NeuroFlinkCEP: Neurosymbolic Complex Event Recognition Optimized across IoT Platforms

Ourania Ntouni
Technical University of Crete
ontouni@tuc.gr

Dimitrios Banelas
Technical University of Crete
dbanelas@tuc.gr

Nikos Giatrakos
Technical University of Crete
ngiatrakos@tuc.gr

ABSTRACT

We demonstrate NeuroFlinkCEP, the first framework that integrates neural and symbolic Complex Event Recognition (CER) over a state-of-the-art Big Data platform, also optimizing neurosymbolic CER upon operating over IoT settings. NeuroFlinkCEP receives expressed patterns as extended regular expressions and automatically transforms them to FlinkCEP jobs per device. To enable detection of simple events involved in CER patterns, NeuroFlinkCEP can integrate any neural model in FlinkCEP jobs. To optimally assign operator execution in-network, we incorporate and extend a state-of-the-art IoT optimizer.

PVLDB Reference Format:
Ourania Ntouni, Dimitrios Banelas, and Nikos Giatrakos. NeuroFlinkCEP: Neurosymbolic Complex Event Recognition Optimized across IoT Platforms. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at URL_TO_YOUR_ARTIFACTS.

1 INTRODUCTION

Complex Event Recognition (CER) [6] differentiates itself from conventional stream processing. Instead of leaving to client applications the responsibility of assigning a meaning to the raw outputs of a query, CER Engines ingest streams of Simple Events (SEs) and a set of patterns corresponding to Complex Events (CEs), i.e., interesting phenomena that should be tracked by the application; and monitor if such CEs are progressively revealed by combining ingested SEs. Consider the following pattern, stemming from a smart factory scenario, where raw robot navigation data are monitored:

$$R_{successful\_delivery} := (\neg StationDetected)^* \cdot (\neg StationDetected) \cdot (StationDetected \wedge \neg DeliveryManeuver)^* \cdot (StationDetected \wedge DeliveryManeuver)$$

The SE *StationDetected* occurs when a robot detects a station in the smart factory. The *DeliveryManeuver* SE occurs when the robot is moving at a certain speed, changing directions while maneuvering to approach the detected factory station. The CE $R_{successul\_delivery}$ is satisfied when initially a robot has not detected a station, then it detects one and — having detected the station — it attains the required speed and repeated change of movement direction to approach it. The CER system continuously evaluates the rapidly ingested robot streams, converts them to the aforementioned SEs and deduces a successful delivery CE.

In such a scenario, each SE represents the detection of a behavior that can only be deduced by a machine or neural learning model. The role of the neural model is to receive streams of frames (e.g., from vision, LIDAR, or other contextual cues) and the maneuvering behavior for delivery (which can be quite nuanced to deduce only from positional streams) and provide classification outcomes, i.e., class/symbol $A = StationDectected$ and symbol $B = DeliveryManeuver$, for CER to be possible. Then, a CER engine will ingest the aforementioned SEs (symbols A, B) and evaluate the occurence of the involved CEs. Evidently, such scenarios call for both neural inference and symbolic CER to operate synergistically.

FlinkCEP [7] is the CER API of a state-of-the-art Big Data platform, namely Apache Flink. FlinkCEP focuses on scaling-out the computation to a number of machines in a computer cluster/cloud, working in parallel on partitions of the streams, to speed up continuous analytic outcomes. FlinkCEP provides a CER language of high expressive power [1, 6] in terms of formulating patterns for CEs. However, there are certain barriers to the adoption of FlinkCEP. First, FlinkCEP requires business analysts, who are not necessarily expert programmers, to write functional programming code. Second, pattern expression and parameterization involve cumbersome notation, making the whole code writing process error-prone [6]. Third, with the proliferation of IoT devices as SE producers, the classic paradigm in which we first accumulate raw data at the cloud and then submit a FlinkCEP job is severely suboptimal [8]. For instance, in our running example, sending video frames from robots to the cloud and then performing CER would deplete the available bandwidth, causing network latencies that would prevent the real-time character of the involved applications. What should be done instead, is to ship trained neural models and FlinkCEP jobs to network devices, assign parts of the SE and/or CER process directly on them (e.g. with Raspberry Pi boards which nowadays have both GPU and CPU capacity), and only a subset of SEs and/or CEs should be delivered to the cloud, alerting for the occurred events.

Despite the fact that few previous efforts have integrated neural and symbolic CER [9], no existing approach has enabled neither parallel processing of neurosymbolic CER nor optimized, distributed neurosymbolic CER over IoT settings. This work contributes advancing the state-of-the-art by tackling all the aforementioned challenges. We demonstrate NeuroFlinkCEP, the first framework that integrates neural (aka syb-symbolic) and symbolic CER over a state-of-the art Big Data platform for parallel processing that is also optimized to operate distributedly over IoT settings composed

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

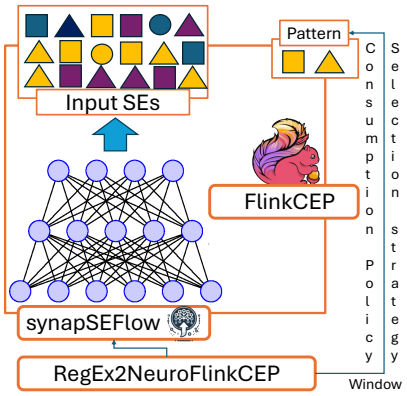

Figure 1: Anatomy of a NeuroFlink-CEP Operator.

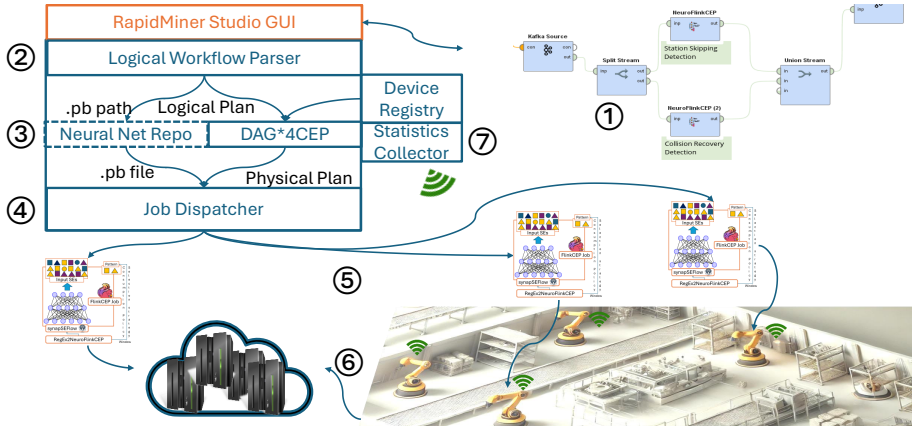

Figure 2: NeuroFlinkCEP Workflow Design, IoT Optimization & Distributed Execution.

of various devices. To alleviate business analysts from the burden of writing FlinkCEP programs, NeuroFlinkCEP receives expressed patterns in the form of extended regular expressions (RegEx) and transforms them to FlinkCEP jobs. To enable detection of SEs and CEs, NeuroFlinkCEP integrates any chosen, domain-specific neural model inside the FlinkCEP job deployed per device. To optimally decide whether operators of the CER workflow should be executed at the cloud or the device network side, NeuroFlinkCEP incorporates a state-of-the-art IoT optimizer [8] for generic streaming workflow optimization and enhances it with CER-specific optimizations [4].

2 ARCHITECTURE

NeuroFlinkCEP's key architectural components are: (i) the RegEx2-NeuroFlinkCEP operator, (ii) the synapSEflow operator and (iii) the DAG*4CER Optimizer. RegEx2NeuroFlinkCEP and synapSEflow operators are nested into a newly introduced *NeuroFlinkCEP operator*. We have developed a NeuroFlinkCEP GUI for graphical workflow design using NeuroFlinkCEP operators and we have incorporated it as an extension to a commercial platform, RapidMiner Studio.

The RegEx2NeuroFlinkCEP operator: receives as input Extended Regular Expressions describing the pattern based on which a CE would be detected, i.e. this nested operator describes the symbolic part of a NeuroFlinkCEP operator. As shown in Figure 1, each such pattern can be parameterized with time windowing constraints as well as selection strategies and consumption policies supported by FlinkCEP. FlinkCEP supports the following SE selection strategies (i) Strict Contiguity: where matching events appear strictly one after the other, (ii) Relaxed Contiguity: where non-matching events appearing in-between the matching ones are ignored, and (iii) Non-Deterministic Relaxed Contiguity: that allows non-deterministic actions between matching events. For event consumption policies, FlinkCEP provides the options: (i) NO_SKIP: produce all matches (ii) SKIP_TO_NEXT: discard partial matches that started with the same CE event (iii) SKIP_PAST_LAST_EVENT: discard partial matches after CE match started but before it ends. SKIP_TO_FIRST[p] and SKIP_TO_LAST[p] are similar to SKIP_TO_NEXT and SKIP_PAST_-LAST_EVENT, respectively, but they use a pattern p to dictate the start (resp. last) event in the CE [7].

The synapSEflow operator: nests the TensorFlow Java API within the NeuroFlinkCEP operator. It receives as input the Tensorflow (.pb) file with a trained neural model for the neural prediction part of the NeuroFlinkCEP operator. The synapSEflow operator loads the trained neural model prescribed and undertakes the responsibility of ingesting the incoming raw streams (e.g, video frames, positional streams in our running example) composing feature vectors. It then feeds these features through the loaded neural model and derives corresponding labels/symbols. synapSEflow also has two more important responsibilities: (i) it directs the simple event outputs of the neural model (depicted as geometric shapes in Figure 1) to the core of FlinkCEP for the pattern matching process to proceed based on the pinpointed SEs, (ii) it listens to a broadcast stream for model updates through Kafka. When a new model is received, it is recorded in that cloud or devices side, local Broadcast State. synapSEflow ensures that local Flink tasks on the cloud or device receive the model and install it, so that all predictions of synapSEflow use the latest version of the trained neural network. An important observation is that, in case no .pb file is specified, the input to a downstream NeuroFlinkCEP operator should be another, upstream NeuroFlinkCEP operator feeding SEs readily available for pattern matching.

The DAG*4CER Optimizer: incorporates a state-of-the-art IoT optimization algorithm, namely DAG* [8] and extends it to DAG*4CER, a variation with plausible CER, FlinkCEP-specific optimizations [4]. We choose to integrate and extend DAG* because it prunes large amount of the operator placement search space, guarantees optimality of the IoT execution plan and has been shown to outperform [8] recent, relevant approaches in the field [2, 3].

The DAG*4CER Optimizer receives as input a logical workflow designed in RapidMiner Studio. This logical workflow incorporates the application logic, but it is deprived of any physical execution details engaging the network of devices. It is composed of various NeuroFlinkCEP operators (each potentially equipped with a different synapSEflow neural model, valid for the SEs of the involved pattern) expressing the rules/patterns to be monitored. The output of the DAG*4CER Optimizer is a physical plan prescribing the network site (cloud or network device) each NeuroFlinkCEP operator should be assigned for execution.

DAG* [8] is an A*-alike algorithm that swiftly outputs a physical execution plan that is guaranteed to be optimal, without exhaustively examining the entire search space. To achieve that, it topologically sorts the logical workflow and progressively examines physical instantiations to network devices for the involved operators. At each step n, the algorithm adds a new operator to the partial physical plan that has been built so far and computes an estimated cost computed as $g(n) = f(n) + h(n)$, where $f(n)$ is the current cost of the partial physical plan and $h(n)$ is a heuristic cost expressing an under-estimation of how much cost will be added to the partial plan to make a full plan, i.e, with all operators assigned to devices. Then, the partial plan is inserted into a priority queue, sorted on $g(n)$. The first partial plan in the queue is then dequeued and the algorithm attempts to expand it by adding a new operator as described above. When the dequeued plan is a full physical plan, the algorithm concludes and outputs the optimal plan found. To output a physical workflow execution graph, instead of a path (that is the output of the original A* algorithm), DAG* imposes two important rules: (i) at each step, only one operator can be examined for physical instantiations to expand a dequeued partial plan, and (ii) no logical operator can be instantiated to any physical implementation unless all of its upstream operators have been included in the currently examined partial plan. Please refer to [8] for details.

However efficient DAG* is in pruning the search space of possible NeuroFlinkCEP operator assignments, it is designed for generic IoT stream processing scenarios. Therefore, it does not incorporate CER-oriented query rewritings that can boost the performance of candidate physical CER plans. In the DAG*4CER Optimizer we extend DAG* with the following CER, FlinkCEP-specific query rewritings during DAG* exploration:

- **Pattern Decomposition**. In case a CE involves SEs sensed on different devices, the existing DAG* would assign the entire pattern evaluation at one device. This may be suboptimal when all other devices will have to communicate their raw data to that device. Our DAG*4CEP provides, to the optimization process, the option to decompose a pattern to its SEs and assign the evaluation of SEs at different devices rather than assigning the evaluation of an entire pattern to one single device. Consider, for instance a pattern `AB+C`. Instead of deploying a NeuroFlinkCEP operator for `AB+C` at one device, DAG*4CEP may decide to assign a NeuroFlinkCEP operator at Device `0` with a neural net at the synapSEflow specialized to detect SEs of type `A` and a NeuroFlinkCEP operator at Device `1`, with a neural net at synapSEflow specialized to detect SEs of type `B` and `C`.
- **Early Filtering** employs selective criteria using the Flink operator `DataStream.filter()`, right after SEs labeled by synapSEFlow become available in Flink, but before they are used in FlinkCEP for pattern matching. It thereby optimizes resource usage by reducing irrelevant event volume, before reaching CER operators like `CEP.pattern()`, if allowed by the pattern.
- **Reordering** involves rearranging pattern evaluation conditions (`.where()` operands) within the CER pattern definition in FlinkCEP itself. This reordering checks less complex, very selective predicates first to promptly rule out non-matching events within the CER engine. Unlike Early Filtering, Reordering does not prune events before they enter FlinkCEP, but optimizes the processing sequence in FlinkCEP. This reduces computational overhead and state complexity in `PatternStream` of FlinkCEP and improves matching performance.
- **Pushing Predicates Upstream** operates like Early Filtering, but for connected NeuroFlinkCEP operators assigned to different devices, i.e., one upstream's operator CEs at one device are the SEs of a connected downstream NeuroFlinkCEP operator at another site. To reduce network communication costs, pushing predicates upstream moves filtering logic directly to source connectors like `KafkaSource` at the upstream operators. It thus eliminates irrelevant events from going downstream, saving considerable network resources and latencies.

The above do not compromise the optimality of the original DAG*, because the DAG*4CER extension does not exclude plans that are destined to be examined by DAG*, but only adds more options by providing query equivalent pattern re-writings [4].

3 USER EXPERIENCE AND DEMO SCENARIOS

Smart Factory, Robotic Scenario: This scenario involves robots moving in a smart factory terrain including physical obstacles and 10 production stations. The mission of the robots is to deliver particles from one production station to the other till full product item compilation. Several GBs of data are provided by DFKI Kaiserlautern in the scope of the EVENFLOW project acknowledged in this work. Neural models are trained via ROS simulations. Examples of Complex Events of interest include: (i) Successful Delivery CE: as in the example of Section 1., (ii) Collision Recovery CE: for collisions occurring between robots or between a robot and physical obstacles, where the robot manages to recover and make its way to a production station, (iii) Station Skipping CE: a robot detects and moves towards a station, but then heads to another station, missing delivery (iii) Prolonged Stop at a Station CE: a robot detects a station and maneuvers to approach it, but it takes too much time without delivering, therefore aborting the approach, (iv) Round-Trip Completion CE: a robot successfully performs a round trip detecting, approaching and successfully delivering production particles across all production stations.

Telecom Scenario: we will use a real, properly anonymized, dataset of Call Detail Record (CDR) data from [5]. A site in the network is either the corporate data center or edge computing nodes (Raspberry Pi, Jetson Nano, Mobile Edge Computing server) at BTS (Base Transceiver Station) near the communication antennas. These capture call metadata from Radio Access Network (RAN) logs and can process event data locally before forwarding it to the core network. Examples of relevant CEs include: (i) Long Call At Night CE: reports long calls to premium locations during night hours, (ii) Frequent Long Calls At Night CE: raises an alarm upon the occurrence of a number of long-lasting calls made to premium locations during night hours per CallerID, (iii) Frequent Long Calls CE: triggers an alarm when multiple calls made to a premium location sum up to a prolonged duration in a day, (iv) Frequent Each Long Call CE: Notifies for a high number of long-lasting calls made to a premium location in a day. Notice that these do not involve predefined values or ranges for "premium locations", "long-lasting", "night hours", "prolonged duration" which depend on various factors such as caller location and past calling behavior. Therefore, involved SEs should be attributed by synapSEflow neural models.

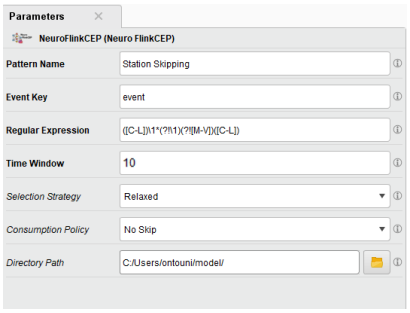

Figure 3: NeuroFlinkCEP Operator Parameterization.

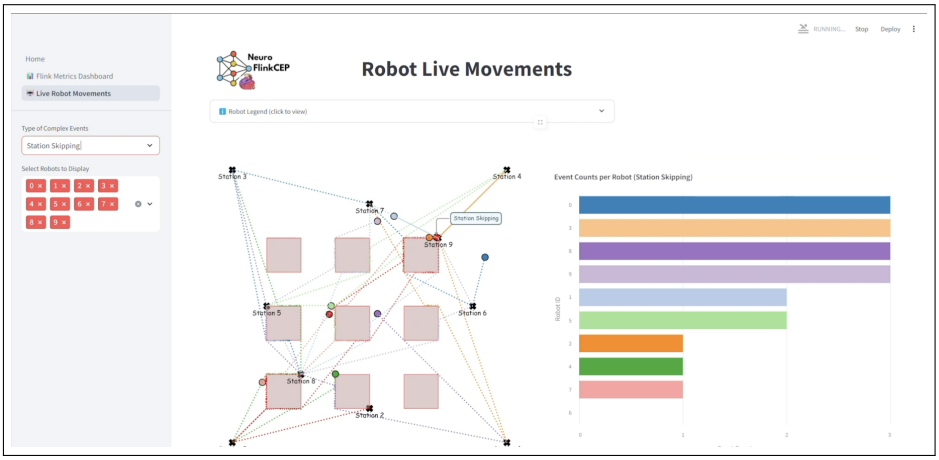

Figure 4: Dashboard for the Smart Factory, Robotic Scenario.

Also note that some of the CEs in the above scenarios ingest other CEs as their own SEs, i.e., interconnecting NeuroFlinkCEP operators, in CER workflows. The DAG*4CER optimizer will have to decide whether each NeuroFlinkCEP operator for the above CER workflows will be placed at some edge device or at the cloud side.

User Experience: During the demonstration, users will be able to choose either scenario and perform the numbered tasks in Figure 2:

① The user interacts with the NeuroFlinkCEP GUI in Rapidminer Studio to design and parameterize their own logical workflows, besides some prebuild ones provided by us, for each application scenario. The user drags and drops each NeuroFlinkCEP operator on a canvas and connects NeuroFlinkCEP operators and Kafka Source/Sinks to define the data flow as shown on the upper right part of Figure 2. As shown in Figure 3, for each NeuroFLinkCEP operator, the user graphically defines the pattern of interest, selection strategy, consumption policy, time window for the nested RegEx2NeuroFlinkCEP operator. Also they specify the `.pb` filepath for the nested synapSEflow operator.

② When the user submits the logical workflow, a Logical Workflow Parser checks its validity. It then converts this logical plan to a JSON file that is fed to DAG*4CEP optimizer and to a Neural Net Repo.

③ The DAG*4CER optimizer detects the available network devices via a Device Registry and examines physical assignments to devices for each NeuroFLinkCEP operator, outputting the optimal physical plan. It also projects the optimized physical plan back to the GUI of RapidMiner Studio. There are 3 options for the user to interact with the DAG*4CER Optimizer: (a) `optimize and deploy`: which instructs the Optimizer to directly feed the optimal plan to the Job Dispatcher, (b) `only optimize`: which instructs the Optimizer to show the suggested physical plan in the GUI for the user to inspect it or change it, before deploying it, (c) `only deploy`: which will feed the workflow, after (b), to the Dispatcher.

④ In ③(a), ③(c), the DAG*4CEP optimizer feeds the physical plan to the Job Dispatcher, while the Neural Net Repo provides the `.pd` files for the neural nets engaged in the CER workflow.

⑤ The Job Dispatcher submits Flink jobs to the network sites according to the assignment of NeuroFlinkCEP operators by DAG*4CEP.

⑥ Detected CEs are continuously visualized in an interactive dashboard. Figure 4 shows the dashboard for the robotic scenario.

⑦ The deployed plan is monitored and statistics including processing and network latency, throughput and other relevant metrics are collected for future DAG*4CEP plan cost estimations.

The user will also be able to bypass DAG*4CER and manually assign operators to network sites (e.g. assign all NeuroFLinkCEP operators at the cloud side) so that the audience can experience the difference between the high latency CE delivery upon centralized processing, versus the optimal execution suggested by DAG*4CER.

Finally, the user will be able to train a neural model and deploy it on currently running NeuroFlinkCEP operators (in their corresponding synapSEflow nested operators, in particular) at runtime.

ACKNOWLEDGMENTS

We thank Fatos Gashi from DFKI Kaiserslautern for providing the data for the Robotic scenario. Research supported by the EU projects CREXDATA under Horizon Europe agreement No. 101092749 and EVENFLOW under Horizon Europe agreement No. 101070430.

REFERENCES

[1] A. Artikis, A. Margara, M. Ugarte, S. Vansummeren, and M. Weidlich. 2017. Complex Event Recognition Languages: Tutorial. In *DEBS*.

[2] X. Chatziliadis, E. Zacharatou, A. Eracar, S. Zeuch, and V. Markl. 2024. Efficient Placement of Decomposable Aggregation Functions for Stream Processing over Large Geo-Distributed Topologies. *Proc. VLDB Endow.* 17, 6 (2024), 1501–1514.

[3] A. Chaudhary, S. Zeuch, and V. Markl. 2020. Governor: Operator Placement for a Unified Fog-Cloud Environment. In *EDBT*.

[4] I. Flouris, N. Giatrakos, A. Deligiannakis, M. Garofalakis, M. Kamp, and M. Mock. 2017. Issues in complex event processing: Status and prospects in the Big Data era. *J. Syst. Softw.* 127 (2017), 217–236.

[5] I. Flouris, V. Manikaki, N. Giatrakos, and et al. 2016. FERARI: A Prototype for Complex Event Processing over Streaming Multi-cloud Platforms. In *SIGMOD*.

[6] N. Giatrakos, E. Alevizos, A. Artikis, A. Deligiannakis, and M. Garofalakis. 2020. Complex event recognition in the Big Data era: a survey. *VLDB J.* 29, 1 (2020).

[7] Apache Flink Project. 2025. *FlinkCEP - Complex Event Processing for Flink.* https://nightlies.apache.org/flink/flink-docs-release-1.20/docs/libs/cep/ Version 1.20.0.

[8] E. Streviniotis, D. Banelas, N. Giatrakos, and A. Deligiannakis. 2025, to appear.. DAG*: A Novel A*-like Algorithm for Optimal Workflow Execution across IoT Platforms. In *ICDE)*.

[9] M. Vilamala, T. Xing, H. Taylor, L. Garcia, M. Srivastava, L. Kaplan, A. Preece, A. Kimmig, and F. Cerutti. 2023. DeepProbCEP: A neuro-symbolic approach for complex event processing in adversarial settings. *Expert Syst. Appl.* 215 (2023).