SHIELD: Evolutionary Synthesis of Privacy-Preserving Pipelines for Live Stream Data Sharing

SHIELD: Evolutionary Synthesis of Privacy-Preserving Pipelines for Live Stream Data Sharing

Silvia Perelli
Department of Civil Engineering and Computer Science Engineering (DICII), University of Rome Tor Vergata
Roma, Italy
silvia.perelli@alumni.uniroma2.eu

Eric Medvet
Department of Engineering and Architecture, University of Trieste
Trieste, Italy
emedvet@units.it

Vincenzo Gulisano
Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
vincenzo.gulisano@chalmers.se

Abstract

Modern data-driven organizations rely on pipelines that transform data from infrastructure, applications, and users, where correctness and performance are critical for reliability and business value. These pipelines are often developed and optimized by teams separate from the data owners defining semantics, so meaningful testing and benchmarking require sharing representative data, even though real data is often sensitive and restricted by legal and commercial constraints.

To enable privacy-preserving data sharing while preserving utility, our framework, named SHIELD, automatically constructs stream processing (SP) pipelines to transform live sensitive data into shareable data while preserving the characteristics required for downstream processing and optimization. Internally, SHIELD leverages evolutionary computation to synthesize executable SP queries under predefined privacy and utility requirements. Using real-world use cases, we show SHIELD can synthesize privacy-preserving pipelines that retain analytical value and scale to realistic workloads.

CCS Concepts

• **Information systems** → **Data management systems**; • **Theory of computation** → **Evolutionary algorithms**; **Streaming models**.

Keywords

Stream Processing; Evolutionary Computing

ACM Reference Format:
Silvia Perelli, Eric Medvet, and Vincenzo Gulisano. 2026. SHIELD: Evolutionary Synthesis of Privacy-Preserving Pipelines for Live Stream Data Sharing. In *The 20th ACM International Conference on Distributed and Event-based Systems (DEBS '26), June 23–26, 2026, Lisbon, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3809481.3812614

DEBS '26, Lisbon, Portugal

ACM ISBN 979-8-4007-2693-4/2026/06
https://doi.org/10.1145/3809481.3812614

1 Introduction

In modern data-driven organizations, correctness and performance are critical for the operational reliability and business value of pipelines transforming data on the fly from infrastructure, applications, and users. These pipelines often span the edge–cloud continuum, integrate heterogeneous sources, and operate under strict throughput/latency and resource constraints [24]. This complexity separates concerns: data owners define semantics and requirements, while specialized teams (or external partners) operate pipelines across devices and runtimes. Meaningful testing and benchmarking therefore require representative data, yet real data is often sensitive and restricted by legal and commercial constraints.

When organizations fall back to synthetic or heavily sanitized data, key *statistical*, *temporal*, and *structural* property changes can undermine tuning and validation, yielding optimizations that do not transfer and may degrade reliability.

Challenges. Designing privacy-preserving transformations that keep data useful for target applications requires multi-objective optimization with intertwined challenges:

1. **Privacy vs. utility.** Privacy preservation intentionally reduces information (e.g., masking, suppression, or perturbation), whereas utility is better preserved by retaining the original information and its statistical, temporal, and structural characteristics.
2. **Application-tailored objectives and tunable metrics.** Which data aspects matter is workload- and application-dependent; balancing privacy and utility thus entails privacy-risk and utility-preservation metrics that data owners can inspect and tune.
3. **Scalable search and deployable pipelines.** Scalable exploration within time and resource budgets requires creating semantically valid pipelines that modern stream processing (SP) runtimes can optimize (e.g., parallelization),

so that fitness evaluation scales during search and selected solutions can later keep up with live data anonymization.

Contribution. SHIELD tackles such challenges by automatically constructing SP pipelines (called *queries* hereafter) that enforce privacy requirements while preserving data properties needed for downstream use:

1. **Search for privacy–utility trade-offs (Challenge 1).** To balance privacy and utility over a large space of candidate transformations, SHIELD uses evolutionary computation (EC) [6, 34] to automatically synthesize queries. Data owners specify privacy goals and utility requirements; EC explores compositions of anonymization operations (e.g., masking, noise injection) that meet these goals while preserving properties required by downstream tasks. The resulting pipelines remain explainable as explicit operator compositions, enabling human validation and adjustment.
2. **Tunable metrics for multi-objective optimization (Challenge 2).** SHIELD explicitly incorporates privacy (e.g., re-identification or inference risk) and utility metrics capturing preservation of statistical, temporal, and structural properties relevant to the target query. These metrics, designed to be interpretable and tunable, are used as objectives in multi-objective optimization, for alternative solutions to be compared and selected transparently.
3. **Scalable evaluation and deployability via SP (Challenge 3).** By expressing data transformation as queries, SHIELD supports on-the-fly data transformation, making safe data sharing feasible in operational settings. The synthesized queries can exploit established SP performance-boosting techniques to evaluate efficiently candidate solutions and keep up with the live data they anonymize.

Besides presenting SHIELD and discussing its components, we provide a fully implemented prototype available at https://zenodo.org/records/20348184, built on the Liebre stream processing engine (SPE) [20] and the JGEA EC [22] framework, and we evaluate SHIELD on two use cases based on GeoLife mobility traces [36] and UCI AirQuality data [30]. Together with clear privacy–utility trade-offs, we show that richer query expressiveness improves solution quality and diversity and that evolved queries' throughput and latency is compatible with live anonymization workloads.

Organization: Section 2 covers preliminaries (SP and EC), Section 3 covers our problem statement, Section 4 presents SHIELD, Section 5 discusses use cases and results, Section 6 presents related work, and Section 7 concludes our work.

2 Preliminaries

2.1 Stream processing (SP)

A *stream* $\boldsymbol{t}$ is an unbounded sequence of *tuples*, each a list of attribute-value pairs $\langle \tau : v_0, a_1 : v_1, \ldots, a_n : v_n \rangle$ [1]. Within $\boldsymbol{t}$, every tuple t has the same attributes, called *type* and denoted as $T(t)$ or $T(\boldsymbol{t})$. The timestamp attribute (τ) is always included in the type of a tuple. We use $t.a$ to denote the value of the attribute a for t; $t.\tau$ denotes the *event time* of t.

A *query* is a directed acyclic graph (DAG) connecting *ingresses*, *operators*, and *egresses* that can be used for processing streams. Ingresses forward *ingress tuples* (e.g., events from sensors) to operators that process and forward/produce tuples. Eventually, *egress tuples* are fed to egresses and delivered to end-users/other applications. We denote by $\boldsymbol{t}_{\text{out}} = q(\boldsymbol{t}_{\text{in}})$ the output of a query q on an input stream $\boldsymbol{t}_{\text{in}}$, *i.e.*, the stream $\boldsymbol{t}_{\text{out}}$ of the egress tuples coming from q. We denote by $T(q)$ the (minimal) set of attributes the query q requires in a stream to process it: that is, the condition $T(\boldsymbol{t}) \supseteq T(q)$ must hold. Note that in general, $T(\boldsymbol{t})$ may be different than $T(q(\boldsymbol{t}))$.

The common operators we consider include both *stateless* (Map and Filter) and *stateful* (Aggregate) operators:

Map $\boldsymbol{t}_{\text{out}} = M(\boldsymbol{t}_{\text{in}}, f_M)$ processes $\boldsymbol{t}_{\text{in}}$ tuples with function f_M to produce stream $\boldsymbol{t}_{\text{out}}$. Function f_M is invoked on each $t_i \in \boldsymbol{t}_{\text{in}}$ to produce zero, one, or more t_o output tuples. Note $T(\boldsymbol{t}_{\text{in}})$ and $T(\boldsymbol{t}_{\text{out}})$ can be equal or can differ, depending on f_M, and that $t_o.\tau = t_i.\tau$ for a t_o produced from t_i.

Filter $\boldsymbol{t}_{\text{out}} = F(\boldsymbol{t}_{\text{in}}, p_F)$ forwards each $t_i \in \boldsymbol{t}_{\text{in}}$ to $\boldsymbol{t}_{\text{out}}$ if a predicate $p_F(t_i)$ holds. Note that $T(\boldsymbol{t}_{\text{in}}) = T(\boldsymbol{t}_{\text{out}})$ and $t_o = t_i$ for a t_o output by processing t_i.

Aggregate $\boldsymbol{t}_{\text{out}} = A(\Gamma(WA, WS, K), \boldsymbol{t}_{\text{in}}, f_O)$ operates on *windows* of tuples, which can be *time-based* [9, 20, 29] or *tuple-based* (count) [1]. Window Γ is defined by its:

- **Window Advance (*WA*), Size (*WS*):** the epochs it covers: $[\ell WA, \ell WA + WS)$, with $\ell \in \mathbb{N}$. For time-based windows, WA/WS are durations in time units; for count ones, numbers of tuples. We refer to one such epoch as window *instance* γ. If $WA < WS$, consecutive γs overlap and Γ is called *sliding*. If $WA = WS$, Γ is called *tumbling* and each tuple falls in only one γ.
- **Key-by attributes *K*:** the subset of $T(\boldsymbol{t}_{\text{in}})$ (even empty) used to keep dedicated γs for tuples with the same *key*.

Function f_O computes the values of an output t_o from a γ.

2.2 Evolutionary computation (EC)

EC [7] is a class of global optimization methods, called evolutionary algorithms (EAs), inspired by natural evolution; EAs perform optimization via iterative, population-based search.

Given a search space S and a partial order $\prec$ over S (where $s_1 \prec s_2$ means that $s_1 \in S$ is *better* than $s_2 \in S$), a typical EA: (1) generates a population of n_{pop} candidate solutions (*i.e.*, elements of S, also called *individuals*) with some non-deterministic initialization procedure, (2) repeats the following steps for a given number n_{gen} of times: (a) it builds an offspring of $n_{\text{offspring}}$ new individuals starting from the current

population (called parents) by repeatedly selecting good individuals and modifying or combining them (this step is called *reproduction*), (b) it merges the offspring and the parents to form a larger population, and (c) it trims to the size n_{pop} of the initial population by selecting surviving individuals.

2.2.1 Selection. Individual selection for reproduction and survival is specific to each EA, but is generally independent of the search space S. In both cases, selection is nondeterministic but based on the comparison of individuals according to the partial order $\prec$: *i.e.*, given s_1, s_2 such that $s_1 \prec s_2$, s_1 has a greater probability of being selected (for reproduction or surviving) than s_2. In many practical cases, candidate solutions are not compared directly, but through one or more quantitative measurements associated with them. In *multi-objective* optimization problems, given a *fitness* function $f : S \to \mathbb{R}^k$, the Pareto-dominance is used (like in our case) as partial order, *i.e.*, $s_1 \prec s_2$ (read as s_1 *dominates* s_2) if $\forall i \in \{1, \ldots, k\}, f_i(s_1) \geq f_i(s_2)$ and $\exists j$ s.t. $f_j(s_1) > f_j(s_2)$ (assuming maximization for each one of the k objectives).

In this work we use the non-dominated sorting genetic algorithm II (NSGA-II), an EA widely used for multi-objective optimization. It follows the general scheme described above and performs selection based on Pareto-dominance and the crowding distance of candidate solutions in the objective space $\mathbb{R}^k$. Namely, in the selection for reproduction it adopts the tournament selection, as follows. Whenever it needs to select one parent from the population, it (1) takes randomly with uniform probability and repetition n_{tour} individuals, (2) picks the subset of them with the lowest Pareto rank, and (3) selects the one of them with the largest *crowding distance*. The Pareto rank is the index of the Pareto front on which an individual lies: 1 for individuals which are not dominated by any other individual in the population, 2 for those dominated only by individuals of rank 1, and so on. The crowding distance is, for a solution s, the sum of the distances along each j-th objective of s to the closest preceding and subsequent solutions. By maximizing the crowding distance, NSGA-II sparsifies the search in the objective space to better approximate the ideal Pareto front of optimal solutions.

For survival selection, NSGA-II uses truncation: it retains the best n_{pop} individuals by lowest Pareto rank, breaking ties by largest crowding distance.

2.2.2 Solution representation. The mechanisms that modify or combine individuals are S-dependent and called *genetic operators*. Typical operators include mutation $o_{\text{mut}} : S \to \mathcal{P}(S)$ and crossover $o_{\text{xover}} : S \times S \to \mathcal{P}(S)$, where $\mathcal{P}(S)$ denotes probability distributions over S. Mutation (the only genetic operator used here) is hence a non-deterministic operator over S. Population initialization is also modeled as a probability distribution over S.

The mutation operator and the initialization step are central to an EA, determining how the population moves (and where it starts) in S toward the Pareto front induced by f. Ideally, o_{mut} ensures locality (*i.e.*, similar solutions yield similar fitness) and closure (*i.e.*, mutations remain in S). Enforcing these properties for arbitrary S—as in our case, where solutions are SP queries—is difficult. We address this by representing candidates indirectly via a context-free grammar (CFG), following context-free grammar GP (CFG-GP).

CFG-GP [31] is a genetic programming (GP) variant [18] (an EA initially adopted to evolve computer programs) where solutions are strings of a language $\mathcal{L}(G)$ generated by a CFG G. We use CFG-GP for representation and NSGA-II for selection. CFG-GP manipulates solutions via their genotypes, represented as production trees of a CFG G. Formally, a CFG G is a tuple (L_T, L_N, R, l_0), where L_T is the set of terminal symbols (the alphabet of the language $\mathcal{L}(G)$), L_N is a disjoint set of non-terminals, $l_0 \in L_T \cup L_N$ is the start symbol, and R is the set of production rules, each with a non-terminal left-hand-side and a sequence of terminals and/or non-terminals as right-hand-side. In CFG-GP, genotypes are production trees: ordered trees with node labels in $L_N \cup L_T$, the root labeled l_0, internal nodes labeled with L_N elements, and leaves with L_T elements. Each non-terminal node with a label $l \in L_N$ has children with labels matching (in order) the right-hand side of a production rule with left-hand side l. Reading the leaves left to right yields a string in $\mathcal{L}(G)$. CFG-GP mutation operates on production trees by randomly selecting a subtree and replacing it with a new subtree compatible with G (*i.e.*, rooted at the same non-terminal). Initialization samples random production trees with height in $[h_{\min}, h_{\max}]$, and mutation enforces the same height constraint.

3 Problem statement

Let u be a user with an input stream $\boldsymbol{t}_u$ (typically a historical sample used to design a query) with tuples carrying numerical values, and a template query q_u, with $T(\boldsymbol{t}_u) \supseteq T(q_u)$. A template query captures the intended semantics but is not necessarily optimized; for example, it may use suboptimal parallelism or an inefficient Aggregate implementation (e.g., single-window instead of multi-window [14]).

The user u wishes to derive from q_u an optimized $q^\star$ such that (a) $q^\star$ is semantically equivalent to q_u and (b) the implementation and deployment parameters of $q^\star$ achieve better performance than those of q_u. Intuitively, $q^\star$ is a production-ready version of q_u, *i.e.*, it scores better on some metrics (e.g., throughput, latency, or CPU utilization) than q_u. We denote by $m_1, \ldots, m_h$ the collection of h metrics that u wants to improve. Ideally, $q^\star(\boldsymbol{t}_u) = q_u(\boldsymbol{t}_u)$. Also, $q^\star$ should improve performance, *i.e.*, $\forall i,\ m_i(q^\star, \boldsymbol{t}_u) > m_i(q_u, \boldsymbol{t}_u)$ where $m_i(q, \boldsymbol{t}) \in \mathbb{R}$ denotes metric m_i computed for q over $\boldsymbol{t}$, and

we assume "the greater, the better" for each m_i. In practice, it is acceptable that outputs are similar and that only a subset of the metrics improves, depending on the user-defined optimization objective and trade-offs. For brevity, we write $\boldsymbol{m}(q, \boldsymbol{t}) \in \mathbb{R}^h$ to denote collectively $m_1(q, \boldsymbol{t}), \ldots, m_h(q, \boldsymbol{t})$.

To derive $q^\star$ from q_u, u relies on a third party e, which may be another user (e.g., a query optimization expert) or an external platform where q_u is deployed and tuned under different configurations. However, e is untrusted. While u can share q_u, $\boldsymbol{t}_u$ cannot be disclosed. Since e operates outside u's trusted domain, any interaction involving data or intermediate results must be assessed from a privacy perspective; thus, e may act as an attacker with respect to u and $\boldsymbol{t}_u$.

We do not assume an *a priori* attacker goal: $\boldsymbol{t}_u$ may, e.g., contain sensitive information that u does not want to disclose, so an attacker may try to infer information about $\boldsymbol{t}_u$; alternatively, if $\boldsymbol{t}_u$ is a subset of a public dataset, an attacker may aim to identify the dataset portion of interest to u.

To address these risks, u seeks to produce a *synthetic stream* $\boldsymbol{t}'_u$, *i.e.*, a modified version of $\boldsymbol{t}_u$ which satisfies $T(\boldsymbol{t}'_u) \supseteq T(q_u)$ and the following three requirements, one concerning privacy and two, fidelity and semantics, concerning utility:

(1) **Privacy:** sharing $\boldsymbol{t}'_u$ with e preserves the privacy of $\boldsymbol{t}_u$.
(2) **Fidelity:** the query $q^\star$ obtained from q_u exhibits similar metrics on $\boldsymbol{t}_u$ and $\boldsymbol{t}'_u$, *i.e.*, $\boldsymbol{m}(q^\star, \boldsymbol{t}_u) \approx \boldsymbol{m}(q^\star, \boldsymbol{t}'_u)$.
(3) **Semantics:** the user query q_u exhibits similar output behavior on $\boldsymbol{t}_u$ and $\boldsymbol{t}'_u$, *i.e.*, its outputs remain consistent with the intended downstream semantics.

For example, if $q^\star$ identifies an optimal parallelism level for an operator in q_u, that choice should remain optimal when $q^\star$ runs on both $\boldsymbol{t}_u$ and $\boldsymbol{t}'_u$. The three requirements trade off: a highly private $\boldsymbol{t}'_u$ (e.g., a "random" stream) may not preserve enough information from $\boldsymbol{t}_u$ to support fidelity and semantics, whereas $\boldsymbol{t}'_u = \boldsymbol{t}_u$ maximizes fidelity and semantics but discloses all information in $\boldsymbol{t}_u$ and thus violates privacy.

We cast the problem of deriving a synthetic stream $\boldsymbol{t}'_u$ from $\boldsymbol{t}_u$ as the problem of building a *pre-processing* query $q_{\text{pre-proc}}$ such that $\boldsymbol{t}'_u = q_{\text{pre-proc}}(\boldsymbol{t}_u)$ meets the three requirements above. We remark that, since fidelity depends on $q^\star$, which is not available when constructing $q_{\text{pre-proc}}$, designing $q_{\text{pre-proc}}$ is particularly challenging. Note that we do not assume q_u and $q^\star$ to behave identically. Since $q^\star$ is not available when constructing $q_{\text{pre-proc}}$, we define fidelity in terms of the workload-relevant conditions q_u exposes on $\boldsymbol{t}_u$ to be preserved on the synthetic stream used by e for optimization.

4 The SHIELD framework

We model the construction of query $q_{\text{pre-proc}}$ as a multi-objective optimization problem. The search space S consists of pre-processing queries and the objectives measure how well $\boldsymbol{t}'_u = q_{\text{pre-proc}}(\boldsymbol{t}_u)$ satisfies the requirements above.

SHIELD takes as input the user's stream $\boldsymbol{t}_u$, query q_u, and metrics $\boldsymbol{m} = m_1, \ldots, m_h$, and outputs a set Q of pre-processing queries spanning different privacy–fidelity–semantics trade-offs, for the user to pick their preferred one.

Internally, SHIELD uses NSGA-II (Section 2.2.1) to solve the multi-objective optimization problem and represents the search space—*i.e.*, the candidate $q_{\text{pre-proc}}$ queries—via the grammar-based CFG-GP encoding (Section 2.2.2). Before the optimization, SHIELD lets users specialize both the objectives through parameters $g, \boldsymbol{m}$, and $\equiv$, presented next, and the CFG to the input $\boldsymbol{t}_u, q_u$.

4.1 Optimization objectives

We define a set of three objectives for the multi-objective problem. These objectives operationalize the three requirements introduced in Section 3: privacy, fidelity, and semantics. Since $q^\star$ is not available to SHIELD during the search, the fidelity and semantics objectives are expressed based on q_u. Together, the three objectives define the fitness function $f : S \to \mathbb{R}^3$ used by NSGA-II: we denote them by f_{privacy}, f_{fidelity}, and $f_{\text{semantics}}$, each being $S \to \mathbb{R}$. For all objectives, the greater, the better, *i.e.*, they have to be maximized.

4.1.1 Privacy objective. The problem of reliably measuring how much information synthetic data discloses about its real counterpart remains open [11]. We therefore adopt a key intuition inspired by k-anonymity. We view tuples in a synthetic stream[1] $\boldsymbol{t}'$ as points in the same attribute space as the original stream $\boldsymbol{t}_u$, possibly displaced by transformations such as noise injection or aggregation. A modified tuple that lies in a sparse region is easier to link to a specific original record, especially under small displacement; conversely, if it lies in a region where many original tuples appear at comparable distances, identifying its true source becomes harder. We thus model privacy as a function of the local density around each modified tuple, in the spirit of k-anonymity.

However, density alone is insufficient: an empty stream would be un-linkable, yet useless. We therefore evaluate privacy while encouraging comparable sizes for $\boldsymbol{t}'$ and $\boldsymbol{t}_u$.

We define f_{privacy} based on these two ingredients (density and size similarity) as follows, given the original stream $\boldsymbol{t}_u$ and a $\boldsymbol{t}'$. First, we define the distance between tuples of $\boldsymbol{t}_u$ and $\boldsymbol{t}'$ as:

$$d(t, t') = \sqrt{\frac{1}{|T|} \sum_{a \in T} \frac{(t.a - t'.a)^2}{\min(\epsilon, \sigma_a^2)}}, \tag{1}$$

where $T = T(\boldsymbol{t}_u) \cap T(\boldsymbol{t}')$ is the set of common attributes in $\boldsymbol{t}_u$ and $\boldsymbol{t}'$, $\sigma_a = \text{sd}(\{t.a, t \in \boldsymbol{t}_u\})$ is the standard deviation of the values of the attribute a for the tuples of $\boldsymbol{t}_u$, and $\epsilon = 10^{-5}$ is a small value to avoid division-by-zero issues. By employing σ_a

[1] We denote it as $\boldsymbol{t}'$ rather than $\boldsymbol{t}'_u$ since it represents only one of the possible synthetic streams found during the evolutionary process.

we normalize the Euclidean distance between tuples. Without normalization, attributes with larger numerical ranges would dominate the distance computation, disproportionately influencing the nearest-neighbor selection and biasing the privacy assessment.

We then define the dispersion value $\sigma_d(t')$ of $t' \in \boldsymbol{t}'$ as:

$$\sigma_d(t') = \mathrm{sd}\left(\{d(t,t'), t \in N^k_{t'}\}\right), \tag{2}$$

where $N^k_{t'} \subseteq \boldsymbol{t}_u$ is the set of the k tuples of $\boldsymbol{t}_u$ closest to t'. The dispersion $\sigma_d(t')$ represents the core indicator of privacy risk: low values indicate high privacy, because the similar distances place the modified tuple t' in a region of uniform density with respect to the original stream $\boldsymbol{t}_u$, creating substantial ambiguity that makes it difficult for an attacker to single out the true original tuple among the k candidates. Conversely, high $\sigma_d(t')$ values indicate increased privacy risk, as highly variable distances characterize the tuple as an outlier, enabling an attacker to trivially re-identify the original tuple by selecting the closest match.

Finally, we define $f_{\text{privacy}}(q; g)$ by aggregating tuple dispersions in $\boldsymbol{t}' = q(\boldsymbol{t}_u)$ through a parameter function $g : \mathbb{R}^* \to \mathbb{R}$ and modulating the result by a factor accounting for size dissimilarity between the original and a synthetic stream:

$$f_{\text{privacy}}(q; g) = \frac{\min\left(\frac{|\boldsymbol{t}_u|}{|q(\boldsymbol{t}_u)|}, \frac{|q(\boldsymbol{t}_u)|}{|\boldsymbol{t}_u|}\right)}{1 + g(\{\sigma_d(t'), t' \in q(\boldsymbol{t}_u)\})}. \tag{3}$$

Function g aggregates dispersions into a stream-level privacy indicator. We evaluate both $g = \mathrm{p}_{99}$ (*i.e.*, the 99-th percentile) to obtain a robust, tail-focused estimate, and $g = \max$ as a stricter worst-case criterion; more generally, users may specify alternative aggregation functions to match their risk model. The modulating factor is ≤ 1 and decreases when the two streams have very different sizes, hence decreasing the value of $f_{\text{privacy}}(q; g)$.

4.1.2 Fidelity objective. Fidelity concerns the performance of $q^\star$ on a synthetic stream, measured using $\boldsymbol{m}$. As we do not have access to $q^\star$ while optimizing the pre-processing query, we focus on the performance of q_u on $\boldsymbol{t}$ and $\boldsymbol{t}' = q(\boldsymbol{t}_u)$. Namely, we define $f_{\text{fidelity}}(q)$ as one minus the Euclidean distance of the performance on the two streams:

$$f_{\text{fidelity}}(q) = 1 - \left\|\boldsymbol{m}(q_u, \boldsymbol{t}_u) - \boldsymbol{m}(q_u, q(\boldsymbol{t}_u))\right\|. \tag{4}$$

4.1.3 Semantics objective. This objective operationalizes the semantics requirement introduced in Section 3. Due to the unavailability of $q^\star$ during optimization, as for the fidelity objective, we define it in terms of the user query q_u. Given an equivalence relation $\equiv$ among tuples, we define $f_{\text{semantics}}(q)$ as the F1 score of the output of q_u on $\boldsymbol{t}$ with respect to that of q_u on $\boldsymbol{t}' = q(\boldsymbol{t}_u)$:

$$f_{\text{semantics}}(q) = 2\frac{\mathrm{Prec}(q)\mathrm{Rec}(q)}{\mathrm{Prec}(q) + \mathrm{Rec}(q)}, \tag{5}$$

where:

$$\mathrm{Prec}(q) = \frac{|\{t' \in q_u(q(\boldsymbol{t}_u)) : \exists t \in q_u(\boldsymbol{t}_u) \text{ s.t. } t' \equiv t\}|}{|q_u(q(\boldsymbol{t}_u))|}, \tag{6}$$

$$\mathrm{Rec}(q) = \frac{|\{t' \in q_u(q(\boldsymbol{t}_u)) : \exists t \in q_u(\boldsymbol{t}_u) \text{ s.t. } t' \equiv t\}|}{|q_u(\boldsymbol{t}_u)|} \tag{7}$$

are the Precision and Recall of $q_u(q(\boldsymbol{t}_u)) = q_u(\boldsymbol{t}')$ with respect to $q_u(\boldsymbol{t}_u)$. Intuitively, F1 measures how closely q_u on the synthetic stream matches q_u on the original stream.

4.2 Solution representation

We consider pre-processing queries including a restricted category of operators and having a simple linear structure. The rationale is to simplify the search space for the evolutionary optimization by including only the operators that are useful for producing streams fitting our needs (privacy, fidelity, and semantics). Moreover, by imposing a linear structure for the query (*i.e.*, a single chain of operators), we make it simpler and hence more transparent. We leave to future work the exploration of more complex structures.

4.2.1 Operators. In the set of available operators, we include templates for a filter operator, for two map operators (one for duplicating tuples and one for adding noise to attributes), and for one aggregate operator (see Section 2.1 for the parameters and functions defined by each such operator). For all of them, we fix part of the parameters and let the evolutionary optimization find optimal values for the remaining parts.

Filter This operator filters out tuples based on the comparison of one attribute a against a threshold v, *i.e.*, its predicate p_F has one of these forms: $t.a < v$ or $t.a > v$. We denote the operator by `filter(`$a \bullet v$`)`, where $\bullet$ is either $<$ or $>$. We include this operator as it constitutes the primary mechanism for eliminating outliers or non-relevant regions of the data space. Intuitively, extreme or very infrequent values may have limited impact on the overall analytical result, while increasing disclosure risk due to their uniqueness or sparsity.

Noise This map operator adds Gaussian noise to one attribute a of the input tuple t. Its map function f_M works as $t \mapsto \{t'\}$ where:

$$\forall a_i, t'.a_i = \begin{cases} t.a_i(1 + \sim N(0,\sigma)) & \text{if } a_i = a \\ t.a_i & \text{otherwise,} \end{cases} \tag{8}$$

with σ being the standard deviation of the Gaussian distribution and $\sim N(0,\sigma)$ representing the sampling from the Gaussian distribution with mean 0 and standard deviation

σ. We denote it by `noise(`a`,`σ`)`. The noise operator implements a perturbation-based anonymization mechanism by injecting stochastic noise into the value of a single numeric attribute. Intuitively, this operator is useful when exact values are not essential to the semantics of the query and small perturbations do not alter its logical outcome, while still effectively reducing the risk of attribute disclosure.

Duplication This map operator duplicates an input tuple t with probability ρ. Its map function f_M hence works as:

$$t \mapsto \begin{cases} \{t, t\} & \text{if } \sim U([0,1]) \leq \rho \\ \{t\} & \text{otherwise,} \end{cases} \tag{9}$$

where $\sim U([0,1])$ denotes the sampling from the uniform distribution in $[0,1]$. We denote it by `duplicate(`ρ`)`. We include this operator because tuple duplication may act as a proxy for fan-out, enabling diverse downstream effects (e.g., Noise) without requiring explicit DAG structuring of the query, while still allowing the exploration of the impact of increased data volume and key frequency on downstream operators.

Aggregate This aggregate operator applies a stateful transformation by considering recent tuples and producing a tuple with the value of one attribute a of them with an aggregation (average, min, or max) of the corresponding input values, *i.e.*, it performs the moving average, min, or max of a. It works with a sliding window of $WS = c$ tuples and $WA = 1$ and a key-by $K = T(\boldsymbol{t}_u) \setminus \{\tau, a\}$ (*i.e.*, all attributes except τ and the aggregated one a), and a function f_O doing $\gamma = (t_1, \ldots, t_c) \mapsto t'$ where:

$$\forall a_i, t'.a_i = \begin{cases} \bullet(\{t.a_i, t \in \gamma\}) & \text{if } a_i = a \\ t_c.a_i & \text{otherwise,} \end{cases} \tag{10}$$

with $\bullet$ being one of avg, min, max and t_c being the last (most recent) tuple in the window γ. We denote it by `aggregate(`a`,`c`, `$\bullet$` )`. We include this operator as it can attenuate short-term fluctuations of attribute values, reducing the exposure of fine-grained temporal information while preserving the structural dynamics of the data.

Note that none of these operators drops/adds attributes from input tuples. The requirement that $T(\boldsymbol{t}'_u) \supseteq T(q_u)$ is automatically satisfied, as $T(\boldsymbol{t}'_u) = T(\boldsymbol{t}_u)$ and $T(\boldsymbol{t}_u) \supseteq T(q_u)$ by problem statement.

4.2.2 Grammar and mapping. We designed a CFG for representing pre-processing queries with the structure and operators described above. We include symbols and rules for representing problem-agnostic constructs (e.g., operators and numerical values) and others for representing variable parts depending on the problem at hand (the attributes). We represent this CFG in Figure 1 in the Backus-Naur form (BNF), a compact notation where production rules are grouped by non-terminal symbol and the first group has, as left-hand-side, the starting symbol l_0. Within groups, the vertical bar | separates alternative production rules. The starting symbol is ⟨ops⟩, which represents a concatenation of operators.

⟨ops⟩ ::= ⟨op⟩ | ⟨op⟩`->`⟨ops⟩
⟨op⟩ ::= ⟨filter⟩ | ⟨dupl⟩ | ⟨noise⟩ | ⟨aggr⟩
⟨filter⟩ ::= `filter(`⟨attr⟩⟨cond⟩⟨val⟩`)`
⟨dupl⟩ ::= `duplicate(`⟨prob⟩`)`
⟨noise⟩ ::= `noise(`⟨attr⟩`,`⟨sigma⟩`)`
⟨aggr⟩ ::= `aggregate(`⟨attr⟩`,`⟨count⟩`,`⟨type⟩`)`
⟨cond⟩ ::= `<` | `>`
⟨type⟩ ::= `avg` | `min` | `max`
⟨attr⟩ ::= $\overbrace{a_1 \mid \ldots \mid a_n}^{\text{problem dependent}}$
⟨val⟩ ::= ⟨d⟩`.`⟨d⟩⟨d⟩`E`⟨sign⟩⟨d⟩
⟨d⟩ ::= `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`
⟨sign⟩ ::= `+` | `-`
⟨prob⟩ ::= `0.`⟨d⟩
⟨sigma⟩ ::= `0.01` | `0.05` | `0.10` | `0.25`
⟨count⟩ ::= `3` | `4` | `5` | `6` | `7` | `8` | `9` | `10`

Figure 1: Grammar for pre-processing queries, in BNF, for a stream t_u with attributes $a_1, \ldots, a_n$.

While performing the optimization with NSGA-II, we map a genotype (*i.e.*, a production tree) to a pre-processing query passing through the intermediate representation of the concatenation (through `->`) of operators expressed as a string of $\mathcal{L}(G)$. We present an example of this mapping process in Figure 2.

We remark that, since the grammar leaves open both the structure and the parameters of the query, the EA optimizes both starting from the random initial starting point set by the initialization procedure.

5 Evaluation

We first present two use cases, then describe the experimental setup, and finally discuss the results. All code and scripts are available at https://zenodo.org/records/20348184.

5.1 Use cases

Both use cases share the following characteristics and competing objectives. (a) The input stream contains sensitive information and cannot be shared as is (*Privacy*). (b) The

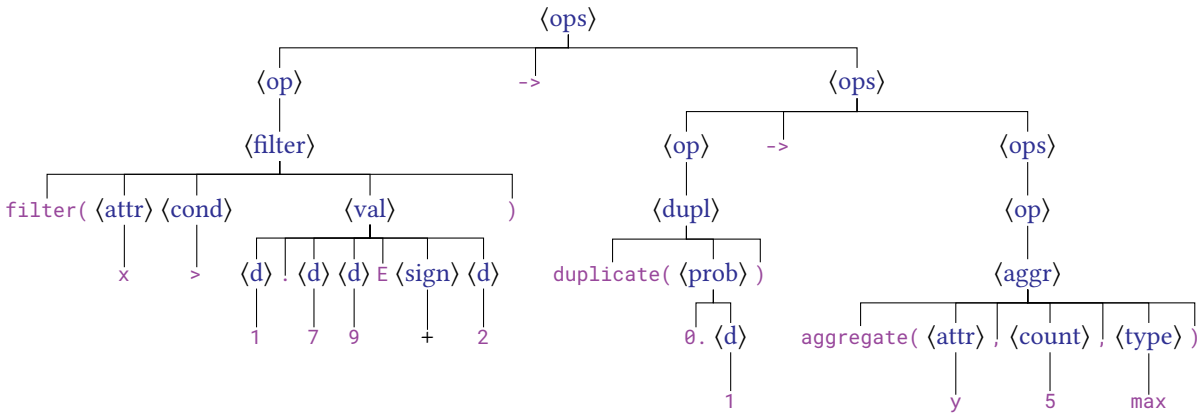

Figure 2: A production tree obtained from the grammar of Figure 1 instantiated with three attributes a_1 = `x`, a_2 = `y`, a_3 = `z` (with `z` not actually used), and corresponding to the intermediate representation `filter(x>1.79E+2) -> duplicate(0.1) -> aggregate(y,5,max)` of a pre-processing query with three operators.

target application must run and be optimized in distributed and/or edge settings, so representative inputs are needed to preserve performance-relevant characteristics (e.g., key skew, rates, and value distributions) (*Fidelity*). (c) Query output must remain reliable (in both cardinality and values) so that optimizations leading to $q^\star$ (see Section 3) do not compromise timely detection or decision-making (*Semantics*).

5.1.1 GeoLife. In this mobility-monitoring use case, the data owner wants to share traces without exposing users' trajectories. Privacy is challenged by the sensitivity and uniqueness of spatio-temporal traces, including rare trajectories and geographically isolated observations. Fidelity matters because mobility analytics are often deployed at scale (e.g., smart-city/edge platforms) and rely on area-specific characteristics (density, skew, and temporal locality) to tune execution parameters. Semantics matters because outputs support timely actions (e.g., alerts or planning decisions).

We use the GeoLife GPS trajectory dataset [36] as $\boldsymbol{t}_u$. It contains traces from ≈ 180 users over multiple years; most points are in Beijing, but trajectories also span many other locations, yielding spatial outliers. We construct a stream of ≈ 55 000 tuples aggregated at 60 min resolution. Each tuple includes a `user` identifier, a UNIX-milliseconds timestamp τ, and average spatial coordinates (`avg_X`, `avg_Y`) for that hour. This mix of dense regions and extreme outliers makes the use case suitable to test whether SHIELD can improve privacy (by mitigating long-tail risks) without distorting mobility patterns in high-density areas.

Our template query q_u emulates a location-based alerting and monitoring application in two stages:

(1) **Temporal aggregation:** compute the moving average of `avg_X` and `avg_Y`, grouped by `user`, with window size 3 h and slide 1 h.
(2) **Urban center filter:** keep tuples whose coordinates (`avg_X`, `avg_Y`) are within distance $r = 2\,000$ of the median point $(\bar{x}, \bar{y})$ computed on the original stream.

Applied to $\boldsymbol{t}_u$, this query produces ≈ 11 200 output tuples (selectivity ≈ 20.2 %).

5.1.2 AirQuality. In this environmental-monitoring use case, privacy concerns arise because pollutant concentrations and recurring patterns may indirectly reveal information about the monitored site (e.g., operational activity or emission profiles). Fidelity matters because these pipelines are often deployed on resource-constrained/distributed infrastructures (e.g., edge gateways) and must preserve performance-relevant stream properties. Semantics matters because outputs are alerts meant to support timely interventions.

We use the AirQuality dataset from the UCI Machine Learning Repository [30] as $\boldsymbol{t}_u$. The original data has ≈ 9 300 hourly records collected over about one year (March 2004–February 2005) by an air-quality multi-sensor device deployed at road level in an Italian city. To exercise semantically richer (thus more challenging) keyed streaming queries, we reshape the time series into a parallel-stream setting: map consecutive hourly records into one reference day and assign them to ≈ 390 distinct `sensorID`s, yielding concurrent sensor observations (*i.e.*, we transform ≈ 390 days of data from one sensor into one day of data from ≈ 390 sensors). The resulting stream lets us stress key-by behavior and key-dependent performance effects. Each event is timestamped in UNIX milliseconds as τ and carries key `sensorID` with pollutant values (e.g., CO and NO_2). The attributes are: τ, `sensorID`, `co`, `pt08_s1_co`, `nmhc`, `c6h6`, `pt08_s2_nmhc`, `nox`, `pt08_s3_nox`, `no2`, `pt08_s4_no2`, `pt08_s5_o3`, `t`, `rh`, `ah`.

Our template query q_u is a three-stage pipeline:

(1) **Initial filtering:** keep tuples with `co` ≥ 2 ∧ `no2` ≥ 40.

(2) **Temporal aggregation:** compute the moving average of `co` and `no2`, grouped by `sensorID`, with window size 3 h and slide 1 h.
(3) **Final alerting:** on the aggregated stream, keep tuples with `co` $\geq 5 \wedge$ `no2` ≥ 100 to emit alert events.

Applied to $\boldsymbol{t}_u$, the pipeline produces ≈ 300 alerts (selectivity $\approx 3.2\%$), acting as an early-warning mechanism for sustained pollution episodes. The adopted thresholds align with established health guidelines for scenarios that may escalate into legal breaches or harmful daily exposure levels [32].

Note that, for this use case, we tune the grammar to evolve candidate solutions by including only attributes used by q_u.

5.1.3 Performance metrics and equivalence criterion. For both use cases, we define a set $\boldsymbol{m}$ of performance metrics which consider the flow of tuples through operators of q_u, rather than hardware-dependent usage metrics (e.g., CPU consumption). This way, we capture structural and workload-related properties of the dataflow induced by the stream $\boldsymbol{t}_u$ and query q_u semantics, and make $\boldsymbol{m}$ independent of hardware. Namely, we define one pair of metrics for the ingress and for each operator (see Section 2.1), computed on the streams they output: one counts tuples and one counts distinct key values (`user` and `sensorID`); we measure over consecutive 1 h time buckets and average over the entire execution of q_u on the stream.

For the equivalence criterion $\equiv$, we say that, for two output tuples t and t', $t \equiv t'$ if both the following conditions are satisfied: (a) $t.\tau = t'.\tau \wedge t.a_{\text{key}} = t'.a_{\text{key}}$, where a_{key} is the key attribute (`user` for GeoLife and `sensorID` for AirQuality), and (b) $\frac{1}{|T|}\sum_{a\in T}\frac{(t.a-t'.a)^2}{\min(\epsilon,\sigma_a^2)} \leq d_{\min}$, where $T = T(\boldsymbol{t}_u)\backslash\{\tau, a_{\text{key}}\}$ and σ_a^2 is defined as in Equation (1). That is, we say two tuples are equivalent if they are "close enough" in the attribute space. We set $d_{\min} = 0.1$ for GeoLife and $d_{\min} = 0.15$ for AirQuality.

5.2 Environment and parameters

We ran the experiments on a virtualized Linux server with an Intel® Xeon® Gold 6418H processor, 92 logical cores, and 220 GB of main memory. The software environment uses Ubuntu 24.04 LTS and Java 21 LTS.

We use JGEA [22] for optimization and Liebre as SPE to execute streaming queries [20]. We evaluate fitness concurrently across candidate individuals, *i.e.*, by running multiple candidate queries at once.

We run the optimization five times per dataset using predefined seeds (1 to 5) due to the stochastic nature of EC. Each evolutionary run uses $n_{\text{gen}} = 50$ iterations, population $n_{\text{pop}} = 100$, and tournament size $n_{\text{tour}} = 2$. We enforced a minimum tree height of $h_{\min} = 4$ and a maximum tree height of $h_{\max} = 16$. For the privacy objective f_{privacy} (see Section 4.1), we set $k = 50$ nearest neighbors for $N_{t'}^k$ to compute the dispersion-based anonymity metric.

5.3 Goals

We assess SHIELD along three experimental dimensions.

(1) **Expressiveness of the template grammar.** We consider three configurations of increasing expressive power for the operators available to the pre-processing query: (a) filter only (F), (b) filter+map (FM), and (c) filter+map+aggregate (FMA). We differentiate variants by changing the rules for non-terminal ⟨op⟩ in grammar G (see Figure 1). In F (⟨op⟩ ::= ⟨filter⟩), anonymization is only through tuple removal. In FM (⟨op⟩ ::= ⟨filter⟩ | ⟨noise⟩ | ⟨dupl⟩), tuples can be altered individually, but transformations remain per-tuple. In FMA (⟨op⟩ ::= ⟨filter⟩ | ⟨noise⟩ | ⟨dupl⟩ | ⟨aggr⟩), transformations may jointly involve multiple tuples, enabling structural modifications across windows or groups.
(2) **Privacy strictness.** We evaluate two variants of privacy objective f_{privacy}: a worst-case formulation with $g = \max$ and a softer percentile-based formulation with $g = p_{99}$ (see Section 4.1). The former emphasizes extreme tuples and enforces strong worst-case protection; the latter tolerates a small fraction of high-risk points.
(3) **Objective ablation.** Along with the main three-objective setting ($f_{\text{privacy}}, f_{\text{fidelity}}, f_{\text{semantics}}$), we study two two-objective variants: ($f_{\text{privacy}}, f_{\text{fidelity}}$) and ($f_{\text{privacy}}, f_{\text{semantics}}$). This ablation isolates the effect of each fidelity dimension and assesses how adding or removing one objective reshapes privacy–fidelity trade-offs.

By combining operator expressiveness, privacy strictness, and objective formulations across both datasets, we obtain a comprehensive view of how anonymization strategies affect the interplay between privacy, fidelity, and semantics.

To assess how well the best solution fits live data anonymization, we also study throughput (number of processed tuples per second) and latency (time between output-tuple emission and reception of the latest input tuple required to produce that output) on two sample queries.

5.4 Results and discussion

Before analyzing SHIELD solutions, we quantify the baseline privacy risk by evaluating the privacy objective on the unmodified stream $\boldsymbol{t}_u$, *i.e.*, by computing $f_{\text{privacy}}(q_\varnothing)$ for the empty pre-processing query $q_\varnothing$, which yields $\boldsymbol{t'} = q_\varnothing(\boldsymbol{t}_u) = \boldsymbol{t}_u$. On GeoLife, this baseline score is 0.92 under $g = p_{99}$ and 0.45 under $g = \max$, showing that the strict worst-case formulation is substantially more challenging in this dataset. On AirQuality, the baseline is 0.87 under $g = p_{99}$ and 0.74 under $g = \max$. Hence, while max remains stricter, the gap between max and p_{99} is smaller than in GeoLife; the following results

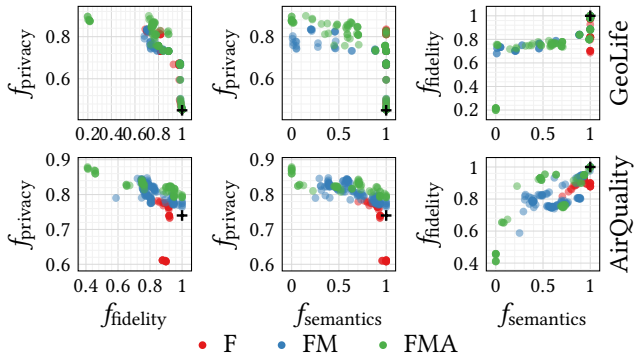

Figure 3: Pairwise fitness values for all non-dominated solutions in the final populations of 5 runs, for the three grammar variants with $g = \max$ in the privacy objective. The black marker (+) corresponds to the empty query $q_\varnothing$.

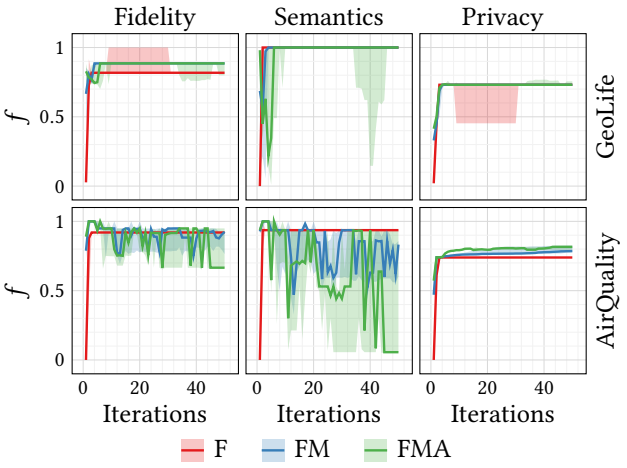

Figure 4: Fitness progression during evolution for the solution at the 75-th percentile of f_{privacy}, for the three grammar variants with $g = \max$. In each plot, the line is the median across 5 runs and the shaded area is the interquartile range.

show how this affects achievable privacy–fidelity–semantics trade-offs.

5.4.1 Expressiveness. Figure 3 reports the main results of the three expressiveness variants of SHIELD (• F, • FM, • FMA) on the two use cases. It shows pairwise objective values $(f_{\text{privacy}}(q), f_{\text{fidelity}}(q), f_{\text{semantics}}(q))$ for each non-dominated query q in the final population of each evolutionary run. The black marker (+) denotes the baseline empty query $q_\varnothing$. For this analysis, we used $g = \max$ in the privacy objective.

We draw three main observations from Figure 3. First, regardless of use case and expressiveness variant, there is a clear trade-off between privacy and fidelity, and between privacy and semantics. Near-fidelity-optimal solutions (rightmost in the left plot of each row in Figure 3), *i.e.*, with score ≈ 1, achieve lower privacy: ≈ 0.68 for GeoLife and ≈ 0.8 for AirQuality (vs. $q_\varnothing$ at 0.45 and 0.74, though), compared with privacy-maximizing solutions (≈ 0.92 and ≈ 0.9). Similarly, for semantics (rightmost in the middle plot), semantics-best solutions reach privacy ≈ 0.85 for GeoLife and ≈ 0.8 for AirQuality. Conversely, fidelity and semantics are positively correlated. For both use cases, the rightmost plot of Figure 3 shows that better semantics corresponds to better fidelity, also reflected by $q_\varnothing$ (+) scoring optimally in both.

Second, although the qualitative interplay between objectives is stable, grammar expressiveness affects overall results. Solutions from FMA (•) generally achieve better values on trade-off objectives (privacy vs. one of the others) than solutions with fewer operators (• and •). Visually, the non-dominated front area (*pairwise hypervolume*) in plots involving privacy is the largest for FMA. The least expressive variant, F (•), is worst in objective values, pairwise hypervolume, and coverage of the ideal Pareto front. In practice, F offers fewer diverse user choices than FMA. Moreover, in AirQuality, some F solutions are worse than baseline $q_\varnothing$ in privacy. Overall, Figure 3 supports using an expressive search space with different operator types.

Third, Figure 3 shows that the two use cases differ in how strongly fidelity and semantics are positively correlated: this correlation is stronger in AirQuality. This is most visible in the rightmost plot, with subtler effects in the other two. Namely, the best privacy achieved at highest fidelity and highest semantics is the same for AirQuality (≈ 0.8), but differs in GeoLife (≈ 0.68 and ≈ 0.85).

Progression of the fitness. While Figure 3 shows end-of-run results, Figure 4 shows progress *during* evolution. For clarity, at each iteration we report objective values for a single solution: the one at the 75-th privacy percentile in the population, denoted $q_{\text{privacy},75}$.

Figure 4 confirms the differences across variants and objective interplay. In particular, F (—) can barely improve $q_{\text{privacy},75}$: filter-only expressiveness keeps the population in regions where privacy gains strongly hurt fidelity and semantics. With the other two variants (— and —), especially on AirQuality, evolution slowly but steadily improves $q_{\text{privacy},75}$. With FMA (—), the EA keeps exploring and finds higher-privacy solutions, with lower fidelity and semantics.

Operator distribution. To better understand how the most effective variant, *i.e.*, FMA, uses all operators, Figure 5 reports the percentage of each operator category during evolution: `filter` (—), `duplicate` and `noise` (—), and `aggregate` (—).

The plots in Figure 5 suggest two main observations. First, for both use cases, optimization takes a few iterations (≈ 10

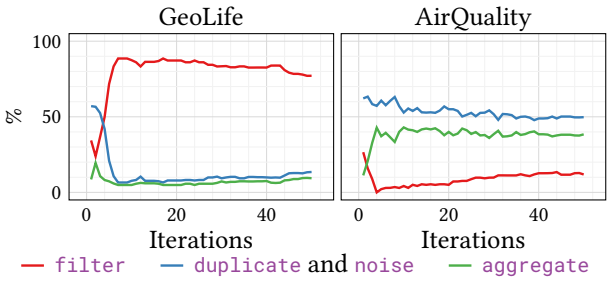

Figure 5: Progression during evolution of operator percentages (mean across 5 runs) for FMA with g = max.

for GeoLife, ≈ 5 for AirQuality) to reach a stable operator distribution in the population. After that, percentages are flat or nearly flat, with a slight trend for AirQuality; this suggests that a larger optimization budget could further improve results, as the EA is still slowly refining queries.

Second, final operator distributions differ markedly across use cases (see Table 1). This suggests the EA can steer search, within the large FMA space, toward regions that best fit each problem. In GeoLife, most high-privacy solutions mainly use filters (—), due to structural outliers: removing tuples causing extreme deviations is the most direct way to improve worst-case anonymity. In AirQuality, filter-based solutions are not dominant. Since extreme tuples are not the main source of privacy risk, gains come from value-level transformations such as noise injection (—) and aggregation (—).

Optimization time. Table 1 reports mean μ_{psot} and standard deviation σ_{psot} of *per-solution optimization time*, *i.e.*, total evolutionary wall time divided by the number of evaluated solutions (always 5 000 = $n_{\text{gen}} n_{\text{pop}}$). We report this for both use cases, the three expressiveness variants, and the two strictness variants (discussed below). Optimizations were not run in a fully isolated software environment: most runs, including across variants, ran in parallel.

For GeoLife, expressiveness has limited impact on μ_{psot}: FMA is ≈ 30 % slower than F. For AirQuality, the impact is larger: FMA has μ_{psot} ≈ 325 % higher than F. We explain this by the higher use of `duplicate`, `noise`, and `aggregate` in this case (see Figure 5); these operators, especially `duplicate` and `aggregate`, are computationally heavier.

Analysis of evolved queries. To assess how best solutions found in each run keep up with live anonymization, we report in Figure 6 throughput and latency results for two candidate pre-processing queries in the FMA, g = max setup. The sample GeoLife query first duplicates tuples with probability 30 %, then aggregates `avg_Y` and filters both `avg_X` and `avg_Y`. The AirQuality query aggregates `nox` over a window of size 3 using `min`, then aggregates `co` over a window of size 5 using `avg`. Figure 6 shows throughput (in kt s^{-1}) and

	g	Grammar	%F	%M	%A	μ_{psot}	σ_{psot}
GeoLife	max	F	100.00	0.00	0.00	130.67	4.50
		FM	90.05	9.95	0.00	166.78	3.87
		FMA	80.04	13.04	6.92	162.16	11.07
	p_{99}	F	100.00	0.00	0.00	124.22	1.57
		FM	88.41	11.59	0.00	155.08	5.74
		FMA	78.42	14.24	7.33	159.66	12.34
AirQuality	max	F	100.00	0.00	0.00	122.06	3.05
		FM	16.55	83.45	0.00	265.77	70.67
		FMA	8.89	53.08	38.03	398.02	62.25
	p_{99}	F	100.00	0.00	0.00	123.52	3.56
		FM	4.55	95.45	0.00	323.94	107.51
		FMA	0.75	53.69	45.57	227.80	2.58

Table 1: Operator distribution (%F for `filter`, %M for `duplicate` and `noise`, %A for `aggregate`) and mean μ_{psot} with standard deviation σ_{psot} of fitness evaluation duration (ms) across configurations and use cases.

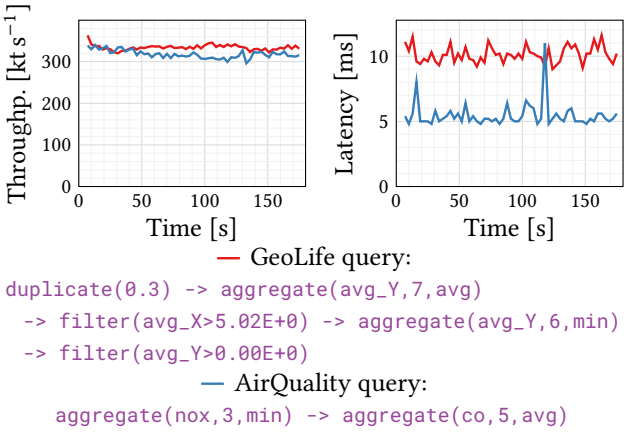

Figure 6: Throughput and latency of two candidate pre-processing queries in the FMA, g = max setup during live anonymization.

latency (in ms) of the two queries obtained by running each query on the corresponding dataset five times and averaging the results. We collected values on a 1 s basis, over 3 min runs, excluding 5 s warm-up and cool-down phases.

While actual throughput and latency in real-world deployments depend on available resources, these results show that the evolved solutions can handle hundreds of thousands of tuples per second with latencies of a few milliseconds on commodity hardware. While both candidate queries sustain similar throughput, we observe lower latency for AirQuality (—)

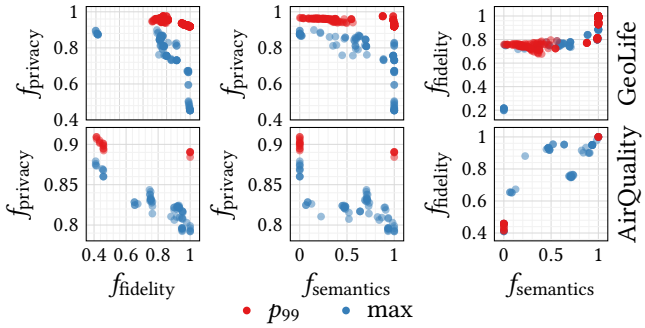

Figure 7: Fitness values for all non-dominated solutions in the final populations of 5 runs for FMA with $g = \max$ or $g = p_{99}$ in the privacy objective (f_{privacy} vs. f_{fidelity} on the left, f_{privacy} vs. $f_{\text{semantics}}$ on the right).

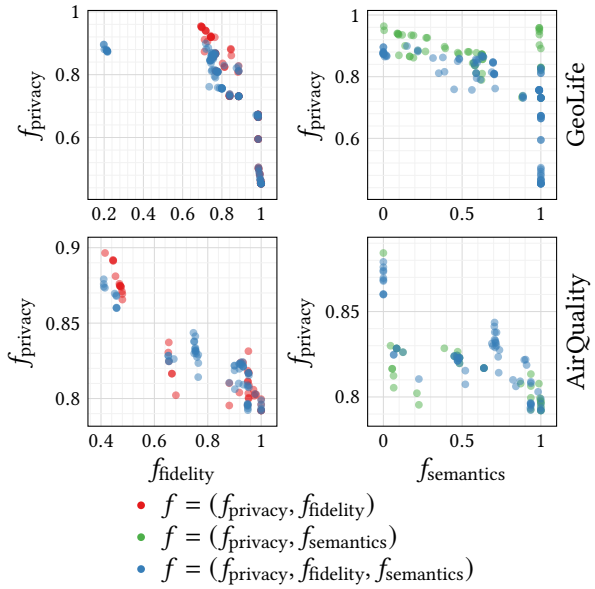

Figure 8: Fitness values for all non-dominated solutions in the final populations of 5 runs for FMA with $g = \max$, using two objectives as fitness (privacy/fidelity on the left, privacy/semantics on the right). Solutions obtained with three objectives are also shown, projected onto the same objective pairs.

than for GeoLife (—), as expected from the shorter AirQuality pipeline (2 operators vs. 5 operators). The latency values remain close and in the same order of magnitude. These figures are promising: even on more constrained hardware, a one-order-of-magnitude reduction would leave throughput in the tens of thousands of tuples/second, and many sensor deployments do not require sub-second reporting.

5.4.2 Privacy strictness. Figure 7, using the same layout as Figure 3, shows objective values for final solutions evolved with less strict aggregation $g = p_{99}$ (•) and FMA expressiveness. For reference, we include results for $g = \max$ (•).

We remark that directly comparing privacy values across the two variants is not meaningful: by definition, for any query q, $f_{\text{privacy}}(q; \max) \geq f_{\text{privacy}}(q, p_{99})$. What matters is how strictness affects the search for solutions that are also good in the other objectives. Figure 7 again shows an apparent difference between datasets. For GeoLife, the final solution set qualitatively approximates the ideal Pareto front similarly for both max (•) and p_{99} (•). For AirQuality, solutions with max diversify less in objective space, clustering in two groups with low and high fidelity/privacy. This matches earlier observations on evolved operator distributions (Figure 5), where the two use cases end with different settings, and is confirmed by final percentages in Table 1. For GeoLife, distributions for $g = \max$ (80 %, 13 %, 7 %, for `filter`, `duplicate`/`noise`, `aggregate`) and $g = p_{99}$ (79 %, 14 %, 7 %) are very similar; for AirQuality they are not: 9 %, 53 %, 38 % vs. 1 %, 54 %, 45 %. In the latter case, a softer privacy measure makes filters much less useful. We leave to future work how to improve diversification in objective space, possibly with quality–diversity (QD) EAs [4].

5.4.3 Objective ablation. Since fidelity and semantics are well correlated, we explored using two objectives instead of three to drive evolutionary search. Figure 8 shows results obtained with FMA and $g = \max$. We compare solutions from $f = (f_{\text{privacy}}, f_{\text{fidelity}})$ (•) and $f = (f_{\text{privacy}}, f_{\text{semantics}})$ (•), *i.e.*, privacy with one of the other two correlated objectives. For reference, we also show in the same objective space the solutions obtained with three objectives (•).

For GeoLife, removing one objective generally improves privacy for the same level of the other objective: • points stay "higher" than • on the left plot, and • points stay "higher" than • on the right. This aligns with the findings discussed above, where fidelity and semantics are not strongly correlated: not asking the EA to pursue both lets it focus more on privacy without harming the remaining objective.

For AirQuality, there is no clear privacy improvement: the non-dominated subset in the two cases is similarly determined by markers of both colors. Here, removing semantics gives no clear advantage because semantics is strongly correlated with fidelity (and vice versa): pursuing one effectively pursues both. Only at very low fidelity values does removing the semantics objective improve privacy: • points are "higher" than • in the bottom-left plot of Figure 8.

All in all, these experiments suggest it may be possible to automatically tune some SHIELD parameters (namely g and the objective set) to better fit the data, for example by running a few test queries to sample objective interplay. We leave this idea for future work.

6 Related work

The need for scalable data processing in pipelines for aggregation, validation, and monitoring is one of the original motivations for SP, with early systems and use cases dating back to the early 2000s. In this sense, the problem of designing effective and efficient data-transformation pipelines is not new. To the best of our knowledge, however, the automatic synthesis of pipelines that explicitly balance privacy and utility (*i.e.*, in our setting, fidelity and semantics, see Section 3) has not been addressed end-to-end. Existing work typically addresses only subsets of the challenges discussed in Section 1, rather than their joint treatment. This is a key limitation for adoption in industrial systems, where these challenges co-occur in practice.

Stream based data pipelines for monitoring and validation. Works such as [10, 13] discuss the design and implementation of stream-based pipelines for monitoring and validation, including efficiency and scalability aspects. These studies, however, often assume monolithic ownership of pipeline development (*i.e.*, a single team, or even a single person, handling all stages), whereas in practice data owners and data engineers are often distinct teams with different responsibilities and constraints. Therefore, while highly relevant as complementary use cases, they simplify the organizational setting highlighted in Section 1.

Privacy-preserving data synthesis. Most approaches in this space operate on finite datasets (e.g., single- [35] or multi-table relational data [25]), do not explicitly account for runtime performance and scalability, or optimize privacy as a single objective. As a result, the privacy–utility trade-off is often only partially modeled. An even larger and flourishing body of works are concerned with data-generation where only utility matters, possibly for coping with data scarcity [8] or lack of fairness [33]. Interestingly, the vast majority of approaches for data synthesis are based on deep-learning frameworks, ranging from variational autoencoders [25], to diffusion models [28], to large language models [3]. Transparency of the generator is often not a goal, nor easily achievable with those frameworks: in our case, the evolved pre-processing queries are compact and inspectable for users to make informed decisions based not only on the privacy–utility trade-offs of the generator but also on its interpretability.

Automatic generation of stream queries. To the best of our knowledge, existing studies on automatic query generation mainly target the synthesis of queries that enforce specific semantics, often from input/output examples [12, 17, 19, 21, 23]. Some approaches also rely on EC [2]. These works, however, address a problem that is orthogonal to ours. Our goal is not to find a query that imposes new output semantics; rather, we seek a pre-processing query that preserves schema compatibility with the downstream workload while enforcing privacy requirements and retaining the data properties needed for downstream use. In other words, we target transformations that preserve intended downstream behavior while modifying privacy and utility characteristics.

Complementary work on bridging SP and EC has also been discussed in a different context, namely using SP to pipeline the exploration of candidate solutions in EC [15].

7 Conclusions and future work

We presented SHIELD, a framework for automatically synthesizing pre-processing stream queries that enable data sharing for optimization while preserving utility. The motivation is that modern data pipelines are often developed and tuned by multiple parties (e.g., data owners, platform teams, and external optimization experts): sharing representative data is essential for validation and tuning, but raw streams are often sensitive and cannot be disclosed, while heavily sanitized data may lose the structural properties needed for reliable optimization. By combining grammar-guided representations with multi-objective evolutionary search, SHIELD produces alternative privacy–fidelity–semantics trade-offs for users to inspect and select. Across two use cases, results show that evolved queries can balance the objectives and scale to online processing workloads.

Future work includes improving efficiency and expressiveness of the search: simplifying candidate queries before fitness evaluation [16], using QD evolutionary methods for better diversification [4], extending the representation from linear operator chains to richer graph-like structures [27], and moving beyond single-stream transformations to multi-stream scenarios with joins and correlations. We also plan to investigate automated tuning of framework parameters (e.g., objective set and strictness) from preliminary data analysis [5], and nested optimization strategies for numerical parameters in grammar-based pipelines [26].

Acknowledgments

Work supported by the Marie Skłodowska-Curie Doctoral Network project RELAX-DN, funded by the European Union under Horizon Europe 2021-2027 Framework Programme Grant Agreement number 101072456, the Chalmers AoA Energy projects DEEP and INDEED, and the Swedish Energy Agency (SESBC) project TANDEM.

References

[1] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Wittle. 2015. The dataflow model: a practical approach to balancing correctness, latency, and

cost in massive-scale, unbounded, out-of-order data processing. *Proc. Endowment* 8, 12 (2015), 1792–1803.

[2] Giulio Appetito, Eric Medvet, and Vincenzo Gulisano. 2025. Automated discovery of cep applications with evolutionary computing. In *Proceedings of the 19th ACM International Conference on Distributed and Event-Based Systems*. 33–38.

[3] Vadim Borisov, Kathrin Seßler, Tobias Leemann, Martin Pawelczyk, and Gjergji Kasneci. 2022. Language models are realistic tabular data generators. *arXiv preprint arXiv:2210.06280* (2022).

[4] Antoine Cully and Yiannis Demiris. 2017. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation* 22, 2 (2017), 245–259.

[5] Matheus Camilo da Silva, Gabriel Marques Tavares, Eric Medvet, and Sylvio Barbon Junior. 2024. Problem-oriented automl in clustering. *arXiv preprint arXiv:2409.16218* (2024).

[6] Ashraf Darwish, Aboul Ella Hassanien, and Swagatam Das. 2020. A survey of swarm and evolutionary computing approaches for deep learning. *Artificial intelligence review* 53, 3 (2020), 1767–1812.

[7] Kenneth De Jong. 2017. Evolutionary computation: a unified approach. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 373–388.

[8] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. 2017. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633* (2017).

[9] flink [n.d.]. Apache Flink. https://flink.apache.org Accessed:2024-02-28.

[10] Marios Fragkoulis, Paris Carbone, Vasiliki Kalavri, and Asterios Katsifodimos. 2023. A survey on the evolution of stream processing systems. *The VLDB Journal* (2023), 1–35.

[11] Georgi Ganev and Emiliano De Cristofaro. 2025. The Inadequacy of Similarity-Based Privacy Metrics: Privacy Attacks Against "Truly Anonymous" Synthetic Datasets. In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 4007–4025.

[12] Lars George, Bruno Cadonna, and Matthias Weidlich. 2016. Il-miner: Instance-level discovery of complex event patterns. *Proceedings of the VLDB Endowment* 10, 1 (2016), 25–36.

[13] Vincenzo Gulisano, Magnus Almgren, and Marina Papatriantafilou. 2014. Online and scalable data validation in advanced metering infrastructures. In *IEEE PES Innovative Smart Grid Technologies, Europe*. IEEE, 1–6.

[14] Vincenzo Gulisano, Alessandro Margara, and Marina Papatriantafilou. 2024. On the Semantic Overlap of Operators in Stream Processing Engines. https://doi.org/10.1145/3652892.3654790

[15] Vincenzo Gulisano and Eric Medvet. 2024. Evolutionary computation meets stream processing. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer, 377–393.

[16] Thomas Helmuth, Nicholas Freitag McPhee, Edward Pantridge, and Lee Spector. 2017. Improving generalization of evolved programs through automatic simplification. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 937–944.

[17] Sarah Kleest-Meißner, Rebecca Sattler, Markus L Schmid, Nicole Schweikardt, and Matthias Weidlich. 2023. Discovering Multi-Dimensional Subsequence Queries from Traces–From Theory to Practice. In *BTW 2023*. Gesellschaft für Informatik eV, 511–533.

[18] John R Koza. 1994. Genetic programming as a means for programming computers by natural selection. *Statistics and computing* 4 (1994), 87–112.

[19] Yan Li and Tingjian Ge. 2021. Imminence monitoring of critical events: A representation learning approach. In *Proceedings of the 2021 International Conference on Management of Data*. 1103–1115.

[20] Liebre SPE. 2022. https://github.com/vincenzo-gulisano/Liebre. Accessed:2024-02-28.

[21] Alessandro Margara, Gianpaolo Cugola, and Giordano Tamburrelli. 2014. Learning From the Past: Automated Rule Generation for Complex Event Processing. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS)*. 47–58.

[22] Eric Medvet, Giorgia Nadizar, and Luca Manzoni. 2022. JGEA: a modular java framework for experimenting with evolutionary computation. In *Proceedings of the genetic and evolutionary computation conference companion*. 2009–2018.

[23] Nijat Mehdiyev, Julian Krumeich, David Enke, Dirk Werth, and Peter Loos. 2015. Determination of Rule Patterns in Complex Event Processing Using Machine Learning Techniques. *Procedia Computer Science* 61 (2015), 395–401. https://doi.org/10.1016/j.procs.2015.09.168

[24] Dimitris Palyvos-Giannas, Katerina Tzompanaki, Marina Papatriantafilou, and Vincenzo Gulisano. 2023. Erebus: Explaining the outputs of data streaming queries. In *Very Large Data Base*, Vol. 16. 230–242.

[25] Daniele Panfilo, Alexander Boudewijn, Sebastiano Saccani, Andrea Coser, Borut Svara, Carlo Rossi Chauvenet, Ciro Antonio Mami, and Eric Medvet. 2023. A deep learning-based pipeline for the generation of synthetic tabular data. *Ieee Access* 11 (2023), 63306–63323.

[26] Federico Pigozzi, Laura Nenzi, and Eric Medvet. 2025. BUSTLE: a versatile tool for the evolutionary learning of STL specifications from data. *Evolutionary Computation* 33, 1 (2025), 91–114.

[27] Berfin Sakallioglu, Giorgia Nadizar, Luca Manzoni, and Eric Medvet. 2025. Evolving Typed Token Processing Networks. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2177–2181.

[28] Juntong Shi, Minkai Xu, Harper Hua, Hengrui Zhang, Stefano Ermon, and Jure Leskovec. 2024. Tabdiff: a mixed-type diffusion model for tabular data generation. *arXiv preprint arXiv:2410.20626* (2024).

[29] storm [n.d.]. Apache Storm. http://storm.apache.org Accessed:2024-02-28.

[30] Saverio Vito. 2008. Air Quality. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C59K5F.

[31] Peter A Whigham. 1995. Inductive bias and genetic programming. (1995).

[32] World Health Organization. 2021. *WHO global air quality guidelines: particulate matter (PM2.5 and PM10), ozone, nitrogen dioxide, sulfur dioxide and carbon monoxide.* Technical Report. World Health Organization. https://www.who.int/publications/i/item/9789240034228

[33] Depeng Xu, Shuhan Yuan, Lu Zhang, and Xintao Wu. 2018. Fairgan: Fairness-aware generative adversarial networks. In *2018 IEEE international conference on big data (big data)*. IEEE, 570–575.

[34] Bing Xue, Mengjie Zhang, Will N Browne, and Xin Yao. 2015. A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on evolutionary computation* 20, 4 (2015), 606–626.

[35] Hengrui Zhang, Jiani Zhang, Balasubramaniam Srinivasan, Zhengyuan Shen, Xiao Qin, Christos Faloutsos, Huzefa Rangwala, and George Karypis. 2023. Mixed-type tabular data synthesis with score-based diffusion in latent space. *arXiv preprint arXiv:2310.09656* (2023).

[36] Yu Zheng, Hao Fu, Xing Xie, Wei-Ying Ma, and Quannan Li. 2011. *GeoLife GPS Trajectory Dataset - User Guide* (geolife gps trajectories 1.1 ed.). https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/