SN Computer Science (2025) 6:1014
https://doi.org/10.1007/s42979-025-04574-1

ORIGINAL RESEARCH

Pattern-Level Privacy Protection in Event-Based Systems

Majid Lotfian Delouee[1,4] · Victoria Degeler[2] · Peter Amthor[3] · Martijn C. Schut[4,5] · Boris Koldehofe[3]

Received: 15 October 2024 / Accepted: 24 November 2025

Abstract

While privacy-preserving mechanisms aim to protect sensitive information at the attribute level, distributed complex event processing (DCEP) systems remain vulnerable to privacy breaches through the detection of event patterns. Even if individual events are not inherently private, their combination into patterns can reveal sensitive knowledge. This article investigates the integration of pattern-level privacy within the APP-CEP context. We address the challenge of selectively applying obfuscation techniques to event-based systems to safeguard sensitive information. Unlike existing methods, we aim to establish event-stream-agnostic privacy by formulating queries and privacy constraints using CEP-like patterns, and constructing pattern dependency graphs to dynamically select obfuscation techniques that have no consequences on detecting other sensitive patterns, as well as non-sensitive patterns required to provide an acceptable quality of service. Additionally, we model potential adversary's knowledge that could compromise privacy and analyze its influence on the obfuscation process. We evaluated APP-CEP's performance using two real-world datasets: online retail transactions and medical records, replayed as temporally ordered event streams. Our results indicate that APP-CEP effectively balances privacy and utility. By modeling background knowledge, we also successfully prevented adversaries from detecting modifications to the input streams.

Keywords Distributed complex event processing · Stream processing · Privacy · Pattern · Adaptation · Medical dataset

Introduction

The ability to extract meaningful insights from individual data has garnered significant attention across diverse domains, including e-commerce, public healthcare, and the Internet of Things (IoT). These applications often demand timely, distributed data processing to enable proactive or predictive actions. This necessitates addressing a spectrum of latency constraints, ranging from minimal delays to strict near real-time requirements [1]. Distributed Complex Event Processing (DCEP) is a cutting-edge paradigm for real-time analysis of data streams in distributed environments. It involves transforming simple events (e.g., raw IoT data) into more complex events that represent meaningful situations (e.g., queries over sensor data). This transformation is achieved using a set of processing logic known as CEP rules [2]. For instance, a traffic monitoring system might infer road congestion based on the following simple events: an average vehicle speed below 20 km/h and a vehicle density exceeding the normal level.

A primary concern in analyzing data from DCEP systems is *privacy* [3]. This creates a tension between providing

✉ Majid Lotfian Delouee
m.lotfian.delouee@rug.nl

Victoria Degeler
v.o.degeler@uva.nl

Peter Amthor
peter.amthor@tu-ilmenau.de

Martijn C. Schut
m.c.schut@amsterdamumc.nl

Boris Koldehofe
boris.koldehofe@tu-ilmenau.de

1 Bernoulli Institute, University of Groningen, Groningen, The Netherlands

2 Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands

3 Department of Computer Science and Automation, Technische Universität Ilmenau, Ilmenau, Germany

4 Department of Laboratory Medicine, Amsterdam UMC, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

5 Amsterdam Public Health, Methodology & Quality of Care, Amsterdam, The Netherlands

Published online: 10 December 2025

optimal Quality of Service (QoS) and protecting user privacy. For instance, in e-commerce, product managers need to analyze sales data to identify trends, but revealing specific purchase details can compromise customer privacy as the data owners [4]. Similar challenges arise in analyzing medical data [5]. These scenarios involve an honest-but-curious adversary model, where both users and nodes of the distributed CEP middleware may seek to exploit sensitive information. To safeguard data owner privacy, DCEP necessitates *Privacy-Preserving Mechanisms* (PPMs). While many PPMs (e.g., access control) function at the individual event level (i.e., by protecting event attributes), privacy needs often involve intricate event patterns. A pattern is a CEP representation of a complex event, defined by one or more operations on simple events (e.g., filtering specific events) to identify a particular situation. For example, a sequence pattern identifies specific events occurring in a predetermined order across single or multiple streams, such as the purchase of a pregnancy test followed by children's toys. Data owners often seek to conceal sensitive patterns within their data. These patterns are named *private patterns*. In contrast, *public patterns* are non-sensitive and essential for delivering services, such as query results. The accuracy of complex event detection is a crucial performance metric. False negatives (FNs), where real-world events are missed, and false positives (FPs), where non-existent events are detected, are key indicators of a DCEP system's effectiveness.

The proposed work highlights the limitations of a single, statically applied PPM in balancing the competing goals of concealing private patterns while detecting public patterns. It argues that a more effective approach requires a dynamic assignment of various obfuscation techniques to address several key challenges. First, the causal dependencies between complex event patterns necessitate a nuanced solution beyond simple event reordering. This technique can hinder public pattern detection if the reordered stream fails to preserve meaningful correlations. Second, an adversary's background knowledge must be considered when selecting an obfuscation model. For instance, the omission of an event in one query but not another may inadvertently reveal information about the original stream. Third, the dynamic nature of DCEP systems, including changes in queries, privacy requirements, and data sources, demands a flexible approach. Finally, the adversary's increasing knowledge over time further complicates the task of maintaining privacy while preserving public patterns.

In this article, we extend our previous findings on APP-CEP [6] by *(i)* concisely defining and mathematically modeling the key concepts of APP-CEP, *(ii)* enhancing the evaluation of the approach on the webshop dataset by involving more public queries which increases the size and complexity of pattern dependency graphs and obfuscation assignment procedure, and *(iii)* analyzing the behavior of APP-CEP over a medical dataset that has more complex pattern structure leading to denser dependency graphs.

In more detail, this article provides the following contributions:

1. A privacy-preserving mechanism that accepts user-defined privacy requirements in the form of event patterns. By analyzing the data and query patterns, it determines the most effective obfuscation method to protect sensitive information while still providing meaningful results.
2. Our proposed PPM leverages a graph-based obfuscation model selection approach. By modeling pattern dependencies and extracting input stream features, we can effectively predict the optimal switching time between obfuscation models. This dynamic approach helps prevent privacy breaches while ensuring scalability.
3. A threat modeling approach that leverages event dependencies and statistical analysis of event streams to estimate adversary background knowledge.
4. Performance evaluation based on two real-world datasets, replayed as temporally ordered event streams, to show the ability of APP-CEP to boost the performance of the DCEP systems in real-world scenarios.

The remainder of this article is structured as follows. We further detail the problem, particularly for an online shop scenario, and motivate the need for pattern-level privacy in DCEP systems in Sect. "Case study: webshop". We introduce an overview of the APP-CEP system model in Sect. "System model". We formalize the problem statement in Sect. "Problem statement". Section "The APP-CEP design" presents the detailed overview of APP-CEP. The evaluation results of APP-CEP are exhibited in Sect. "Evaluation". The related work is presented in Sect. "Related work". Finally, Sect. "Conclusion" concludes our article and points to our future work.

Case Study: Webshop

In a recent project by the National Statistical Institute of Norway [7], several grocery chains have been ordered to share their receipt data. To protect privacy, this institute claims that by performing *pseudonymization*, the account number, which can identify a person, will be changed to another unique number (i.e., attribute-level access control). However, pattern-level correlation plus background knowledge would enable the adversaries to realize customers' identities, violating their privacy.

In this section, we present a real-world use case demonstrating the application of APP-CEP in a scenario similar to the Norwegian example (see Fig. 1). We focus on an online retailer selling all-occasion gifts, where the transaction stream is analyzed to detect predefined public and private patterns. Webshop managers (i.e., users) can use our platform to identify trends and patterns (i.e., situations of interest). By analyzing customer purchase data, we can detect top-selling products, popular time periods, and other valuable insights. Every customer purchase creates an event in our system, enabling us to track and analyze transactions replayed and processed in temporal order. Traditional privacy-preserving mechanisms, such as state-of-the-art PPMs, often fall short in such scenarios. These methods typically focus on event-level protection, such as obfuscating event attributes. However, our approach provides a more robust solution by considering the entire transaction stream.

APP-CEP allows customers to specify privacy requirements, such as concealing a Christmas party. Sensitive information is represented using CEP patterns (e.g., sequence, conjunction, negation). For instance, a Christmas party might be modeled as *CONJ(Invitation Card, Christmas Ribbon, Christmas Decorations)*. To protect privacy, obfuscation techniques (e.g., dropping Invitation Card events) can be applied. However, this might impact the accuracy of non-sensitive queries (e.g., top-selling products). APP-CEP carefully considers various factors, including pattern dependencies, event distribution, and query complexity, to balance privacy and utility. Its goal is to provide intelligent query responses while safeguarding customer privacy.

Extending our analysis beyond the webshop scenario, we assessed the capabilities of APP-CEP on a medical dataset featuring more entangled private and public pattern sets. This dataset allowed for a more rigorous evaluation of our method under more complex conditions beyond the simple pattern characteristics in the webshop scenario. A detailed description of this medical dataset and its characteristics is provided in Sect. "Evaluation".

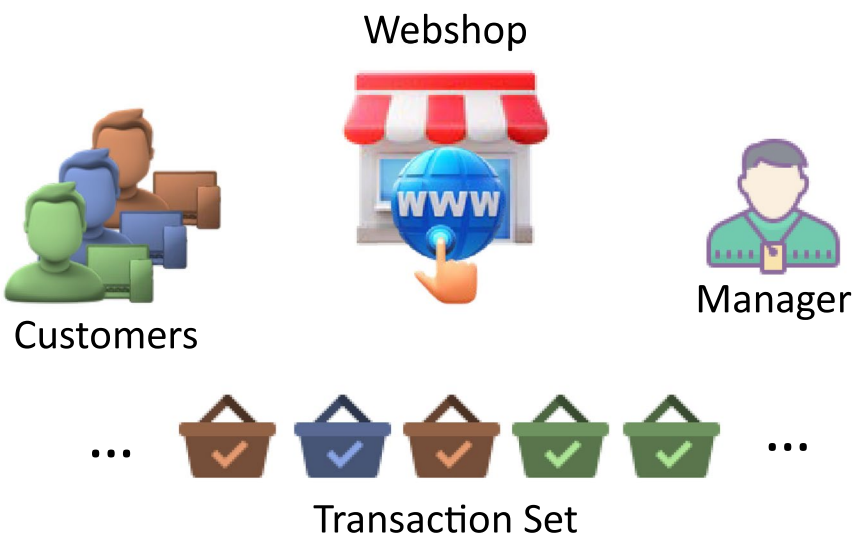

Fig. 1 Webshop scenario. The system aims to protect customers' privacy while answering manager's queries [6]

System Model

This section presents the components of APP-CEP, the model of event sources, and obfuscation models.

APP-CEP Model

A typical DCEP system involves multiple *producers* (e.g., IoT sensors) generating simple event streams. Each stream is associated with one or more *Data Owners* (DOs), protecting their privacy. For instance, a car's location data belongs to its driver. *Consumers* (e.g., applications, services, etc.) express their interests as continuous queries, forming the set $Q = \{q_1, ..., q_n\}$. These queries must be answered by processing incoming events. *Brokers* (e.g., CEP engines) execute event processing tasks (e.g., drop, reorder, union, tamper, etc.) using corresponding *operators* over input streams (live or replayed in our evaluation) and deliver the resulting streams.

A query q_i determines the logic to detect complex events by applying standard CEP operators

over simple events' attributes (e.g., pattern matching). To this end, each detection logic needs to be hosted by a specific operator to be executed.

Moreover, each data owner can articulate its privacy requirements and assign weights to them. This information is represented as a set:

$$PR(DO_i) = \{(pr_1, w_{pr_1}), ..., (pr_m, w_{pr_m})\}$$

where pr_i is a privacy requirement and w_{pr_i} is its associated weight.

In the CEP context, a pattern specifies a composition of events subject to temporal or logical constraints. Formally, we represent a pattern as a tuple $\langle$E, ω, C, W$\rangle$, where E is an ordered sequence of event types, ω denotes the CEP operator (e.g., SEQ, CONJ), C is a set of temporal or attribute constraints, and W is the time window. This formal definition provides the structural basis for the dependencies and privacy–utility formulation discussed in the following section.

To prioritize queries effectively, the system should identify patterns that indicate the importance of certain queries. This is especially crucial in scenarios like healthcare, where life-related queries might have higher priority than others. Table 1 presents the list of notations and their description for the proposed dynamic obfuscation model assignment mechanism.

Table 1 Notations and their meaning

Notation	Description
P	Set of event producers ($p \in P$)
C	Set of event consumers ($c \in C$)
B	Set of event brokers ($b \in B$)
Q	Set of issued queries ($q \in Q$)
Ω	Set of operators ($\omega \in \Omega$)
$\mathcal{O}$	Set of available obfuscation models ($o \in \mathcal{O}$)
$\mathcal{O}^{pr}$	Set of available obfuscation models applicable to private pattern *pr*
DO_i	The *i*th data owner
$PR(DO_i)$	Privacy requirements of *i*th data owner
(pr_i, w_i)	The pair of *i*th privacy requirement and its corresponding weight
$\mathcal{PU}$	Set of public patterns ($pu \in \mathcal{PU}$)
$\mathcal{PR}$	Set of private patterns ($pr \in \mathcal{PR}$)
$\mathcal{D}$	Set of event dependencies ($d \in \mathcal{D}$)
$\rightarrow$	Causal dependency between events
$\parallel$	Parallel occurrence of two or more events
$\neg$	No occurrence of an event
$\bar{X}$	Periodic occurrence of an event
$\nrightarrow$	Infeasible causal dependency between events
$\mathcal{G}$	Set of pattern dependency graphs ($G \in \mathcal{G}$)
g_o	The pattern dependency subgraph for the obfuscation model *o*
$\mathcal{F}$	Set of input stream features ($f \in \mathcal{F}$)
$deg^+_{g_o}(pr)$	The number of outgoing edges from private pattern *pr* in pattern dependency subgraph g_o
α	Mapping solution of obfuscation models to private patterns

Event Source Model

Event sources (producers) generate simple event streams. In IoT scenarios, sensors (e.g., GPS trackers) act as event sources. Based on the number of targets they cover, we categorize event sources into two types:

- **DO-specific:** These sources only generate data about a single target (e.g., a phone's GPS).
- **Multi-DO:** These sources can generate data about multiple targets (e.g., a security camera).

Event sources must be registered with the system due to privacy concerns and we assume they are connected via a wireless network. Data owners can submit queries as CEP consumers, and CEP operators can be hosted on cloud or fog nodes with sufficient computing capabilities.

Obfuscation Model

Obfuscation techniques are near real-time methods that modify event attributes, such as their order, frequency, or values, to conceal sensitive information (private patterns) [8]. These techniques can be applied to event streams before they reach the Complex Event Processing (CEP) middleware. To ensure data integrity, obfuscation should be implemented as close as possible to the data sources, on trusted resources, preventing inconsistencies in the delivered streams.

Among common obfuscation techniques, we consider a PPM is able to hire the following techniques:

- **Suppression:** Removing the occurrence of events by dropping them from the event stream, e.g., removing the event $HomePage\_Visit^{c_1}$ to conceal page navigation behavior of customer c_1.
- **Reordering:** Changing the order of two events by swapping their timestamps, e.g., changing the occurrence time of $Buy_i^{c_1}$ with the event $Buy_j^{c_1}$ to conceal the order of purchases for a specific customer c_1.
- **Injection:** Introducing fake events and inserting them in unique places in the stream, e.g., injecting the event $Buy_j^{c_1}$ multiple times to conceal a specific interest to item *i*. It shows the person c_1 interested in both items, i.e., *i* and *j*.
- **Tampering:** Changing attributes of an event to a wrong value or noise injection in event timestamps, e.g., changing the value of item type *i* in $Add\_To\_Basket_i^{c_1}$ to *j* which produces $Add\_To\_Basket_j^{c_1}$ in order to conceal unhealthily shopping habit of a diabetic customer from an insurance company.

- **Generalization:** Anonymizing the event's attributes by changing to a general value, e.g., generalizing an event $Buy^{c_2}_{apple}$ to $Buy^{c_2}_{fruit}$ to conceal a customer's shopping interests.
- **Hybrid:** A combination of thereof, e.g., suppression of event $Buy^{c_2}_{medicine}$ and injecting event $Buy^{c_2}_{drink}$ to conceal a customer's decease.

Problem Statement

A stream processing system takes as input sets of public and private patterns, which can be defined by domain experts or produced autonomously [9]. A privacy-preserving mechanism dynamically compares incoming events to these patterns, applying obfuscation models in near real-time based on the data within a sliding window [8]. To balance privacy and quality of service, the system must adaptively switch between different PPMs. This process, known as *transition*, allows for optimal privacy protection while maintaining performance [10].

Transitions between privacy-preserving mechanisms must be carefully managed to minimize downtime and resource consumption. Such transitions can create gaps where privacy requirements are not met (i.e., periods with no applied obfuscation). Besides, frequent switching between mechanisms (oscillation) can lead to performance degradation, especially if the pattern sets are scaled up since the number of switches will drastically increase. To address these challenges, it's essential to develop strategies for predicting optimal transition points and minimizing the overhead associated with the process.

Existing systems often statically assess an adversary's background knowledge in the initial stage. However, a more sophisticated adversary (e.g., *Omnipresent adversary*) can learn over time by issuing additional queries and correlating the results. This dynamic nature highlights a limitation of current approaches. To address this, adaptive assignment of obfuscation techniques is essential, considering the adversary's evolving knowledge. A robust privacy protection mechanism should therefore include a means to dynamically model the adversary's background knowledge.

The primary goal of this article is to dynamically acquire data owners' privacy requirements as patterns and then effectively conceal them. The aim is to maximize query service quality while considering potential adversarial knowledge. We evaluate performance using the following metrics:

- For public pattern detection, we computed True Positives (TP_{PUB}) as the set of correctly detected instances of public patterns, and False Positives (FP_{PUB}) as the set of incorrectly identified public pattern matches.
- For private patterns, we utilized metrics like True Obfuscated (TO_{PRIV}), which is the set of successfully obscured instances of private patterns, and False Revealed (FR_{PRIV}), which indicates incorrectly violated private pattern matches.
- For adversarial inference, we enumerate those pattern instances from the set of all private patterns (i.e., *PRIV*) revealed by considering adversary knowledge, presented by the set *ADV*.

Each event type (e.g., true positive, *TP*) is assigned a weight (e.g., W_{TP}) to reflect its relative importance. This weighting scheme accounts for the varying consequences of different event types in different applications. For instance, in some cases, accurately detecting critical events might be prioritized over avoiding false positives, or vice versa. In privacy preservation, the importance of concealing sensitive patterns (private patterns) might outweigh the need to avoid inadvertently creating new ones. Conversely, In a healthcare setting, timely detection of life-threatening conditions (public patterns) is crucial, even if it means compromising some patient privacy (private patterns). The assigned weights allow for this trade-off to be explicitly considered in the system's decision-making. Furthermore, the significance of individual patterns within their respective sets (private or public) is not uniform. Therefore, we assign unique weights to each pattern: w_{to}, w_{fr} for private patterns, and w_{tp}, w_{fp} for public patterns. Additionally, the relative importance of revealed private patterns compared to other event types (e.g., false positives) can be adjusted using W_{Adv}. Furthermore, the priority among private patterns within the adversary list can be controlled with w_a. Here, *tp*, *fp*, *to*, *fr*, and *a* denote individual detected pattern instances belonging to the sets TP_{PUB}, FP_{PUB}, TO_{PRIV}, FR_{PRIV}, and *ADV*, respectively.

According to the mentioned event groups, a *Privacy-Utility Trade-off* (PUT) function is formulated as an optimization problem, enabling the performance comparison of obfuscation models. We extend the objective function formulation in [8] i) to deal with the dynamicity of matched private and public patterns, ii) and quantify the adversary's background knowledge. More formally, the resulting formula is:

$$
\begin{aligned}
Max\Bigg\{ & \left(W_{TP} \sum_{tp \in TP_{PUB}} w_{tp}\right) - \left(W_{FP} \sum_{fp \in FP_{PUB}} w_{fp}\right) \\
& + \left(W_{TO} \sum_{to \in TO_{PRIV}} w_{to}\right) - \left(W_{FR} \sum_{fr \in FR_{PRIV}} w_{fr}\right) \\
& - \left(W_{Adv} \sum_{a \in ADV} w_a\right) \Bigg\}
\end{aligned} \tag{1}
$$

This objective function is versatile and applicable to various application contexts. The first four terms in this equation are computed exactly from detected instances of public and private patterns. The final term reflects adversarial inference, which cannot be directly observed and is instead simulated using event dependency information (cf. Section "Event Dependencies"). If adversarial knowledge is unavailable or intentionally excluded for fair comparison, the final term can be omitted. The weighting scheme enforces a soft priority for privacy; violations of private patterns are penalized more heavily, but limited privacy loss may be tolerated if outweighed by substantial gains in public pattern detection. This ensures that assignments are guided by exactly computed PUT values during stream processing under a balanced privacy–utility principle.

The APP-CEP Design

Figure 2 outlines the core functionalities of our system, APP-CEP. Building partially upon existing methods for converting natural language privacy requirements and situations of interest into CEP-like patterns [11], we assume that data owners and users can input both public and private patterns. The primary objective of APP-CEP is to ① analyze pattern dependencies by Identifying relationships between different patterns, ② determine event dependency sets by defining groups of events that are relevant to each other, ③ extract input stream features by analyzing the characteristics of the data streams being processed, and ④ select appropriate obfuscation techniques via choosing obfuscation methods that satisfy privacy-utility requirements, even without prior knowledge of the actual events in the input streams. The system can dynamically adjust its selection based on the obfuscation results observed during execution. We will delve deeper into the functionality of these four components in the following sections.

Two dynamic databases, the *stream database* and the *obfuscation database*, aid the APP-CEP in the obfuscation process. The stream database stores historical events (e.g., transaction history) for analyzing event dependencies and extracting stream features. The obfuscation database contains various models for applying obfuscation techniques (e.g., dropping the first event). We assume data owners can actively contribute to the obfuscation process by providing event dependencies. For instance, a webshop customer might specify their preferred pick-up location to ensure that location tracking applications obfuscate relevant patterns.

Existing methods for multiple pattern-type privacy protection [8] rely on inspecting actual events to calculate utility values. This allows for selecting the optimal obfuscation technique for each private pattern based on these values. However, these approaches often struggle with adapting to system dynamics, such as changing data owners' privacy

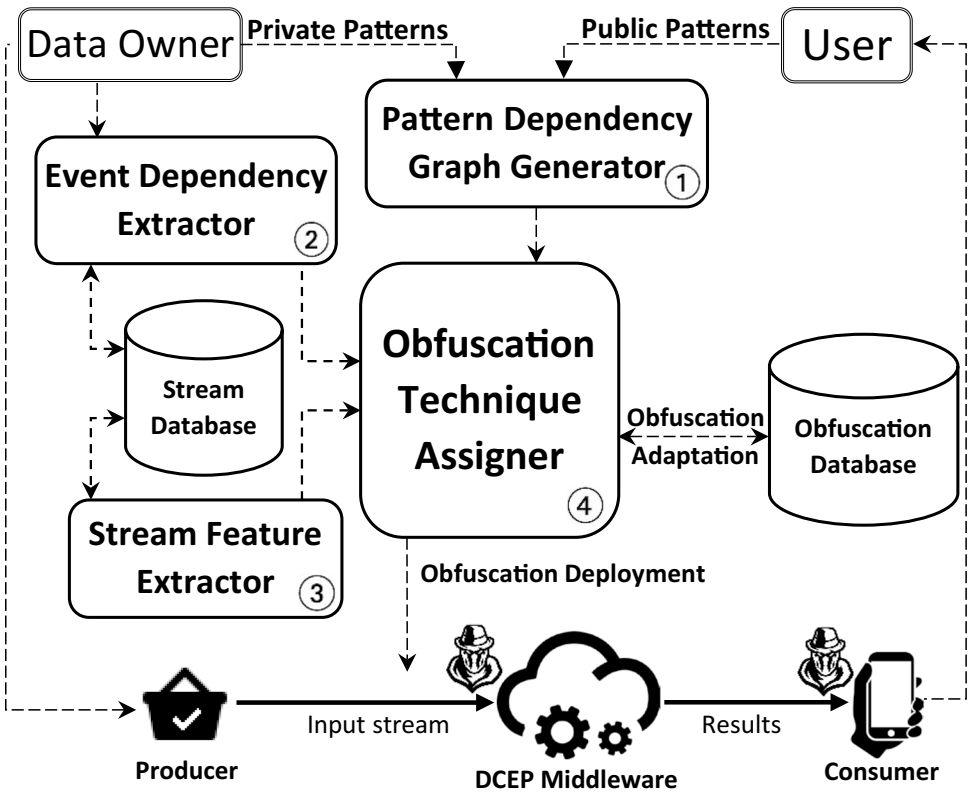

Fig. 2 The APP-CEP architecture [6]. Solid arrows represent event data flowing from producers through the Stream Database and Complex Event Processing (CEP) engine toward consumers. Dashed arrows represent control signals such as pattern updates and reconfiguration triggers. The Event Dependency Extractor derives dependencies from the Stream DB, the Pattern Dependency Graph maintains relationships among patterns, and the Obfuscation Technique Assigner (OTA) dynamically selects obfuscation models to enforce privacy while maintaining utility

requirements. A straightforward solution involves switching between obfuscation techniques whenever a model with higher utility is found. However, this approach can suffer from high overhead during transitions and potential oscillations between techniques, leading to degraded privacy protection. To address these challenges, more sophisticated adaptation mechanisms are needed.

However, one could realize that relying only on the actual events appearing in the input streams causes transition overheads and oscillation issues. To address the challenges of relying solely on actual events in input streams, we propose a proactive approach. By predicting the transition times of obfuscation techniques, we can anticipate data/operator migrations and minimize inefficiencies in both time and resource usage. This prediction can be achieved by analyzing the correlations between patterns (public and private) as well as between events. Extracting features from historical streams allows us to identify patterns and prevent unnecessary obfuscation transitions, leading to improved performance. For instance, estimating the number of pattern matches based on event distribution can help avoid oscillations between different obfuscation techniques, such as rapidly switching between dropping and reordering.

Nevertheless, generating more up-to-date insights about patterns and also events helps APP-CEP to wisely adapt the system to maximize the overall performance of the obfuscation procedure.

Pattern Dependency Graph

In an event-based system, privacy can be ensured by altering the original event streams to obscure sensitive patterns. However, these modifications can unintentionally affect the detection of both public and private patterns that depend on the same streams. We refer to these interdependencies as *pattern dependencies*. A pattern dependency occurs when the detection of one pattern is influenced by the detection or absence of another.

Definition 1 A **Pattern Dependency (*d*)** determines the positive or negative impacts of obfuscating a special private pattern *u* on the matches of pattern *v* (i.e., public or private).

The impact between patterns is determined by shared events. If a basic event *e* occurs in private pattern *u* and either the same or a generalized form of *e* (e.g., 'buy_fruit' generalizing 'buy_apple') appears in pattern *v*, a dependency between *u* and *v* is established. We define the pattern dependency as follows:

$$u \xrightarrow{d} v : \exists\, e \in u \mid (e \in v) \vee (generalization(e) \in v) \tag{2}$$

By analyzing dependencies between private and public patterns, we can predict optimal obfuscation transition times. Specifically, pattern dependency analysis enables a PPM to select obfuscation techniques minimizing negative impacts on other patterns. Knowing the active periods of patterns (i.e., query or privacy requirements validity) allows the method to estimate the optimal timing for switching obfuscation techniques.

Definition 2 A **Pattern Dependency Graph (*G*)**

illustrates an overview of all dependencies between patterns in which each vertex (node) *n* in the graph is a pattern that either belongs to $\mathcal{PU}$ or $\mathcal{PR}$. Besides, an edge (*u*, *v*) in *E* refers to a unidirectional dependency *d* (expressed by $u \xrightarrow{d} v$) which shows that applying at least one of the available obfuscation models on *u* will impact the detection of matches for *v*.

The graph is unweighted: edges indicate the existence of influence but not its magnitude.

The dependency graph is defined as follows:

$$G(V,E) = \begin{cases} V = \{n \mid n \in \mathcal{PU} \cup \mathcal{PR}\} \\ E = \{(u,v) \mid u \in \mathcal{PR} \;\&\; v \in V \;\&\; u \xrightarrow{d} v\} \end{cases} \tag{3}$$

Constructing a dependency graph of public and private patterns provides a clear representation of their interrelationships. This enables us to leverage graph discovery and mining techniques for efficient obfuscation model assignment to private patterns. Considering the following public and private patterns helps to understand the pattern dependency graph generation process better.

$$\begin{aligned} PUBLIC = \{\ & \{pu_1 \mid Buy_{CD}^{p_i} \rightarrow Return_{CD'}^{p_i}\}, \\ & \{pu_2 \mid Buy_{BC}^{*} \rightarrow Buy_{BC}^{*} \rightarrow Buy_{BC}^{*}\}, \\ & \{pu_3 \mid Buy_{AD}^{p_i} \parallel Buy_{FJ}^{p_i}\}\ \} \\ PRIVATE = \{\ & \{pr_1 \mid Buy_{IC}^{p_1} \parallel Buy_{CR}^{p_1} \parallel Buy_{CD}^{p_1}\}, \\ & \{pr_2 \mid Buy_{IC}^{p_1} \parallel Buy_{BC}^{p_1}\}, \\ & \{pr_3 \mid Buy_{AD}^{p_1} \rightarrow Buy_{DM}^{p_1}\}\} \end{aligned} \tag{4}$$

These sample pattern sets (i.e., public and private sets) indicate the following event correlations:

- pu_1: A causal relationship between buying and returning two types of Christmas decorations (CD) with the same customer ID, which shows the customers' interests.
- pu_2: Multiple purchases of the same product (i.e., birthday candle (BC)) by different customers, which helps recommendation systems.
- pu_3: The concurrent ordering of alcoholic drinks (AD) and fruit juice (FJ) with the same customer ID, which detects customers' drinking interests.

- pr_1: Simultaneous purchasing of invitation cards (IC), Christmas ribbons (CR), and Christmas decorations, which reveals a Christmas party.
- pr_2: The concurrent purchase of invitation cards and birthday candles, which reveals a birthday party.
- pr_3: A causal ordering dependency of alcoholic drinks and diabetic medicine (DM) reveals a bad lifestyle for a diabetic person.

The respective pattern dependency graph for the mentioned patterns in Eq. 4 is depicted in Fig. 3. Here, the submitted privacy requirements belonging to an individual customer (i.e., p_1) have been used to define private patterns. Besides, queries represented as public patterns are generally defined, not for a specific customer ID. Signs ($\rightarrow$) and ($\|$) indicate the causal dependency in a sequence pattern and parallel occurrence of events in a conjunction pattern, respectively.

To construct a pattern dependency graph, we consider each graph node as a pattern. For every private pattern, unidirectional arrows are drawn from it to all dependent patterns, both public and private. A pattern is considered a *neighbor* of a private pattern *pr* if there's a direct connection (edge) from *pr* to that pattern. For instance, if we plan to obfuscate private pattern pr_1, suppression of event $Buy_{IC}^{p_1}$ helps to obfuscate private pattern pr_1 (positive impact). However, suppression of the event $Buy_{CD}^{p_1}$ will result in false negatives in the detection of public pattern pu_1 (negative impact).

The graph representation of pattern dependencies provides valuable insights for selecting appropriate obfuscation models. For instance, the number of outgoing edges from a node indicates how many other patterns would be affected if that node were to be obfuscated. If a node is disconnected (i.e., has no neighbors), it's independent of other patterns. This simplifies the obfuscation process, making a static model suitable until the node's connectivity changes in the updated graph. Any obfuscation model that meets the Privacy-Utility trade-off goal can be chosen in such cases.

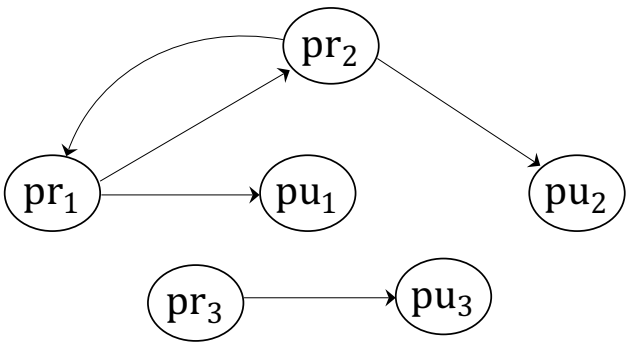

Fig. 3 The global pattern dependency graph generated for pattern sets defined in Eq. 4 [6]. Edges are directed (unidirectional) and unweighted

Definition 3 ZOD node. In a pattern dependency sub-graph g_o, the out-degree of a node (i.e., deg^+) equals the number of its outgoing edges. A zero Out-Degree (ZOD) node is a private pattern whose obfuscation procedure does not influence the detection of other patterns; its deg^+ equals zero. We defined a ZOD node as follows:

$$ZOD(g_o) = \{u \mid u \in \mathcal{PR} \cap V(g_o) \ \& \ \ deg^+(u) = 0\} \quad (5)$$

On the other side, if a private pattern has one or more outgoing edges in the global dependency graph, an alternative solution would be to break down the global dependency graph into obfuscation-specific sub-graphs (e.g., suppression(e_1) sub-graph, which drops the first event of all matches). This way, APP-CEP can find a sub-graph in which this specific node has no neighbors. For instance, Fig. 4 shows the generated sub-graphs of four different obfuscation models. Here, nodes pr_1 and pr_3 are disconnected nodes in the Suppression(e_2), and pr_2 has no neighbors in the Suppression(e_1). Therefore, these obfuscation models are potential candidates for corresponding private patterns. Note that Reorder(e_1, e_2) is not among available obfuscation models for pr_1 and pr_3 because the conjunction operator is used in the definition of those patterns between the first and second events, and such obfuscation model is not able to conceal these patterns. Similarly, Tamper(e_3) is not applicable for patterns of length 2, e.g., pr_2 and pr_3.

By construction, edges in *G* always originate from a private pattern and terminate at either another private pattern or a public pattern; no edges emerge from public patterns. As a result, cycles can only occur in the special case where two private patterns share common events, producing mutual dependencies. In such cases, the corresponding subgraphs have a degree greater than zero. The assignment algorithm filters out such subgraphs from the solution space, ensuring that cycles do not affect the correctness of the obfuscation assignment. We note that cycles may, in principle, have either positive or negative effects. For example, obfuscating both private patterns by dropping their common event might successfully protect them without influencing other public patterns (positive effect), but in other cases, such dropping could harm the detection of some public patterns (negative effect). However, the magnitude and direction of such effects depend on the number of public pattern instances currently present in the stream, which changes over time. Exploiting these effects would therefore require frequent updates of the dependency graph and continuous reassignment of obfuscation techniques. Such frequent transitions can themselves degrade utility and stability. For this reason, APP-CEP deliberately selects the safe option with zero edges, which minimizes transitions and ensures consistent protection across dynamic conditions.

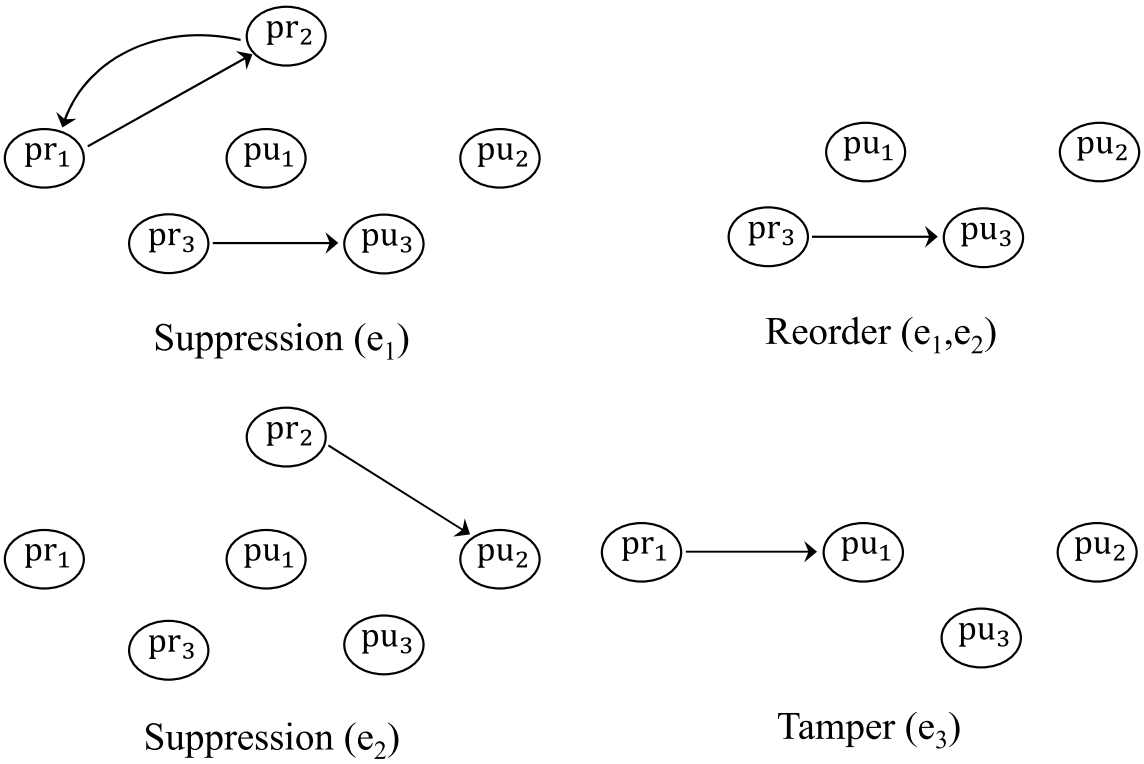

Fig. 4 The alternative pattern dependency sub-graphs for the global dependency graph illustrated in Fig. 3 [6]. Edges are directed (unidirectional) and unweighted

Event Dependencies

Modeling adversaries' background knowledge remains a challenging problem in privacy research. While some studies have explored statistical prediction of specific privacy attacks (e.g., [8]), they often overlook the potential value of data owner information. For instance, an adversary might infer the inter-arrival times of events from historical data and use this knowledge to reconstruct the original stream by manipulating suppression probabilities. However, incorporating data owner information could enhance the accuracy and effectiveness of adversary models.

To effectively protect privacy, APP-CEP allows for dynamic specification of event dependencies. This enables modeling potential relationships between events that an adversary could exploit to infer sensitive information. For example, if a customer frequently buys two products together, the system can avoid publishing one without the other to prevent privacy breaches. In practice, event dependencies in the proposed approach are obtained from three complementary sources: user-provided rules submitted at runtime, dependencies inferred from historical streams by the Event Dependency Extractor (using the Stream Database), and expert-defined infeasible combinations. Four types of event dependencies can be defined for consideration during the obfuscation assignment process.

Causal Dependency (→)

In this category, a relationship between two events can be specified in two ways: firstly, one event is the reason for occurrences of the other (i.e., cause and effect), and secondly, one event always happens before the other. For example, in our use-case scenario, a causal dependency can be derived from history as: CD ($Submit\_Order_i^{c_1}$, $Payment\_Successful_i^{c_1}$) which means the event of submitting an order is the cause, and the successful payment event is the effect. Also, a customer c_1 might express that he always checks the available balance before adding a product p to the shopping basket. Therefore, the customer c_1 can provide a causal dependency to the system as: CD ($Check\_Balance^{c_1}$, $Add\_To\_Basket_i^{c_1}$)

In these cases, APP-CEP considers the dependency, e.g., does not suppress the cause event and keeps the effect event or reorders these two events.

Parallel Occurrence (||)

Special events (two or more) happen always, or at least with a high probability, together or within a short timeframe. To optimize obfuscation, these events should be treated as a *single combined event*. As an example in the proposed webshop scenario, a customer c_1 might consistently purchase products *p* and *q* together. This leads to a parallel occurrence of the corresponding two events, represented as: *PO* ($Add\_To\_Basket_p^{c_1}$, $Add\_To\_Basket_q^{c_1}$).

Infeasible Events ($\nrightarrow$)

Another type of event relationship also shows the impossible occurrences of one or several events or patterns. For instance, when a customer c_1 is navigated to the bank payment page as a result of the $Submit\_Order$ event with id *i*, it is not possible to observe two events of $Payment\_Successful$ and $Payment\_Rejected$ afterward for this special order submission in the stream, that can be expressed by:

$IE[CD(Submit\_Order_i^{c_1}, Payment\_Successful_i^{c_1}), CD(Submit\_Order_i^{c_1}, Payment\_Rejected_i^{c_1})]$

Such dependencies usually cannot be expressed by a data owner, but require help from domain experts in the application scenario.

Periodic Events ($\bar{X}$)

Beyond event dependencies, statistical analysis of specific event types for individual data owners can reveal valuable insights. We assume that a malicious actor has the necessary computational power and access to historical data to identify statistical correlations. For example, a customer c_1 might have a shopping habit of buying a special product *p* each day. Hence, such a habit can be expressed by:

PE ($Buy_p^{c_1}$, *day*) Therefore, applying obfuscation techniques that involve removing or modifying events (such as suppression, tampering, or generalization) can compromise privacy by revealing the stream modification to an adversary. However, reordering techniques might still be viable in these situations.

Stream Features

Even with knowledge of data owners' privacy requirements and the relationships between patterns and events, the dynamic nature of input streams introduces another crucial factor. The number of matches for a pattern significantly influences its relevance and the need for obfuscation.

For instance, a "Christmas party" pattern is more likely to be revealed during the Christmas season. Therefore, obfuscating a private pattern that indirectly reveals a Christmas party at a non-Christmas time (e.g., July) might be acceptable, as the probability of detecting such a party is low during those periods.

To effectively assign obfuscation models, APP-CEP must extract features from the input streams. This involves analyzing the stream database and incoming stream data (live or replayed in our evaluation) to adapt features dynamically. A key feature is the average number of matches for each pattern. For example, the number of matches for a "Christmas party" pattern could be calculated weekly, influencing the weight assigned to this private pattern in the PUT function

Beyond pattern-based features, event-related stream characteristics offer significant advantages for selecting effective obfuscation models. For example, event distribution can provide a more accurate weight for each suppression model associated with a private pattern [12]. If a specific event *e* within a private pattern *pr* occurs frequently in historical data, suppressing *e* may lead to a higher false negative rate (FN) in detecting neighboring patterns that also contain *e*.

This concept can be extended to subsets of a private pattern. For instance, analyzing the distribution of two-event subsets can help determine the effectiveness of reordering-based obfuscation. By considering these event-related features, we can tailor the obfuscation process to minimize privacy risks while preserving data utility.

Obfuscation Technique Assignment

The *obfuscation technique assigner* module, the central component of APP-CEP, serves as the system's decision-maker. It integrates data from three key modules: ① pattern dependency graph generator which constructs a graphical representation of the relationships between all types of patterns, ② event dependency extractor which identifies dependencies between events within the data streams, and ③ stream feature extractor which extracts relevant features from the incoming data streams. By analyzing these inputs, the obfuscation technique assigner determines the most appropriate obfuscation model for each private pattern, ensuring that APP-CEP can effectively protect sensitive information while maintaining data utility.

In Algorithm 1, the system initializes sets of public and private patterns, along with event dependencies derived from the stream's history and available obfuscation operators (line 1). As new public or private patterns are identified, the system dynamically updates the obfuscation technique assignment (lines 2-7). Additionally, the data owner can input event dependencies at runtime (lines 8-10). The *OT_Assigner* function constructs pattern dependency graphs for each obfuscation model, regularly updating them based on changes in both public and private patterns. Furthermore, it

extracts relevant input stream features from historical data and continuously updates these features using the current stream data (lines 12-13). This ensures that the obfuscation techniques remain aligned with the evolving privacy requirements and data characteristics

According to the event dependency set and stream features, for each private pattern *pr*, APP-CEP performs the following tasks: first, it removes those obfuscation models whose obfuscation can be realized by an adversary (lines 16-20). For example, if the result of reordering violates a causal dependency, such an obfuscation model will be removed from the list of available obfuscation models. In the remaining obfuscation model space (i.e., $\mathcal{O}^{pr}$), APP-CEP looks for a member whose dependency graph has a node with an out-degree (i.e., $deg^{+}_{g_o}(pr)$) equal to zero for *pr*, so-called *zero-out-degree* (ZOD) node (lines 22-26). One of the obfuscation models containing such a node will be assigned to the *pr*. A random obfuscation model will be assigned to a *pr* if no ZOD node has been found for this private pattern (lines 27-32). Finally, the assigned obfuscation model for each private pattern is determined and will be updated upon any change in patterns or event dependencies (line 33).

1: **Initialization:**
   $\mathcal{PU}$, $\mathcal{PR} \leftarrow \varnothing$, $\mathcal{D} \leftarrow$ Event Dependencies, $\mathcal{O} \leftarrow$ Obfuscation Techniques;
2: **upon** $(Submit(Q_i))$ **do**
3: $\mathcal{PU} \leftarrow \mathcal{PU} \cup$ CEP-Pattern(Q_i); ▷ Query
4: Obfuscation_Assigner ($\mathcal{PU}$, $\mathcal{PR}$, $\mathcal{D}$);
5: **upon** $(Submit(Pr_i))$ **do**
6: $\mathcal{PR} \leftarrow \mathcal{PR} \cup$ CEP-Pattern(Pr_i); ▷ Priv Req
7: Obfuscation_Assigner ($\mathcal{PU}$, $\mathcal{PR}$, $\mathcal{D}$);
8: **upon** $(Submit(d_i))$ **do**
9: $\mathcal{D} \leftarrow \mathcal{D} \cup d_i$; ▷ Event Dependency
10: Obfuscation_Assigner ($\mathcal{PU}$, $\mathcal{PR}$, $\mathcal{D}$);
11: **function** OBFUSCATION_ASSIGNER($\mathcal{PU}$,$\mathcal{PR}$,$\mathcal{D}$)
12: $\mathcal{G} \leftarrow$ Pattern Dependency Graphs;
13: $\mathcal{F} \leftarrow$ Input Stream Features;
14: **for** $pr \in \mathcal{PR}$ **do**
15: $\mathcal{O}^{pr} \leftarrow \mathcal{O}$;
16: **for** $o \in \mathcal{O}$ **do**
17: **if** o violates $\mathcal{D}|\mathcal{F}$ **then**
18: $\mathcal{O}^{pr} \leftarrow \mathcal{O}^{pr} - o$;
19: **end if**
20: **end for**
21: $\widehat{\mathcal{O}}^{pr} \leftarrow \varnothing$;
22: **for** $o \in \mathcal{O}^{pr}$ **do**
23: **if** $deg^{+}_{g_o}(pr) = 0$ **then**
24: $\widehat{\mathcal{O}}^{pr} \leftarrow \widehat{\mathcal{O}}^{pr} \cup o$;
25: **end if**
26: **end for**
27: **if** $\widehat{\mathcal{O}}^{pr} \neq \varnothing$ **then**
28: $o_{pr} \leftarrow \widehat{\mathcal{O}}^{pr}_1$;
29: **else**
30: $o_{pr} \leftarrow Random(\mathcal{O}^{pr})$;
31: **end if**
32: **end for**
33: $\alpha \leftarrow \{(pr_1, o_{pr_1}), ..., (pr_n, o_{pr_n})\}$; ▷ Solution
34: **end function**

Algorithm 1 Obfuscation technique assignment

More formally, the solution α can be represented as follows.

$$\alpha = \left\{ (pr, \Phi(pr)) \mid pr \in \mathcal{PR} \,\&\, o \in \mathcal{O} \,\&\, n = |\mathcal{PR}| \right\} \tag{6}$$

and,

$$\Phi(pr_i) = \begin{cases} o & \text{if } \exists\, o \in \mathcal{O} : \{\, pr_i \in ZOD(g_o) \,\& \\ & \quad \nexists\, d \in \mathcal{D} : o \text{ violates } d \,\& \\ & \quad \nexists\, f \in \mathcal{F} : o \text{ violates } f \,\} \\ Random(\mathcal{O}^{pr_i}) & \text{otherwise} \end{cases} \tag{7}$$

While the ZOD node selection approach may not always yield the optimal QoS, it offers a balanced solution by ensuring both effective privacy obfuscation for private patterns and efficient detection of public patterns. Calculating the exact Privacy Utility Trade-off (PUT) value can be time-consuming, as it requires precise matching counts for each input stream. To mitigate this, we propose a heuristic approach that estimates the PUT value based on statistical approximations or representative samples. In cases where a ZOD node cannot be found, we believe that randomly assigning an obfuscation model is a reasonable strategy. While not ideal, it provides a fallback mechanism to ensure that some level of privacy protection is applied, even in challenging scenarios. We note that the randomized fallback may occasionally result in lower utility compared to the optimal or ZOD-based choices. However, this impact is bounded because fallback is triggered only when no ZOD node exists in the candidate subgraph, which is rare in practice. Moreover, since fallback assignments are short-lived and the system continuously re-evaluates obfuscation decisions, their effect on long-term performance is limited. The design thus favors guaranteed privacy enforcement over marginal utility gains in these edge cases.

It is important to note that the obfuscation assignment problem is combinatorial and has been shown to be NP-hard [12]. As a result, computing the exact optimal assignment at runtime is intractable in CEP settings where low latency is essential. APP-CEP therefore focuses on efficient heuristics that achieve stable performance. While no formal approximation ratio is provided, our evaluation demonstrates that APP-CEP consistently outperforms graph-based and window-based baselines, indicating that the heuristic remains close to practical optimality under real-world workloads.

Runtime assignments are therefore guided by the PUT formulation introduced in Sect. 4, which provides a formal and reproducible basis for balancing privacy and utility.

Complexity. Let $n_p = |V|$ denote the number of patterns (nodes) and $n_e = |E|$ the number of dependency edges in the graph $G = (V, E)$. Let k be the maximum number of events per pattern (a small constant).

$$\begin{aligned} \text{Dependency extraction}: &\ O(n_p \cdot k), \\ \text{Graph construction}: &\ O(n_e), \\ \text{Candidate filtering}: &\ O(n_p + n_e), \\ \text{Fallback assignment}: &\ O(1). \end{aligned}$$

Hence the total cost per decision step is

$$O(n_p \cdot k) + O(n_e) + O(n_p + n_e) + O(1) = O(n_p \cdot k + n_p + 2n_e + 1).$$

Since k is constant and Big-O ignores constant factors,

$$O(n_p \cdot k + n_p + 2n_e + 1) = O(n_p + n_e).$$

Defining $n := n_p + n_e$, both time and space are $O(n)$.

In the worst case of a dense graph where $n_e = O(n_p^2)$, the bound becomes $O(n_p^2)$; in our datasets the graphs are sparse.

Evaluation

The evaluation investigates the following questions:

1. Can graph-based pattern-level privacy provide a trade-off between privacy concerns and the system's promised QoS?
2. What limitations exist when applying pattern-level privacy in real-world scenarios?

We established a single Virtual Machine (VM) running Ubuntu 20.04 on Oracle VM VirtualBox Manager, allocating 6 CPU cores and 24GB of RAM. The APP-CEP prototype is implemented in Java on top of the Apache Flink streaming framework, using the *FlinkCEP* library [13] for event pattern detection under event-time semantics. Event streams are ingested from CSV files and replayed through Flink's source operators, while the dependency graph construction and obfuscation assignment logic are implemented as custom stateful operators within the Java codebase. To assess our approach, we analyzed two publicly available real-world datasets.[1]1. The first dataset is composed of transactions of an online retailer selling all-occasion gifts [14]. A part of the first dataset, including 12000 customer purchase records, was selected for this evaluation. We selected the 50 most frequent patterns (length 3) from which public and private pattern sets were randomly formed. We also performed another statistical analysis

[1] https://github.com/majid-lotfian/APP-CEP.

Table 2 Partial list of defined public and private patterns used to analyze the medical dataset: public patterns representing useful inquiries in the medical system, and private patterns highlighting situations that have consequences such as increased insurance premiums for patients

#	Public/private	Pattern definition
1	Public	Outpatient Visit followed by Laboratory Test within 30 days
2	Public	Outpatient Visit followed by Medication Prescription within 30 days
3	Public	Emergency Visit followed by an Outpatient Visit within 30 days
4	Public	A hospital admission followed by diagnostic imaging (CT, MRI) within the same stay
5	Public	A medication prescription followed by a refill within 90 days
6	Private	An outpatient visit followed by any kind of readmission within 30 days
7	Private	An emergency visit followed by a diagnosis of a chronic disease within 30 days
8	Private	An emergency visit followed by ICU admission within 30 days
9	Private	Admission for surgery followed by readmission within 30 days post-discharge
10	Private	An outpatient visit followed by a prescription of high-cost medication within 30 days

to extract the most parallel purchased items and periodically purchased items as dependencies considered for the adversary's background knowledge.

2. The second one is a medical dataset with information concerning hospital records of patients diagnosed with diabetes [15]. The dataset represents ten years (1999-2008) of clinical care at 130 US hospitals and integrated delivery networks. Each row concerns hospital records of patients diagnosed with diabetes, who underwent laboratory, medications, and stayed up to 14 days. We formed 20 patterns for public and 10 for private patterns by consulting with a medical expert and randomly selecting patterns from them for evaluation that are partly represented in Table 2. We also defined 8 event dependencies (two for each dependency type).

Both datasets were processed as continuous event streams during evaluation. For the retail dataset, we replayed events using their original timestamps to preserve the temporal structure of transactions. For the medical dataset, each hospital encounter was treated as an event and ordered chronologically according to the dataset records, which were collected sequentially from operational EHR systems over a ten-year period. In both cases, replaying the datasets in event-time order preserves the temporal dependencies that are essential for pattern detection. Our focus is therefore on evaluating privacy–utility trade-offs in streaming settings, rather than on measuring end-to-end system latency, which is outside the scope of this work.

In each experiment, we randomly selected subsets of public and private patterns from the defined pool for that dataset (50 frequent length-3 patterns in the retail dataset; 20 public and 10 private patterns in the medical dataset, partly chosen with expert input). To reduce the effect of randomness, we repeated this sampling process 10 times and report averages. The number of public patterns in each experiment is shown in the figures (e.g., 3 PU, 6 PU, etc.). Pattern lengths are fixed as specified in the dataset descriptions.

Given the novelty of modeling adversarial background knowledge and graph-based obfuscation assignment, we selected two baseline approaches for comparison.

- **Window-based:** This approach simulates existing state-of-the-art mechanisms (e.g., [8]) by dividing the input stream into fixed-size windows. It selects an obfuscation technique for each private pattern match based on its expected utility, considering the potential matches for all patterns after applying the technique.
- **Graph-based:** Similar to APP-CEP, this approach selects obfuscation techniques based on pattern dependencies. However, it does not factor in adversarial background knowledge, relying solely on the pattern dependency graph or OT-specific sub-graphs.

Evaluation Results for Webshop Dataset

Figure 5 shows the total percentage of revealed private patterns, considering both matches discovered by the system and those known to the adversary. As the number of public patterns increases, APP-CEP consistently outperforms other approaches by revealing significantly fewer private patterns. Even with more queries, Graph-based and Window-based techniques exhibit variability, while APP-CEP maintains a consistently low revelation rate of around ten percent. The random assignment of obfuscation techniques, used when the system fails to locate a suitable ZOD node in the global or sub-graphs, is reflected in the performance decline observed in the last three bars. Addressing this limitation with a more intelligent approach would be a significant challenge but is crucial for ensuring the scalability and effectiveness of our future work.

The *F1-score*, a widely used metric in machine learning [16], provides a balanced evaluation of a classifier's performance by combining *Precision* and *Recall* [17]. In stream processing, it can be employed to compare mechanisms based on their overall accuracy. Regarding public pattern detection accuracy, our experiments revealed that increasing the number of involved queries led to a gradual decline in performance for all three approaches (Fig. 6). However,

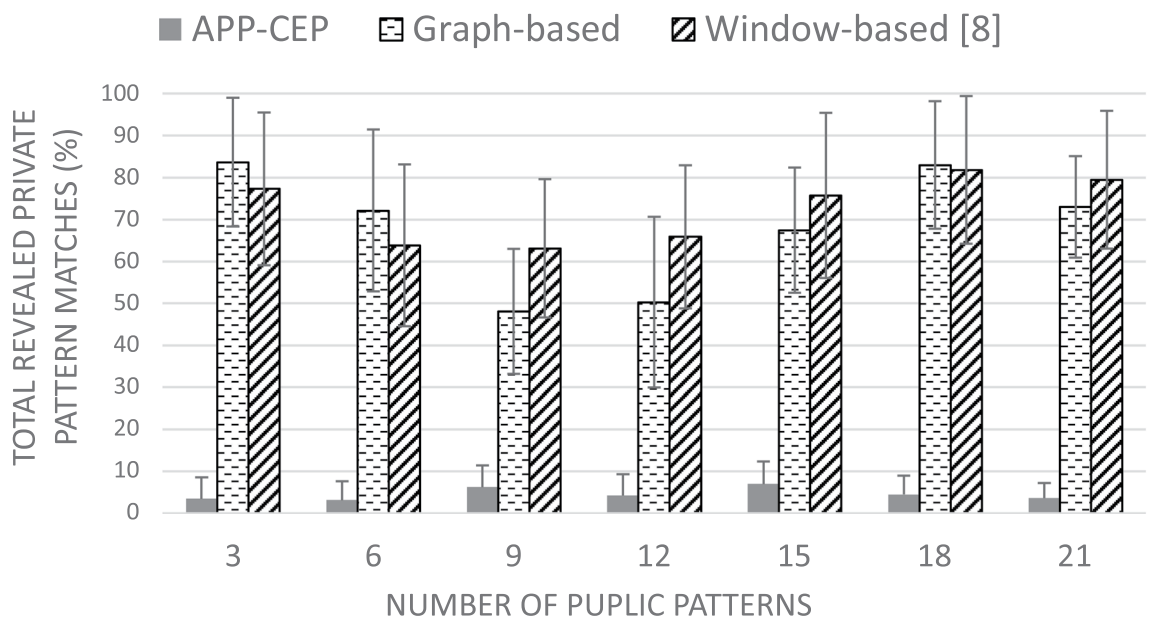

Fig. 5 Percentage of total revealed private pattern matches Vs. The increasing number of public patterns for a set of 3 private patterns for the webshop dataset [6]

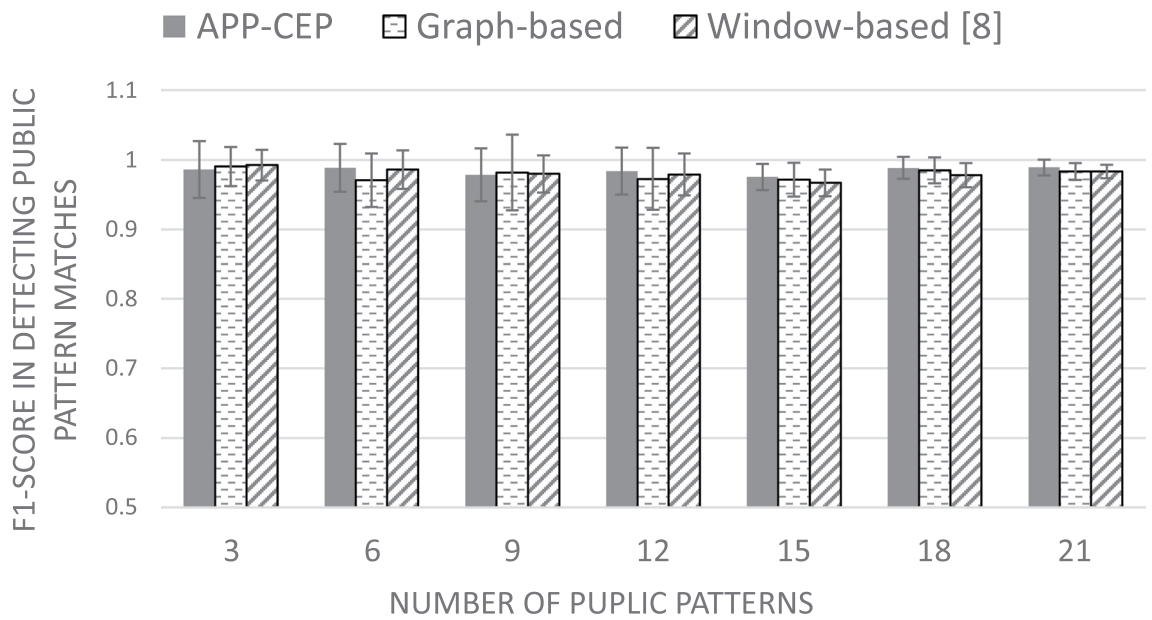

Fig. 6 F1-score in detecting public pattern matches Vs. The increasing number of public patterns for a set of 3 private patterns for the webshop dataset [6]

APP-CEP consistently outperformed or matched the Window-based approach, demonstrating a slight improvement over existing methods, even under these challenging conditions.

As illustrated in Fig. 7, the performance of APP-CEP significantly outperforms the Graph-based and Window-based approaches in terms of revealing private pattern matches. Interestingly, increasing the number of dependencies considered in query processing leads to a slight, yet unexpected, increase in the percentage of revealed matches for all three approaches. However, this increase remains negligible for APP-CEP compared to the performance of the Graph-based and Window-based approaches.

The Graph-based and Window-based approaches exhibit similar performance in Fig. 7, with the Graph-based approach revealing slightly more private pattern matches. Protecting against a more knowledgeable adversary, represented by increased dependencies, presents a significant

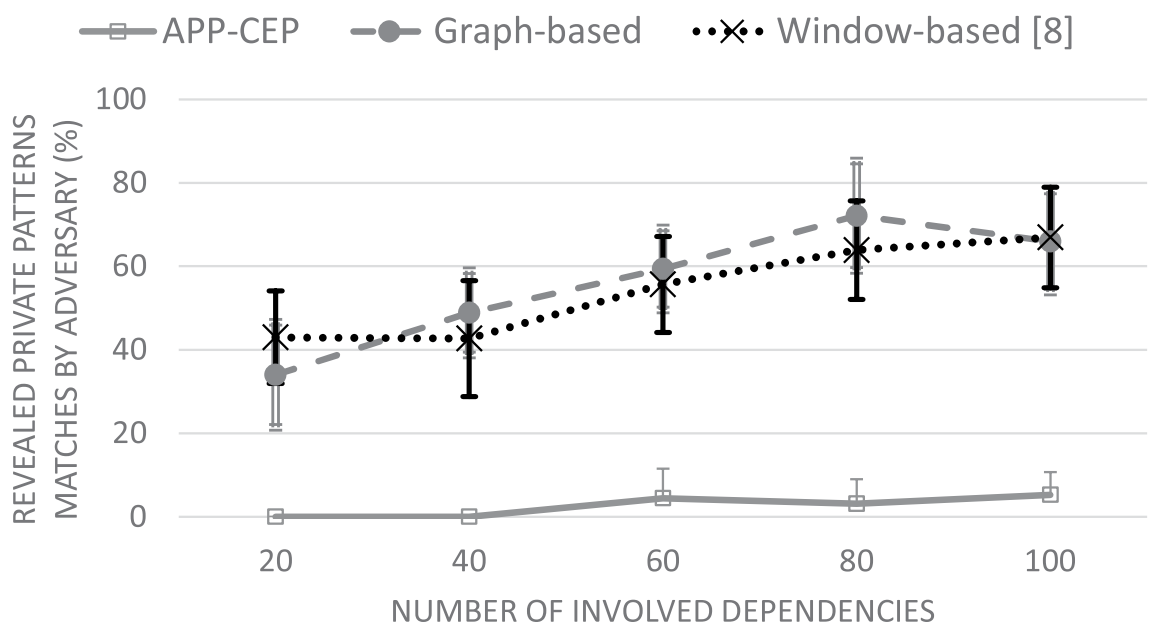

Fig. 7 Percentage of revealed private pattern matches by adversary Vs. The increasing adversary's background knowledge (number of dependencies) for the webshop dataset [6]

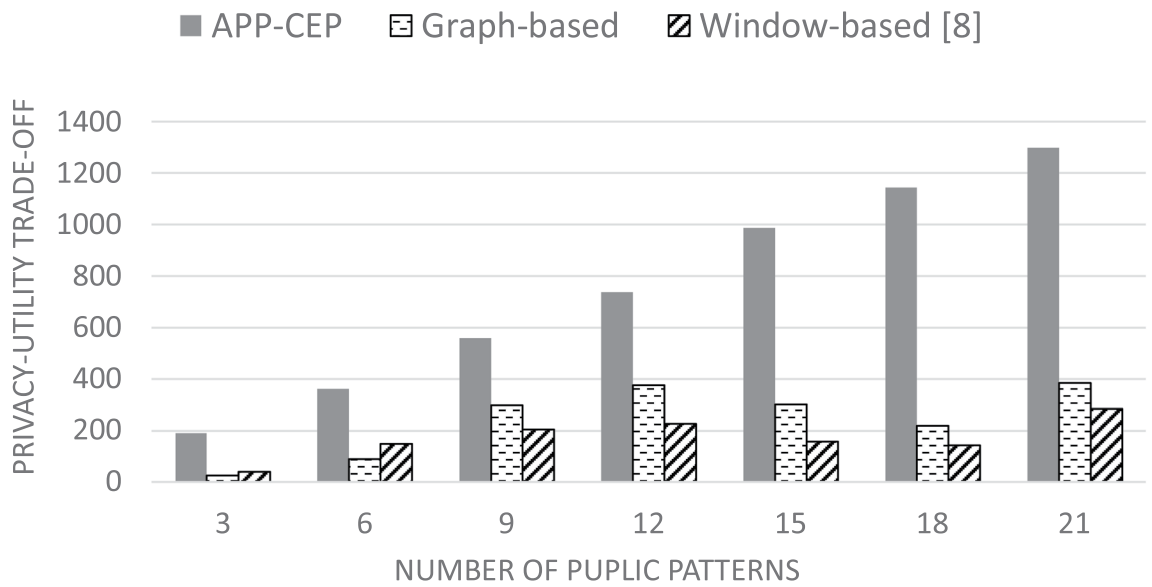

Fig. 8 Privacy-utility trade-off Vs. The increasing number of public patterns for a set of 3 private patterns for the webshop dataset [6]

challenge. Without compromising system utility, it's nearly impossible to completely prevent the revelation of private patterns. The proposed approach demonstrates that a balance between privacy and utility can be achieved, even without relying on specific events in the stream. Furthermore, compensating for revealed matches becomes increasingly difficult without modeling the adversary's additional background knowledge.

Figure 8 presents a comparative analysis of the privacy-utility trade-off for the three proposed approaches. As outlined in Sect. " Problem Statement", our primary objective is to maximize the trade-off value (PUT) as defined by the PUT equation. To calculate the PUT value, we must carefully select the weights assigned to revealed private patterns and expected revealed matches by an adversary. We've opted for a balanced approach, assigning equal weights to both categories.

The weight for each pattern set combination is determined by the ratio of average actual public matches to average actual private matches. For example, in a scenario with 3 public patterns and 3 private patterns, the weight was

calculated to be 1.0301. Additionally, we've set the weight for detected public pattern matches to 1.

Our findings demonstrate that APP-CEP is capable of achieving the desired Privacy Utility Trade-off (PUT) even when the number of queries involved in the system is increased. While the Window-based approach initially showed promising results, its performance declined as more public pattern matches were detected. This is primarily due to the significant impact that revealed private matches have on the PUT value.

The Graph-based approach, on the other hand, consistently outperformed the Window-based approach. However, it also experienced a decline in performance for similar reasons, albeit to a lesser extent. These results highlight the challenges of balancing privacy preservation with utility in dynamic environments where adversaries can leverage public patterns to infer sensitive information.

Evaluation Results for Medical Dataset

We observed that the webshop dataset primarily comprised of unrelated simple patterns which limited the complexity of the pattern dependency graphs. By contrast, the second dataset features patterns with more shared common events, resulting in denser and more intricate dependency structures. This shift in dataset complexity allows us to rigorously evaluate our method's robustness and scalability under challenging conditions. It should be noted that for the sake of simplicity, we limited the available obfuscation techniques to *Suppression*, *Reordering*, and *Tampering*.

Figure 9 shows the percentage of revealed private pattern matches, including those inferred from the adversary's background knowledge, as the number of public patterns grows. Error bars represent simulation variability. Our APP-CEP significantly outperforms other methods by minimizing private pattern disclosure. While Graph-based and Window-based approaches fluctuate with increasing queries (between 55% and 82%), APP-CEP maintains a consistently low disclosure rate (below 13%). The random obfuscation strategy used when the optimal obfuscation method is unavailable impacts performance, as seen mostly in the last four bars. A direct comparison of Fig. 9 with its counterpart for the first dataset (i.e., Fig. 5) highlights the impact of increased pattern dependency complexity. While the medical dataset indeed produces denser dependency graphs, leading to a higher total rate of private pattern disclosure for our approach, it still maintains a significant advantage over the baseline strategies, namely Graph-based and Window-based. This indicates that our method's effectiveness in mitigating privacy leakage is robust even in the face of more challenging data conditions.

Figure 10 presents the accuracy rate, calculated as the ratio of true positive matches to the total number of matches, extracted as the ground truth. A notable limitation of our proposed method is a decreased accuracy rate due to the dense dependency graphs in the medical dataset. This reduction occurs because our approach prioritizes privacy preservation over maximizing the detection of public patterns. Consequently, obfuscation models are selected primarily to protect sensitive information rather than optimize utility. In

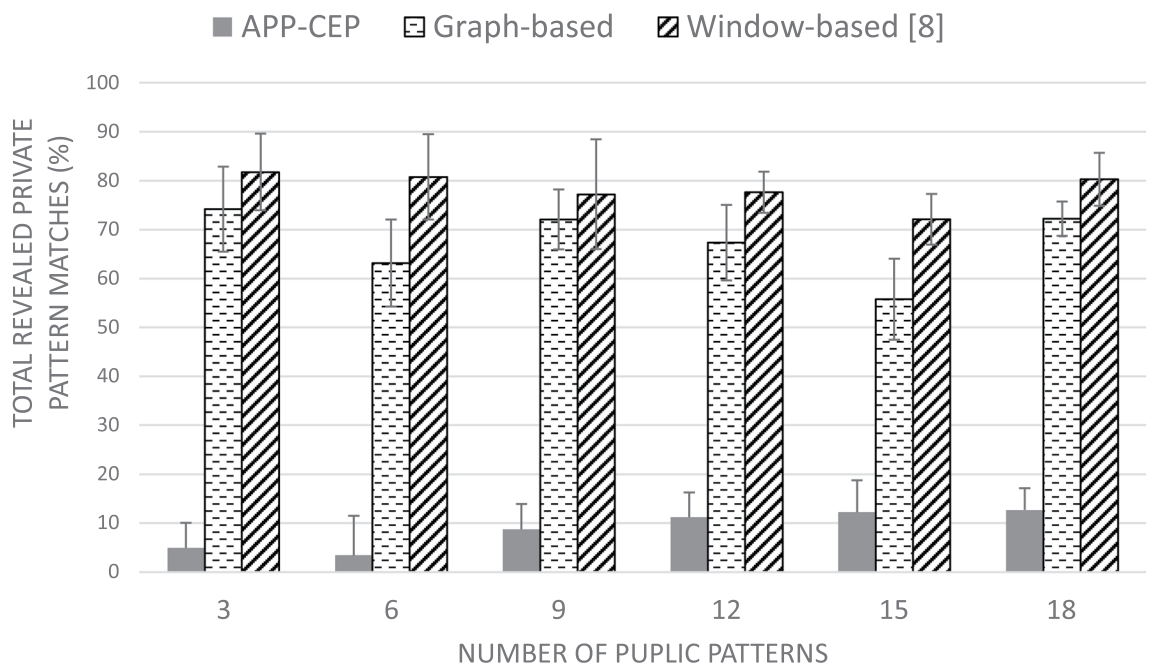

Fig. 9 Percentage of total revealed private pattern matches Vs. The increasing number of public patterns for a set of 3 private patterns for the medical dataset

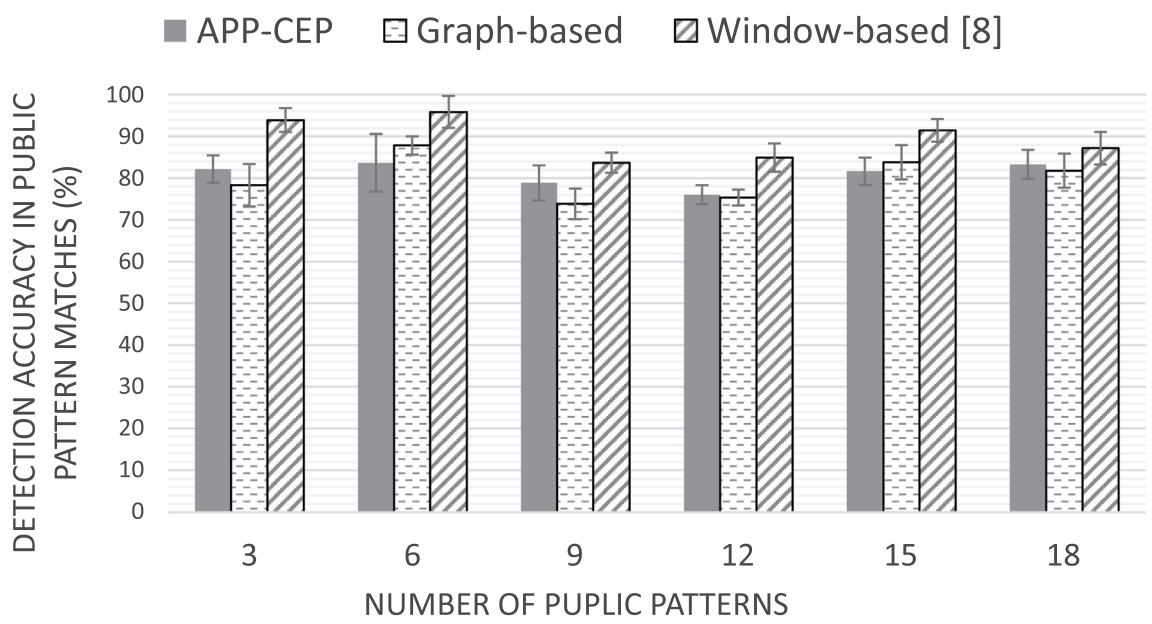

Fig. 10 Accuracy of pattern detection for public patterns Vs. The increasing number of public patterns for a set of 3 private patterns for the medical dataset

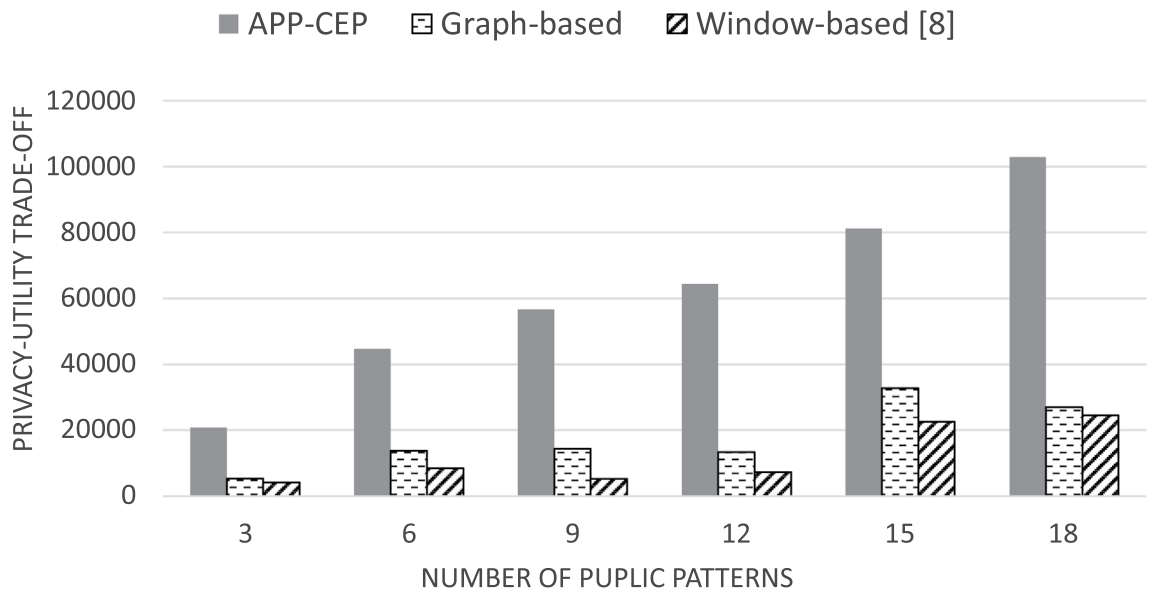

Fig. 11 Privacy-Utility Trade-off Vs. The increasing number of public patterns for a set of 3 private patterns for the medical dataset

contrast, the Window-based approach demonstrates better performance in identifying public patterns, outperforming both graph-based alternatives.

Figure 11 provides a comprehensive evaluation of our method by employing the Privacy-Utility Trade-off (PUT) metric. Consistent with the findings from the first dataset, our approach demonstrates superior performance in balancing privacy and utility, surpassing both baseline strategies even with an increasing number of public patterns. While the PUT values of baseline methods exhibit fluctuations, our method consistently improves as more public patterns are incorporated. The notable decrease in PUT value for other methods when using 18 public patterns highlights the challenge of preserving privacy while maintaining utility in complex scenarios. This decline is primarily attributed to an increased rate of private pattern disclosure, emphasizing the importance of our method's privacy-centric approach. Although Window-based methods are excellent in detecting public patterns, their inability to effectively mitigate private pattern leakage renders them less suitable for privacy-sensitive domains like healthcare.

Ultimately, the evaluation results prove the expected improvements provided by graph-based pattern-level privacy protection, solving the literature's privacy-utility trade-off gap. APP-CEP performs better in almost all aspects discussed in this section than approaches presented in previous studies due to considering both pattern dependencies and modeling the adversary's background knowledge.

Scalability. The scalability of APP-CEP depends on the size of the dependency graph $G = (V, E)$ rather than the raw event volume, since high-throughput stream processing is handled by the underlying CEP engine (FlinkCEP). As shown in Section 5.4, the complexity of the assignment procedure is $O(n)$ where $n := |V| + |E|$, with a quadratic bound $O(|V|^2)$ only in the extreme case of a dense dependency graph. In practice, the graphs derived from our datasets are sparse, and sparse structures are also common in CEP workloads. Therefore, APP-CEP introduces only lightweight overhead and remains applicable at much larger scales than those used in our evaluation.

Related Work

Prior work on privacy in event-based systems encompasses two main directions. The first focuses on *DCEP* itself, where privacy mechanisms are tightly coupled with event detection, obfuscation, and dependency management. These studies aim to protect sensitive patterns while maintaining detection accuracy for non-sensitive patterns, often trading between utility and privacy within the DCEP framework. The second direction builds on broader paradigms for privacy preservation in streaming systems, including *differential privacy*, *anonymization techniques*, and *machine learning-based approaches*. These approaches provide formal guarantees or flexible adaptation, but they generally operate outside the context of DCEP and often struggle with preserving fine-grained event correlations or handling adversarial inference over pattern dependencies.

Privacy in Complex Event Processing

While data sharing among stakeholders enhances the value of IoT applications, ensuring robust privacy preservation remains a significant challenge [18]. In DCEP systems, striking a balance between privacy protection and Quality of Service is essential. Protecting individual privacy should not compromise the application's functionality, which often relies on extensive data for optimal performance [19, 20]. Therefore, privacy-preserving mechanisms must be lightweight and capable of accurately detecting query patterns without hindering application functionality.

Recent trends indicate that data owners are becoming increasingly conscious of how their data is analyzed and are seeking greater control over its usage [21–23]. However, providing adequate avenues for data owners to express their privacy concerns remains an under-explored area. This is particularly crucial in the context of DCEP systems, where sensitive information is shared and processed across multiple entities.

Early research on privacy-preserving pattern obfuscation proposed probabilistic models to estimate event suppression for optimal utility within a window-based framework [12]. These methods leveraged advanced pattern match cardinality estimation techniques to calculate event distribution [24]. While suppression was initially the primary focus, subsequent work introduced alternative obfuscation operators, such as reordering [5]. These approaches, however, often faced limitations in their ability to handle diverse pattern types beyond sequences. Additionally, their effectiveness was highly dependent on the specific characteristics of the input data streams, making them less adaptable to dynamic environments. Moreover, these methods typically lacked the flexibility to apply customized obfuscation techniques to individual private patterns.

To address these challenges, more recent research has explored techniques that can effectively obfuscate a wider range of pattern types, including temporal patterns, spatial patterns, and combinations thereof. These methods often employ machine learning or deep learning approaches to learn complex patterns and apply appropriate obfuscation strategies. Furthermore, they strive to be more adaptable to varying input data distributions and dynamic environments, enabling more robust privacy protection.

The concept of multi-operator multi-pattern privacy protection was first proposed in [8]. While this ILP-based approach effectively maximizes Quality of Service while safeguarding privacy, it is currently restricted to three pattern types and three obfuscation operators. Moreover, the approach suffers from instability due to its frequent switching between obfuscation models in response to changes in the input stream. This oscillation can degrade performance by imposing unnecessary transition overhead.

A crucial oversight in previous work is the lack of thorough consideration for the adversary's background knowledge, which can be exploited to circumvent obfuscation techniques. To address these limitations, future research should explore the respective directions.

Beyond DCEP: Alternative State-of-the-Art Approaches

A prominent direction of other paradigms for privacy preservation in streaming systems is *differential privacy*

(DP), which provides strong theoretical guarantees and has recently been extended toward pattern-level protection in event processing [25]. While DP ensures formal privacy, the required noise injection can distort event correlations and pattern occurrences, which is particularly problematic in settings where dependencies between events are crucial. Related work on semantics-aware anonymization in distributed stream processors, such as the k_s-anonymization framework for Apache Flink [26], demonstrates how lightweight anonymization can be integrated into large-scale systems. However, these methods typically operate at the attribute level and do not explicitly account for dependencies between patterns. Another line of research employs machine-learning-based mechanisms to balance utility and privacy in continuous data analysis. For instance, Fomichev et al. [27] trained models to obfuscate sensitive attributes while retaining useful features. While flexible, such methods require substantial training data, incur non-trivial computational overhead, and usually overlook the role of adversarial background knowledge in exploiting inter-pattern dependencies.

Recent work on differentially private stream aggregation [28] aims to protect continuous aggregate queries under formal privacy bounds. These methods focus on global statistics, but preserving fine-grained pattern semantics and dependencies remains challenging. In contrast, APP-CEP takes a different path; instead of relying on noise injection or costly model training, it leverages event and pattern dependencies directly. This enables lightweight and adaptive protection that maintains utility for public patterns while constraining privacy loss even under adversarial inference.

Analytical Comparison. Compared to these approaches, none of the above solutions directly address the combinatorial and temporal interactions that arise in distributed CEP systems. Differential privacy methods [25, 28], although powerful, assume perturbation at the attribute or aggregate level and do not model how noise propagates through event sequences or affects the detectability of multi-event patterns. As a result, they cannot guarantee the concealment of sensitive pattern instances without severely degrading utility. Semantics-aware anonymization frameworks such as ks-anonymization [26] similarly operate at the record level and lack mechanisms for reasoning about how privacy-preserving actions applied to one event influence other concurrently active patterns. Likewise, learning-based obfuscation approaches [27] focus on attribute transformation and require extensive training data, but they do not incorporate adversarial background knowledge or dependency structures that allow attackers to exploit temporal relationships between events. In contrast, APP-CEP explicitly models event-level and pattern-level dependencies through its dependency graph and dynamically selects obfuscation strategies that are compatible with both privacy goals and quality-of-service requirements. This capability (i.e., reasoning about how one obfuscation action cascades across multiple pattern contexts) is absent in DP-based, anonymization-based, and ML-based approaches. By integrating dependency modeling with online obfuscation assignment, APP-CEP addresses a gap that existing techniques do not cover and provides privacy guarantees tailored to the semantics of complex event processing.

Conclusion

This article introduces APP-CEP, a novel approach for dynamically integrating pattern-level privacy into event-based systems to safeguard sensitive information while maintaining acceptable quality of service (QoS). Furthermore, our mechanism generates pattern dependency graphs to selectively apply obfuscation techniques for concealing sensitive patterns, enabling proactive obfuscation planning. Our evaluation on the webshop dataset, processed as a temporally replayed event stream to ensure reproducibility, revealed that APP-CEP surpasses both window-based [8] and graph-based baselines in minimizing the percentage of revealed private pattern matches. Additionally, APP-CEP demonstrated slightly improved performance in detecting public pattern matches accurately. Moreover, by incorporating additional background knowledge, APP-CEP demonstrates a reasonable performance in mitigating the adversary's ability to infer sensitive patterns. Besides, the observed privacy-utility trade-off confirms the effectiveness of APP-CEP in addressing the research gap identified in this study. The proposed approach paves the way for a new paradigm in which obfuscation techniques are dynamically applied to event streams regardless of actual events in the streams. In the context of the medical dataset, APP-CEP effectively managed more complex dependency graphs. Evaluation results demonstrated a reasonably lower rate of revealed private patterns (under 13%) compared to baseline graph-based and window-based [8] approaches (in the range of 55% to 82%), indicating minimal privacy leakage. Our evaluation also demonstrates that APP-CEP maintains its performance in the privacy-utility trade-off under increasing query loads.

Future work will focus on developing an intelligent algorithm for selecting optimal obfuscation techniques when dependency sub-graphs lack ZOD nodes. This will involve defining a comparative metric to quantify both the benefits and drawbacks of obfuscations within sub-graphs. Besides, careful consideration must be given to stakeholder interests (e.g., as an input to the system) to effectively deploy this approach in real-world settings. Future work will also

involve extending our evaluation to diverse use cases to comprehensively assess the system's capabilities and limitations in addressing varying application requirements.

Declarations

Conflict of interest We declare the following potential Conflict of interest: individuals affiliated with the University of Groningen, University of Amsterdam, Vrije Universiteit Amsterdam, Amsterdam UMC, Technische Universität Ilmenau, University of Oslo, and University of Darmstadt may have a Conflict of interest in the review of this article due to institutional or collaborative relationships.

References

1. Lotfian Delouee M, Koldehofe B, Degeler V, Towards adaptive quality-aware complex event processing in the internet of things. In: Proceedings of the 18th International Conference on Mobility, Sensing and Networking (MSN'22). IEEE; 2022. p. 571–575 .
2. Lotfian Delouee M, Koldehofe B, Degeler V, Aqua-cep: Adaptive quality-aware complex event processing in the internet of things. In: Proceedings of the 17th ACM International Conference on Distributed and Event-Based Systems (DEBS'23). ACM; 2023. p. 13–24.
3. Khalil A, Lotfian Delouee M, Degeler V, Meuser T, Anta AF, Koldehofe B, Driving towards efficiency: Adaptive resource-aware clustered federated learning in vehicular networks. In: Proceedings of the 22nd Mediterranean Communication and Computer Networking Conference (MedComNet'24). IEEE; 2024.
4. Lotfian Delouee M, Koldehofe B, Degeler V, Poster: Towards pattern-level privacy protection in distributed complex event processing. In: Proceedings of the 17th ACM International Conference on Distributed and Event-Based Systems. ACM; 2023. p. 185–186 .
5. Palanisamy SM, Dürr F, Tariq MA, Rothermel K, Preserving privacy and quality of service in complex event processing through event reordering. In: Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems. 2018. p. 40–51.
6. Lotfian Delouee M, Degeler V, Amthor P, Koldehofe B, App-cep: Adaptive pattern-level privacy protection in complex event processing systems. In: Proceedings of the 10th International Conference on Information Systems Security and Privacy - ICISSP. SciTePress; 2024. p. 486–497.
7. Gundersen M, Statistics Norway demands to know exactly what Norwegians buy in the grocery store. https://nrkbeta.no/2022/05/28/ssb-krever-a-fa-vite-noyaktig-hva-nordmenn-kjoper-i-matbutikken/, Accessed on Oct 9th, 2023 ;2022.
8. Palanisamy SM, Towards multiple pattern type privacy protection in complex event processing through event obfuscation strategies. In: International Workshops on Data Privacy Management, Cryptocurrencies and Blockchain Technology: (DPM' 20 and CBT' 20). Springer; 2020. p. 178–194.
9. Lotfian Delouee M, Pernes DG, Degeler V, Koldehofe B, Towards federated llm-powered cep rule generation and refinement. In: Proceedings of the 18th ACM International Conference on Distributed and Event-based Systems. 2024. p. 185–186.
10. Alt B, Weckesser M, Becker C, Hollick M, Kar S, Klein A, et al. Transitions: a protocol-independent view of the future internet. Proc IEEE. 2019;107(4):835–46.
11. Stach C, Steimle F, Recommender-based privacy requirements elicitation-epicurean: an approach to simplify privacy settings in iot applications with respect to the gdpr. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. 2019. p. 1500–1507.
12. He Y, Barman S, Wang D, Naughton JF, On the complexity of privacy-preserving complex event processing. In: Proceedings of the 13th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. 2011. p. 165–174.
13. Flink: Flink-CEP. https://nightlies.apache.org/flink/ flink-docs-master/docs/libs/cep/, Accessed on Oct 9th, 2023 (2017).
14. Chen D, Sain SL, Guo K. Data mining for the online retail industry: a case study of rfm model-based customer segmentation using data mining. J Database Market Custom Strat Manag. 2012;19:197–208.
15. Clore J, Cios K, DeShazo J, Strack B, Diabetes 130-US Hospitals for Years 1999-2008. UCI Machine Learning Repository. (2014). https://doi.org/10.24432/C5230J.
16. Schut MC, Dongelmans DA, De Lange DW, Brinkman S, Keizer N, Abu-Hanna A. Development and evaluation of regression tree models for predicting in-hospital mortality of a national registry of covid-19 patients over six pandemic surges. BMC Med Inform Decis Mak. 2024;24(1):7.
17. Li L, Zhong B, Hutmacher C Jr, Liang Y, Horrey WJ, Xu X. Detection of driver manual distraction via image-based hand and ear recognition. Accident Anal Prev. 2020;137:105432.
18. Tran T, Nguyen P, Erdogan G, A systematic review of secure iot data sharing. In: Proceedings of the 9th International Conference on Information Systems Security and Privacy (ICISSP'23). 2023. p. 95–105.
19. Plagemann T, Goebel V, Hollick M, Koldehofe B, Towards privacy engineering for real-time analytics in the human-centered internet of things. 2022 arXiv:2210.16352.
20. Schilling B, Koldehofe B, Rothermel K, Ramachandran U, Access policy consolidation for event processing systems. In: 2013 Conference on Networked Systems. IEEE. 2013. p. 92–101 .
21. Tokas S, Erdogan G, Stølen K, Privacy-aware iot: State-of-the-art and challenges. In: Proceedings of the 9th International Conference on Information Systems Security and Privacy (ICISSP'23). 2023. p. 450–461.
22. Leicht J, Heisel M, P2bac: Privacy policy based access control using p-lpl. In: Proceedings of the 9th International Conference on Information Systems Security and Privacy (ICISSP'23). 2023. p. 686–697.
23. Barber F, Furnell S, Benchmarking consumer data and privacy knowledge in connected and autonomous vehicles. In: Proceedings of the 8th International Conference on Information Systems Security and Privacy (ICISSP'22). 2022. p. 426–434.

24. Wang D, He Y, Rundensteiner E, Naughton JF, Utility-maximizing event stream suppression. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. 2013. p. 589–600.
25. Gu H, Plagemann T, Benndorf M, Goebel V, Koldehofe B, Differential privacy for protecting private patterns in data streams. In: 2023 IEEE 39th International Conference on Data Engineering Workshops (ICDEW). IEEE;2023. p. 118–124.
26. Groneberg P, Voigt S, Janke T, Loechel L, Wolf K, Grünewald E, Pallas F, Prink: ks-anonymization for streaming data in apache flink. In: International Conference on Availability, Reliability and Security. Springer. 2025. p. 24–45.
27. Fomichev M, A path to holistic privacy in stream processing systems. In: Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services, 2023:627–628.
28. Zhang B, Doroshenko V, Kairouz P, Steinke T, Thakurta A, Ma Z, et al. Differentially private stream processing at scale. Proceedings of the VLDB Endowment. 2024;17(12):4145–58.