Optimized Edge-to-Cloud Complex Event Recognition and Forecasting on Altair AI Studio

Ourania Ntouni
Technical University of Crete
Chania, Greece
ontouni@tuc.gr

Dimitrios Banelas
Technical University of Crete
Chania, Greece
dbanelas@tuc.gr

Elias Alevizos
NCSR "Demokritos" & The American College of Greece
Athens, Greece
alevizos.elias@iit.demokritos.gr

Nikos Giatrakos
Technical University of Crete
Chania, Greece
ngiatrakos@tuc.gr

***Abstract*—Industrial analytics increasingly span the cloud-to-edge continuum. Still, in IoT multi-engine and heterogeneous device deployments, operationalizing a single analytical workflow across cloud clusters, gateways and edge nodes typically requires expert knowledge in runtime selection, cross-engine connectivity and configuration management so that end-to-end performance metrics are optimized. On top of that, complex event recognition and forecasting (CER/F) analytics are essential to transform the lot of raw streams collected from devices across the continuum to actionable insights for proactive and reactive measures. This paper presents an open source EdgeToCloud Extension for Altair AI Studio, which enables data scientists to (i) design *logical* CER/F workflows in a device- and engine-agnostic manner and (ii) automatically obtain and deploy an *optimized physical deployment* across cloud, fog and edge execution engines. At the core of the extension are (i) a neurosymbolic CER operator for transforming perceptual streams to meaningful symbols (simple events) and perform CER (ii) a CER/F operator which deploys a state-of-the-art CER/F open-source framework, namely Wayeb, on edge devices for event forecasting (iii) an EdgeToCloud Optimization operator and service that searches for efficient operator-to-engine/device assignments and materializes the resulting optimal execution plan. We report on the system architecture, operator design, implementation details of the logical-to-physical compilation loop, as well as on case studies demonstrating end-to-end CER/F optimization and deployment for mass use.**

I. Introduction

Industrial analytics increasingly span the cloud-to-edge continuum in several application settings. Predictive maintenance in smart manufacturing benefit from early warnings from vibration/thermal/current streams. Industrial safety and situational awareness aims at detecting hazardous conditions and triggers alerts near the point of action. Energy and smart-grid monitoring heavily rely on local anomaly detection and forecasting with system-wide state estimation, while environmental and emergency monitoring necessitate early detection and forecasting from distributed sensor network streams for first responders to timely act.

The traditional approach of transferring all data in the cloud and perform the processing there is, however, suboptimal in such settings. This is due to excessive network latencies for transmitting the raw data and the fact that the available computational resources of devices across the cloud-to-edge continuum are not exploited. The natural alternative would be to assign some operators of an analytics workflow to be executed on various devices in the network, offloading computations to the lower layers, i.e., edge and fog, reducing transmitted data volumes, minimizing network latencies and decreasing cloud dependencies. Efficiently utilizing the computational capacity of all types of devices results in overall more efficient data processing across the infrastructure.

In practice, however, deploying a single analytical workflow across heterogeneous devices (cloud clusters, gateways, edge nodes) is still cumbersome. Multi-node and multi-engine environments require expert knowledge in network node and runtime selection, cross-engine connectivity (contracts, schemas) as well as configuration management (deployment packaging, endpoints, resource limits). At the same time, modern industrial applications need more than simple stream transformations. They require *Complex Event Recognition and Forecasting* (CER/F) to convert large volumes of raw perceptual streams into actionable, higher-level situations and insights. Recent work has shown that neurosymbolic approaches can bridge perception and symbolic reasoning for robust recognition at the edge and across IoT platforms [1], while dedicated CER/F engines (such as Wayeb [2]) can provide probabilistic forecasts of complex event occurrences by employing automata- and model-based techniques to enable proactive, instead of reactive measures [2]. Operationalizing such CER/F pipelines end-to-end across edge and cloud remains a deployment and orchestration challenge.

This paper presents an open source EdgeToCloud Extension [1] for Altair AI Studio [3] that addresses this gap by integrating an optimization-driven CER/F deployment loop directly into a commerical platform used across multiple industries. The EdgeToCloud Extension enables data scientists, who are not programmers or IoT experts, to: (i) graphically design *logical* CER/F workflows in a device- and engine-agnostic manner, and (ii) automatically obtain and deploy an *optimized physical CER/F workflow* across cloud/fog and edge execution engines on various network devices.

We thank Ralf Klinkenberg from Altair (RapidMiner) for providing necessary developer licenses and Altair AI Hub access. Work funded from the EU Horizon Europe programmes CREXDATA under grant agreement No 101092749 and EVENFLOW under grant agreement No 101070430.

[1] https://github.com/DBanelas/tuc-crexdata-extensions

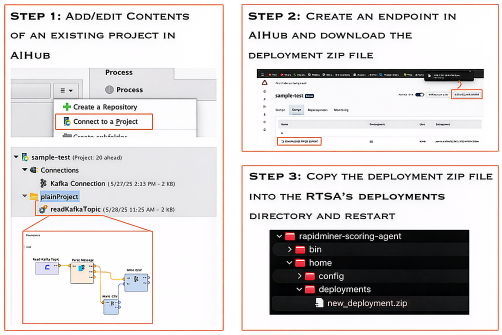

Fig. 1. Prior State of Practice for Executing Workflows on RTSAs

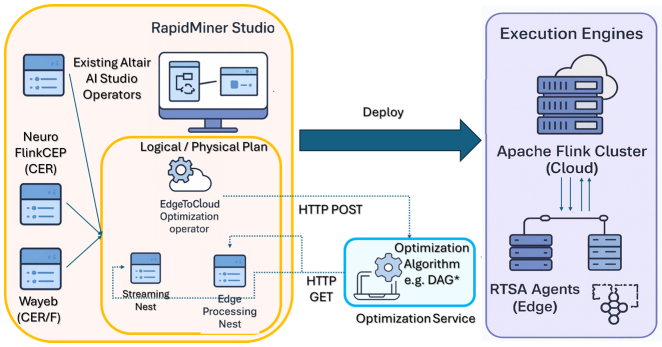

Fig. 2. High-level EdgeToCloud architecture.

II. Background

Altair AI Studio (former RapidMiner Studio) [3] is a visual tool for designing data science workflows. Users build pipelines by connecting and parameterizing visual operators into a graph (e.g., Figure 3). Running the graph executes each operator in order and produces models and analytics results.

RapidMiner Real-Time Scoring Agents and the AI Hub: A RapidMiner Real-Time Scoring Agent (RTSA) [4] is a lightweight runtime service that hosts deployed Altair/RapidMiner processes (workflows) as callable endpoints for low-latency, high-throughput analytics. Being lightweight with reduced hardware requirements, RTSAs do not exploit parallelism. Therefore, Apache Flink is employed for higher capacity devices at the cloud and fog network layers. RTSAs are still deployable at the cloud/fog side, but their ultimate utility is towards resource constrained fog and edge devices.

Within the scope of this paper, Altair AI Hub [5] acts as the "device registry". It keeps track of the RTSA agent instances, so the system can discover which execution targets are currently available and addressable. Mechanistically, AI Hub provides an internal service-registry endpoint that Web API agents use to register themselves to AI Hub, along with grouping metadata that AI Hub can use for orchestration and routing. The EdgeToCloud Extension we develop relies on AI Hub's maintained view of connected agents (rather than hard-coding device lists) when selecting admissible RTSA execution targets for fog/edge processing workflow fragments. On the other hand, it uses standard Altair AI Studio connectors for cloud/fog devices that can run Apache Flink.

Prior State of Practice: Prior to the development of the EdgeToCloud Extension introduced in the current work, Altair AI Studio did not include any support for CER/F. Additionally, its stream processing capabilities were limited to Flink and restricted to the cloud side of the continuum. RTSAs were available with Altair AI Hub acting as the device registry. However, besides the fact that RTSAs could not process streams, the entire deployment of a (non-CER/F) workflow had no automatic optimization facilities with the deployments being entirely manual: As shown in Figure 1, to deploy an existing workflow to AI Hub before the proposed EdgeToCloud Extension was developed, first the user had to open a project in AI Hub and ensure it is properly connected with the workflow in their local Altair AI Studio workspace. Then, the user would update and save the project contents so that the intended process, connections and components are correctly configured. Next (Step 2 in Figure 1), the user had to create an endpoint for the process within the AI Hub and export the project using the platform's deployment export option, which would produce a deployment ZIP package. Next, a copy of this deployment ZIP file had to be manually placed into the RTSA deployments (see the RTSA folder structure in Step 3 in Figure 1). Finally, the user had to restart the RTSA so as to reload the deployments and activate the newly installed endpoint. These steps had to be repeated for each device manually assigned to execute a workflow.

Motivation for CER/F: CER/F capabilities across the continuum are essential because they (i) lift raw tuple streams to semantically meaningful complex events and temporal patterns, enabling decisions to be expressed over situations rather than low-level streaming tuples with no specific isolated meaning, (ii) reduce communication and processing pressure by filtering and condensing data early before propagating it across the continuum, (iii) support timely reaction and proactive control by recognizing critical situations as they emerge and forecasting their likely evolution and (iv) provide a natural core for hybrid reasoning that combines pattern specifications with learning-based inference under distributed, heterogeneous execution conditions.

Consider the robotic workflow in Figure 3. The input is a high-rate stream of per-robot telemetry (e.g., position, velocity, goal status, deadlock and contact flags) that is informative only when interpreted over time. The workflow aims at recognizing production station skipping complex events by matching temporal patterns such as, a robot "moving towards station X" followed by "moving towards station Y" without an intervening "stopped at station X". It also aims at forecasting collision complex events by detecting short-horizon temporal combinations of risk cues (e.g., hard braking or deadlock episodes followed by collision/contact within horizon H).

Without CER/F support, downstream components would receive only raw telemetry tuples and would need to (a) implement their own stateful logic to reconstruct intent and temporally relate measurements, (b) continuously join and

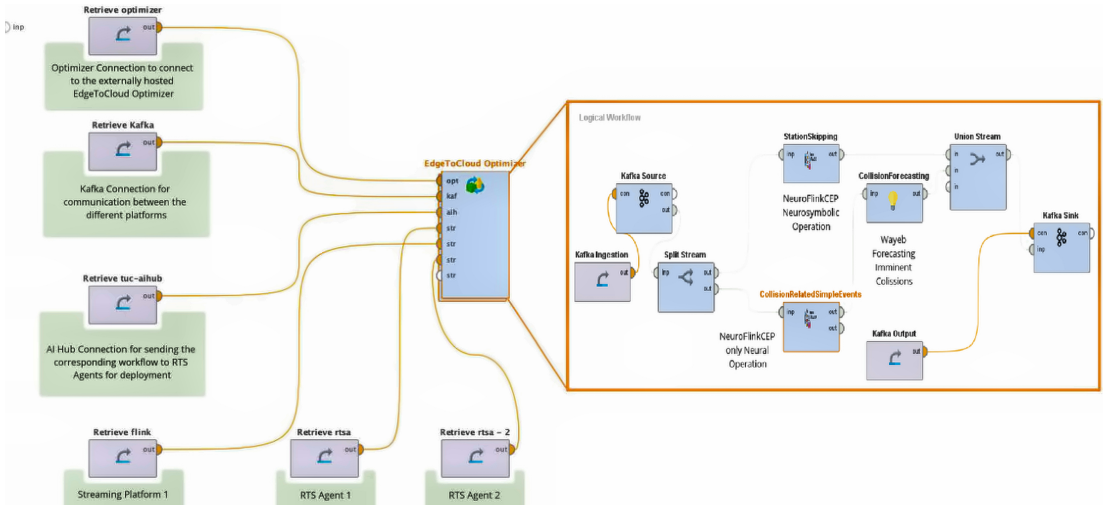

Fig. 3. Logical workflow inside the EdgeToCloud Optimization operator. The Optimization operator connects to the optimization service, the AI Hub, Kafka backbone and the available devices. Operators and their connections form a DAG which gets serialized and sent to the optimization service.

window trajectories to detect the absence of expected events (e.g., "did not stop at station X"), and (c) pollute the data plane by forwarding high-rate signals to whichever component needs to infer situations. This yields brittle application code (pattern logic scattered across many analytics operators), delayed reactions (situation inference happens "late" and often centrally at the cloud) and unnecessary traffic (raw tuples are shipped even though most are irrelevant to station-skipping or collision-risk assessment). With CER/F support, the system can explicitly encode the two key operational questions—"did a robot skip an intended station?" and "is a collision likely soon?"—as temporal patterns over simple events and produces actionable complex events directly in the stream. As a result, the communication backbone carries small, semantically meaningful event notifications instead of continuous raw telemetry. Downstream components can react immediately (e.g., trigger replanning or human operator alerts).

Optimization Setup: We consider a *logical* CER/F workflow authored in Altair AI Studio as a directed acyclic graph (DAG) $G = (V, E)$. Each vertex $v \in V$ is an operator and each directed edge $(u, v) \in E$ captures a data dependency (stream) from u to v. A logical workflow is *engine-agnostic*: it specifies *what* should be computed (operator structure and parameters) but not *where* or *how* it is deployed across the continuum.

Each operator v is associated with: (i) a capability set $\mathcal{S}(v)$ of admissible execution engines (e.g., $\{\mathsf{Flink}\}$, $\{\mathrm{RTSA}\}$, or both), (ii) expected resource consumption/execution latency statistics, (iii) placement constraints, and (iv) interface metadata required to connect multi-engine fragments. The deployment environment is a set of sites $\mathcal{P}$ connected by a network with bandwidth/communication latency statistics.

Given $(G, \mathcal{P})$, we want to find a *physical deployment plan* π that assigns each operator $v \in V$ to a site and engine in $\mathcal{S}(v)$, and inserts the necessary cross-engine connectors so that the overall plan is executable. The optimization aims at minimizing end-to-end latency, respecting resource limits.

III. System Overview

Figure 2 summarizes the EdgeToCloud Extension internal architecture, which adds a *logical-to-physical* compilation loop to Altair AI Studio that bridges (i) interactive, graphical workflow design, (ii) multi-device/engine deployments.

The user has the ability to design logical workflows incorporating the full capacity of Altair AI Studio operators that already exist in the platform. This involves the entire operator arsenal supported by Altair AI Studio, including ETL and data preparation, transformation (map/filter/join/aggregate) analytics, classical Machine Learning (ML), AutoML and deployment-oriented inference operators which can be packaged into a workflow. Via the EdgeToCloud Extension two fundamental operators elevate the Studio's capacilities to CER/F. The NeuroFlinkCEP [1] and the CER/F operator [2].

The EdgeToCloud design and execution process starts from the middle left side of Figure 2. The newly developed EdgeToCloud Optimization operator is a composite/nest operator. As illustrated in Figure 4, the EdgeToCloud Optimization operator connects to (i) a Kafka communication backbone, (ii) the optimization service which will process the logical workflow and output the physical one and (iii) the available IoT devices (2 RTSA-enabled devices and 1 Raspberry Pi (RPi) in Figure 4). The user can double click on the EdgeToCloud Optimization operator and they will be provided with a clean design canvas to author the logical workflow of interest (Figure 3).

The logical worfklow is a conceptual wofklow, incorporating application logic, but being deprived of any physical execution details. To author a logical workflow, the user graphically creates a directed acyclic graph by dragging and dropping operators on a canvas and connecting them via edges to define the dataflow. When the user presses a submit button, the logical workflow is transfered to the optimization service (middle-bottom of Figure 2). The optimization service runs a state-of-the-art IoT optimization algorithm [1] and replies

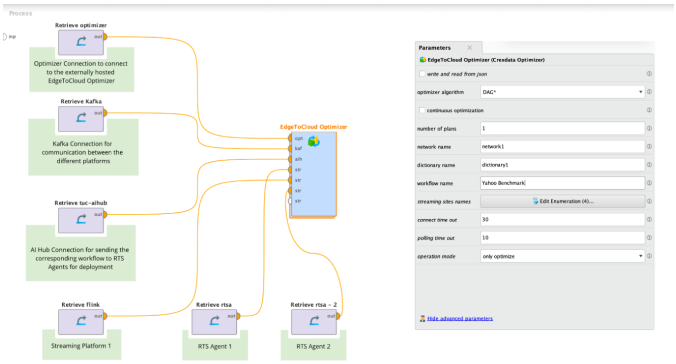

Fig. 4. EdgeToCloud Optimization operator with connections to Kafka, the optimization service, AI Hub, RTSA devices, and a Flink-enabled device.

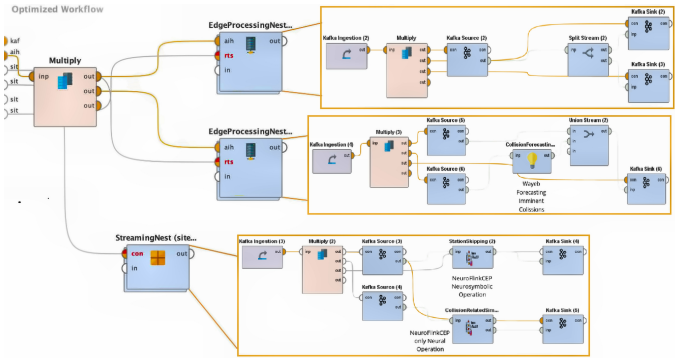

Fig. 5. Optimized physical workflow across the cloud-to-edge continuum, as StreamingNest (Flink-side) and EdgeProcessing Nest (RTSA-side) operators.

back with the physical workflow (bottom left of Figure 2).

As shown in the example of Figure 5, the physical workflow is composed of a number of StreamingNest and EdgeProcessingNest operators. These are nest operators that host fragments (sub-workflows) of the submitted workflow, decomposed based on their assignment for execution on the various devices of the continuum. StreamingNests include those parts of the workflow that are assigned for execution on higher capacity cloud/edge sites. On the other hand, EdgeProcessingNests include parts of the workflow assigned to more resource contrained fog/edge devices.

Finally, the physical workflow is automatically dispatched for execution to the involved IoT devices (right-hand side of Figure 2): as Flink jobs for more powerful cloud/fog devices and as RTSA deployment packages at fog/edge devices. In Figure 5, we can observe a physical workflow fragmented into three parts, connected via Kafka topics. Two such parts are assigned for execution on two devices running RTSAs, while another one is assigned for execution at a RPi (Flink)-side.

IV. Implementation Details

A. RTSA Streamification via Sleep-Triggered Micro-Batching

The first, most fundamental step in realizing our EdgeToCloud Extension was to operate RTSA endpoint deployments in a streaming-like fashion, so that they can process tuples continuously produced across the continuum.

RTSAs are not originally designed as streaming engines. Rather, they are positioned for low-latency decisioning use cases via web services. However, RTSAs already support a continuous execution mode and expose an optional sleep delay between successive decisions [4]. In continuous mode, a RTSA repeatedly re-executes a deployment endpoint (Section II) with an optional sleep delay between runs. Our contribution is to bind each re-execution to Kafka I/O, so each run consumes newly arrived Kafka records, processes them, produces outputs back to Kafka, and then sleeps, turning repeated endpoint executions into continuous data-driven micro-batches.

To enable continuous workflow execution as new tuples arrive across workflow fragments assigned to the involved devices, we introduce a Sleep-Triggered Micro-Batching mechanism that streamifies RTSA endpoint deployments. Our Sleep-Triggered Micro-Batching resembles the classic Apche Spark Streaming micro-batch paradigm. A time-driven trigger partitions an unbounded stream into successive slices of duration Δ, where Δ is configured via the RTSA sleep interval parameter (e.g., `sleep=5000` ms for $\Delta = 5$ s). `sleep` serves as an explicit stream trigger. If no Δ is specified, RTSA uses a default sleep (1 ms), yielding near back-to-back executions (subject to runtime overhead and data availability) and thus approximating an "as-fast-as-possible" micro-batching regime.

B. Automatic Streaming Deployments at the Fog/Edge

Having transformed RTSAs into stream processing engines, our next target is to automate the entire, previously manual, deployment process discussed in Section II. The newly developed EdgeProcessingNest operator is destined to encapsulate a physical workflow directly deployable and executable on RTSAs running at the fog/edge. From the perspective of the top-level Studio process, the EdgeProcessingNest can be used as a standalone operator. As shown in Figure 7, an EdgeProcessingNest operator has two input ports: one providing the connection details to the AI Hub registry and one pointing to the specific RTSA destined to execute the physical workflow. When used as a standalone operator, the user has to double click on the EdgeProcessingNest and author the physical workflow inside it (Figure 7). An additional operator parameter is the sleep interval. To complete the automation of the deployment, once the user presses the submit button, a REST service is called and the deployment ZIP is immediately shipped for execution to `http://$SA_HOST/admin/deployments` (where `SA_HOST` is the host name of the RTSA) via a `curl` command which is automatically issued: `curl -F file=@"/deployment.zip" http://host:8090/admin/deployments`.

C. Streaming Jobs at the Cloud/Fog-side

The StreamingNest operator (bottom of Figure 5) follows the same design principles. It differs from the EdgeProcessingNest in that, it admits one input port which the user should bind a connection (`host:port`) to the Apache Flink available at some site at the cloud/fog-side of the network.

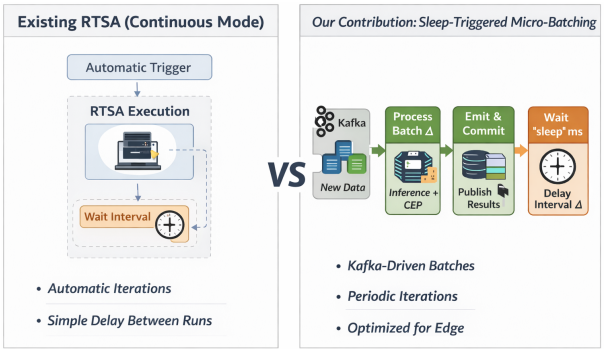

Fig. 6. Repurposing RTSA as stream processing engines via micro-batching.

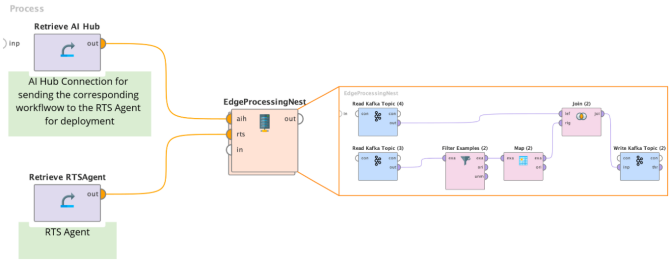

Fig. 7. A non-CER/F physical workflow inside an EdgeProcessingNest.

Again, StreamingNest can be used as a standalone operator calable of executing physical workflows on the Flink engine running at the specified `host:port`. The user can double click on this operator and author the physical workflow for the specified Flink-enabled network site. Upon submission, the workflow is compiled into a JAR file that is automatically shipped and deployed to the specified network device.

We use a hybrid setup (Flink true streaming. RTSA micro-batch) to avoid busy-polling and amortize overhead on constrained edge agents. End-to-end dataflow is discussed below.

D. Establishing a Communication Backbone

To automatically execute physical workflows at the cloud/fog and fog/edge sites independently, a communication backbone is necessary so that StreamingNests and EdgeProcessingNests interconnect and exchange analytics results.

In the very design of the EdgeToCloud Extention, Kafka constitutes the shared data plane between the StreamingNests and the EdgeProcessingNests, acting as the communication backbone of a physical workflow that is executed across the continuum. Our focus is to establish an engine-agnostic message contract that can be adapted to schemas of different operators, applications and use cases. To also bind together processing engines destined to run on devices of expectedly different processing speeds, we want to temporally decouple consumers and producers of intermediate processing results.

Rather than invoking operators across engine or device boundaries, our deployment bases each cross-(Streaming- or EdgeProcessing-)Nest connection to Kafka topic(s), effectively turning inter-fragment edges into durable, replayable communication channels. Consequently, physical workflow fragments become independently running components that interact exclusively through Kafka production/consumption.

Topic-per-edge wiring. Given a workflow composed of a number of StreamingNest and EdgeProcessingNest operators, every boundary edge that crosses these Nests should be mapped to a Kafka topic: (i) RTSA fragments publish intermediate tuples/events to topics that are consumed by Flink operators, and (ii) Flink operators publish enriched/aggregated tuples/events to topics that are consumed by RTSA fragments. This explicit wiring decouples the two engines while preserving a clear, inspectable realization of the physical dataflow.

Engine- and Device-agnostic message contract. Kafka messages exchanged at these boundaries are JSON-formatted and follow a lightweight envelope–payload structure. The envelope carries workflow-level metadata (e.g., schema version, message type, workflow/fragment identifiers). The payload contains the stream tuple (key,value) pair itself as JSON object, so both the tuple key and value can be validated against explicit schemas, enabling contract-based interoperability and evolution. This keeps each dataflow edge on the physical workflow engine- and device-agnostic and simplifies interoperability and schema evolution throughout the workflow.

Decoupling – Buffering – Replay. Kafka's log-based semantics provide temporal decoupling between producers and consumers, allowing Flink and RTSA fragments to operate at different processing rates while absorbing burstiness via topic buffering. Durability and offset-based consumption enable fragments to resume after restarts and, when necessary, replay data from committed offsets, without tight runtime coupling.

E. From Raw/Perceptual Streams to CER/F Workflows

So far, we have achieved streaming capabilities (Section IV-A), physical workflow execution (Sections IV-B and IV-C) and the establishment of an effective communication backbone (Section IV-D) useful for orchestration of workflows executed distributedly across the continuum. We have yet to discuss how CER/F capabilities are incorporated at the core of the EdgeToCloud Extension. CER/F capabilities are essential for the reasons explained in Section II.

1) Neurosymbolic CER at the Cloud/Fog: The Neurosymbolic CER operator, namely NeuroFlinkCEP [1], bridges an important gap for enabling CER/F. Perceptual, raw streams should be transformed to symbols before being ingested by CER/F engines such as FlinkCEP or Wayeb [2] which rely on automata for pattern matching and detection or forecasting of complex events. Instead of achieving this goal by imposing crisp thresholds provided by domain experts, NeuroFlinkCEP patches a pre-trained neural model (`.pb` TensorFlow file) so that perception-level features and signals are translated (classified) into *simple events*. Then, it forwards these simple event symbols to FlinkCEP for symbolic pattern matching. In that, complex events evolving as a combination (sequences, logical conjuctions, disjunctions etc) are recognized [1].

Remarkably, this operator can act as a purely neural one in case no pattern to be matched is specified. In the latter

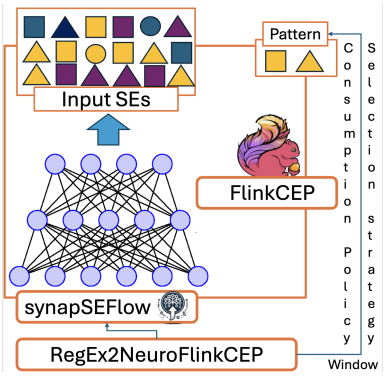

Fig. 8. NeuroFlinkCEP CER Operator Internal Architecture.

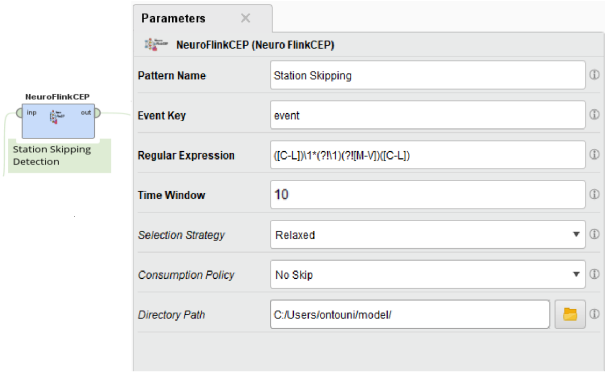

Fig. 9. NeuroFlinkCEP Operator Interface.

case, it outputs only simple event symbols which can then be processed by the forecasting (Wayeb) component of the EdgeToCloud Extension, discussed later on in this section.

Figure 8 depicts the internal architecture of the NeuroFlinkCEP operator in the EdgeToCloud Extension. The operator includes two modules. The `synapSEFlow` module implements the perception-to-symbol grounding: it consumes the raw stream and runs a neural detector to emit simple event symbols. Then, the `RegEx2NeuroFlinkCEP` module, having received a symbolic pattern specification in its parameters, translates it into a deployable FlinkCEP program. Simple events from `synapSEFlow` are fed into `RegEx2NeuroFlinkCEP`, which uses FlinkCEP to evaluate and produce higher-level matches (complex events) that can drive downstream CER/F logic.

Operator interface. Figure 9 illustrates the interface of the NeuroFlinkCEP operator we developed in Altair AI Studio. The user specifies: (i) *Pattern Name*: a user-defined identifier for the pattern, used for bookkeeping and labeling outputs, (ii) *Event Key*: the attribute (field) of the incoming stream tuple used as the event type/key on which pattern matching is performed, (iii) *Regular Expression*: an extended regular expression that specifies the event-sequence pattern to be detected over the chosen event key, (iv) *Time Window*: the temporal window (e.g., in seconds) within which the pattern must be completed to be considered a match, (v) *Selection Strategy*: the match-selection semantics used when multiple partial/overlapping matches are possible (i.e., how the operator chooses among competing matches), (vi) *Consumption Policy*: the event-consumption policy that governs whether matched events may participate in subsequent matches (i.e., overlap handling), (vii) *Directory Path*: filesystem path to the directory containing the pre-trained TensorFlow model that is loaded and patched into the `synapSEFlow` module at runtime.

Engine mapping. This operator is admissible to Flink-enabled devices. This is because it encapsulates a neural network whose size and GPU demands make it suitable for more powerful fog (e.g. Raspberry Pi 4/5 or NVIDIA Jetson Nano boards) or cloud sites, rather than resource-contrained ones. Upon being placed inside a StreamingNest operator, NeuroFlinkCEP is compiled into a FlinkCEP job whose JAR file gets shipped and executed to the corresponding device.

2) Wayeb for CER/F at the Fog/Edge: The CER/F operator integrates Wayeb, an open-source CER/F engine that, besides CER, supports probabilistic forecasting of complex event occurrences over streams [2]. Wayeb consumes streams of simple events (possibly provided by NeuroFlinCEP), recognizes complex events via automata as well as attaches statistical prediction models (i.e., Markovian Models) to estimate the likelihood and timing of future complex events [2].

The internal architecture of Wayeb is shown in Figure 11. Wayeb operates as a three-stage pipeline that turns an incoming simple event stream into probabilistic forecasts for the completion of a CER pattern. First, it relies on a learned *prediction suffix tree* (PST) that captures the conditional distribution of the next symbol (simple event) given the recent event-history context (top of Figure 11). In parallel, the target pattern is represented as a finite-state machine (FSM) whose states encode partial progress toward an accepting (completion) state and are updated deterministically as simple events arrive (middle of Figure 11). By combining the PST-derived next-event probabilities with the FSM transitions, Wayeb induces a probabilistic evolution over FSM states and estimates a *waiting-time* (time-to-completion) distribution, i.e., the probability that the pattern will be completed after k additional events. Finally, this distribution is used to construct forecasts (bottom of Figure 11) in the form of high-confidence completion windows (intervals over the number of future events within a given horizon), which are emitted when their completion probability exceeds a user-defined confidence threshold and satisfy constraints such as a maximum admissible spread.

Operator interface. In Altair AI Studio (Figure 11), the Wayeb CER/F operator exposes (i) *JAR path*: file path to the Wayeb executable Java archive launched by the operator; (ii) *fsm file path*: path to file of the serialized target CER/F pattern to be detected/forecast, (iii) *threshold*: confidence threshold $\theta \in (0, 1]$ that a forecast must exceed to be considered as important and get emitted, (iv) *horizon*: forecast horizon H, i.e., the maximum lookahead (typically in number of future events/steps, depending on the stream encoding), (v) *maxSpread*: maximum admissible spread $S_{\max}$ (window width) of the reported forecast interval, (vi) *center*: preferred

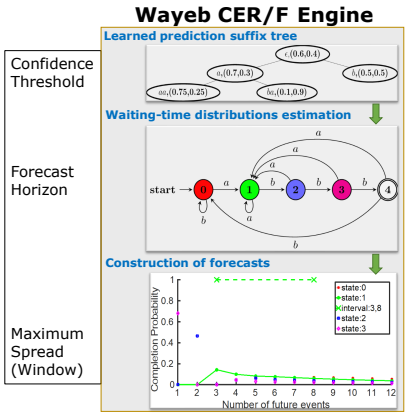

Fig. 10. Wayeb CER/F Operator Internal Architecture.

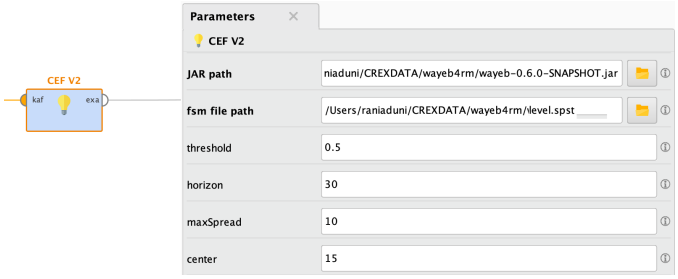

Fig. 11. Wayeb CER/F Operator Interface.

TABLE I
MAIN PARAMETERS OF THE EDGETOCLOUD OPTIMIZATION OPERATOR.

Parameter	Purpose
Optimization mode	Only Optimize / Only Deploy / Optimize and Deploy.
Polling time out	Frequency of HTTP GET polling for optimization responses.
Dictionary Name	JSON file with statistics about operator execution per site.
Network Name	JSON file with engine availability per site (Flink, RTSA, or both).

center c of the forecast interval (same units as H), to anchor the interval reported subject to $S_{\max}$ and θ. The operator streams out recognized and/or forecast complex events.

Engine mapping. This operator is admissible to any Java-enabled device running a RTSA. In contrast to NeuroFlinkCEP, Wayeb is purely model- and automaton-driven (PST+FSM) and therefore CPU-bound, with no GPU requirements. Its runtime footprint can be as lightweight as determined by the size of the serialized forecasting artifact (`.spst`) and the pattern automaton. Consequently, it is well suited to fog and edge devices such as ARM64 gateway/dev boards based on Cortex-A53/A57/A72 SoCs, where NeuroFlinkCEP is typically infeasible due to Flink runtime and TensorFlow inference footprint. When placed inside an EdgeProcessingNest operator, the Wayeb JAR, together with its model file, is automatically shipped and executed to the target device consuming events from Kafka and publishing forecasts.

F. EdgeToCloud Optimization operator

The EdgeToCloud Optimization operator (Figure 4) is a nest operator in which users design the logical workflow (Figure 3). Figure 4 shows the operator and its external connections to the Kafka backbone, the optimization service, AI Hub, available RTSA devices and a Flink-enabled site. Inside the operator, users focus on business logic and dataflow (ingestion, preprocessing, ETL, feature extraction, complex event recognition and forecasting). Figure 3 shows an example logical workflow. At execution time, the operator extracts this logical graph and associated operator parameters and builds an engine-agnostic JSON representation. It enriches the logical graph with metadata capturing placment constraints and resource demands. It then serializes and sends the request, while registering callbacks for retrieving responses.

Execution modes and parameters: A central parameter is the *optimization mode*. Users can choose among: (i) *Only Optimize*: obtain the suggested physical workflow without direct deployment, enabling inspection and manual edits, (ii) *Only Deploy*: deploy a saved physical workflow or one returned by Only Optimize and (iii) *Optimize and Deploy*: obtain the physical workflow and ship its parts for execution to the involved devices. Additional parameters control the asynchronous request loop and supply planning context. Table I summarizes the main operator parameters.

Optimization service interface: The optimization service back-end receives logical workflows, searches for efficient deployment plans using a state-of-the-art optimization algorithm [1] and returns assignments of operators to execution engines and physical devices. Communication is fully decoupled and asynchronous: Studio publishes optimization requests via HTTP POST, the service processes requests asynchronously, and results are periodically polled via HTTP GET.

Planning context and end-to-end latency cost model: Upon receiving a request, the service parses the logical workflow description and reconstructs an internal representation of the operator graph. It retrieves infrastructure information (candidate edge nodes and respective RTSAs, characteristics of the cloud/fog-side Flink runtime(s) and topology/latency of network links). Using the dictionary and network files (Table I), it associates logical operators with resource demands/statistics and each network device with capabilities/engine availability. The objective is to find deployment plans providing good end-to-end latency. For each candidate plan, a cost model estimates the overall latency, taking into account properties of Flink and RTSA, along with the network devices they can run on, and the communication latency induced between the involved devices.

Optimization Algorithm: The optimization service relies on DAG*4CEP [1], an A*-alike optimization algorithm that, in accordance with our problem statement (Section II), constructs an optimal physical execution plan for a logical CER/F workflow without enumerating all possible options of operator to execution engine/device combinations. It starts from a topo-

logically sorted logical workflow and incrementally extends a partial physical plan by mapping exactly one additional operator per step to admissible devices/engines, evaluating each candidate with $g(n) = f(n) + h(n)$, where $f(n)$ is the accumulated end-to-end latency cost of the partial plan and $h(n)$ is an admissible heuristic that underestimates the remaining latency cost for a full plan. Candidate partial plans are maintained in a priority queue ordered on $g(n)$. The algorithm repeatedly expands the most promising (based on $g(n)$) plan, until a complete physical workflow positions itself at the front of the queue, at which point it returns the optimal plan. Note that at this point, every other plan standing behind the optimal has a greater $g(n)$ despite the understestimation of $h(n)$. Therefore, all other physical plans in the queue can be safely pruned. Beyond generic placement, DAG*4CEP incorporates CER/F-specific rewritings: (i) *Pattern Decomposition* distributes sub-patterns across devices to avoid raw data transfers, (ii) *Early Filtering* suppresses irrelevant events immediately after neural grounding (synapSEFlow) to reduce downstream load, (iii) *Predicate Reordering* arranges pattern constraints so cheaper, more selective checks are evaluated earlier, reducing state and computation and (iv) *Predicate Pushdown* moves filtering constraints close to upstream Kafka topics to restrict communication. See [1], [6] for details.

Physical plan encoding and construction: Once a plan is returned, the service encodes it as JSON that can be consumed directly by Studio. The plan is translated into: (i) StreamProcessingNest specifications defining the Flink sub-workflows, Kafka inputs/outputs and deployment parameters and (ii) EdgeProcessingNest specifications per fog/edge-side sub-workflow, including mapping of operators to specific fog/edge devices and Kafka topics for ingestion/forwarding. The extension constructs the corresponding Streaming- and EdgeProcessing- Nest operators as shown in Figure 5.

V. An Industry 4.0 Case Study

We now present a real world application scenario from an Industry 4.0 use case. The scenario involves robots moving in a smart factory terrain including physical obstacles and production stations. Robots deliver particles from one production station to the other till full product item compilation. Complex Events (CEs) of interest include: (i) `Collision Forecast` CE: for collisions that are imminent to occur among robots or among a robot and physical obstacles in its way to a production station, (ii) `Station Skipping` CE: a robot detects and moves towards a station, but then heads to another station, missing delivery. The respective logical workflow is presented on the right-hand side of Figure 3.

As shown in the figure, perceptual streams are ingested and then the stream is split into two branches. The upper branch includes a NeuroFlinkCEP operator aiming at recognizing `Station Skipping` CEs. In this operator the full neurosymbolic capacity of NeuroFlinkCEP is exploited, i.e., the neural model converts robot trajectory streams to classified portion movements such as (moving to station X, stopped at station X, stopped away from station X etc), while the symbolic part attemts to match patterns of consequtive (moving to station X) events followed by (moving to station Y) events. The lower branch is oriented towards `Collision Forecast` CEs. The NeuroFlinkCEP operator uses only its neural component to transform raw trajectory streams to symbols and subsequently Wayeb is used to foreacast imminent collisions, i.e., Hard braking, Deadlock, Rotational Movement simple events likely signal an upcoming Collision Detected event within the given time horizon. The two streams of `Station Skipping` and `Collision Forecast` CEs are then unioned and output to a Kafka topic.

Figure 5 shows the result of the CER/F optimization and deployment in the EdgeToCloud Extension. With two RTSAs and one RPi available, the optimizer assigns the part of the workflow up to the split operator (top-right of Figure 5) to the first RTSA which essentially performs the data ingestion and routing process. The two NeuroFlinkCEP operators are assigned for execution to the RPi running Flink (right-bottom of Figure 5). Finally, the forecasting and output sink parts have been assigned to the second RTSA (middle-right of Figure 5).

VI. Conclusion

The EdgeToCloud Extension addresses a practical gap in operationalizing streaming and CER/F analytics across heterogeneous engines and devices of the continuum. The current design offers a variety of practical benefits: ①**Abstraction and usability:** The complexity of the underlying infrastructure is abstracted away from the user. ② **Separation of concerns:** The logical workflow specifies *what* should be executed, while the physical workflow specifies *how* it is executed across the cloud-to-edge continuum. ③ **Operational flexibility:** New edge nodes or cloud clusters can be added without changing the logical design. Physical workflows allow human-in-the-loop by remaining editable by users before submission to further tune optimized workflows when needed. ④ **Edge-to-cloud synergy:** Seamless cooperation between edge, fog and cloud execution is achieved through the EdgeProcessingNest and StreamingNest operators. Our approach reduces the operational burden of multi-node, multi-engine deployments and provides a systematic path to actionable CER/F deployments.

References

[1] O. Ntouni, D. Banelas, and N. Giatrakos, "Neuroflinkcep: Neurosymbolic complex event recognition optimized across IoT platforms," *Proceedings of the VLDB Endowment*, vol. 18, no. 12, pp. 5355–5358, 2025.

[2] E. Alevizos, A. Artikis, and G. Paliouras, "Complex event forecasting with prediction suffix trees," *The VLDB Journal*, vol. 31, no. 1, pp. 157–180, 2022.

[3] Altair Engineering Inc., "Altair AI studio (formerly rapidminer studio)," accessed: 2026-02-10. [Online]. Available: https://altair.com/altair-ai-studio

[4] *Create a deployment file*, Rapidminer real-time scoring (scoring agent) documentation ed., Altair RapidMiner Documentation, accessed: 2026-02-08. [Online]. Available: https://docs.rapidminer.com/9.8/scoring-agent/create-deployment.html

[5] Altair Engineering Inc., "Altair AI hub," accessed: 2026-02-10. [Online]. Available: https://altair.com/altair-ai-hub

[6] E. Streviniotis, D. Banelas, N. Giatrakos, and A. Deligiannakis, "DAG*: A novel A*-alike algorithm for optimal workflow execution across IoT platforms," in *ICDE*, 2025, pp. 807–820.