Short Paper

Icewafl: A Configurable Data Stream Polluter

Christoph Schinninger
opta data Österreich GmbH
Linz, Austria
c.schinninger@optadata.at

Fabian Panse
University of Augsburg
Augsburg, Germany
fabian.panse@uni-a.de

Constantin Kühne
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
constantin.kuehne@student.hpi.uni-potsdam.de

Lisa Ehrlinger
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
lisa.ehrlinger@hpi.de

ABSTRACT

Due to their high velocity, data streams pose additional challenges for analysis and quality assurance compared to static data. To select a suitable computational model to analyze a data stream, or the right data quality tool to clean it, it is important to have benchmark data that reflects the characteristics of the stream such as trends and seasonality, but also potential data errors. Although several data polluters have been developed to inject errors into existing data, none of them supports the creation of temporal data errors as they occur in data streams.

Therefore, we propose Icewafl, a data polluter that allows the injection of temporal errors to create benchmark datasets. Icewafl is built on top of Apache Flink to enable seamless integration with existing data stream pipelines and efficient processing of large-scale data. We show that Icewafl can be used to evaluate (1) the error detection capabilities of data quality tools and (2) the robustness of forecasting methods.

1 INTRODUCTION

With the growing amount of data collected by sensors, the processing and analysis of data streams is becoming increasingly important [14]. Sensor data plays a key role in many different areas, such as healthcare [42], manufacturing [25], or environmental monitoring [26, 33]. As these sensors are often managed by different parties (e.g., crowdsensing [5, 30]), this can lead to great heterogeneity and contamination within the collected data. However, data streams can originate not only from sensors, but also from simulation programs, such as those used for climate models [21] or digital twins [41]. The requirement to efficiently process data streams has led to the development of special processing frameworks such as Apache Flink [6], Apache Spark Streaming [2, 44], Kafka Streams [38], or Hazelcast [18].

Data streams are often analyzed with machine learning (ML) or online forecasting methods. An important criterion in the selection of a suitable method for a specific task is the robustness of this method against various types of data errors, such as those resulting from defective or miscalibrated sensors [26]. The wide range of potential data errors also makes it essential to measure the quality or clean data streams, either with dedicated data quality (DQ) tools [15] or specific cleaning algorithms (including ML models [23]). In both cases, it is essential to have suitable benchmark data to evaluate the effectiveness of DQ measurement and data cleaning tasks, as well as the robustness of ML models.

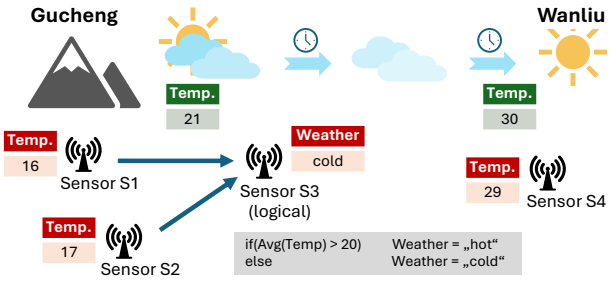

Figure 1: Motivating scenario with data error dependencies

Data polluters[1] allow users to deliberately insert errors into data to enable the systematic evaluation of how algorithms perform against different types of data errors [37]. In contrast to static benchmark datasets, data polluters allow the creation of customized benchmark data that matches specific use cases, aligning with the *fitness for use* principle of data quality [40]. Although several data polluters have been proposed in the past (e.g., [3, 10, 20, 36, 39]), none of them support the pollution of streaming data or can generate errors that affect the temporal characteristics of data streams.

Motivating example. As outlined in Figure 1, errors in data streams often depend on parallel or previously processed tuples and the errors associated with them (as also discussed in [4, 12]). In our scenario, the measurements of the two weather sensors S1 and S2 are influenced by the same confounding factors (the shadow of the mountains and clouds) due to their spatial proximity. As these clouds drift, they subsequently impact the measurements from sensor S4 after a time delay. Sensor S3, being a logical sensor that derives its values from S1 and S2, inherits any measurement errors present in these source sensors, creating a chain of error propagation.

Challenges. Based on the shortcomings of benchmark data in general [36], we identified the following key challenges for data pollution in streams: (C1) the characteristics of data streams can introduce *temporal errors* that do not occur in static databases, and the severity of these errors can vary over time [4]. (C2) As the motivating example shows, real-world errors can have *complex dependencies* that are neither easy to model nor to generate. (C3) The *configuration requirements* for a data polluter vary greatly between users, making it challenging to balance ease of use with the power to generate complex error scenarios.

[1] Note that we distinguish between *data pollution* and *data synthesis*. While the latter refers to the creation of a benchmark dataset from scratch [37], the former refers to injecting errors into an existing dataset.

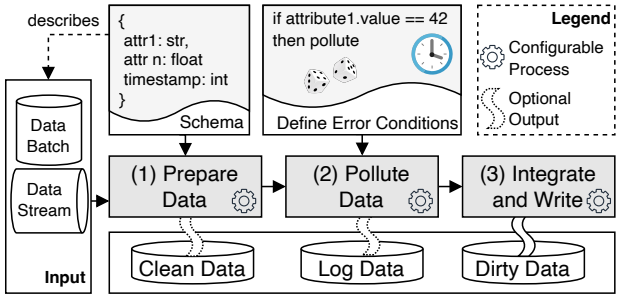

Figure 2: The configurable pollution process of Icewafl

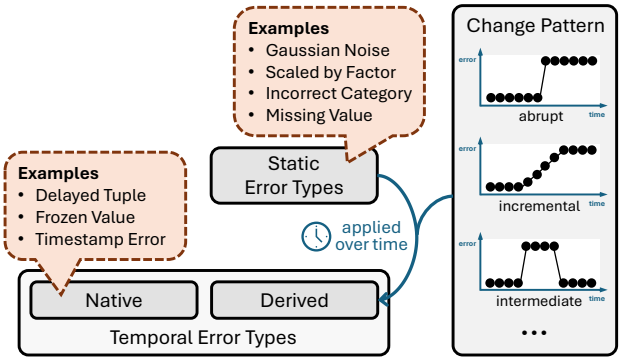

Figure 3: Error types supported by Icewafl

Contribution. In this paper, we present Icewafl[2] (Inserting Customizable Errors with Apache Flink), a configurable polluter for data streams. Our contributions are:

(1) *Temporal error types* that are either temporal by definition (e.g., a delayed tuple) or result from combining static error types with change patterns (e.g., time-depending noise). By introducing these types (cf. Figure 3), we address Challenge C1.
(2) A *novel pollution model* for data streams based on simple polluters that can be combined into pollution pipelines. Each polluter introduces specific errors based on given conditions. This modular approach balances simplicity for inexperienced users (who can use predefined error types and standalone polluters) with high expressiveness for experts who need to model complex real-world error patterns. Expert users can create sophisticated error scenarios through composite polluters, nested conditions, and integrated sub-pipelines (Challenge C3), while also modeling temporal dependencies between errors (Challenge C2).
(3) A *comprehensive set of experiments* to demonstrate the utility of Icewafl in the generation of benchmark data streams for the evaluation of DQ tools and forecasting methods.

We implemented Icewafl on top of the popular data stream processing framework Apache Flink[3] to enable distributed pollution and a simple integration into existing data streaming pipelines.

Structure. In Section 2, we formally define our novel pollution model for data streams. The correctness and suitability of Icewafl's pollution capabilities are evaluated in Section 3 and a comparison to existing benchmark generators is provided in Section 4. Section 5 concludes the paper and gives an outlook on future work.

2 DATA STREAM POLLUTION

Figure 2 illustrates Icewafl's end-to-end pollution process. Its major objective is to allow the *reproducible* creation of *temporal* and *diverse* data errors such that the resulting benchmark datasets reflect typical data stream characteristics. However, it can also be used to pollute batch data. The pollution workflow is divided into three steps: (1) the input is parsed and prepared, (2) the actual pollution is executed and optionally logged to reproduce it, and (3) the polluted data stream(s) are integrated in case of multiple streams and persisted. The input to the first two steps are the schema of the data stream and the error configurations used to pollute the data. Our formal definitions for the pollution process are based on Ioannou and Velegrakis [24] and Foroni et al. [16].

[2]https://github.com/chri-schi/Icewafl
[3]https://flink.apache.org

2.1 Data Stream Handling

The pollution process can either take a real data stream or a data stream split into small batches (i.e., micro-batching) as input. Within our framework, each input is treated tuple-wise as a data stream. Based on Oliveira et al. [34] and Davari et al. [13], we define a multivariate data stream D as a sequence of tuples:

$$D = t_1, t_2, \ldots, t_n \tag{1}$$

where $|D| = n$ is the number of tuples inside the data stream. In the case of unbounded streams, n is unknown. The schema of the stream consists of k attributes $A = A_1, \ldots, A_k$ where each tuple has a value for each attribute. We refer to data as multivariate if $k > 1$, and assume that each attribute A_i has a domain $dom(A_i)$. It holds $dom(A) = dom(A_1) \times \ldots \times dom(A_k)$. As it is a data stream, we expect the schema to also contain a timestamp attribute.

In the preparation step, tuples receive a unique identifier (ID) and a replicated timestamp τ, which are not affected by the pollution process and used to identify the tuples throughout the entire pollution process. The assigned ID enables direct comparison between the original (clean) data and its polluted version, serving as a ground truth reference for each tuple. While the original timestamp may be subject to pollution and is used for the output stream, τ is used as event time during the pollution process and is not part of the final output.

2.2 Data Stream Pollution Model

Icewafl uses data polluters to inject errors into the input data. Each polluter p is defined by a triple $\langle e, c, A^p \rangle$ where e is an error function, c is a condition that determines whether an error is injected or not, and $A^p \subseteq A$ is the subset of attributes targeted for pollution. To capture the temporal characteristics of data streams, Icewafl provides the event time as an additional argument to these polluters which can affect the error function and the condition. Thus, an error function $e\colon dom(A) \times 2^A \times T \rightarrow dom(A)$ takes a tuple, a set of attributes and the event time $\tau \in T$ as input and outputs a transformed tuple. In summary, a polluter p can also be defined as:

$$p(t, \tau) = \begin{cases} e(t, A^p, \tau), & \text{if } c(t, \tau) = true \\ t, & \text{else,} \end{cases} \tag{2}$$

where depending on the condition c, input tuple t is either polluted with the error e or returned unchanged.

As shown in Equation 2, we use conditions to control the pollution behavior. In alignment with Schelter et al. [39], data errors can be inserted (i) completely at random (ii) depending on the values to be polluted [29], or (iii) depending on the values of

the input tuple that are not to be polluted. In addition, Icewafl supports temporal conditions based on the event time and composite conditions that allow to conjoin any of the aforementioned conditions.

Icewafl distinguishes between *static* and *temporal error types* (see Figure 3). Static errors types are independent of the event time. Temporal errors types are either *native* or *derived*. While native error types are special error functions that are temporal by definition (e.g., a delayed tuple), derived error types result from combining a static error type with a pattern of change over time (based on the changed patterns introduced in [17]). In the latter case, the event time is used as an additional input argument for the otherwise static error function (e.g., noise is added based on the hour of the day) or a static error is applied with an increased/decreased probability during a specific time interval [4] (e.g., over the next five minutes, the probability of missing values increases from 40% to 90%).

2.2.1 Pollution pipelines. Since a single polluter is usually not sufficient to model real-world error patterns, we allow the composition of multiple polluters within a *pollution pipeline*. In Icewafl, we distinguish between two different types of polluters: (1) standard polluters that introduce specific data errors and (2) composite polluters that structure the pollution pipeline. Composite polluters can register an arbitrary number of standard polluters that actually insert the errors. Through nesting, composite polluters allow modeling more complex pollution strategies, for example, two error types that always occur together or a set of errors that are mutually exclusive.

A *pollution pipeline* P is a sequence of o polluters $p_1, p_2, ..., p_o$. The pipeline applied to an input tuple t results in an output tuple $t' = P(t,\tau) = p_o(p_{o-1}(\ldots p_1(t,\tau)\ldots,\tau),\tau)$. A polluted data stream D^p consists of tuples from the original data stream D that have passed the entire pollution pipeline P, i.e., $D^p = \{P(t,\tau) \mid t \in D\}$.

2.2.2 Integration scenarios. To simulate scenarios where multiple data streams are integrated, Icewafl is able to split the pollution pipeline into a predefined number of independent sub-pipelines which can finally be merged again. By applying different polluters to each sub-pipeline, we are able to model stream-specific error patterns. In addition, merging several pipelines can lead to additional error types, such as fuzzy duplicates [32, 34].

2.3 Algorithmic Discussion

The entire pollution workflow is summarized in Algorithm 1. In the preparation step, each tuple is assigned a new ID (Line 2) and the timestamp ts is replicated, with the replica denoted as τ (Line 3). Subsequently, m (overlapping) sub-streams are extracted from the previously enriched data stream (Line 4). In the pollution step, each tuple of each sub-stream is passed through the sub-stream's corresponding pollution pipeline (Lines 5-9) and polluted by an individual polluter (Line 9) if it meets the required condition (Line 8). In the final step (Lines 10-11), the m polluted sub-streams are combined by the union of their tuples (without τ but with an additional sub-stream identifier) and both the clean and polluted data stream are returned. The complexity of the individual steps is $O(n)$ (Step 1), $O(n \cdot m \cdot l)$ (Step 2) and $O(n \cdot m \cdot log(n \cdot m))$ (Step 3), which leads to a total complexity of $O(n \cdot m \cdot (\frac{1}{m} + l + log(n \cdot m)))$ with $n = |D|$ and l being the size of the largest pollution pipeline. The algorithm is deterministic (and thus reproducible) if the same seeds are used for polluters using random error functions and/or conditions.

Algorithm 1: Icewafl's data stream pollution process

Input: data stream D, m pollution pipelines $P_1, \ldots, P_m$
Output: clean data stream D_c, polluted data stream D_p

```
#Step 1: Prepare data
1  for t = (a_1, ..., a_k, ts) ∈ D do
2      t ← newIdentifier(t) ⊕ (a_1, ..., a_k, ts);
3      t ← (id, a_1, ..., a_k, ts) ⊕ ρ_τ(ts);
4  D_1, ..., D_m ← createOverlappingSubStreams(D,m);
   #Step 2: Pollute data
5  for i ← 1 to m do
6      for t = (id, a_1, ..., a_k, ts, τ) ∈ D_i do
7          for p = (e, c, A^p) ∈ P_i do
8              if c(t, τ) == true then
9                  t ← e(t, A^p, τ);
   #Step 3: Integrate and output pipeline results
10 D_p ← ⋃_{i=1}^{m} {(id, i, a_1, ..., a_k, ts) | (id, a_1, ..., a_k, ts, τ) ∈ D_i};
11 D_p ← sortByTimestamp(D_p);
12 return D, D_p
```

3 EXPERIMENTAL EVALUATION

In this section, we show the usefulness of Icewafl in two ways: by evaluating the functionality of a DQ tool in Section 3.1 and by analyzing the robustness of three different forecasting methods against data errors in Section 3.2. Finally, we demonstrate that Icewafl's pollution process introduces only minimal runtime overhead in Section 3.3. We used the following two datasets for these experiments:

The *Beijing Multi-Site Air-Quality Dataset* [8] contains data from 12 air-quality monitoring sites located in China and weather data from their closest weather stations. The dataset consists of 420,768 tuples and 18 attributes. It was collected every hour for almost four years, from March 1, 2013 to February 28, 2017.

The *Wearable Device Dataset* [27] contains sensor data from activity trackers worn by volunteers in their everyday lives. It consists of multiple tables that present the sensor data in different levels of detail. For our experiments, we combined the HRTable table (containing heart rate recordings) and the activity data stored in the MainTable for the volunteer with ID 0216-0051-NHC. We re-sampled the HRTable data to match the time granularity of the MainTable data. The final dataset spans over a period of nearly 11 days (264.75 hours) from February 26 to March 7, 2016.

3.1 Experiment 1: Data Quality Tools

In the first experiment, we show that Icewafl can be used to evaluate the functionality of DQ tools. In other words, Icewafl can pollute data streams such that the injected errors can be successfully detected with these DQ tools. Note that there is a mutual dependency between the evaluation of data polluters and DQ tools. On the one hand, the correctness of a pollution strategy (i.e., data error creation) can only be verified with a proper error detection mechanism (e.g., implemented in a DQ tool). On the other hand, the accuracy of a DQ tool's error detection mechanism can only be verified using a polluted dataset. Thus, this

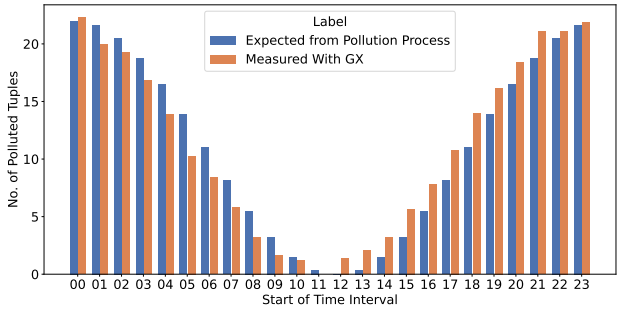

Figure 4: No. of errors measured with GX and no. of errors expected from the pollution process (per hour)

experiment shows not only the usefulness of Icewafl to evaluate DQ tools, but also that the configured pollution strategy is correctly applied on the input data.

We used the widely used DQ tool Great Expectations[4] (GX) to analyze the introduced error patterns. GX is a DQ monitoring platform that supports a user in defining constraints (so called *expectations*) on the data. Expectations can be viewed as data characteristics that are expected to be found in clean data and hence can be used to identify DQ issues. For our experiment, we used GX OSS (Open Source Software), which is available as Python package.

We evaluated three different pollution scenarios that we created using the wearable device dataset. The experiment shows that Icewafl allows to define complex error scenarios on data streams, which can be successfully detected by GX. For all error scenarios, we repeated the pollution process 50 times since Icewafl's error conditions introduce probabilities and are therefore non-deterministic. Each resulting output data stream is checked independently with GX and the overall result is the average obtained by all evaluations.

3.1.1 Random temporal errors. In the first scenario, we used temporal conditions to model an error pattern that changes over the time of a day (t) using a sinusoidal function [7]: $p(t) = 0.25 \cdot cos(\frac{\pi}{12} \cdot t) + 0.25$. Here, $cos(\frac{\pi}{12} \cdot t)$ is one daily cycle, the 0.25 multiplier reduces the error probability, and +0.25 sets the probability to $[0, 0.5]$. The polluter introduces `null` values in the Distance attribute. We used `expect_column_values_to_not_be_null` as expectation to detect attribute values set to `null`. In total, we discovered an average of 259.6 errors with GX, leading to an average error proportion of 24.58% with a variance of 1.22%. Figure 4 shows the detected number of polluted tuples measured with GX (orange) as well as the expected number of polluted tuples (blue) per hour. We can see that the polluted data errors fit the data errors detected with GX very well, which means that the expectation-based error detection mechanism of GX is expressive enough to find these errors.

3.1.2 Software update. In this scenario, an erroneous update of the operating system installed on the wearable device is simulated. The scenario consists of multiple pollution types occurring together based on shared conditions. Thus, the concept of composite polluters is used. Figure 5 shows the Icewafl configuration used in this setup. A composite polluter ("Software Update") resides at the top of the hierarchical pollution pipeline, which uses a condition that indicates the date when the software update was

[4]https://greatexpectations.io

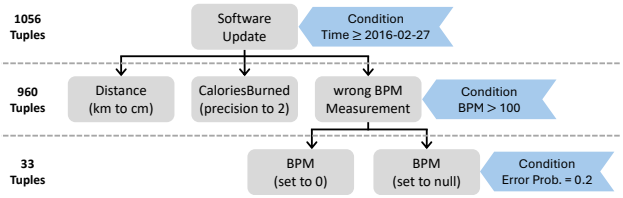

Figure 5: Setup for the software update scenario

installed on the device. For tuples recorded after the software update, the top-level polluter delegates the pollution process to three registered polluters: (1) one that transforms the unit of the Distance values from km to cm, (2) one that rounds the values of the CaloriesBurned attribute to a precision of 2, and (3) another composite polluter that manipulates the values of BPM if they exceed a value of 100. This composite polluter harbors two polluters that are executed in series. The first polluter sets a BPM value to 0 and the second polluter (activated with a probability of 20%) sets BPM values to null.

The evaluation of the generated data with GX turned out to be more complex for this experiment than it was in the random error experiment. Since more different pollution types were configured, more expectations were necessary to detect them: (i) We used `expect_column_pair_values_a_to_be_greater_than_b` to detect tuples were the value of the Step attribute is lower than the value of the Distance attribute. We assumed that after changing the unit of the Distance values from km to cm they become greater than the Step values. (ii) For the detection of the reduced precision in the CaloriesBurned attribute, we used the `expect_column_values_to_match_regex` expectation. We specified a regex pattern for valid CaloriesBurned values that allows a precision $p \leq 3$. (iii) Tuples whose BPM values were set to 0 were detected using the `expect_multicolumn_sum_to_equal` expectation and (iv) tuples with BPM=null were detected using the `expect_column_values_to_not_be_null` expectation. The former expectation applies to tuples that have a valid BPM value of 0. Here we assume that the sum of the values from the attributes ActiveMinutes, Distance, and Steps is also 0, i.e., the activity tracker was not worn. In the case a BPM value greater than 100 was set to 0 during the pollution process, the values of the other attributes usually still contain values that are greater than 0 and thus the expectation fires. Interestingly, the original data stream already contains two tuples that violate this constraint. In addition to the expected 26.4 dirty tuples (80% of 33), GX therefore detected two more tuples. Table 1 shows the number of errors that were expected to be inserted by Icewafl (including the already prevalent errors) along with the average number of errors that were measured with GX. As we can see, there are only minor differences.

3.1.3 Bad network connection. Data errors caused by a bad network connection were simulated using a polluter that delays tuples for an hour. The data stream was polluted only between 01:00 pm and 02:59 pm, controlled by a temporal condition that ensured the polluter activation only in this time interval. Within this temporal condition, an additional nested condition added a 20% probability of activating the time delay. The defined timespan includes 88 tuples, from which we expect 20% to become polluted by Icewafl. Thus, there should be around 17.6 delayed tuples in the polluted data stream. The evaluation was again evaluated using GX. We applied the `expect_column_values_to_be_increasing`

Table 1: No. of errors expected in the polluted data stream and no. of errors measured with GX for the software update scenario

Attribute	No. of Errors Expected after Pollution	No. of Errors Measured with GX
BPM=0 (Prob. 0.8)	26.4 (+2)	28
BPM=null (Prob. 0.2)	6.60	6
Distance	374	374
CaloriesBurned	960	960

expectation on the Time attribute to detect late tuples, since delayed tuples disturb the strictly increasing order of timestamps inside the data stream. Indeed, this expectation worked quite well as the average number of detected errors was 17.02 which is very close to the expected number of 17.6 tuples.

3.2 Experiment 2: Forecasting Methods

The objective of the second experiment was to show that Icewafl can be used to evaluate the susceptibility of forecasting methods to data errors. We selected three methods specifically suited for online analysis: ARIMA [22], ARIMAX [9], and Holt-Winters [22] and used the implementations from the Python library River[5] [31].

3.2.1 Data and pollution setup. The goal of the forecasting task was to predict the NO2 concentration for a period of 12 hours in three different Chinese regions Gucheng, Wanshouxigong, and Wanliu using the Beijing Air-Quality dataset. In total, there are 35,064 tuples per region without any missing tuples. Prior to the analysis, we imputed missing values for each region in the NO2 attribute using the forward/backward fill method `ffill` of Python Pandas[6]. We refer to each data stream per region in the following as D_r. Table 2 describes all data splits we performed to properly train, validate and test the models. The testing was performed on D_{eval}, D_{noise}, and D_{scale} respectively.

We evaluated the three forecasting methods by means of two different pollution scenarios: (i) noise that increased over time and (ii) scaling of numerical values that increased over time.

Temporally increasing noise. In the first scenario, multiplicative uniform noise was added to the data, yielding D_{noise}. First, a value was picked from the uniform distribution $U(a, b)$, with a being the lower and b being the upper bound of the distribution. Thereafter, depending on the result of a fair coin toss, the picked value is used as a factor to either increase or decrease the values of the polluted attribute. By updating a and b for each encountered tuple (event time τ_i), we ensured that the magnitude of the noise introduced into the data increased as time progressed:

$$\pi(ts_i) = \frac{\pi_{max} \cdot hours(\tau_i - \tau_0)}{hours(\tau_n - \tau_0)} \tag{3}$$

where π acts as a placeholder for either a or b, τ_0 and τ_n are the event time of the first and the last tuple of the data stream, and the constant π_{max} is the upper bound for the magnitude of the noise, i.e., the value of a parameter when processing the last tuple in a data stream. The function *hours* converts the difference of two timestamps to hours.

Temporally increasing scale error. In the second pollution scenario (see D_{scale}), we scaled numerical attribute values with the factor 0.125 for four-hour intervals. The applied polluter depends on two conditions (a probability condition and a temporal condition) and is only activated if both evaluate to true. The first condition activates the polluter with a prior probability of 0.01. The second condition actives the polluter according to

$$p(activation|\tau_i) = \frac{hours(\tau_i - \tau_0)}{hours(\tau_n - \tau_0)} \tag{4}$$

where *activation* is the probability of pollution and τ the event time of the first (τ_0), current (τ_i) and last (τ_n) tuple. Thus, activation probability increases over event time.

Table 2: Data splits to evaluate the forecasting methods

Data Stream	Description
D_{train}	1st year of D_r minus the last 12h
D_{valid}	last 12h of 1st year of D_r
D_{eval}	last year of D_r
D_{scale}	D_{eval} with all numerical attributes scaled by 0.125
D_{noise}	D_{eval} with all numerical attributes polluted with univariate noise

3.2.2 Model configuration and hyperparameter determination. Prior to the experiments, we determined suitable settings for the hyperparamters of the evaluated forecasting methods using grid search in combination with a 5-fold time series cross validation from the Python library scikit-learn[7]. The resulting parameter settings can be found in our GitHub-repository.

Holt-Winters and ARIMA belong to the class of auto-regressive models and rely only on previous measurements of the target variable when making forecasts. Thus, the input for these models during training and prediction were only the values of the NO2 attribute. ARIMAX, on the other hand, also considers correlations between attribute values in the same tuple. Thus, in addition to NO2 attribute, the ARIMAX models also received the attributes TEMP, PRESM, and WSPM as input as well as the sine and cosine encodings of the month and the hour of the event timestamp.

3.2.3 Model execution. The generalization error of the three forecasting models, or more precisely, its evolution over time, was examined on each of the three data streams D_{eval}, D_{noise}, and D_{scale} using the same procedure. The models received data tuple-wise in an online fashion. The training periods span 504 hours (= 3 weeks). After each learning period, a model made a forecast for the next 12 hours, yielding the evaluation data stream. After evaluation, the data collected in the evaluation period was released to be used for the next training period. Because the applied polluters are non-deterministic (through conditions and nesting), we repeated the evaluation of each model for ten different data streams, each polluted using the same configuration and reported mean values.

3.2.4 Results and discussion. Figure 6 (D_{noise}) and Figure 7 (D_{scale}) show the results for the region of Wanshouxigong. In both cases, the mean average error (MAE) generally increases as time progresses (with some exceptions). For the noise, however, this trend is much more significant than for the scale errors. Although ARIMAX performs slightly better at the beginning, all three forecasting methods behave very similarly on D_{scale}. The situation is different when we introduce noise into the data stream. Here, ARIMAX is significantly more robust than its two competitors. The results for the other regions are similar. In

[5]https://riverml.xyz/

[6]https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.ffill.html

[7]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html

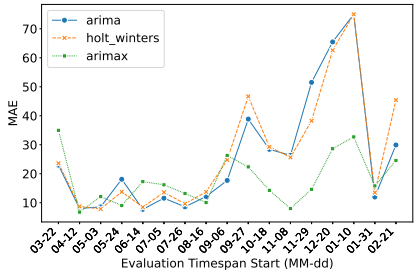

Figure 6: MAE for the data stream of the Wanshouxigong region polluted with temporally increasing noise

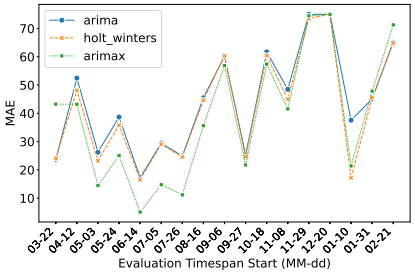

Figure 7: MAE for the data stream of the Wanshouxigong region polluted with temporally increasing scale errors

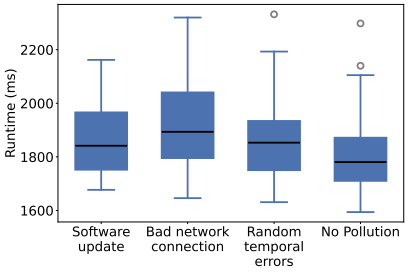

Figure 8: Runtime overhead (in ms) for the pollution scenarios from Section 3.1 compared to unpolluted baseline

summary, the experiment illustrates that the choice of the method should be made depending on the types of errors contained in the data stream.

3.3 Experiment 3: Runtime Overhead

In the third experiment, we took a look at the runtime overhead of Icewafl to see its effects on data stream processing frameworks. For this, we executed each scenario from Section 3.1 50 times on a cluster compute node with 2 Intel Xeon Gold 5220S CPU cores and 3 GB main memory. We compare the scenarios to a pipeline in which the same data stream was loaded and written to disk without polluting it. Figure 8 shows the results as box plots, yielding a small runtime increase of 3–7% for all pollution scenarios when compared to the pipeline without pollution. Thus, the overhead is marginal.

4 RELATED WORK

Over the past decades, various tools have been developed to generate structured (relational) [3, 10, 11, 16, 20, 39] and semi-structured [1, 36] benchmark datasets. While tools such as datumPIPE [1], Deimos [19], and GouDa [36] create new data from scratch before injecting errors in it, most other tools, such as BART [3], DWreck [10], and Jenga [39], pollute existing datasets with additional errors. However, almost all of these tools are limited to static data. The only tool that takes temporal aspects into account is EMBench++ [24]. Unlike Icewafl, however, it does not support temporal error patterns based on the event time as error argument or within error conditions. Instead, it offers simple modifiers to evolve tuples over time (e.g., by replacing a categorical value), resulting in a set of sequential snapshots of these tuples. In other words, it transforms a static tuple set into a data stream. DaPo [20] and F4U [16] are the only two tools that, like Icewafl, are able to distribute the pollution process. In contrast to Icewafl, however, they are based on Apache Spark and therefore do not provide a native data stream interface. Thus, to the best of our knowledge, Icewafl is the first data polluter that generates suitable benchmark data for data streaming scenarios.

5 CONCLUSION AND OUTLOOK

In this paper, we introduced Icewafl, a configurable data polluter that allows to inject temporal errors into data streams using a novel pollution model. To parallelize the pollution process in distributed environments and integrate it easily into existing streaming pipelines, we implemented Icewafl on top of Apache Flink. We evaluated the pollution capabilities and usefulness of Icewafl with two experiments: (1) to evaluate the error detection capability of a DQ tool, and (2) to analyze the robustness of three different forecasting methods against data errors. In addition, we demonstrated that the pollution process of Icewafl introduces only minimal runtime overhead.

We are planning the following steps for future work:

(1) A key challenge in generating realistic errors for data streams is modeling their complex temporal dependencies. Icewafl's current pollution model can handle some of these dependencies through the event time (e.g., in the error condition) and by polluting dependent tuples in the same sub-pipeline. However, modeling more sophisticated dependency patterns requires knowledge about the data stream's history and modeling of arbitrary relationships between past events. To address this, we plan to extend our model to incorporate time-dependent states of the data stream and dependencies between tuple-specific random variables.
(2) Managing inter-tuple dependencies is particularly challenging in distributed pollution scenarios. Here, we plan to leverage Flink's *keyed process functions* [35], as they enable the computation of (current and past) states of the data stream across individual computing nodes.
(3) After implementing these first two steps, we will conduct comprehensive performance evaluations, since the loss of tuple independence will make efficient parallelization conceptually and technically more challenging.
(4) To demonstrate the broad utility of Icewafl, we plan to deepen our study on DQ tools and explore new fields of application, such as testing whether existing approaches to time series synthesis [28, 43] are agnostic to different temporal error types and patterns. Such an analysis will reveal the suitability of synthesis approaches for different use cases: synthesis approaches that do not adopt errors from the real data stream are beneficial for applications that require clean data. On the other hand, approaches that preserve error patterns from the real data stream can be used to generate synthetic data that is suitable for error analysis tasks, such as training ML models for error detection.

ACKNOWLEDGMENTS

The research reported in this paper has been partly funded by BMK, BMAW, and the State of Upper Austria in the frame of the SCCH competence center INTEGRATE [(FFG grant no. 892418)] part of the FFG COMET Competence Centers for Excellent Technologies Programme as well as the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 495170629.

REFERENCES

[1] Samir Al-janabi and Ryszard Janicki. 2019. *Generation and Corruption of Semi-Structured and Structured Data.* Springer International Publishing, 159–169. https://doi.org/10.1007/978-3-030-11286-8_7

[2] Michael Armbrust, Tathagata Das, Joseph Torres, Burak Yavuz, Shixiong Zhu, Reynold Xin, Ali Ghodsi, Ion Stoica, and Matei Zaharia. 2018. Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. ACM, 601–613. https://doi.org/10.1145/3183713.3190664

[3] Patricia C. Arocena, Boris Glavic, Giansalvatore Mecca, Renée J. Miller, Paolo Papotti, and Donatello Santoro. 2015. Messing up with BART: Error Generation for Evaluating Data-Cleaning Algorithms. *Proceedings of the VLDB Endowment* 9, 2 (2015), 36–47. https://doi.org/10.14778/2850578.2850579

[4] João Marcelo Borovina Josko. 2019. A Formal Taxonomy of Temporal Data Defects. In *Data Quality and Trust in Big Data: 5th International Workshop, QUAT 2018, in conjunction with WISE 2018, Dubai.* Springer, 94–110.

[5] Andrea Capponi, Claudio Fiandrino, Burak Kantarci, Luca Foschini, Dzmitry Kliazovich, and Pascal Bouvry. 2019. A Survey on Mobile Crowdsensing Systems: Challenges, Solutions, and Opportunities. *IEEE Commun. Surv. Tutorials* 21, 3 (2019), 2419–2465. https://doi.org/10.1109/COMST.2019.2914030

[6] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink: Stream and Batch Processing in a Single Engine. *The Bulletin of the Technical Committee on Data Engineering* 38, 4 (2015).

[7] Debaditya Chakraborty and Hazem Elzarka. 2019. Advanced Machine Learning Techniques for Building Performance Simulation: A Comparative Analysis. *Journal of Building Performance Simulation* 12, 2 (2019), 193–207.

[8] Song Chen. 2019. Beijing Multi-Site Air-Quality Data. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5RK5G.

[9] Lin-Ya Chiu, Dan Jeric Arcega Rustia, Chen-Yi Lu, and Ta-Te Lin. 2019. Modelling and Forecasting of Greenhouse Whitefly Incidence Using Time-Series and ARIMAX Analysis. *IFAC-PapersOnLine* 52, 30 (2019), 196–201. https://doi.org/10.1016/j.ifacol.2019.12.521 6th IFAC Conference on Sensing, Control and Automation Technologies for Agriculture AGRICONTROL 2019.

[10] Ashish Chouhan, Ajinkya Prabhune, Paneesh Prabhuraj, and Hitesh Chaudhari. 2020. DWreck: A Data Wrecker Framework for Generating Unclean Datasets. In *2020 IEEE Sixth International Conference on Big Data Computing Service and Applications (BigDataService)*. 78–87. https://doi.org/10.1109/BigDataService49289.2020.00020

[11] Peter Christen and Dinusha Vatsalan. 2013. Flexible and Extensible Generation and Corruption of Personal Data. In *22nd ACM International Conference on Information and Knowledge Management, CIKM*. ACM, 1165–1168. https://doi.org/10.1145/2505515.2507815

[12] Tamraparni Dasu, Rong Duan, and Divesh Srivastava. 2016. Data Quality for Temporal Streams. *IEEE Data Eng. Bull.* 39, 2 (2016), 78–92.

[13] Narjes Davari, Bruno Veloso, Rita P. Ribeiro, Pedro Mota Pereira, and João Gama. 2021. Predictive Maintenance Based on Anomaly Detection Using Deep Learning for Air Production Unit in the Railway Industry. In *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*. 1–10. https://doi.org/10.1109/DSAA53316.2021.9564181

[14] Daniele Dell'Aglio, Emanuele Della Valle, Frank van Harmelen, and Abraham Bernstein. 2017. Stream Reasoning: A Survey and Outlook. *Data Science* 1, 1-2 (2017), 59–83.

[15] Lisa Ehrlinger and Wolfram Wöß. 2022. A Survey of Data Quality Measurement and Monitoring Tools. *Frontiers in Big Data* 5 (2022), 850611.

[16] Daniele Foroni, Matteo Lissandrini, and Yannis Velegrakis. 2021. Estimating the Extent of the Effects of Data Quality Through Observations. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 1913–1918. https://doi.org/10.1109/ICDE51399.2021.00176

[17] João Gama, Indre Zliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A Survey on Concept Drift Adaptation. *ACM Comput. Surv.* 46, 4 (2014), 44:1–44:37. https://doi.org/10.1145/2523813

[18] Can Gencer, Marko Topolnik, Viliam Ďurina, Emin Demirci, Ensar B Kahveci, Ali Gürbüz, Ondřej Lukáš, József Bartók, Grzegorz Gierlach, František Hartman, et al. 2021. Hazelcast Jet: Low-latency Stream Processing at the 99.99 th Percentile. *Proceedings of the VLDB Endowment* 14, 12 (2021).

[19] Alberto Hernández Chillón, Diego Sevilla Ruiz, and Jesús García Molina. 2020. Deimos: A Model-Based NoSQL Data Generation Language. In *Advances in Conceptual Modeling*. Springer International Publishing, 151–161.

[20] Kai Hildebrandt, Fabian Panse, Niklas Wilcke, and Norbert Ritter. 2020. Large-Scale Data Pollution with Apache Spark. *IEEE Transactions on Big Data* 6, 2 (2020), 396–411. https://doi.org/10.1109/TBDATA.2016.2637378

[21] Fei Hu, Chaowei Yang, John L Schnase, Daniel Q Duffy, Mengchao Xu, Michael K Bowen, Tsengdar Lee, and Weiwei Song. 2018. ClimateSpark: An In-Memory Distributed Computing Framework for Big Climate Data Analytics. *Computers & Geosciences* 115 (2018), 154–166.

[22] Robert J. Hyndman and George Athanasopoulos. 2018. *Forecasting: Principles and Practice* (2nd ed.). OTexts: Melbourne, Australia.

[23] Ihab F. Ilyas and Theodoros Rekatsinas. 2022. Machine Learning and Data Cleaning: Which Serves the Other? *J. Data and Information Quality* 14, 3, Article 13 (2022), 11 pages. https://doi.org/10.1145/3506712

[24] Ekaterini Ioannou and Yannis Velegrakis. 2019. EMBench^{++}: Data for a Thorough Benchmarking of Matching-Related Methods. *Semantic Web* 10, 2 (2019), 435–450. https://doi.org/10.3233/SW-180331

[25] Anja Klein and Wolfgang Lehner. 2009. Representing Data Quality in Sensor Data Streaming Environments. *Journal of Data and Information Quality (JDIQ)* 1, 2 (2009), 1–28.

[26] Anja Klein and Wolfgang Lehner. 2009. Representing Data Quality in Sensor Data Streaming Environments. *ACM J. Data Inf. Qual.* 1, 2 (2009), 10:1–10:28. https://doi.org/10.1145/1577840.1577845

[27] Weng Khong Lim, Sonia Davila, Jing Xian Teo, Chengxi Yang, Chee Jian Pua, Christopher Blöcker, Jing Quan Lim, Jianhong Ching, Jonathan Jiunn Liang Yap, Swee Yaw Tan, Anders Sahlén, Calvin Woon-Loong Chin, Bin Tean Teh, Steven G. Rozen, Stuart Alexander Cook, Khung Keong Yeo, and Patrick Tan. 2018. Beyond Fitness Tracking: The Use of Consumer-Grade Wearable Data from Normal Volunteers in Cardiovascular and Lipidomics Research. *PLOS Biology* 16, 2 (2018), 1–18. https://doi.org/10.1371/journal.pbio.2004285

[28] Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. 2020. Using GANs for Sharing Networked Time Series Data: Challenges, Initial Promise, and Open Questions. In *IMC '20: ACM Internet Measurement Conference*. ACM, 464–483. https://doi.org/10.1145/3419394.3423643

[29] Roderick JA Little and Donald B Rubin. 2019. *Statistical Analysis with Missing Data*. Vol. 793. John Wiley & Sons.

[30] Jinwei Liu, Haiying Shen, Husnu S. Narman, Wingyan Chung, and Zongfang Lin. 2018. A Survey of Mobile Crowdsensing Techniques: A Critical Component for The Internet of Things. *ACM Trans. Cyber Phys. Syst.* 2, 3 (2018), 18:1–18:26. https://doi.org/10.1145/3185504

[31] Jacob Montiel, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, Heitor Murilo Gomes, Jesse Read, Talel Abdessalem, and Albert Bifet. 2021. River: Machine Learning for Streaming Data in Python. *J. Mach. Learn. Res.* 22, 1, Article 110 (2021).

[32] Felix Naumann and Melanie Herschel. 2010. *An Introduction to Duplicate Detection.* Number 3 in Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

[33] Luís ML Oliveira and Joel JPC Rodrigues. 2011. Wireless Sensor Networks: A Survey on Environmental Monitoring. *Journal of Communications* 6, 2 (2011), 143–151.

[34] Paulo Oliveira, Fátima Rodrigues, and Pedro Rangel Henriques. 2005. A Formal Definition of Data Quality Problems.. In *Proceedings of the 2005 International Conference on Information Quality, ICIQ 2005.*

[35] PyFlink Documentation. 2025. *KeyedCoProcessFunction.* Retrieved February 18, 2025 from https://nightlies.apache.org/flink/flink-docs-release-1.17/api/python/reference/pyflink.datastream/api/pyflink.datastream.functions.KeyedCoProcessFunction.html

[36] Valerie Restat, Gerrit Boerner, André Conrad, and Uta Störl. 2022. GouDa - Generation of Universal Data Sets: Improving Analysis and Evaluation of Data Preparation Pipelines. In *Proceedings of the Sixth Workshop on Data Management for End-To-End Machine Learning (DEEM '22)*. ACM, Article Article 2, 6 pages. https://doi.org/10.1145/3533028.3533311

[37] Shazia Sadiq, Tamraparni Dasu, Xin Luna Dong, Juliana Freire, Ihab F. Ilyas, Sebastian Link, Miller J. Miller, Felix Naumann, Xiaofang Zhou, and Divesh Srivastava. 2018. Data Quality: The Role of Empiricism. *SIGMOD Rec.* 46, 4 (2018), 35–43. https://doi.org/10.1145/3186549.3186559

[38] Matthias J. Sax, Guozhang Wang, Matthias Weidlich, and Johann-Christoph Freytag. 2018. Streams and Tables: Two Sides of the Same Coin. In *Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics (BIRTE '18)*. ACM, Article 1, 10 pages. https://doi.org/10.1145/3242153.3242155

[39] Sebastian Schelter, Tammo Rukat, and Felix Biessmann. 2021. JENGA: A Framework to Study the Impact of Data Errors on the Predictions of Machine Learning Models. In *Proceedings 24th International Conference on Extending Database Technology (EDBT 2021) Industrial and Application Track.* https://www.amazon.science/publications/jenga-a-framework-to-study-the-impact-of-data-errors-on-the-predictions-of-machine-learning-models

[40] Richard Y. Wang and Diane M. Strong. 1996. Beyond Accuracy: What Data Quality Means to Data Consumers. *Journal of Management Information Systems* 12, 4 (1996), 5–33.

[41] Julius Weyl, Ulfia A. Lenfers, Thomas Clemen, Daniel Glake, Fabian Panse, and Norbert Ritter. 2019. Large-scale Traffic Simulation for Smart City Planning with Mars. In *Proceedings of the 2019 Summer Simulation Conference (SummerSim 2019)*. ACM, 2:1–2:12. https://dl.acm.org/citation.cfm?id=3374140

[42] Przemyslaw Woznowski, Alison Burrows, Tom Diethe, Xenofon Fafoutis, Jake Hall, Sion Hannuna, Massimo Camplani, Niall Twomey, Michal Kozlowski, Bo Tan, et al. 2017. SPHERE: A Sensor Platform for Healthcare in a Residential Environment. *Designing, Developing, and Facilitating Smart Cities: Urban Design to IoT Solutions* (2017), 315–333.

[43] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. 2019. Time-series Generative Adversarial Networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS*. 5509–5519. https://proceedings.neurips.cc/paper/2019/hash/c9efe5f26cd17ba6216bbe2a7d26d490-Abstract.html

[44] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. 2016. Apache Spark: A Unified Engine for Big Data Processing. *Commun. ACM* 59, 11 (2016), 56–65.