The Journal of Systems & Software 203 (2023) 111744

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

A Grey Literature Review on Data Stream Processing applications testing☆

Alexandre Vianna [a,b,*], Fernando Kenji Kamei [a,c], Kiev Gama [a], Carlos Zimmerle [a], João Alexandre Neto [a]

[a] Centro de Informatica, Universidade Federal de Pernambuco (UFPE), Brazil
[b] Instituto Federal de Pernambuco (IFPE), Brazil
[c] Instituto Federal de Alagoas (IFAL), Brazil

ARTICLE INFO

Article history:
Received 2 December 2022
Received in revised form 13 March 2023
Accepted 4 May 2023
Available online 11 May 2023

Dataset link: GLR research data (Original data)

Keywords:
Data streams
Grey literature
Software testing

ABSTRACT

Context: The Data Stream Processing (DSP) approach focuses on real-time data processing by applying specific techniques for capturing and processing relevant data for on-the-fly results, i.e. without necessarily requiring prior storage. Like in any other software, testing plays a vital role in the quality assurance of DSP applications. However, testing such kind of software is not a simple task. In this context, some factors that make challenging testing are message temporality, parallelism, data volume, complex infrastructure, variability, and speed of messages.
Objective: This work aims to map and synthesize industry knowledge and experience regarding DSP application testing. Specifically, we want to know about challenges, test purposes, test approaches, test data sources, and adopted tools.
Method: To achieve the objective, we performed a Grey Literature Review (e.g., blog posts, white papers, discussion lists, lecture themes at technical events, professional social networks, software repositories, and other web-published) on testing DSP applications. We searched the grey literature using Google's regular search engine in addition to specific searches on technical software development content websites. The selected studies were analyzed using qualitative and quantitative techniques.
Results: Results are based on evidence from 154 selected sources. The challenges for testing DSP applications are the complexity of DSP applications, test infrastructure complexity, timing, and data acquisition issues. The main test objectives identified are functional suitability, performance efficiency, reliability, and maintainability. The main test approaches reported: Performance Testing, Regression Testing, Property-Based Testing, Chaos Testing, and Contract/Schema Testing. The strategies adopted by practitioners to obtain test data: Historical Data, Production Data Mirroring, Semi-Synthetic Data, and Synthetic Data. We also report 50 tools used in various testing activities, which are used for: automating infrastructure, generating test data, test utilities, dealing with timing issues, load generation, simulation, and others. Furthermore, we identified gaps and opportunities for future scientific work.
Conclusion: This work selected and summarized content produced by practitioners regarding DSP application testing. We identified that knowledge, techniques, and tools intrinsic to the practice were not present in the formal literature, so this study helps reduce the gap between industry and academia on this topic. The document has delivered benefits to industry practitioners and academic researchers.

1. Introduction

Over the last few years, there has been a considerable increase in data generated by computer systems such as Internet of Things (IoT) devices, smartphones, and user interaction on social networks, websites, and applications. The demand for faster processing of these large volumes of data has also grown, and such requirements bring technological challenges that traditional techniques do not meet. Conventional data processing strategies are based on batch processing, which stores all data before

☆ Editor: Burak Turhan.

* Correspondence to: Centro de Informatica, Federal University of Pernambuco (CIn/UFPE), Av. Jornalista Anibal Fernandes, S/N, Cidade Universitaria, 50.740-560, Recife, PE, Brazil.
E-mail addresses: alexandre.vianna@igarassu.ifpe.edu.br (A. Vianna), fernando.kenji@ifal.edu.br (F.K. Kamei), kiev@cin.ufpe.br (K. Gama), cezl@cin.ufpe.br (C. Zimmerle), jasn3@cin.ufpe.br (J.A. Neto).

https://doi.org/10.1016/j.jss.2023.111744
0164-1212/

processing. However, this method adds latency and may not handle large volumes of data as fast as some applications require. For instance, low latency is critical in some applications like credit card fraud detection (Carcillo et al., 2018). Approaches in the context of Big Data have been proposed to address this problem (Philip Chen and Zhang, 2014; Kaisler et al., 2013), and, under those circumstances, Data Stream Processing (DSP) has emerged as a branch to handle real-time applications (Liu et al., 2014).

The DSP approach focuses on real-time data processing. It applies specific techniques for capturing and processing relevant data for on-the-fly results, i.e. without necessarily requiring prior storage (Stonebraker et al., 2005). This approach makes it possible to process data just after its generation, and it is recommended when there is a need to continuously query a data stream to detect patterns within a short time interval (Dell'Aglio et al., 2017). The short response time of DSP is valuable for some business sectors, such as the financial market, banking system, e-commerce, marketing, social networks, telephony industry, and infrastructure monitoring (Garofalakis et al., 2016). The solutions and tools around DSP are quickly advancing because of the continuous need to provide solutions to business demands. Today, DSP is already a reality in the software industry. Many tools have been created in-house by companies to meet strict real-time requirements (Gorawski et al., 2014), such as Storm by Twitter (Toshniwal et al., 2014), Kafka by LinkedIn Kreps et al. (2011), Millwheel by Google (Akidau et al., 2013), and Puma by Facebook (Chen et al., 2016). Other tools adopted by the industry emerged in the academic context, such as Spark, developed at the AMPLab at the University of California (Zaharia et al., 2016), and Flink, born from a university research project funded by the German Research Foundation (DFG) (Alexandrov et al., 2014). Nowadays, Storm, Kafka, Spark, and Flink have become open-source projects.

As with other software, testing is crucial to the quality of DSP applications. However, testing this type of software is a challenging task. In this scenario, testing is hampered by several factors, including message temporality, parallelism, data volume, data variability, and message speed. Designing a test infrastructure covering all these factors is difficult since it is challenging to simulate configurations in which errors manifest. Developing an efficient and effective DSP testing system is complex and requires people who are skilled and capable of planning. Several aspects are essential, such as a comprehensive understanding of the data to be processed and technical details of DSP infrastructure.

Nevertheless, the relevance of testing DSP in the industry can be identified in many informal sources such as tools discussion lists (Apache Software Foundation, 2022), lecture themes at technical events (Malaska, 2019; Wiesman, 2018; Gamov, 2020), Q&A websites (Anon, 2017), professional social networks (Waehner, 2022), blog posts (Aladev, 2021), open-source software repositories (authorjapps, 2019; Leopardi, 2017; Karau, 2016), and other web-published materials. Hence, it is noticeable that the community has been promoting some collective advances in specific issues, such as developing strategies, good practices, open-source tools, and libraries for testing DSP applications. Still, the industry know-how in DSP application testing is fragmented in several documents on the internet, such as websites, blogs, forums, and software repositories.

Although widely adopted in the industry, there is limited formal literature on testing DSP applications, as we identified through an ad-hoc search and reported previously (Vianna et al., 2019). The existing formal literature brings contributions addressing particular points of testing DSP applications by proposing strategies, techniques, and tools for particular cases. However, there is a lack of academic work approaching the subject more broadly, bringing together and relating the various aspects of the entire context. Furthermore, to the best of our knowledge, no academic work has promoted the selection and synthesis of informal literature produced by the industry on testing DSP applications. In order to fill this gap, we propose to carry out a Grey Literature Review (GLR) of wide scope, addressing several aspects of testing DSP applications and contributing to the curation and consolidation of the experience and knowledge produced by the industry. Approaching the theme more broadly, including several aspects, bringing the industry's perspective and relating it to academic contributions is a relevant complement to research in this area.

GLR is suitable for exploring complex topics where there is little consolidated formal literature and with emerging issues currently evolving, especially in the industrial context (Garousi et al., 2019). There is also high interest from the practitioners' community in this topic, as shown by a previously conducted qualitative study (Vianna et al., 2019). Thus, GLR is an appropriate approach and capable of bringing exciting results to the context of testing DSP applications. In this article, we conduct a GLR to select, analyze, and synthesize the relevant knowledge and experiences developed by practitioners working in the industry in the context of testing DSP applications. We are especially interested in knowing (i) what approaches are adopted, (ii) which tools have been used, (iii) what are the processes developed, (iv) which testing objectives, and (v) what challenges are faced by the industry to test DSP applications. We focused our analysis on the period between 2010 and 2020 (inclusive), which was marked by the evolution and consolidation of modern stream processing and the emergence of many of the major stream processing systems used nowadays (Carbone et al., 2020). We conducted searches on Google's regular search engine and specialized Information Technology (IT) websites to carry out the review. We selected 154 grey sources from 1667 search results. We extracted relevant information using coding techniques and analyzed it using a systematic qualitative data analysis approach from the selected sources.

This work contributes to providing practitioners and researchers with a single document summarizing the practical and theoretical knowledge developed by the community on DSP application testing, testing techniques, strategy for obtaining test datasets, a list of tools, and their purpose of usage for DSP application testing. This publication may be the first step for practitioners wishing to implement DSP application testing and a starting point for researchers in the area.

The paper is structured as follows. Core DSP concepts and a summary of the state-of-the-art in testing DSP applications are in Section 2. Section 3 describes the proposed Research Questions (RQ), while Section 4 describes the GLR method and its application. Section 5 presents the results, followed by a discussion on the findings in Section 6. Finally, Section 7 concludes our paper and presents directions for future work.

2. Background

This Section provides an overview and fundamental concepts about the DSP field. Afterwards, we discussed some studies related to testing DSP applications.

2.1. Data stream processing

The first studies about DSP were published in the 60s (Stephens, 1997). They presented some concepts about dataflow systems and the evolution of Database Management Systems (DBMSs). In the 70s, the Lucid language was developed with the objective of processing dataflows (precursor of the data

stream) (Wadge and Ashcroft, 1985). The volume of data traffic increased with the expansion of personal computers and the Internet in the 1980s. The data volume peaked in the 21st century with smartphones and IoT devices. Unfortunately, traditional data management systems did not give support in real-time to this data volume (Zhao et al., 2017). In the current century, another challenge is the speed of value changes in data streams since it has to be processed continuously in real-time. Researchers are taking advantage of parallel processing and distributed computing studies and have proposed techniques to deal with many of these challenges. Map-reduce (Condie et al., 2010), continuous queries (Babu and Widom, 2001) and temporal data models (Krämer and Seeger, 2004) nowadays integrate the base of DSP.

The evolution of concepts and techniques associated with DSP leads us to understand a data stream as a sequence of continuously generated data in which the order of arrival cannot be controlled. Data is not necessarily stored prior to processing; its computation must be done as fast as possible. DSP uses sliding windows, and the delay is in milliseconds or seconds (Babcock et al., 2002). This approach is more efficient than traditional methods based on batches because the data storage and grouping may lead to high delays results. To achieve real-time processing, a DSP approach involves building filters with SQL-like operations that process incoming streams and calculate results in a distributed computing system (Cherniack et al., 2003). Filters may involve a sequence of operations-based transformations, such as selections, aggregations, joins, and the operators defined by relational algebra (Cugola and Margara, 2012a). DSP involves detecting patterns in data under very short delays (i.e., low latency).

The software industry has successfully adopted solutions using this approach for applications with high data volume requiring real-time answers. For example, anti-fraud systems, real-time dashboards, clickstream for user behavior analysis (Hanamanthrao and Thejaswini, 2017), analysis of user engagement in marketing campaigns, user sentiment on social networks (Hasan et al., 2018), geofences (virtual fences based on location sensors), monitoring IoT data in smart cities (Tönjes et al., 2014), investment robots in the stock market and equipment failure detection. In these situations, the result of any analysis would be worthless if delivered late. DSP is constantly evolving, and its importance has become remarkable in recent years. There are many challenges around data streams, such as the autonomous management of DSP, fault tolerance (Vianello et al., 2018), resource elasticity (de Assuncao et al., 2018), data stream mining (Krempl et al., 2014), evolving data streams (Bifet et al., 2009), and integration with machine learning algorithms (Krawczyk, 2016).

2.2. State of the art in data stream testing

Although in literature there are no review or exploration studies on testing practices in DSP applications, there are works focused on individual tools and approaches, and some studies on topics related to both testing and quality of data stream software.

Kallas et al. Kallas et al. (2020) proposed DiffStream, a differential testing library for Apache Flink. It is implemented in Java and can be used with common testing frameworks such as JUnit or stand-alone Flink programs. The tool focuses on differential output testing for distributed DSP systems, verifying whether two implementations produce equivalent output streams in response to a given input stream. The tool may serve different purposes, such as checking for bugs in MapReduce programs, supporting the development of hard-to-parallelize applications, and monitoring data stream applications with a minimal performance impact. The effectiveness and usability of the proposed test framework was evaluated by conducting four case studies. DiffStream is available on GitHub[1] as an open-source tool.

Other works propose to apply property-based testing techniques to DSP application testing. This approach focuses on generating test cases based on properties defined by a set of Boolean expressions. Property-based testing checks that a system under test abides by a property. A significant benefit of applying this approach in the DSP context is greater coverage of test cases. Espinosa et al. Espinosa et al. (2019) introduced FlinkCheck, the property-based testing tool for Apache Flink. FlinkCheck provides a bounded temporal logic for generating function inputs and stating properties. FlinkCheck randomly generates a specified number of finite input stream prefixes with the required structure and evaluates the Flink runtime's output streams. In this context, (Riesco and Rodríguez-Hortalá, 2019) proposed a combination of temporal logic and property-based testing to data stream applications testing. An API for testing Spark Streaming programs was made available, and it allows to write of tests in a Scala functional language with the ScalaCheck library.

The work by Zvara et al. (2017) presented a holistic tracing framework designed for batch and streaming systems. It may be used to debug DSP applications by tracing individual input records. Thus, issues caused by outliers become traceable in complex topologies. They built a prototype implementation for an open-source data processing engine, Apache Spark. In the evaluation, the tool helps to identify in real-time a variety of common problems (e.g., bottlenecks, irregular characteristics of incoming data, anomalies propagated unexpectedly through the system). It can also assist developers in improving the system performance by detecting inefficiencies in the compute topology and reducing latency. This approach is valuable to prevent developers from spending countless hours identifying these problems manually, but it only works for problems in real-time running environments.

Restream (Schleier-Smith et al., 2016) is a tool that can replay streams from historical event logs. The tool is designed for accelerated replay and parallel processing, preserving sequential semantics. ReStream was compared against multiple implementations built on Apache Spark and outperformed them all, surpassing the single-threaded implementation and exceeding the performance of Spark. Although ReStream simulates streams precisely in a controlled environment, there is a lack of fault tolerance simulation scenarios. On the contrary, it handles fault tolerance cases by avoiding them during the simulation, such as ignoring duplicated messages and resending messages that may have been lost. Also, in the context of data stream replay, there is Penguin Gu et al. (2018). Still, it was designed for data stream analytical purposes in scenarios where large portions of historical data must be processed with ordered timestamps.

Still, on the replay of historical data, there is the work of Gu et al. (2018), which also proposes a specific tool, Penguin, for replay stream data. However, Penguin was designed more for analytical purposes in scenarios where some portions of the extensive historical data need to be processed and then processed in a specified replay order (e.g., ordered by the timestamps). The interesting thing about this tool is that it comes with specific operators to configure replay characteristics, for example, replay speed, message order, cache usage, message selection filters, merge messages, and map operation to perform a function on each record to generate a new message. These features may be helpful to test environments as they may be able to modify replay characteristics and approximate the simulation of the production environment or to specific test scenarios desired.

[1] https://github.com/fniksic/diffstream

The work by Xu et al. (2013) discusses the validation and detection of abnormality in data streams by models, which can be defined as an analytical model in terms of functions over sensor measurements. They implemented two general strategies to validate streams from industrial equipment: model-and-validate and learn-and-validate. The experiment shows that both approaches apply to real industrial data from IoT devices. However, these approaches may involve scalability challenges. Also, Pizonka et al. (2018) says that the quality of these data plays a pivotal role in many IoT applications, which demands continuous monitoring and validation of streaming data. They propose VortoFlow, a tool that enables users to capture validity requirements regarding value ranges as part of an information model and express it with a Domain-Specific Language. They demonstrated the general feasibility and applicability of VortoFlow using the case of a weatherboard.

Other works like (Feiler, 2010) also explore model-based validation of stream data in the context of IoT and indicate data quality as one of the root causes in IoT data stream. They identified the end-to-end flow of data streams through the system as one of four root causes of IoT system problems. Multiple components are involved in handling the data stream, and they all can affect it by inserting noise and data distortion. They pointed out that the most recurring sources of problems are time sensitivity, data format, out-of-range values limits, concurrency, latency, and missing elements.

Due to the amount of information that flows through a data stream, data test generation is a relevant topic for testing data stream software. In some situations, real historical data is unavailable, and the test generation can be the only way to have test data. It is necessary to generate a considerable number of events with specific structures and values to test the functionalities required by these systems. The proposed tool by Gutiérrez-Madroñal et al. (2018), IoT–TEG, uses high-level languages, such as XML, for describing events patterns and rules for event test generation. It has controlled randomization to generate values within a set between a minimum and maximum value.

Nevertheless, this approach is quite simple as it generates random data within ranges. It would be interesting to generate data in patterns similar to real events; for example, generating latitude and longitude data from a vehicle's Global Positioning System (GPS) in the logic sequence should represent a behavior compatible with the automobile. A more intelligent data generation mechanism may be achieved through statistical models or machine learning techniques over real data.

3. Research questions

The general goal is to synthesize industry knowledge and experience regarding DSP application testing. To achieve this goal, we conducted a GLR to answer the following RQs:

RQ1: What are the challenges to DSP application testing? As mentioned earlier, the DSP context brings factors that hinder DSP application testing, such as message temporality, parallelism, data volume, data variability, messages speed, etc. RQ1 aims to identify and characterize the factors that, from the practitioners' point of view, make it difficult to conduct tests in DSP applications.

RQ2: What are the testing purposes? There can be multiple purposes when promoting software testing. RQ2 seeks to understand what developers want when testing, thereby identifying whether there are test purposes that are especially recurrent in the context of DSP.

RQ3: What are the approaches and specific types of tests performed? Practitioners may have developed good testing practices within their companies as an evolving area where procedures are not consolidated in the industry. With RQ3, we want to identify and record the reported practices.

RQ4: What are the strategies adopted by practitioners to obtain testing data? Obtaining test data is a recurrent issue in software testing, especially for privacy and security concerns. DSP is centered on processing vast volumes of data. Thus testing data is a critical issue. RQ4 seeks to identify which strategies practitioners have adopted to deal with test data.

RQ5: What are the tools, and under what circumstances are they used in DSP application testing? Altogether, there are many software testing tools, some of which can be applied to the context of DSP, while others can be adapted to address specific issues in this context. Also, the community may have developed new tools specifically for DSP application testing. RQ5 aims to formulate a catalogue with the main tools adopted by practitioners and specify how they are used.

4. Research method

We elaborated our GLR protocol based on the guidelines of Garousi et al. (2019). Fig. 1 presents an overview of our GLR process, it has four main phases: (1) review planning, (2) search process, (3) source selection and (4) data extraction and data synthesis. In the following sections, we describe the protocol phases. For replication purposes, the research data used in this study is available online (Vianna et al., 2022).

4.1. Search process

The development of search strings is the first activity of the search process. We started with terms related to the search questions and then refined the strings by performing various tests and tuning in order to obtain more relevant results. In the end, we reached five search strings, each associated with an RQ. Table 1 shows the search strings.

The next activity is the execution of searches. We conducted automatic searches with the Google search engine in the default configuration and specialized target websites, then supplemented with a manual snowballing step. This strategy is well described in the guidelines by Garousi et al. (2019). The adoption of similar strategies has been experienced and discussed by other GLR works such as (Godin et al., 2015), Mahood et al. (2014), and Adams et al. (2016). Many reviews, such

Table 1
Search Strings.

Associated RQ	Search String
RQ1	(challenges OR difficulties) AND (“data stream” OR datastream) AND (test OR testing)
RQ2	(purposes OR objective OR goal) AND (“data stream” OR datastream) AND (test OR testing)
RQ3	(“testing approach” OR “testing strategy” OR “unit test” OR “integration test” OR “system test” OR “acceptance test” OR “functional test” OR “load test” OR “performance test”) AND (“data stream” OR datastream) AND (test OR testing)
RQ4	(“testing data” OR “test data”) AND (“data stream” OR datastream) AND (test OR testing)
RQ5	(“testing framework” OR “testing tool” OR “testing library”) AND (“data stream” OR datastream) AND (test OR testing)

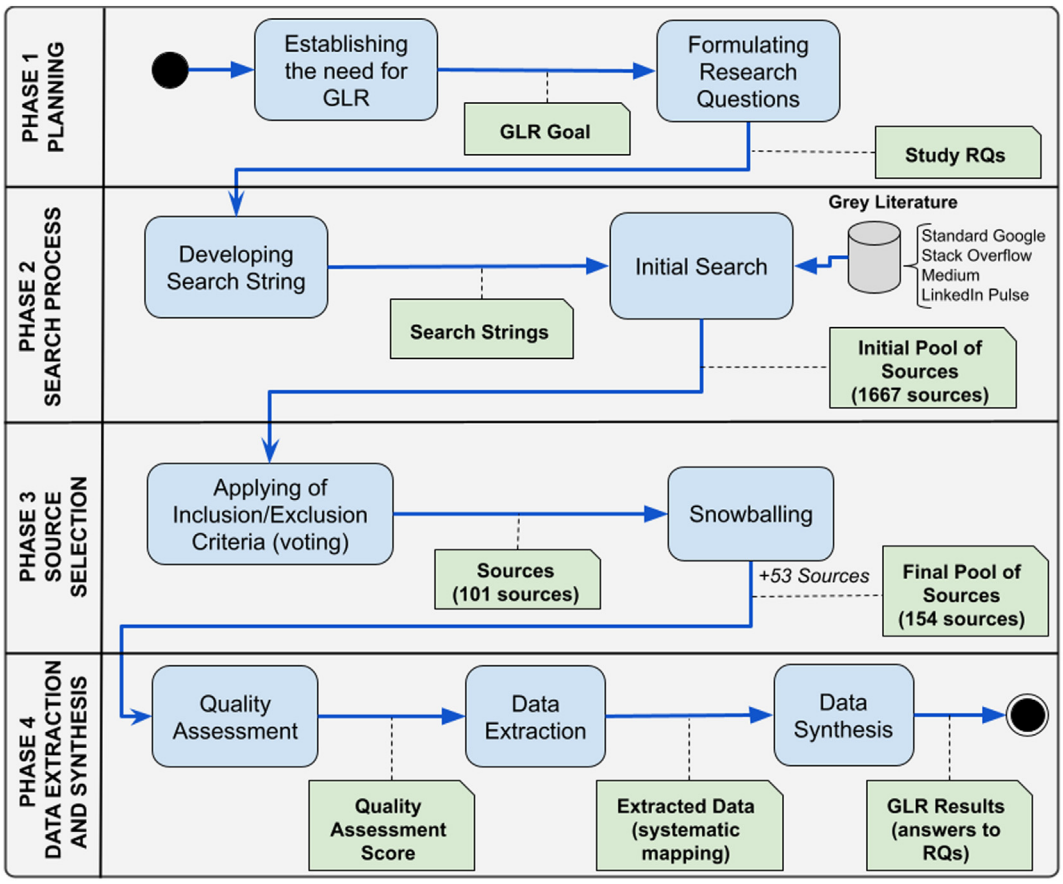

Fig. 1. Grey Literature Review Process.

(Tom et al., 2013), Garousi and Mäntylä (2016), Kulesovs (2015) and Mäntylä and Smolander (2016), mainly use Google Search Engine for grey literature searches. The regular Google service is a tool for general searches across the Internet and returns some relevant key sources. Sometimes, it also delivers irrelevant content because google's search algorithm and the ranking system consider many factors not aligned with our aims (Google, 2021; Fu et al., 2006).

Taking this into account, we conducted searches on *specialized websites*, as recommended by Kamei et al. (2021) to find sources on targeted specialized websites with IT technical content. Therefore, we selected three popular websites among software engineering practitioners: (1) Medium,[2] a blog platform where professionals write short articles reporting experiences and technical issues, (2) Stack Overflow,[3] a popular question and answer website for IT professionals with 100+ million users discussing practical and theoretical programming topics and, (3) LinkedIn Pulse[4] is a blog platform integrated into the professional networking website LinkedIn, where professionals and companies post technical articles related to their projects.

The execution of the searches followed these steps: (1) We performed each of the five strings of searches on regular Google and saved the results; (2) We configured the Google engine to perform searches within the domains of the three specialized websites, Stack Overflow, Medium, and LinkedIn Pulse. We run each of the five strings for each specialized site and aggregate the results with those from the first step, and finally; (3) We remove the duplicate links. For all cases, we stopped saving results due to saturation criteria when a new page of 10 results no longer brought relevant sources. In the end, our search process resulted in 1667 sources.

Finally, we complemented the source pool using the snowballing technique (Wohlin, 2014), exploring reference lists and back-links during other steps. We employed scripts to perform some search tasks programmatically, such as collecting metadata, checking for duplicate sources, and downloading site content for research records.

4.2. Source selection

This phase aimed to select the relevant sources that should proceed to the extraction and analysis phase. The selection begins by applying the inclusion and exclusion criteria detailed in Section 4.2.1; sources that did not meet the criteria were discarded. In the next stage, sources are submitted to a quality assessment according to Section 4.2.2.

4.2.1. Inclusion and exclusion criteria

Our inclusion and exclusion criteria were initially developed based on Garousi's guidelines, and then each criterion was discussed with other authors. The inclusion and exclusion criteria are composed of questions for which the answers can be exclusive, Yes or No. For a source to be included, it must be answered "Yes" for all inclusion criteria and "No" for all exclusion criteria. In this GLR, the exclusion criteria are the denial of the inclusion criteria, so we present Table 2 only with the inclusion criteria.

[2] www.medium.com
[3] www.stackoverflow.com/questions
[4] www.linkedin.com/pulse

Table 2
List of inclusion criteria.

#	Description
IC1	The source is essentially textual (may contain figures), and the text is in English and can be fully accessed.
IC2	The source is not duplicated or republished.
IC3	The source is not an academic paper.
IC4	The originality is not in doubt.
IC5	The source was published between January 2010 and December 2020 (included).
IC6	The source is related to the testing of DSP applications.
IC7	The source addresses some of the issues related to DSP application testing: challenges, tools, processes, approaches, best practices, guidelines, practical examples, tutorials, experience reports, and opinions.

4.2.2. Quality assessment

The sources are evaluated through a quality assessment checklist based on the AACODS framework (Tyndall, 2010), designed to enable evaluation and critical appraisal of such grey literature. It evaluates the source considering the following aspects: Authority, Accuracy, Coverage, Objectivity, Date, and Significance. The checklist's development process had the collaboration of a researcher specialized in GLR, and then it was submitted to a pilot study involving the other two collaborators. The checklist's application is blindly paired, and in cases of disagreement, a third evaluator participates in the decision. Table 3 presents the final checklist.

The checklist is partially composed of Yes/No questions, and other questions accept numerical values greater than zero, such as the number of references, As proposed in Garousi's guidelines, we included the source's impact aspect, which is composed of metrics about interactions with the source, such as the number of views, shares, and likes. We also collected specific metrics for software repositories, such as the number of watches, stars, and forks. Additionally, we also considered the source classification as a quality factor. GLR sources can be classified according to the outlet control, from the 1st tier with the most credibility to the 3rd tier with the least credibility (Adams et al., 2017).

The quality assessment's final output is a numeric score between 0 and 1, summarizing the quality grade for each source. In order to calculate the quality score, each Yes/No response was scaled to values 1 and 0, while responses with numeric values were normalized to values between 0 and 1. However, we realized that due to the significant differences between the obtained values, the simple normalization of values between 0 and 1 would make most of the values zero after rounding. It was necessary to carry out a distribution transformation to smooth out the significant discrepancies between the values. Therefore, we conducted a sensitivity analysis using different distribution functions and analyzed the goodness-of-fit of each distribution to our original data. The exponential distribution best fit because it smoothed high values while retaining the underlying distribution's characteristics. After transforming and normalizing the numerical values, we separately calculated the average score for each aspect of Table 3 and obtained the overall average.

4.3. Data extraction and synthesis

Data extraction is based on systematic mapping, presented in Table 4. It was prepared to structure the content extracted from the selected sources, and the results were then synthesized and reported. The systematic mapping relates the RQs (column 1) with the corresponding attribute/aspect (column 2) and a set of categories (column 3). Column 4 indicates whether the category selection is Multiple (M) or Single (S) for the corresponding attribute/aspect. The development of the predefined set of categories was based on our past experiences with a qualitative study (Vianna et al., 2019). As the initial set of categories can restrict the findings, new categories that emerge from the sources through the conduction of open and axial coding are included in the systematic mapping (Hashimov, 2015).

The extraction process was carried out through peer review with the help of Google Sheets spreadsheets to perform the systematic mapping from Table 4. The researchers carefully analyzed the source, marking the categories related to it. For each marked category, the researcher had to incorporate an annotation with the text copied from the original source representing the category; annotations are traceability links between the extracted data and primary sources. Additionally, the researcher could include comments on the source that would be useful in the synthesis phase. After the extraction, a systematic comparison of the pairs' results was performed, and any disagreements were resolved by discussion.

The synthesis approach was based on qualitative methods with both quantitative and qualitative data analysis techniques. We adopted a qualitative analysis based on open, axial, and selective coding (Saldaña, 2015). The first step was open coding, in which textual data is associated with codes, followed by axial coding, which explored the codes' relationships, and selective coding when codes were grouped into core categories. The pre-defined categories from the systematic mapping guided the coding process. After collecting the data, we were able to structure the categories better and synthesize the evidence found in the coding with a focus on answering the RQs. Finally, in the quantitative analysis, conversely, we evaluated the most frequent codes related to RQs in order to get more findings, such as the most cited test tools, the most recurring problems, the most discussed challenges, and common test purposes.

5. Results

At the beginning of this section, we briefly describe the results obtained in phases 2, 3, and 4 of our GLR process, while the subsections provide detailed results. Phase 2, the search process, produced an initial set of 1667 studies. In phase 3, the source selection and the inclusion and exclusion criteria application resulted in 101 studies. Finally, 53 studies were added from snowballing, completing the final set with 154 studies. Fig. 2 summarizes the sources' selection process, indicating the number of sources discarded through each inclusion criterion. In phase 4, data extraction and synthesis, we extracted data from 154 selected studies and then performed systematic mapping to synthesize the findings corresponding to each RQ. The following subsections summarize the demographics data, quality assessment results and quotations as evidence related to the RQs proposed in Section 3. The list of sources is available in A and is referenced throughout the text as *Source [ID]*.

5.1. Demographic data

From a total of 154 sources selected, 42 were found with standard Google searches, while 59 came from searches on specialized sites, including 25 from Medium, 28 from Stack Overflow, and six from LinkedIn Pulse. Snowballing has added another 53 fonts to the pool. The most common source type is blog posts, with 78 sources, of which 27 are on organizational blogs, 12 on individual blogs, and 39 on technical blog platforms. The following are 33 Q&A websites, 17 Software/Tool Documentation, 17 Tool Repository, 3 Email Discussion List, 4 Slide Presentation, and 2 White Paper. Fig. 3 shows, as a bar chart, the number of sources

Table 3
Quality assessment checklist.

Aspect	Question
Authority of the producer	• Is the organization/author reputable? (in a general way), E.g., the Software Engineering Institute, large IT companies, and well-known organizations or individuals in the field. • Produced/published other work (grey or formal) in the field? (Data Stream)
Accuracy	• Does the source have a clearly stated aim? • How many references/links? Total • Supported by authoritative, documented references or credible sources? (Grey literature or formal references
Coverage	• Does the source clearly addresses a specific RQ?
Objectivity	• Is the statement a subjective opinion? • Are the conclusions free of bias? Is there no vested interest?, E.g., a tool comparison by authors that are working for a particular tool vendor.
Date	• Year • Does the item have a clearly stated date related to content? • How many contemporary references? Check the bibliography for contemporary material. (up to 5 years from source year)
Significance	• Does it add context? (background) • Does it enrich or add something unique to the research?
Impact(Quantify interactions)	General websites: • Views • Likes, claps or upvotes • Comments • Backlinks (Distinct Referring Domains) GitHub • Watch • Star • Fork
Outlet type	• 1st tier Grey Literature (measure = 1): High outlet control/ High credibility: Bachelor/Master/Ph.D. thesis, Software/Tool Documentation, White Paper, Blog Post (in company website) • 2nd tier Grey Literature (measure = 0.5): Moderate outlet control/ Moderate credibility: Blog Post (technical blog platform), Q&A Websites, Slide presentation, Tool Repository, Wiki Articles • 3rd tier Grey Literature (measure = 0): Low outlet control/ Low credibility: Email Discussion List, Blog Post (individual), Tweets

Table 4
Systematic map.

RQ	Attribute/Aspect	Categories	(M)ultiple/ (S)ingle
–	Contribution type	Guidelines, Method (technique), Tool, Tutorial, Best Practice, Experience Reports, Philosophical & Opinion	S
RQ1.1	Issues and challenges	Data Privacy, Performance, Costs of Tests, Message Schema, Contracts Checking, Fault Tolerance, Exactly-Once Semantics, Concept Drift, Test Automation, Lack of Tools, Lack of Specialized People or Know-how, Latency	M
RQ1.2	Testing approaches	Load/Stress Test, Unit Test, Integration Tests, System Tests, Generative Test, Property-based Test, Regression Tests, Data Stream Simulation, End-to-End Tests	M
RQ1.3	Tools	Testing Frameworks, Data Generators, Simulation Tools, Data Replay Tools, Mocking tools, Contract Testing Tool	M
RQ1.4	Test purposes and goals	Correctness of Results, Bug detection, Performance, Fault Tolerance	M
RQ1.5	Test data	Historical Data, Synthetic Data, Sampling Production Data, Custom Data Set	M

selected each year. We observed that we did not have obtained sources from years prior to 2013 and that the number of sources increased each year in subsequent years, reaching 59 in 2020. We observed that the tendency towards an increase in the number of grey sources in recent years might have been amplified by search engine bias and automatic date updating in some sources, such as tool documentation. Another possible reason for this trend is that some web content becomes unavailable over time. For example, companies may change their name or publishing policies, and then technical blogs may be discontinued, leading to fewer sources being found in the first few years.

Among the companies, we highlight the ones with the highest participation, Apache Foundation, Amazon Web Services, and Confluent; they all provide services, tools, and training related to DSP. The contents produced by these companies are the official tools documentation and posts on organizational blogs. Other traditional technology companies, such as Dell, Google, and Microsoft, are also present among the authors but with lesser participation. Companies in diverse business sectors, such as Airbnb, Alibaba, Otto Group, and Deliveroo, employ DSP in their technological solutions and have produced content on organizational blogging platforms.

Individual authors include independent consultants, book authors, members of the open-source community, and employees of companies working in the DSP field. The most frequent author affiliation is Industry, with 146 sources, 3 Academia, and 5 Collaborations.

5.2. Quality assessment results

As described in Section 4.2.2, the quality assessment score is ranged between 0 and 1, representing the sources' average quality according to the aspects from Table 3. Fig. 4 presents the histogram of the number of sources distributed in each score range. The average score of the 154 sources was 0.59, with the highest score of 0.92 and the lowest being 0.28. The average score on standard Google was 0.61, on Medium 0.57, on Stack Overflow 0.51, on LinkedIn Pulse 0.44, and on snowballing 0.63. The higher average score of Google's standard search may be

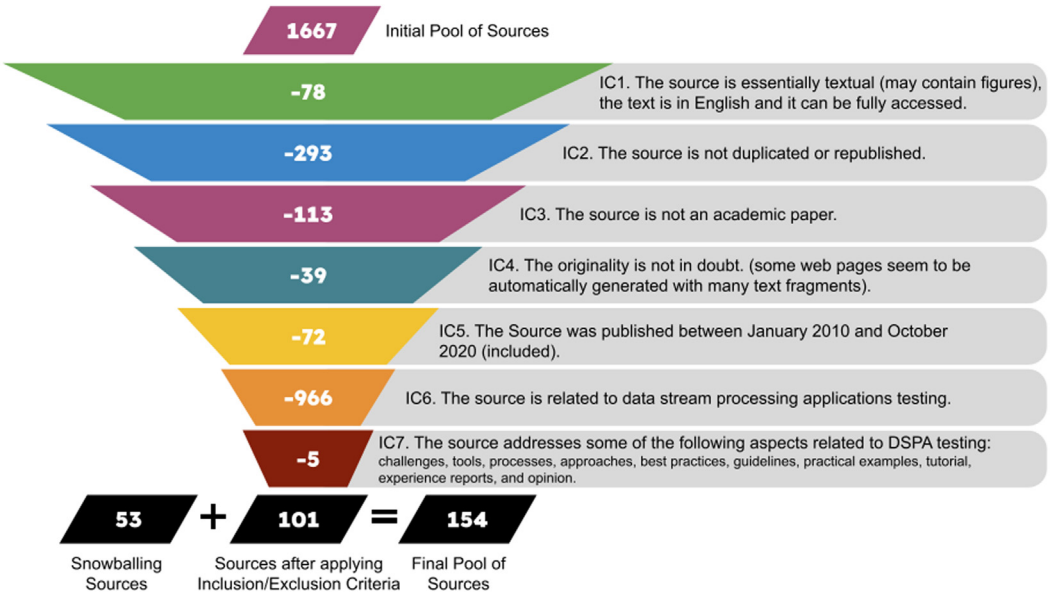

Fig. 2. Result of Source Selection in Numbers.

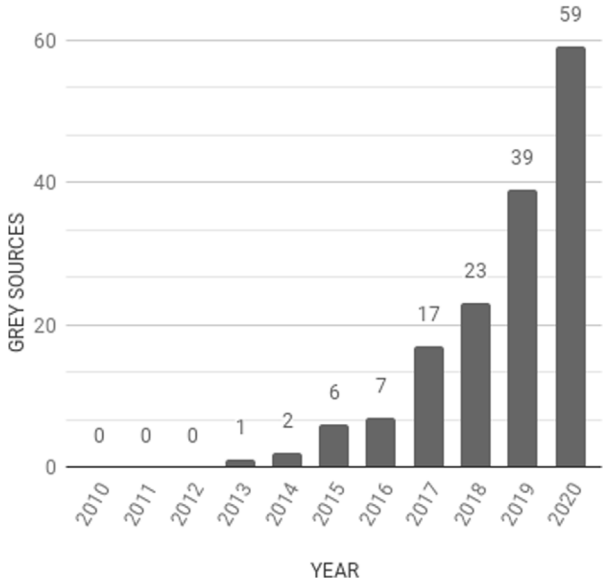

Fig. 3. Selected Sources by Year.

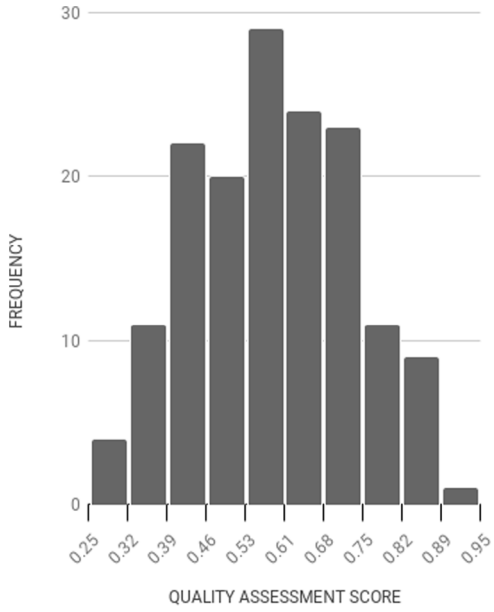

Fig. 4. Quality Assessment Score Histogram.

explained by the fact that this type of search brought up 1st levels Grey Literature sources, such as software/tool documentation, white paper, and blog posts on company websites. The quality assessment result was considered in the analysis, and the source score values were brought together with the evidence presented throughout the text.

5.3. RQ1: What are the challenges to DSP application testing?

In a total of 154 selected studies, we found 92 sources presenting at least one mention of challenges or difficulties in testing DSP applications. Our analysis found many discussions in the community on the subject. At least 36 sources were topics in mailing lists or Q&A websites, on which practitioners raised specific technical questions and reported difficulties in testing DSP applications. In some cases, users reported generic difficulties and challenges addressed to particular aspects. Table 5 summarizes the most recurrent challenges and the number of sources related to each; these challenges are detailed in the subsections below. The following paragraphs provide some quotes where practitioners report general difficulties and challenges when testing DSP applications.

Table 5
Most significant challenges.

Challenge	Source #	%
The complexity of Data Stream Processing Applications	21	13.86
Testing Infrastructure	19	12.5
Test Data	15	9.87
Time issues	9	5.92

We found passages where the practitioners explicitly expressed that it is challenging to test applications involving data streams. In the following, we present the quote [Source 70] illustrating this case.

> *"The pipeline code remains mostly the same, but writing tests for streaming pipelines becomes tricky as you now have to factor in when a test input has arrived."* [Source 70]

The [Source 94] also evidences developers' difficulties in testing DSP applications. This source is a topic on a Q&A website where the user reports problems to test a multi-stream aggregation operation. We highlight three excerpts; first, the user explains the context and expresses the difficulty and time spent on the activity, and then he reveals practitioner's lack of theoretical knowledge about data stream processing is a problem. The last excerpt brings another user's response in which he agrees that certain aspects make the test difficult.

> *"I've set up a simple aggregation averaging values from multiple streams together and I'm trying to test it. I've been burning a lot of time and I can't seem to get the concepts straight in my head".*
>
> *"I feel I'm doing something conceptually wrong here - I just don't know what other approach to take".*
>
> *"Your observation is correct. Caching makes testing hard because it introduces non-determinism".* [Source 94]

Since developers find it hard to conduct tests on DSP applications, we selected the main challenges. Below we present the challenges together with a brief description as well as quotes taken from the sources in which they are reported.

5.3.1. The complexity of DSP applications

DSP applications are naturally complex applications that make developing and running tests complicate as well. As stated in two blog posts in [Source 17] and [Source 19], DSP applications may involve multiple transformation stages, each of which has complex rules and logic and may be difficult to test.

> *"Data pipelines are by the very nature very complex to build and even more difficult to test"* [Source 17]
>
> *"Some of the transformations were quite complex, and the creation of an exhaustive test set is equal to ETL creation"* [Source 19]

Data stream processing (DSP) applications may process data from many different sources, such as sensors, social media feeds, and transactional databases. This data can come in different formats, have different schemas, and arrive at different times, with no deterministic structure.

DSP applications can be complex to test because they are distributed applications that continuously consume messages and process business rules indefinitely. At the same time, applications may process data from many different sources, such as sensors, social media feeds, and transactional databases. This data can come in different formats, have different schemas, and arrive at different times with no deterministic structure. as reported in [Source 21]. Moreover, technical specificities bring other difficulties, such as asynchronicity reported in [Source 80], temporal issues, cluster configuration, partitions, and offsets. In the blog post [Source 127], the practitioner highlights the challenge of planning tests due to DSP scenarios continuously keep changing variables such as records, partitions, offsets, and exception scenarios. This may harm test coverage and generate extra test efforts in infrastructure automation maintenance.

> *"Pipelines can sometimes be hard to test. Especially considering data can come from many different places with no deterministic structure"* [Source 21]
>
> *"One of the reasons integration solutions are not tested thoroughly is the fact that testing asynchronous, message-based middleware solutions is challenging"* [Source 80]
>
> *"The difficult part is, some part of the application logic or a DB procedure keeps producing records to a topic and another part of the application keeps consuming the records and continuously processes them based on certain business rules. The records, partitions, offsets, exception scenarios, etc keep on changing, making it difficult to think in terms of what to test, when to test, and how to test"* [Source 127]

5.3.2. Testing infrastructure complexity

Stream processing infrastructure requires much investment and is notoriously unstable and very difficult to manage. This view is shared by professionals and was exposed in several of the selected sources, the excerpt from [Source 15] illustrates this view.

> *"The streaming infrastructure is notoriously unstable and very difficult to manage. It would require a lot of resource investment, which in absence of a strong business case, would be difficult to explain to the engineering management"* [Source 15]

The [Source 126] excerpt states that tests often work perfectly in the test environment. However, when running the application in the real world, several problems arise due to technical specificities of the production environment, such as a service present in the tests or network latency and throughput factors. The blog post [Source 89] relates that the production network characteristics must also be accurately simulated, and services such as firewalls must be considered.

> *"When everything is running correctly (telemetry data is valid, our code is bug-free, and the Elasticsearch cluster is healthy), this works like a charm. However, in the real world, things don't go smoothly and we can run into the following problems: Even though we check the incoming telemetry data, if there's a blind spot in our validation logic, we might accept invalid data and let it continue through the processing pipeline".* [Source 126]
>
> *"Because test runs in a different network configuration, it did not help us weed out setup problems that only exist in production. For example, if a new version makes a call to an external database, it may work fine in the test environment but fail in production because of firewall settings. Ideally, the test network would be identical to production but in reality, this was not the case and many hours of team members were needlessly wasted on these issues".* [Source 89]

To perform realistic tests is necessary to build a test infrastructure that represents the production environment as much as possible. Simulating the production environment in tests is a challenge, as the infrastructure for DSP solutions can rely on geographically distributed clusters, and may have several third-party services, diverse integrated technologies, and particular configurations. In [Source 88], the misconfiguration was exposed as a point of difficulty, affecting the results of end-to-end and load tests.

> *"Running the load test Spinning up a load test infrastructure is quite a hands-on process and prone to multiple types of misconfiguration. As you can imagine, end-to-end load testing is difficult as results can be affected and tainted by infrastructure and config not being representative of production".* [Source 88]

Configuring and deploying the test infrastructure is not easy, even using infrastructure automation tools, as explained in the blog post [Source 53]. Furthermore, as mentioned in the blog post [Source 26], the infrastructure may be based on serverless services. Such services must be simulated not to incur costs when running the tests.

> *"Unit testing for Kinesis Data Analytics is complicated because it is a managed (serverless) service".* [Source 26]

> *"Lots of moving parts, not so easy to configure and test. Even with automated provisioning implemented with Vagrant, Docker, and Ansible".* [Source 53]

All of this makes it a challenge to create and maintain test infrastructures equivalent to the production environment. Overall, we found 19 sources evidencing the complexity of the infrastructure as a relevant factor when carrying out tests for DSP applications.

5.3.3. Time issues

We identified that time-related issues are a significant challenge when testing DSP applications, as evidenced by numerous references in grey literature sources. We found 45 passages in which practitioners discuss time-related aspects of DSP application testing. The excerpt from a blog post [Source 109] summarizes the importance of time in stream processing. It highlights its relationship to certain features, such as out-of-order events, late events, and aggregation or joining operations.

> *"Time in stream processing is very important it shows out-of-order events, it can be used to determine late events and enables aggregation or joining operations".* [Source 109]

In the production environment, the messages of a data flow are generated with timing characteristics inherent to the real business context. The time interval between messages follows the production environment clock, which becomes a problem during testing since the clock will be different in the testing environment. In testing, the difference in consumption time can affect the output stream's assertion against the expected result. In particular, this point was exposed in two excerpts from a blog [Source 55].

> *"The main problem with testing Streams is their time-based nature. To compare the output stream to an expected another set of data, you will have to do the assertion at the instant where the engine consumes the input".*
>
> *"While requiring more setup and care than testing batch jobs, it is also possible to test Spark Streaming jobs. Controlling the time is the key to having a predictable output to compare results to expected ones".* [Source 55]

A typical time issue problem occurs with temporal windows because of the time interval that delimits these windows' size. This way, windows may differ in size in the test environment if the message intervals are out of control, affecting the test results significantly. The concern with time control in tests of sliding windows was exposed in [Source 59]. The frequency in which events are generated also affects the results of algorithms and functions that evaluate time factors, as quoted in [Source 4].

> *"Streaming has an extra bit of complexity as we need to produce data for ingestion in a timely way. At the same time, Spark's internal clock needs to tick in a controlled way if we want to test timed operations as sliding windows".* [Source 59]

> *"Testing the functionality of a user-defined function, which makes use of managed state or timers is more difficult because it involves testing the interaction between the user code and Flink's runtime".* [Source 4]

In these cases, the test environment must have the ability to control the system clock completely and replay the data stream in which the message intervals are similar to the real scenario. Another issue is the ability to control the clock speeding up the tests' execution, which is opportune; otherwise, the tests would take a long time to run. This issue, among others already discussed, was exposed in [Source 53]. In the discussion in Section 6.1.1, we covered details about accelerating tests by controlling the clock.

> *"Spark Streaming transformations are much more complex to test. Full control over the clock is needed to manually manage batches, slides, and windows. Without a controlled clock, you would end up with complex tests with many Thread.sleeep calls. And the test execution would take ages. "* [Source 53]

5.3.4. Obtaining test data

Altogether, our review found 59 passages associated with the Test Data category, including nine snippets explicitly focusing on the importance of realistic test data and highlighting the challenges of acquiring it. As exposed in [Source 89], adopting historical data for tests are often unavailable for security reasons. The practitioner also mentions that historical data is not a reliable test oracle, even when available.

> *"Copying real data from production to test was both technically impractical and problematic from a security standpoint. More importantly, the results are not easily verifiable because we didn't know to generate expected results from a stream of copied data".* [Source 89]

Next, [Source 88] states that test data must be similar to production data and [Source 89] reinforces that it is challenging to mimic the complexities of real data.

> *"Simulating the real world is difficult. In data-centric systems, your test data must be representative of what your production systems are receiving".* [Source 88]

> *"Test data does not reflect real-life permutations. Try as we might, we could never create test data sets that mimic the complexities of real customer data".* [Source 89]

Manually generating custom data can be massive, time-consuming, and non-trivial, even for those who know the data schema and the business context. For example, in the excerpt from [Source 147], the practitioner reports difficulties in manually generating data, and the excerpt [Source 23] describes the complexity of data associated with a single purchase event in an e-commerce system, reinforcing how laborious manual data generation is. In a discussion on a developer mailing list [Source 69], a practitioner expresses dissatisfaction with building custom data sources. Indeed, manually generating massive amounts of test data is impractical. Those pieces of evidence reinforce the challenges of obtaining test data.

> *"Finding streaming data sources is a hard task, and, especially if you just want to test the basics of the platform, setting them up properly can be quite cumbersome. Creating fake data by hand is also not trivial. Even if you know your data schema, creating a coherent set of rows from scratch is challenging".* [Source 147]

"In the E-commerce world there can be hundreds of data/metadata tagged to a single product, let's take an example if you are ordering a"Smart TV" through any of your favourite E-commerce application then there will be 'n' number of information tagged to it like Brand, Price, Size, Model, Offers, Description, etc.". [Source 23]

"Is the recommended way to create custom data sources and data sinks? I've meanwhile started down this road, but still hoping for a better way". [Source 69]

On the other hand, random test data is relatively easy to produce with a producer application and some random data library. Random data can be helpful for quickly generating load in performance tests where data patterns are irrelevant. However, as warned in [Source 84], these data are unrealistic and could significantly differ from the test results.

"I would also like to note that there is no point in using random data for messages during the testing, since they can differ in size significantly from the current ones, and this difference may affect the test results. Please, take test data seriously". [Source 84]

5.4. RQ2: What are the testing purposes?

From a business perspective, the test's purposes are related to financial impacts on the company and the competitiveness of the business's product; this view is shared in [Source 14]. DSP applications handle high-volume operations per minute, turning a minor bug into a potential catastrophe and, consequently, a race against time to fix failures. Depending on the business sector, a minute of DSP failing may harm a startup as well as large companies; the quotation from [Source 25] supports this statement.

"Business decisions based on false data are costly for all parties involved. It can mean work is repeated, performance is reduced, money is lost or crucially, confidence in your outputs is hit. Confidence in the insights and recommendations you and your team are providing is crucial in gaining the trust of your stakeholders and making a larger impact with your work. You don't want one small mistake to affect future performance, both for you and your team". [Source 14]

"At the end of the day your best efforts can come crashing down (pun intended) all due to a minor little nothing bug in your application logic. The problem here is that your little nothing bug can cost your startup, company, ego – whatever drives you – lots and lots of pain, frustration, even big money or potentially everything". [Source 25]

In the technical context, performing software testing aims to improve software quality. When analyzing the sources, we observed that the test purposes placed by the practitioners corresponded to characteristics of the ISO/IEC 25010 software quality model (Anon, 2011). The ISO/IEC 25010 quality model specifies which quality characteristics are considered when evaluating a software product's properties. In this sense, we decided to use the characteristics of the quality model to categorize the findings related to testing purposes. We identified and mapped the characteristics of the quality model with the practitioners' observations on the purpose of DSP application tests, such as functional suitability, performance efficiency, reliability, and maintainability. Below we present quotes from [Source 14] and [Source 26] in which practitioners expose the purpose of testing focused on the quality assurance of software products.

"Ensuring your deliverables are as high quality as possible means accounting for as many error scenarios as possible. Once the above infrastructure and QA processes have been applied, you'll quickly find a vast improvement in bug detection, error management, deliverable quality, and stakeholder satisfaction". [Source 14]

"Delivering quality software starts with testing. The feature processor needs to calculate features correctly so that both it and other components function as they should. Verifying this correct functioning implies at least some form of unit and integration testing". [Source 26]

In the following subsections, we bring the testing goals unveiled in the review categorized in the characteristics of the ISO/IEC 25010 quality model.

5.4.1. Functional suitability

Delivering correct results was identified as a recurrent concern among practitioners, with the subject being addressed on at least 25 occasions. [Source 100] and [Source 85] reinforce that testing the functional correctness of critical business requirements and validating the proposed solution are important testing goals.

"Conduct Business Test Cases: These tests aim to verify that the data fulfils the mission-critical business requirements. The tests check whether data has been mapped appropriately during the transformation stage according to the business rules..". [Source 100]

"But there is another important vector for developers to consider: validation. Validating that a solution works is just as important as implementing one. It provides assurances that the application is working as designed, can handle unexpected events, and can evolve without breaking existing functionality, etc" [Source 85]

As stated in [Source 4] and [Source 73], it is critical to focus on unit testing the correctness of user-defined functions related to business rules. However, these tests do not include integrating modules and other production environment features, which does not give confidence that functions will not fail in production. DSP is exceptionally complex and involves many factors such as windows mechanisms, timing issues, infrastructure instability, and production environment characteristics that can completely modify the results. In the case of instabilities of brokers in a cluster may cause loss of messages or duplicated message processing. In this sense, treating the functional correctness of DSP applications solely based on unit tests is not appropriate. Furthermore, the notion of correctness in DSP applications does not strictly mean delivering correct results. Some incorrect results are tolerated depending on the context, while others are not. The [Source 140] bring a practitioner's comment illustrating this point.

"Testing User-Defined Functions: Usually, one can assume that Flink produces correct results outside of a user-defined function. Therefore, it is recommended to test those classes that contain the main business logic with unit tests as much as possible". [Source 4]

"Confluent mainly recommends unit tests for Kafka Streams applications. Accordingly, complex business logic should be encapsulated in separate domain classes and unit tested as if it was a traditional application". [Source 73]

"I'm wondering how would we define correctness? For example, with bouncing brokers, message loss might be unavoidable, and with bouncing consumers, duplicate messages might be unavoidable. If we simply require the numbers to be equal, this would restrict the testing applicability". [Source 140]

5.4.2. Performance efficiency

Performance efficiency characteristic was recurrently found in the studies. In total, we found 41 quotes associated with this subject and mapped the three sub-characteristics of the quality model: time behavior, resource utilization, and capacity.

Time-behavior. The DSP is often associated with high-performance requirements as it must process a high volume of messages in short response times. Several factors may affect DSP applications' time behavior, such as storage write speed, algorithmic characteristics, network throughput, network latency, memory, processing power, DSP topology, configurations, etc. The quote from [Source 2] argues that the application must run under stress in order to understand how data volume affects latency, evidencing that test time behavior is among the test purposes in DSP.

> *"Until you've seen a system under stress, you don't understand it Load testing your Kinesis streams can help you get a more intuitive feel for the limitations of Kinesis and their implications to your particular application. Seeing the details of how the latency of the system can be affected by the amount of incoming data, the choice of partition key and number of shards is only possible by generating load on the system..".* [Source 2]

Resource utilization. Efficient stream processing requires robust infrastructure to meet performance requirements, and the cost of contracting many infrastructure services and elastic cloud infrastructure can be high. The rational use of infrastructure resources is crucial because it financially impacts companies and their products' competitiveness. Hence, performance testing aims to assess the application's utilization of hardware resources to promote optimizations. For example, [Source 57] highlights the importance of application performance in the context of cost–benefit analysis.

> *"Understanding performance is also important from a sales perspective. Everyone wants to be able to work faster and quicker while spending less money and using fewer resources. You can't sell a system without understanding its performance characteristics".* [Source 57]

Capacity. Knowing the capabilities of a DSP infrastructure is essential for application operation. Furthermore, understanding the bottlenecks and what resources are needed to scale can be crucial when pushing the limits of the infrastructure in use. [Source 2] highlights precisely this issue; exercising the infrastructure capabilities to know the operating limits is among the test purposes in the context of DSP.

> *"Make the best use of provisioned capacity - each writes operation needs to specify a partition key, which effectively pushes scalability planning onto the users Kinesis. The choice of partition key can have far-reaching implications into how incoming data is split across the shards in a stream and the resulting load on the consumers. Testing ahead of time will allow you to confirm whether your partitioning scheme is production-ready and work out a better one if not".* [Source 2]

5.4.3. Reliability

Stream processing is usually associated with companies with high-reliability requirements, and we often found a recurrence of this subject in grey fonts. In total, we found 32 quotes related to reliability. For example, in [Source 29], investment in testing was explicitly cited as an element in ensuring the reliability of the Alibaba Group's Blink DSP framework. Next, we bring findings related to the fault tolerance recoverability subcategories of the quality model.

> *"Blink is a distributed open-source computing framework for data stream processing and batch processing, supporting the real-time processing of big data for thousands of services in the Alibaba Group. To ensure its reliability, the Blink test team was set up in 2017 to establish a complete Blink test system from scratch and guarantee the quality of Blink".* [Source 29]

Fault tolerance. The complex nature of DSP applications involves many factors that increase the likelihood of system failures. In the presence of hardware failures, network oscillations, unavailability of external services, glitches and bugs, the application must tolerate these failures and continue running. To ensure the fault tolerance mechanisms are operating correctly, tests in this context aim to verify their effectiveness. We have identified in the grey literature that testing fault tolerance capability is a testing purpose in the DSP context. For instance, [Source 29] demonstrates the developers' concern about assessing fault tolerance by simulating abnormal system execution.

> *"For distributed systems that support high concurrency, multi-node, and complex physical environments, it is difficult to avoid physical node anomalies such as disk fullness and network latency. As a highly available distributed system, Blink must ensure the stable operation of the system and the normal output of data under abnormal conditions. By taking on 'the best way to avoid failure is to continually fail' approach, the team simulated all possible exceptions during the Blink task run to thereby verify the stability of Blink".* [Source 29]

Recoverability. Data stream processing (DSP) applications must operate continuously but can experience severe failures that overwhelm the tolerance mechanisms, rendering the application inoperable. In the context of DSP, recoverability includes restoring service, performing pending operations, and correcting incorrect results that may have been delivered. Retrievability is a significant concern for practitioners, as is evident in the grey literature. For example, passages from [Source134] and [Source85] highlight the importance of testing disaster recovery in the context of the DSP.

> *"Testing Recovery is also relevant. There may be many forms of recovery. In terms of SOPs, we want and expect most scenarios to be included in the SOPs and to be trustworthy. Recovery may be for a particular user or organization (people/business centred) and/or technology-centred, e.g., recovering at-risk machine instances in a cluster of servers".* [Source 134]

> *"If you are planning for disaster recovery, simulate an event that results in one datacenter failure, validate Kafka failover and failback, test your runbook and verify consumers can resume reading data where they left off".* [Source 85]

5.4.4. Maintainability

Due to the complexity of the DSP system, maintainability is a recurrent concern of practitioners and is observed in several passages in the grey literature. [Source 26] states that maintainability is a relevant aspect to consider in DSP context; the passage explicitly references testability, suitability, and code reusability.

> *"In general, I also urge you to think about the testability, suitability, and code reusability of a solution before starting on your path".* [Source 26]

Modifiability. Low degrees of modifiability can result in costly bug fixing and system evolution processes. Additionally, it can lead to more bug insertions into previously functioning features, known as regression. For instance, [Source 12] highlights the detrimental effects of regression bugs on releases, while [Source15] emphasizes the importance of upgrading without impacting business operations.

> *"Writing unit tests can ensure our code is working as expected, is much less "expensive" than if a bug or regression makes it all the way to a release".* [Source 12]

> *"Every new piece of software requires backups, disaster recovery, ability to upgrade without impacting business, dev-ops support, ability to talk to other pieces of software seamlessly among other things".* [Source 15]

5.5. RQ3: What are the approaches and specific types of tests performed?

We identified 155 quotes from 82 grey sources reporting testing approaches for the DSP context. In some sources, testing approaches are discussed from the perspective of unit, integration, or system testing, and these results are presented in Section 5.5.1. We also find reports of specific approaches, which we bring the most relevant in the following subsections. We adopted the SEWBOK Version 3 terminology for software testing presented in chapter 4 of the guide (Bourque and Fairley, 2014).

5.5.1. Approaches by target of test

Unit Testing. Unit tests are typically automated tests where software components are tested in isolation to validate if each module presents the correct results. Unit testing is the foundation for proceeding with integration testing, and a greater unit testing coverage is essential for regression testing. As presented in [Source 60], DSP application unit tests have some particularities according to the type of operation: stateless operators, stateful operators, and timed process operators.

> *"The strategy of writing unit tests differs for various operators. You can break down the strategy into the following three buckets: Stateless Operators, Stateful Operators and Timed Process Operators".* [Source 60]

Stateless operator unit testing has no extra difficulties and follows traditional practices for writing a test case. However, testing stateful operators requires careful handling of component states, verifying that the state values have been updated correctly and that cache data is properly purged when necessary. Finally, testing timed process operators introduce additional challenges as it requires controlling the application clock and providing events timestamps. Tests of this type of operator can also involve manipulating time variables in order to expose the module to different test scenarios.

Stream processing platforms provide utility classes to support unit testing. Additionally, some platforms, such as Apache Flink and Kafka Streams, have operator state-checking and clock-controlling features, as reported in [Source 120]. In addition, we identified other tools and libraries adopted for unit testing: Flink Spector, Fluent Kafka Streams Tests, Mockito, Mockedstreams, Kafka for Junit, ScalaTest, and Scalacheck.

> *"With the ability to test Kafka Streams logic using the TopologyTestDriver and without utilizing actual producers and consumers, testing becomes much quicker, and it becomes possible to simulate different timing scenarios".* [Source 120]

Integration Testing. In integration tests, the target is a set of logically integrated components. The objective is to verify if the interfaces among modules work fine and expose problems arising from their interaction. Integration testing brings up the challenges related to time issues identified by RQ1 in Section 5.3.3. The passage [Source 22] highlights issues concerning time issues in integration testing.

> *"Integration test is a way of how to test services in isolation but with required dependencies. Embedded Kafka cluster combines broker, zookeeper, and schema-registry in one. To test asynchronous systems, I choose awaitility library. It is a useful tool for handling timeouts and asynchronous asserts".* [Source 22]

Timing issues come into play when modules interact, and problems related to timeouts and asynchronous operations may break the application. At this point, the DSP infrastructure must provide application clock handling and event timestamp configuration capability. Another valuable feature is the checkpointing mechanism, which stores consistent snapshots of all the states in timers and stateful operators, including connectors, windows, and any user-defined state. The review also referenced the Awaitility Library, a DSL that allows the expression of expectations of an asynchronous system.

In addition to time issues, integration testing requires the extra effort of building test infrastructure, as third-party dependencies must be satisfied and DSP mechanisms provided. The solutions adopted by practitioners are based on local infrastructure on a reduced scale and various tools and libraries to mock services, APIs, and infrastructures. Therefore, the integration tests involve the development of infrastructure code and dependency injection to abstract read sources and write targets. Our GLR has identified several tools for these purposes, the main ones being: Embedded Kafka Cluster, WireMock, TestContainers, LocalStack, KineasLite, Kcat, Ducktape, and Jackdaw. Section 5.7 introduces these tools and their uses.

System testing. System test targets the entire software product with all integrated modules, dependencies, and services, as the goal is to verify the functional and non-functional behaviors. The system test is related to the challenge *The complexity of DSP Applications* described in Section 5.3.1 because, at this stage, the typical characteristics of distributed applications and DSP emerge. We list some of the most relevant features, such as non-determinism, process concurrency, process parallelism, message asynchrony, latency, glitches, node crashes, random hardware failures, out-of-order messages, lost messages, and duplicate messages. In our GLR, we have identified the chaos engineering-based testing approach to deal with some of these issues; Section 5.5.5 describes this approach.

DSP application testing is also related to the *Testing Infrastructure* challenge presented in Section 5.3.2. Unlike integration tests, the system tests must run in the infrastructure as accurately as possible in the production environment. In this case, mocks and service simulations are avoided whenever possible. This point can be especially challenging in SaaS-based infrastructures because of the cost involved and the difficulties in configuring and customizing the test environment in the cloud.

With this in mind, infrastructure automation is a must-have approach at this testing stage, as you need to run many services and perform many specific configurations. In our GLR, we find tool-based approaches to assist in infrastructure automation. We list the tools identified: Terraform, Jenkins, Confluent CLI, Ansible, Docker, Vagrant, and AWS CloudFormation. In [Source 85], there is a passage where the Terraform tool is mentioned to build cloud test infrastructure.

"If you want to run those services in the cloud, check out the tools for Confluent Cloud. These Terraform scripts provision Confluent Platform services to connect automatically to your Confluent Cloud cluster. They handle creating the underlying infrastructure details, like VPC, subnets, firewalls, storage, compute and load balancers..". [Source 85]

5.5.2. Performance test

We have selected 75 quotes from 27 grey sources related to performance tests, and we bring quote [Source 13] as an example that highlights the relevance of this type of test for DSP applications. Performance tests push the application to specific workloads to evaluate performance metrics. The goals are to identify performance bottlenecks and find the maximum load where the application is stable.

"Never go to Production without the load testing. After you go through all the test activities during the development sprint, Unit Testing, and Integration Testing, now is the time for Load Testing. Load Testing is a kind of Performance Testing that determines a system's performance under real-life load conditions". [Source 13]

In the context of DSP, several variables affect performance, such as the number of nodes in the cluster, hardware resources (processing and memory), network (bandwidth, latency, throughput), number of topics, number of producers, number of consumers, message size, timeout values, cache sizes, replication factor, etc. One approach is to execute performance testing with different settings, as exposed by quote [Source 141].

"What performance characteristics should I expect in my particular configuration? What should I consider a severe latency spike? What's the impact of dropping a broker? The reason I'm interested in load testing is to form a collection of expectations, so these questions are better understood in the reality of production infrastructure". [Source 141]

Thus, performance testing is a tool that also helps tune the system for specific scenarios. It helps to estimate the adequate hardware resources and adjust the various DSP tools settings. From a business perspective, [Source 13] emphasizes that performance testing also brings cost savings by facilitating developers to optimize the use of computational resources.

"However, they will be more interested to hear the user experiences and cost savings, etc, which as I mention the business language. So by doing the Load/ Performance Testing, we can: 1. Act proactively to identify/mitigate potential customer satisfaction risks by devoting X amount of technical, financial, and human resources. 2. Reduce the cost and increase the potential sells, etc". [Source 13]

Stream processing platforms provide several capabilities for generating loads for performance tests. However, we identified that practitioners also use other tools and libraries with functionalities to meet specific needs and to configure more accurate traffic patterns. Quotes [Source 134] and [Source 141] illustrate precisely these aspects of data load generation tools. Our GLR has identified several load-testing tools: JMeter, Pepper-Box, ZeroCode Samurai, Kafkameter, Sangrenel, Artillery, Gatling, and Ducktape. We also identified the Grafana tool adopted to visualize metrics in performance tests.

"We knew we needed ways to generate traffic patterns that ranged from simple text to ones that closely resembled the expected production traffic profiles. Over the period of the project, we discovered at least five candidates for generating load: Kafkameter, Pepper-box, Kafka Tools, Sangrenel, and Ducktape" [Source 134]

"To help get me there, I put together a little load testing tool. It's appropriately named after a gritty pirate weapon: Sangrenel. It works by generating random messages of configured byte sizes to bombard Kafka clusters with, and periodically spits out throughput and top 10% worst latency averages (it's a synchronous publisher and meters the latency from message send time to receiving an ack from a broker)". [Source 141]

5.5.3. Regression test

Regression tests focus on verifying that each software modification does not introduce problems in other system components. Regression tests are associated with the maintainability objectives from Section 5.4.4, as they bring more security when making changes. The lecture on the [Source 78] Source slides "Test strategies for data processing pipelines" by Albertsson, L. cites regression testing as a good practice. The subject is also present in other selected studies, such as quote [Source 89].

"One thing we need to make sure of is we don't introduce new bugs or regressions as we change our code". [Source 89]

Regression testing is mainly associated with functional and performance testing in the DSP context. The functional aspect consists of checking whether the changes affect the results of other functions, which can be achieved through good unit test coverage, as exposed in [Source 12]. The performance aspect involves verifying whether the changes affected the system's performance. The primary performance regression testing strategy involves collecting performance metrics and evaluating the results against previous versions. The excerpt from [Source 29] provides evidence of the importance of this issue and a brief description of the process.

"Runtime performance testing. This is mainly to ensure that the performance at the operational level has no regression. It mainly includes end-to-end performance testing and module performance testing. The end-to-end performance test firstly figures out the test scenario and pays attention to the job processing time of each scenario under a specified data volume. The module performance test mainly includes network layer performance testing, dispatching layer performance testing, fail-over performance testing, and so on. It pays more attention to the job processing time in a specific scenario". [Source 29]

Regression testing often requires robust automation of the development infrastructure, including high unit test coverage and Continuous Integration (CI) mechanisms, as noted in [Source 12]. Because manual steps make it impossible to run the tests repeatedly. In the case of performance tests, a trustworthy test environment is preferable to the production environment, which increases the cost of test infrastructure.

"Unit test coverage is an interesting subject; I use to be crazy enough to trace the 100% coverage, however, I learn my lesson, the most important about unit tests is: How confident you are about your code and make sure the CI/CD automation catch as much as possible the coding level regression bug". [Source 12]

5.5.4. Property-based test

Property-based testing is a black-box testing technique in which a software component property is validated against inputs from random data generators. Properties are rules that describe the component's behavior at a high level, thus establishing a formal validation of results through testing. In traditional test suites, developers must manually specify multiple test cases to cover the main corner cases. This approach is totally manual labor, limiting testing cases and leaving corner cases out.

On the other hand, property-based testing tools automate the construction of relevant test cases. The process starts with automatically generating various inputs to find fault cases and then trying to isolate what exactly it is that made it break by a mechanism called shrinking. Shrinking involves refining the generated inputs to a minimal set for failure case reproduction. This approach allows the fast generation of thousands of test cases relevant to different components.

In our GLR, we found evidence of the use of property-based testing in the industrial context and comments from practitioners reinforcing the technique's benefits for DSP application testing. Below, the quote [Source 149] illustrates this issue. Finally, we also find tools and libraries, like Scalacheck, StreamData, and Ecto Stream Factory, that have been used to perform this type of test; such tools are listed in Section 5.7.

> *"Using property-based testing has some advantages. First of all, it lets you test the properties of your code over many more values than you otherwise would with example-based testing. While it's true that random data generation can't cover all the possible values that a piece of code can deal with, the confidence in your codebase can still increase over time because the property-based tests will likely generate different values on each run. Example-based testing means your test data will not change over time".* [Source 149]

5.5.5. Chaos test

Chaos engineering seeks to address the following question: *"How much confidence can we have in the complex systems that we put into production?"*. Basiri et al. (2016) address chaos engineering in modern distributed systems and define the approach as a controlled experiment that measures the system's capacity to run under realistic conditions.

Large-scale DSP fits into the proposed context for chaos engineering. These systems work on distributed hardware and software infrastructures so complex with infinite fault variables. Therefore, the reliability of infrastructures cannot be guaranteed at all times. So the idea is to accept that failures will happen, and chaos engineering focuses on exercising the system to work in adverse conditions. This view of Chaos Engineering is shared in quote [Source 114].

> *"Failures are inevitable. Just as we need to test our application to find bugs in our business logic before they affect our users. We need to test our application against infrastructure failures. And we need to do it before they happen in production and cause irreparable damage. The discipline of Chaos Engineering shows us how to use controlled experiments to uncover these weaknesses".* [Source 114]

Chaos Engineering seeks to meet the reliability objective exposed in Section 5.4.3, including fault tolerance and recoverability. Quote [Source 29] exemplifies developers' concern about recoverability, in which testing involves verifying that the system was able to restore a checkpoint when exposed to an abnormal scenario. Although the practitioner uses the term stability test, the description of the testing process fits the concept of chaos engineering as it subjects the application to an abnormal scene in order to test recoverability.

> *"The stability test is set to an iterative loop. Each round of iterations involves releasing the abnormal scene, submitting the task, waiting for the job recovery, and verifying. The verification is mainly to check whether the checkpoint, container and slot resources meet the expectations".* [Source 29]

Networking is one of the developers' concerns regarding DSP fault tolerance, as network instabilities such as latency, throughput, and connection loss can affect the timing issues of DSP. By the way, timing issues were identified as one of the testing challenges in section 5.1.3. An example is the timing window mechanisms that are especially sensitive to latency variations. This issue is exposed in quote [Source 8], while quote [Source 114] brings a comment about latency injection in tests.

> *"When processing a stream of events in real-time, you may sometimes need to group events together and apply some computation on a window of those events. Suppose we have a stream of events, where each event is a pair consisting of the event number and the timestamp when the event was sent to our system, and that we can tolerate events that are out-of-order but only if they are no more than twenty seconds late".* [Source 8]

> *"So that's how you can inject error and latency using the Thundra SDK. It's useful for simulating a whole host of possible failures and seeing how they affect your system".* [Source 114]

In hardware infrastructure, fault tolerance testing includes taking down machines or cluster nodes and throwing memory, CPU, and storage exceptions. Quote [Source 29] exposes this situation. On the other hand, the quote [Source 114] introduces the possibility of injecting faults into application functions, and this mechanism can be helpful in simulating the impact of programming bugs. Finally, DSP applications also need to be prepared to deal with failures and instabilities of third-party services, which can even break your application. This concern is presented in [Source 14].

> *"The first [abnormal scenes] is related to the running environment, including machine restart, network exception, disk exception, CPU exception, and so on".* [Source 29]

> *"Error Injection: The happy path appears to be working, great! Now let's see if we can uncover some weaknesses by injecting failures into the functions".* [Source 114]

> *"In reality, third parties may not have the same resources, infrastructure, practices or approach that you may be used to. This difference can lead to errors in ingestion, processing or presentation which may be slow to fix if you need to ask a third party to reformat or reproduce the data for any reason".* [Source 14]

5.5.6. Contract/Schema testing

The schema works as a contract to specify the message format in DSP systems, and validation consists of checking the compatibility of messages in accordance with the scheme. The motivation for schema validation is to prevent the application from crashing because of non-standard format messages. Message contract is a typical concern in data-centered systems, especially in DSP, where there is usually more interaction between multiple systems, consuming and producing data in the ecosystem. Message incompatibilities can be caused by programming errors, failure to communicate between development teams or schema updates. The larger the system, in terms of subsystems interacting and third-party services involved, the greater the chance of schema incompatibility failures. As commented on in [Source 154], systems aimed at managing schemas are often not used, but if they were adopted, it would help deal with schema updates.

> *"We've found most people who have implemented a large-scale streaming platform without schemas controlling the correctness of data have to lead to serious instability at scale. These compatibility breakages are often particularly painful when used with a system like Kafka because producers of events may not even know of all the consumers, so manually testing compatibility can quickly become impossible".* [Source 154]

The solution to this problem involves both software testing and fault tolerance approaches. In tests, message validation can occur firstly by exercising the different message patterns in unit tests, with a focus on message producer and consumer components. Later, at the integration level, the interaction between modules can also expose schema incompatibilities.

However, tests only deal with applications you have control over, not covering incompatibilities caused by updates in third-party systems. Therefore fault tolerance approaches are necessary in these cases, where the idea is to prevent, at runtime, the application from crashing when faced with a non-standard message. This issue was addressed by Tom Seddon source [Source 153], which presents an approach based on interoperability between schemas with different versions to support schema evolutions.

> *"These guarantees mean consumer applications can have expectations of the format of the data and be less vulnerable to breaking due to corrupt messages. Another important aspect of resilience is being able to update the data model without breaking clients that are depending on a different schema version, which means ensuring we have backward and forward compatibility".* [Source 153]

Finally, as mentioned in [Source 153], adopting Domain Specific Languages (DSL) as Avro to register the schema is the first step to control schema updates. It is now possible to version the registry and control schema updates. Furthermore, tools like Confluent Schema Registry centrally manage schema registration, providing compatibility models between different schema versions for all involved systems. Quote [Source 85] sums up the essence of this approach.

> *"One of the other appealing aspects of Avro is that it manages schema evolution and backward and forwards compatibility for you, by keeping track of writers and readers schema".* [Source 153]

> *"After your applications are running in production, schemas may evolve but still need to be compatible with all applications that rely on both old and new versions of a schema. Confluent Schema Registry allows for schema evolution and provides compatibility checks to ensure that the contract between producers and consumers is not broken. This allows producers and consumers to update independently as well as evolve their schemas independently, with assurances that they can read new and legacy data".* [Source 85]

5.6. RQ4: What are the strategies adopted by practitioners to obtain testing data?

Practitioners consider it essential to take test data seriously, and having a good test dataset is crucial to performing realistic tests on DSP applications. Test data is necessary at all levels of testing, and in each situation, there are different requirements. For example, in unit testing, the data should enable finding corner cases, while in performance testing, a huge volume of test data is needed to stress the application. The following subsections present strategies for obtaining test datasets identified by this GLR.

5.6.1. Historical data

Having access to real historical data helps to reproduce bugs, as the data patterns in which bugs manifest are often complex. Although historical data seems to be a good option, it is not always possible to use them for privacy and security reasons. However, this is not a problem in some contexts, as data anonymization is sufficient for data security. In part, some professionals advocate the use of historical data for testing, and the [Source 51] illustrates this point of view.

> *"Testing such systems should be with real-world data, running long enough to surface such issues, and with a very focused eye on the subtle changes in the system behaviour during tests. But even doing this won't assure everything is covered".* [Source 51]

5.6.2. Production data mirroring

In this strategy, replicas of production data streams are redirected to a server with the application under test. This approach also has the data privacy issue, but we identified some proposed solutions to maintain data privacy. As explained in [Source 89], the test version runs in "shadow mode" where the computed results are not published and are just automatically compared with the results of the production version.

> *"If real data will not come to the test, the test will go to the real data. Once a version passes unit and integration tests on the test environments, it is deployed to the production network and subjected to real, production datasets. However, the new version is executed in Shadow mode. In Shadow mode, a task does not propagate results down the pipeline, instead, they are simply stored in a designated location. A compare task then picks those results, along with the result-set of the live version, and compares them for correctness".* [Source 89]

As noted in [Source 89], the result comparison strategy should adopt specific methods that involve statistics to compare results, as values can vary widely due to other factors, even for accurately replicated environments.

> *"Obviously, the compare task cannot validate results simply by making sure they are the same. The new version may very well produce different results. So as part of the work on the new version some specific compare-validation code is developed that takes into account the nature of the change created".* [Source 89]

The advantage of mirroring production data is to submit the application under test will to a real-world stream. This technique is also valuable for checking regressions of a new application version. On the other hand, the tests are limited to cases where it is possible to compare results. For example, new features don't have comparison parameters because they did not exist in the previous version.

5.6.3. Semi-synthetic data

When there is a limited amount of real data available, one strategy is to expand the data by generating test data based on real data. This solution can be achieved by various techniques, from the simple mixing of real data with synthetic data to more advanced techniques based on statistical models. The passage in [Source 25] exemplifies this strategy adoption, where developers modeled realistic test data for an actual live traffic telephony application.

> *"The pattern for loading a scenario comes from trial by fire while trying to figure out the easiest way to generate repeatable tests based off of real fake data. My team at Twilio (Voice Insights) came up with some neat internal generators of fake telephony data – modeled off of real live traffic. Through the generator, we can create true-to-life scenarios instead of just testing randomly generated data or some bogus nonsense! This sometimes means we generate thousands of scenarios that will be loaded and tested".* [Source 25]

5.6.4. Synthetic data

This strategy involves generating synthetic data with complex structures described by message schema or templates defined by DSL (Domain Specific Language). The [Source 9] and [Source 118] excerpts evidence this strategy adoption by the Kafka Connect Datagen tool. When the schema is not available, tools like the Alooma Mapper infer the schema from the data, as shown in [Source 142].

> *"To define the set of "rules" for the mock data, kafka-connect-datagen uses Avro Random Generator. The configuration parameters quickstart or schema.filename specifies the Avro schema, or the set of "rules", which declares a list of primitives or more complex data types, length of data, and other properties about the generated mock data".* [Source 9]

> *"However, you can also define your own schema specifications if you want to customize the fields and their values to be more domain-specific or to match what your application is expecting. Under the hood, this datagen connector uses Avro Random Generator, so the only constraint in writing your own schema specification is that it is compatible with Avro Random Generator".* [Source 118]

> *"A Mapper component automatically maps fields from each data source to the target data store. Schemas are inferred automatically and simple incongruencies..".* [Source 142]

Some data generators provide advanced functionality to configure data's timing and distribution aspects. Kinesis Data Generator is an example of a tool with many specialized features. In quote [Source 125], the Lock to Real-Time functionality is described to adjust the temporal distribution of the data. While citation [Source 125] describes a mechanism for customizing data randomization patterns. This strategy involves generating more complex structure data described by message schema or templates defined by DSL (Domain Specific Language).

> *"If 'Lock to Real Time' is deselected, the KDG will prompt you for start and end date-times and will generate data for each second between those start and end times in 'ticks'. The number of seconds of data that is generated in each 'tick' is the value in 'Seconds of data per tick' and the data generation thread will yield 'Wait time between ticks' (milliseconds) between each tick".* [Source 125]

> *"Sometimes you don't want randomness to be completely random. You might want to choose elements from an array, but you want the randomness to be weighted such that over time, each element is chosen a certain number of times, relative to other elements in the array".* [Source 125]

This GRL identified tools adopted by practitioners to generate test data. The most relevant ones include Avro Random Generator (Confluent Inc., 2021a), Kinesis Data Generator (Amazon Inc., 2021), Kafka Datagen Connector (Confluent Inc., 2021d), Ksql-datagen (Confluent Inc., 2017), Mockaroo (Mockaroo LLC, 2021), Ecto Stream Factory (Barchenkov, 2019), and StreamData (Leopardi and Valim, 2021). On some occasions, practitioners report supplementing the tools with in-house scripts to obtain more representative test data.

5.7. RQ5: What are the tools and under what circumstances are they used in the context of DSP application testing?

To answer RQ5, we present a list of tools adopted for DSP application testing. The tools were extracted from the GLR sources. For each cited tool, we verified the existence and compatibility with the purpose it was mentioned. Besides, we analyzed the intended use of each tool and related it to the testing approaches from Section 5.5.

Table 7 lists the 50 tools, and then the columns in the table are described. The *Tool Name* column displays the tool's name. The authorship of the tool is in the *Author* column, which can be an individual, organization, or community. When there is an original author, but the project is now maintained and developed by an organization or community, we put the original author's name followed by the organization or community. The *License* column displays the type of software license. The *Main Testing Approaches Support* column associates the tool with testing approaches from Section 5.5. The column *Focused on data stream context?* discriminates whether the tool was developed for the DSP context or is a tool proposed for another context. Finally, *Document Citations* quantifies the grey sources that cite the tool.

6. Discussion

In this section, we first revisit our main findings, discussing some and then the implications for research and practice.

6.1. Summary of research findings

Throughout this subsection, we go through each RQ, discussing the related findings.

6.1.1. RQ1. What are the challenges to DSP application testing?

Our first RQ focused on identifying professionals' challenges when testing the DSP applications. The RQ1 findings also reinforced the claim that testing DSP applications are challenging overall, as 92 of 154 grey sources address testing challenges in this context. Table 6 summarizes the four main challenges that emerged from the grey literature, and the remainder of the section addresses each of them in detail.

The complexity of DSP Applications. The complexity factor of DSP is already well known, and it has been reported in the formal literature (Namiot, 2015; Cugola and Margara, 2012b). The grey literature reports the factors that practitioners consider most challenging, such as non-determinism, asynchronicity, complex transformations, a lack of theoretical knowledge on stream processing, and scenarios with continuously changing variables.

The DSP non-determinism makes test development challenging since multiple test runs can produce different results, making it hard to reproduce test scenarios and validate the correctness. In the formal literature, we find research testing strategies that may be adopted for DSP testing. For example, the study by Boroday et al. (2007) proposes a test generation approach based on model verification for non-deterministic systems, while Leesatapornwongsa et al. (2016) characterizes typical bugs related to non-determinism in distributed applications. Diaz et al. (2021) propose test criteria to guide the selection of test cases in message-passing parallel programs; it helps to reveal nondeterminism-related defects, particularly in loops. Chen et al. (2015) provide a survey on deterministic replay focused on testing and debugging systems by controlling race conditions and non-deterministic factors.

Asynchronous stream processing can introduce race conditions, where the data processing order affects results, making their identification and resolution challenging. Yu et al. (2017)

Table 6
Key Points of Challenges to DSP application testing.

Challenges	Key points
The complexity of DSP Applications	• Non-determinism, asynchrony, complex transformations, lack of theoretical knowledge and changing variables make the test challenging. • Testing strategies must test each transformation individually and then together, which can be time-consuming and requires significant effort. • Qualification of the test team and theoretical knowledge regarding DSP is required.
Test Infrastructure	• Building the test infrastructure as close as possible to the production environment is important, but it can be expensive. • Automation tools and mocking services help reduce costs and simplify the construction of test infrastructure.
Obtaining test data	• Good test data is essential for realistic testing. • Real data is difficult to obtain due to security and data privacy regulations. • Historical data provides an initial data model, but generating and validating outputs is necessary to build a reliable oracle.
Time issues	• Timing affects aggregation, joining mechanisms, and the order in which events are generated, processed and ingested. • Special handling of timing issues is needed when setting up tests, and tool support is essential for controlling the application's clock and watermarks. • Choosing the processing time interval depends on the application's timing characteristics and requires a balance between precision and computational costs. • Controlling the clock in the test environment allows speed-up test execution.

proposed a framework to support testing for process-level race conditions, which considers asynchronous variables and provides some insights to address the problem in the DSP context. Asynchronicity also makes it challenging to test functions with timing dependencies, where the output of one processing step depends on the output of a previous step.

DSP applications can involve a sequence of multiple transformations, which can have complex rules and logic, where the output of one transformation stage depends on the output of a previous stage (Dávid et al., 2018). Consequently, building a test oracle also becomes a complex task. Testing strategies must test each transformation individually and then together, which can be time-consuming and requires significant effort.

A practitioner also reported that a lack of theoretical knowledge about data stream processing made testing challenging. DSP testing requires qualified professionals with specific knowledge of the context; otherwise, they may be unable to interpret test results or elaborate efficient test strategies. This report refers to our statements regarding the complexity of DSP applications presented in the introduction and reiterated throughout the paper. Furthermore, the comment reinforces the need for research work in the area to provide contributions to the industry and practitioners. Bath's study (Bath, 2020) addresses the different specialist competencies that professional testers of the next generation may need. Testing big data applications is discussed as a challenging context for testers due to characteristics such as performance, scalability, functional correctness, data currency, and testing of backup and recovery capabilities. This study supports our grey literature findings regarding the challenges of testing DSP applications.

Finally, it was also reported that the context of DSP applications is subject to continuously changing variables, such as data schema, configurations, infrastructure, partitions, offsets, and exception scenarios. These variables can make maintaining test coverage challenging, as existing test cases can become incompatible with changes, and test data may require updates to its structure. Constant changes also make it challenging to maintain automated tests and configure the test environment to accurately reflect the production environment. Regarding test case obsolescence in evolving software, Imtiaz et al. (2019) conducted a systematic literature review that reports insights into how to prevent and repair test breakages.

Test infrastructure. From grey sources, we extracted reports on the importance of building the test infrastructure as similar as possible to the production environment. Generally, the DSP applications adopt a large-scale cloud infrastructure based on microservices architecture, where the network interconnects the various distributed services. The first problem is the cost of allocating such infrastructure in the test environment, which typically includes allocating hardware resources, network services, tool licenses, and the consumption of third-party services. The second problem is the qualification of the testing team. In addition to knowledge of infrastructure technologies, they must also be skilled in testing techniques and frameworks. The mixed-methods study conducted by Waseem et al. (2021) reported desirable skills for testing microservices systems, such as writing good unit test cases, analytical and logical thinking, and knowledge of test automation tools. In short, the infrastructure's complexity entails allocating costs and demands a qualified team.

The grey literature indicates infrastructure automation tools to reduce the time and resources spent on building test infrastructure. The tools help set up and configure services automatically, which were cited at the end of Section 5.3.2 and listed in Table 7. In the formal literature, Rafi et al. (2012) promoted a systematic literature review and a survey with 115 practices to investigate both views regarding the benefits and limits of test automation. The results corroborated the grey literature and indicated that a high degree of infrastructure automation saves costs. While Hynninen et al. (2018) promoted a survey to explore industry practices concerning software testing, the study reported that more and more companies had adopted test automation and more sophisticated testing infrastructure, even in mission-critical software. Although test automation is an answer to facing infrastructure complexity, it is not an easy task, and practitioners considered one of the most challenging among all testing activities, according to a survey with 72 practitioners promoted by Garousi et al. (2020).

In addition, the grey literature also indicates mocking services, APIs, and infrastructures as a strategy to reduce costs and simplify the construction of the test infrastructure. Table 7 lists some tools for service mocks and DSP mechanisms. In the formal literature, Chen et al. (2017) state that mocking reduces testing costs in large-scale systems by reducing dependencies on external service providers. However, the author pointed out that the cost reduction would be more significant with automated approaches for creating and maintaining mocking services.

To conclude, the formal literature brings some contributions that help practitioners deal with the challenging context of the DSP applications test infrastructure. Benkhelifa et al. (2019)

Table 7
Tools list.

Tool name	Author	License	Main testing approaches support	Features and specific uses	Focused on data stream context?	Documents citation
Alooma Plataform	Alooma Inc.	Proprietary	Contract/Schema Test	Data Validation, Data Replay	Yes	1
Ansible	Michael DeHaan and Red Hat Inc.	GNU General Public License v3.0	System Test	Test Infrastructure Automation	No	2
Apache Griffin	Apache Foundation	Apache License 2.0	Contract/Schema Test; Regression Test	Data Validation	No	3
Artillery	Artillery Software Incorporation	Open Source MPLv2 license	Performance Test	Load generator, Testing Utilities	No	1
Avro Random Generator	Confluent Inc.	Apache License 2.0	Unit Test; Integration Test; Performance Test; Contract/Schema Test	Data Generator	Yes	2
Awaitility Library	Johan Haleby and Community	Apache License 2.0	Integration Test	Asynchronous System Test, Time Issues	No	1
AWS Deequ	Amazon Inc.	Apache License 2.0	Contract/Schema Test	Data Validation	No	3
Chaos Monkey	NetFlix	Apache License 2.0	Chaos Test	Chaos Generation	No	1
Confluent CLI	Confluent Inc.	Confluent Community License Agreement Version 1.0	System Test	Test Infrastructure Automation	Yes	1
Ducktape	Confluent Inc.	Apache License 2.0	Uni Test; Integration Test; System Test; Performance Test	Testing Utilities	Yes	1
Ecto Stream Factory	Igor Barchenkov	MIT License	Uni Test; Integration Test; Performance Test; Property-based Test	Data Generator	Yes	1
Embedded Kafka	Community	MIT License	Unit Test; Integration Test	Mock Stream Processing Topologies	Yes	6
Fake.js	Marak	Copyright Marak Squires	Unit Test	Data Generator	No	2
Faker (Python)	Daniele Faraglia	MIT License	Unit Test	Data Generator	No	1
Flink Spector	ottogroup	Apache License 2.0	Unit Tests; Regression Test	Framework to define unit tests for Apache Flink	Yes	6
Flink Testing Utilities	Apache Foundation	Apache License 2.0	Unit Tests; Regression Test	Testing Utilities, Time Issues, Clock Control	Yes	6
Fluent Kafka Streams Tests	Backdata	MIT License	Unit Tests; Regression Test	Unit Testing Utilities for Kafka	Yes	5
Gatling	Gatling Corp.	Apache License 2.0	Performance Test	Load Test	No	1
Grafana	Grafana Labs	AGPL-3.0 License	Performance Test	Log and Monitoring	No	4
Great Expectations	Superconductive	Apache License 2.0	Contract/Schema Test	Data Validation	No	3
Intel Platform Analysis Technology	Intel Corp.	Proprietary	Performance Test	Log and monitoring of hardware and operation systems metrics	No	1
Jackdaw - Test Machine	Funding Circle	BSD-3-Clause License	Integration Tests	Testing Utilities	Yes	1
Jenkins	Jenkins	MIT License	System Test	Test Infrastructure Automation	No	5
Jepsen	Jepsen	Apache License 2.0	Chaos Test	Chaos Generation	No	1
JMeter	Apache Foundation	Apache License 2.0	Performance Test	Load Test	No	7
Kafka Datagen Connector	Confluent Inc.	Apache-2.0 License	System Test; Performance Test	Data Generator	Yes	3
Kafka for Junit	Markus Günther	Apache License 2.0	Unit Test; Regression Test	Unit Testing Utilities, Junit Support fo Kafka	Yes	5
Kafka Streams Testing Utilities	Apache Foundation	Apache License 2.0	Unit Tests; Regression Test	Testing Utilities, Time Issues	Yes	8
Kafkameter	Signal	Apache License 2.0	Performance Test	Load Generator	Yes	2

(*continued on next page*)

Table 7 (*continued*).

Tool name	Author	License	Main testing approaches support	Features and specific uses	Focused on data stream context?	Documents citation
Kcat	Magnus Edenhill - Apache	Copyright (c) 2014–2021 Magnus Edenhill	Unit Test; Integration Test	Testing Utilities	Yes	4
Kinesis Data Generator	Amazon Inc.	Apache License 2.0	System Test; Performance Test	Data Generator, Load Generator	Yes	4
Ksql-datagen	Confluent Inc.	Confluent Community License Agreement Version 1.0	Performance Test; System Test	Data Generator	Yes	4
MemoryStream - Apache Spark	Apache Foundation	Apache License 2.0	Unit Test; Regression Test	Replay data stream values stored in memory	Yes	4
Mockaroo	Mockaroo LLC	Proprietary	Unit Test; Integration Test; System Test	Data Generator	No	2
Mockedstreams	Jendrik Poloczek	Apache License 2.0	Unit Tests	Mock Stream Processing Topologies	Yes	3
Mockito	Community	MIT License	Unit Test	Mocking framework for unit test	No	5
Nifi Script Tester	Matt Burges	Apache License 2.0	Unit Test	Infrastructure Mock	Yes	2
Passert - Apache Beam	Apache Foundation	Apache License 2.0	Unit Tests; Regression Test	Assertion for data collections	Yes	1
Pepper-Box	Great Software Laboratory	Apache License 2.0	Performance Test	Load Generator	No	5
Sangrenel	Jamie Alquiza	Apache License 2.0	Performance Test	Load Generator	Yes	2
SBT Coverage	Community	Apache License 2.0	Integration Test	Tracking code coverage	No	1
Scalacheck	Typelevel.scala	BSD-3-Clause license	Unit Test; Property-based test	Automated property-based testing	No	2
ScalaTest	Community	Apache License 2.0	Unit Tests; Integration Tests	Testing Utilities	No	16
Spark Testing Base	Holden Karau	Apache License 2.0	Unit Test; Integration Test; Regression Test	Base classes to write tests for Spark, Time Issues	Yes	8
Spring Cloud Stream Test Support	Spring	Apache License 2.0	Unit Tests; Integration Tests; Regression Test	Testing Utilities	Yes	3
StreamData	Andrea Leopardi and José Valim	Apache License 2.0	Property-based Test	Data Generator	Yes	3
Terraform	HashiCorp	MPL-2.0 License	System Test	Test Infrastructure Automation	No	4
Thundra	Thundra	Proprietary	Chaos Test	Chaos Generation	No	2
WireMock	Tom Akehurst and Community	Apache License 2.0	Unit Test; Integration Test	Mock APIs and External Services	No	3
ZeroCode Samurai	Zerocode	Apache License 2.0	Performance Test	Testing Utilities, Automated support to tests	Yes	4

reviewed cloud-based testing research, identified challenges, and characterized requirements associated with cloud testing environments, then proposed a framework for cloud-based virtual infrastructure testing servitization. Harsh et al. (2019) present a cloud testing architecture focused on large-scale distributed applications, the proposed solution benefits from managing resources and services to execute end-to-end tests simulating realistic operational conditions. Both works present approaches compatible with typical test infrastructures of DSP applications, supporting practitioners in constructing test infrastructures.

Obtaining Test Data. In the grey sources, we encountered statements emphasizing the importance of having good test data for realistic testing. However, in most cases, accurate test data based on real data is out of the question due to data security and privacy regulations such as the General Data Protection Regulation (GDPR). Strategies based on manually generating custom data can be a huge, time-consuming, and non-trivial task, whereas automatic test data generation strategies also present challenges in generating relevant test cases.

When historical data is available to develop the test dataset, it only provides an initial model of how the data streams are in the production environment, which does not guarantee a reliable test oracle. In grey literature, a practitioner reported that historical data results are not easily verifiable. Historical data may consist of only the inputs without the desired outputs, or if there are outputs, they may be unreliable. Therefore, generating and validating outputs to construct an oracle is necessary. This activity depends on the documentation containing the characterization and examples of desired outputs. Another problem is that historical data may not include new features.

In the formal literature, we found studies in line with the grey literature's findings regarding testing data challenges. The work by Felderer et al. (2019) provides an overview of the current state of testing data-intensive software and cites test data quality as a relevant factor for testing data-intensive systems. The study by Benkhelifa et al. (2019), focused on testing cloud-based infrastructure, considers security issues and legal impediments to real business data use a challenge. Finally, the difficulty of generating high-quality data sets covering all relevant characteristics is a practical and methodological obstacle, as exposed by the study by Hummel et al. (2018).

Time issues. Time is a fundamental factor in DSP, as it affects aggregation and joining mechanisms, as well as the order in which events are generated, processed, and ingested (Stonebraker et al., 2005). The grey literature highlights the difficulty of simulating business-realistic temporal characteristics and evaluating time constraints in a test environment. Therefore, special handling of timing issues is required when setting up tests, and tooling support is essential for controlling the application's clock and watermarks. Testers can adopt a simulated clock to gain control for testing purposes in test environments. DSP frameworks offer control functions and interfaces in their test utilities to support the simulation of processing time.

Practitioners have reported that test execution would take a long time without clock control mechanisms in the test environment. Although no formal literature explicitly addresses this topic, the documentation of DSP frameworks such as Flink,[5] Kafka,[6] and Spark[7] describes clock control, functions for skipping some test cycles, and artificial watermark generation mechanisms. Test acceleration can be achieved using a simulated clock to generate time events faster than in real-time. For example, Flink supports event-time processing, where timestamps are used to process events. Clock acceleration in Flink can be achieved by implementing a version of the WatermarkGenerator interface to generate artificial watermarks that advance event time. Another option is to accelerate test execution by skipping some test cycles through functions that advance the clock. For instance, this approach can be implemented in Apache Kafka using the *TopologyTestDriver.advanceWallClockTime* function from Kafka Testing Utilities, which effectively skips over periods of time in the test environment to speed up the tests. However, this approach can result in a loss of precision and potentially hide bugs. Therefore, it is essential to consider the trade-off between test execution speed and result accuracy.

Another aspect related to time issues that testers should be aware of is the processing time interval in the test environment. The choice of the processing time interval may impact the testing process's accuracy and efficiency. A long interval may lead to inaccurate results, while a short processing interval means more frequent updates and accurate results but accompanied by an increase of computational overhead. Choosing the processing time interval depends on the application's characteristics, particularly the algorithms and functions that evaluate time factors or rely on timely updates. Some functions may work adequately over varying processing time ranges, while others may be quite sensitive and require more restricted processing time intervals. Experiments can be conducted to select the processing time, and it is essential to balance the need for accuracy and computational costs. In the test environment, it is recommended to reproduce the configuration of the processing interval faithful to the one used in production.

In the formal literature, we find research that addresses the difficulties that temporal aspects impose on testing distributed applications. Works such as (Lima and Faria, 2017; Lima, 2019; Lima et al., 2020) address the conformance checking, observability, and controllability of time-constrained distributed systems. The test infrastructure communication overhead influences the test results, so checking the conformance of the observed execution traces against the specification becomes challenging. Researchers proposed UML-based specifications enriched with time constraints and algorithms for decentralized conformance checking in this context. Moreover, the study by Hierons and Ural (Hierons and Ural, 2008) presents an algorithmic approach for checking the interaction sequence of distributed components and the test specification. The approach is free from controllability and observability problems. Finally, the study by (Charaf and Azzouzi, 2016) presents an approach to generate timed distributed testing rules that avoid synchronization problems by considering the delay of messages between the under-test implementation and testers.

6.1.2. RQ2. What are the testing purposes?

This section discusses the main testing purposes extracted from the grey literature. The testing purposes were categorized according to the ISO/IEC 25010 quality model since the software testing aims to improve software quality, and we identified an association with the model categories. Table 8 summarizes the main aspects of each testing purpose, and the remainder of the section describes them in more detail.

Functional suitability. In the grey literature, we identified that functional correctness is one of the concerns when testing DSP applications. However, we also found evidence suggesting that the level of concern about delivering correct results changes according to the role of DSP play in organizations' business. The findings indicate that using unit tests for business rules functions is the first step towards achieving correctness. However, the unit tests are insufficient to ensure that these functions will not fail in production by not exercising many features of complex distributed software. Although more costly and challenging, the grey sources indicate that testing strategies should include integrated and system tests in environments similar to production, where many DSP characteristics will be present.

In the academic context, we find discussions about the variation of the notion of the correctness of complex data-intensive systems. The work by Hummel et al. (2018) highlights the difficulty of determining whether an operation result is correct or incorrect. Such difficulty hinders the testability of this type of software, as it is not possible to have accurate test data to test. Furthermore, as Gunawi et al. (2014) explained, in some business contexts, incorrect results may be tolerated in preference to other requirements such as reliability. These statements support the GLR's finding that the notion of correctness varies according to the context. Following the same path, the work by Akidau et al. (2015) treats the level of correctness of a DSP system in a tradeoff with non-functional requirements. This approach starts from the premise that the correctness level can vary according to the problem domain and available resources. The work proposes the Dataflow Model, an approach based on formal models to help developers properly balance the dimensions of correctness, latency, and cost dimensions to suit their context.

Regarding correctness-focused approaches, Kallas et al. (2020) propose an approach based on differential tests to verify the correctness of DSP applications. The work presents a testing framework that includes the DiffStream library for assisted execution of two implementations in response to the same input stream.

[5] https://nightlies.apache.org/flink/flink-docs-master/docs/dev/datastream/event-time/generating_watermarks/

[6] https://kafka.apache.org/24/javadoc/org/apache/kafka/streams/TopologyTestDriver.html

[7] https://spark.apache.org/docs/latest/api/java/org/apache/spark/util/Clock.html

Table 8
Key points of testing purposes.

Testing purpose	Key point	Description
Functional suitability	Functional correctness	A key concern is ensuring that DSP applications deliver correct results. Unit tests for business rules functions are the first step towards achieving correctness, but integrated and system tests in environments similar to production are also necessary.
	Variation in correctness	The notion of correctness varies according to the business context, as incorrect results may be tolerated in preference to other requirements such as reliability. The level of correctness of a DSP system can also be traded off with non-functional requirements.
	Differential testing	Approaches based on differential testing, such as DiffStream, can be used to verify the correctness of DSP applications. This involves executing two implementations in response to the same input stream and comparing the equivalence of their outputs.
	Mutation testing	Mutation testing improves test suite performance by generating syntactic variations of the original program using mutation operators to reveal programming errors caught by the test suite.
Performance efficiency	Time-behavior	Time behavior is affected by storage write speed, algorithmic characteristics, network throughput, latency, memory, processing power, DSP topology, and configurations. Testing the application under stress to understand data volume and latency is crucial.
	Resource utilization	The rational use of infrastructure resources is crucial as it financially impacts companies and their products' competitiveness. Performance testing assesses the application's utilization of hardware resources to promote optimizations.
	Capacity	Knowing infrastructure capabilities and understanding the bottlenecks and resources needed to scale is crucial. Testing the infrastructure's operating limits helps to map bottlenecks and delimit operational limits.
Reliability	Fault tolerance	Data stream processing applications must be fault-tolerant, as hardware failures, network oscillations, and other issues can cause the system to fail.
	Disaster recovery	The ability of the system to recover from severe failures that overwhelm fault tolerance mechanisms. The system must be restored to normal operation as quickly as possible after a catastrophic failure, and checkpoints and data loss must be considered.
	Chaos testing	It can help uncover potential system weaknesses and provide valuable insights for improving fault tolerance and recovery mechanisms.
	Overhead and performance degradation	Fault tolerance mechanisms can lead to performance degradation due to overhead, so tradeoffs between fault tolerance and the impact on system performance must be considered.
Maintainability	Modifiability	The ability to modify code without introducing new defects is crucial for cost-effective bug fixing and system evolution. This can be achieved through regression testing, ensuring that codebase changes do not negatively impact existing functionality.
	Architecture	A DSP system's architectural design can significantly impact its maintainability. Microservices-based architectures, for example, may reduce the testing effort required for regression testing by restricting tests to the modified microservice and its interfaces.

The framework also provides an algorithm to compare the equivalence of the outputs generated by the two implementations. We also found the research by Gutiérrez-Madroñal et al. (2019) addressing the mutation testing technique to improve the performance of test suites in verifying the correctness of event processing programs. The technique generates syntactic variations mutants of the original program by applying mutation operators. Artificial faults injected into the original program represent developers' programming errors, and running the tests reveals whether the test suite caught the inserted error.

Performance efficiency. We found 41 quotes associated with performance efficiency when testing DSP applications. The testing aims to verify the application response times in application contexts where time behavior is essential. We also identified a concern with the rational use of resources. In these cases, tests aim to measure the ideal amount of allocated resources, subsidizing the development of strategies for efficient resource sizing. Finally, the tests can also focus on the system's operational capacity, helping to map bottlenecks and delimit the operational limits of the infrastructure.

We highlight [Source 10], a white paper produced by a hardware vendor in which the results of performance experiments on a platform focused on DSP are presented. This white paper highlights the industry's concern about the DSP performance of hardware platforms. There is also a concern about performance in the academic context, as we found several works discussing the performance of DSP systems (Tantalaki et al., 2020; Mishra et al., 2020; Zeuch et al., 2019). The study by Karimov et al. (2018) introduces recommendations and metrics for the performance evaluation of flow data processing systems. Other studies also compare the performance of DSP frameworks (Samosir et al., 2016; Chintapalli et al., 2016; Shahverdi et al., 2019) as well as the performance evaluation of specific techniques (Del Monte et al., 2020; Tun et al., 2019). Academic works also carry out studies on the efficient use of resources by data flow frameworks in cloud-based infrastructures. Chatterjee and Morin (2018) perform a comparative study about the use of resources of three frameworks for the same application.

Reliability. The business context of the data stream requires high-reliability software; in this context, the GLR reliability-related findings essentially focus on fault tolerance and recoverability. Fault tolerance targets keeping the system running under adverse conditions, such as hardware, network, and software end third-party services. At the same time, recoverability focuses on the system's ability to recover from a failure with the minor damage possible. Chaos Testing strategies reported by practitioners in Section 5.5.5 address fault tolerance and recoverability. In this approach, controlled experiments are conducted using specific tools to simulate failures, testing the application's ability to continue running in the face of failures and its ability to recover from failure scenarios. These experiments help uncover potential system weaknesses and provide valuable insights for improving fault tolerance and recovery mechanisms. By exposing the system to various failure scenarios in a controlled environment, Chaos Testing can also help to build confidence in the system's ability to handle unexpected events.

Fault tolerance in DSP has been addressed by several works over time, confirming the relevance of the issue that came to light in the GLR. Academic studies such as (Shah et al., 2004; Balazinska et al., 2009; Akber et al., 2021) bring fault tolerance techniques

to the DSP context. Wang et al. (2022) perform a systematic overview and classification of fault tolerance approaches for DSP. It also proposes an evaluation framework tailored to measure quality in terms of runtime overhead and recovery efficiency of fault tolerance approaches. Finally, the work presents three directions for developing future fault tolerance approaches in DSP: Adaptive Checkpointing, Integration of Modern Hardware, and Parallel Recovery. Although fault tolerance is a desirable feature in DSP, such mechanisms can cause performance degradation due to overhead. Vianello et al. (2018) performed an evaluation experiment to measure the overhead cost generated by a fault tolerance mechanism in a DSP system. The results indicate a 10-fold increase in latency when these mechanisms are active. The fault tolerance and overhead tradeoff have also been reported in the grey literature Zachary Ennenga in [Source 18].

Maintainability. The maintainability is pervasive in several quotes, particularly those associated with the DSP Complexity and Test Infrastructure challenges presented in Section 5.3.1. Such challenges increase the difficulty and cost of DSP application modification and testing. Mostly we have found quotes referring to the practitioners' concern with the ability to modify code without introducing new defects. At this point, practitioners emphasize the importance of regression testing in dealing with modifiability, and such a strategy is described in Section 5.5.3.

In the formal literature, studies focus more on architectural aspects as a relevant factor of maintainability in the context of DSP software (de Assuncao et al., 2018; Hoque and Miranskyy, 2018; del Rio Astorga et al., 2018). However, as the Silva et al. (2018) study exposes, regression testing generally requires robust automation of the development infrastructure, and the construction and maintenance of such a test suit for continuously updated regression tests are challenging. The difficulty of maintaining large numbers of tests can be mitigated through regression test reduction techniques such as Requirement Based, Coverage Based, Genetic Algorithm, etc (Suleiman et al., 2017). Especially the study by (Hoque and Miranskyy, 2018) shows that in contrast to monolithic architectures, microservices-based architectures restrict the execution of regression tests to the modified microservice and its interfaces, leading to a reduction of testing efforts.

6.1.3. RQ3. What are the approaches and specific types of tests performed?

This RQ identified and selected which testing approaches practitioners have been using according to the grey literature. The findings are based on 155 citations in 82 grey sources reporting the testing approach. We first present the testing approaches according to the target of test, then focus on the most relevant specific approaches. As an aid, we have included Tables 9, 10, 11, 12, 13, and 14 with a brief summary of each testing approach in the following subsections. These tables highlight fundamental aspects, testing purposes, challenges, tools, and variables to consider for each approach.

Approaches by Target of Test. This topic covers DSP application testing considering relevant aspects regarding unit, integration, and system testing levels. Although unit tests are essential for verifying module functionality, practitioners note that unit testing may not address distributed systems' unique characteristics and non-functional requirements, such as end-to-end response time. Stepien and Peyton (2020) and Buchgeher et al. (2020) address the limitations of unit testing in distributed systems. Hill et al. (2009) and Li et al. (2006) discuss strategies and tools for developing effective unit tests that meet distributed systems characteristics. Integration and system-level tests are crucial for addressing the unique characteristics of distributed systems, which are significant causes of defects in DSP applications. However, setting up realistic system test infrastructures can be complex and expensive due to the increasing number of diverse technologies and services. We identified in the grey literature tools for components mocking and infrastructure automation that has been used to address this challenge. The academic literature highlights the difficulty of testing large-scale distributed systems due to the complexity and size of the infrastructure, as

Table 9
Summary of approaches by target of test.

Approaches by Target of Test
Fundamental aspects: unit tests for DSP applications involve particularities on stateful and timed process operators. Integration and system-level testing are important to address the specific characteristics of distributed systems, which are associated with several defects in DSP applications. **Related challenge:** The *Testing Infrastructure Complexity* challenge as we found reports about the complexity and cost of setting up realistic system test infrastructures. The *Time Issues* challenge is also addressed on behalf of timed process operators. **Tools adopted:** Terraform, Jenkins, Confluent CLI, Ansible, Docker, Vagrant, and AWS CloudFormation. (component mocking and infrastructure automation to address complex infrastructures) **Stateful operations:** stateful operations involve maintaining a state between successive invocations of the operation, which can lead to complex testing scenarios. The key challenges in testing stateful operations include verifying that the state values are updated correctly, and that cache data is purged as required. DSP platforms, such as Apache Flink and Kafka Streams, provide utility classes to support stateful operator testing. **Timed process operators:** rely on timestamps and the application clock to control their behavior, making it necessary to manipulate time variables. Testing timed process operators require controlling the clock and providing event timestamps. DSP platforms provide utility classes and features to support timed process operator testing, as discussed in the topic *Time Issues* from Section 6.1.1.

Table 10
Summary of performance testing key points.

Performance Testing
Fundamental aspects: Focus on evaluating whether the performance meets the application requirements, optimizes computational resources, and brings cost savings. For optimization purposes, tests must be executed with different configurations due to the many variables affecting the results. **Testing Purpose:** Performance Testing essentially addresses the *Performance Efficiency* testing purpose. **Tools adopted:** JMeter, Pepper-Box, ZeroCode Samurai, Kafkameter, Sangrenel, Artillery, Gatling, Ducktape and Grafana. These tools support generating loads and monitoring metrics. **Variables to consider:** Hardware resources, network resources, number of topics, producers, consumers, message size, timeout values, cache sizes, replication factors, etc.

Table 11
Summary of regression testing key points.

Regression Testing
Fundamental aspects: address regression bugs and application performance degradation introduced by software evolution. Automated CI-based test infrastructure is necessary to perform these tests. **Testing Purpose:** Regression testing primarily addresses the testing purpose of Maintainability. **Tools adopted:** Apache Griffin, Flink Spector, Flink Testing Utilities, Fluent Kafka Streams Tests, Kafka for Junit, Kafka Streams Testing Utilities, MemoryStream, Passert, Spark Testing Base and Spring Cloud Stream Test Support. **Variables to consider:** Configuration changes, non-determinism of distributed systems, microservices-based architectures, test coverage of input parameters, data privacy issues, and cost of resources and time. **Microservices-based architectures:** are widely adopted in DSP applications infrastructure but bring peculiarities to regression tests. Building a regression testing infrastructure that supports isolated microservice and integration tests with other components that may affect their functioning is necessary.

Table 12
Summary of property-based testing key points.

Property-based Testing
Fundamental aspects: automatically generate representative test cases based on variable properties and refine them through a process called shrinking. It is a quick and inexpensive strategy to get more relevant test data. **Related Challenge:** It relates to the *Obtaining Test Data* challenge, as it is a test approach that automatically generates test data mitigating the problem of missing real data and generating irrelevant data. **Tools adopted:** ScalaCheck, StreamDatam and EctoStreamFactory property-based testing library. Apache Flink and Apache Spark Streaming frameworks have property-based testing tooling based on ScalaCheck. **Variables to consider:** properties of the input data and the temporal logic to guide the generation of random streams with time stamps. **Mocking API:** The adaptation of property-based testing tools to generate mock API data for unit and integration tests when API services are unavailable in testing environments.

Table 13
Summary of chaos testing key points.

Chaos Testing
Fundamental aspects: reduce the likelihood of failures in production. Chaos test subjects the applications to adverse conditions, such as network instabilities, hardware failures, and third-party service instabilities, and tests its fault tolerance ability and recoverability. **Testing Purpose:** Essentially, it is related to testing purpose *Reliability*. **Tools adopted:** Chaos Monkey randomly terminates service instances to test their fault tolerance. Jepsen allows the simulation of network partitions and faults to test the system's consistency and availability. Thundra is a serverless observability platform with a chaos engineering tool for testing serverless applications' fault tolerance. WireMock includes a feature for simulating network faults to test the system's fault tolerance. **Optimizing fault tolerance configurations:** Chaos testing can be used to optimize fault tolerance configurations for DSP jobs, which can help balance performance and availability. **Variables to consider:** Network instabilities, hardware failures, software failures, third-party service instabilities, timing issues, resource utilization and QoS constraints.

Table 14
Summary of contract/schema testing key points.

Contract/Schema Testing
Fundamental aspects: Ensure that messages exchanged between different system components comply with a specified message schema. This is essential for preventing compatibility breakages in data stream processing applications, which can result in crashes or incorrect data processing. **Testing Purpose:** Supports the *Maintainability* testing purposes, as it helps prevent contract incompatibility caused by updates. **Tools adopted:** Apache Avro provides functionality for schema evolution and backward and forward compatibility. AWS Deequ supports both static and dynamic schema validation. Great Expectations provides functionality for defining and validating data expectations, including schema validation. **Variables to consider:** message format, message source and destination, integration with third-party systems, backwards and forward compatibility, and recovery from errors related to schema incompatibilities.

evidenced by studies such as Hanawa et al. (2010) and Harsh et al. (2019).

Performance Testing. The abundance of findings from the grey literature on performance testing highlights its relevance to the industrial context. At the same time, several academic works address the performance evaluation of DSP applications and associated contexts. Academic works bring tooling and technical contributions that meet the concerns expressed by practitioners about performance tests, so on this topic, there is consonance between industry and academia.

Findings from the grey literature indicate that performance testing aims to assess whether the performance meets the application requirements and helps bring cost savings by enabling developers to optimize computational resources. Practitioners state that DSP application performance tests must be executed with different configurations due to the multiple variables that affect the results, including hardware and network resources, number of topics, producers, consumers, message size, timeout values, cache sizes, replication factors, and more. Our research listed several tools for generating loads and monitoring metrics during performance testing.

Academic interest in performance testing has been increasing since the 2000s, particularly in contexts related to DSP's typical features, such as large-scale distributed applications, cloud, and microservices that handle numerous simultaneous requests (Dumitrescu et al., 2004; Zhou et al., 2013; De Barros et al., 2007). In 2007 (De Barros et al., 2007) proposed various techniques to overcome challenges in building environments and performance test tools that accurately simulate the same conditions observed in the production environment of large-scale distributed systems, also providing techniques to characterize load patterns for tests. In 2015, (Jiang and Hassan, 2015) surveyed the state of load testing research and practice, reporting techniques for designing, executing, and analyzing the results of a load test. The paper by Eismann et al. (2020) lists the difficulties that microservices architectures bring to performance testing, as it requires additional care when setting up test environments or conducting tests.

In the formal literature, we can find several benchmarking works of DSP frameworks (Cappellari et al., 2016; Chintapalli et al., 2016; Samosir et al., 2016; Sun et al., 2018; Karimov et al., 2018; Shahverdi et al., 2019; Mishra et al., 2020). These works focus on evaluating the performance of DSP frameworks in specific application contexts by varying functionalities and configurations. They also document methods and tools for conducting performance tests and list performance metrics considered relevant to the DSP context, such as memory consumption, CPU load, response time, and network throughput.

Other studies bring specific contributions of methods and tools to evaluate DSP applications' performance. For example, Pagliari et al. (2020) propose the NAMB tool (Not only A Micro-Benchmark), a generic generator of DSP applications prototypes that provides high-level descriptions language to support developers in quickly building and assessing the performance impact of design choices. Garcia et al. (2022b) presents SPBench, a framework for benchmarking DSP applications which aims to support users in creating custom benchmarks of real-world DSP applications. The tool offers an Application Programming Interface (API) and Command Line Interface (CLI).

We also find academic works proposing and evaluating techniques to improve DSP performance in specific contexts. Nardelli et al. (2019) proposed heuristics to calculate the placement of DSP applications in geo-distributed infrastructures. Wu et al. (2018) propose reducing the intra-node inter-process communication latency by reducing memory copy operations and waiting time for every single message. Garcia et al. (2022a) analyzes the impact of micro-batch and data intensity on DSP applications with different parallel programming interfaces.

Regression Testing. Grey literature shows practitioners are concerned about regression bugs and application performance degradation caused by software evolution. To address these issues, practitioners consider regression testing essential, and the need for an automated CI-based test infrastructure to perform these tests has emerged. While there is no specific formal literature on regression testing in the DSP context, related works in real-time distributed systems, microservices architectures, and large-scale software have been explored and discussed.

Yamato (2015) proposed a testing framework for real-time distributed systems to prevent new bugs introduced by configuration changes. (Babaei and Dingel, 2021) tackled the non-determinism issue in distributed systems regression testing with MRegTest, a replay-based framework that facilitates deterministic

replay of traces and reduces testing costs by replaying executions that modify critical user-specified variables.

Microservices-based architectures in DSP applications present challenges for regression testing. (Kargar and Hanifizade, 2018) proposed an automated method that integrates microservices regression tests into Continuous Delivery (CD) steps. Gazzola et al. (2022) propose a tool that monitors deployed service executions at run-time, recording executions that can be later converted into regression test cases for future version services. However, data privacy issues may arise when monitoring production execution, as noted by the authors.

Academic studies also have explored the resource and time costs of running numerous regression tests on large-scale systems, as shown in Orso et al.'s work (Orso et al., 2004). Techniques such as minimization, selection, and prioritization have been developed to address these issues. Minimization eliminates redundant test cases, selection identifies relevant test cases for recent changes, and prioritization orders test cases for early failure detection (Yoo and Harman, 2012; Suleiman et al., 2017).

Property-based Testing. Property-based testing was reported as a technique adopted in the DSP context to generate representative test cases based on variable properties and undergo a refinement process called shrinking, which minimizes the number of inputs required to reproduce a failure. This method helps address the problem of missing real data for testing while mitigating irrelevant data from random data generators. It is a low-cost strategy that can be easily applied and yield good test coverage.

Grey Literature shows that the DSP community is already aware of the benefits of the property-based testing approach for the DSP context; it has also been documented in the formal literature. Espinosa et al. (2019) and Riesco and Rodríguez-Hortalá (2019) have brought property-based testing tooling to the Apache Flink and Apache Spark Streaming, respectively. These tools are based on the ScalaCheck library and use temporal logic to generate random streams and verify time-related properties. Furthermore, API testing is one of the uses of property-based testing, and the work of Karlsson et al. (2020, 2021) exemplifies its applicability for this purpose. Such use is also relevant for DSP, where APIs are increasingly incorporated. Additionally, property-based tools can be adapted to generate mock API data for unit and integration tests when API services are unavailable in testing environments.

Chaos Testing.

As distributed software and hardware components increase, system abnormalities at runtime become more likely. Chaos engineering has been reported in the grey literature as a technique for exercising and verifying the fault tolerance of distributed, large and complex software systems. Practitioners are concerned about infrastructure failures in a production environment that could cause the application to malfunction. They reported testing practices in which the system is deliberately exposed to failure in order to verify that the auto-recovery mechanisms work satisfactorily.

Chaos engineering is attracting the attention of both industry and research, as it is a technique for exercising and verifying the fault tolerance of distributed, large and complex software systems. Netflix employed the approach to testing the fault tolerance of microservices-based infrastructures. The work by Tucker et al. (2018) analyzes the implementation of the chaos engineering techniques and reports the benefits generated for the company's business.

Chaos engineering has been explored in the formal literature in contexts associated with DSP, such as cyber–physical systems (Konstantinou et al., 2021), cloud infrastructure (Torkura et al., 2020), and containerized applications (Simonsson et al., 2021). Specifically, in the DSP context, the work by Geldenhuys et al. (2021) proposes the Khaos tool. This tool employs Chaos Engineering principles to allow automatic runtime optimization of fault tolerance configurations for DSP jobs.

Chen et al. (2022) introduced a framework for adopting chaos engineering in large-scale distributed big data systems. The framework focuses on fault tolerance, resilience, and reliability and includes steps such as exception injection, exception recovery, system correctness verification, observable real-time data statistics, and result reporting. The work is based on real industrial applications and draws from experiences introducing Chaos Engineering at the Swedish company ICA Gruppen AB.

Contract/Schema Testing. Reports in the grey literature warn that a lack of data schema correctness control can cause compatibility breakages in DSP applications. Practitioners recommend adopting unit and integration tests to validate that interfaces follow the message schema. A compatibility fault tolerance approach is recommended for third-party modules, such as external APIs. The ability to interoperate between different schemas versions to support schema evolution has also been identified. Schema registry management tools, such as Apache Avro and Confluent Schema Registry, can provide compatibility models among schema versions.

Data schema incompatibility is a common problem in distributed systems, and several research projects have addressed it (Blanchi and Petrone, 2001; Baqasah et al., 2015; Filip et al., 2019; Litt et al., 2021). DSP applications, which use APIs extensively, also face API interface compatibility issues. In this matter, Bustamante and Garc'©s (Bustamante and Garcés, 2020) addressed API incompatibility by adding an intermediary service between IoT devices and API servers that identifies different protocol versions and performs compatibility adjustments. Yasmin et al. (2020) proposed an automatic verification approach to identify impacted operations by deprecated API elements, collecting the OpenAPI Specification (OAS) format and performing a source code scan to verify calls.

In the context of DSP, Corral-Plaza et al. (2020) proposed an architecture for processing heterogeneous data sources that typically involve JSON, XML, YAML, or unstructured raw data. To address recognizing and adapting to evolution changes, the authors proposed five steps to transform heterogeneous data into simple events: Data Consumption, Data Format Detection, Data Homogenization, Schema Generation, and Simple Event Generation. The architecture was evaluated in a real-world case study of IoT sensors, demonstrating the ability to process and analyze heterogeneous data automatically. This work is also classified as a fault tolerance approach as it provides automatic tolerance to schema changes.

6.1.4. RQ4. What are the strategies adopted by practitioners to obtain testing data?

In our grey literature review, we identified 65 quotes from 33 studies addressing issues on data for testing DSP applications. Practitioners generally recognize the importance of having representative test data that closely resemble real data. Questions regarding data privacy and security emerged, and several strategies were listed. These strategies typically involve using various types of synthetic data generators, historical or production real data, and combinations of real and synthetic data. Table 15 summarizes the key points of the strategies for obtaining test data. The remainder of this section discusses the advantages and disadvantages of the strategies and presents relevant works from the formal literature associated with each strategy.

Historical Data. Practitioners consider using historical data for testing interesting because it exposes the application to realistic

Table 15
Key Points of Strategies for Obtain Testing Data.

Strategies to obtain testing data	Key points
Historical Data: It is real stream data obtained from the application in production.	• Exposes the application to realistic scenarios. • Difficulties in obtaining historical data due to privacy and security concerns. • May not provide coverage for detecting bugs in the future. • Having historical data does not necessarily mean having a test oracle. • May not be applicable to new features or versions of an application.
Production Data Mirroring: Strategy based on redirecting a replica of input data streams from the production to the test environment. Compares the outputs of two software versions running simultaneously.	• The application is tested with real data. • Test could be very time-consuming as there is no possibility of speeding up the execution. • Incurs costs associated with building and maintaining a replica of the production infrastructure. • Execution in "shadow mode" is a strategy based on an automatic comparison engine to ensure that sensitive data remains secure.
Semi-synthetic data: The technique involves automatic data generation methods based on real data. Advanced techniques are based on extracting statistical properties from real data to establish a data model to generate new data.	• Suitable for customizing the data in order to contemplate more complex test cases. • Can be used to expand a limited amount of real data available. • Address privacy issues, as semi-synthetic data generation techniques focused on privacy preservation, have been reported.
Synthetic data: Automatic data generation. Various techniques are available, ranging from simple random data generators to sophisticated algorithms for data generation.	• Take as input a formal description of the data schema to generate test data. • Exploring information extracted from UML diagrams helps generate more representative test data. • Generators employ various statistical methods that are configurable by the user. As a result, a solid understanding of statistics is crucial to effectively utilize these tools and generate high-quality test data.

scenarios and facilitates real-world bug reproduction in a testing environment. In favor of using historical data, practitioners argue that real data streams can be quite complex, making them difficult to reproduce manually or with automatic generators. While there are advantages, several quotes also claim the impossibility of obtaining real data due to privacy and security concerns. Such issues were reported in Section 5.3.4, which deals with test data-related challenges and later discussed in 6.1.1.

Furthermore, historical data may not provide adequate coverage for detecting bugs in the future. Additionally, having access to historical data does not necessarily mean having a test oracle, as they can only be inputs without corresponding expected outputs. Even when outputs are available, the oracle is not guaranteed to be reliable. Also, historical data may not be applicable to new features or versions of an application, rendering it incompatible. As a result, historical data may not be sufficient for comprehensive testing, and complementary strategies for obtaining data should be considered.

Even when there are no restrictions on the use of real data, there are technical challenges to capturing and storing data streams since the data's high volume, and speed requires a specific mechanism for this purpose. In this sense, researchers propose solutions based on multiple services to efficiently collect and store massive data stream (Malensek et al., 2011; Alshamrani et al., 2020; Lv et al., 2021). In the formal literature, no works explicitly address using real historical data for testing DSP applications. However, in recent years, several works have been published proposing techniques for anonymizing real data to bypass privacy issues while mitigating the data's loss of usefulness (Begum and Nausheen, 2018; Kenthapadi and Tran, 2018; Madan and Goswami, 2018; Majeed and Lee, 2020; Sharma et al., 2020; Hossayni et al., 2021; Ramya Shree et al., 2022; Vasa and Thakkar, 2022).

Production Data Mirroring. We found the description of the mirroring production data to a testing infrastructure to test and evaluate a new version of the software. Our previous study (Vianna et al., 2019) also reported this approach in interviews. In [Source 89], an approach is described where an application is executed in "shadow mode" to compare the outputs of two software versions executed simultaneously. This approach is helpful for testing without compromising data privacy, as an automatic comparison engine ensures that sensitive data remains secure. This method provides an alternative to using real data for testing purposes while maintaining the confidentiality of the data. We observed that this type of test could be very time-consuming as there is no possibility of speeding up the execution, and it also incurs costs associated with building and maintaining a replica of the production infrastructure.

In the formal literature, we relate this approach to differential testing, in which the oracle is the consistency between the outputs of executions of comparable systems (McKeeman, 1998; Gulzar et al., 2019). Although old, the approach has been adapted and used in the context of modern software. For example, the work by Godefroid et al. (2020) employed it for regression testing of REST API services in a cloud environment. A similar approach has been adopted in the industrial context, known as A/B testing. Large e-commerce companies like Netflix and Amazon apply it to evaluate the performance of changes in business variables. This approach redirects some users to a variant version of the e-commerce site to measure the sales conversion rate caused by specific changes (Koukouvis et al., 2016; Saleem et al., 2019; Rahutomo et al., 2020; Wingerath et al., 2022).

In the background (Section 2.2), we describe the work of Kallas et al. (2020), which precisely brings the differential approach to testing DSP applications. At this point, we identify a correspondence between the efforts of academia and industry. Although several reports of employing this approach for diverse testing purposes, none mention its use as a solution to perform tests with real data while maintaining data privacy as described in the grey sources.

Semi-synthetic data. The semi-synthetic data approach, also called hybrid data, consists of the automatic generation of test data based on real data. In the grey literature, it was reported to generate a telephony data model from real data to create realistic fake data. Although we collected only one practitioner report, there are several works in this direction in the formal literature. Semi-synthetic data generation has been employed targeting different application contexts, for example, smart grid (Lahariya et al., 2020) and health care data (Wang et al., 2021). In general terms, this approach extracts statistical properties from the data and establishes a model of the data pattern in order to feed automatic generators (Li et al., 2016; Popić et al., 2019; Tan et al., 2019; Behjati et al., 2019; Manco et al., 2022).

For example, we bring the research by Tan et al. (2019), which investigates machine learning techniques to generate synthetic, dynamic, and representative test data. The proposed approach is based on building a statistically representative model of a real (reference) population (Tan et al., 2019). The work considers that Recurrent Neural Networks (RNNs) and Generative Adversarial Networks (GANs) techniques are suitable for synthetic data generation since these neural networks can learn the statistical properties of data sets. The authors emphasize this approach's ability to maintain data privacy since the work was motivated to provide a solution for sharing test data between institutions without compromising sensitive data. Although the technique has been pointed out as a way to preserve data privacy, this statement is not valid in some business contexts where the data distribution pattern is sensitive information.

Specifically promoting using semi-synthetic data to test DSP applications, Grulich et al. (2019) proposed an open-source out-of-order data stream generator. The tool takes a real data stream as input, including event timestamps, and simulates arbitrary fractions of out-of-order tuples and their respective delays (Grulich et al., 2019). The generation process considers histograms and statistics on the temporal distribution of events. This approach is an example of modifying real data to meet a specific testing purpose, in which case the purpose was to test out-of-order events. Kim et al. (2018) provide the method that generates data for performance testing using the frequency distribution of the real data streams. Komorniczak et al. (2022) also reported employing semi-synthetic streams generated based on real-world data to evaluate concept drift detectors. Although the focus is slightly different, the concept drift is within the scope of machine learning applications in DSP, and the testing of detectors needs stream data. The evaluation considered synthetic and semi-synthetic data, as the difference in effectiveness between these data sets was a variable to be controlled.

Considering all the works cited, using semi-synthetic data seems to be a promising direction to address privacy issues when testing DSP applications. Additionally, we emphasize that such a technique can only be adopted when real data is accessible to generate these models.

Synthetic data. Synthetic data generation has been reported along with various tools to support this strategy. Generally, synthetic data generators take as input a formal description of the data schema to generate test data. Although the data is random initially and only follows the schema format specification, practitioners have also reported building more elaborate scripts and implementing rules to generate more representative data. For example, these scripts enable the developer to assign weights to the proportion of data generated for each field. It was also reported that some tools, such as Kinesis Data Generator, provide specific features to configure temporal aspects, thus making it possible to define data periodicity through statistical distribution functions with user-configurable parameters.

In the formal literature, Popić et al. (2019) state that the primary motivation for adopting this strategy is the unavailability of real data for privacy reasons, and on the other hand, the author points out the low ability to generate realistic test data as the weakness of the strategy. In this sense, researchers have directed efforts to develop techniques to improve test data generation. As the grey literature reports, formal research also investigates using statistical resources to improve the generated data (Mannino and Abouzied, 2019). For the same purpose, there are works exploring information extracted from UML diagrams (Jaffari et al., 2020; Lafi et al., 2021) and the application's symbolic execution (Ye and Lu, 2021).

In line with approaches based on statistical models, (Mannino and Abouzied, 2019) proposed the Synner, a synthetic data generator based on specifying the statistical properties of data to generate real-looking synthetic data. With this tool, the developer specifies statistical properties through a data generation language or rich visual resources. Data samples, histograms, and basic statistics, such as mean and standard deviation, are constantly updated in a spreadsheet view as the user interacts with the interface. The tool makes it possible to generate random data or data fitting well-known or custom statistical distributions. In addition, it also provides a mechanism to add noise and statistical properties to the relationship between fields.

Another path reported in the formal literature is analyzing information from UML diagrams to complement and guide the generation of test cases. For example, the approach by Jaffari et al. (2020) used the activity diagram to generate more representative test data, while Lafi et al. (2021) employed natural language processing to extract information from the use case description to guide data generation. The analysis of modeling documents is an interesting method to be investigated, as such diagrams describe aspects like time, concurrency, and state characteristics, which are relevant for triggering user-defined events in DSP application tests.

Looking at the data stream context, Ye and Minyan proposed SPOT, a test data generator based on symbolic execution for DSP applications. This approach generates test data by solving path conditions obtained by mapping test inputs during execution and thus identifying the respective execution path (Ye and Lu, 2021). SPOT also supports temporal issues, such as the interval between data and the data order variation by iterative algorithms. The promoted evaluation aimed to compare the SPOT symbolic execution approach with the property-based testing approach implemented by the DiffStream (Kallas et al., 2020) and FlinkCheck (Espinosa et al., 2019) tools (both tools were discussed in the Background Section 2.2). In summary, the evaluation showed that SPOT test data has benefits in triggering program failure more efficiently and with better path coverage than property-based solutions. The approach proposed by Ye and Minyan has shown promise in generating higher-quality synthetic datasets for testing DSP applications.

Focusing on machine learning in the DSP context, Iglesias et al. (2020) proposed a synthetic data generator named MDCStream. The tool generates temporal-dependent numerical datasets to stress-test stream data classification, clustering, and outlier detection algorithms. MDCStream provides functionalities to create challenges related to non-stationarity, concept drift, and algorithm adaptiveness. The tool allows the developer to configure several cluster-specific characteristics of the generated data, such as the range of dimensions, the number of elements, shape, orientation, statistical distributions, dependencies, feature correlations in the cluster, and cluster rotations. MDCStream is a practical alternative for generating test data for machine learning algorithms that process data streams and is also helpful for discovering new classes, clusters, and anomalies.

Despite being considered an automatic and first-hand approach as a quick alternative to generating test data, a manual effort to customize the data generation cannot be ignored. The effort may involve configuring various generator parameters and specifying the data's statistical distributions. Furthermore, generating good test data depends on the professional's knowledge of statistical distributions and complex data patterns to exercise certain test target functionality.

6.1.5. RQ5. What are the tools and under what circumstances are they used in the context of DSP application testing?

This section briefly discusses the tools for testing DSP applications extracted from the grey literature. We identified 50 tools, of which 26 were developed for the DSP context, while the other

Table 16
Tools, mains uses and associated challenges.

Main uses of tools in testing DSP applications & Associated challenge	Tool Reference
Testing Utilities provide various utility resources to support testing activities, such as integration with other testing tools, command-line interfaces, domain-specific test specification languages, test annotations, and scripts for testing automation in CI environments. Test utilities help to address the challenge *The complexity of DSP Applications* from Section 5.3.1, as they bring functionality to deal with DSP characteristics.	ScalaTest (Community, 2021b), Jackdaw Test Machine (Funding Circle, 2021), Ducktape (Confluent Inc., 2021c), Kafka Streams Testing Utilities (Apache Foundation, 2021c), Spark Testing Base (Karau, 2021), Embedded Kafka Cluster (Community, 2021a), Flink Spector (ottogroup, 2019), Flink Test Utils (Apache Foundation, 2021a), Kcat Magnus Edenhill - Apache (2021), and Kafka Datagen Connector (Confluent Inc., 2021d).
Infrastructure automation supports the management of complex test infrastructures, including automation features such as code deployment, network configuration, cloud management, infrastructure as code, command line interfaces, and remote management. In the context of DSP testing, these tools can be used to build CI environments, enable automated test execution, and help developers address the challenge *Testing Infrastructure Complexity* presented in Section 5.3.2.	Jenkins (Jenkins, 2021), Terraform (HashiCorp, 2021), Ansible (Michael DeHaan and Red Hat Inc., 2021) and Confluent CLI (Confluent Inc., 2021b).
Data Generation supports on generating test data and provides features such as the generation of data based on the schema registry, the configuration of timestamps, shaping the fake data distribution into real data or custom distribution, and DSL for specifying data properties. These tools help practitioners to deal with the challenge *Obtaining Test Data* discussed in Section 5.3.4.	Ksql-datagen (Confluent Inc., 2017), Kafka Datagen Connector (Confluent Inc., 2021d), StreamData (Leopardi and Valim, 2021), Avro Random Generator (Confluent Inc., 2021a), Mockaroo (Mockaroo LLC, 2021), Ecto Stream Factory (Barchenkov, 2019), Faker.js (Java Script) (Marak, 2021), Faker (Python) (Faraglia, 2021), and Kinesis Data Generator (Amazon Inc., 2021).
Time issues provide specific functionality to deal with time issues, such as controlling the current time of the function under test. Such tools help practitioners to deal with the challenge *Time Issues* identified in Section 5.3.3.	Awaitility (Johan Haleby and Community, 2021) (library that facilitates asynchronous testing by a DSL to express test expectations), Flink Test Utils (Apache Foundation, 2021a), Kafka Streams Testing Utilities (Apache Foundation, 2021c) and Spark Testing Base (Karau, 2021).

24 are tools for other contexts but also used in some DSP testing activity. For each tool, we looked in the official documentation for additional information about license, authorship, functionalities, and purpose of use, and then we made notes about this information.

We observed that of the 50 tools listed, only four are proprietary, while the others are some variations of open-source licenses. In part, the predominance of open-source software can be explained by adopting a business model around open-source software, as reported by August et al. (2021) and Shahrivar et al. (2018). In this model, companies contribute to developing open-source tools with the community while commercializing software customizations, training, certifications, consulting, support, and infrastructure services. The companies Data Brics and Confluent are examples of this business model, as they provide services centered on open-source DSP frameworks such as Apache Spark and Apache Kafka. Even the case of Confluent is explicitly mentioned in the work of August et al. (2021), and it is categorized as an open-source distributor business model by (Riehle, 2021). This model encourages the community to participate in developing and improving various tools around these technologies. The Embedded Kafka tool is an example of a tool with hundreds of community contributions.

Another aspect observed is the participation of large companies versus individuals in developing and maintaining the tools. The work by Suhada et al. (2021) suggests that companies' motivation to participate in open-source projects comes from the strategy based on Open Innovation to capture new external ideas and solutions. At the same time, individual contributors seek personal and career development. The selected tools with large companies behind the creation or maintenance are JMeter (Apache Foundation, 2021b) from Apache Foundation, Flink Spector (ottogroup, 2019) from Ottogroup, Fluent Kafka Streams (Backdata, 2021) Tests from Backdata, Kinesis Data Generator (Amazon Inc., 2021) from Amazon Incorporation, and Ksql-datagen (Confluent Inc., 2021d) from Confluent Incorporation. Alternatively, there are also contributions from individuals who initially create the tools and later make them open-source projects to be maintained by the community and supporting companies. Examples of individual contributions include Kafka for Junit (Günther, 2021) by Markus Günther, Spark Testing Base by (Karau, 2021), Awaitility Library (Johan Haleby and Community, 2021) by Johan Haleby, Ansible (Michael DeHaan and Red Hat Inc., 2021) by Michael DeHaan, and Sangrenel (Alquiza, 2021) by Jamie Alquiza.

In order to identify the use of the tools in the context of testing DSP applications, we analyzed the practitioners' comments extracted from the grey literature and the official documentation regarding the features. Following, the Table 16 bring the tools grouped according to their main uses in testing DSP applications, and then we associate them with the challenges raised by RQ1.

6.2. Implications for research and practice

This work benefits researchers and practitioners by summarizing knowledge fragmented in various grey sources in a single document and including related scientific papers to complement or confront findings. In general, this work's main contribution is narrowing the gap between academia and the industry. Throughout the discussion, we have included observations considering implications for practice and research. The following subsections summarize the most relevant implications.

6.2.1. Implications for practice

This document provides helpful information for practitioners to guide decision-making concerning designing and implementing tests in Data Stream Processing applications. The critical information for practitioners is the list of testing purposes, challenges, approaches, tools, and strategies for obtaining test data. We have included throughout the discussion valuable notes for practitioners, such as advantages, disadvantages, and specific details regarding the techniques and tools presented. We also associate the tools, and testing approaches with the testing objectives and challenges, which helps the practitioner consider using the presented techniques and tools.

Additionally, this document includes complementary content extracted from the academic literature (usually not consulted by practitioners), such as new testing approaches and tools created in the academic context. For example, regarding test data generation, the formal literature contributes with approaches based on machine learning to improve the quality of generated testing data and preserve data privacy. The cited formal literature also introduces the mutation testing technique to increase the

effectiveness of test suites. All of the selected content, which is related to each other, commented on, and linked to the formal literature, has resulted in a robust and reliable document for practitioners. In summary, practitioners have a starting point of study covering many aspects related to DSP application testing in this document.

6.2.2. Implications for research

From the research point of view, the work contributes by selecting and summarizing the intrinsic practice content fragmented into several informal documents. Such information is now part of the academic documentation and is available for future research works. Furthermore, we identified research studies corroborating with practices in the industry. For example, when discussing synthetic data generation, we reported grey literature and formal literature efforts on using statistical resources to improve the generated data.

We have also identified research-based techniques that can be evaluated for use in the testing of DSP. For instance, analyzing modeling documents can help improve test data generation quality since these diagrams describe aspects such as time, concurrency, and state characteristics. On the other hand, we identified in the grey literature approaches not yet explored in the formal literature. For example, no papers cover equivalent testing techniques based on production data stream mirroring, as reported in Section 5.6.2. The approach resembles differential testing with production data but preserves data privacy by an automatic conferencing mechanism called shadow mode.

Finally, we highlight two promising issues for future research that lack formal literature. The first is data privacy in testing activities since it is a topic of increasing importance these days. Although there are several proposed solutions, the problem has several facets to be explored. The second is fault tolerance testing approaches since these DSP applications should always keep running and recover from runtime failures automatically. In particular, chaos engineering is a testing approach that can be further explored for this purpose. We also emphasize the need for research involving experiments to validate the effectiveness of testing approaches in the context of DSP applications. These studies are essential for systematizing testing processes and consolidating the testing techniques reported in this paper.

6.3. Threats to validity

Although we have conducted our research based on consolidated guidelines for GLR and followed expert recommendations, this study is still subject to validity threats typical of this chosen method. For example, limitations in searching for grey literature and subjectivity and inaccuracies when selecting sources, extracting data, and analyzing results. Since the study preparation, we have been concerned with these possible threats to validity, which we have systematically identified throughout the process and promoted mitigation strategies. We organized our threats according to the classification schema proposed by Ampatzoglou and colleagues (Ampatzoglou et al., 2020): Study Selection Validity, Data Validity, and Research Validity.

Study Selection Validity.

This category includes threats to the validity of the search process and studies filtering steps. For the search process, we adopted a general-purpose search engine, but the criteria for selecting and ranking results from these tools are unclear and may yield biased or irrelevant results. We seek to mitigate this issue by conducting targeted searches on domains popularly known to contain technical content produced by practitioners, such as Stack Overflow, Medium, and LinkedIn Pulse. In addition, to mitigate possible search engine inefficiencies, we performed snowballing, which added 53 more sources to the review.

At the end of the search process, we identified some content republished in different domains but with slight variations, updates, or increments. While it is common practice for authors to mark them as "reposted" and include the link to the original post, not all do. To identify similar textual content, we adopted the tool WCopyfind,[8] which identified 15 cases of duplicate content.

In the source selection phase, threats are related to elaborating and applying inclusion and exclusion criteria. For example, conflicting or very generic criteria can hamper the selection process. To mitigate these issues, we built an initial set of criteria based on the area's literature and then held discussions among authors to refine the criteria. To minimize the chances of participants misunderstanding the criteria, we prepared a document of guidelines and examples and conducted a training session and a pilot study. Only after aligning the divergences did the selection process begin. Finally, each source from the initial set was analyzed by two reviewers independently. In case of disagreement, a third reviewer participated in conflict resolution by discussion or a tie-breaking vote.

Data Validity.

This category addresses threats to the validity of the data extraction and synthesis phase. The choice of variables to be extracted may influence the subsequent study phases. To mitigate this threat, we built a set of categories based on terms associated with the RQs and the categories adopted in previous studies. In addition, throughout the extraction, we promoted the refinement and inclusion of newly emerged categories. To minimize interpretation bias in the extraction process, we conducted a pilot study, pre-extraction meetings to align the understanding of the categories, and discussions during extraction to resolve conflicts. Also, the extraction process was paired, in which two authors performed the extraction and mutually reviewed each other's work.

Finally, the first author performed the analysis, and to minimize analysis bias, periodic discussions were held between all authors. Besides, to support the analysis, we considered quantitative data from the categories to identify the most relevant topics and searched the academic literature to confront or confirm the findings.

Research Validity.

Choosing the wrong methodology is a threat to the study as a whole, so we selected ours in consultation with experimental software engineering experts and confirmed that the method is suitable with GLR adoption checklists from Garousi's guidelines. Once the methodology was chosen, we invited a specialist in Grey Literature to participate in the research by supporting the application of the method. Inadequately formulated or non-comprehensive RQs can bias the research or make it irrelevant. So, the goals of this article were based on topics identified in a previous study we conducted (Vianna et al., 2019) based on grounded theory, which included data collected with practitioners through a survey (101 participants) and 12 interviews. In the current work, we promoted discussions among authors to define the RQs. Although we selected more than 154 sources, the results may not be generalizable as they are based on a fraction of the existing content. To mitigate this threat, we reinforced our findings by comparing them with the literature related to each find. Study non-repeatability is a well-known threat; therefore, we have documented the protocol employed and described each step of the process in the article. Documents and data necessary to reproduce the study are available online (Vianna et al., 2022).

[8] https://plagiarism.bloomfieldmedia.com/software/wcopyfind/

7. Conclusions and future work

This work promoted research into practitioners' knowledge of DSP application testing by addressing grey literature in the field. The GLR selected 154 from the initial 1667 sources, and then the content of interest was extracted by coding technique in order to obtain answers to the RQs.

Below we list the main results obtained according to the RQs. **RQ1** The challenges for DSP application testing: (1) The complexity of DSP Applications, (2) Testing Infrastructure Complexity, (3) Time issues, and (4) Obtaining Test Data. **RQ2** The main testing purposes identified: (1) Functional suitability, (2) Performance efficiency, (3) Reliability, and (4) Maintainability. **RQ3** The main test approaches reported: (1) Performance Test, (2) Regression Test, (3) Property-based test, (4) Chaos test, and (5) Contract/Schema Testing. **RQ4** The strategies adopted by practitioners to obtain test data: (1) Historical Data, (2) Production Data Mirroring, (3) Semi-synthetic data, and (4) Synthetic data. **RQ5** We reported 50 tools used in various testing activities, which are used for: the automation of test infrastructure, test data generation, test utilities, time issues, load generations, mocking, and others.

In the discussion, we incorporated observations regarding the findings of the grey literature while correlating them with academic articles to either complement, confirm or confront them. In short, the review of grey literature indicated that the industry has advanced in the practice of testing DSP applications and has proposed techniques and tools to meet its requirements. We even reported approaches and tools developed by the industry but not reported in the formal literature that could be the subject of research investigation. At the same time, we identified that academia has also been promoting advances in the subject. Some academic contributions are aligned with what has been developed by practitioners in the industry. In contrast, others bring complementary advances that can benefit the industry.

Finally, this work results from applying proper research methodologies combined with the effort to search, select and analyze content scattered in many informal documents produced by practitioners and published on the internet. The main contribution is the summarization of various knowledge related to the testing of DSP applications, which is made available in a systematically organized document for reference by industry practitioners and researchers in the field.

Future Work. Our study suggested directions for future work, which could be explored by researchers in the area, such as techniques for generating high-quality test data, fault tolerance testing methods, strategies for preserving data privacy during tests, and ways to lower testing costs. Specifically, we aim to use the structured knowledge gathered in this article to develop a guideline for testing DSP applications. The proposed guideline begins with the assumption that each project has unique characteristics to consider during the design and development of tests; for example, the application context guides which test objectives and quality parameters to focus on during the tests. We also highlight the significance of time, infrastructure, financial, and professional resources as key factors in the decision-making process of the test project. Consequently, selecting testing methods, tools, and strategies to acquire test data relies on contextual criteria within a decision structure. In summary, these guidelines will assist practitioners by offering a roadmap of options to guide them in designing and implementing DSP application testing activities in diverse situations.

CRediT authorship contribution statement

Alexandre Vianna: Conceptualization, Investigation, Data curation, Writing – original draft, Project administration, Visualization. **Fernando Kenji Kamei:** Methodology, Validation. **Kiev Gama:** Writing – review & editing, Supervision, Validation. **Carlos Zimmerle:** Validation, Writing – review & editing. **João Alexandre Neto:** Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The research data is stored in an appropriate repository with DOI.

GLR research data (Original data) (Figshare)

Acknowledgments

We thank the anonymous reviewers whose comments and suggestions helped improve this manuscript. This work is partially supported by INES, Brazil (www.ines.org.br), CNPq, Brazil grant 465614/2014-0, CAPES, Brazil grant 88887.136410/2017-00, and FACEPE, Brazil grants APQ-0399-1.03/17 and PRONEX, Brazil APQ/0388-1.03 /14.

Appendix A. The selected sources

S1 Alexandre Vicenzi; "Test and document your data pipeline" https://www.alexandrevicenzi.com/posts/test-and-document-your-data-pipeline PDF File QA Score: 0.35

S2 Artillery; "Load Testing AWS Kinesis With Artillery" https://artillery.io/blog/load-testing-aws-kinesis PDF File QA Score: 0.64

S3 Allan MacInnis, Jared Warren, Amazon; "Test Your Streaming Data Solution with the New Amazon Kinesis Data Generator" https://aws.amazon.com/blogs/big-data/test-your-streaming-data-solution-with-the-new-amazon-kinesis-data-generator PDF File QA Score: 0.73

S4 Apache Foundation; "Apache Flink 1.11 Documentation: Testing" https://ci.apache.org/projects/flink/flink-docs-stable/dev/stream/testing.html PDF File QA Score: 0.86

S5 Amazon; "Testing Your Delivery Stream Using Sample Data" https://docs.aws.amazon.com/firehose/latest/dev/test-drive-firehose.html PDF File QA Score: 0.61

S6 Cloudera; "Test and validation" https://docs.cloudera.com/csa/1.4.0/development/topics/csa-test-validation.html PDF File QA Score: 0.60

S7 Microsoft Corporation; "Test an Azure Stream Analytics job with sample data" https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-test-query PDF File QA Score: 0.66

S8 Aeldung; "Introduction to Apache Flink with Java" https://www.baeldung.com/apache-flink PDF File QA Score: 0.71

S9 Confluent; "Generate Custom Test Data by Using the ksql-datagen Tool" https://github.com/confluentinc/kafka-connect-datagen PDF File QA Score: 0.89

S10 Dell EMC; "Confluent Kafka Performance Characterization" https://infohub.delltechnologies.com/static/media/48e14554-a272-4cb0-8996-80ac6dd016b8.pdf PDF File QA Score: 0.56

S11 Javier Ramos, ITNEXT; "Big Data Quality Assurance. Introduction" https://medium.com/@javier.ramos1/big-data-quality-assurance-635c368a3e28 PDF File QA Score: 0.78

S12 Wei Huang; "Build a Real-time Data Pipeline during the weekend in Go-Part 1" https://medium.com/@jayhuang75/build-a-real-time-data-pipeline-during-the-weekend-in-go-30f9c63e207a PDF File QA Score: 0.59

S13 Wei Huang; "Build a Real-time Data Pipeline during the weekend in Go-Part 3" https://medium.com/@jayhuang75/build-a-real-time-data-pipeline-during-the-weekend-in-go-part-3-3aa944d10caf PDF File QA Score: 0.58
S14 Michael Gendy; "The Power of Quality Assurance — Designing Robust QA Processes for SQL Data Analysis Pipelines" https://medium.com/@michaelgendy/the-power-of-quality-assurance-designing-robust-qa-processes-for-sql-data-analysis-pipelines-2b85e9a3928a PDF File QA Score: 0.42
S15 Ashish Mrig; "Design & Strategies for Building Big Data Pipelines" https://medium.com/@mrashish/design-strategies-for-building-big-data-pipelines-4c11affd47f3 PDF File QA Score: 0.67
S16 Milan Sahu; "Data Quality testing with AWS Deequ" https://medium.com/@sahu.milan1988/data-quality-testing-with-aws-deequ-53b6f765de08 PDF File QA Score: 0.38
S17 Vivek N; "Quality Engineering our Data Pipelines" https://medium.com/@vivekn.19/quality-engineering-our-data-pipelines-fdcb379dc951 PDF File QA Score: 0.50
S18 Zachary Ennenga, Airbnb Engineering & Data Science; "Scaling a Mature Data Pipeline — Managing Overhead" https://medium.com/airbnb-engineering/scaling-a-mature-data-pipeline-managing-overhead-f34835cbc866 PDF File QA Score: 0.65
S19 Irina Pashkova, GreenM; "Data Warehouse Testing. It's rare luck when you are given a ..." https://medium.com/greenm/data-warehouse-testing-3c0fb955da1d PDF File QA Score: 0.52
S20 Frank Dekervel, Kapernikov; "Writing a high quality data pipeline for master data with Apache Spark — Part 3" https://medium.com/kapernikov/writing-a-high-quality-data-pipeline-for-master-data-with-apache-spark-part-3-9d12dbaf0822 PDF File QA Score: 0.50
S21 Talha Malik, The Startup; "Structuring a Robust Data Pipeline" https://medium.com/swlh/structuring-a-robust-data-pipeline-24ff67783782 PDF File QA Score: 0.61
S22 Nikita Zhevnitskiy; "Testing Kafka based applications". https://medium.com/test-kafka-based-applications/https-medium-com-testing-kafka-based-applications-85d8951cec43 PDF File QA Score: 0.62
S23 Pavan Kumar S, TestVagrant; "Key Points To Remember While Testing Data Streams" https://medium.com/testvagrant/what-is-data-streaming-and-key-points-to-know-before-testing-it-7a16fe861e05 PDF File QA Score: 0.35
S24 Anton Bakalets; "Flink Job Unit Testing. Write a unit test ensuring your Flink..." https://medium.com/@anton.bakalets/flink-job-unit-testing-df4f618d07a6 PDF File QA Score: 0.45
S25 Scott Haines; "Testing Spark Structured Streaming Applications:" https://medium.com/@newfrontcreative/testing-spark-structured-streaming-applications-like-a-boss-95a1b261cd35 PDF File QA Score: 0.69
S26 Jacobus Herman, Big Data Republic; "AWS Kinesis Data Analytics: a cautionary review" https://medium.com/bigdatarepublic/kinesis-data-analytics-sql-a-cautionary-review-fb9ddd06e5d9 PDF File QA Score: 0.66
S27 Nikita Gupta, Henrique Ribeiro Rezende, Coding Stories; "How to test Kinesis with LocalStack" https://medium.com/coding-stories/how-to-test-kinesis-with-localstack-356b55c69e2 PDF File QA Score: 0.51
S28 Alibaba Cloud, DataSeries, Digoal; "Using PostgreSQL for Real-Time IoT Stream Processing Applications" https://medium.com/dataseries/using-postgresql-for-real-time-iot-stream-processing-applications-965741c57315 PDF File QA Score: 0.53
S29 Alibaba Tech, HackerNoon.com; "From Code Quality to Integration: Optimizing Alibaba's Blink Testing Framework" https://medium.com/hackernoon/from-code-quality-to-integration-optimizing-alibabas-blink-testing-framework-dc9c357319de PDF File QA Score: 0.66
S30 Brandon Stanley, Slalom Data & Analytics; "Amazon Kinesis Data Streams: Auto-scaling the number of shards" https://medium.com/slalom-data-analytics/amazon-kinesis-data-streams-auto-scaling-the-number-of-shards-105dc967bed5 PDF File QA Score: 0.67
S31 Shash, Andy LoPresto, daggett; "Faster way of developing and Testing new Nifi Processor" https://stackoverflow.com/questions/44748548/faster-way-of-developing-and-testing-new-nifi-processor PDF File QA Score: 0.61
S32 zavalit, Diego Reico; "apache flink - Failing to trigger StreamingMultipleProgramsTestBase Test in Scala" https://stackoverflow.com/questions/49155762/failing-to-trigger-streaming multipleprogramstestbase-test-in-scala
PDF File QA Score: 0.39
S33 Salvador Vigo, AbhishekN; "Java - Generate fake" stream data. Kafka - Flink"" https://stackoverflow.com/questions/51934554/generate-fake-stream-data-kafka-flink PDF File QA Score: 0.31
S34 user3139545, David Anderson; "Java - Is there a notion of virtual time in Apache Flink tests like there is in Reactor and RxJava" https://stackoverflow.com/questions/54855358/is-there-a-notion-of-virtual-time-in-apache-flink-tests-like-there-is-in-reactor PDF File QA Score: 0.55
S35 Tilak, Oleg Zhurakousky; "Spring Cloud Stream - Integration testing" https://stackoverflow.com/questions/55739806/spring-cloud-stream-integration-testing PDF File QA Score: 0.59
S36 jerrypeng, Valdon Mesh (KIC); "Java - Test apache pulsar functions in an embedded standalone environment" https://stackoverflow.com/questions/56515083/test-apache-pulsar-functions-in-an-embedded-standalone-environment PDF File QA Score: 0.34
S37 SunilS, Andy LoPresto; "How Can Apache NiFi Flow Be Tested?" https://stackoverflow.com/questions/57062240/how-can-apache-nifi-flow-be-tested PDF File QA Score: 0.56
S38 Pavan, Chesnay Schepler; "Java - Unit-testing flink application with streaming data" https://stackoverflow.com/questions/40845683/unit-testing-flink-application-with-streaming-data PDF File QA Score: 0.35
S39 Till Rohrmann, Mike; "Junit - How to stop a flink streaming job from program" https://stackoverflow.com/questions/44441153/how-to-stop-a-flink-streaming-job-from-program PDF File QA Score: 0.58
S40 Timo Walther, user8298342; "Scala - mock object for flinks DataStream" https://stackoverflow.com/questions/45067426/mock-object-for-flinks-datastream PDF File QA Score: 0.41
S41 Diego Reiriz Cores, Yohei Kishimoto, UberHans; "Java - Why is Apache Flink droping the event from datastream?" https://stackoverflow.com/questions/49278579/why-is-apache-flink-droping-the-event-from-datastream PDF File QA Score: 0.47
S42 Piotr Nowojski, William Speirs; "Unit Testing Flink Streams with Multiple Event Streams" https://stackoverflow.com/questions/50454231/unit-testing-flink-streams-with-multiple-event-streams PDF File QA Score: 0.53
S43 David Anderson, Richard Deurwaarder; "Apache Flink - End to End testing how to terminate input source" https://stackoverflow.com/questions/51242886/apache-flink-end-to-end-testing-how-to-terminate-input-source PDF File QA Score: 0.61
S44 Felder, David Anderson; "Testing Flink window" https://stackoverflow.com/questions/60418113/testing-flink-window PDF File QA Score: 0.55
S45 Felder, David Anderson; "Testing Flink with embedded Kafka" https://stackoverflow.com/questions/60476733/testing-flink-with-embedded-kafka PDF File QA Score: 0.50
S46 YRQ; "Junit - How to test a Datastream with jsonobject in Apache Flink" https://stackoverflow.com/questions/61866226/how-to-test-a-datastream-with-jsonobject-in-apache-flink PDF File QA Score: 0.34

S47 user13906258, Yeis Gallegos; "Junit - How to test keyed-broadcastprocessfunction in flink?" https://stackoverflow.com/questions/62919920/how-to-test-keyedbroadcastprocessfunction-in-flink PDF File QA Score: 0.41

S48 Dennis Layton; "DataOps: Building Trust in Data through Automated Testing" https://www.linkedin.com/pulse/dataops-building-trust-data-through-automated-testing-dennis-layton PDF File QA Score: 0.41

S49 Gopinath Mandala; "Applications are going Serverless. How will QA respond?" https://www.linkedin.com/pulse/applications-going-serverless-how-qa-respond-gopinath-mandala PDF File QA Score: 0.28

S50 Arseniy Tashoyan; "Developing Event-Driven Applications to Prevent Accidents" https://www.linkedin.com/pulse/developing-event-driven-applications-prevent-arseniy-tashoyan PDF File QA Score: 0.63

S51 Shachar Bar; "A million-to-one shot, Doc, million-to-one" https://www.linkedin.com/pulse/million-to-one-shot-doc-shachar-bar-berezniski- PDF File QA Score: 0.37

S52 Vijayendra Yadav, Niels Basjes, Arvid Heise, David Anderson; "Apache Flink User Mailing List archive. - [Flink Unit Tests] Unit test for Flink streaming codes" http://apache-flink-user-mailing-list-archive.2336050.n4.nabble.com/Flink-Unit-Tests-Unit-test-for-Flink-streaming-codes-td37119.html PDF File QA Score: 0.56

S53 Marcin Kuthan; "Spark and Spark Streaming Unit Testing - Passionate Developer" http://mkuthan.github.io/blog/2015/03/01/spark-unit-testing PDF File QA Score: 0.73

S54 Bartosz Gajda; "Unit Testing Apache Spark Structured Streaming using MemoryStream - Bartosz Gajda -" https://bartoszgajda.com/2020/04/13/testing-spark-structured-streaming-using-memorystream PDF File QA Score: 0.43

S55 Raphael Brugier, Ippon Technologies; "Testing strategy for Spark Streaming - Part 2 of 2" https://blog.ippon.tech/testing-strategy-for-spark-streaming PDF File QA Score: 0.73

S56 Anuj Saxena, Knoldus; "Spark Streaming: Unit Testing DStreams" https://blog.knoldus.com/spark-streaming-unit-testing-dstreams PDF File QA Score: 0.58

S57 Dipin Hora; "Performance testing a low-latency stream processing system" https://blog.wallaroolabs.com/2018/03/performance-testing-a-low-latency-stream-processing-system PDF File QA Score: 0.78

S58 Cloudflow; "Testing a Flink Streamlet" https://cloudflow.io/docs/dev/develop/test-flink-streamlet.html PDF File QA Score: 0.52

S59 Felipe Fernández, Codurance; "Testing Spark Streaming: Unit testing" https://codurance.com/2016/08/02/testing-spark-streaming-unit-testing PDF File QA Score: 0.74

S60 Kartik Khare; "Apache Flink: A Guide for Unit Testing in Apache Flink" https://flink.apache.org/news/2020/02/07/a-guide-for-unit-testing-in-apache-flink.html PDF File QA Score: 0.78

S61 Apache Foundation; "GitHub - Flink End to End Tests" https://github.com/apache/flink/tree/master/flink-end-to-end-tests PDF File QA Score: 0.54

S62 JAppsConsultants; "GitHub - Hello Kafka Stream Testing: The most simple way to test Kafka based applications or micro-services e.g. Read/Write during HBase/Hadoop or other Data Ingestion Pipe Lines" https://github.com/authorjapps/hello-kafka-stream-testing PDF File QA Score: 0.56

S63 Amazon; "GitHub - Amazon Kinesis Data Generator: A UI that simplifies testing with Amazon Kinesis Streams and Firehose. Create and save record templates, and easily send data to Amazon Kinesis". https://github.com/awslabs/amazon-kinesis-data-generator PDF File QA Score: 0.68

S64 Igor Barchenkov; "GitHub - Ecto Stream Factory: Generate test data for regular and property-based tests and seed your database" https://github.com/ibarchenkov/ecto_stream_factory PDF File QA Score: 0.38

S65 Jendrik Poloczek; "GitHub - Mockedstreams: Scala DSL for Unit-Testing Processing Topologies in Kafka Streams" https://github.com/jpzk/mockedstreams PDF File QA Score: 0.66

S66 Ottogroup; "GitHub - Flink Spector: Framework for Apache Flink unit tests" https://github.com/ottogroup/flink-spector PDF File QA Score: 0.70

S67 Google Cloud Platform; "Testing the Flink Operator with Apache Kafka" https://googlecloudplatform.github.io/flink-on-k8s-operator/docs/kafka_test_guide.html PDF File QA Score: 0.60

S68 Apache Foundation; "Apache Kafka" https://kafka.apache.org/documentation/streams/developer-guide/testing.html PDF File QA Score: 0.65

S69 Filipe Correia, Stephan Ewen, Alexander Kolb; "Unit testing support for flink application?" http://apache-flink-user-mailing-list-archive.2336050.n4.nabble.com/Unit-testing-support-for-flink-application-td4130.html PDF File QA Score: 0.61

S70 Anton Sitkovets; "Testing in Apache Beam Part 2: Stream" https://medium.com/@asitkovets/testing-in-apache-beam-part-2-stream-2a9950ba2bc7 PDF File QA Score: 0.59

S71 Eugene Lopatkin; "Apache Spark Unit Testing Part 3" https://medium.com/@eugene_lopatkin/apache-spark-unit-testing-part-3-streaming-79af91e5d4d1 PDF File QA Score: 0.41

S72 Ahsan Nabi Dar; "Testing Analytics Event Stream. Over the years we have made improvement..." https://medium.com/@hsan_nabi_dar/testing-analytics-event-stream-ba1977b649c6 PDF File QA Score: 0.41

S73 Arvid Heise, Lawrence Benson, Bakdata Data Engineering Blog; "Fluent Kafka Streams Tests. A Java test DSL for Kafka Streams" https://medium.com/bakdata/fluent-kafka-streams-tests-e641785171ec PDF File QA Score: 0.82

S74 Lawrence Benson, Arvid Heise, Bakdata Data Engineering Blog; "Schema Registry mock for Kafka Streams Tests" https://medium.com/bakdata/transparent-schema-registry-for-kafka-streams-6b43a3e7a15c PDF File QA Score: 0.73

S75 ProgrammerSought; "Flink unit test - Programmer Sought" https://programmersought.com/article/53161299873 PDF File QA Score: 0.49

S76 Landon Robinson, Jack Chapa, Databricks, SpotX; "Headaches and Breakthroughs in Building Continuous Applications" https://pt.slideshare.net/databricks/headaches-and-breakthroughs-in-building-continuous-applications PDF File QA Score: 0.56

S77 Seth Wiesman, MediaMath; "Flink Forward San Francisco 2018: Seth Wiesman - Testing Stateful Streaming Applications" https://pt.slideshare.net/FlinkForward/flink-forward-san-francisco-2018-seth-wiesman-testing-stateful-streaming-applications PDF File QA Score: 0.48

S78 Lars Albertsson; "Test strategies for data processing pipelines, v2.0" https://pt.slideshare.net/lallea/test-strategies-for-data-processing-pipelines-v20?next_slideshow=1 PDF File QA Score: 0.59

S79 Johannes Haug; "Pystreamfs" https://pypi.org/project/pystreamfs PDF File QA Score: 0.45

S80 Artem Bilan, Spring; "How to test Spring Cloud Stream applications (Part I)" https://spring.io/blog/2017/10/24/how-to-test-spring-cloud-stream-applications-part-i PDF File QA Score: 0.85

S81 Noodle, Mutt; "Automated testing - Test Strategies for Stream Processing/Event Processing - Software Quality Assurance & Testing Stack Exchange" https://sqa.stackexchange.com/questions/25915/test-strategies-for-stream-processing-event-processing PDF File QA Score: 0.30

S82 Ian Jones, Ran Wasserman, Raj Saxena; "Java - Amazon Kinesis + Integration Tests" https://stackoverflow.com/questions/30777368/amazon-kinesis-integration-tests PDF File QA Score: 0.46

S83 Todd McGrath; "Spark Streaming Testing with Scala Example" https://supergloo.com/spark-streaming/spark-streaming-testing-scala PDF File QA Score: 0.45

S84 Roman Aladev, Blaze Meter; "Apache Kafka - How to Load Test with JMeter" https://www.blazemeter.com/blog/apache-kafka-how-to-load-test-with-jmeter PDF File QA Score: 0.77

S85 Yeva Byzek, Confluent; "Stream Processing Tutorial Part 2: Testing Your Streaming Application" https://www.confluent.io/blog/stream-processing-part-2-testing-your-streaming-application PDF File QA Score: 0.92

S86 Raghunandan Gupta; "Spring Kafka Integration: Unit Testing using Embedded Kafka" https://www.linkedin.com/pulse/spring-kafka-integration-unit-testing-using-embedded-gupta PDF File QA Score: 0.47

S87 Ram Ghadiyaram; "Test data generation using Spark by using simple Json data descriptor with Columns and DataTypes to load in dwh like Hive". https://www.linkedin.com/pulse/test-data-generation-using-spark-simple-json-columns-load-ghadiyaram PDF File QA Score: 0.46

S88 Vidhu Bhatnagar; "Load Testing Our Event Pipeline 2019" https://klaviyo.tech/load-testing-our-event-pipeline-2019-42c984b90aae PDF File QA Score: 0.52

S89 Oren Raboy, Totango Engineering; "Testing Spark Data Processing In Production" https://labs.totango.com/testing-spark-data-processing-in-production-8436e976c43 PDF File QA Score: 0.60

S90 Chip; "Mocking - How can I instantiate a Mock Kafka Topic for junit tests?" https://stackoverflow.com/questions/33748328/how-can-i-instantiate-a-mock-kafka-topic-for-junit-tests/34009141#34009141 PDF File QA Score: 0.39

S91 Luciano Afranllie, CyanogenMod; "Performance - Datastream generator for Apache Kafka" https://stackoverflow.com/questions/41550056/datastream-generator-for-apache-kafka PDF File QA Score: 0.37

S92 imehl, Dmitry Minkovsky; "Testing - Test Kafka Streams topology" https://stackoverflow.com/questions/41825753/test-kafka-streams-topology PDF File QA Score: 0.55

S93 Brandon Barker, Eugene Lopatkin, Matt Powers, Shankar Koirala, Vidya Technology and Training; "Scala - How to write unit tests in Spark 2.0+?" https://stackoverflow.com/questions/43729262/how-to-write-unit-tests-in-spark-2-0 PDF File QA Score: 0.73

S94 Matthias J. Sax, Jaker; "Apache kafka - Testing KafkaStreams applications" https://stackoverflow.com/questions/48599228/testing-kafkastreams-applications PDF File QA Score: 0.68

S95 David O, Matthias J. Sax; "Testing window aggregation with Kafka Streams" https://stackoverflow.com/questions/52643391/testing-window-aggregation-with-kafka-streams PDF File QA Score: 0.64

S96 Ramon Jansen Gomez, Jordan Moore, Matthias J. Sax, Levani Kokhreidze, Vassilis; "Java - How can do Functional tests for Kafka Streams with Avro (schemaRegistry)?" https://stackoverflow.com/questions/52737242/how-can-do-functional-tests-for-kafka-streams-with-avro-schemaregistry PDF File QA Score: 0.62

S97 Val Bonn, Vasyl Sarzhynskyi, Matthias J. Sax; "Java - How to test a Kafka Stream app without duplicating the topology? Use of TopologyTestDriver?" https://stackoverflow.com/questions/52991065/how-to-test-a-kafka-stream-app-without-duplicating-the-topology-use-of-topology PDF File QA Score: 0.58

S98 Kambo, Michael G. Noll, Sarwar Bhuiyan; "Java - Unit testing a kafka topology that's using kstream joins" https://stackoverflow.com/questions/55063686/unit-testing-a-kafka-topology-thats-using-kstream-joins PDF File QA Score: 0.61

S99 Jeahyun Kim, perkss; "Can I test kafka-streams suppress logic?" https://stackoverflow.com/questions/57686563/can-i-test-kafka-streams-suppress-logic PDF File QA Score: 0.46

S100 Ronan Mahony, techburst; "ETL Testing Best Practices". https://techburst.io/etl-testing-best-practices-7594b721ca91 PDF File QA Score: 0.52

S101 Jyoti Dhiman; "How to do load testing of a real-time pipeline?" https://towardsdatascience.com/load-testing-of-a-real-time-pipeline-d32475163285 PDF File QA Score: 0.43

S102 Holden Karau; "GitHub - Spark Testing Base: Base classes to use when writing tests with Spark" https://github.com/holdenk/spark-testing-base PDF File QA Score: 0.74

S103 Raphael Brugier, Ippon Technologies; "Testing strategy for Apache Spark jobs - Part 1 of 2" https://blog.ippon.tech/testing-strategy-apache-spark-jobs PDF File QA Score: 0.73

S104 Andrea Leopardi, Jose Valim; "StreamData — StreamData v0.5.0" https://hexdocs.pm/stream_data/StreamData.html PDF File QA Score: 0.74

S105 Superconductive; "Welcome to Great Expectations! — great_expectations documentation" https://docs.greatexpectations.io/docs/why_use_ge PDF File QA Score: 0.75

S106 Apache Foundation; "Griffin - Streaming Use Cases" http://griffin.apache.org/docs/usecases.html PDF File QA Score: 0.67

S107 Eugene Lopatkin; "Apache Spark Unit Testing Part 2 — Spark SQL" https://medium.com/@eugene_lopatkin/apache-spark-unit-testing-part-2-spark-sql-35c76ed592b0 PDF File QA Score: 0.44

S108 Ran Silberman; "How to Unit Test Kafka — Ran Silberman" https://ransilberman.com/2013/07/19/how-to-unit-test-kafka PDF File QA Score: 0.47

S109 Stuart Perks; "The Perks of Computer Science" https://perkss.github.io/#/DistributedSystems/Streaming PDF File QA Score: 0.56

S110 Apache Foundation; "TopologyTestDriver (kafka 2.8.0 API)" https://kafka.apache.org/28/javadoc/org/apache/kafka/streams/TopologyTestDriver.html PDF File QA Score: 0.46

S111 Marcos Schroh, Konstantin Knauf, David Anderson, Felder; "testing - How to properly test a Flink window function?" https://stackoverflow.com/questions/56755349/how-to-properly-test-a-flink-window-function PDF File QA Score: 0.61

S112 Markus Günther; "User Guide to Kafka for JUnit" https://mguenther.github.io/kafka-junit PDF File QA Score: 0.58

S113 Konstantin Knauf, Apache Foundation; "GitHub - Flink Testing Pyramid: Example of a tested Apache Flink application". https://github.com/knaufk/flink-testing-pyramid PDF File QA Score: 0.63

S114 Yan Cui, Thundra; "Chaos test your Lambda functions with Thundra" https://blog.thundra.io/chaos-test-your-lambda-functions-with-thundra PDF File QA Score: 0.78

S115 Nick Dimiduk, Rich Metzger, Stephan Ewen, Till Rohrmann, Alexander Kolb; "Apache Flink User Mailing List archive. - Published test artifacts for flink streaming" http://apache-flink-user-mailing-list-archive.2336050.n4.nabble.com/Published-test-artifacts-for-flink-streaming-td3379.html#a3560 PDF File QA Score: 0.53

S116 Todd McGrath; "Kafka Streams Testing with Scala Part 1" https://supergloo.com/kafka-streams/kafka-streams-testing-scala-part-1 PDF File QA Score: 0.57

S117 Robin Moffatt; "Quick 'n Easy Population of Realistic Test Data into Kafka" https://rmoff.net/2018/05/10/quick-n-easy-population-of-realistic-test-data-into-kafka PDF File QA Score: 0.40

S118 Yeva Byzek, Confluent; "Easy Ways to Generate Test Data in Kafka" https://www.confluent.io/blog/easy-ways-generate-test-data-kafka PDF File QA Score: 0.85

S119 Todd McGrath; "Kafka Test Data Generation Examples" https://supergloo.com/kafka/kafka-test-data PDF File QA Score: 0.64

S120 Jukka Karbanen, Confluent; "Easy Kafka Streams Testing with TopologyTestDriver - KIP-470" https://www.confluent.io/blog/test-kafka-streams-with-topologytestdriver PDF File QA Score: 0.84

S121 Matthias J. Sax; "KIP-247: Add public test utils for Kafka Streams - Apache Kafka - Apache Software Foundation" https://cwiki.apache.org/confluence/display/KAFKA/KIP-247%3A+Add+public+test+utils+for+Kafka+Streams PDF File QA Score: 0.55

S122 Bakdata Data Engineering Blog; "GitHub - Fluent Kafka Streams Tests: Fluent Kafka Streams Test with Java" https://github.com/bakdata/fluent-kafka-streams-tests PDF File QA Score: 0.71
S123 Zerocode, JAppsConsultants; "GitHub - Zerocode: A community-developed, free, open source, microservices API automation and load testing framework built using JUnit core runners for Http REST, SOAP, Security, Database, Kafka and much more. Zerocode Open Source enables you to create, change, orchestrate and maintain your automated test cases declaratively with absolute ease". https://github.com/authorjapps/zerocode PDF File QA Score: 0.70
S124 LocalStack; "GitHub - Localstack: A fully functional local AWS cloud stack. Develop and test your cloud & Serverless apps offline!" https://github.com/localstack/localstack PDF File QA Score: 0.69
S125 Amazon; "Kinesis Data Generator" https://awslabs.github.io/amazon-kinesis-data-generator/web/help.html PDF File QA Score: 0.88
S126 Serkan Özal, Amazon; "How Thundra Decreased Data Processing Pipeline Delay By 3x on Average and 6x on P99" https://aws.amazon.com/pt/blogs/apn/how-thundra-decreased-data-processing-pipeline-delay-by-3x-on-average-and-6x-on-p99 PDF File QA Score: 0.77
S127 Zerocode; "Kafka Testing Introduction" https://knowledge.zerocode.io/knowledge/kafka-testing-introduction PDF File QA Score: 0.74
S128 Markus Günther; "Using Kafka for JUnit with Spring Kafka" https://mguenther.net/2021/03/using_kafka_for_junit_with_spring_kafka/index.html PDF File QA Score: 0.49
S129 Markus Günther; "Writing system tests for a Kafka-enabled microservice" https://mguenther.net/2021/02/writing_system_tests_for_kafka_microservices/index.html PDF File QA Score: 0.44
S130 Markus Günther; "Writing component tests for Kafka consumers" https://mguenther.net/2021/02/writing_component_tests_for_kafka_consumers/index.html PDF File QA Score: 0.44
S131 Markus Günther; "Writing component tests for Kafka producers" https://mguenther.net/2021/01/writing_component_tests_for_kafka_producers/index.html PDF File QA Score: 0.45
S132 Andy Mac Giolla Fhionntog, Philipp, James; "Python - Data testing framework for data streaming (deequ vs Great Expectations)" https://stackoverflow.com/questions/64711388/data-testing-framework-for-data-streaming-deequ-vs-great-expectations PDF File QA Score: 0.37
S133 L Johnson, Dmitri T; "Apache kafka - Adding SSL parameters to pepper_box config in jmeter" https://stackoverflow.com/questions/62783931/adding-ssl-parameters-to-pepper-box-config-in-jmeter PDF File QA Score: 0.49
S134 Julian Harty; "Better Software Testing Blog - Seeking ways to improve the efficiency and effectiveness of our craft" http://blog.bettersoftwaretesting.com PDF File QA Score: 0.86
S135 Sysco Corporation; "GitHub - Kkafka Testing: Test examples of kafka-clients: unit, integration, end-to-end" https://github.com/sysco-middleware/kafka-testing PDF File QA Score: 0.68
S136 Scale Out Data; "Unit Testing Kafka Streams with Avro - Scalable Data Processing on the JVM Stack" https://scaleoutdata.com/unit-testing-kafka-streams-with-avro-schemas PDF File QA Score: 0.55
S137 JAppsConsultants; "What is Zerocode Testing" https://github.com/authorjapps/zerocode/wiki/What-is-Zerocode-testing PDF File QA Score: 0.67
S138 JAppsConsultants; "Kafka Load Testing Getting Started" https://knowledge.zerocode.io/knowledge/kafka-load-testing-getting-started PDF File QA Score: 0.43
S139 Satish Bhor; "Pepper-Box Kafka Load Generator" https://dzone.com/articles/pepper-box-kafka-load-generator PDF File QA Score: 0.29
S140 Jay Kreps, Atlassian Corporation; "Performance testing - Apache Kafka - Apache Software Foundation" https://cwiki.apache.org/confluence/display/KAFKA/Performance+testing PDF File QA Score: 0.68
S141 Jamie Alquiza; "Load Testing Apache Kafka on AWS" https://grey-boundary.io/load-testing-apache-kafka-on-aws PDF File QA Score: 0.58
S142 Rami Amar, Alooma; "ETL Testing: The Future is Here" https://www.alooma.com/blog/etl-testing-the-future-is-here PDF File QA Score: 0.71
S143 Matthew Powers; "spark-fast-tests" https://github.com/MrPowers/spark-fast-tests PDF File QA Score: 0.73
S144 Andy Chambers, Confluent; "Testing Event-Driven Systems" https://www.confluent.io/blog/testing-event-driven-systems PDF File QA Score: 0.76
S145 Knoldus; "Writing Unit Test for Apache Spark using Memory Streams" https://blog.knoldus.com/apache-sparks-memory-streams PDF File QA Score: 0.56
S146 Amazon; "Serverless Streaming Architectures and Best Practices" https://d1.awsstatic.com/whitepapers/Serverless_Streaming_Architecture_Best_Practices.pdf PDF File QA Score: 0.75
S147 Francesco Tisiot, Aiven; "Create your own data stream for Kafka with Python and Faker" https://aiven.io/blog/create-your-own-data-stream-for-kafka-with-python-and-faker PDF File QA Score: 0.80
S148 Matthias J. Sax, Peyman, Gnos, thinktwice; "How to unit test a Kafka stream application that uses session window" https://stackoverflow.com/questions/57480927/how-to-unit-test-a-kafka-stream-application-that-uses-session-window PDF File QA Score: 0.49
S149 Andrea Leopardi; "StreamData: Property-based testing and data generation" https://elixir-lang.org/blog/2017/10/31/stream-data-property-based-testing-and-data-generation-for-elixir PDF File QA Score: 0.84
S150 Apache Foundation; "Apache Griffin" https://github.com/apache/griffin PDF File QA Score: 0.60
S151 Anupama Shetty, Neil Marshall; "Testing Spark: Best Practices" https://docplayer.net/3780717-Testing-spark-best-practices.html PDF File QA Score: 0.42
S152 Thomas Groh, Apache Foundation; "Testing Unbounded Pipelines in Apache Beam" https://beam.apache.org/blog/test-stream PDF File QA Score: 0.59
S153 Tom Seddon, Deliveroo; "Improving Stream Data Quality With Protobuf Schema Validation" https://deliveroo.engineering/2019/02/05/improving-stream-data-quality-with-protobuf-schema-validation.html PDF File QA Score: 0.86
S154 Confluent, Jay Kreps; "Why Avro for Kafka Data?" https://www.confluent.io/blog/avro-kafka-data PDF File QA Score: 0.73

References

Adams, J., Hillier-Brown, F.C., Moore, H.J., Lake, A.A., Araujo-Soares, V., White, M., Summerbell, C., 2016. Searching and synthesising 'grey literature'and 'grey information'in public health: critical reflections on three case studies. Syst. Rev. 5 (1), 1–11.

Adams, R.J., Smart, P., Huff, A.S., 2017. Shades of grey: guidelines for working with the grey literature in systematic reviews for management and organizational studies. Int. J. Manag. Rev. 19 (4), 432–454.

Akber, S.M.A., Chen, H., Jin, H., 2021. FATM: A failure-aware adaptive fault tolerance model for distributed stream processing systems. Concurr. Comput.: Pract. Exper. 33 (10), e6167.

Akidau, T., Balikov, A., Bekiroğlu, K., Chernyak, S., Haberman, J., Lax, R., McVeety, S., Mills, D., Nordstrom, P., Whittle, S., 2013. Millwheel: Fault-tolerant stream processing at internet scale. Proc. VLDB Endow. 6 (11), 1033–1044.

Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernández-Moctezuma, R.J., Lax, R., McVeety, S., Mills, D., Perry, F., Schmidt, E., et al., 2015. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. Proc. VLDB Endow. 8, 1792–1803.

Aladev, R., 2021. How to do kafka testing with jmeter. [online] (Accessed: 2022-07-19).
Alexandrov, A., Bergmann, R., Ewen, S., Freytag, J.-C., Hueske, F., Heise, A., Kao, O., Leich, M., Leser, U., Markl, V., et al., 2014. The stratosphere platform for big data analytics. VLDB J. 23 (6), 939–964.
Alquiza, J., 2021. Sangrenel. https://github.com/jamiealquiza/sangrenel.
Alshamrani, S., Waseem, Q., Alharbi, A., Alosaimi, W., Turabieh, H., Alyami, H., 2020. An efficient approach for storage of big data streams in distributed stream processing systems. Int. J. Adv. Comput. Sci. Appl. 11 (5).
Amazon Inc., 2021. Kinesis data generator. https://github.com/awslabs/amazon-kinesis-data-generator.
Ampatzoglou, A., Bibi, S., Avgeriou, P., Chatzigeorgiou, A., 2020. Guidelines for managing threats to validity of secondary studies in software engineering. In: Contemporary Empirical Methods in Software Engineering. Springer, pp. 415–441. http://dx.doi.org/10.1007/978-3-030-32489-6_15.
Anon, 2011. ISO/IEC 25010:2011: Systems and software engineering – systems and SoftWare quality requirements and evaluation (square) – system and software quality models. [online].
Anon, 2017. Test strategies for stream processing / event processing.
Apache Foundation, 2021a. Flink testing utilities - apache flink. https://mvnrepository.com/artifact/org.apache.flink/flink-test-utils.
Apache Foundation, 2021b. Jmeter. https://github.com/apache/jmeter.
Apache Foundation, 2021c. Kafka streams testing utilities. https://mvnrepository.com/artifact/org.apache.kafka/kafka-streams-test-utils.
Apache Software Foundation, 2022. Apache Flink mail archive. (Accessed: 2022-07-19).
August, T., Chen, W., Zhu, K., 2021. Competition among proprietary and open-source software firms: the role of licensing in strategic contribution. Manage. Sci. 67 (5), 3041–3066.
authorjapps, 2019. Kafka testing hello world examples. [online].
Babaei, M., Dingel, J., 2021. Efficient replay-based regression testing for distributed reactive systems in the context of model-driven development. In: 2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems. MODELS, IEEE, pp. 89–100.
Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J., 2002. Models and issues in data stream systems. In: Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. pp. 1–16.
Babu, S., Widom, J., 2001. Continuous queries over data streams. ACM Sigmod Record 30 (3), 109–120.
Backdata, 2021. Fluent kafka streams tests. https://github.com/bakdata/fluent-kafka-streams-tests.
Balazinska, M., Hwang, J.-H., Shah, M.A., 2009. Fault tolerance and high availability in data stream management systems. In: Encyclopedia of Database Systems. Vol. 11, p. 57.
Baqasah, A., Pardede, E., Rahayu, W., 2015. Maintaining schema versions compatibility in cloud applications collaborative framework. World Wide Web 18 (6), 1541–1577.
Barchenkov, I., 2019. Ecto stream factory. https://github.com/ibarchenkov/ecto_stream_factory.
Basiri, A., Behnam, N., De Rooij, R., Hochstein, L., Kosewski, L., Reynolds, J., Rosenthal, C., 2016. Chaos engineering. IEEE Softw. 33 (3), 35–41.
Bath, G., 2020. The next generation tester: Meeting the challenges of a changing IT world. Future Softw. Quality Assurance 15–26.
Begum, S.H., Nausheen, F., 2018. A comparative analysis of differential privacy vs other privacy mechanisms for big data. In: 2018 2nd International Conference on Inventive Systems and Control (ICISC). pp. 512–516.
Behjati, R., Arisholm, E., Bedregal, M., Tan, C., 2019. Synthetic test data generation using recurrent neural networks: a position paper. In: 2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering. RAISE, IEEE, pp. 22–27.
Benkhelifa, E., Hani, A.B., Welsh, T., Mthunzi, S., Guegan, C.G., 2019. Virtual environments testing as a cloud service: a methodology for protecting and securing virtual infrastructures. IEEE Access 7, 108660–108676.
Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., Gavaldà, R., 2009. New ensemble methods for evolving data streams. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp. 139–148.
Blanchi, C., Petrone, J., 2001. Distributed interoperable metadata registry. D-Lib Mag. 7 (12), 1082–9873.
Boroday, S., Petrenko, A., Groz, R., 2007. Can a model checker generate tests for non-deterministic systems? Electron. Notes Theor. Comput. Sci. 190 (2), 3–19.
Bourque, P., Fairley, R.E. (Eds.), 2014. SWEBOK: Guide to the Software Engineering Body of Knowledge, Version 3.0 IEEE Computer Society, URL http://www.swebok.org/.
Buchgeher, G., Fischer, S., Moser, M., Pichler, J., 2020. An early investigation of unit testing practices of component-based software systems. In: 2020 IEEE Workshop on Validation, Analysis and Evolution of Software Tests. VST, IEEE, pp. 12–15.
Bustamante, R., Garcés, K., 2020. Managing evolution of API-driven IoT devices through adaptation chains. In: CIbSE. pp. 85–95.
Cappellari, P., Chun, S.A., Roantree, M., 2016. ISE: A high performance system for processing data streams. In: DATA. pp. 13–24.
Carbone, P., Fragkoulis, M., Kalavri, V., Katsifodimos, A., 2020. Beyond analytics: The evolution of stream processing systems. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. pp. 2651–2658.
Carcillo, F., Dal Pozzolo, A., Le Borgne, Y.-A., Caelen, O., Mazzer, Y., Bontempi, G., 2018. Scarff: a scalable framework for streaming credit card fraud detection with spark. Inf. Fusion 41, 182–194.
Charaf, M.E.H., Azzouzi, S., 2016. Timed distributed testing rules for the distributed test architecture. In: 2016 4th IEEE International Colloquium on Information Science and Technology (CiSt). IEEE, pp. 314–319.
Chatterjee, S., Morin, C., 2018. Experimental study on the performance and resource utilization of data streaming frameworks. In: 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. CCGRID, IEEE, pp. 143–152.
Chen, G., Bai, G., Zhang, C., Wang, J., Ni, K., Chen, Z., 2022. Big data system testing method based on chaos engineering. In: 2022 IEEE 12th International Conference on Electronics Information and Emergency Communication. ICEIEC, IEEE, pp. 210–215.
Chen, T.-H., Syer, M.D., Shang, W., Jiang, Z.M., Hassan, A.E., Nasser, M., Flora, P., 2017. Analytics-driven load testing: An industrial experience report on load testing of large-scale systems. In: 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP). IEEE, pp. 243–252.
Chen, G.J., Wiener, J.L., Iyer, S., Jaiswal, A., Lei, R., Simha, N., Wang, W., Wilfong, K., Williamson, T., Yilmaz, S., 2016. Realtime data processing at facebook. In: Proceedings of the 2016 International Conference on Management of Data. pp. 1087–1098.
Chen, Y., Zhang, S., Guo, Q., Li, L., Wu, R., Chen, T., 2015. Deterministic replay: A survey. ACM Comput. Surv. 48 (2), 1–47.
Cherniack, M., Balakrishnan, H., Balazinska, M., Carney, D., Cetintemel, U., Xing, Y., Zdonik, S.B., 2003. Scalable distributed stream processing. In: CIDR. Vol. 3, pp. 257–268.
Chintapalli, S., Dagit, D., Evans, B., Farivar, R., Graves, T., Holderbaugh, M., Liu, Z., Nusbaum, K., Patil, K., Peng, B.J., et al., 2016. Benchmarking streaming computation engines: Storm, flink and spark streaming. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops. IPDPSW, IEEE, pp. 1789–1792.
Community, 2021a. Embedded kafka. https://github.com/embeddedkafka/embedded-kafka.
Community, 2021b. ScalaTest. https://github.com/scalatest/scalatest.
Condie, T., Conway, N., Alvaro, P., Hellerstein, J.M., Elmeleegy, K., Sears, R., 2010. MapReduce online. In: Nsdi. Vol. 10, p. 20.
Confluent Inc., 2017. Ksql-datagen. https://docs.confluent.io/5.4.0/ksql/docs/tutorials/generate-custom-test-data.html.
Confluent Inc., 2021a. Avro random generator. https://github.com/confluentinc/avro-random-generator.
Confluent Inc., 2021b. Confluent CLI. https://github.com/confluentinc/confluent-cli.
Confluent Inc., 2021c. Ducktape. https://github.com/confluentinc/ducktape.
Confluent Inc., 2021d. Kafka datagen connector. https://github.com/confluentinc/kafka-connect-datagen.
Corral-Plaza, D., Medina-Bulo, I., Ortiz, G., Boubeta-Puig, J., 2020. A stream processing architecture for heterogeneous data sources in the Internet of Things. Comput. Stand. Interfaces 70, 103426.
Cugola, G., Margara, A., 2012a. Processing flows of information: From data stream to complex event processing. ACM Comput. Surv. 44 (3), 15.
Cugola, G., Margara, A., 2012b. Processing flows of information: From data stream to complex event processing. ACM Comput. Surv. 44 (3), 1–62.
Dávid, I., Ráth, I., Varró, D., 2018. Foundations for streaming model transformations by complex event processing. Softw. Syst. Model. 17, 135–162.
de Assuncao, M.D., da Silva Veith, A., et al., 2018. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. J. Netw. Comput. Appl..
De Barros, M., Shiau, J., Shang, C., Gidewall, K., Shi, H., Forsmann, J., 2007. Web services wind tunnel: On performance testing large-scale stateful web services. In: 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07). IEEE, pp. 612–617.
Del Monte, B., Zeuch, S., Rabl, T., Markl, V., 2020. Rhino: Efficient management of very large distributed state for stream processing engines. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. pp. 2471–2486.
del Rio Astorga, D., Dolz, M.F., Fernández, J., García, J.D., 2018. Paving the way towards high-level parallel pattern interfaces for data stream processing. Future Gener. Comput. Syst. 87, 228–241.
Dell'Aglio, D., Della Valle, E., van Harmelen, F., Bernstein, A., 2017. Stream reasoning: A survey and outlook. Data Sci. 1 (1–2), 59–83.

Diaz, S.M., Souza, P.S., Souza, S.R., 2021. Structural testing for communication events into loops of message-passing parallel programs. Concurr. Comput.: Pract. Exper. 33 (18), e6082.
Dumitrescu, C., Raicu, I., Ripeanu, M., Foster, I., 2004. Diperf: An automated distributed performance testing framework. In: Fifth IEEE/ACM International Workshop on Grid Computing. IEEE, pp. 289–296.
Eismann, S., Bezemer, C.-P., Shang, W., Okanović, D., van Hoorn, A., 2020. Microservices: A performance tester's dream or nightmare? In: Proceedings of the ACM/SPEC International Conference on Performance Engineering. pp. 138–149.
Espinosa, C.V., Martin-Martin, E., Riesco, A., Rodríguez-Hortalá, J., 2019. FlinkCheck: property-based testing for Apache flink. IEEE Access 7, 150369–150382.
Faraglia, D., 2021. Faker (python). https://github.com/joke2k/faker.
Feiler, P.H., 2010. Model-based validation of safety-critical embedded systems. In: 2010 IEEE Aerospace Conference. IEEE, pp. 1–10.
Felderer, M., Russo, B., Auer, F., 2019. On testing data-intensive software systems. In: Security and Quality in Cyber-Physical Systems Engineering. Springer, pp. 129–148.
Filip, I.-D., Postoaca, A.V., Stochitoiu, R.-D., Neatu, D.-F., Negru, C., Pop, F., 2019. Data capsule: Representation of heterogeneous data in cloud-edge computing. IEEE Access 7, 49558–49567.
Fu, H.-H., Lin, D.K., Tsai, H.-T., 2006. Damping factor in google page ranking. Appl. Stoch. Models Bus. Ind. 22 (5–6), 431–444.
Funding Circle, 2021. Jackdaw - test machine. https://github.com/FundingCircle/jackdaw.
Gamov, V., 2020. I don't always test my streams, but when I do, I do it in production.
Garcia, A.M., Griebler, D., Schepke, C., Fernandes, L.G.L., 2022a. Evaluating micro-batch and data frequency for stream processing applications on multi-cores. In: 2022 30th Euromicro International Conference on Parallel, Distributed and Network-Based Processing. PDP, IEEE, pp. 10–17.
Garcia, A.M., Griebler, D., Schepke, C., Fernandes, L.G., 2022b. Spbench: a framework for creating benchmarks of stream processing applications. Computing 1–23.
Garofalakis, M., Gehrke, J., Rastogi, R., 2016. Data stream management: A brave new world. In: Data Stream Management: Processing High-Speed Data Streams. Springer, pp. 1–9.
Garousi, V., Felderer, M., Kuhrmann, M., Herkiloğlu, K., Eldh, S., 2020. Exploring the industry's challenges in software testing: An empirical study. J. Softw.: Evol. Process 32 (8), e2251.
Garousi, V., Felderer, M., Mäntylä, M.V., 2019. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. Inf. Softw. Technol. 106, 101–121. http://dx.doi.org/10.1016/j.infsof.2018.09.006, URL http://www.sciencedirect.com/science/article/pii/S0950584918301939.
Garousi, V., Mäntylä, M.V., 2016. When and what to automate in software testing? A multi-vocal literature review. Inf. Softw. Technol. 76, 92–117.
Gazzola, L., Goldstein, M., Mariani, L., Mobilio, M., Segall, I., Tundo, A., Ussi, L., 2022. ExVivoMicroTest: ExVivo testing of microservices. J. Softw.: Evol. Process e2452.
Geldenhuys, M.K., Pfister, B.J., Scheinert, D., Kao, O., Thamsen, L., 2021. Khaos: Dynamically optimizing checkpointing for dependable distributed stream processing. arXiv preprint arXiv:2109.02340.
Godefroid, P., Lehmann, D., Polishchuk, M., 2020. Differential regression testing for REST APIs. In: Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. pp. 312–323.
Godin, K., Stapleton, J., Kirkpatrick, S.I., Hanning, R.M., Leatherdale, S.T., 2015. Applying systematic review search methods to the grey literature: a case study examining guidelines for school-based breakfast programs in Canada. Syst. Rev. 4 (1), 1–10.
Google, 2021. Google's search algorithm and ranking system. [online] (Accessed: 2021-02-13).
Gorawski, M., Gorawska, A., Pasterak, K., 2014. A survey of data stream processing tools. In: Information Sciences and Systems 2014. Springer, pp. 295–303.
Grulich, P.M., Traub, J., Breß, S., Katsifodimos, A., Markl, V., Rabl, T., 2019. Generating reproducible out-of-order data streams. In: Proceedings of the 13th ACM International Conference on Distributed and Event-Based Systems. pp. 256–257.
Gu, R., Zhou, Y., Wang, Z., Yuan, C., Huang, Y., 2018. Penguin: Efficient query-based framework for replaying large scale historical data. IEEE Trans. Parallel Distrib. Syst. 29 (10), 2333–2345.
Gulzar, M.A., Zhu, Y., Han, X., 2019. Perception and practices of differential testing. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). IEEE, pp. 71–80.
Gunawi, H.S., Hao, M., Leesatapornwongsa, T., Patana-anake, T., Do, T., Adityatama, J., Eliazar, K.J., Laksono, A., Lukman, J.F., Martin, V., et al., 2014. What bugs live in the cloud? a study of 3000+ issues in cloud systems. In: Proceedings of the ACM Symposium on Cloud Computing. pp. 1–14.
Günther, M., 2021. Kafka for junit. https://github.com/mguenther/kafka-junit.
Gutiérrez-Madroñal, L., García-Domínguez, A., Medina-Bulo, I., 2019. Evolutionary mutation testing for IoT with recorded and generated events. Softw. - Pract. Exp. 49 (4), 640–672.
Gutiérrez-Madroñal, L., Medina-Bulo, I., Domínguez-Jiménez, J.J., 2018. IoT–TEG: Test event generator system. J. Syst. Softw. 137, 784–803.
Hanamanthrao, R., Thejaswini, S., 2017. Real-time clickstream data analytics and visualization. In: 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology. RTEICT, IEEE, pp. 2139–2144.
Hanawa, T., Banzai, T., Koizumi, H., Kanbayashi, R., Imada, T., Sato, M., 2010. Large-scale software testing environment using cloud computing technology for dependable parallel and distributed systems. In: 2010 Third International Conference on Software Testing, Verification, and Validation Workshops. IEEE, pp. 428–433.
Harsh, P., Ribera Laszkowski, J.F., Edmonds, A., Quang Thanh, T., Pauls, M., Vlaskovski, R., Avila-García, O., Pages, E., Gortázar Bellas, F., Gallego Carrillo, M., 2019. Cloud enablers for testing large-scale distributed applications. In: Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion. pp. 35–42.
Hasan, M., Orgun, M.A., et al., 2018. A survey on real-time event detection from the twitter data stream. J. Inf. Sci..
HashiCorp, 2021. Terraform. https://github.com/hashicorp/terraform.
Hashimov, E., 2015. In: Miles, M.B., Huberman, A.M., Saldaña, J. (Eds.), Qualitative Data Analysis: a Methods Sourcebook and the Coding Manual for Qualitative Researchers.
Hierons, R.M., Ural, H., 2008. Checking sequences for distributed test architectures. Distrib. Comput. 21 (3), 223–238.
Hill, J.H., Turner, H.A., Edmondson, J.R., Schmidt, D.C., 2009. Unit testing non-functional concerns of component-based distributed systems. In: 2009 International Conference on Software Testing Verification and Validation. IEEE, pp. 406–415.
Hoque, S., Miranskyy, A., 2018. Architecture for analysis of streaming data. In: 2018 IEEE International Conference on Cloud Engineering (IC2E). IEEE, pp. 263–269.
Hossayni, H., Khan, I., Crespi, N., 2021. Data anonymization for maintenance knowledge sharing. IT Prof. 23 (5), 23–30.
Hummel, O., Eichelberger, H., Giloj, A., Werle, D., Schmid, K., 2018. A collection of software engineering challenges for big data system development. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications. SEAA, IEEE, pp. 362–369.
Hynninen, T., Kasurinen, J., Knutas, A., Taipale, O., 2018. Software testing: Survey of the industry practices. In: 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics. MIPRO, IEEE, pp. 1449–1454.
Iglesias, F., Ojdanic, D., Hartl, A., Zseby, T., 2020. Mdcstream: Stream data generator for testing analysis algorithms. In: Proceedings of the 13th EAI International Conference on Performance Evaluation Methodologies and Tools. pp. 56–63.
Imtiaz, J., Sherin, S., Khan, M.U., Iqbal, M.Z., 2019. A systematic literature review of test breakage prevention and repair techniques. Inf. Softw. Technol. 113, 1–19.
Jaffari, A., Yoo, C.-J., Lee, J., 2020. Automatic test data generation using the activity diagram and search-based technique. Appl. Sci. 10 (10), 3397.
Jenkins, 2021. Jenkins. https://github.com/jenkinsci/jenkins.
Jiang, Z.M., Hassan, A.E., 2015. A survey on load testing of large-scale software systems. IEEE Trans. Softw. Eng. 41 (11), 1091–1118.
Johan Haleby and Community, 2021. Awaitility library. https://github.com/awaitility/awaitility.
Kaisler, S., Armour, F., Espinosa, J.A., Money, W., 2013. Big data: Issues and challenges moving forward. In: 2013 46th Hawaii International Conference on System Sciences. IEEE, pp. 995–1004.
Kallas, K., Niksic, F., Stanford, C., Alur, R., 2020. DiffStream: differential output testing for stream processing programs. Proc. ACM Program. Lang. 4 (OOPSLA), 1–29.
Kamei, F., Wiese, I., Lima, C., Polato, I., Nepomuceno, V., Ferreira, W., Ribeiro, M., Pena, C., Cartaxo, B., Pinto, G., Soares, S., 2021. Grey literature in software engineering: A critical review. Inf. Softw. Technol. 106609. http://dx.doi.org/10.1016/j.infsof.2021.106609, URL https://www.sciencedirect.com/science/article/pii/S0950584921000860.
Karau, H., 2016. Spark testing base. [online].
Karau, H., 2021. Spark testing base. https://github.com/holdenk/spark-testing-base.
Kargar, M.J., Hanifizade, A., 2018. Automation of regression test in microservice architecture. In: 2018 4th International Conference on Web Research. ICWR, IEEE, pp. 133–137.
Karimov, J., Rabl, T., Katsifodimos, A., Samarev, R., Heiskanen, H., Markl, V., 2018. Benchmarking distributed stream data processing systems. In: 2018 IEEE 34th International Conference on Data Engineering. ICDE, IEEE, pp. 1507–1518.

Karlsson, S., Čaušević, A., Sundmark, D., 2020. Quickrest: Property-based test generation of openapi-described restful APIs. In: 2020 IEEE 13th International Conference on Software Testing, Validation and Verification. ICST, IEEE, pp. 131–141.

Karlsson, S., Čaušević, A., Sundmark, D., 2021. Automatic property-based testing of graphql apis. In: 2021 IEEE/ACM International Conference on Automation of Software Test. AST, IEEE, pp. 1–10.

Kenthapadi, K., Tran, T.T.L., 2018. PriPeARL: A framework for privacy-preserving analytics and reporting at linkedin. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management.

Kim, S., Park, J., Kim, K.H., Shon, J.G., 2018. A test data generation for performance testing in massive data processing systems. In: Advanced Multimedia and Ubiquitous Engineering. Springer, pp. 207–213.

Komorniczak, J., Zyblewski, P., Ksieniewicz, P., 2022. Statistical drift detection ensemble for batch processing of data streams. Knowl.-Based Syst. 252, 109380.

Konstantinou, C., Stergiopoulos, G., Parvania, M., Esteves-Verissimo, P., 2021. Chaos engineering for enhanced resilience of cyber-physical systems. In: 2021 Resilience Week. RWS, IEEE, pp. 1–10.

Koukouvis, K., Cubero, R.A., Pelliccione, P., 2016. A/b testing in e-commerce sales processes. In: International Workshop on Software Engineering for Resilient Systems. Springer, pp. 133–148.

Krämer, J., Seeger, B., 2004. A Temporal Foundation for Continuous Queries over Data Streams. Univ..

Krawczyk, B., 2016. Learning from imbalanced data: open challenges and future directions. Prog. Artif. Intell. 5 (4), 221–232.

Krempl, G., Žliobaite, I., Brzeziński, D., Hüllermeier, E., Last, M., Lemaire, V., Noack, T., Shaker, A., Sievi, S., Spiliopoulou, M., et al., 2014. Open challenges for data stream mining research. ACM SIGKDD Explor. Newsl. 16 (1), 1–10.

Kreps, J., Narkhede, N., Rao, J., et al., 2011. Kafka: A distributed messaging system for log processing. In: Proceedings of the NetDB. pp. 1–7.

Kulesovs, I., 2015. iOS applications testing. In: Environment. Technologies. Resources. Proceedings of the International Scientific and Practical Conference. Vol. 3, pp. 138–150.

Lafi, M., Alrawashed, T., Hammad, A.M., 2021. Automated test cases generation from requirements specification. In: 2021 International Conference on Information Technology. ICIT, IEEE, pp. 852–857.

Lahariya, M., Benoit, D.F., Develder, C., 2020. Synthetic data generator for electric vehicle charging sessions: Modeling and evaluation using real-world data. Energies 13 (16), 4211.

Leesatapornwongsa, T., Lukman, J.F., Lu, S., Gunawi, H.S., 2016. Taxdc: A taxonomy of non-deterministic concurrency bugs in datacenter distributed systems. In: Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 517–530.

Leopardi, A., 2017. StreamData. [online].

Leopardi, A., Valim, J., 2021. StreamData. https://github.com/whatyouhide/stream_data.

Li, Y., Dong, T., Zhang, X., Song, Y.-d., Yuan, X., 2006. Large-scale software unit testing on the grid. In: GrC. pp. 596–599.

Li, N., Lei, Y., Khan, H.R., Liu, J., Guo, Y., 2016. Applying combinatorial test data generation to big data applications. In: 2016 31st IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 637–647.

Lima, B., 2019. Automated scenario-based integration testing of time-constrained distributed systems. In: 2019 12th IEEE Conference on Software Testing, Validation and Verification. ICST, IEEE, pp. 486–488.

Lima, B.C., Faria, J., 2017. Conformance checking in integration testing of time-constrained distributed systems based on UML sequence diagrams. In: Proceedings of the 12th International Conference on Software Technologies - ICSOFT. pp. 459–466.

Lima, B., Faria, J.P., Hierons, R., 2020. Local observability and controllability analysis and enforcement in distributed testing with time constraints. IEEE Access 8, 167172–167191.

Litt, G., Hardenberg, P.v., Henry, O., 2021. Cambria: schema evolution in distributed systems with edit lenses. In: Proceedings of the 8th Workshop on Principles and Practice of Consistency for Distributed Data. pp. 1–9.

Liu, X., Iftikhar, N., Xie, X., 2014. Survey of real-time processing systems for big data. In: Proceedings of the 18th International Database Engineering & Applications Symposium. ACM, pp. 356–361.

Lv, Y., Liu, R., Jin, P., 2021. Water-wheel: Real-time storage with high throughput and scalability for big data streams. In: Proceedings of the 33rd International Conference on Software Engineering and Knowledge Engineering. KSI Research Inc., pp. 634–635. http://dx.doi.org/10.18293/seke2021-204.

Madan, S., Goswami, P., 2018. A privacy preserving scheme for big data publishing in the cloud using k-anonymization and hybridized optimization algorithm. In: 2018 International Conference on Circuits and Systems in Digital Enterprise Technology. ICCSDET, IEEE, pp. 1–7.

Magnus Edenhill - Apache, 2021. Kcat. https://github.com/edenhill/kcat.

Mahood, Q., Van Eerd, D., Irvin, E., 2014. Searching for grey literature for systematic reviews: challenges and benefits. Res. Synth. Methods 5 (3), 221–234.

Majeed, A., Lee, S., 2020. Anonymization techniques for privacy preserving data publishing: A comprehensive survey. IEEE Access 9, 8512–8545.

Malaska, T., 2019. Mastering spark unit testing.

Malensek, M., Pallickara, S.L., Pallickara, S., 2011. Galileo: A framework for distributed storage of high-throughput data streams. In: 2011 Fourth IEEE International Conference on Utility and Cloud Computing. IEEE, pp. 17–24.

Manco, G., Ritacco, E., Rullo, A., Saccà, D., Serra, E., 2022. Machine learning methods for generating high dimensional discrete datasets. Wiley Interdiscip. Rev.: Data Min. Knowl. Discov. 12 (2), e1450.

Mannino, M., Abouzied, A., 2019. Is this real? Generating synthetic data that looks real. In: Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology. pp. 549–561.

Mäntylä, M.V., Smolander, K., 2016. Gamification of software testing-an MLR. In: International Conference on Product-Focused Software Process Improvement. Springer, pp. 611–614.

Marak, 2021. Fake.js. https://github.com/marak/Faker.js/.

McKeeman, W.M., 1998. Differential testing for software. Digit. Tech. J. 10 (1), 100–107.

Michael DeHaan and Red Hat Inc., 2021. Ansible. https://github.com/ansible/ansible.

Mishra, L., Varma, S., et al., 2020. Performance evaluation of real-time stream processing systems for internet of things applications. Future Gener. Comput. Syst. 113, 207–217.

Mockaroo LLC, 2021. Mockaroo. https://www.mockaroo.com/.

Namiot, D., 2015. On big data stream processing. Int. J. Open Inf. Technol. 3 (8), 48–51.

Nardelli, M., Cardellini, V., Grassi, V., Presti, F.L., 2019. Efficient operator placement for distributed data stream processing applications. IEEE Trans. Parallel Distrib. Syst. 30 (8), 1753–1767.

Orso, A., Shi, N., Harrold, M.J., 2004. Scaling regression testing to large software systems. ACM SIGSOFT Softw. Eng. Notes 29 (6), 241–251.

ottogroup, 2019. Flink spector. https://github.com/ottogroup/flink-spector.

Pagliari, A., Huet, F., Urvoy-Keller, G., 2020. Namb: A quick and flexible stream processing application prototype generator. In: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing. CCGRID, IEEE, pp. 61–70.

Philip Chen, C.L., Zhang, C.Y., 2014. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. Inform. Sci. 275, 314–347. http://dx.doi.org/10.1016/j.ins.2014.01.015.

Pizonka, S., Kehrer, T., Weidlich, M., 2018. Domain model-based data stream validation for internet of things applications. In: MODELS Workshops. pp. 503–508.

Popić, S., Pavković, B., Velikić, I., Teslić, N., 2019. Data generators: a short survey of techniques and use cases with focus on testing. In: 2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin). IEEE, pp. 189–194.

Rafi, D.M., Moses, K.R.K., Petersen, K., Mäntylä, M.V., 2012. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In: 2012 7th International Workshop on Automation of Software Test. AST, IEEE, pp. 36–42.

Rahutomo, R., Lie, Y., Perbangsa, A.S., Pardamean, B., 2020. Improving conversion rates for fashion e-commerce with a/b testing. In: 2020 International Conference on Information Management and Technology (ICIMTech). IEEE, pp. 266–270.

Ramya Shree, A., Kiran, P., Mohith, N., Kavya, M., 2022. Sensitivity context aware privacy preserving disease prediction. In: Expert Clouds and Applications. Springer, pp. 11–20.

Riehle, D., 2021. The open source distributor business model. Computer 54 (12), 99–103.

Riesco, A., Rodríguez-Hortalá, J., 2019. Property-based testing for Spark Streaming. Theory Pract. Log. Program. 19 (4), 574–602.

Saldaña, J., 2015. The Coding Manual for Qualitative Researchers. Sage.

Saleem, H., Uddin, M.K.S., Habib-ur Rehman, S., Saleem, S., Aslam, A.M., 2019. Strategic data driven approach to improve conversion rates and sales performance of e-commerce websites. Int. J. Sci. Eng. Res. (IJSER).

Samosir, J., Indrawan-Santiago, M., Haghighi, P.D., 2016. An evaluation of data stream processing systems for data driven applications. Procedia Comput. Sci. 80, 439–449.

Schleier-Smith, J., Krogen, E.T., Hellerstein, J.M., 2016. Restream: Accelerating backtesting and stream replay with serial-equivalent parallel processing. In: Proceedings of the Seventh ACM Symposium on Cloud Computing. ACM, pp. 334–347.

Shah, M.A., Hellerstein, J.M., Brewer, E., 2004. Highly available, fault-tolerant, parallel dataflows. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data. pp. 827–838.

Shahrivar, S., Elahi, S., Hassanzadeh, A., Montazer, G., 2018. A business model for commercial open source software: A systematic literature review. Inf. Softw. Technol. 103, 202–214.
Shahverdi, E., Awad, A., Sakr, S., 2019. Big stream processing systems: an experimental evaluation. In: 2019 IEEE 35th International Conference on Data Engineering Workshops. ICDEW, IEEE, pp. 53–60.
Sharma, A., Singh, G., Rehman, S., 2020. A review of big data challenges and preserving privacy in big data. In: Advances in Data and Information Sciences. Springer, pp. 57–65.
Silva, P., Paiva, A.C., Restivo, A., Garcia, J.E., 2018. Automatic test case generation from usage information. In: 2018 11Th International Conference on the Quality of Information and Communications Technology. QUATIC, IEEE, pp. 268–271.
Simonsson, J., Zhang, L., Morin, B., Baudry, B., Monperrus, M., 2021. Observability and chaos engineering on system calls for containerized applications in docker. Future Gener. Comput. Syst. 122, 117–129.
Stephens, R., 1997. A survey of stream processing. Acta Inform. 34 (7), 491–541.
Stepien, B., Peyton, L., 2020. Test coordination and dynamic test oracles for testing concurrent systems. In: SOFTENG 2020: The Sixth International Conference on Advances and Trends in Software Engineering. pp. 22–27.
Stonebraker, M., Çetintemel, U., Zdonik, S., 2005. The 8 requirements of real-time stream processing. ACM Sigmod Record 34 (4), 42–47.
Suhada, T.A., Ford, J.A., Verreynne, M.-L., Indulska, M., 2021. Motivating individuals to contribute to firms' non-pecuniary open innovation goals. Technovation 102, 102233.
Suleiman, D., Alian, M., Hudaib, A., 2017. A survey on prioritization regression testing test case. In: 2017 8th International Conference on Information Technology. ICIT, IEEE, pp. 854–862.
Sun, D., Yan, H., Gao, S., Zhou, Z., 2018. Performance evaluation and analysis of multiple scenarios of big data stream computing on storm platform. KSII Trans. Int. Inf. Syst. (TIIS) 12 (7), 2977–2997.
Tan, C., Behjati, R., Arisholm, E., 2019. A model-based approach to generate dynamic synthetic test data: A conceptual model. In: 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops. ICSTW, IEEE, pp. 11–14.
Tantalaki, N., Souravlas, S., Roumeliotis, M., 2020. A review on big data real-time stream processing and its scheduling techniques. Int. J. Parallel Emergent Distrib. Syst. 35 (5), 571–601.
Tom, E., Aurum, A., Vidgen, R., 2013. An exploration of technical debt. J. Syst. Softw. 86 (6), 1498–1516.
Tönjes, R., Barnaghi, P., Ali, M., Mileo, A., Hauswirth, M., Ganz, F., Ganea, S., Kjærgaard, B., Kuemper, D., Nechifor, S., et al., 2014. Real time iot stream processing and large-scale data analytics for smart city applications. In: Poster Session, European Conference on Networks and Communications. sn, p. 10.
Torkura, K.A., Sukmana, M.I., Cheng, F., Meinel, C., 2020. Cloudstrike: Chaos engineering for security and resiliency in cloud infrastructure. IEEE Access 8, 123044–123060.
Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J.M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., et al., 2014. Storm@ twitter. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. ACM, pp. 147–156.
Tucker, H., Hochstein, L., Jones, N., Basiri, A., Rosenthal, C., 2018. The business case for chaos engineering. IEEE Cloud Comput. 5 (3), 45–54.
Tun, M.T., Nyaung, D.E., Phyu, M.P., 2019. Performance evaluation of intrusion detection streaming transactions using apache kafka and spark streaming. In: 2019 International Conference on Advanced Information Technologies. ICAIT, IEEE, pp. 25–30.
Tyndall, J., 2010. AACODS checklist for appraising grey literature. [online] (Accessed: 2021-02-13).
Vasa, J., Thakkar, A., 2022. Deep learning: Differential privacy preservation in the era of big data. J. Comput. Inf. Syst. 1–24.
Vianello, V., Patiño-Martínez, M., Azqueta-Alzúaz, A., Jimenez-Péris, R., 2018. Cost of fault-tolerance on data stream processing. In: European Conference on Parallel Processing. Springer, pp. 17–27.
Vianna, A., Ferreira, W., Gama, K., 2019. An exploratory study of how specialists deal with testing in data stream processing applications. In: 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM, IEEE, pp. 1–6.
Vianna, A., Kamei, F., Gama, K., Zimmerle, C., Neto, J., 2022. Research data. http://dx.doi.org/10.6084/m9.figshare.22259539, (Accessed: 2023-03-12).
Wadge, W.W., Ashcroft, E.A., 1985. LUCID, the Dataflow Programming Language. Vol. 303, Academic Press London.
Waehner, K., 2022. Testing your apache kafka data with confidence. [online] (Accessed: 2022-07-19).
Wang, Z., Myles, P., Jain, A., Keidel, J.L., Liddi, R., Mackillop, L., Velardo, C., Tucker, A., 2021. Evaluating a longitudinal synthetic data generator using real world data. In: 2021 IEEE 34th International Symposium on Computer-Based Medical Systems. CBMS, IEEE, pp. 259–264.
Wang, X., Zhang, C., Fang, J., Zhang, R., Qian, W., Zhou, A., 2022. A comprehensive study on fault tolerance in stream processing systems. Front. Comput. Sci. 16 (2), 1–18.
Waseem, M., Liang, P., Shahin, M., Di Salle, A., Márquez, G., 2021. Design, monitoring, and testing of microservices systems: The practitioners' perspective. J. Syst. Softw. 182, 111061.
Wiesman, S., 2018. Testing stateful streaming applications.
Wingerath, W., Wollmer, B., Bestehorn, M., Succo, S., Ferrlein, S., Bücklers, F., Domnik, J., Panse, F., Witt, E., Sener, A., Gessert, F., Ritter, N., 2022. Beaconnect: Continuous web performance A/B testing at scale. Proc. VLDB Endow. 15 (12), 3425–3431. http://dx.doi.org/10.14778/3554821.3554833.
Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. pp. 1–10.
Wu, S., Liu, M., Ibrahim, S., Jin, H., Gu, L., Chen, F., Liu, Z., 2018. Turbostream: Towards low-latency data stream processing. In: 2018 IEEE 38th International Conference on Distributed Computing Systems. ICDCS, IEEE, pp. 983–993.
Xu, C., Wedlund, D., Helgoson, M., Risch, T., 2013. Model-based validation of streaming data:(industry article). In: Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems. ACM, pp. 107–114.
Yamato, Y., 2015. Automatic verification technology of software patches for user virtual environments on iaas cloud. J. Cloud Comput. 4 (1), 1–14.
Yasmin, J., Tian, Y., Yang, J., 2020. A first look at the deprecation of restful APIs: An empirical study. In: 2020 IEEE International Conference on Software Maintenance and Evolution. ICSME, IEEE, pp. 151–161.
Ye, Q., Lu, M., 2021. SPOT: Testing stream processing programs with symbolic execution and stream synthesizing. Appl. Sci. 11 (17), 8057.
Yoo, S., Harman, M., 2012. Regression testing minimization, selection and prioritization: a survey. Softw. Test. Verif. Reliab. 22 (2), 67–120.
Yu, T., Srisa-an, W., Rothermel, G., 2017. An automated framework to support testing for process-level race conditions. Softw. Test. Verif. Reliab. 27 (4–5), e1634.
Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., et al., 2016. Apache spark: a unified engine for big data processing. Commun. ACM 59 (11), 56–65.
Zeuch, S., Monte, B.D., Karimov, J., Lutz, C., Renz, M., Traub, J., Breß, S., Rabl, T., Markl, V., 2019. Analyzing efficient stream processing on modern hardware. Proc. VLDB Endow. 12 (5), 516–530.
Zhao, X., Garg, S., Queiroz, C., Buyya, R., 2017. A taxonomy and survey of stream processing systems. In: Software Architecture for Big Data and the Cloud. Elsevier, pp. 183–206.
Zhou, J., Li, S., Zhang, Z., Ye, Z., 2013. Position paper: Cloud-based performance testing: Issues and challenges. In: Proceedings of the 2013 International Workshop on Hot Topics in Cloud Services. pp. 55–62.
Zvara, Z., Szabó, P.G., Hermann, G., Benczúr, A., 2017. Tracing distributed data stream processing systems. In: 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS* W). IEEE, pp. 235–242.

Alexandre Vianna is a Ph.D. student at the Federal University of Pernambuco (Brazil) and an Associate Professor at the Federal Institute of Pernambuco (IFPE). His field of study is Software Engineering and Distributed Systems. His Ph.D. is being conducted under the supervision of Kiev Gama, with research focusing on the quality of data stream processing applications, specifically on testing techniques. He has experience researching and developing techniques for managing feature variabilities of Product Lines in the context of ERP systems at UFRN. Currently, he is also involved in teaching, research, and extension activities focused on innovation in Information Technology.

Fernando Kenji Kamei is Ph.D. in Computer Science and Associate Professor at the Instituto Federal de Alagoas (IFAL), Brazil. I research the intersections of Software Engineering and Empirical studies. My Ph.D. degree was at the Universidade Federal de Pernambuco (UFPE) in Brazil, under supervise of Dr. Sérgio Soares and Dr. Gustavo Pinto. My research investigation was about Grey Literature in Software Engineering. I am also interested in research about: Use the Empirical methods to improve Software Engineering, Social Aspects of Software Engineering, and Software Quality and Agile methods.

Kiev Gama is an associate professor of computer science at the Universidade Federal de Pernambuco (UFPE), Brazil. He received his Ph.D. in computer science from the University of Grenoble, France. His main research interests are related to the domains of Software Engineering (e.g, human aspects of SE) and Distributed Systems (e.g., reactive systems). He has also explored different aspects of time-bounded collaborative events (e.g., hackathons, game jams) within contexts such as open innovation and education.

Carlos Zimmerle is a Ph.D. student and researcher at the Federal University of Pernambuco (Brazil). His focus revolves around conducting researches in the area of programming language design and software engineering. Currently, he has invested his time in comprehending the design aspects of dataflow languages, more specifically in the area involving reactive programming. He got his Master degree at the Federal University of Pernambuco in 2019.

João Neto is currently a Master's student in Computer Science at CIn/UFPE and Software Engineer at CESAR (Recife's Center for Advanced Studies and Systems), being part of the development team of the meta platform for the Internet of Things, KNoT. He has a bachelor's degree in Information Systems from the University of Pernambuco (UPE), where he studied the Internet of Things, Smart Cities, Computer Networks, and Embedded Systems.