associated with the job, the implementation of which is slightly complex. Currently, we provision Flink to handle maximum traffic and are exploring autoscaling as future work. Both setups ensure comparable performance after tuning, resulting in similar latency.

3.3 Data Validation

The switch from Approach 1 to Approach 2 was verified by setting up a system where the same events were duplicated and processed through both pipelines. Initially, we directed 0.1% of the traffic through each pipeline to separate storage tables and compared the tables for accuracy. After confirming correctness (match rate & schema), we gradually increased the traffic to 100% before transitioning to production. Finally, we decommissioned the old pipeline.

4 CONCLUSION & FUTURE WORK

This paper explores the implications of design choices for processing streaming events to generate training samples for machine learning models. We address key challenges such as real-time label joining, handling out-of-order events, and managing concurrent updates. By comparing trade-offs, we illustrate how to make informed design decisions for specific use cases. We further see how we can leverage the intrinsic properties of frameworks and platforms to simplify the system and make it more cost-effective. A one-line takeaway is that our decisions must consider cost, correctness, latency, maintainability, and the trade-offs we are prepared to accept.

The subsequent steps for this project involve implementing autoscaling for the Flink job. We also plan to minimize the source topic count by consolidating them under one topic, which will enhance system management and help further decrease costs.

5 ACKNOWLEDGEMENTS

We sincerely thank Arya Ketan for his guidance on design choices and Shubham Dhal for his dedicated assistance with the implementation. Their contributions greatly improved the quality of this project.

REFERENCES

[1] Vibhatha Abeykoon, Supun Kamburugamuve, Kannan Govindrarajan, Pulasthi Wickramasinghe, Chathura Widanage, Niranda Perera, Ahmet Uyar, Gurhan Gunduz, Selahattin Akkas, and Gregor Von Laszewski. 2019. Streaming machine learning algorithms with big data systems. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 5661–5666.

[2] Mohammed Al-Laham and Ibrahiem MM El Emary. 2007. Comparative study between various algorithms of data compression techniques. *IJCSNS* 7, 4 (2007), 281.

[3] Tania Banerjee and Sartaj Sahni. 2015. Pubsub: An Efficient Publish/Subscribe System. *IEEE Trans. Comput.* 64, 4 (2015), 1119–1132. https://doi.org/10.1109/TC.2014.2315636

[4] Albert Bifet, Ricard Gavalda, Geoffrey Holmes, and Bernhard Pfahringer. 2023. *Machine learning for data streams: with practical examples in MOA.* MIT press.

[5] Brad Fitzpatrick. 2004. Distributed caching with memcached. *Linux J.* 2004, 124 (aug 2004), 5.

[6] Heitor Murilo Gomes, Jesse Read, Albert Bifet, Jean Paul Barddal, and João Gama. 2019. Machine learning for streaming data: state of the art, challenges, and opportunities. *ACM SIGKDD Explorations Newsletter* 21, 2 (2019), 6–22.

[7] Jinlin Guo, Haoran Wang, Xinwei Li, and Li Zhang. 2021. Stream classification algorithm based on decision tree. *Mobile Information Systems* 2021, 1 (2021), 3103053.

[8] Jiawei Han, Jian Pei, and Hanghang Tong. 2022. *Data mining: concepts and techniques.* Morgan kaufmann.

[9] Muhammad Hanif, Hyeongdeok Yoon, and Choonhwa Lee. 2020. A Backpressure Mitigation Scheme in Distributed Stream Processing Engines. In *2020 International Conference on Information Networking (ICOIN)*. 713–716. https://doi.org/10.1109/ICOIN48656.2020.9016513

[10] Waqas Jamil, NC Duong, W Wang, Chemseddine Mansouri, Saad Mohamad, and Abdelhamid Bouchachia. 2018. Scalable online learning for flink: SOLMA library. In *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*. 1–4.

[11] Yuchin Juan, Damien Lefortier, and Olivier Chapelle. 2017. Field-aware factorization machines in a real-world online advertising system. In *Proceedings of the 26th International Conference on World Wide Web Companion*. 680–688.

[12] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. 2016. Field-aware factorization machines for CTR prediction. In *Proceedings of the 10th ACM conference on recommender systems*. 43–50.

[13] Supun Kamburugamuve, Pulasthi Wickramasinghe, Saliya Ekanayake, and Geoffrey C Fox. 2018. Anatomy of machine learning algorithm implementations in MPI, Spark, and Flink. *The International Journal of High Performance Computing Applications* 32, 1 (2018), 61–73.

[14] Asterios Katsifodimos and Sebastian Schelter. 2016. Apache Flink: Stream Analytics at Scale. In *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*. 193–193. https://doi.org/10.1109/IC2EW.2016.56

[15] Jay Kreps. 2011. Kafka : a Distributed Messaging System for Log Processing. https://api.semanticscholar.org/CorpusID:18534081

[16] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M Patel, Karthik Ramasamy, and Siddarth Taneja. 2015. Twitter heron: Stream processing at scale. In *Proceedings of the 2015 ACM SIGMOD international conference on Management of data*. 239–250.

[17] S. Lam and A. Xu. 2022. *System Design Interview - An Insider's Guide: Volume 2.* Number v. 2. Amazon Digital Services LLC - Kdp. https://books.google.com/books?id=1Sr7zgEACAAJ

[18] Dingcheng Li, Xu Li, Jun Wang, and Ping Li. 2020. Video Recommendation with Multi-gate Mixture of Experts Soft Actor Critic. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, China) *(SIGIR '20)*. Association for Computing Machinery, New York, NY, USA, 1553–1556. https://doi.org/10.1145/3397271.3401238

[19] Qian Lin, Beng Chin Ooi, Zhengkui Wang, and Cui Yu. 2015. Scalable distributed stream join processing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 811–825.

[20] Qingyun Liu, Zhe Zhao, Liang Liu, Zhen Zhang, Junjie Shan, Yuening Li, Shuchao Bi, Lichan Hong, and Ed H. Chi. 2023. Multitask Ranking System for Immersive Feed and No More Clicks: A Case Study of Short-Form Video Recommendation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management* (Birmingham, United Kingdom) *(CIKM '23)*. Association for Computing Machinery, New York, NY, USA, 4709–4716. https://doi.org/10.1145/3583780.3615489

[21] Zhuoran Liu, Leqi Zou, Xuan Zou, Caihua Wang, Biao Zhang, Da Tang, Bolin Zhu, Yijie Zhu, Peng Wu, Ke Wang, and Youlong Cheng. 2022. Monolith: Real Time Recommendation System With Collisionless Embedding Table. arXiv:2209.07663 [cs.IR] https://arxiv.org/abs/2209.07663

[22] Ramesh Marpu and Bairam Manjula. 2024. Streaming machine learning algorithms with streaming big data systems. *Brazilian Journal of Development* (2024). https://api.semanticscholar.org/CorpusID:266830686

[23] Shadi A Noghabi, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, Indranil Gupta, and Roy H Campbell. 2017. Samza: stateful scalable stream processing at LinkedIn. *Proceedings of the VLDB Endowment* 10, 12 (2017), 1634–1645.

[24] Srijan Saket, Olivier Jeunen, and Md. Danish Kalim. 2024. Monitoring the Evolution of Behavioural Embeddings in Social Media Recommendation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Washington DC, USA) *(SIGIR '24)*. Association for Computing Machinery, New York, NY, USA, 2935–2939. https://doi.org/10.1145/3626772.3661368

[25] Srijan Saket, Sai Baba Reddy Velugoti, and Rishabh Mehrotra. 2023. Formulating Video Watch Success Signals for Recommendations on Short Video Platforms.. In *LERI@ RecSys*. 41–48.

[26] Khalid Sayood. 2017. *Introduction to data compression.* Morgan Kaufmann.

[27] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. 2014. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 147–156.

[28] Tawfik Yasser, Tamer Arafa, Mohamed El-Helw, and Ahmed Awad. 2023. Keyed Watermarks: A Fine-grained Tracking of Event-time in Apache Flink. In *2023 5th Novel Intelligent and Leading Emerging Sciences Conference (NILES)*. 23–28. https://doi.org/10.1109/NILES59815.2023.10296717

[29] Yang Zhang, Yimeng Bai, Jianxin Chang, Xiaoxue Zang, Song Lu, Jing Lu, Fuli Feng, Yanan Niu, and Yang Song. 2023. Leveraging watch-time feedback for short-video recommendations: A causal labeling framework. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 4952–4959.