SEAM-EZ: Simplifying Stateful Analytics through Visual Programming

Zhengyan Yu
zhyyu@conviva.com
Conviva
Beijing, China

Hun Namkung
Conviva
Foster City, California, USA
hnamkung@conviva.com

Jiang Guo
Conviva
Beijing, China
jguo@conviva.com

Henry Milner
Conviva
Foster City, California, USA
hmilner@conviva.com

Joel Goldfoot
Conviva
Foster City, California, USA
jgoldfoot@conviva.com

Yang Wang
Conviva, University of Illinois at Urbana-Champaign
Champaign, Illinois, USA
ywang@conviva.com

Vyas Sekar
Conviva, Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
vsekar@conviva.com

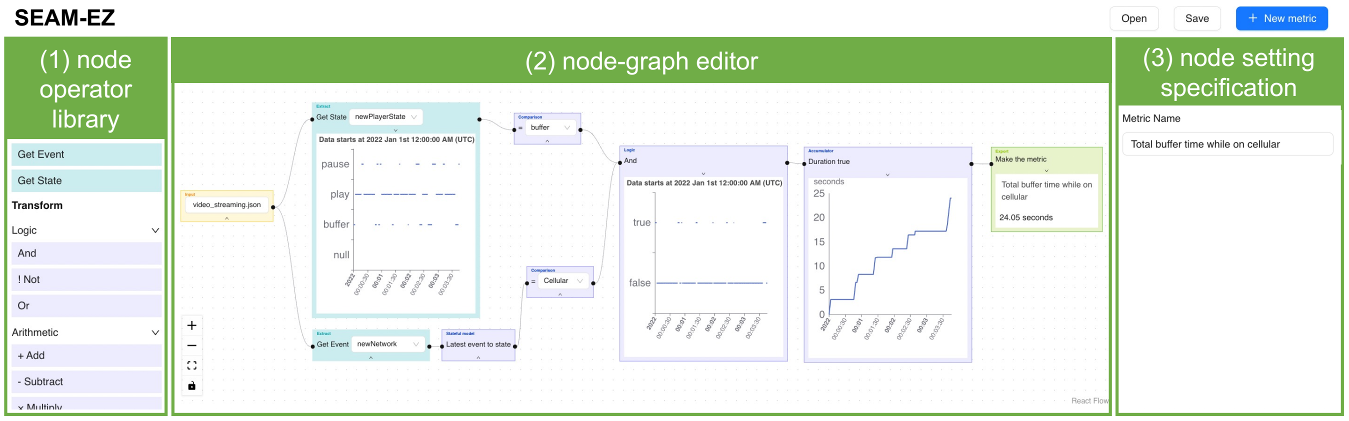


Figure 1: The main user interface (UI) of SEAM-EZ that allows users to visually and quickly create stateful metrics. The UI includes three panels: (1) a node operator library, (2) a node-graph editor, and, and (3) a panel for node setting specification. Users can drag and drop operators from the library into the node-graph editor to compose stateful metrics and every node has a data preview.

ABSTRACT

Across many domains (e.g., media/entertainment, mobile apps, finance, IoT, cybersecurity), there is a growing need for *stateful analytics over streams of events* to meet key business outcomes. Stateful analytics over event streams entails carefully modeling the sequence, timing, and contextual correlations of *events* to dynamic attributes. Unfortunately, existing frameworks and languages (e.g., SQL, Flink, Spark) entail significant code complexity and expert effort to express such stateful analytics because of their dynamic and stateful nature. Our overarching goal is to simplify and democratize stateful analytics. Through an iterative design and evaluation process including a foundational user study and two rounds of formative evaluations with 15 industry practitioners, we created SEAM-EZ, a no-code visual programming platform for quickly creating and validating stateful metrics. SEAM-EZ features a node-graph editor, interactive tooltips, embedded data views, and auto-suggestion features to facilitate the creation and validation of stateful analytics. We then conducted three real-world case studies of SEAM-EZ with 20 additional practitioners. Our results suggest

CHI '24, May 11–16, 2024, Honolulu, HI, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0330-0/24/05
https://doi.org/10.1145/3613904.3642055

that practitioners who previously could not or had to spend significant effort to create stateful metrics using traditional tools such as SQL or Spark can now easily and quickly create and validate such metrics using SEAM-EZ.

CCS CONCEPTS

• **Human-centered computing → Empirical studies in accessibility**;

KEYWORDS

visual programming, data analytics, stateful computation, metrics

ACM Reference Format:

Zhengyan Yu, Hun Namkung, Jiang Guo, Henry Milner, Joel Goldfoot, Yang Wang, and Vyas Sekar. 2024. SEAM-EZ: Simplifying Stateful Analytics through Visual Programming. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24), May 11–16, 2024, Honolulu, HI, USA.* ACM, New York, NY, USA, 23 pages. https://doi.org/10.1145/3613904.3642055

1 INTRODUCTION

The rise of big data and machine learning applications has created many design opportunities for human-computer interaction (HCI) researchers and practitioners. There has been a growing body of HCI work that designs interactive systems to support these data intensive applications (e.g., [19, 51]). In this paper, we focus on a pervasive yet understudied data challenge in HCI: *stateful analytics* over data streams. *Stateful* analytics can be described as [1]:

> *"... when the output depends on a sequence of events across one or more input streams. For example, the customer account lifecycle in a business might involve multiple stages, such as account creation, plan upgrades, downgrades, and cancellation. To derive attributes like the lifetime of an account or the latest plan the account is on, we need to track the sequence of these events ..."*

To illustrate the need for stateful analytics, let us consider some example scenarios as seen in Figure 2:

- *App monitoring:* Figure 2a shows events from an application login session and the server load status of the backend server. The app developers may be interested in the user experience, such as modeling the page load time or the time between the user clicking on the app to the time the content is rendered on the device screen. Furthermore, they may want to correlate these experience metrics with the backend load, e.g., is the "page load" time because the server load is high?
- *Fitness tracking:* Figure 2b shows events from a fitness tracker showing a user's stress change measurements, and the type of activity at various points in time (e.g., start times of rest, work, run). In this example, we may be interested in measuring the total duration of "high stress" level when the user is "working" to help the user avoid prolonged stress exposure; e.g., suggesting taking breaks between meetings.

While these examples are from diverse domains, the common pattern here is that turning the raw event stream into actionable insights (e.g., server management, model user experience, and health outcomes), entails carefully modeling the *sequence* of events, *timing* between events, and *temporal correlations* with other system variables. For example, Figure 2a shows the logic to model the page load time by tracking the time and sequence between the "red" (i.e., app load starts) and "green" (i.e., page load finishes) events and also correlating the value at each point in time with the server load state. The need for stateful analytics over time varying event streams is far more ubiquitous (e.g., [12, 27, 42]).

1.1 Stateful vs. Stateless Analytics.

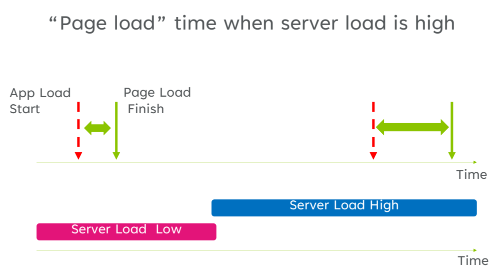
To understand *stateful* analytics, it may be useful to contrast it to *stateless* analytics. Suppose, in the previous figure, we are interested in the total number of load start or "red" events. Calcuating this measurement over the data will be the same irrespective of the sequence between the red/green events, the time difference between the occurrences of those events, and the current value of the server load; i.e., it is stateless. In contrast, to compute the stateful metric of "page load time when server load is high," we must consider event sequence, timing, and other system attributes at the same time.

Expressing such stateful analytics in real-world settings is challenging. It requires *expert developers* to create complicated SQL queries or write custom code in sophisticated frameworks like Apache Spark or Apache Flink [35]. Let us consider the fitness analysis example from earlier. To express this analysis in SQL we would end up writing very complex query logic as shown in Figure 3. We do not expect a casual reader to actually understand this query. Indeed, it takes even expert data analysts several tens of minutes to be able to understand this, let alone write this code in the first place! The problem is not confined to SQL: we have faced very similar difficulties implementing these stateful metrics in event-processing systems such as Apache Spark or Apache Flink, as they offer too low level code APIs, often drawn from a similar tabular mental model [11, 13]. In production systems, developers have to write code like this for hundreds of metrics, the problem is further exacerbated due to long development time and lots of subtle bugs.

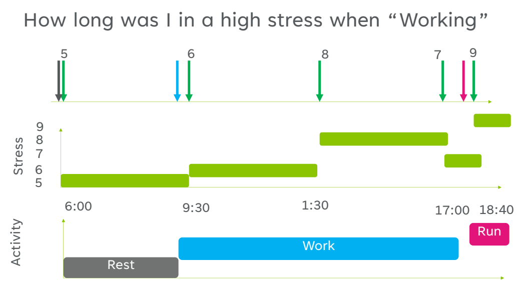
The root cause of the issue is that almost all existing data analysis systems are based on a tabular mental model. In general, this mental model and tooling are versatile for *stateless* analytics such as group-by operations, filters, and aggregations, which are agnostic to the sequence, timing, and temporal effects. Unfortunately, these existing tools do not offer native or intuitive abstractions for manipulating time, duration, and event sequences. Thus, the status quo is that *stateful* analytics requires expert code developers to create complex code to express their data analysis intents.

1.2 Visual Programming for Stateful Metrics

In this work, we envision an easy-to-use interactive system by which our target users (details below) can directly create sophisticated stateful analytics and immediately see the results on dashboards *without writing a single line of code.* In this context, a recent proposal in the database community from Milner et al., proposed a Timeline abstraction that enables a "geometric" basis for viewing data as visual timelines of events, measurements, and continuously evolving system variables [35]. Milner et al., also introduce high-level operators to support stateful analytics. While this approach was shown to enable *system performance* improvements, it still relies on code APIs to use the abstraction. As such the problem of designing interactive systems to simplify the creation of stateful analytics remains an open question. Note that our goal is not to



(a) App Monitoring



(b) Fitness Tracking

Figure 2: Examples of stateful analytics from different domains

```
WITH filledTable(sessionId, time, StressChange, ActivityChange) AS (
    SELECT sessionId, time,
           LAST_VALUE(StressChange IGNORE NULLS) OVER (PARTITION BY sessionId ORDER BY time) AS StressChange,
           LAST_VALUE(ActivityChange IGNORE NULLS) OVER (PARTITION BY sessionId ORDER BY time) AS ActivityChange
    FROM raw_data
),
sessionStartEndTable(sessionId, start_time, end_time) AS (
    SELECT sessionId, CAST(min(time) / 60 AS INTEGER), CAST(max(time) / 60 + 1 AS INTEGER)
    FROM raw_data
    GROUP BY sessionId
),
queryPointsTable(sessionId, time, is_query) AS (
    SELECT sessionid, UNNEST(generate_series(start_time, end_time)) * 60, true
    FROM sessionStartEndTable
    GROUP BY sessionId, start_time, end_time
),
withQueryTable(sessionId, time, StressChange, ActivityChange, is_query) AS (
    SELECT sessionId, time, StressChange, ActivityChange, is_query
    FROM (
        SELECT
            COALESCE(filledTable.sessionId, queryPointsTable.sessionId) as sessionid,
            COALESCE(filledTable.time, queryPointsTable.time) as time,
            StressChange, ActivityChange,
            queryPointsTable.is_query
        FROM filledTable
        FULL OUTER JOIN queryPointsTable ON filledTable.sessionId = queryPointsTable.sessionId
        AND filledTable.time = queryPointsTable.time
    )
),
durationTable(sessionId, time, StressChange, ActivityChange, duration, is_query) AS (
    SELECT sessionId, time, StressChange, ActivityChange,
           LEAD(time, 1) OVER (PARTITION BY sessionId ORDER BY time) - time, is_query
    FROM withQueryTable
),
resultTable AS (
    SELECT * FROM (
        SELECT sessionId, time AS time, is_query,
               SUM(CASE WHEN StressChange > 7 AND ActivityChange = 'work'  THEN duration ELSE 0 END)
               OVER (PARTITION BY sessionId ORDER BY time) AS value
        FROM durationTable
    )
    WHERE is_query = true
),
SELECT sessionId, time, value
FROM resultTable
```

Figure 3: Complex SQL code that an expert developer may need to write for a fitness scenario to express the stateful query: "time in high stress when working"

completely replace SQL or traditional data analytics tools. Rather, our goal is to tackle the unaddressed problem of stateful analytics where existing data analytics abstractions (e.g. tabular) and capabilities (e.g., SQL, Spark, Flink) are lacking. Said differently, our goal in this paper is not to have the full feature set of SQL for general-purpose data analysis, but rather our focus on a more scoped problem of making it easy to express stateful analytics. We envision SEAM-EZ can be integrated into existing data analytics pipelines, including SQL-based workflows; but this is outside the scope of the visual builder which is the focus of this paper.

1.3 Target Users

To make our problem more concrete, consider the following concrete setting and user personas. We consider an Internet video analytics setting where video players send raw player event measurements to a data analytics backend. This analytics provider may have several kinds of use case scenarios:

- *Technical Customer Service Representatives:* Customer service representatives often need to do some stateful analysis in response to customer requests to troubleshoot issues in content delivery; e.g., how many sessions have a particular kind of buffering or video start failure?
- *Technical Product Managers:* Product managers may want to define and analyze new kinds of stateful analytics to define new Quality of Experience metrics to track business relevant outcomes; e.g., buffering after ad play, or the user exits after ad play [18].
- *Data Scientists:* Data scientists may be asked to do new kinds of on-demand stateful analytics to inform the product delivery; e.g., are subtitles aligned with the video in time or are subtitles failing to load within 10 seconds for specific content assets?
- *Software Engineers:* Software engineers often want to troubleshoot user experience issues or performance issues by implementing stateful analytics over the client- and system metrics, event logs, and traces they collect.

Our primary target users are those who have domain expertise (e.g. video streaming, app monitoring) and have some technical background (e.g., basic knowledge of SQL or Python) but they lack the higher level expertise to write stateful analytics queries such as those expressed in Figure 3 in SQL, Spark, Flink, or other equivalent tools. As mentioned above, their job titles might range from technical product managers to technical customer service engineers and data scientists. Based on our conversations with numerous customers from various industries, these professionals have domain expertise but do not have the level of sophistication of expert programming skills in streaming applications to implement the stateful metrics of interest. In addition, our proposed system should also support expert developers who can implement stateful analytics by writing complex code. Our system should allow them to more easily and efficiently create stateful analytics compared to writing low-level code or queries.

1.4 User-Centered Design of SEAM-EZ

We took a user-centered approach in designing such an interactive system, similar to the recent Rapsai system for AI-based multimedia applications [19]. To better understand practitioners' existing practices with stateful analytics, we conducted a foundational user study with six industry practitioners who have created, customized and/or used stateful metrics to gather insights about their common tasks and challenges in this process. The study helped us derive a number of design goals. This foundational study informed the iterative design and evaluation of SEAM-EZ[1], a no-code visual programming platform for quickly creating stateful metrics. We were inspired by the prior successes of visual programming [45] in simplifying various programming tasks (e.g., [19]). SEAM-EZ features a node-graph editor, interactive tooltips, embedded data views, and auto-suggestion capabilities to facilitate metrics creation and validation. While there have been prior efforts for visual query builders for SQL, to the best of our knowledge, our work is the first attempt to apply visual programming for stateful analytics.

We performed two rounds of formative evaluations with 11 practitioners to iteratively improve the SEAM-EZ prototype. We then conducted three case studies of SEAM-EZ with a total of another 20 practitioners. Our results suggest that practitioners found SEAM-EZ intuitive and most of them were able to use SEAM-EZ to create stateful metrics in the case studies. Our participants also preferred using SEAM-EZ than using a SQL editor to write SQL queries. Similar to visual programming systems such as Rapsai, conducting longitudinal quantitative experiments/comparisons is very challenging with developers, due to their time constraints. We acknowledge this is a limitation that is not unique to our system-oriented work. Our qualitative data include not only survey results but also interviews and task observations. We triangulate these different data to derive our high-level findings. We also identified areas to improve SEAM-EZ.

1.5 Contributions

Our work makes three main contributions:

- *Problem understanding:* To the best of our knowledge, our work presents the first study to empirically characterize how practitioners currently deal with stateful analytics and identify design goals for interactive systems that support creating and validating stateful analytics.
- *System contributions:* We designed and implemented the first (to the best of our knowledge) no-code platform for stateful analytics. This novel system includes (1) a visual programming frontend that allows users to drag and drop, and then connect operators to create stateful metrics in form of directed acyclic graphs (DAGs); and (2) a performant backend that can compute complex stateful metrics in both batch file and streaming modes. Similar to other work such as Rapsai that applied visual programming to complex domains such as machine learning, one contribution of our work is in demonstrating the viability and utility of visual programming in the complex domain of stateful analytics. Applying visual programming to stateful metrics is non trivial, entailing many challenges, e.g., (1) what concepts from stateful metrics need to be modeled by visual nodes and edges, and (2) how to promote learning on the new concepts of the Timeline framework. For instance, the tooltip animation is a novel feature in visual programming systems that we have not seen before.
- *Validation and reflection:* We conducted a case study of SEAM-EZ and it yields promising results,. For instance, those who were not able to create stateful metrics before could now create such metrics using SEAM-EZ. We reflect on our experiences and lessons learned in applying visual programming to stateful metrics and outline future research.

[1]SEAM-EZ stands for Stateful Event Analytics Made Easy; a tool that makes what used to be a complex problem seem easy!

2 RELATED WORK

To the best of our knowledge, SEAM-EZ is the first visual programming system for creating and validating stateful metrics. Below, we discuss prior work in abstractions for data analysis and visual programming.

Data processing systems. There is a large body of prior work on technical approaches to support processing of event data and/or time series data; e.g., temporal SQL [28], time series databases [38], complex event processing [20], and functional reactive programming [49]). While new code generation techniques offered by modern "generative AI" co-pilots can reduce some of the syntactic burden of using SQL-like tools (e.g., [41]), they do not solve the semantic burden of debugging analytics intents.

More closely related to our work is the recent work in the database community by Milner et al., which identified key shortcomings in these existing processing paradigms. This work proposes a new *Timeline* abstraction that argues for raising the level of programming for enabling stateful analytics. Here, event data is viewed as logical timelines and we use new types of operators for manipulating data to express stateful analysis. For a more detailed discussion on these techniques, we refer interested readers to [35]. While we take inspiration from this abstraction, this prior work does not tackle HCI and visual programming challenges related to such an approach, which is our focus.

Visual programming. The seminal idea of visual programming (VP), i.e., directly using visual elements (e.g" blocks, lines) for constructing computer programs, dates back to the 60's [45]. Since then, there have been a wide range of VP applications for various domains. Here we focus on the usage of VP in understanding and/or creating complex computational systems. VP has been applied in various aspects of software programming, such as program visualization [8, 10], program understanding [29], rapid prototyping [6, 40], domain-specific programming [31, 39], program execution [24], visual art programming [33], and robotics programming [26]. Visual programming (VP) has also been applied in many scientific disciplines, such as geo-analysis [46], clinical rules [32], microarray analysis [15], sequencing data analytics [34], image analytics [22], general scientific data analysis [53], urban visual analytics [16], transportation [23], and story creation [14, 17]. There are also many commercial tools that use a visual programming approach, such as Blender [21], Maya [3], Teachable Machine [7], and TikTok Effect House [2].

More recently, VP has been used in machine learning and AI systems, such as clustering and classification [9], semantic segmentation [25], natural language processing [30, 48], explainable machine learning [43, 50], ensemble models [47], and model comparisons [36, 52]. Most related to SEAM-EZ is the literature on VP systems for building AI pipelines; e.g., AI Chains for chaining large language model prompts [5, 51] and Rapsai for rapid prototyping AI-based multimedia applications [19]. However, they do not tackle stateful analytics. Our overarching goal in this work is to simplify the process of creating stateful analytics and offer a dramatically easier alternative. To this end, we envision lowering the barrier of entry to express complex stateful analysis using a "no-code" visual programming approach.

3 ITERATIVE DESIGN AND EVALUATION PROCESS

Our iterative design and evaluation process included (1) a foundational user study to understand their current practices and challenges with stateful metric creation and validation, (2) design and prototyping of the visual programming metric tool, (3) two rounds of formative user evaluations of the prototype, and (4) case studies of the improved prototype. This iterative approach was inspired by the development of tools such as [19, 36, 51]. The sections below will detail each of these research components. In general, for the qualitative data, we reviewed and discussed the data and created affinity diagrams to identify the main findings. We received a certificate from the IRB office allowing us to conduct this work.

To understand the user needs and challenges for a rapid prototyping system for creating and validating stateful computation metrics, we conducted a foundational user study using semi-structured interviews with six practitioners who had experience in creating and validating metrics. We present key process and findings here and the detailed protocol in the Appendix.

3.1 Foundational Study with Semi-Structured Interviews

We recruited six practitioners via group email invitations at our organization, a key player in the industry of system monitoring and data analytics. As part of the product, analysts at our organization build and maintain stateful analysis metrics relevant for modeling user perceived quality of experience. Our participants (I1-I6) are 30-40 years old. Five of them are male and the other is female. They included customer support engineers, data scientists, and software engineers. None of them were familiar with SEAM-EZ and were not directly involved in this project. Each interview took about one hour consisting of two main components: background and stateful analytics practices. Five of them were conducted face-to-face and the other was done online via Zoom.

Background. In the 1st phase (10 mins), we asked about participants' demographics, educational and professional background. All of our participants self-reported having a background in STEM (science, technology, engineering, and math) and having at least one year of experience in creating and validating stateful metrics.

Stateful analytics practices. In the 2nd phase (50 mins), we asked the participants about their general procedure of their stateful metric creation and validation process, the challenges they have encountered, and the tools they have used in this process.

3.2 Tasks and Challenges

Two researchers organized participants' responses using affinity diagrams to identify the main findings. Figure 4 shows a typical workflow of stateful metric creation and validation, as well as five main tasks (T) and six challenges (C) that our participants had in that process. It is important to note that this is an iterative process and different tasks might feed into each other (e.g., metric validation can lead to refinement of metric specification).

3.2.1 Task (T1): Prepare event data. We use the term *event data* to denote broadly any event-level measurement data such as website/app/ad usage data or video streaming system data that could

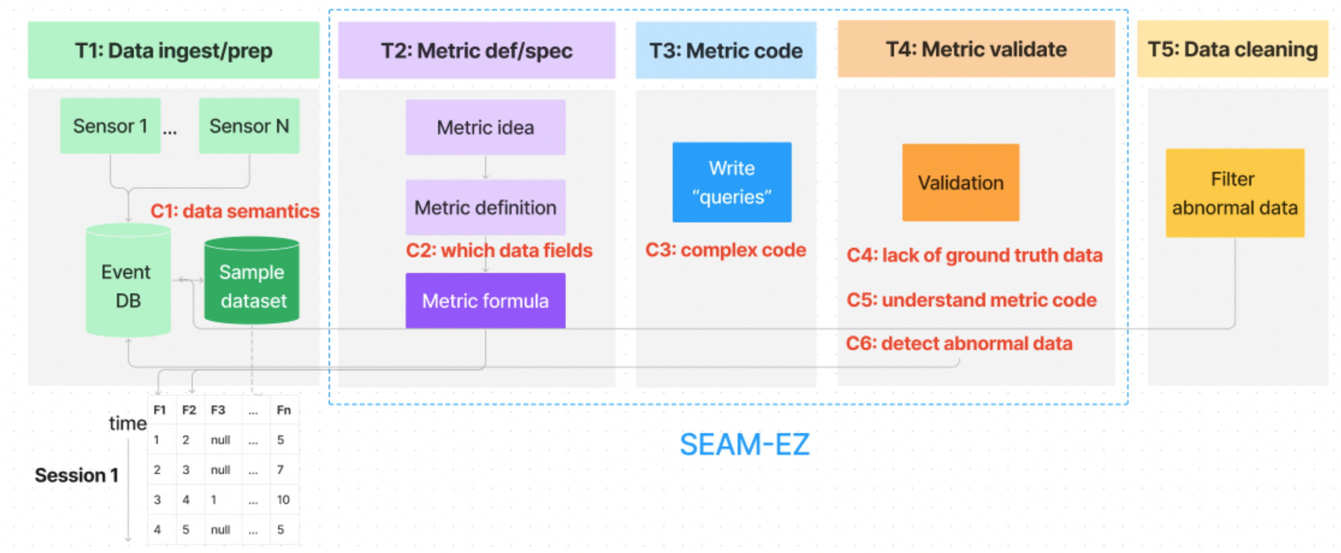


Figure 4: Workflow of creating and validating metrics. There are five main tasks: (1) data ingestion and preparation, (2) metric definition and specification, (3) metric implementation, (4) metric validation, and (5) data cleaning/sanitization. We also observed six challenges (C1-6) that they encountered in this process. F1-Fn represent the 1st field/column through the nth field in the representative sample (often a time-series dataset for a single session).

be used to create metrics. Typically, sensors (e.g., web browsers, video players) collect usage or system data. These data are stored in an "event" database and a few representative samples (i.e., events from one or more user sessions) are used for metrics building. A data team is often responsible for this task.

Challenge (C1): Lack of data semantics. A key challenge in data preparation reported by two participants I2 and I5 was that the raw event data often lack semantics. For instance, different sensors or devices might collect the same data under different names. For instance, I5 explained: "*Say customer X*[2] *contracted 2 companies, one for iOS, one for Android, and they developed this...you are developing something and you are in your own world because the ecosystem is completely different. There's no consistency of management across the platforms.*" Without a standard of data semantics, it is hard to create metrics that cover multiple devices (e.g., iOS and Android devices). Once such a standard is put in place, the raw event data could be structured in the event database and a representative sample (typically covers a single context, e.g., a user session) could then be drawn.

3.2.2 Task (T2): Create metric definitions and specifications. The ideas for metrics often come from product or operations teams. The former are in charge of creating products or services for customers, whereas the latter are responsible for ensuring the technical systems are operating as expected. Once a metric idea is formed, metric creators then develop a clear metric definition (e.g., average page load time is the average time user browsers take to load a web page). Next, the metric creators specify the formula to calculate the metric value by using the data fields from the sample session dataset. Data scientists or software engineers often finish this task.

[2]X is a popular video content provider; redacted for confidentiality reasons.

Challenge (C2): Lack of insights about which event data fields to use for metric calculation. The event data often include different measurements, some of which are proxies of others. For instance, I2 gave us an example of a metric called dropped frame ratio. Originally he used a event data field called dropped frame in the metric formula. However, he said "*the [data] team reached out and asked why use that field. It doesn't have much data.*" It turns out that the dropped frame field has many missing values from various devices and platforms which made it a poor choice in the metric calculation. Instead, they eventually chose two other fields (encoded frames and loaded frames) to calculate dropped frames. This illustrates the challenge of figuring out which data field(s) should be used to calculate a metric.

3.2.3 Task (T3): Implement metrics. Once the metric specification has been created, it becomes the *boundary object* [44] between different teams such as the data team, data science, and software engineering teams. The software engineering team usually implements the metric calculation by writing code according to the metric specification.

Challenge (C3): Complex code The metric code is often complex, error-prone and time-consuming to write in part because using traditional SQL or frameworks like Apache Spark or Flink to implement stateful computations is difficult. For instance, I3 described a situation when she was trying to diagnose a common problem and created a simple metric to calculate the count of tickets from the customer call center by device. She originally tried SQL queries and pivot tables in Excel and did not succeed. She then tried another approach "*I had a VBA program that I would paste into Excel and the DBA program.*" A co-worker helped her with the VBA program even for a simple metric.

3.2.4 *Task (T4): Validate metrics.*

It is one thing to implement a metric, and it is another thing to ensure the metric is implemented as expected. Our participants talked about various strategies for metric validation, ranging from using a Venn diagram to figure out the metric value (I1) to creating one's own metric implementation in data analysis tools like R or Spark (I2), from checking the aggregated metric values on a dashboard (I4) to comparing correlated metrics (I6). However, they encountered a few key challenges.

Challenge (C4): Hard to detect abnormal data Sometimes the event measurement data exhibit abnormal behavior. I5 provided a telling example. A metric such as the video startup time may be affected by specific attributes like the video preview or mini-play feature; i.e., the time to play should look at the actual fullscreen content not the time when the mini-play preview started.

Challenge (C5): Lack of ground truth data Ground truth data about what the metric values should be is very valuable but often missing. For instance, I6 pointed out: "*the biggest challenge in metric validation is the lack of ground truth data...if the sensor is doing something wrong, it's sending you bad measurements. How do you know?*"

Challenge (C6): Hard to understand metric implementation code This is related to the fact that metrics are hard to implement and therefore also hard for the metric validators to understand. For instance, I2 explained that he had to check the metric implementation code to try to understand the metric logic: "*My lack of understanding of how it is really working. And even going into the code, I still sometimes have to guess.*"

3.2.5 *Task (T5): Data Cleaning.*

If they identify abnormal data in the event measurements, they usually have two options: first, changing the metric definition by covering many corner cases; and second, filtering out the abnormal data, i.e., not using them in the metric value calculation. Today, they prefer the second option because of the overhead of creating and updating metric definitions.

Our focus in this work is primarily on Tasks T2, T3, and T4. While SEAM-EZ's features can help the practitioners for simplifying current workflows for T1 and T5, we leave them for future work.

4 DESIGN GOALS

Informed by the foundational study results, we identified five design goals to address the challenges faced by our participants. In this work (SEAM-EZ), we focus on Tasks T2–T4 and their corresponding design goals G1–G4.

G1. Provide a visual programming platform for simplifying metric creation and validation (C3, C6) Today, analysts have to write complex query code to calculate stateful metrics. This makes the metric implementation and subsequent validation complex, error-prone, and time-consuming. Therefore, it might be beneficial to provide users a no-coding visual programming platform to rapidly create and validate metrics.

G2. Support users to understand the dataset (C2, C4) Users often want to use visual cues to play with the data to see the names of the various fields and their values. This goal should allow users to explore and examine (e.g., slice and dice using the Timeline operators as well as visualizing) the underlying dataset, for instance, to detect potential abnormal behavior or patterns (C4).

G3. Support user provided datasets (C5) Users are allowed to provide their own dataset and they might know the ground truth associated with that the dataset. This could support both metric creation and validation. In industry settings, a common practice is to run experiments with controllable and configurable end points to learn the ground truth metrics. This design goal (G3) allows users to import the ground truth data they obtained to create stateful metrics.

G4. Provide timeline operator semantics (C6) While the Timeline idea and operators can raise the level of abstraction [35], these concepts and operators would be new to users. Therefore, it is critical to explain semantics about these operators so that users can more easily and appropriately use these operators to create metrics.

5 SEAM-EZ: SYSTEM ARCHITECTURE

We designed SEAM-EZ and its underlying Timeline framework implementation iteratively over a year with frequent feedback from various teams of customer satisfaction representatives, data scientists, backend software engineers and product managers from the formative evaluations and informal conversations. Here we first describe main components of the SEAM-EZ UI, and then discuss how the interaction between UI frontend and backend is implemented.

High-level overview. At a high level, SEAM-EZ provides a "drag and drop" composition interface for users to create custom stateful metrics on their datasets. In essence, stateful metrics can be created by chaining together different operators [35] as a directed acyclic graph (DAG). To satisfy our design goals, the key design decisions we made are: exposing the operators or building blocks; enabling a user-friendly interaction for chaining together operators; and providing visual cues and hints to the users as they build the metric to validate their semantic intents. To this end, the current UI of SEAM-EZ consists of three coordinated panels: (1) node/operator library, (2) node-graph editor, and (3) node setting specification. SEAM-EZ UI was mostly written in JavaScript (React) and the underlying Timeline operators were implemented in Rust, with a custom Python backend to translate requests between the UI and the Rust backend.

5.1 Nodes (Timeline Operators) Library

The nodes library panel lists a set of Timeline operators (currently 19), as described by prior work [35]. The *operators* library provides building blocks that users can compose to express stateful metrics in the form of a directed acyclic graph (DAG). We largely adapt the operator concepts from Milner et al., [35]. Our contribution here is in helping users better understand their semantics and use them to create stateful metrics in a visual programming interface rather than writing code. We group the operators into different categories for easy navigation.

5.1.1 *Input Category.*

By using `Dataset` operator, users can choose the input dataset file that they wish to analyze. Initially, only JSON format is supported for the input dataset file. However, it is straightforward to expand support for additional formats such as CSV.

5.1.2 *Extract Category.*

`GetEvent` and `GetState` operators are for extracting a timeline out of the input dataset file by specifying

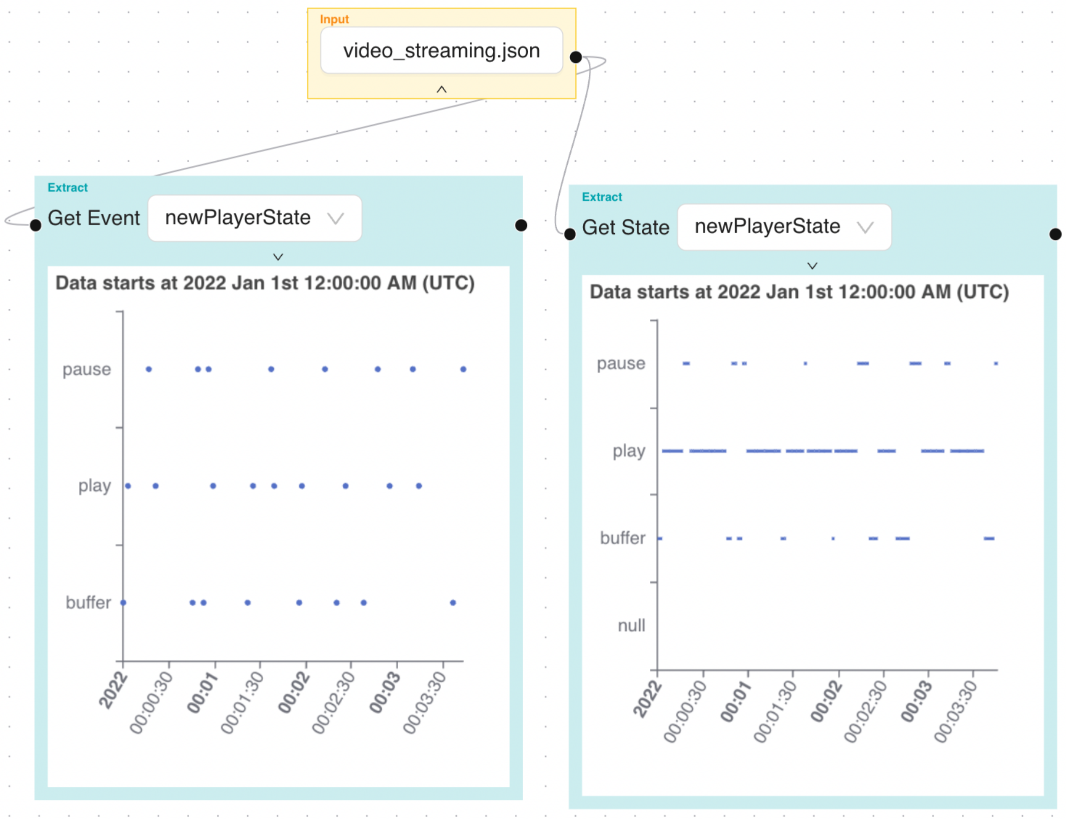
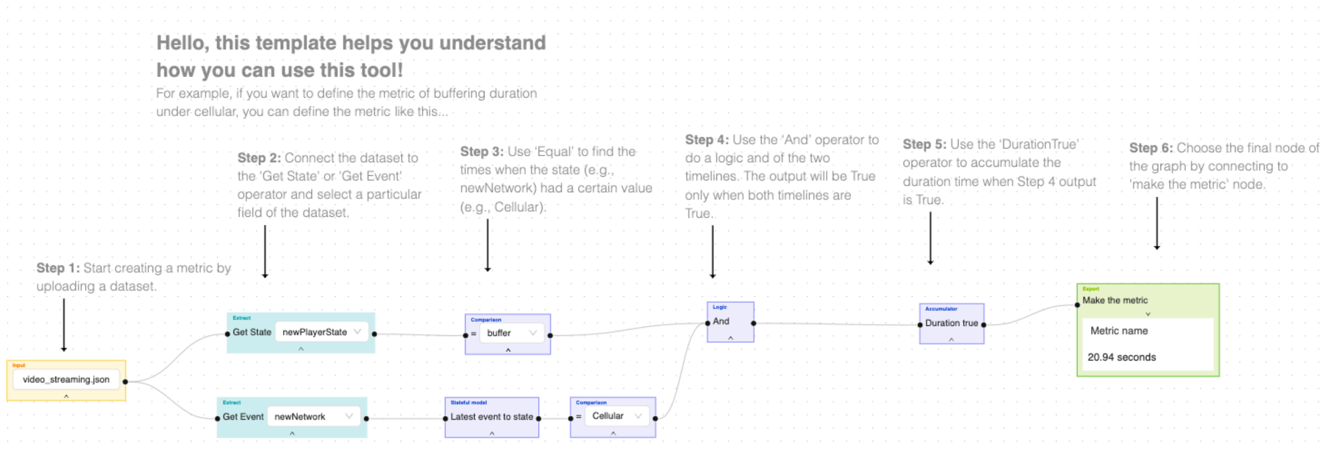


Figure 5: `GetEvent` **vs** `GetState`**.** `GetEvent` **outputs each measurement entry** $\langle time, fieldname, value_{time} \rangle$ **as a discrete event occurring at time** *time* **(visualized as a dot on the graph). In contrast,** `GetState` **outputs each measurement entry as a logical** ***state transition*** **event (visualized as a line on the graph).**

the data field name. Both operators must be connected to the `Dataset` operator. `GetEvent` operator outputs the event timeline and `GetState` operator outputs the state timeline. Given a set of $\langle time, fieldname, value_{time} \rangle$ measurement entries in the input dataset file (e.g., the *fieldname* is newPlayerState and buffer is one of the $value_{time}$ in Figure 5), the `GetEvent` essentially visualizes each $value_{time}$ as a discrete event occurring at time *time* with the associated *fieldname*. The `GetState` alternatively interprets each measurement entry $\langle time, fieldname, value_{time} \rangle$ as a logical *state transition* event in a finite state machine, and visualizes the *fieldname* as a *step function* over time, where the value of the step function on the Y-axis is set to the $value_{time}$ until the next measurement reported for that *fieldname*.

5.1.3 Transform Category.

- *Logic operator group.* This group of operators handles logic operations. `And` (&) and `Or` (|) operators perform logical operations. `Not` operator flips the Boolean value. For these logic operators, the value type of the input timelines must be *boolean* and the value type of the output timelines is also *boolean*.
- *Arithmetic operator group.* This group performs arithmetic operations; `Add` (+), `Subtract` (-), `Multiply` (×), `Divide` (÷). The value type of the input timelines and the output timelines for these operators is *number*.
- *Comparison operator group.* This operator group performs comparison operations; `GreaterThan` (>), `GreaterThanOr EqualTo` (≥), `LessThan` (<), `LessThanOrEqualTo` (≤). The value type of the input timelines for these four operators must be *number*. `Equal` (=) operator is an exception because it can accept other value types for the input timelines such as *string* (e.g., string value equality comparison). Thus, the equal operator can allow the input timelines of *any* value type. The value type of the output timeline for all of these comparison operators is always *boolean*.
- *Stateful model operator group.* This operator group performs stateful operations. `HasBeenTrue` operator receives an input timeline of *boolean* value type and outputs the state timeline of *boolean* value type. The output timeline is false until it sees the first true value. Afterwards, it remains true forever. `LatestEventToState` converts the event timeline into the state timeline.
- *Accumulation operator group.* `DurationTrue` operator receives the state timeline of *boolean* value type as input and it outputs the state timeline of *number* value type. The output number value means the total duration in seconds while the input the state timeline is in the true state.



Hello, this template helps you understand how you can use this tool!

For example, if you want to define the metric of buffering duration under cellular, you can define the metric like this...

Step 1: Start creating a metric by uploading a dataset.

Step 2: Connect the dataset to the 'Get State' or 'Get Event' operator and select a particular field of the dataset.

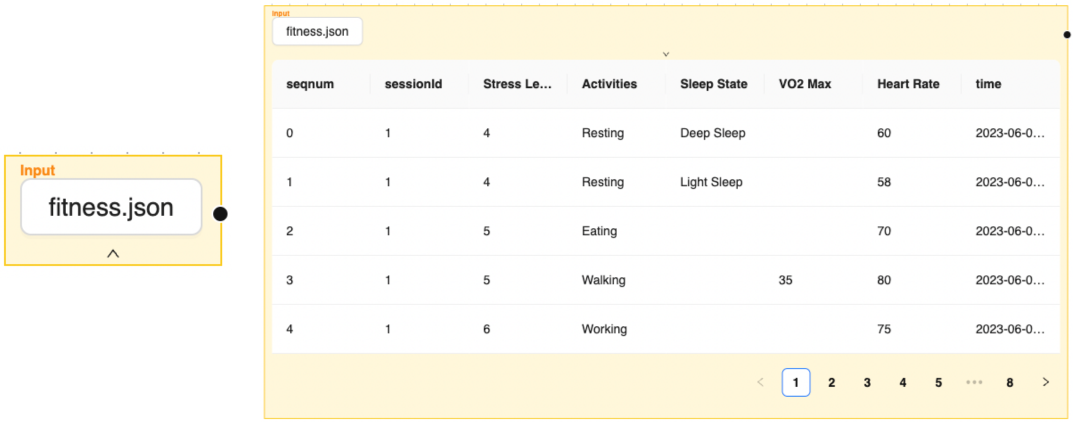
Step 3: Use 'Equal' to find the times when the state (e.g., newNetwork) had a certain value (e.g., Cellular).

Step 4: Use the 'And' operator to do a logic and of the two timelines. The output will be True only when both timelines are True.

Step 5: Use the 'DurationTrue' operator to accumulate the duration time when Step 4 output is True.

Step 6: Choose the final node of the graph by connecting to 'make the metric' node.

Figure 6: Metric template. When users first launches SEAM-EZ**, the node-graph editor canvas shows this template.**



seqnum	sessionId	Stress Le...	Activities	Sleep State	VO2 Max	Heart Rate	time
0	1	4	Resting	Deep Sleep		60	2023-06-0...
1	1	4	Resting	Light Sleep		58	2023-06-0...
2	1	5	Eating			70	2023-06-0...
3	1	5	Walking		35	80	2023-06-0...
4	1	6	Working			75	2023-06-0...

Figure 7: Preview the dataset after it is uploaded. Display/hide the preview by clicking the top area of the box around the arrow.

5.1.4 *Export Category.* `MakeMetric` operator is attached to the final node of interest in the DAG so that the system will use the output of the specified node to create the metric.

5.2 Node-Graph Editor

We designed a node-graph editor to help users rapidly create stateful metrics in a visual programming platform **(G1)**.

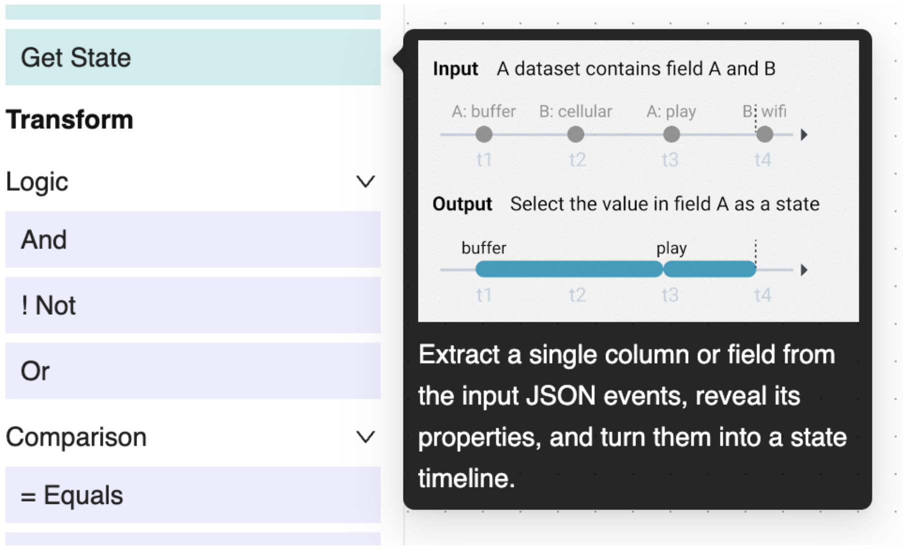
Metric template. Based on the participant feedback from the formative evaluations that users need an overview of the metric creation process in SEAM-EZ, we added a metric template (Figure 6) as a starting page of the editor canvas when a user first opens SEAM-EZ. To create a new metric, users can either directly modifies the metric template or click the *New metric* button on the top of SEAM-EZ landing page.

Drag-and-drop. In the node-graph editor, users can drag nodes/operators from the nodes library (left panel), connect nodes by dragging a line between their connectors (dots on the node edge).

Dataset preview. Metrics creation often starts with dragging a dataset operator into the editor canvas and selecting an input dataset to ingest into the system **(G3)**. If a user is not familiar with the dataset then the user needs to open and explore the dataset. To make this step more efficiently, SEAM-EZ directly embeds a dataset preview **(G2)** (see Figure 7) inside of the editor canvas.

Operators tooltips with animations. We anticipate that users might be new to the Timeline concept and its operators [35]. Therefore, in addition to the video tutorials and metric template, we designed SEAM-EZ to include operator tooltips **(G4)**. When a user mouses over an operator either in the nodes library (left panel) or the editor canvas (center panel), a tooltip will pop up which includes a textual explanation of the operator semantics as well as a dynamic animation. We believe that animation is a good design choice for illustrating the time-elapsed aspect of stateful operators as it directly shows how the timeline object might change its value as time changes. Figure 8 shows an example of such a tooltip (for the `Equals` operator).

Auto-suggestions for next operators. To explore the next operator to connect the current operator to, a user can click on the output connector of the current operator, then an *auto-suggest* list of candidate operators (Figure 9) will pop up so that the user



Extract a single column or field from the input JSON events, reveal its properties, and turn them into a state timeline.

Figure 8: Operator tooltip with animation. Example: `GetState` operator. The animation shows applying `GetState` with field A, i.e., the continuous value over time for field A (field A has the value *buffer* from time t1 to t3 and a different value of *play* from time t3 to t4).

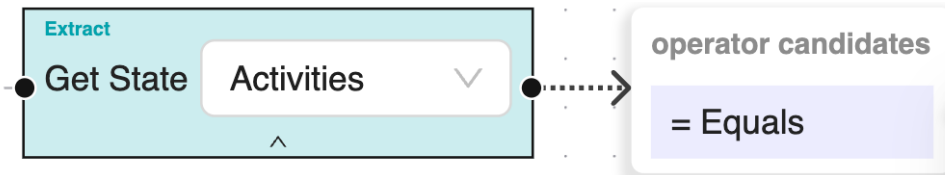


Figure 9: Auto-suggest operator candidate(s). In this example, the `GetState` operator with string values can only connect to an `Equals` operator, so the operator candidate just displays the `Equal` operator.

can explore and choose from. These auto-suggestion candidates are shown based on the rule that the output type of the current operator and the input type of the next operator are compatible.

Timeline preview. Once a user adds and connects operators in the editor canvas to compose the metric DAG, the user can get a Timeline preview of any node/operator in the DAG by clicking on the bottom edge of a node (Figure 10). This feature was informed by participant feedback from our formative evaluations and we designed it as a validation feature. The preview provides just-in-time visual cues to enable users to check whether they are composing the metric DAG as they expected. This feature also allows users to debug the metric DAG if they identify any discrepancy between the preview and their expected results.

5.3 Node Setting Specification

The third panel (right panel in Figure 1) of the SEAM-EZ UI is for users to specify or modify the settings of a node/operator by selecting options from a drop-down menu, or to edit the metric name when selecting the "Make the metric" node.

5.4 System Implementation

The system implementation of SEAM-EZ is decomposed into the UI frontend and the backend. The UI frontend is implemented with JavaScript React Flow and the backend is a HTTP web server implemented with Python FastAPI framework. Within the backend, we have a TimeLine Backend (TLB) binary with which we can execute the metric DAG.

TimeLine Backend (TLB). At a high level, TimeLine Backend (TLB) gets two inputs: the input dataset file and the metric DAG with the target output node. With these two inputs, TLB generates a file that contains timeline output for the requested node. We implemented TLB following the detailed system design principles from Milner et al.'s work [35]. The system can support both batch and streaming modes of operation.

Steps to integrate the UI frontend and the backend. Figure 11 shows five steps to run the metric DAG and visualize timeline preview into the UI frontend.

(1) Upload the input dataset file. Users of SEAM-EZ first select the input dataset file they want to analyze. This file will be uploaded to the backend.
(2) Click a node to see the timeline preview. After the user created the metric DAG, users can interactively see the timeline preview of any of the nodes that they created by clicking

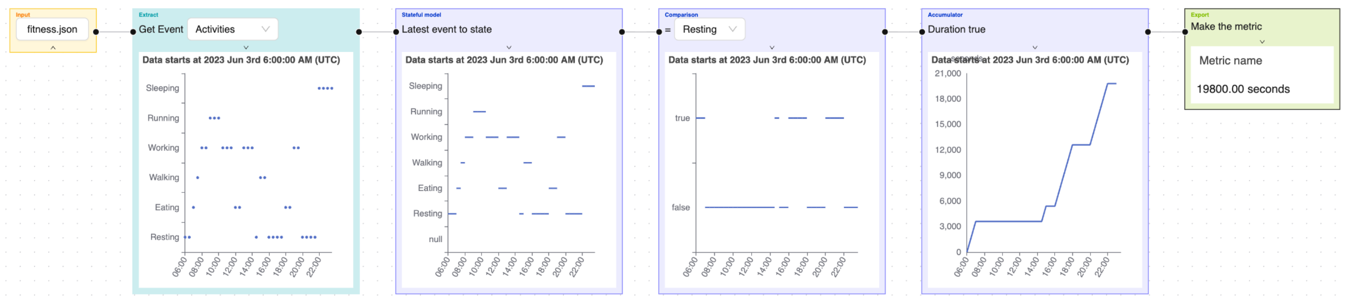


Figure 10: Users can get a timeline preview of any operator in a metric DAG. We used four colors to depict the different steps of composing a metric DAG: yellow (adding a node of a dataset), cyan (interpreting the fields as discrete events or states over time), purple (transforming the data, e.g. equals, greater than, and, duration true), and green (export the metric).

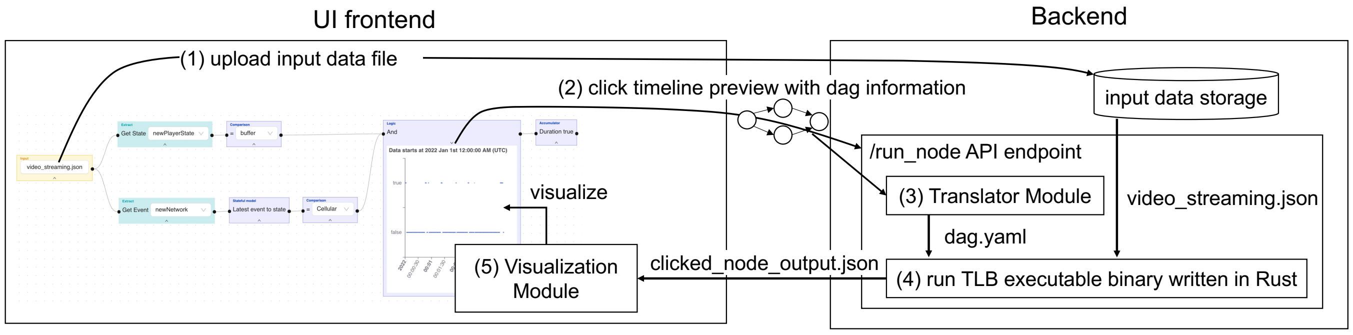


Figure 11: Overview of SEAM-EZ **System Implementation**

the node. Then, the UI frontend sends an HTTP request to run_node API endpoint with the DAG information.

(3) The DAG information is generated by the built-in export API of the ReactFlow framework, thus it cannot be directly understood by TLB. Instead, the metric DAG must follow a predefined set of rules so that the internal compiler of TLB can parse and run the metric DAG. Thus, we built a translator module to translate exported DAG information into a YAML (a human-readable markup language) file that represents the metric DAG and can be parsed by TLB.
(4) Using the DAG YAML file and the input dataset file, TLB runs the the metric DAG and outputs the timeline in JSON format. Then the API endpoint returns the timeline output results to the the UI frontend.
(5) When the UI frontend receives the timeline output results, it is in the form of a set of spans of time and the corresponding y-axis values. Since visual cue is the one of key contributions of SEAM-EZ, we have a visualization module to plot results on the UI frontend using the JavaScript Apache ECharts library.

Performance of SEAM-EZ. SEAM-EZ is performant due to the efficiency of the backend core engine TLB. TLB is built on the Rust programming language, which achieves significant throughput improvement over other JVM/Scala based implementations of the Timeline concept. This enables users to see the results of adding operators in near real time. For example, for the metric DAG illustrated in Figure 6, the average latency to receive results from the TLB backend, after clicking the last node, is sub-second latency of 382ms, with both the UI frontend and TLB running on a local machine. This interactivity also helps users explore different operators and learn the tool as they use it.

6 FORMATIVE USER EVALUATIONS

We conducted two rounds of formative user evaluations of our prototype with 11 practitioners in the same organization. We present the high-level study design and findings below. Detailed study prototype can be found in the Appendix C.2.

6.1 First Round Formative Evaluations

Five practitioners (F1-F5) participated in our first round formative evaluations of the prototype. This group included three product managers, one software engineer, and one designer. They were all males. One of the software engineer (I5) participated in the foundational user study as well as the formative evaluation. Each formative evaluation session took about 30-40 minutes and consisted of three main parts: watching short video tutorials (about the Timeline framework, the operators, and how to use the prototype UI), conducting tasks to create metrics using the prototype, and answering exit interview questions (such as their perceived pros and cons of the prototype, their suggestions for improving the prototype, and the system usability scale). Upon participant permission, each session was (screen/audio) recorded via Zoom with auto-caption so we can observe and analyze how they used the prototype.

High-level findings. All five participants were intrigued by the visual no-code approach and commended that this approach could simplify and expedite data exploration and metric creation compared to writing complex code. Four participants had technical backgrounds and were able to create metrics correctly for the tasks. However, a designer who did not have a computer science or technical background (e.g., did not know SQL or a programming language) struggled to create metrics because he had difficulties in understanding concepts of the Timeline framework and some of its operators such as GetState. After all, the stateful analytics and the Timeline framework are technical. We need design features to educate novice users about these concepts. It is important to note that only one participant (the software engineer) created stateful metrics before. While the three project managers (participants) did not directly create metrics before, they were able to do so with SEAM-EZ. This was a promising early sign that SEAM-EZ might enable a wider range of users to create stateful metrics.

Usability issues. We categorized these issues into (1) conceptual issues, i.e., those related to the Timeline operators (e.g., did not understand what a particular operator does), (2) effectiveness issues, i.e., those that led to incorrect metrics (e.g., the GET operator expects field names (e.g., network), but users typed a field value (e.g., WIFI)), and (3) efficiency issues, i.e., those that led to creating correct metrics but in an inefficient way (e.g., it was tedious to find and copy/paste field names or field values from the dataset).

Design changes. We improved the prototype design according to these identified issues. For instance, the improved prototype added a tooltip for each operator with an animated data graph to illustrate what the operator does to the data, a data preview of the dataset, a starting DAG template when the tool is first opened, a graph of the current data for each operator in the DAG, an auto-suggest feature that provides a list of "next" operators whose input data type should be compatible with the output data type of the "previous" operator, and a drop-down list of values so users can just select from rather than needing to find, copy and paste the values manually.

6.2 Second Round Formative Evaluations

After updating the prototype based on the identified issues from the 1st round evaluations, we conducted another round of formative evaluations with six more practitioners from the same organization (F6-11). This group included two females and four males. They are one product manager, two software engineers, and three data scientists. One data scientist from the foundational user study (I4) also participated in the 2nd round formative evaluation.

The second round formative evaluation was very similar to the first round except that we added interview questions about participants' attitudes towards the three major design changes: operator tooltips with animations, data preview, and auto-suggest operators.

High level findings. Overall, all participants were able to correctly create metrics in the tasks. Similar to the first round results, not all participants in this round created stateful metrics before. The fact that they were all able to do so with SEAM-EZ was promising. In addition, the ideas of the three new design features were mostly well-received albeit there were some usability issues with the implementation.

Usability issues. The participants encountered a few glitches using the new prototype. One issue was related to the new auto-suggest feature, which was automatically triggered and popped up whenever a user clicked anywhere on a node in the canvas. Many participants found this annoying. Another issue was that the button to "expand" the data preview was too small to see and/or click on. Once the data preview is shown, participants had to manually resize the canvas to see the entire DAG. A few participants noted that the video tutorials had some discrepancy with the prototype UI (e.g., the structure of the operators was different).[3] Some participants also wished they could see the computed metric value of the input (a single context, e.g., a user session) dataset directly to make metric validation easier. A similar request was to connect the prototype to a dashboard where they could see the aggregated metric results of applying a metric across multiple contexts (e.g., user sessions) and group by different attributes such as devices (e.g., Android devices) and locations (e.g., India).

Design changes. We updated the prototype design to address those identified issues and suggested features. For instance, we connected the prototype to an aggregate metric dashboard, included the computed metric value based on the single context (e.g., a user session) input dataset in the final node of a metric DAG, changed the triggering of the auto-suggest feature to only when a user clicks on the right middle point of an operator (to draw an edge to another operator), making the "expand" data preview feature more prominent to see and click on, and updating the video tutorials to match with the updated prototype UI.

7 CASE STUDIES

After improving the prototype based on user feedback from our two rounds of formative user evaluations, we reached out to 21 additional practitioners and ran three real-world case studies to

[3]This was an unfortunate oversight on our part; the tutorials used the older version of the UI without the new features.

evaluate SEAM-EZ capabilities in metric creation and validation. These case studies include a video streaming use case, a website browsing use case, and a fitness tracking use case. We chose them because they represent a diverse set of common use cases for stateful analytics [35].

7.1 Participant Demographics and Study Protocol

We recruited 21 participants (labeled P1-P21) via email invitations within the same organization. There were three female and 18 male participants. Their roles included data scientists, machine learning engineers, software engineers, and product managers. 19 of 21 participants had at least one year of experience with SQL. These participants matched the profiles of our main target users who have domain expertise (e.g., video streaming) and some technical knowledge such as SQL but lack advanced expertise in SQL or Spark/Flink or streaming SQL that is typically required to write stateful analytics. Table 1 in the Appendix summarizes the demographics of our participants.

The session for each participant lasted about one hour. Some of the sessions were done in person while the others were done online via Zoom. Each session consisted of the following study components: (1) a background interview (10 minutes) that asks about their prior experiences with metrics; (2) short video tutorials (15 minutes) about the Timeline concepts [35] and SEAM-EZ, the tool itself; (3) one metric task with think-aloud (15 minutes); and (4) a post-task interview that asks about their experiences with and feedback on SEAM-EZ (10 minutes). Every participant was randomly assigned to one of the three metric tasks.

After the case studies, we kept SEAM-EZ available to all people at the organization and any of them could provide their feedback on SEAM-EZ via a quick feedback survey, mainly asking for tool improvement suggestions (see Appendix). We also sent a more detailed exit survey to our case study participants asking them to rate their experiences using SEAM-EZ and MySQL Workbench, one of the most popular SQL query tools, in various aspects (see Appendix). We decided not to ask participants to create a similar metric using another tool (e.g., a SQL query editor) because we found it took too long, which would make the study less practical to our participants who are busy professionals.

7.2 Case Study Use Cases

We included three realistic use cases in our case study. The use case datasets and their corresponding metric DAGs are included in the Appendix. These tasks represent an easy and medium level of complexity that we observed in common real-world stateful analytics from our target users in different industries. While SEAM-EZ could support more complex stateful metrics, see a detailed discussion and an example (Figure 13) in the Appendix, we decided not to include complex tasks in the case studies due to the practical time limit of these study sessions with busy working professionals.

CS1. Video Streaming. This use case represents both system states (e.g., a video player is playing) and events (e.g., a video player switched to a different network) as well as user events (e.g., a user seeks/skips in a video) in the context of online video streaming. We asked our participants to create a metric to compute "*the total time spent in buffering while using CDN1 and on Cellular Network.*" We invited seven participants (P1, P4, P6, P7, P13, P16, and P19) in CS1.

CS2. Website Browsing. The second use case focuses on users browsing websites. The dataset includes browsing events such as page view on a particular web page. We asked our participants to create a metric to compute "the average time spent on a particular page (demo.com/index.html) for a session (note: the data is from a single session)." We invited seven participants (P2, P5, P8, P11, P14, P17, and P20) in CS2.

CS3. Fitness Tracking. The third use case represents a person monitoring his or her own fitness using a tracker (e.g., Fitbit). The dataset includes events such as the person performing various activities such as resting, running, eating, and working, as well as the stress level and heart rate. We asked our participants to create a metric to compute "how long was this person in a "high" range (120 and higher bpm) for heart rate while running in a session (note: the data is from a single session)?" We invited seven participants (P3, P9, P10, P12, P15, P18, and P21) in CS3.

8 FINDINGS AND DISCUSSION

Through observing our participants performing the metric creation tasks in the three case studies, we found that SEAM-EZ helped support fast prototyping and validation of stateful metrics. Overall, participants were able to create the metrics in the tasks. Next, we discuss our findings from the exit survey, and major insights from the case studies.

8.1 SEAM-EZ vs. SQL Editor: More Intuitive and Less Time-Consuming

More Intuitive. Our participants unanimously felt that SEAM-EZ is intuitive in creating stateful metrics in part thanks to its graphic interfaces and affordances (e.g., direction manipulations of computational operators) via visual programming. For instance, P19 "SEAM-EZ *is what you see is what you get, it is very intuitive. For other tools, like Navicat, it needs the user have very good experience to SQL or DSL knowledge.*" P4 elaborated "SEAM-EZ *is graph based. Very clear to see each state. SQL is showing everything in numbers/texts/datatypes.*" [4] P18 complimented on the overall process of creating a metric similar to creating a flow chart as well as the charts (data previews) of steps in the process. "*The design is good. The charts are intuitive. The interaction is flexible. It is like a flow chart.*" P1 also echoed this point "*I can see the intermediate results using plots which are intuitive.*" P16 highlighted that the visual approach is particularly useful for individuals with less technical background"*It is graphical so very intuitive...It would be a good tool for nontechnical people.*" P10 pointed out the inclusiveness of SEAM-EZ: "*SQL is not for everyone. You need to have SQL knowledge.* SEAM-EZ *is definitely for everyone. You don't need SQL knowledge.*"

Less Time-Consuming. All but three participants felt writing SQL queries would take much longer than using SEAM-EZ to create metrics. For example, P1 explained he could see intermediate results from SEAM-EZ (data previews) but "*it'll take a longer time to verify the intermediate results. It is because I can not see the intermediate result and need to verify the sub queries.*" Others felt writing SQL

[4] Navicat is an example of a visual frontend to SQL [37].

would be more difficult and time-consuming. Similarly, P9 commented "*...with help from google or chatgpt in SQL I would struggle quite a bit to write that query and likely needed help from google...*" Some participants felt the SQL approach would take longer to validate the metrics. For instance, P2 hinted at the usefulness of the auto-suggest and timeline preview features in SEAM-EZ that other tools do not provide: "*SQL will take more time to validate and [it] misses some auto-suggest for operation. Compared to other tools like python, I don't have a quick visual response.*"

However, some of the users were less enthusiastic about SEAM-EZ in some cases; e.g., P4, P14, and P15 felt SQL would take less time under certain situations because they were very familiar with SQL and/or SEAM-EZ lacks support for what they need to do. For instance, while P4 felt in general drawing DAGs is better than writing SQL for creating metrics, he needed SEAM-EZ to directly support more timeline flexibility, e.g., 5th minute of every session. Right now, SEAM-EZ does not have built-in support for that and he needs to write SQL for that. For P14, while he only has 1.5 years of experience with SQL, he often uses SQL in his current work. Therefore, he is very used to the SQL or tabular way of thinking. Initially, he had troubles differentiating two novel operators that SEAM-EZ introduced: GetState vs. GetEvent. He went back to the tutorial and eventually figured them out. But he felt using SQL would takes him less time because he did not need to pick up the new operators. For P15, while he commented SEAM-EZ is intuitive and he can use it to build the metric from scratch, he also noted "*The [*SEAM-EZ *] operators cannot fully support my daily work*" and he gave examples such as average, min, and max. Our latest version of SEAM-EZ has since then added those operators. P11 thought it is more straightforward in SQL in terms of how to manipulate data. After all, users need to learn a new computational framework (Timeline) and its novel operators (e.g., GetState) to be able to use SEAM-EZ.

More Transparent and Greater Control. Figure 12 shows the quantitative results of our exit survey. We ran non-parametric Wilcoxon signed rank sum tests to evaluate the difference between SEAM-EZ and the MySQL Workbench in the responses of our seven-point Likert scale exit survey questions while considering the ratings are repeated measures because each participant gave two ratings one for an aspect of each tool. We found significant effects where users rated SEAM-EZ better (higher) in all five aspects: the user satisfying with the final results ($V = 5, p < .0001$), the system helping think through the outputs ($V = 2.5, p < .0001$), the system being transparent about how it arrives the final results ($V = 7, p < .0001$), the user feeling in control when creating with the system ($V = 7, p < .0001$), the user feeling that s/he was collaborating with the system to come up with the outputs ($V = 3.5, p < .0001$).

Context-dependent choice of tool. Participants had various preferences for SEAM-EZ or the SQL editor. For instance, P19 felt "SEAM-EZ *is more natural and intuitive, for SQL I need to be careful and think more.*" Others preferred SEAM-EZ for time-based analyses. For instance, P12 elaborated "*For scenarios that are time-related and can conveniently utilize time-series databases, I believe* SEAM-EZ *has great potential because it's inherently designed for these computing scenarios. For instance, when we collect a large amount of events information reported by IOT devices, wanting to batch filter useful information based on certain conditions is a perfect scenario. However, using SQL might involve a lot of windowing, which can complicate the problem.*" By windowing, this participant was referring to calculation across a set of table rows that are somehow related to the current row. P9 echoed "SEAM-EZ *for anything where I need to look at sequences of events or durations in certain states. SQL if I already have metrics and I can just query based on those metrics or for example ratios of those metrics.*" Others made tool choices based on the expected complexity of the metrics. For instance, P5 commented "SEAM-EZ *is for basic data analysis, testing data cleaning pipelines. SQL is for building complex metrics involving operations not available in* SEAM-EZ*, for code sharing across org.*" It is important to note that metrics created in SEAM-EZ can be stored and shared with others in the same organization. P15 noted the benefits of SEAM-EZ in terms of communication among teams. "*For* SEAM-EZ*, demo the metrics building progress to others, let others have the confident that the metric is right and for beginners of SQL and users that don't have SQL experience,* SEAM-EZ *is better. But for very complex scenario, SQL is better.*" Delving deeper, we realized that he did not think SEAM-EZ would support creation of complex metrics because the DAG will get too complicated or busy quickly. However, SEAM-EZ does support zoom in/our of the DAG. Since this was a first attempt at building a visual programming system for stateful analytics, we focused on relatively simple tasks. This is an interesting direction for future work to evaluate the utility of SEAM-EZ on more complex stateful analytics.

8.2 SEAM-EZ Supports Stateful Data Explorations

In the process of creating stateful metrics, our participants often started with exploring the dataset. They commented on how SEAM-EZ supports easy stateful data explorations. For instance, P15 said "*It is more intuitive. If using Prometheus, I need to understand the metrics first then I can use it. But for using* SEAM-EZ*, it is like an exploration tool. Build the metric from scratch.*"[5] P17 shared a similar sentiment "[SEAM-EZ] is easy to explore data but user has to write raw SQL in [SQL editor]. Cannot estimate it directly." P4 explained how he would explore the data "*If I have no idea how to start a metric I'd try to explore it in* SEAM-EZ *to get some idea. If it's pretty straightforward how to calculate I would use SQL.*"

8.3 SEAM-EZ Supports Rapid Prototyping of Stateful Metrics

Our participants also thought that SEAM-EZ supports rapid prototyping of stateful metrics in part thanks to its visual (programming) mode. P9 was very enthusiastic about SEAM-EZ: "*doing this visually helps massively. I have done similar in the past with SQL and it is very very hard. I am not a SQL developer just know some SQL).*" P13 added that the visual approach also helped with seeing the bigger picture of creating metrics as a pipeline: "*As you can see, use* SEAM-EZ *can go through the whole pipeline very clear. But use SQL only can image in my mind.*" P20 shared a similar thought: "*At first I was thinking how to build the timeline. But writing SQL is the same. I need to think as well. This tool visualize the process.*" P19 added another benefit of the visual representation, i.e., reducing communication costs "*If I*

[5]Prometheus is a popular open-source platform used by system operators to visualize real time metrics on a dashboard [4].

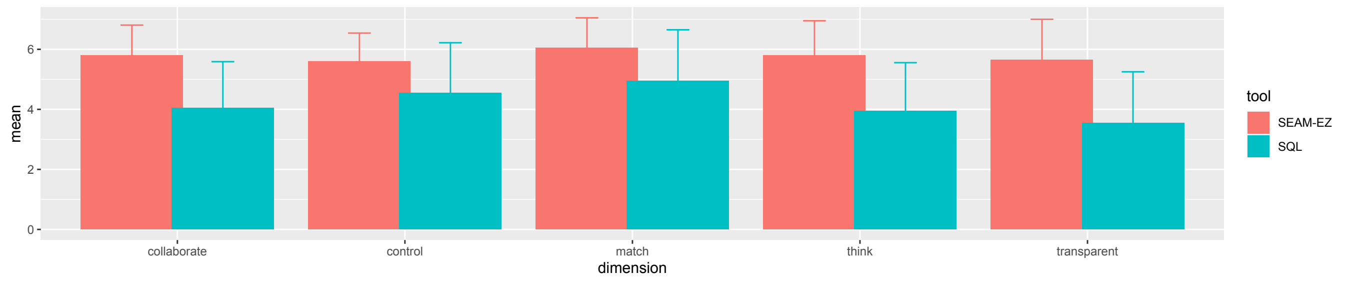


Figure 12: SEAM-EZ **was rated better (higher) than the SQL editor in all aspects: the user satisfying with the final results, the system helping the user think through the outputs, the system being transparent about how it arrives the final results, the user feeling in control when creating with the system, the user feeling that s/he was collaborating with the system to come up with the outputs.**

*want to describe the building process for the metric, using this tool [*SEAM-EZ *] can help reduce the communication cost.*" Furthermore, P12 spoke positively about the data preview feature "*Using visual tools to preview data is always more intuitive and quicker than writing SQL directly. Moreover, the process control is less likely to cause errors.*"

8.4 SEAM-EZ Supports Validation of Stateful Metrics

After creating stateful metrics, our participants also were tasked to validate the metrics they created. They spoke about their experience in using SEAM-EZ to perform the validation. For instance, P17 said he "*calculated the data and charts of the outputs.*" Similarly, P21 used the data preview of the intermediate outputs "*I can see the intermediate outputs. If I use codes, it is not that intuitive.*" P20 also went through the entire timeline to validate the metric: "*Compared with SQL, there is more certainty using this tool. I need the review from other people if I write SQL. I can read through the timeline to validate the process by myself.*" However, P19 had a concern about showing the data when it's a large dataset. Alternatively, they (e.g., P17) would write (SQL or Python) code to validate the results.

8.5 Expected Usage of SEAM-EZ

We asked our participants what scenarios they would use SEAM-EZ to answer. Many participants simply said any stateful metrics (e.g., P5, P9, P13, P20) or time-stamped data (e.g., P8). Others talked about data exploration (e.g., P6), system monitoring (e.g., P14) and diagnosis (e.g., P4). Some participants gave specific use cases. For instance, P7 would use it to analyze users such as Daily Active Users (DAU), the total number of sessions, and average duration per session. Similarly, P17 would use it to calculate the conversion of users in ecommerce. P11 instead would use it to calculate psychological measures (e.g., depression). P15 felt SEAM-EZ misses some operators such as Avg, Min, and Max that his daily work needs.

8.6 Improvement Suggestions

While SEAM-EZ and its features (e.g., drag-and-drop, data previews, tooltips) were largely well received by our participants, they did provide suggestions on how to improve the current version of SEAM-EZ.

More support on learning about a new computational framework. SEAM-EZ was built upon Milner et. al's Timeline framework for stateful analytics [35]. As such, a key design challenge of SEAM-EZ is that it needs to introduce concepts of the Timeline framework and its operators to users. In our case studies, we observed that it took some time for many participants to figure out the specific Timeline operators: GetState, GetEvent, HasBeenTrue, and DurationTrue. In addition to our video tutorials, starting page/template and timeline data preview, the operator tooltips and animations were reported particularly useful in helping participants understand these concepts. However, there is still room for improvement on the education aspect. Their ideas included an on-boarding wizard that walks users step-by-step through the metric creation process in SEAM-EZ. They also would like additional ready-to-use metric templates in SEAM-EZ.

Improvements on usability and new features. Our participants also recommended a number of improvements for SEAM-EZ in terms of usability. For instance, the suggested only triggering the auto-suggest operator list when users drag an edge from a node/operator, automatically updating the final metric value if the metric DAG is updated, and providing error messages when users make mistakes. In addition, they suggested adding a few new features, such as, showing the SQL code equivalent of a DAG alongside the DAG, as well as supporting multiple DAGs on the canvas, more time flexibility (e.g., the 5th minute of every session regardless of the clock time), event time comparisons (e.g., event A happens before event B), and additional validation features (e.g., show the number of records filtered out by applying a comparison/condition). These are all useful ideas for future versions of SEAM-EZ.

Creating more advanced or composite DAGs. Our current focus is in establishing the utility of SEAM-EZ for stateful analytics tasks for our main target users that would be exceedingly difficult in existing frameworks (e.g., SQL, Spark). In the future to support even more complex analytics and more advanced use cases, we envision adding features for users to create user-defined templates to further accelerate their work. For instance, if they find the same set of data preprocessing tasks on the same dataset to create timelines of interest, they can create user-defined composite operators that compose existing low-level operations for common processing patterns.

8.7 Design Reflections on Visual Programming Tools for Stateful Metrics

Benefits. We see that several participants commended SEAM-EZ for making stateful metric creation easier and faster for them who used SQL to do so in the past. Perhaps, equally important, SEAM-EZ makes progress in democratizing metric creation because it enables those (e.g., P6, P12) who could not traditionally create metrics (using SQL or other streaming frameworks) to be able to create metrics by drawing DAGs rather than writing code.

What to do. In terms of designing visual programming tools for stateful analytics, our design experience and case studies suggest that a number of principles are highly useful. For instance, it is valuable to show the direct mapping between visual programming constructs (DAG) and the underlying data. This is related to the high-level design goal (G2: support users to understand the dataset) that we identified from our foundational user study. One concrete feature we implemented was the data preview feature for each operator in the DAG pipeline so that users can see what the operator does to transform the underlying data. We believe that this principle is relevant to any visual programming tools for data analytics. Another useful principle is providing semantics of visual programming constructs, for instance, nodes or timeline operators in SEAM-EZ. This is particularly needed in our case because the underlying Timeline framework and many of its operators are new to our target users. One concrete feature we implemented was tooltips with animation for each operator. Another feature we implemented was disallowing incompatible operators to be connected to avoid errors. We believe this principle is relevant for any visual programming tools as long as the nodes and the edges represent new concepts to the users.

What not to do. Based on our case studies, the following should be carefully considered or even avoided when designing visual programming tools for stateful analytics. For instance, the visual programming tool design should avoid inserting suggestions that interrupt users' typical workflow. In our case, while our participants felt positive about the general idea of auto-suggest of operators in a DAG, our current implementation triggered such auto-suggestions when users clicks on an operator node in the DAG. In the case studies, we observed that sometimes users did not mean to create a new node but the auto-suggest feature was triggered, which confused the users. Another thing that visual programming tools for stateful analytics should avoid is to expose advanced concepts/operators as visual programming constructs (e.g., nodes) to novice users. For instance, a few participants initially were confused about the differences between two new (Timeline) operators: GetState vs. GetEvent. The current SEAM-EZ exposes both operators to all users. We observed that GetState was used more often than GetEvent in real-world stateful metrics. To follow this principle, SEAM-EZ could expose GetState to users and hide GetEvent in an advanced operator category or tab. Since event streams and states are fundamental elements of stateful analytics, we believe this principle is relevant for any visual programming tools for stateful analytics.

8.8 Limitations and Future Work

Our user-centered research, design and evaluation have a number of limitations. First, we did not compare SEAM-EZ with an alternative end to end tool that may have similar UIs as SEAM-EZ but without the node-graph editor. Therefore, we cannot accurately and quantitatively measure the effect of the node-graph editor of SEAM-EZ on participants' experience in metric creation and validation. However, qualitatively we heard our participants liking the node-graph editor and its intuitiveness as we reported earlier. Furthermore, all (30+) participants in our work were from the same large corporation. While their job roles were diverse and they channeled in what they observed from their external customers about creating stateful analytics, future work would surely benefit from involving practitioners from different organizations.

Second, our case studies of SEAM-EZ mainly yield quantitative Likert-scale ratings and qualitative insights about the SEAM-EZ and the SQL counterpart. Conducting large-scale quantitative comparisons would be ideal but very challenging to do in practice. We had planned to have participants perform metric creation tasks using both SEAM-EZ and SQL. However, in our pilots, several participants either said they do not know how to write SQL for the stateful metrics or they would need a lot more time than the one hour that each study session had. In the end, we decided to have participants comment on SQL similar to how prior work [19] studied the comparison. In addition, the tasks in our case studies did not cover the most complex stateful metrics we have observed in practice. We decided not to include those more complex tasks because of the pragmatics of running study sessions that lasted only one hour with busy professionals. Because SEAM-EZ is a new system and thus some participants might say good things about it because of the novelty effect. However, the potential bias could be the other way around because of the status quo bias. Future research with more resources could perform longitudinal studies to examine practitioners' preferences over time.

In addition to our participants' suggestions, we believe that there are promising future directions for a VP system like SEAM-EZ. First, the system can have more features to support metric explanation and validation. For instance, the system could generate a textual explanation of a metric DAG and the user can check whether the explanation matches his or her expectation of the DAG, e.g., as a way to check whether the user has created the correct metric or whether the user should use the metric as expected. Second, given the rise of large language models (LLMs), the system could leverage LLMs to directly generate metric DAGs based on a user's description of the metric. In fact, our preliminary investigation suggested that it is possible for LLMs to generate DAGs, but usually the DAGs are not completely correct, so the user needs to modify or refine them. This would be similar to using and adapting an existing DAG or metric template.

9 CONCLUSION

Stateful analytics is pervasive across numerous application domains ecommerce, social media, fintech, cybersecurity, and IoT. Today, however, creating stateful metrics is tedious, time-consuming, and error prone. We present the iterative design and evaluation of

SEAM-EZ, a no-code visual programming platform for stateful metrics creation and validation. Our case studies show that SEAM-EZ supports flexible explorations of time-based data as well as rapid creation and validation of stateful metrics. Future work can further improve systems like SEAM-EZ by leveraging large language models (LLMs) for automatically generating metric DAGs based on metric description and vice versa.

ACKNOWLEDGMENTS

We thank Pai Liu and Zhan Yang for assisting the frontend implementation, and Qichu Gong for the help on complex DAGs. We thank Aditya Ganjam, Jibin Zhan, Smart Sun, Ningning Hu and Hui Zhang for their insights into the problem space. We thank our participants for sharing their insights and anonymous reviewers for their thoughtful feedback.

REFERENCES

[1] [n. d.]. Diving Deeper into Psyberg: Stateless vs Stateful Data Processing.
[2] [n. d.]. Create AR effects and filters. https://effecthouse.tiktok.com/
[3] [n. d.]. Maya Software | Get Prices & Buy Official Maya 2024 | Autodesk. https://www.autodesk.com/products/maya/overview
[4] [n. d.]. Prometheus Monitoring System. https://prometheus.io/.
[5] [n. d.]. PromptChainer: Chaining Large Language Model Prompts through Visual Programming | Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems. https://dl.acm.org/doi/10.1145/3491101.3519729
[6] [n. d.]. Scratch - Imagine, Program, Share. https://scratch.mit.edu/
[7] [n. d.]. Teachable Machine: Approachable Web-Based Tool for Exploring Machine Learning Classification | Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems. https://dl.acm.org/doi/10.1145/3334480.3382839
[8] [n. d.]. Visual programming, programming by example, and program visualization: a taxonomy | Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. https://dl.acm.org/doi/10.1145/22627.22349
[9] Mohammed Ali, Ali Alqahtani, Mark W. Jones, and Xianghua Xie. 2019. Clustering and Classification for Time Series Data in Visual Analytics: A Survey. *IEEE Access* 7 (2019), 181314–181338. https://doi.org/10.1109/ACCESS.2019.2958551 Conference Name: IEEE Access.
[10] Yining Cao, Jane L E, Zhutian Chen, and Haijun Xia. 2023. DataParticles: Block-based and Language-oriented Authoring of Animated Unit Visualizations. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3544548.3581472
[11] Paris Carbone et al. 2015. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society TCDE* 36, 4 (2015).
[12] Qi Alfred Chen et al. 2014. QoE Doctor: Diagnosing Mobile App QoE with Automated UI Control and Cross-layer Analysis. In *Proc. ACM SIGCOMM IMC.*
[13] Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar, Thomas Graves, Mark Holderbaugh, Zhuo Liu, Kyle Nusbaum, Kishorkumar Patil, Boyang Jerry Peng, et al. 2016. Benchmarking streaming computation engines: Storm, flink and spark streaming. In *2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*. IEEE, 1789–1792.
[14] John Joon Young Chung, Wooseok Kim, Kang Min Yoo, Hwaran Lee, Eytan Adar, and Minsuk Chang. 2022. TaleBrush: Sketching Stories with Generative Pretrained Language Models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems (CHI '22)*. Association for Computing Machinery, New York, NY, USA, 1–19. https://doi.org/10.1145/3491102.3501819
[15] Tomaz Curk, Janez Demsar, Qikai Xu, Gregor Leban, Uros Petrovic, Ivan Bratko, Gad Shaulsky, and Blaz Zupan. 2005. Microarray data mining with visual programming. *Bioinformatics* 21, 3 (Feb. 2005), 396–398. https://doi.org/10.1093/bioinformatics/bth474
[16] Zikun Deng, Di Weng, Shuhan Liu, Yuan Tian, Mingliang Xu, and Yingcai Wu. 2023. A survey of urban visual analytics: Advances and future directions. *Computational Visual Media* 9, 1 (2023), 3–39. https://doi.org/10.1007/s41095-022-0275-7
[17] Griffin Dietz, Nadin Tamer, Carina Ly, Jimmy K Le, and James A. Landay. 2023. Visual StoryCoder: A Multimodal Programming Environment for Children's Creation of Stories. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. Association for Computing Machinery, New York, NY, USA, 1–16. https://doi.org/10.1145/3544548.3580981
[18] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. 2011. Understanding the impact of video quality on user engagement. In *Proc. ACM SIGCOMM 2011 Conf.*
[19] Ruofei Du, Na Li, Jing Jin, Michelle Carney, Scott Miles, Maria Kleiner, Xiuxiu Yuan, Yinda Zhang, Anuva Kulkarni, and Xingyu Liu. 2023. Rapsai: Accelerating Machine Learning Prototyping of Multimedia Applications through Visual Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–23.
[20] Opher Etzion, Mani Chandy, Rainer von Ammon, and Roy Schulte. 2006. Event-Driven Architectures and Complex Event Processing. In *2006 IEEE International Conference on Services Computing (SCC'06)*. xxx–xxx. https://doi.org/10.1109/SCC.2006.49
[21] Blender Foundation. [n. d.]. blender.org - Home of the Blender project - Free and Open 3D Creation Software. https://www.blender.org/
[22] Primož Godec, Matjaž Pančur, Nejc Ilenič, Andrej Čopar, Martin Stražar, Aleš Erjavec, Ajda Pretnar, Janez Demšar, Anže Starič, Marko Toplak, Lan Žagar, Jan Hartman, Hamilton Wang, Riccardo Bellazzi, Uroš Petrovič, Silvia Garagna, Maurizio Zuccotti, Dongsu Park, Gad Shaulsky, and Blaž Zupan. 2019. Democratized image analytics by visual programming through integration of deep models and small-scale machine learning. *Nature Communications* 10, 1 (Oct. 2019), 4551. https://doi.org/10.1038/s41467-019-12397-x Number: 1 Publisher: Nature Publishing Group.
[23] Liang Gou, Lincan Zou, Nanxiang Li, Michael Hofmann, Arvind Kumar Shekar, Axel Wendt, and Liu Ren. 2021. VATLD: A Visual Analytics System to Assess, Understand and Improve Traffic Light Detection. *IEEE transactions on visualization and computer graphics* 27, 2 (Feb. 2021), 261–271. https://doi.org/10.1109/TVCG.2020.3030350
[24] Devamardeep Hayatpur, Daniel Wigdor, and Haijun Xia. 2023. CrossCode: Multi-level Visualization of Program Execution. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3544548.3581390
[25] Wenbin He, Lincan Zou, Arvind Kumar Shekar, Liang Gou, and Liu Ren. 2022. Where Can We Help? A Visual Analytics Approach to Diagnosing and Improving Semantic Segmentation of Movable Objects. *IEEE transactions on visualization and computer graphics* 28, 1 (Jan. 2022), 1040–1050. https://doi.org/10.1109/TVCG.2021.3114855
[26] Gaoping Huang, Pawan S. Rao, Meng-Han Wu, Xun Qian, Shimon Y. Nof, Karthik Ramani, and Alexander J. Quinn. 2020. Vipo: Spatial-Visual Programming with Functions for Robot-IoT Workflows. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376670
[27] Dan Jamieson. 2021. The Growing Power of IoT in Preventative Maintenance. https://bit.ly/3pXYRvd.
[28] Christian S. Jensen and Richard T. Snodgrass. 2009. Temporal Query Languages. In *Encyclopedia of Database Systems*, LING LIU and M. TAMER ÖZSU (Eds.). Springer US, Boston, MA, 3009–3012. https://doi.org/10.1007/978-0-387-39940-9_407
[29] Peiling Jiang, Fuling Sun, and Haijun Xia. 2023. Log-it: Supporting Programming with Interactive, Contextual, Structured, and Visual Logs. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. Association for Computing Machinery, New York, NY, USA, 1–16. https://doi.org/10.1145/3544548.3581403
[30] David Johnson, Giuseppe Carenini, and Gabriel Murray. 2020. NJM-Vis: interpreting neural joint models in NLP. In *Proceedings of the 25th International Conference on Intelligent User Interfaces (IUI '20)*. Association for Computing Machinery, New York, NY, USA, 286–296. https://doi.org/10.1145/3377325.3377513
[31] G. Karsai. 1995. A configurable visual programming environment: a tool for domain-specific programming. *Computer* 28, 3 (March 1995), 36–44. https://doi.org/10.1109/2.366147 Conference Name: Computer.
[32] Dave Krebs, Alexander Conrad, and Jingtao Wang. 2012. Combining visual block programming and graph manipulation for clinical alert rule building. In *CHI '12 Extended Abstracts on Human Factors in Computing Systems (CHI EA '12)*. Association for Computing Machinery, New York, NY, USA, 2453–2458. https://doi.org/10.1145/2212776.2223818
[33] Jingyi Li, Joel Brandt, Radomír Mech, Maneesh Agrawala, and Jennifer Jacobs. 2020. Supporting Visual Artists in Programming through Direct Inspection and Control of Program Execution. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3313831.3376765
[34] Franco Milicchio, Rebecca Rose, Jiang Bian, Jae Min, and Mattia Prosperi. 2016. Visual programming for next-generation sequencing data analytics. *BioData Mining* 9, 1 (April 2016), 16. https://doi.org/10.1186/s13040-016-0095-3
[35] Henry Milner, Yihua Cheng, Jibin Zhan, Hui Zhang, Vyas Sekar, Junchen Jiang, and Ion Stoica. 2023. Raising the Level of Abstraction for Time-State Analytics With the Timeline Framework. In *13th Annual Conference on Innovative Data Systems Research (CIDR '23)*.
[36] Sugeerth Murugesan, Sana Malik, Fan Du, Eunyee Koh, and Tuan Manh Lai. 2019. DeepCompare: Visual and Interactive Comparison of Deep Learning Model Performance. *IEEE Computer Graphics and Applications* 39, 5 (Sept. 2019), 47–59. https://doi.org/10.1109/MCG.2019.2919033 Conference Name: IEEE Computer

Graphics and Applications.
[37] Navicat. [n. d.]. Navicat GUI | DB Admin Tool for MySQL, Redis, PostgreSQL, MongoDB, MariaDB, SQL Server, Oracle & SQLite client. https://navicat.com/en/
[38] Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, and Kaushik Veeraraghavan. 2015. Gorilla: a fast, scalable, in-memory time series database. *Proceedings of the VLDB Endowment* 8, 12 (Aug. 2015), 1816–1827. https://doi.org/10.14778/2824032.2824078
[39] Alex Repenning. 1993. Agentsheets: a tool for building domain-oriented visual programming environments. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems (CHI '93)*. Association for Computing Machinery, New York, NY, USA, 142–143. https://doi.org/10.1145/169059.169119
[40] Kurt J. Schmucker. 1996. Rapid prototyping using visual programming tools. In *Conference Companion on Human Factors in Computing Systems (CHI '96)*. Association for Computing Machinery, New York, NY, USA, 359–360. https://doi.org/10.1145/257089.257368
[41] Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cícero Nogueira dos Santos, and Bing Xiang. 2020. Learning Contextual Representations for Semantic Parsing with Generation-Augmented Pre-Training. *CoRR* abs/2012.10309 (2020). arXiv:2012.10309 https://arxiv.org/abs/2012.10309
[42] Seungwon Shin, Zhaoyan Xu, and Guofei Gu. 2012. EFFORT: Efficient and Effective Bot Malware Detection. In *Proc. INFOCOM*.
[43] Thilo Spinner, Udo Schlegel, Hanna Schäfer, and Mennatallah El-Assady. 2020. explAIner: A Visual Analytics Framework for Interactive and Explainable Machine Learning. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (Jan. 2020), 1064–1074. https://doi.org/10.1109/TVCG.2019.2934629 Conference Name: IEEE Transactions on Visualization and Computer Graphics.
[44] Susan Leigh Star and James R. Griesemer. 1989. Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. *Social Studies of Science* 19, 3 (1989), 387–420. https://www.jstor.org/stable/285080 Publisher: Sage Publications, Ltd..
[45] William Robert Sutherland. 1966. *The on-line graphical specification of computer procedures.* Thesis. Massachusetts Institute of Technology. https://dspace.mit.edu/handle/1721.1/13474 Accepted: 2005-09-21T22:40:43Z.
[46] Masahiro Takatsuka and Mark Gahegan. 2002. GeoVISTA Studio: a codeless visual programming environment for geoscientific data analysis and visualization. *Computers & Geosciences* 28, 10 (Dec. 2002), 1131–1144. https://doi.org/10.1016/S0098-3004(02)00031-6
[47] Justin Talbot, Bongshin Lee, Ashish Kapoor, and Desney S. Tan. 2009. EnsembleMatrix: interactive visualization to support machine learning with multiple classifiers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. Association for Computing Machinery, New York, NY, USA, 1283–1292. https://doi.org/10.1145/1518701.1518895
[48] Ian Tenney, James Wexler, Jasmijn Bastings, Tolga Bolukbasi, Andy Coenen, Sebastian Gehrmann, Ellen Jiang, Mahima Pushkarna, Carey Radebaugh, Emily Reif, and Ann Yuan. 2020. The Language Interpretability Tool: Extensible, Interactive Visualizations and Analysis for NLP Models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, 107–118. https://doi.org/10.18653/v1/2020.emnlp-demos.15
[49] Riccardo Terrell. [n. d.]. Real-Time Stream Analysis in Functional Reactive Programming. https://www.infoq.com/presentations/stream-analysis-fp/
[50] James Wexler, Mahima Pushkarna, Tolga Bolukbasi, Martin Wattenberg, Fernanda Viégas, and Jimbo Wilson. 2020. The What-If Tool: Interactive Probing of Machine Learning Models. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (Jan. 2020), 56–65. https://doi.org/10.1109/TVCG.2019.2934619 Conference Name: IEEE Transactions on Visualization and Computer Graphics.
[51] Tongshuang Wu, Michael Terry, and Carrie Jun Cai. 2022. Ai chains: Transparent and controllable human-ai interaction by chaining large language model prompts. In *Proceedings of the 2022 CHI conference on human factors in computing systems*. 1–22.
[52] Xiwei Xuan, Xiaoyu Zhang, Oh-Hyun Kwon, and Kwan-Liu Ma. 2022. VAC-CNN: A Visual Analytics System for Comparative Studies of Deep Convolutional Neural Networks. *IEEE Transactions on Visualization and Computer Graphics* 28, 6 (June 2022), 2326–2337. https://doi.org/10.1109/TVCG.2022.3165347 Conference Name: IEEE Transactions on Visualization and Computer Graphics.
[53] Y. Zhang, M. Ward, N. Hachem, and M. Gennert. 1993. A visual programming environment for supporting scientific data analysis. In *Proceedings 1993 IEEE Symposium on Visual Languages*. 368–370. https://doi.org/10.1109/VL.1993.269562

A CASE STUDY PARTICIPANTS

Table 1 summarizes the demographics of our case study participants.

PID	Gender	Role	Task	Age	SQL experience (years)
P1	Male	Data scientist	T1	35-44	10
P2	Female	Machine learner	T2	25-30	1-2
P3	Male	Software engineer	T3	35-44	10
P4	Male	Data scientist	T1	25-34	8
P5	Female	Data scientist	T2	25-34	6
P6	Male	Machine learner	T1	25-34	0.16 (two months)
P7	Male	Data scientist	T1	25-34	8
P8	Male	Software engineer	T2	25-34	6
P9	Male	Software engineer	T3	25-34	10-15
P10	Male	Product manager	T3	35-44	1
P11	Male	Machine learner	T2	25-34	1-2
P12	Female	Product manager	T3	25-34	0
P13	Male	Software engineer	T1	25-34	5
P14	Male	Software engineer	T2	25-34	1.5
P15	Male	Software engineer	T3	25-34	over 10
P16	Male	Software engineer	T1	35-44	15
P17	Male	Engineer manager	T2	35-44	over 10
P18	Male	Software engineer	T3	25-34	5-6
P19	Male	Engineer manager	T1	35-44	almost 20
P20	Male	Software engineer	T2	25-34	3-4
P21	Male	Software engineer	T3	35-44	3

Table 1: Demographics of Case Study Participants

B SEAM-EZ CAN SUPPORT MORE COMPLEX STATEFUL METRICS

Thus far, we have shown metrics of low and medium level complexity with a selected list of operators that were sufficiently effective for user studies. However, SEAM-EZ can easily support more complex stateful analytics for practitioners in the industry with more operators implemented under the hood (e.g., if operator). For example, Figure 13 is a complex metric that we use in production. The input data source is raw events data collected on user activity from one specific user on a web or mobile application equipped with video players. Based on the user interaction in the app, we are interested in counting the number of page view changes (node #25) and how long the user stayed on that page (node #26). At the same time, we have the notion of a session, which is a time range where all of the time difference between consecutive user actions is less than one minute (e.g., if a user is inactive for more than one minute, we end and renew the session). This can be captured by `HasBeenTrue` (node #19) with a sliding window capability. We can count the number of sessions (node #30) and the duration of each session (node #31). One step further, the user actions can be controlled in a more fine-grained fashion. If (node #17) the user is watching the video (node #4), we consider various video-related activities (node #5 - #10) as user actions. If not, we should use other events (node #3) as user actions ignoring (#14, #15) system-generated log events (#11, #12).

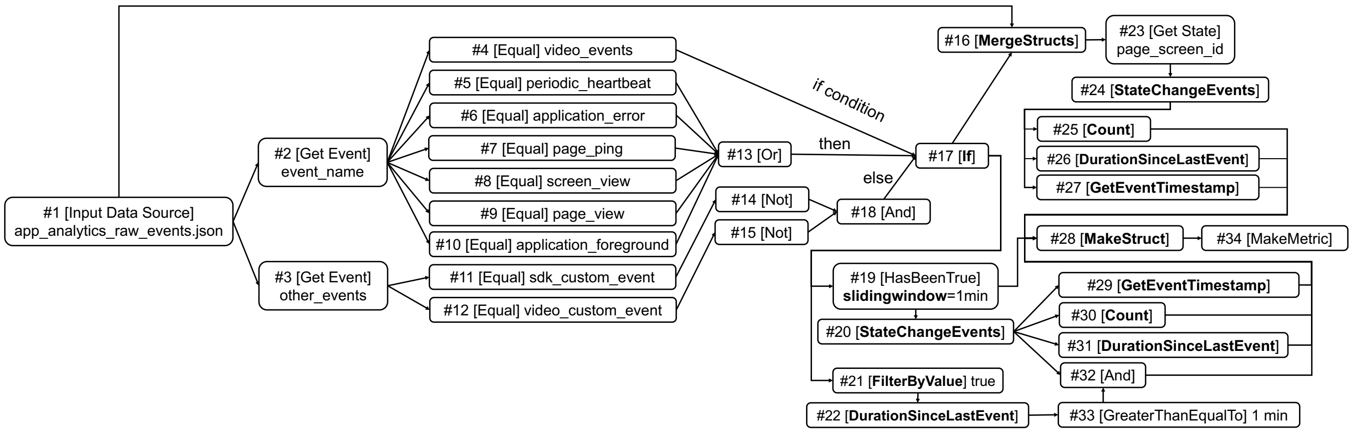

#1 [Input Data Source] app_analytics_raw_events.json
#2 [Get Event] event_name
#3 [Get Event] other_events
#4 [Equal] video_events
#5 [Equal] periodic_heartbeat
#6 [Equal] application_error
#7 [Equal] page_ping
#8 [Equal] screen_view
#9 [Equal] page_view
#10 [Equal] application_foreground
#11 [Equal] sdk_custom_event
#12 [Equal] video_custom_event
#13 [Or]
#14 [Not]
#15 [Not]
#18 [And]
if condition
then
else
#17 [If]
#16 [MergeStructs]
#23 [Get State] page_screen_id
#24 [StateChangeEvents]
#25 [Count]
#26 [DurationSinceLastEvent]
#27 [GetEventTimestamp]
#28 [MakeStruct]
#34 [MakeMetric]
#19 [HasBeenTrue] slidingwindow=1min
#20 [StateChangeEvents]
#29 [GetEventTimestamp]
#30 [Count]
#31 [DurationSinceLastEvent]
#32 [And]
#21 [FilterByValue] true
#22 [DurationSinceLastEvent]
#33 [GreaterThanEqualTo] 1 min

Figure 13: An example of complex stateful metric: count page view changes and measure durations on each page view within a session.

Bold text in Figure 13 means they are more advanced operators that are not covered in section 5.1 [6]. We confirm that SEAM-EZ is readily extensible and generalizable to add more operators and to create these complex metrics in SEAM-EZ, while it is still intuitive to understand and create the DAG. If we compare it to writing SQL query for this complex DAG, writing SQL query is much more complicated and tedious, less intuitive to debug, and difficult to verify (e.g., it will be much worse than Figure 3). On the other hand, SEAM-EZ is more convenient and it takes much less time to finish by intuitive step-by-step visual verification process of SEAM-EZ.

On a separate note, the DAG can be even more complex than Figure 13. This does not imply a limitation of visual programming for stateful analytics because writing SQL queries for more complex DAG will be even more complex. As discussed earlier, we can support complex DAGs in common use cases by enabling composite user-defined operators that can logically "collapse" the DAG.

C STUDY PROTOCOL

C.1 Foundational User Study

Step 1 (background) 10 mins

Thanks for coming! My name is XYZ. I'll be leading today's study session, which will take about 1 hour. I really appreciate you are generous in spending time and sharing your experience with them. The goal of this study is to understand your experience around metrics.

Could we record the meeting and your screen so we could analyze the detailed study data? Thank you!

I'd like to start by knowing a bit more about you.

- What's your role/job? How long have you been working in this kind of role?
- What's your educational background?
- Do metrics play any role in your job? If so, what is it/role?

Step 2 (Metric practice)

- Have you ever done anything with metrics? For instance, creating / debugging / understanding / using metrics?
- When was the last time you created a metric? Could you explain what you did? What was the situation? Why did you need to create a metric? What metric did you create? How did you create it? Did you use any tools? Did you modify the initial metric? How did you use the metric afterwards? Did you work with anyone or you created the metric yourself? Did you encounter any difficulties or challenges in this process?
- When was the last time you debugged or evaluated a metric? Could you explain what you did? What was the situation? How did you do it? Did you use any tools? Did you work with anyone or you created the metric yourself? Did you encounter any difficulties or challenges in this process?
- When was the last time you modified a metric? Could you explain what you did? What was the situation? How did you do it? Did you use any tools? Did you work with anyone or you created the metric yourself? Did you encounter any difficulties or challenges in this process?
- Beside creating, debugging/evaluating, and modifying metrics, are there any other ways that you have interacted with metrics? Could you give me an example?

Now, I want to switch gear and ask your observations of the external customers.

- Which customer have you worked with most closely?
- How do they use existing metrics in the Dashboard? Could you give me a concrete example?
- How do they create metrics in the Dashboard? Could you give me a concrete example?
- How do they ensure the created metrics are correct? Could you give me a concrete example?

[6] `StateChangeEvents` is the exact opposite of `LatestEventToState`, it converts the state timeline to the event timeline. `FilterByValue` is keeping only the events that have the specified value. `durationSinceLastEvent` is similar to `DurationTrue`, it measures the time after any event and it resets to zero at every event. Essentially, it measures time ever since the timeline encounters an event. `Count` is counting the number of events, `GetEventTimestamp` outputs the timestamp of the events. `MergeStructs` and `MakeStruct` is a way to combine multiple data fields into one timeline. `SlidingWindow` is a feature that determines timeline is true or false when there has been true value within the specified time window.

- Do they have difficulties in creating metrics in the Dashboard?

C.2 Formative Evaluations

Step 1 (intro) 3 mins Same as the step 1 in the previous study.

Step 2 (Pre-read) 12 mins

Next, I'd like you to watch the following short video tutorials for onboarding: [videos]. If you'd like more details on these topics, please refer to the detailed references.

Q: any questions about any of these videos? [researcher writes them down]

Step 3 (tasks) 15 mins

Next, we'd like you to conduct some tasks using this tool for about 15 minutes.

[Task description] Imagine that you are tasked to build custom metrics for the following dataset. This is a simplified dataset of events like video heartbeats for a single video session:

The structure of the dataset is as follows. Each heartbeat will have a timestamp/session field/seqnum and optional attributes depending on the measurement:

Fields:
time: Values: timestamp of event
sessionId: Values: 1
newPlayerState: Values: play, pause, buffer
newNetwork: WIFI, Cellular
userAction: seek
newCdn: Values: cdn1, cdn2
seqnum: Values: integers

Given this data, your task is to create custom metrics to answer the following question(s) using this tool. Please try to finish as many tasks as possible and it's ok if you don't finish them. For each task below, please first describe how you would create a metric for the task using your own words. Then, you can use the tool to create the metric.

- Task 1: Total time spent in buffering while on Wifi Network (note: the data is from a single session).
- Task 2: Total time spent in buffering while using CDN1 and on Cellular Network (note: the data is from a single session).

Now, let's try a different dataset. This is a simplified dataset of browser events for a single browser session: The structure of the dataset is as follows.

Fields: values
session: Values: string
eventType: Values: PageAction, PageView, BrowserInteraction, PageViewTiming
timestamp: Values: timestamp of event
pageUrl: Values: string

- Task 3: Total time spent on a particular page (demo.com/index.html) for a session (note: the data is from a single session).
- Task 4: Average time spent on a particular page (demo.com/index.html) for a session (note: the data is from a single session)

[Task instructions] As you perform these tasks, I'll be observing how you do them but please also think aloud (i.e., describe your thoughts) so that I can better understand your thought process and experience. [If the participant gets stuck for 15 mins, move on]

- First, please use natural language to describe how you would create a metric for this task.
- Second, please use the tool to create the metric for this task.

Step 4: Post-task interview (10 mins)

[overall experience]

- What's your overall experience with the tool?
- Did you achieve what you hoped to achieve (i.e., completing the task, getting the expected results)? If not, why?
- How did you know whether you have completed the task (created the metric) correctly?
- What's the most challenging part of completing this task?
- Did you get stuck in any part of the task?

[Tool design]

- Anything you like about this tool?
- Anything you don't like about this tool? anything (e.g., labels) confusing or difficult?
- What do you think about the following features? Tooltips, data preview, and auto-suggest.
- Any suggestions to improve the tool?
- How would a tool like this affect your [your customers'] work or job?

C.3 Case Studies

Step 1 (background) 5 mins

Thanks for coming! My name is XYZ. I'll be leading today's study session, which will take about 60 minutes. The goal of this study is to get your insights on data analytics and metrics.

You will be asked to perform some tasks to create metrics with certain tools. Please feel free to ask us questions. If you experience something difficult, it's because of the tools not because of you. Could we record the meeting and your screen so we could analyze the detailed study data? Thank you!

[Participants are given the consent form. Proceed the study, if they provide their verbal consent.]

[please join this Zoom meeting, share your screen, and record the meeting w/ auto transcription enabled]

I'd like to start by knowing a bit more about you · What's your role/job?

· What's your age group? [18-24, 25-34, 35-44, 45-54, 55+]

· What's your gender identity? [male, female, other]

· Have you created any metrics that monitor some kind of system or human behavior? If so, could you give me an example?

· How many years of experience do you have with creating metrics?

· Have you used any metrics? If so, could you give me an example?

· How many years of experience do you have with using metrics?

· Do you know SQL? How many years of experience do you have with SQL?

Step 2 (video tutorials) 10 mins

I'd like you to create some metrics with a tool. Before that, I'd like you to watch the following short video tutorials about the tool.

We will start with the high-level concepts and then move on to how the tool works.

IntroToTimeLine.mp4 - Introduction to the Timeline Concept

IntroToTimeLineOper.mp4 - Introduction to TimeLine Operators

MetricBuilderWalkThrough.mp4 - Example data and metric

If you'd like more details on these topics, please refer to the detailed references (e.g., operator cheat sheet) [links].

Q: any questions about any of these videos? [researcher writes them down]

Step 3 (tasks) 30 mins

Now, we'd like you to conduct some tasks using this tool for about 30 minutes.

[researcher note] pay attention to how they explore the tool

[Each task takes 15 mins. Start with our pre-defined task and then if time allows they can try a task/query of their own.]

[Task instructions]

As you perform these tasks, I'll be observing how you do them but please also try to think aloud (i.e., verbally describe your thoughts) so that I can better understand your thought process and experience.

[If the participant gets stuck with a task for 15 mins, move on]

First, please use natural language to describe how you would create a metric for this task.

Second, please use the tool to create the metric for this task.

Third, please use the tool to validate the metric is correct.

[Task description]

Imagine that you are tasked to build custom metrics for the following dataset.

This is a simplified dataset of video streaming for a single video session:

The structure of the dataset is as follows. Each record will have a timestamp, a session ID, a seqnum and optional attributes depending on the measurements:

Data Fields:

time: Values: timestamp of event

sessionId: Values: 1

newPlayerState: Values: play, pause, buffer

newNetwork: WIFI, Cellular

userAction: seek

newCdn: Values: cdn1, cdn2

seqnum: Values: integers

Given this data, your task is to create custom metrics to answer the following question using this tool below: Tool

Task 1: Total time spent in buffering while using CDN1 and on Cellular Network (note: the data is from a single session)

Now, let's try a different dataset. This is a simplified dataset of web browsing for a single browser session:

The structure of the dataset is as follows.

Data Fields: values

session: Values: string

eventType: Values: PageAction, PageView, BrowserInteraction, PageViewTiming

timestamp: Values: timestamp of event

pageUrl: Values: string

Task 2: Average time spent on a particular page (demo.com/index.html) for a session (note: the data is from a single session)

Step 4: Post-task interview (10 mins)

[overall experience]

· What's your overall experience with the tool?

· Did you achieve what you hoped to achieve (i.e., completing the task, getting the expected results)? If not, why?

· How did you know whether you have completed the task (created the metric) correctly?

· What's the most challenging part of completing this task?

· Did you get stuck in any part of the task?

[Tool design]

· Anything you like about this tool?

· Anything you don't like about this tool?

· Any suggestions to improve this tool?

· What questions would you like to answer using this tool?

C.4 Exit Survey (for case study participants)

Inspired by [19, 51], we invited participants to fill out an exit survey after the case studies. We included screenshots and URLs for both our tool and MySQL workbench to assist the participants in completing the survey. The participants self-rated their experience using these tools on a few aspects in the form of 7-point Likert scale questions. Each question was asked twice, once for each tool, randomly in a counter-balance fashion. The participants explained their reasoning for these ratings in open-ended questions.

Likert scale questions:

- **Match goal**: I'm satisfied with my final results from the system; it met my task goals (e.g., creating and validating metrics).
- **Think through**: The system helped me think through what kinds of outputs I would want to complete the task goal and how to complete the task.
- **Transparent**: The system is transparent about how it arrives at its final results; I could roughly track its progress.
- **Controllable**: I felt I had control creating with the system. I can steer the system towards the task goal.
- **Collaborative**: In [each tool], I felt I was collaborating with the system to come up with the outputs.

Open-ended questions:

- **Timing**: How long do you estimate it takes you to create a typical metric using [each tool], and why?
- **Difference**: What were the differences, if any, between the experience of completing the task using [the two tools]? How about comparing to other tools or systems if you have been using any in the past?
- **Vision**: If you were creating metrics in your work, in what situations would you prefer to use [each tool]? Can you think of 1-3 concrete examples?

We note a number of limitations of the exit survey.

- We compared our tool to MySQL Workbench in the survey, because MySQL Workbench is a popular SQL query tool. However, because the two tools have quite different UIs, we could not precisely isolate the impact of individual features (e.g., node-graph editor) of our tool on participants' experience in metric creation and validation. A more precise measurement of the impact of the node-graph editor would have been possible if we compared our tool to a sandbox system [cite AI Chaining paper], whose UI looks like our tool but it does not have the node-graph editor. Due to time constraints, we were unable to create a Sandbox system for another round of case studies. In the future, longitudinal field deployments of our tool in the wild could further reveal the pros and cons of our tool.

- Our sample size is relatively small and our participants were all from the same organization. Therefore, our results may not generalize to data analytics practitioners in general.
- We provided sample datasets for our case studies. These datasets represent a diverse set of realistic use cases. However, they do not cover all possible real-world use cases. Future studies could cover additional use cases including those from the end users.

C.5 Quick Feedback Survey (for any users)

- Do you have any suggestions to improve [the tool]?
- What features would you like to add to [the tool]?
- What questions would you like to answer using [the tool]?
- What's your job/role?

D METRIC DAGS FOR CASE STUDY TASKS

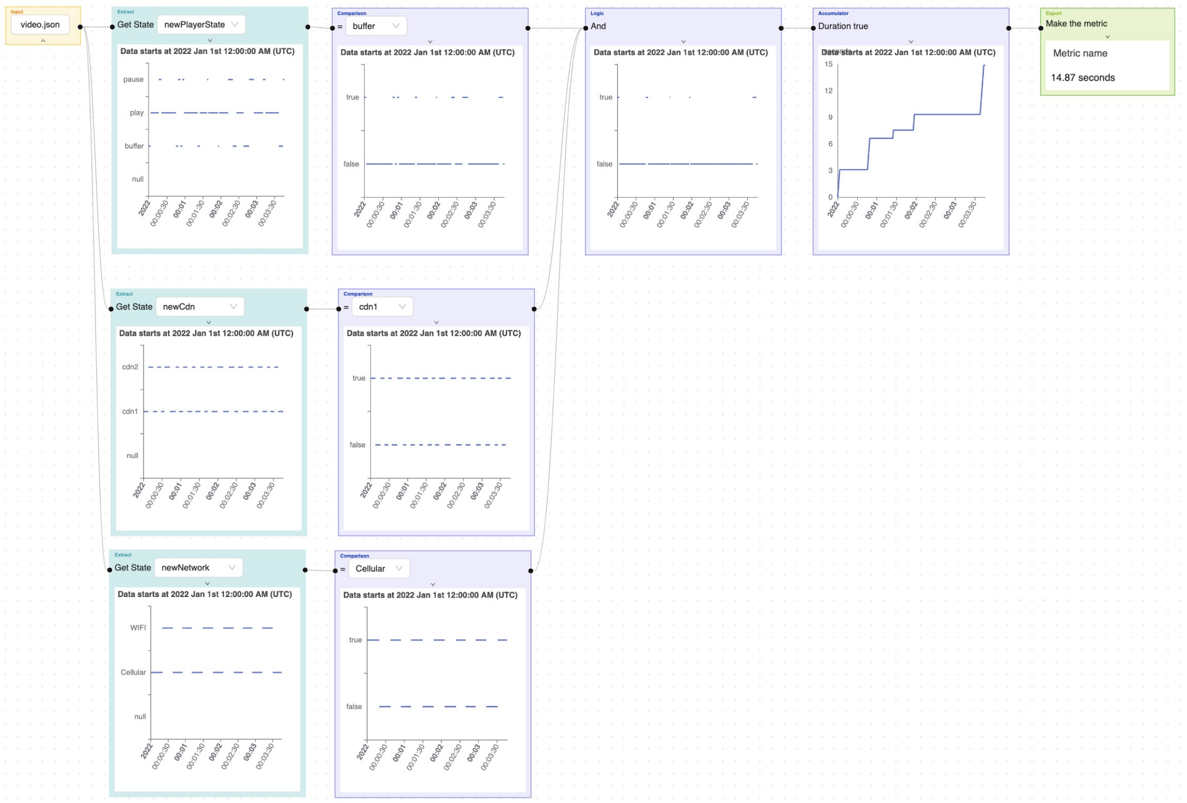


Figure 14: The metric DAG for case study 1: video streaming.

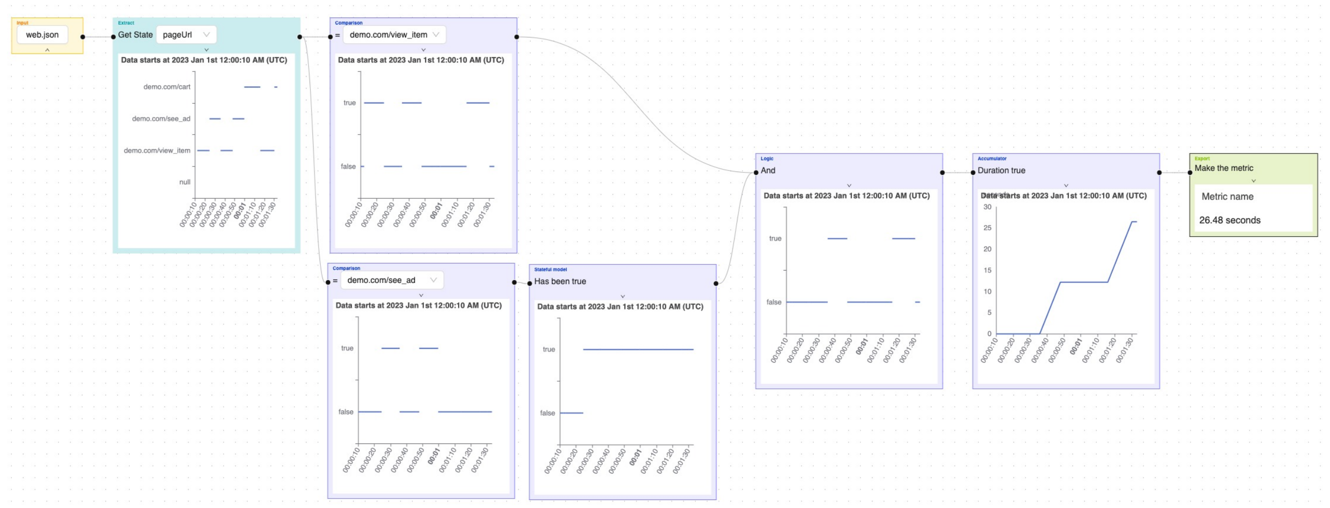


Figure 15: The metric DAG for case study 2: website browsing.

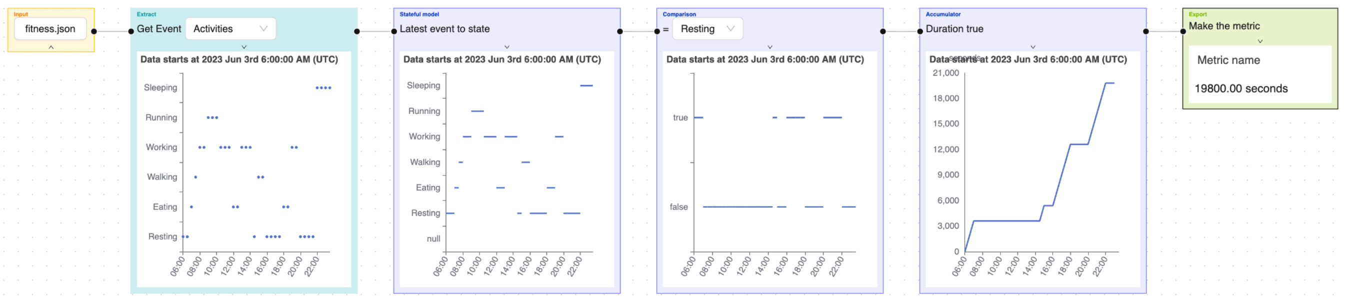


Figure 16: The metric DAG for case study 3: fitness tracking.