Competency module of shift-left architecture in big data

Victoria Vysotska[1, †], Iryna Kyrychenko[2, †], Vadym Demchuk[3, †] and Nadiia Babkova[4, †]

[1] *Lviv Polytechnic National University, Stepan Bandera street 12, Lviv, 79013, Ukraine*
[2,3] *Kharkiv National University of Radio Electronics, Nauky Ave. 14, Kharkiv, 61166, Ukraine*
[4] *National Technical University «Kharkiv Polytechnic Institute», 2, Kyrpychova str., Kharkiv, 61002, Ukraine*

Abstract
This research article explores the integration of a Competency module within the Shift-Left Architecture framework tailored for Big Data streaming systems. This study demonstrates that the proposed module significantly enhances maintainability, enables early issue detection, and improves risk assessment. Leveraging automation, real-time monitoring, and proactive validation, the Competency module streamlines configuration management, optimises streaming pipelines and accelerates processing efficiency. Key findings reveal its capability to validate source configurations proactively, reconfigure processing engines, and propose enhancements, reducing manual intervention and minimising downtime. This research contributes a robust framework that strengthens the efficiency, reliability, and scalability of Big Data streaming architectures, offering valuable insights for implementing Shift-Left principles effectively.

Keywords
Shift-Left Architecture, Big Data Streaming, Early Issue Detection, Real-Time Monitoring, Processing Optimization, Performance Tuning, Parameter Tuning, Machine Learning, Real-time Configuration Adaptation, Stream Processing, Automatic Parameter Tuning, System Optimization, Infrastructure Cost Savings, Data Pipeline Management

1. Introduction

The evolution of data processing architectures can be categorised into distinct generations, each addressing the limitations of its predecessor.

The Extract-Transform-Load (ETL) model's initial stage entails pulling raw data from on-premises databases, processing it with limited storage and computational power, and storing the refined data in a data warehouse. Although this method was suitable for its era, it faces notable drawbacks, such as limited processing power, inadequate scalability, and challenges in efficiently storing and analysing past data [1].

In response to these issues, a second-generation architecture emerged, defined by the Extract-Load-Transform (ELT) model. This method allows raw data to be quickly fed into a scalable, cost-efficient Data Lake, utilising a cloud-based, multi-tier medallion framework. Data transformation is carried out with highly parallelised, cloud-optimised computing resources, offering exceptional scalability and performance. The resulting data is channelled to downstream business applications, supporting more robust analytics and profound insights [2].

Fig. 1 represents the diagram of an ETL and ELT data pipeline.

Despite its advantages, the ELT paradigm comes with disadvantages that can prevent efficiency and data quality [3]:

PhD Workshop on Artificial Intelligence in Computer Science at 9th International Conference on Computational Linguistics and Intelligent Systems (CoLInS-2025), May 15–16, 2025, Kharkiv, Ukraine
* Corresponding author.
† These authors contributed equally.
Victoria.A.Vysotska@lpnu.ua (V. Vysotska); iryna.kyrychenko@nure.ua (I. Kyrychenko); vadym.demchuk@nure.ua (V. Demchuk); Nadjenna@gmail.com (N. Babkova)
0000-0001-6417-3689 (V. Vysotska); 0000-0002-7686-6439 (I. Kyrychenko); 0000-0003-3700-2344 (V. Demchuk); 0000-0002-2200-7794 (N. Babkova)

- Processing data in batches often leads to delays and inconsistencies. In micro-batch processing, extra steps are needed to align and integrate the data with its current state, adding complexity to the workflow.
- Different teams — such as AI, Data Platform, and Marketing — frequently create independent pipelines for the same data sources to support their respective systems. This redundancy wastes resources and drives up operational costs.
- Without comprehensive documentation, "similar-but-slightly-different" pipelines multiply, causing inefficiencies and making maintenance more challenging.
- Reverse ETL, widely used in contemporary setups like data warehouses, Delta Lakes, or Data Lakehouses, adds further redundancy and data duplication, amplifying processing inefficiencies.

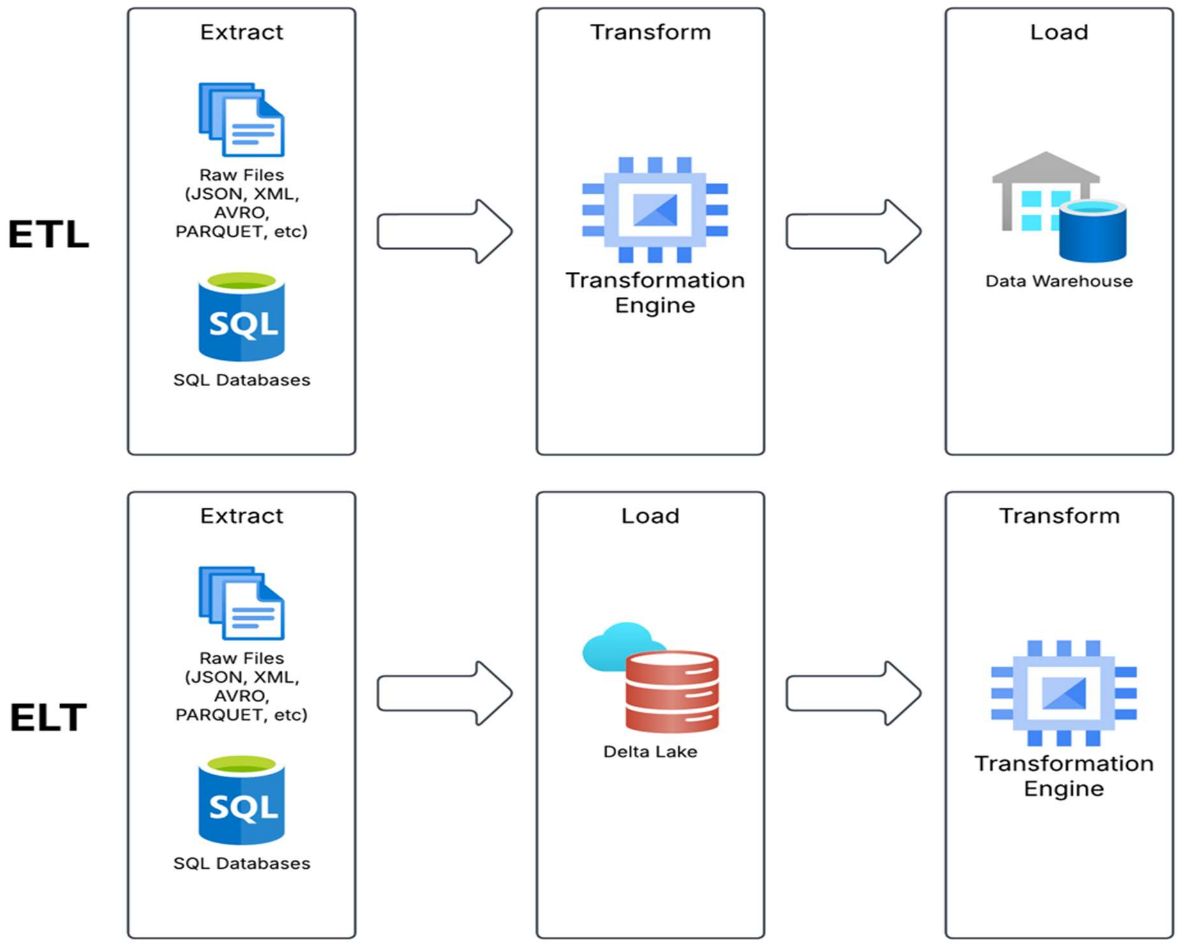

Figure 1: An architecture of ETL/ELT flow

Data has become the most valuable asset in today's business landscape. To thrive in a highly competitive environment, organisations require enriched, trustworthy, and contextualised data delivered in near real-time to drive decision-making and innovation.

The Shift-Left Architecture was developed as an innovative solution to tackle these challenges. In the context of Big Data streaming, Shift-Left Architecture is a design pattern that repositions data processing and governance nearer to the point of data origin, drawing inspiration from Shift-Left Testing in software engineering. This approach prioritises real-time data processing as it is generated, leveraging tools like Apache Kafka and Apache Flink to create immediate data products for Data Warehouses (e.g., Snowflake), Data Lakes, and Data Lakehouses (e.g., Databricks). By focusing on enriching, transforming, and constructing data products accurately at the source, Shift-Left adheres to the "build once, use many times" philosophy. It ensures that data products are efficiently prepared and seamlessly reused downstream, minimising redundancy while enhancing data quality across the enterprise [4].

The Shift-Left Architecture (see Fig. 2) reimagines Big Data workflows by shifting data processing and governance closer to the source. Early data cleaning, aggregation, and enrichment guarantee that downstream systems — whether analytical platforms like Data Lakes, Data Warehouses, and Data Lakehouses or operational systems like microservices—receive well-structured, high-quality data. It reduces repetitive processing, shortens time to value, and maximises business impact from the outset [5].

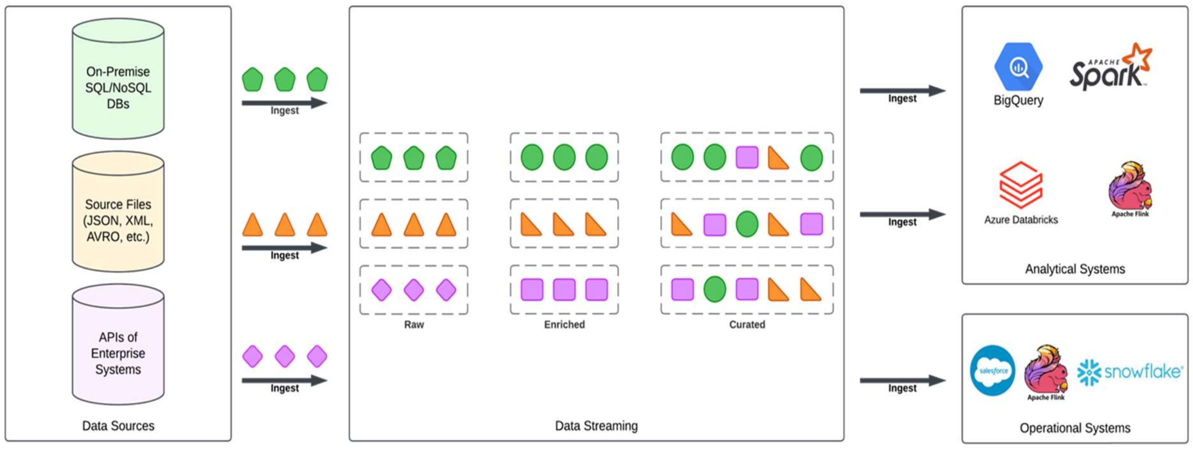

Figure 2: The Shift-Left Architecture

A standout feature of Shift-Left is its alignment with the data mesh paradigm, enabled by real-time data products. Integrating transactional and analytical workloads using technologies such as Apache Kafka, Apache Flink, and Apache Iceberg delivers consistent, high-quality data across the organisation. Streaming data can be processed instantly or fed into modern analytics platforms like Snowflake, Databricks, or Google BigQuery, creating a cohesive foundation for AI and analytics initiatives [4, 5].

Building on these strengths, Shift-Left Architecture emphasises the critical role of early issue detection and risk management. By spotting potential problems at the data source, organisations can address risks before they cascade into downstream systems, avoiding expensive rework and operational setbacks. A specialised Competency Module could be embedded within the Shift-Left framework to strengthen this proactive stance, focusing on issue resolution and risk evaluation. This module would arm data engineers and analysts with the expertise and tools to detect, assess, and fix data quality issues, schema discrepancies, and security risks at the source. Embedding this capability early in the data lifecycle would bolster the resilience and effectiveness of the data ecosystem, amplifying the transformative impact of the Shift-Left approach.

2. Related Works

A significant body of prior research has explored the ETL and ELT methodologies. These investigations examined ETL and ELT workflows, identifying obstacles to handling vast amounts of real-time data, with efforts aimed at enhancing efficiency, minimising delays, and guaranteeing dependability — critical factors for industries like finance, healthcare, manufacturing, and telecommunications.

One of the well-known approaches in big data is ETL (Extract-Transform-Load), which has become a standard for some time. Nishanth Reddy Mandala's research, "ETL in Data Lakes vs. Data Warehouses" [6], reveals the benefits of the ETL method for Data Lake and Data Warehouse architectures. Among strengths, some significant weaknesses include latency, delayed data availability, rigid schema, lack of flexibility, and high maintenance and scalability costs.

The ELT (Extract-Load-Transform) approach was introduced to address the ETL challenges. The book "Delta Lake - Deep Dive" [7], written by Nikhil Gupta & Jason Yip, uncovers crucial aspects of the Lakehouse paradigm, considering the ELT approach for data pipelines. While ELT overcomes the challenges of ETL, this multi-hop architecture still has weaknesses:

- Delayed Updates – the longer the data pipeline and the more tools involved, the more time it takes to refresh the data product.
- Extended Time-to-Market – development work is duplicated because each business unit must repeat identical or similar processing tasks instead of leveraging a centralised, curated data product.
- Higher Expenses – analytics platforms thrive financially on compute usage rather than on storage. The increased reliance of business units on tools like DBT boosts profits for analytics SaaS providers.
- Redundant Work – many organisations utilise multiple analytics systems — such as various Data Warehouses, Data Lakes, and AI platforms — resulting in repeated processing efforts with ELT across these environments.
- Inconsistent Data – integration methods like Reverse ETL and Zero ETL, among others, result in discrepancies between analytical and operational applications. Connecting a real-time consumer or mobile app API to a batch-processing layer won't deliver uniform outcomes.

Another work, "A reference architecture for serverless big data processing" [8], addresses the challenges posed by existing ETL/ELT architectures. It emphasises the importance of reducing time-to-market for data products by leveraging serverless platforms due to their high scalability.

Confluent Platform introduces Shift-Left architecture for Big Data to enhance data processing, as this approach was initially applied to testing. The article "Shift Left: Headless Data Architecture" [9] considers the key advantages of the approach:

- Enhanced Data Processing Efficiency: Shift-Left reduces the need for extensive data transfers by processing data closer to the source, thereby decreasing both the time and costs associated with large-scale data movement.
- Real-Time Over Batch Processing: By enabling real-time or near-real-time data handling rather than batch methods, Shift-Left supports applications that rely on current data, such as predictive analytics, machine learning systems, and operational decision-making.
- Better Data Quality: Processing data closer to its origin enables early detection and correction of issues, preventing quality problems from spreading downstream and ensuring reliable, high-quality data.
- Integrated Workloads: The Shift-Left approach connects transactional (operational) and analytical tasks, facilitating seamless real-time data sharing across applications for use cases such as real-time inventory tracking and personalised customer interactions.
- Faster Innovation and Expansion: This architecture speeds up the delivery of data-driven applications to the market, enabling businesses to deploy data products more quickly [9, 10].

The next work, "Shift Left. Unifying Operations and Analytics With Data Products" by Adam Bellemare [11], explores how the headless data architecture, termed "Shift-Left," bridges the gap between operational and analytical data. The text evaluates existing data architectures — Delta Lake, Data Warehouse, and Data Lakehouse — highlighting the challenges in transferring data from operational to analytical systems. It identifies the key limitations of these architectures:

- Downstream consumers bear full responsibility for the ETL/ELT process. However, without ownership of the source, systems must ensure that data remains relevant, available, and consistent.
- These architectures are expensive, requiring significant data copying and processing power. They create redundant processing due to cross-team communication problems, and outages or inconsistencies affect all downstream systems.
- It requires data reconciliation — restoring data quality after applying different quality gates after de-formalising, restructuring, and enriching.

- Curated data is not reusable for operational workloads, as the analytical workloads are optimised only for OLAP (online transaction processing systems).

The author highlights another aspect: processing insufficient data is the weakest part of the Shift-Left architecture. Due to the architecture's reliance on immutable data streams, where data is append-only and cannot be altered once written, handling corrupted data poses a significant challenge. Unlike traditional ETL/ELT processes, which employ a "stop-the-world" approach to remove, correct, and reprocess insufficient data, ensuring downstream systems receive consistent records, streaming architectures lack this flexibility. Corrupted data in streams can lead to severe consequences, including financial losses and irreversible business decisions. To mitigate these risks, Bellemare proposes several strategies for managing and preventing insufficient data in streams:

- Prevention Strategy: Emphasising rigorous design, thorough testing, and robust validation rules to ensure data integrity from the outset.
- Issue Correction Event Design: Establishing mechanisms to notify downstream services of specific data updates, facilitating corrective actions without altering the original stream.
- Rewind, Rebuild, and Retry Strategy: When other methods fail, recreate data streams with correct data as a last resort.

Despite these strategies, the Shift-Left architecture remains highly sensitive to insufficient data. While Bellemare underscores the importance of prevention, the proposed solutions do not entirely address challenges arising from workload fluctuations or provide proactive measures to anticipate and prevent potential incidents. A stronger focus on predictive analytics and adaptive workload management could further enhance the architecture's resilience.

Since the approach eliminates multiple stages for processing data and moves data extraction, sanity checks, and enrichment closer to the source, incorrect data schemas and improper streaming or processing settings can lead to several problems in the early processing stages, including performance degradation and data loss. In the next chapter, we propose a new approach to extend the existing Shift-Left Architecture with a module that will identify and mitigate potential issues nearer to the source.

3. Competency Module of Shift-Left Architecture

In Big Data, shift-left architecture refers to processing and governing data closer to its source rather than moving it to a centralised location for processing, such as in traditional Extract-Transform-Load (ETL) pipelines. This approach aligns with modern data strategies prioritising real-time processing, cost efficiency, and data quality. It is particularly relevant in environments leveraging data streaming technologies like Apache Kafka and Flink and platforms like Databricks and Snowflake, which support decentralised processing [4].

The strengths of Shift-Left Architecture are [9]:

- Data processing efficiency. By processing data near its source, Shift-Left reduces the need for large-scale data movement, which is often costly and time-consuming.
- Real-time processing instead of batch enables real-time or near-real-time data processing, which is critical for applications requiring up-to-date information, such as predictive analytics, machine learning models, and operational decision-making.
- Improved data quality. So that issues can be identified and corrected closer to the source. It prevents data quality problems from propagating downstream, ensuring only high-quality, trustworthy data is used.
- Unified workloads. Shift-left architecture facilitates the unification of transactional (operational) and analytical workloads. It allows for consistent real-time data sharing across applications, enabling scenarios like real-time inventory management and personalised customer experiences.
- Innovation and growth. The architecture is designed to speed up the time to market for data-driven applications, ensuring that data products reach the business more quickly.

While it has significant strengths, the following weaknesses have been identified through recent analyses [12]:

- Complexity in implementation. Transitioning to Shift-Left Architecture often requires significant changes to existing data pipelines and architectures. It can be complex and resource-intensive, especially for organisations with legacy systems. Even if a new pipeline is designed, some complex logic can be more safely implemented than streaming, as streaming pipelines have multiple challenges.
- Dependency on advanced tools. Effective implementation relies on modern tools like data streaming platforms (e.g., Apache Kafka, Flink) and cloud-native technologies. Hence, the designed applications are unlikely to be cloud-agnostic.
- Balancing speed and quality. While Shift-Left emphasises real-time processing, ensuring data quality remains a challenge. Organisations must embed robust quality checks at the source to avoid compromising accuracy in favour of speed.

Thus, the new architecture brings the following challenges:

- Source dependence. Since data is processed near its origin, the source system's quality, format, and configuration are critical. If the source is poorly configured or ineffective (e.g., producing incomplete or erroneous data), it can directly impact the downstream processes.
- Configuration issues. Misconfigurations at the source and processing engine, such as incorrect data schemas and improper streaming or processing settings, can lead to problems in the early processing stages. These issues may not be easily mitigated since centralised validation or transformation is less likely in modern architectures.
- The effectiveness of the source system is also crucial. If the source system is slow or unreliable, it can bottleneck the entire pipeline, reducing the benefits of real-time processing.

This section introduces a solution that addresses the existing challenges within Shift-Left Architecture.

3.1. Methodology of the Competency Module

The module incorporates the Holistic Adaptive Optimisation Technique (HAOT), a method engineered to overcome the shortcomings of conventional parameter tuning approaches for Delta Lake [13]. HAOT offers a thorough and flexible optimisation framework that employs machine learning to persistently evaluate and adjust the configurations of sources, streaming engines, and sinks dynamically in real-time. In this article, we apply the HAOT technique within Shift-Left Architecture.

HAOT functions by leveraging real-time performance feedback and making adaptive configuration adjustments. The method encompasses the following essential steps:

- Ongoing data collection that tracks performance metrics of the streaming application — such as throughput, latency, and resource usage — while also monitoring the configurations of sources, streaming engines, and sinks.
- Employing machine learning algorithms to evaluate the gathered data and uncover connections between the configurations of sources, streaming engines, and sinks and their effects on application performance, including building a predictive model to assess the performance outcomes of different configurations.
- Using insights from this relational analysis to dynamically tweak the configurations of sources, streaming engines, and sinks for enhanced performance, guided by machine learning models that forecast optimal settings based on current conditions.
- Continuously refining the machine learning models with fresh performance data as the streaming application operates, enabling the system to adjust to evolving data trends and operational environments, thereby maintaining effective and relevant optimisation over time.

We added a Competency module to the existing architecture to address the challenges mentioned at the beginning of this section. This module offers the key benefits mentioned in Table 1.

Table 1
Key Features of a Competency Module in Shift-Left Architecture

Area of Enhancements	How the Competency Module Contributes	Supporting Mechanisms/Examples
Maintainability	Automates validation, suggests enhancements, and standardises processes	Automated configuration checks, performance analytics, and best practices enforcement
Early Issue Detection	Proactively validates monitors in real time.	Real-time metrics tracking (e.g., Prometheus), dead-letter queues, and early anomaly detection
Risk Assessment	Implements data quality checks to identify risks early	Schema validation, versioning strategies, and continuous monitoring for bottlenecks

3.2. Implementation Details of the Competency Module

The new design brings the following components to the existing Shift-Left architecture (Fig. 3):

- Streaming sources represent the starting points of data streams, encompassing diverse real-time data producers like IoT devices, social media updates, log files, or sensors. These sources generate a steady flow of data fed into the streaming pipeline, which delivers data into the system in real-time or near real-time.
- After collecting data from streaming sources, a streaming processing cluster manages it. This cluster comprises distributed computing resources designed to handle large data volumes in real-time. It performs various functions, including filtering, aggregating, transforming, and analysing the streaming data. Operating continuously, the cluster frequently employs parallel processing and fault-tolerant techniques to manage high data throughput and ensure data accuracy. It is built to scale effectively to accommodate fluctuating volumes of incoming data streams.
- Data Collection Service is tasked with retrieving and structuring data from diverse sources for optimisation purposes, utilising connectors. The data may encompass performance metrics, system logs, environmental factors, and other relevant information, depending on the specific streaming pipeline. The Data Collection Service ensures that data is gathered consistently, efficiently, and securely while preprocessing it — through actions like cleaning, normalising, and transforming — to prepare it for analysis and modelling. This service is vital, as the quality and applicability directly influence the optimisation process's success.
- The ML Model for Parameter Tuning is the central feature of the proposed HAOT implementation. The model is engineered to analyse collected data, uncovering patterns, correlations, and trends that might be missed during human observation. It can leverage a range of algorithms — such as regression, classification, clustering, or deep learning — tailored to the specific challenge. Trained on historical data, the model predicts outcomes, enhances processes or delivers insights. As additional data is gathered and conditions evolve, the model can be retrained or fine-tuned, allowing it to adapt to emerging patterns and boost its precision and utility over time. For this service, we will implement a Long Short-Term Memory (LSTM) model for parameter tuning due to the time-series nature of streaming data [14].
- ML Model for Risk Assessment is a new component of the HAOT implementation proposed by the Competency Module framework. This model analyses collected metrics, identifying potential risks, anomalies, and vulnerabilities that may not be readily noticeable to human

analysts. It utilises various algorithms — such as anomaly detection, classification, time-series forecasting, and deep learning — customised to tackle specific risk-related challenges in streaming pipelines. This module will be trained on historical data that includes performance metrics, error logs, and configuration states, and the model will predict risk probabilities, flag potential issues, and provide actionable mitigation insights [15, 16]. The implementation of this module is beyond the scope of this article.

- The ML Model for Issue Detection is also a new component for HAOT implementation provided by the Competency module, which is designed to evaluate and enhance data pipelines within the Shift-Left Architecture for Big Data streaming. This model incrementally analyses pipeline performance data to pinpoint inefficiencies, bottlenecks, and areas for improvement that might otherwise go unnoticed. It employs diverse algorithms — such as clustering, anomaly detection, regression, or reinforcement learning — tailored to detect issues and recommend optimisations across the pipeline [15, 16]. As a previous component, this also will be trained on historical and real-time data; the model identifies suboptimal configurations, suggests alternative source setups (e.g., adjusting Kafka partitions), proposes different processing engines (e.g., switching from Flink to Spark Streaming), and even recommends architectural adjustments (e.g., adding redundancy). Implementing this module is also out of the scope of this paperwork.
- The Control Module functions as the central decision-maker in this framework. It leverages the insights and suggestions provided by the ML model to make choices or modifications to the system or process under optimisation. It includes tweaking parameters or refining strategies as needed. The module is engineered to execute these adjustments in a deliberate, trackable, and reversible way, facilitating ongoing monitoring and fine-tuning based on performance feedback and shifting conditions. It acts as the bridge connecting the analytical outputs of the ML model with the system's operational elements, ensuring seamless and effective optimisation. By maintaining a structured approach, the Control Module guarantees that changes enhance efficiency while preserving system stability. It is pivotal in translating data-driven recommendations into practical, impactful actions [13].
- The Control Centre is a critical component of the proposed Competency Module within the HAOT implementation, designed explicitly for Shift-Left Architecture in Big Data streaming. This web-based interface serves as the centralised hub where users can access and analyse the outputs of the ML Model for Risk Assessment and Issue Detection. It provides actionable insights that are intuitively and visually appealing, including identified risks (e.g., potential bottlenecks or data quality issues) and detected inefficiencies (e.g., suboptimal source configurations or processing engine recommendations). Designed for real-time interaction, the Control Centre enables users to monitor pipeline health, review suggested optimisations, and assess risk probabilities through dashboards, charts, and alerts. It helps users make informed decisions by consolidating complex analytical results into clear, actionable recommendations, bridging the gap between machine learning outputs and operational responses. The interface also accommodates user feedback, allowing adjustments to be flagged for the Control Module to implement, ensuring seamless integration with the broader HAOT framework. Ultimately, the Control Centre enhances visibility and control, aligning with the Shift-Left paradigm's emphasis on proactive management and early intervention in streaming pipelines.

The proposed Competency Module within the HAOT framework for Shift-Left Architecture in Big Data streaming integrates a comprehensive set of components to enhance pipeline optimisation. The Data Collection Service ensures high-quality, preprocessed data as the foundation for analysis, directly impacting the effectiveness of subsequent processes. The ML Model for Parameter Tuning, using an LSTM approach, facilitates dynamic configuration optimisation by uncovering patterns in time-series data. Meanwhile, the ML Models for Risk Assessment and Issue Detection improve the framework by proactively identifying risks and inefficiencies while offering tailored recommendations for enhancement. The Control Module connects these analytical insights to actionable outcomes, adjusting in a controlled and adaptive manner. Together, these components create a cohesive system that boosts maintainability, encourages early issue detection, and fortifies risk management, paving the way for efficient and reliable streaming pipelines. Although the implementation details of the Risk Assessment and Issue Detection models exceed the scope of this article, their conceptual integration emphasises the framework's potential for comprehensive optimisation.

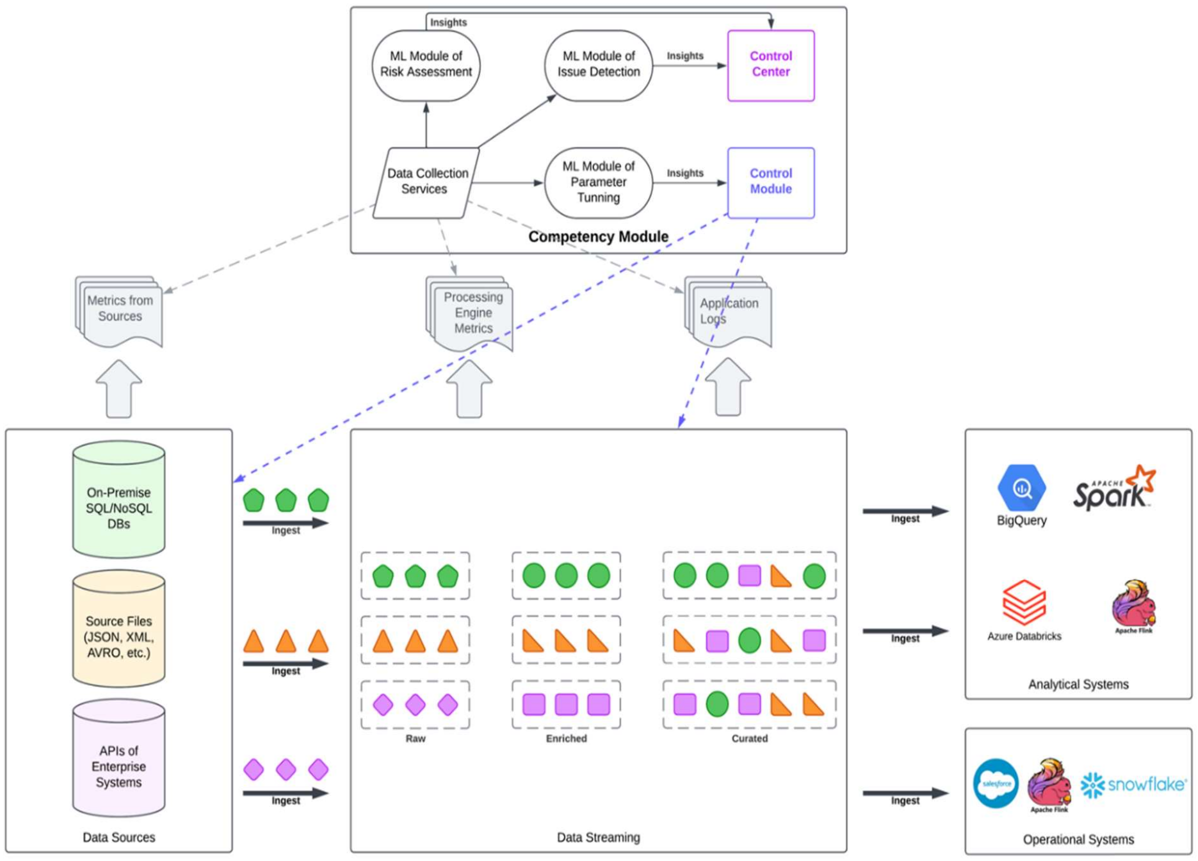

Figure 2: Competency Module in Shift-Left Architecture

4. Experiment

This article examines the proposed Competency Module, which implements HAOT specifically for Shift-Left architecture. The key components of the experiment include the ML Model for Parameter Tuning, the Data Collection Service, and the Control Module.

To evaluate the approach, we apply the following parameter-tuning statement. It can be defined as finding optimal options using the arguments of the maxima (argmax) approach [17, 18]. Given the pipeline job j that processes an input data stream s over cluster resources r and loads data to the sink t, the parameter tuning can be considered as evaluating an optimal configuration c_{opt}, that maximises performance metric function F over configuration space S:

$$c_{opt} = \underset{c \in S}{\operatorname{argmax}} F(j, s, t, r, c). \tag{1}$$

To assess the implementation, we chose data streaming pipelines designed using the Shift-Left approach with the following technology stack:

- Sources: Confluent Kafka.
- Processing Engine: Kafka Streams on a Kubernetes Cluster.
- Sink: Confluent Kafka

These pipelines operated for one month, generating comprehensive logs and metrics that enabled us to select parameters using the machine learning model. Given the wide range of parameters these technologies involve, training the model with every possible variable is currently unfeasible. However, the main goal of this study is to assess whether this method is viable. The parameters selected for the experiment are outlined in Table 1.

Table 2
Parameters of the Tested Distributed Data Streaming Pipeline

Technology	Component	Parameter	Description
Confluent Kafka	Source/Target	partitions	Defines several partitions of a Kafka topic. Ideally, this parameter would be fine-tuned by the control module, but we will bypass this step for now since it involves a challenging task [19].
		cleanUpPolicy	Determines how log segments are managed for a topic. The two main options are "delete," which removes old log segments after a retention period, and "compact," which retains the latest value for each key by removing older duplicates, ensuring efficient storage use [19].
		retention	Refers to the duration or size limit for which messages are stored in a topic's log before being eligible for deletion or compaction, as defined by the CleanUp policy [19].
		min.insync.replicas	Specifies the minimum number of in-sync replicas that must acknowledge a write to succeed. In Kafka Streams, it affects the durability of data written to output and internal topics, ensuring consistency based on the underlying Kafka topic configuration [19].
Kafka Streams	Processing Engine	linger.ms	Sets the maximum time (in milliseconds) the internal producer buffers data before sending it to output topics, balancing latency and throughput. The value is zero by default, meaning records are sent immediately [20].
		compression.type	Specifies the compression codec for data written to output topics. It controls the internal producer's compression, with possible values: none, gzip, snappy, lz4, or zstd. The default value is none (no compression) [20].
		fetch.max.bytes	Sets the maximum data size (in bytes) the internal consumer fetches from input topics in a single request. The default is 50 MB [20].
		max.poll.records	Sets the maximum number of records the internal consumer fetches from input topics in a single poll() call. The default is 1 [20].
		request.timeout.ms	Sets the maximum time (in milliseconds) for the internal producer or consumer to

			wait for a response from the broker before timing out. The default is 30 seconds [20].
		max.poll.interval.ms	Sets the maximum time (in milliseconds) for the internal consumer to block between poll() calls before being considered failed. The default value is 5 minutes [20].
Kubernetes	Compute Environment	minReplicas	Specifies the minimum number of pod replicas running for a given workload. It ensures the application maintains at least this many instances, even under low load, for availability and resilience [21].
		maxReplicas	Specifies the maximum number of pod replicas created for a workload. It sets an upper limit on scaling to prevent excessive resource usage under high load. [21]

In the experiment, the following options were unchanged. The Kafka topic had 12 partitions (as this parameter is not flexible yet in Kafka Architecture), and cleanUpPolicy was Delete (as the change of this field requires recreating the topic). The ML model set the other options – the min/max number of pod replica (minReplicas, maxReplicas), topic retention period, compression codec for records (compression.type), the max size of bytes to fetch data from the source (fetch.max.bytes), max number of records an individual consumer can pull from Kafka (max.poll.records), timeout of waiting a response from broker (for consumer and producer, request.timeout.ms), the timeout of pulling block of data from a topic (request.timeout.ms) and the max time for buffering data before sending downstream (linger.ms).

The input dataset had the characteristics mentioned in Table 3.

Table 3
Input Dataset Characteristics

Property	Value
Size	100 million
Record Data Type	Avro using Confluent serialiser
Average Record Size (In Kafka)	274 bytes – the value is low as the Avro format uses compact binary encoding to reduce the size of the serialised data
Total Dataset Size (In Kafka)	24 GB

The Kubernetes cluster had 16 CPU cores and 64 GB of memory in total. The performance metrics were taken from the Confluent Control Centre, and application logs were used with Loki and Grafana. Table 4 has the list of metrics and their destination [22, 23, 24].

Table 4
Application Metrics

Metric	Source	Destination
CPU Utilisation	Application Pod	Extracted from Kubernetes, collected in Grafana
Memory Utilisation	Application Pod	Extracted from Kubernetes, collected in Grafana
Network Utilisation	Application Pod	Extracted from Kubernetes, collected in Grafana
Kafka Lag	Kafka	Dashboard in Confluent Control Centre

The experiment setup:

- Base. The first part of the experiment involved running the pipeline with metrics collection and parameter tuning activated solely for Kaka Streams. It served as the baseline for performance metrics, where only the internal components of Spark Streaming were optimised based on available data without external influences from other pipeline components.

- HAOT Applied. In the second part, the experiment extended metrics collection and parameter tuning to include the processing engine Kafka Streams and Kafka parameters as a Source and Target (mentioned in Table 2). This approach represents the holistic application of HAOT, where the optimisation technique is applied across the entire data pipeline rather than in isolated segments.

5. Results

The runs for the first experiment were with default values for configurations (Table 5).

Table 5
Parameters of the run "Base"

Technology	Component	Parameter	Value
Confluent Kafka	Source	partitions	12 (Static)
Confluent Kafka	Source and Parget	cleanUpPolicy	Delete (Static)
Confluent Kafka	Source and Parget	retention	7 days (default)
Kafka Streams	Processing Engine	linger.ms	0 (default)
		compression.type	None (default)
		fetch.max.bytes	50 MB (default)
		max.poll.records	1 (default)
		request.timeout.ms	30 seconds (default)
		max.poll.interval.ms	5 minutes (static)
Kubernetes	Processing Environment	minReplicas	1 instance
		maxReplicas	3 instances

For the second experiment, we applied HAOT, where the ML model provided different parameters, as mentioned in Table 6.

Table 6
Parameters of the run "HAOT Applied."

Technology	Component	Parameter	Value
Confluent Kafka	Source	partitions	6 (Static)
Confluent Kafka	Source and Parget	cleanUpPolicy	Delete (Static)
Confluent Kafka	Source and Parget	retention	3 days (min allowed)
Kafka Streams	Processing Engine	linger.ms	500 milliseconds (max allowed)
		compression.type	Snappy
		fetch.max.bytes	512 MB (max allowed)
		max.poll.records	724
		request.timeout.ms	120 seconds (max allowed)
		max.poll.interval.ms	15 minutes (max allowed)
Kubernetes	Processing Environment	minReplicas	3 instances
		maxReplicas	10 instances (max allowed)

The results of average metric values of the two runs ("Base" and "HAOT Applied") are presented in Table 7.

Table 7
Experiment Results

Metric	Description	The Base Run	HAOT Applied	Difference
CPU Utilisation (per instance)	The bigger, the better	15%	91%	76%
Memory Utilisation (per instance)	The lower, the better	1476 MB	1684 MB	-14%
Network Utilisation in MB per second	The lower, the better	22.8 MB	18 MB	21%
Kafka Lag per second	The lower, the better	28616	805	97%

6. Discussion

In the experiment, CPU utilisation metrics increased, indicating that compute resources were used more effectively. This improvement in computing power utilisation suggests that the system manages workloads more efficiently under the modified configuration. Memory utilisation rose by 14%, a change due to adjustments in specific Kafka consumer parameters: linger.ms, fetch.max.bytes, and max.poll.records. The increase in linger.ms likely allowed the producer to buffer more data before sending, optimising throughput at the expense of higher memory usage. Similarly, raising fetch.max.bytes enabled the consumer to retrieve larger data batches per fetch request while increasing max.poll.records allowed more records to be processed per poll, collectively contributing to greater memory demand.

A significant outcome was the 97% reduction in Kafka lag per second, driven by an enhanced record-pulling mechanism. This improvement resulted from tuning fetch.max.bytes and max.poll.records, which enabled the consumer to retrieve more data in fewer, more extensive requests, thereby reducing the frequency of polls and minimising lag. Additionally, adjustments to request.timeout.ms and max.poll.interval.ms played a crucial role in mitigating network-related issues. By extending the timeout thresholds, these parameters offered greater resilience against delays caused by an overloaded Kafka broker, ensuring the consumer could wait longer for responses without failing, thus stabilising performance under high load.

The ML model guiding the experiment recommended higher values for several parameters, but these values were limited: linger.ms, fetch.max.bytes, request.timeout.ms, max.poll.interval.ms, and maxReplicas (from Kubernetes). For instance, a higher linger.ms improved batching efficiency while increasing fetch.max.bytes and max.poll.interval.ms optimised data retrieval and processing intervals. While these elevated values significantly boosted overall system performance—evidenced by reduced lag and better resource utilisation—they also introduced a trade-off: the processing delay for individual records increased. This latency resulted because larger batches (via linger.ms and fetch.max.bytes) and extended timeouts (via request.timeout.ms and max.poll.interval.ms) prioritised throughput over per-record responsiveness, potentially causing single-record processing to wait longer in the pipeline. In the Kubernetes context, setting a higher maxReplicas in the HorizontalPodAutoscaler (HPA) enabled the system to scale to more pod instances under peak demand, enhancing throughput and fault tolerance.

The experiment markedly improved Kafka consumer efficiency and system scalability, with CPU and memory resources better utilised and Kafka lag nearly eliminated. However, the configuration's focus on batch optimisation and network resilience, combined with increased replica scaling, suggests a design favouring high-throughput workloads over low-latency, single-record processing.

7. Conclusions

In summary, integrating a Competency module into the Shift Left architecture has significantly enhanced service performance and stability, underscoring its value as a crucial improvement. This enhancement enables earlier detection and improves the system's overall reliability. However, challenges remain, particularly in achieving scalability, managing the complexity of machine learning models, and maintaining an appropriate balance between optimisation frequency and system stability. Looking ahead, efforts will focus on refining this approach to tackle these challenges, enhancing the efficiency of the machine learning algorithms, and broadening its application to a broader range of streaming platforms and use cases. These advancements are expected to further augment the benefits of the Competency module within the Shift Left framework. Furthermore, we will examine the second part of the Competency module — Risk Assessment and Issue Resolution — to strengthen the framework. These developments are expected to further solidify the Competency module's advantages within the Shift Left architecture.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] P. Strengholt, Data Management at Scale: Modern Data Architecture with Data Mesh and Data Fabric, 2nd. ed., O'Reilly Media, Inc., 2023, pp. 173-175.

[2] H. Dulay, and S. Mooney, Streaming Data Mesh: A Model for Optimising Real-Time Data Services, 1st. ed., O'Reilly Media, Inc., 2023, pp. 36-39.

[3] Gaurav Ashok Thalpati, Practical Lakehouse Architecture: Designing and Implementing Modern Data Platform at Scale, O'Reilly Media, Inc., 2024, pp. 372-381.

[4] Confluent, What is Shift Left?, 2025.URL: https://www.confluent.io/learn/what-is-shift-left.

[5] Kai Waehner, The Shift Left Architecture — From Batch and Lakehouse to Data Streaming, 2024. URL: https://kai-waehner.medium.com/the-shift-left-architecture-from-batch-and-lakehouse-to-data-streaming-d1ea7306ea30.

[6] Nishanth Reddy Mandala, ETL in Data Lakes vs. Data Warehouses, v. 1 of ESP Journal of Engineering & Technology Advancements, 2021. doi:10.56472/25832646/JETA-V1I2P123.

[7] N. Gupta, J. Yip, Delta Lake - Deep Dive, Databricks Data Intelligence Platform, Apress, Berkeley, CA, 2024, pp. 61–88. doi:10.1007/979-8-8688-0444-1_4.

[8] S. Werner, S. Tai, A reference architecture for serverless big data processing, volume 155 of Future Generation Computer Systems, 2024. doi:10.1016/j.future.2024.01.029.

[9] Confluent, Shift Left: Headless Data Architecture, 2024. URL: https://www.confluent.io/blog/shift-left-headless-data-architecture-part-2.

[10] An Overview of Shift Left Architecture, 2025. URL: https://www.deltastream.io/shift-left-architecture-an-overview/.

[11] A. Bellemare, Shift Left Unifying Operations and Analytics With Data Products, 2024. URL: www.confluent.io/resources/ebook/unifying-operations-analytics-with-data-products.

[12] Navdeep Singh Gill, Mastering Shift Left Architecture for Real-Time Data Products, 2025. URL: https://www.xenonstack.com/blog/shift-left-architecture-data-products.

[13] V. Vysotska, I. Kyrychenko, V. Demchuk, I. Gruzdo, Holistic Adaptive Optimization Techniques for Distributed Data Streaming Systems, CEUR Workshop Proceedings, Vol-3668, 2024, ISSN 16130073. doi:10.31110/COLINS/2024-2/009. https://ceur-ws.org/Vol-3668/paper9.pdf

[14] M. Trotter, T. Wood, and J. Hwang, Forecasting a Storm: Divining Optimal Configurations using Genetic Algorithms and Supervised Learning, IEEE International Conference on Autonomic Computing (ICAC), 2019. doi:10.1109/ICAC.2019.00025.

[15] M. T. Islam, S. Karunasekera, and R Buyya, Performance and Cost-Efficient Spark Job Scheduling Based on Deep Reinforcement Learning in Cloud Computing Environments, IEEE Transactions on Parallel and Distributed Systems, 2021.

[16] Kyrychenko, I., Tereshchenko, G. Proniuk, G., Geseleva, N. "Predicate Clustering Method and its Application in the System of Artificial Intelligence", CEUR-WS, 2023. V.3396, PP.395 - 406.

[17] H. Herodotou, Y. Chen and J. Lu, A Survey on Automatic Parameter Tuning for Big Data Processing Systems, ACM Computing Surveys (CSUR), Vol. 53, 2020.

[18] H. Sagaama, N. B. Slimane, M. Marwani, and S. Skhiri, Automatic Parameter Tuning for Big Data Pipelines with Deep Reinforcement Learning, IEEE Symposium on Computers and Communications (ISCC), 2021. doi:10.1109/ISCC53001.2021.9631440.

[19] Kafka Doc, 2025. URL: https://kafka.apache.org/documentation/#brokerconfigs.

[20] Kafka Streams Configuration Reference for Confluent Platform, 2025. URL: https://docs.confluent.io/platform/current/installation/configuration/streams-configs.html.

[21] HorizontalPodAutoscaler Walkthrough, 2025. URL: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough.

[22] E. Eldor, Kafka Troubleshooting in Production: Stabilising Kafka Clusters in the Cloud and On-premises, Apress Berkeley, CA, 2023, pp. 25-36.

[23] B. R. Prasad, and S. Agarwal, Performance Analysis and Optimisation of Spark Streaming Applications Through Effective Control Parameters Tuning, in: P. K. Sa, M. N. Sahoo, M. Murugappan (Ed.), Progress in Intelligent Computing Techniques: Theory, Practice, and Applications, Vol. 1, Springer Publishing Company, Incorporated, 2018, pp. 99-110.

[24] G. Tereshchenko, I. Kyrychenko, V. Vysotska, Z. Hu, Y. Ushenko, M. Talakh, Hybrid System for Image Storage and Retrieval in Big Data Environments, International Journal of Image, Graphics and Signal Processing(IJIGSP), Vol.17, No.3, pp. 55-84, 2025. DOI:10.5815/ijigsp.2025.03.04