Octave: an End-User Programming Environment for Analysis of Spatiotemporal Data for Construction Students

Daniel Manesh*, Andy Luu*, Mohammad Khalid†, Jiangyue Li*, Chinedu Okonkwo‡, Abiola Akanmu†, Ibukun Awolusi‡, Homero Murzi§, and Sang Won Lee*

*Department of Computer Science, Virginia Tech, Blacksburg, Virginia, Email: danielmanesh@vt.edu

†Myers-Lawson School of Construction, Virginia Tech, Blacksburg, Virginia

‡School of Civil & Environmental Engineering, and Construction Management, UTSA, San Antonio, Texas

§Department of Engineering Education, Virginia Tech, Blacksburg, Virginia

***Abstract*—The construction industry is a new avenue for big data and data science with sensors and cyber-physical systems deployed in the field. Construction students need to develop computational thinking skills to help make sense of this data, but existing data science environments designed with textual programming languages create a significant barrier to entry. To bridge this gap, we introduce Octave, an end-user programming environment designed to help non-expert programmers analyze spatiotemporal data (e.g., as gathered by a GPS sensor) in an interactive graphical user interface. To aid exploration and understanding, Octave's design incorporates a high degree of liveness, highlighting the interconnection between data, computation, and visualization. We share the underlying design principles behind Octave and details about the system design and implementation. To evaluate Octave, we conducted a usability study with students studying construction. The results show that non-programmer construction students were able to learn Octave easily and were able to effectively use it to solve domain-specific problems from construction education. The participants appreciated Octave's liveness and felt they could easily connect it to real-life problems in their field. Our work informs the design of future accessible end-user programming environments for data analysis targeting non-experts.**

***Index Terms*—Construction, Data analytics, End-user Programming, Spreadsheets, Spatiotemporal Data**

I. Introduction

The advancement of machine learning and Big Data have raised expectations in various domains for the applications of data science and the value it can return. The construction industry is one such example, where various sensing technologies are increasingly deployed in the field, enabling data-driven analysis and decision-making [1]. In practice, domain experts do not always have a background in data analytics; construction specialists need to develop computational thinking and data analysis skills to make use of the sensor data.

Many existing programming languages used for data analysis have a high barrier to entry (e.g., python, R, Matlab). Rather than using an existing programming language, an end-user programming (EUP) environment can offer the benefits of a flexible programming tool but with a more straightforward interface that is tailored to a specific use-case [2]. However, even a simple EUP environment, such as a spreadsheet application, can be challenging with overwhelming amounts of raw data, especially if the data cannot be easily and effectively visualized. We propose an alternative environment that allows non-programmers to grok data in an exploratory fashion by maintaining a tight connection between visualizations, algorithms, and the underlying data. We posit that such an environment will help students and domain experts learn computational concepts.

We test this idea in our ongoing project Octave (**O**bservable **C**onnections between **T**ables, **A**lgorithms, and **V**isualizations in an **E**UP Environment), a programming environment for building data analysis and computational thinking skills. Octave specifically targets university students studying Construction Engineering Management (CEM) and related fields and is designed to help them analyze spatiotemporal sensor data, e.g., from GPS sensors deployed at a construction site. We validate our design with a user study involving 13 students studying construction. We found that they were able to quickly learn Octave, and easily understood how it could be used to solve the domain-specific problems used in a construction class.

While the scope of Octave is limited to trajectory data, we believe our study shows the feasibility of the underlying programming model, where users can specify an algorithm through direct manipulation with the target object visually or through a separate interface with a tight connection with the object (in this case, a vehicle trajectory). We believe this approach can generalize to the design of other domain-specific end-user programming environments.

II. Background and Related Works

A. Facilitating Computational Thinking

Many researchers have argued that the value of computational thinking extends beyond the computer science discipline [3]–[6]. For example, computational thinking skills can help someone to reformulate problems in computation-friendly ways or to ask insightful questions they might not have thought to ask otherwise [7]. Components of computational thinking encompass not only algorithmic design and abstraction, but also data collection, representation, and analysis [8].

It has also been shown that when teaching computational thinking, grounding lessons in the context of domain-relevant tasks has a positive effect on learning outcomes [9]. This suggests there is value in a programming environment like Octave, which can teach computational thinking while tackling a domain-specific problem but without the overhead of learning a general programming language.

B. Benefits of Liveness

Researchers and practitioners used the idea of real-time program feedback as having "liveness" in programming environments, where changes to a program are instantly rendered to the output [10]–[12]. We utilize this notion of liveness as one of Octave's core design principles, which is to maintain a tight connection between the data, visualization, and computation (section IV-B). Researchers have previously validated the effects of having an immediate and clear connection between code and a program state in helping programmers have a clear mental model [11], [13], [14]. In addition, high levels of liveness are shown to help programmers with debugging and explaining their programs to others [15], [16].

Data science and analysis can be thought of as *exploratory programming* [17], characterized by rapid experimentation, often going back and forth between code and visualization. For this reason, researchers have argued that data science is well-suited for liveness [18]. In fact, a lack of liveness is seen as a problem with traditional computational notebooks, where graphs and code can easily get out of sync [19]. Researchers have created data science environments around the design idea that the visualizations should always reflect the current version of the code [20]–[22]. Although Octave is an EUP environment designed for simple data analysis tasks, we still believe liveness is an essential feature for helping our users explore and understand their data.

C. Spatiotemporal Data

Spatiotemporal data can fit into three broad categories: spatial time-series data, spatial event data, and trajectory data [23]. Among these, trajectory data is considered particularly complex to analyze [24]. Trajectory data can either be *quasi-continuous* or it can contain gaps—in the extreme case, the data may contain only the origin and destination points [23]. Octave targets quasi-continuous trajectory data, as would be collected by a GPS sensor.

Our work draws ideas from the literature on Geographic Information System (GIS) technologies for visualizing, manipulating, and querying spatiotemporal data. For example, the ArcGIS environment allows analyzing spatiotemporal data through a variety of methods, including a python notebook interface [25] and with database query languages like PostgreSQL [26]. GIS tools are targeted toward experts, which makes them less suitable for our purposes as an introduction of computational thinking skills to construction students.

Researchers have built tools to analyze spatiotemporal data without the use of a general-purpose programming language. Tools that handle trajectory data often focus on visualizations, for example, displaying different trajectories views along with aggregate visualizations like heatmaps [27]–[29]. Some tools allow both visualization and complex queries over spatiotemporal data. One such system is TaxiVis, an environment for querying origin-destination taxi trip data [30]. Another example is VAUD, an end-user environment for visualizing and querying complex, heterogeneous spatiotemporal data, including—but not limited to—trajectory data [31]. Octave differs from these systems in that one of our primary goals is to help students develop computational thinking skills with some exposure to programming concepts such as a boolean expressions or state diagrams. The computational steps to query the data must be both simple enough for students to learn easily, yet powerful enough to be meaningful.

III. Design Principles

Octave aims to facilitate computational thinking among construction students for a domain-specific problem: the analysis of spatiotemporal trajectory data. To inform our design, we did a preliminary task analysis using an assignment from an existing construction class. We describe the task below and show how it led us to the core design principles of Octave.

A. Task Analysis: Productivity Scenario

We began our design process by considering a domain-specific problem one of the authors teaches to CEM students in our University. The scenario, which we will refer to as the *Productivity Scenario*, is as follows. Suppose we have GPS sensor data from a dump truck that is tasked with repeatedly picking up a volume of material from one location and transporting it to another. Each round trip (or *cycle*) the vehicle makes can be divided into four sections: *load*, *haul*, *dump*, and *return* (Figure 1) [32]. *Load* and *dump* refer to when the vehicle is being loaded with material or is depositing material, and can be approximated using the GPS coordinates, i.e., whether the vehicle is inside the *loading zone* or *dumping zone*. *Haul* and *return* are defined as when the vehicle is in transit, either heading towards the dumping zone (*haul*) or towards the loading zone (*return*). Sections of the trajectory that do not fit into these categories are classified as *non-productive movement*.

In this scenario, a data analyst in a construction company wishes to measure the productivity of the vehicle. For example, how many round-trips did the vehicle make, and thus how much material was moved? How long did each leg of the journey take, and were there any bottlenecks? The answers to these questions can help analysts understand how resources are used in practice and can support future decision-making (e.g., whether more trucks are needed).

We started our design process by solving the Productivity Scenario with example data previously used in the construction classroom. We devised two solutions using different common programming environments: a spreadsheet programming language and SQL. In both cases, our first step was to classify whether each data point was inside of the *loading zone* or *dumping zone*. To do this, we graphed the data using the

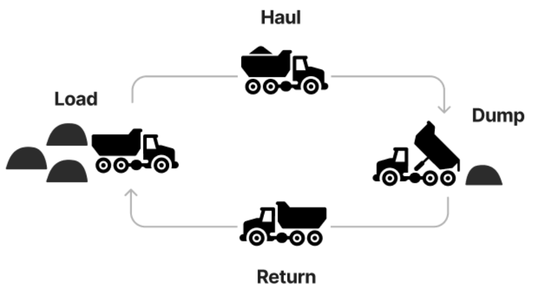


Fig. 1. The "Productivity Scenario" describes the common construction task of materials transport. To complete one cycle, the dump truck: loads material (load), transports the material to the dumping zone (haul), dumps the material (dump), and then returns to the loading zone (return).

spreadsheet program, then evaluated if each GPS coordinate was within the loading or dumping zone by calculating the euclidean distance between the center of the loading or dumping zones and the GPS coordinate.

After creating the two columns for *load* and *dump*, we created two additional boolean-valued columns to capture the *haul* and *return* states. To do this, we needed to consider both the current and previous state values of *load* and *dump*. Take *haul*, for example, where the truck is moving from the loading zone to the dumping zone. Regardless of the GPS coordinate, the *haul* state is TRUE when the *load* and *dump* states are FALSE, the truck was previously in the *load* state, and the truck's next state will be the *dump* state. The *return* state is defined similarly, but going from state *dump* to *load*.

Finally, we wanted to get summary data for each leg of the journey. Taking the *load* step as an example, we wanted to know how many times the truck visited the loading zone and how long the truck stayed during each visit. For this task, the common aggregation methods, such as pivot tables in spreadsheet programming or GROUP-BY statements in SQL, were not sufficient on their own. If we had used "GROUP BY *load*", for example, all the data inside the loading zone would form a single group, rather than being broken out by each visit. Instead, we added an integer-valued counter column for *load* which we incremented each time the state transitioned from FALSE to TRUE (e.g., the truck entered the loading zone). From there, we grouped the data by both the *load* state and the counter column to get our summary information.

B. Design Principles

Based on our task analysis, we derived two core design principles for Octave, which we describe below.

(DP1) Temporally-aware Querying by Spatiotemporal State. *Spatiotemporal state* refers to augmenting trajectory data with states, which are TRUE/FALSE values associated with each data point. In our task analysis, this was a natural way to decompose the problem into smaller steps. *Temporally-aware querying* refers to grouping operations which automatically separate data which are not temporally adjacent. During our task analysis, we felt it was cumbersome to create separate columns for counters and then do a grouping operation for the counter values, and wished this feature was built into the tools we were using.

(DP2) Highly-coupled Spatiotemporal Data Representations. Raw data and any user-defined states should be tightly coupled with its visualization. For example, if a user hovers over spatiotemporal data in a table (raw data), it should highlight the corresponding data points on a map (visualization). From our task analysis, we needed visualization to locate the loading and dumping zones, and wished that the visualization were connected to the data such that we did not have to define the geometric equation for being inside of a region by hand. Furthermore, visualizing the data was useful for checking our work, and we found it inconvenient that visualizations were not more readily available.

IV. System Design and Implementation

The interface for Octave is presented in Figure 2. On the left half of the screen is the *visualization pane*, which shows a map of the trajectory data, and on the right half is the *table pane*, which shows the data in a tabular format. The computational model of Octave is simple: users augment the data with boolean-valued columns called *states*, then run analyses by creating *summary tables* which aggregate the data based on those states. The table pane has a tab for the original data as well as tabs for each summary table the user has created. Finally, above the visualization and table panes is the *state toolbar*, which displays existing states and allows the user to create new ones.

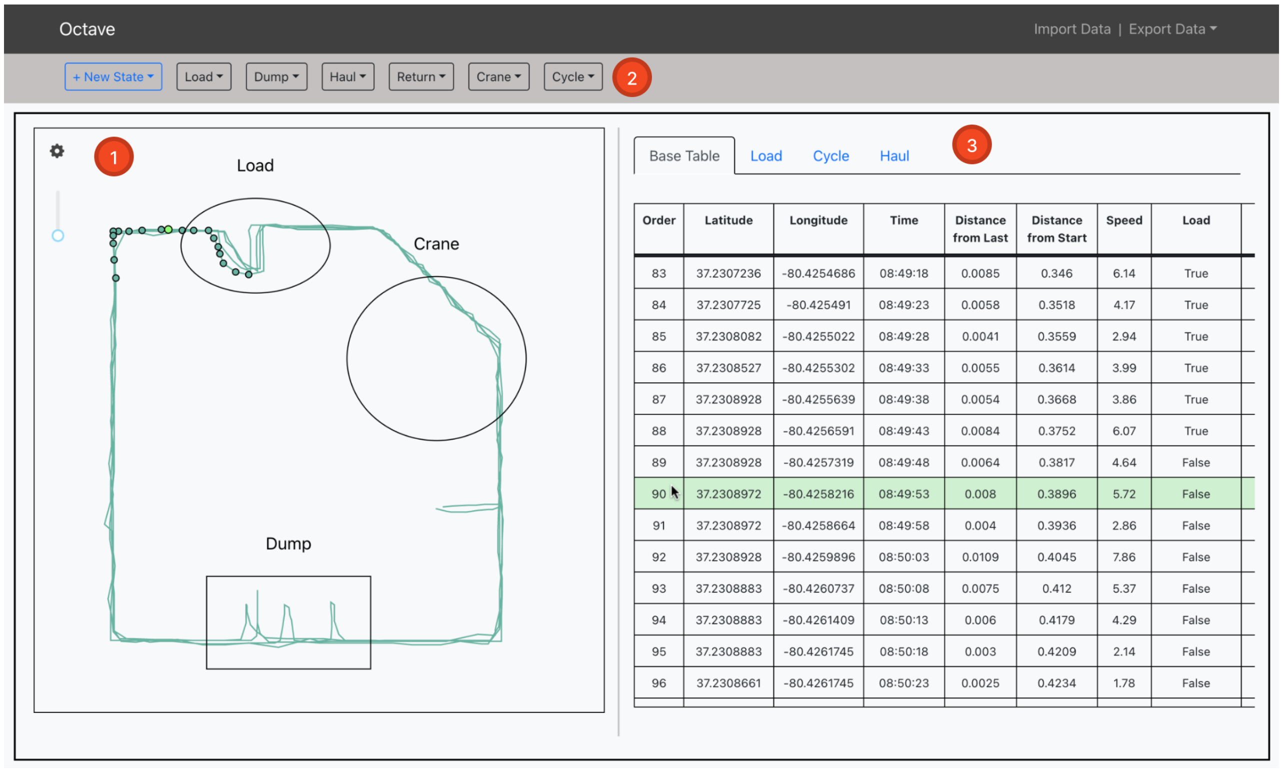
We now describe these components in detail and relate them to the design principles from section III.

A. DP1: Temporally-aware Querying by Spatiotemporal State

1) ***Binary state as a programming primitive****:* State in Octave is defined as a state of one spatiotemporal data point—a row in the table—that can take on a boolean value (TRUE or FALSE) based on its spatiotemporal properties or based on other states previously defined by the user. Ultimately, states are used to create summary tables, which provide aggregate statistics based on the state. For example, if a state represents a region on the map, the summary table will display statistics like how many times the trajectory entered the region and how long it spent there on average. Summary tables are described in detail in section IV-A2.

To create a new state, the user selects the "New State" button (Figure 2.2), selects the state type, and then follows the directions. Below, we describe the several types of states users can create in Octave, each of which has its own user interface for creation.

Region states represent a physical region. Regions are defined by rectangular or oval shapes on the visualization pane, which can be placed, resized, and rotated by the user. The state is TRUE only for the data points inside the region. In the Productivity Scenario example, the *load* and *dump* states can be defined as region states encompassing the *loading zone* and *dumping zone*, respectively.



Order	Latitude	Longitude	Time	Distance from Last	Distance from Start	Speed	Load
83	37.2307236	-80.4254686	08:49:18	0.0085	0.346	6.14	True
84	37.2307725	-80.425491	08:49:23	0.0058	0.3518	4.17	True
85	37.2308082	-80.4255022	08:49:28	0.0041	0.3559	2.94	True
86	37.2308527	-80.4255302	08:49:33	0.0055	0.3614	3.99	True
87	37.2308928	-80.4255639	08:49:38	0.0054	0.3668	3.86	True
88	37.2308928	-80.4256591	08:49:43	0.0084	0.3752	6.07	True
89	37.2308928	-80.4257319	08:49:48	0.0064	0.3817	4.64	False
90	37.2308972	-80.4258216	08:49:53	0.008	0.3896	5.72	False
91	37.2308972	-80.4258664	08:49:58	0.004	0.3936	2.86	False
92	37.2308928	-80.4259896	08:50:03	0.0109	0.4045	7.86	False
93	37.2308883	-80.4260737	08:50:08	0.0075	0.412	5.37	False
94	37.2308883	-80.4261409	08:50:13	0.006	0.4179	4.29	False
95	37.2308883	-80.4261745	08:50:18	0.003	0.4209	2.14	False
96	37.2308661	-80.4261745	08:50:23	0.0025	0.4234	1.78	False

Fig. 2. The interface for Octave. 1) The Visualization Pane: a two-dimensional visualization of the spatiotemporal data. Note that the data points visible correspond exactly to the visible rows in the table, and the bright green data point corresponds to the row under the cursor. 2) The States Toolbar: this shows the list of states which have been defined by the user, and also has a button for the user to create a new state. States appear as new boolean-valued columns in the original data table. 3) The Data Pane: this shows all of the data in the table in a spreadsheet-like format. The data pane has a tab for the uploaded data (i.e., *Base Table*) and additional tabs for summary tables, which group and aggregate data, similar to a pivot table from traditional spreadsheet programming environments.

Timespan states represent a continuous stretch of time. Timespans are defined by adjusting the endpoints of a slider, and the state is TRUE only for data points within the timespan. In the Productivity Scenario, a timespan state might be used to exclude noisy data points at the start or end of data collection.

Conditional states allow the user to define boolean expressions using logical and comparison operators. The expressions are evaluated for each row, and can involve values for any of the columns, including both user-defined states or existing measurements, like speed or elevation. Notably, these expressions can *only* reference values from the current row. In our Productivity Scenario, a user might create a conditional state to investigate if the vehicle has any unexpected slowdowns outside of the dumping and loading zones. To capture this logic, the conditional state might use the expression "*Speed* < 5 *AND NOT load AND NOT dump*".

Combination states provide a way to combine two existing states into a new one, not only with binary operators (such as AND/OR/XOR) but also with the specification of state transition requirements. After selecting the two states to combine, users are presented with an interface displaying a graph with nodes and edges (Figure 3), where nodes represent all possible combinations of the state values (e.g., TRUE-TRUE, TRUE-FALSE), and edges connect nodes when transitions between those pairs of state values actually occur in the data. The user can then click on some combination of nodes and edges to indicate when their new state should be TRUE. If they *select a node*, all rows with that combination of states will be TRUE. *Selecting or deselecting edges* acts as a transition constraint on the previous or next state pair. This way, users can create a new state based on binary operations (e.g., AND/OR) *and* state transitions (e.g., all data points in state B where the preceding state is neither A nor B, and the following state is A).

In our Productivity Scenario example, the *haul* and *return* states can both be defined using combination states. To define *haul*—where the loaded truck travels from the loading zone to the dumping zone—a user creates a new combination state that combines the region states *load* and *dump* (Figure 3). On the combination state interface, there will be three nodes available: *Load*, *Dump*, and *Neither*. In this case, a node for *Both* will not exist because there is no intersection between the loading zone and dumping zone. The interface will also display arrows based on how the truck travels: there will be incoming and outgoing arrows connecting *Neither* to both *Load* and *Dump*, but there will be no arrow directly between *Load* and *Dump* because there is no way that the truck can travel from the loading zone to the dumping zone without passing the neutral area (*Neither*). From here, the user can define the *haul* state by selecting the node *Neither*, along with the incoming arrow from *Load* and the outgoing arrow to *Dump*.

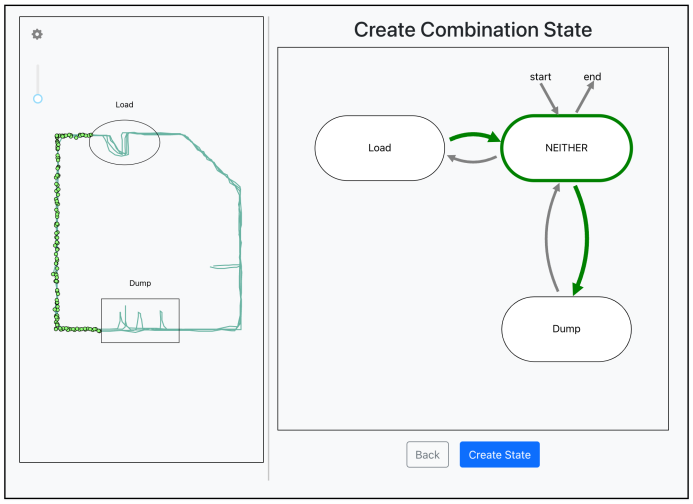
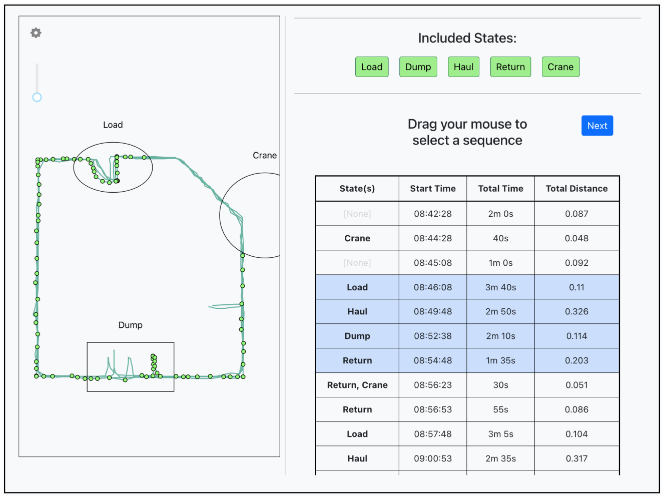


Fig. 3. The interface for creating a combination state. The current selection captures all groups of points which have *load* and *dump* states FALSE (*NEITHER* node selected), where the previous group of points has *load* TRUE (incoming arrow from *Load*) and the next group of points has *dump* TRUE (outgoing arrow to *Dump*). The selected data points are displayed on the visualization pane to the left and updated in real time as the user selects and deselects nodes and edges.



Drag your mouse to select a sequence

State(s)	Start Time	Total Time	Total Distance
[None]	08:42:28	2m 0s	0.087
Crane	08:44:28	40s	0.048
[None]	08:45:08	1m 0s	0.092
Load	08:46:08	3m 40s	0.11
Haul	08:49:48	2m 50s	0.326
Dump	08:52:38	2m 10s	0.114
Return	08:54:48	1m 35s	0.203
Return, Crane	08:56:23	30s	0.051
Return	08:56:53	55s	0.086
Load	08:57:48	3m 5s	0.104
Haul	09:00:53	2m 35s	0.317

Fig. 4. The interface for creating a sequence state. The bottom table groups the data by the states selected at the top. The user can select an example sequence by dragging their mouse over the bottom table, and the data points from that sequence are shown on the visualization pane. Note that to correctly define a cycle for the Productivity Scenario, in the above example the user needs to exclude the region state *Crane*.

Sequence states combine a specifically ordered sequence of existing states into a new state. To create a sequence state, the user first selects which states to consider. Then, they are presented with a table which groups temporally adjacent data points based on the boolean values of those states (Figure 4). For example, if we consider two states, A and B, the first group of points might have both states TRUE (the table will display "A, B"), the next group of points might have just B TRUE (the table will display "B"), and the next group might have neither TRUE (table will display "–"). The user drags the mouse over the table to select an example of the sequence they want to define. The remaining instances of that sequence are picked out automatically, and the new state will be TRUE for every data point that is a part of one of these sequences. The user can preview the new state before they save it.

In the Productivity Scenario example, a sequence state can represent a *cycle*. To define the cycle, the user creates a sequence state involving states *load*, *haul*, *dump*, and *return*, then finds and highlights that sequence in the table. Note that this differs from a conditional state which represents the union of *load*, *haul*, *dump*, and *return*. Such a conditional state would be insufficient for two reasons. First, it might capture *non-productive movement*, say if the truck returned to the loading zone before reaching the dumping zone. Second, the conditional state would incorrectly group together adjacent cycles. Sequence States do not have this problem because of their special grouping semantics, as described in the following section (IV-A2).

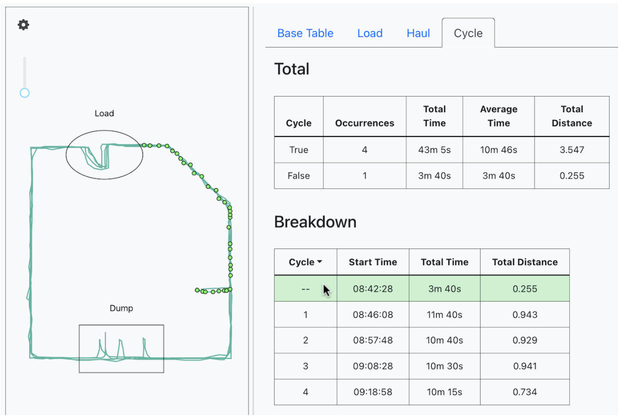
*2) **Summarize by State**:* Using the states they have defined, users can analyze the data through table summary operations. To create a summary, the user selects an existing state, and then selects the "create summary" option. Summaries automatically group the data by the chosen state, similar to a GROUP BY clause in SQL. Unlike in SQL, however, points are only grouped together if they are also temporally adjacent. Each group of data points is associated with aggregate statistics, such as the elapsed time and total distance traveled.

The summary for a state contains two tables which appear in a single tab in the data pane: the *breakdown table* and the *total summary table* (Figure 5). The *breakdown table* contains one row for each time a state is *visited*, i.e., each block of time for which the state was TRUE. Each row contains the visit number, the start time of the visit, the total time of the visit, and the total distance traveled. The user can opt to expand this table to also include groups where the state is FALSE.

The *total summary table* contains a single row with aggregate data, such as the total number of visits and the average time per visit. If the breakdown table is expanded to show the FALSE groups, the total summary table will contain a second row summarizing the stretches of time outside the state.

Summaries for sequence states work slightly differently than for other states. First, the breakdown table for sequence state summaries can be expanded to show each step of the sequence in a separate row. This is in addition to being able to show or hide the groups for which the sequence state is FALSE. Second, the grouping for sequence states works slightly differently in that data points are separated into different groups each time the sequence restarts. For example, consider a sequence state which uses the sequence (A, B), for some states A and B. If the actual data contains the sequence (A, B, A, B), then instead of grouping all those points together, the two (A, B) pairs are grouped separately.

Let us again go back to the Productivity Scenario. By creating a summary table for the *load* state, users can understand the average loading time. This can be compared to summary



Total

Cycle	Occurrences	Total Time	Average Time	Total Distance
True	4	43m 5s	10m 46s	3.547
False	1	3m 40s	3m 40s	0.255

Breakdown

Cycle	Start Time	Total Time	Total Distance
--	08:42:28	3m 40s	0.255
1	08:46:08	11m 40s	0.943
2	08:57:48	10m 40s	0.929
3	09:08:28	10m 30s	0.941
4	09:18:58	10m 15s	0.734

Fig. 5. Summary tables for Sequence State *cycle*. Statistics about each individual cycle are displayed in the "Breakdown" table, and the "Total" table contains statistics about all cycles together. The "Breakdown" table also shows the initial stretch of time, which has no visit number because the state *cycle* is FALSE. These data points are shown on the map because the cursor is currently over this row.

tables for the *dump*, *haul*, and *return* steps to determine which steps take the longest and where efficiency can be improved. Figure 5 shows the summary for *cycle*, a Sequence State. From this summary, the user can determine the total number of cycles and the total time, which can be used to calculate the per-hour productivity of the vehicle.

B. DP2: Highly-coupled spatiotemporal data representations

Octave provides two tightly coupled views of the data: a non-editable table interface (Figure 2.3) and a 2D visualization (i.e., map; Figure 2.2). The visualization always shows the full trajectory as a connected path (i.e., lines), while data points along that path (i.e., dots) appear in response to user interactions with the data pane and state toolbar. The visualization pane communicates to the user which data points are relevant.

When the "Base Table" is shown in the data pane, the rows currently visible in the table are all drawn as dots on the visualization pane. Thus, users can scroll through the data table to scrub forwards and backwards through time, getting a sense of what direction the trajectory is moving based on the dots shown on the path. Furthermore, if the user hovers over a particular row in the Base Table, the corresponding data point in the visualization pane is highlighted with a brighter green color, as seen in Figure 2.

When a summary table is shown in the data pane, no data points are present by default on the trajectory path in the visualization pane. When the user hovers over a particular row, all of the data points that are part of that grouping will appear as dots along the trajectory, as seen in Figure 5. Thus, users can quickly visualize which group of points are summarized by each row.

Having a clear mental model of how their code (or algorithm) will change the data is crucial for a programmer's sensemaking process [33]. To that end, whenever a user creates a state, the visualization is updated to help the user understand what will happen. With combination states, for example, as the user specifies the state by clicking nodes and edges on the interface (see Figure 3), the data points which will be TRUE for the state are automatically updated and displayed as highlighted circles on the visualization. That way, the user can explore the interface, getting immediate feedback for what-if questions (e.g., what if I deselect this arrow?). In our Productivity Scenario, it would have been challenging for users to make sense of new spatiotemporal states, such as *return* and *haul* in their spatial context just by looking at raw data or state diagrams. This tight connection also lets the user verify that their new state captures the logic they want. Sequence states work similarly; when selecting a sequence, the data points in the example sequence appear on the visualization (Figure 4). Once the example sequence is chosen, the user is presented with the breakdown summary table based on the sequence state, which they can use to verify correctness before committing their new state.

Finally, users can hover their mouse over an existing state in the state toolbar (Figure 2.2) to show dots for all of the data points for which that state is TRUE. This way, they can easily remind themselves of what each state represents.

C. Other Features

In addition to the features already discussed, Octave supports importing data, exporting data, and importing site layouts (i.e., background images). To import data, users can either choose an existing data set to load or select a CSV file from their computer. If they choose their own CSV file, they are shown a preview of the table, and they must specify which columns refer to the latitude, longitude, and time.

Octave allows users to export their data so they can send it to others or use it elsewhere. Users can click the "export data" button on the top-right corner of the screen to download a CSV of the data in the currently active tab. If the active tab is the Base Table, the exported data is a CSV with the original data, along with a boolean-valued column for each state the user has defined. If the active tab is a summary table, the data is exported as two separate CSVs: one for the breakdown table and one for the total summary table.

Users also have an option to import or export an entire project. Projects are exported as JSON, which can either be imported back into Octave or processed with another program.

Finally, we allow users to upload an image of a site layout (i.e., map) to display under the trajectory data. This feature was thought crucial for construction students as the GPS trajectory is contextualized by the site layout, which closely represents the user's mental model. We also give the user the ability to move, rotate, and resize their data to fit the uploaded image, as their mental model and uploaded image may not be consistent with the true geographical orientation (e.g., the South entrance of the construction site may not exactly be south based on the GPS data). A slider on the left side of the visualization allows the user to adjust the opacity of the site map, so they can either make it more prominent or fade it out. Note that uploading a site layout is a completely optional step, and Octave can be used without a map in the background.

D. Implementation

Octave is implemented as a web application using the React framework. For the trajectory visualization, we use the D3 JavaScript library. Octave can easily handle data sets with up to 10,000 rows without impacting user experience, and uses the react-window library to selectively render rows in the Base Table. Octave does not store data in a back-end, and we instead rely on the browser's localStorage feature to save users' sessions, which caches the most recent data when they revisit the website if they use the same computer.

V. User Study and Methodology

We conducted a usability study to evaluate if Octave was easy to learn and use, and to gauge students' perceptions and attitudes towards our system. Participants were introduced to Octave and then given two tasks to complete under observation by the researchers. Afterward, we administered a standard SUS questionnaire [34], followed by a short exit interview to gather qualitative feedback. We chose not to compare Octave with an existing tool like SQL because our target population—CEM students—would typically not know SQL, and we would not have time to teach them during the study. Note that at the time of the study, Octave did not have conditional or sequence states implemented.

A. Participants

We recruited 13 participants (8 men, 5 women; ages 19-31, median 21) through our University's mailing lists. All participants were students studying construction or related fields (8 Construction Engineering Management, 3 Building Construction, 1 Civil Engineering, and 1 Environmental Design and Planning). Two participants were graduate students, and the remaining 11 were undergraduates. The self-reported programming experience of the students varied: while two indicated intermediate or advanced skills, seven rated themselves as beginners, and the remaining four said they had never programmed before.

B. Procedure

The study took place remotely via Zoom and lasted one hour. Participants were compensated for their time with a $20 electronic gift card. The study was divided into three parts: a tutorial, tasks, and an exit interview. During the tutorial, which lasted about 20 minutes, we introduced key features of Octave, including uploading data, uploading a site map, aligning data, creating region states, creating combination states, and creating summaries. We explained these features of Octave to the participant while sharing our screen, and then afterward had the participant share their screen and walk through the same steps we had taken.

For the second part of the study, we gave participants 20 minutes to solve two tasks using Octave. The tasks were both based on realistic scenarios in the construction domain. The first task was a safety analysis, where participants had to determine how many times and for how long a worker came dangerously close to a crane. We provided a site layout image for this task, which showed the radius of the crane. This task required creating a single region state as well as a summary table. The second task was a more complicated round-trip analysis, where the participant had to determine how many round trips were made between two points, and which part of the journey (i.e., A to B vs. B to A) was faster. This task required creating two region states, two combination states, and three summary tables. For this task, there was no site map in the background.

For each task, rather than going through the standard data-import flow, we had participants load the data with one click via a special "User Study" option in the menu bar. During the tasks, we typically did not interact with the participant, though we allowed them to ask clarifying questions. After each task, we had the participant show us their solution and explain the steps they took.

The final 15-20 minutes of the study was dedicated to feedback. We first gave participants a short survey which contained an SUS questionnaire followed by a few demographic questions. Then, we had an exit interview where we asked participants about their experience. We asked if they found Octave easy or difficult to learn, as well as what they liked or disliked about Octave. We asked what other features they thought Octave should have, and what other types of analyses they thought would be good to do with Octave or a similar system. Finally, we asked specific questions about the interface, like how they felt about having the data displayed in a table alongside the visualization.

VI. Results

All 13 participants were able to successfully complete both tasks using Octave. The average time to complete Task 1 was two minutes and twelve seconds, and for Task 2 was five minutes and eight seconds. These times do not include the time it took participants to explain their solutions. Eight participants solved the tasks without asking the researchers anything. Of the remaining participants, three asked clarifying questions and two needed a small hint to complete the task. Two participants created extraneous states which were not necessary to solve the task, but were still able to complete the task successfully.

A. SUS Questionnaire Results

The average SUS score for Octave was 85.96, which can be described with the adjective *Excellent* [35]. The SUS score can also be interpreted as a "letter grade", with the lower cutoff for *A-* at 78.9 [36]. Running a one-tailed t-test against this value, we see our mean SUS is significantly greater ($p = 0.016$). Thus, we can conclude that our SUS "grade" is at least an A-, with statistical significance.

Among the items on the SUS questionnaire, the statement that scored the worst was: "I think that I would need assistance to be able to use this website". This is not too surprising, given that there is no tutorial built into Octave, and we gave participants a tutorial ourselves before they used the website.

B. *Qualitative Feedback*

Easy to Learn; Simple to Use. All participants felt Octave was easy to learn, and they often described Octave as "simple" and "straightforward". P1 said they "felt pretty confident using it right away", and P2 thought that "most people should be able to get it in a couple of minutes". Participants thought it was easy to find the functionality they needed and that there were not too many extra or unnecessary features. Many participants appreciated the tight connection of the visualization with data, especially when scrolling through the data table. Two others mentioned they valued being able to understand and learn through exploration or "playing around".

Technical Terminology. Although it did not give them trouble, P11 imagined the terminology in Octave might be a barrier to usage for others. For example, they thought the term "states" felt technical, and that the TRUE/FALSE values for region states could alternatively be called "within" and "outside". On the other hand, P8 specifically mentioned the terminology in Octave was understandable, especially compared to other software they have used in the past.

Combination States. Although all participants were able to effectively create combination states during Task 2, some participants reported confusion in the exit interview. P11 felt that combination states felt very technical, and was not sure why they needed a node to represent both states being FALSE, denoted by *Neither* in the state diagram. P5 felt that combination states might not work well for a complex real-life scenario with a large number of region states.

New Features and Other Use Cases. Participants came up with different ideas to make Octave more usable. Some participants thought it would be useful to have a sense of scale shown on the map, which P6 suggested could be done by drawing a latitude and longitude grid. A few participants mentioned adding different colors to the regions or tables. For example, P12 mentioned it would be easier to detect colors than read the TRUE/FALSE state values in the table. P1 thought it could be nice to color-code regions, and P10 imagined color-coding either regions or sets of data points based on specific conditions, e.g., using red color to highlight if the data has been inside a region for too long. Finally, some participants thought the site layout map should either be aligned automatically with the data, or the map should be procured automatically, for example using Google Maps.

When we asked what types of problems Octave might help solve, participants gave a variety of responses. Many participants returned to the safety and productivity use cases from the two tasks they had solved earlier. Regarding safety, P3 suggested they might like to determine if a sensor attached to a moving vehicle came too close to a sensor attached to a moving worker. This would require expanding Octave to handle multiple trajectories. Regarding productivity, some participants suggested running simulations, which could then be analyzed using Octave. P2 expressed interest in using Octave for self-tracking, e.g., how long they spend in various places as they travel. Finally, P5 and P11 thought Octave would be especially useful for communication, with P11 lamenting that people will often "just throw you a spreadsheet" which they found difficult to understand and interpret.

C. *Limitations*

Our study only tests the region states and combination states: while we believe the other state types (sequence and conditional states) are simple enough, they have not yet been validated. Nevertheless, we believe our study evaluates the two main design principles of Octave, and shows the promise of this type of end-user programming and data analysis.

Our study focuses on the feasibility of our approach rather than assessing Octave's ability to impart computational thinking skills. While such an evaluation is important, we save it for a future classroom deployment.

VII. Discussion and Future Work

Our study highlights some of the tensions between using a general-purpose programming environment versus a domain-specific one. On one hand, experience with Octave might not directly translate to other programming environments. For example, although Octave might help someone understand how to decompose a problem into smaller parts, the individual parts might still be difficult to implement in another environment (e.g., calculating geometry for regions). On the other hand, participants were able to pick up concepts from Octave quickly and appreciated how approachable it was. Thus, Octave can introduce computational concepts like boolean-valued states and grouping and aggregation functions without being overwhelming. We plan to design and develop lectures, class activities, and assignments to validate Octave in deployment studies in the authors' universities.

One future direction for Octave could be to help users more directly translate their learning to other technologies, like spreadsheets. For example, Octave could show users the code (e.g., spreadsheet formulas) needed to define boolean-valued states. Octave could also make it more explicit how the summary tables are related to pivot tables or GROUP BY statements—this might be particularly helpful, as grouping and aggregate functions are difficult concepts to grasp in query languages like SQL [37] [38].

We believe Octave's design principles could also be used to improve existing EUP environments such as spreadsheets. Regarding Design Principle 1, to better support spatiotemporal data—and ordered data in general—grouping operations in spreadsheets (e.g., pivot tables) could support separating groups which are not temporally adjacent. Regarding Design Principle 2, a spreadsheet programming interface could let users create new data using a visual interface and existing input could be tightly connected with visual objects (i.e., charts).

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 2111003. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] A. Renz, M. Z. Solas, P. R. Almeida, M. Bühler, P. Gerbert, S. Castagnino, and C. Rothballer, "Shaping the Future of Construction: A Breakthrough in Mindset and Technology," in *World Economic Forum*, 2016.

[2] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw, and S. Wiedenbeck, "The State of the Art in End-User Software Engineering," *ACM Comput. Surv.*, vol. 43, Apr. 2011.

[3] C. Letondal, "Participatory Programming: Developing Programmable Bioinformatics Tools for End-Users," in *End user development*, pp. 207–242, Springer, 2006.

[4] T. Li and T. Wang, "A Unified Approach to Teach Computational Thinking for First Year Non–CS Majors in an Introductory Course," *IERI Procedia*, vol. 2, pp. 498–503, 2012.

[5] H. Nishino, "Misfits in Abstractions: Towards User-Centered Design in Domain-Specific Languages for End-User Programming," in *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, OOPSLA '11, (New York, NY, USA), p. 215–216, Association for Computing Machinery, 2011.

[6] J. Freeman, B. Magerko, D. Edwards, T. Mcklin, T. Lee, and R. Moore, "Earsketch: Engaging Broad Populations in Computing through Music," *Communications of the ACM*, vol. 62, no. 9, pp. 78–85, 2019.

[7] J. Wing, "Research Notebook: Computational Thinking—What and Why," *The Link magazine*, vol. 6, pp. 20–23, 2011.

[8] J. L. Weese and R. Feldhausen, "STEM Outreach: Assessing Computational Thinking and Problem Solving," in *2017 ASEE Annual Conference & Exposition*, 2017.

[9] A. J. Magana, M. L. Falk, C. Vieira, and M. J. Reese Jr, "A Case Study of Undergraduate Engineering Students' Computational Literacy and Self-Beliefs about Computing in the Context of Authentic Practices," *Computers in Human Behavior*, vol. 61, pp. 427–442, 2016.

[10] B. Victor, "Inventing on Principle," 2012.

[11] S. McDirmid, "Usable Live Programming," in *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software*, pp. 53–62, 2013.

[12] S. L. Tanimoto, "A Perspective on the Evolution of Live Programming," in *2013 1st International Workshop on Live Programming (LIVE)*, pp. 31–34, IEEE, 2013.

[13] S. Burckhardt, M. Fahndrich, P. de Halleux, S. McDirmid, M. Moskal, N. Tillmann, and J. Kato, "It's Alive! Continuous Feedback in UI Programming," in *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, pp. 95–104, 2013.

[14] M. I. Gorinova, A. Sarkar, A. F. Blackwell, and D. Syme, "A Live, Multiple-Representation Probabilistic Programming Environment for Novices," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, (New York, NY, USA), p. 2533–2537, Association for Computing Machinery, 2016.

[15] H. Kang and P. J. Guo, "Omnicode: A Novice-Oriented Live Programming Environment with Always-On Run-Time Value Visualizations," in *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, (New York, NY, USA), p. 737–745, Association for Computing Machinery, 2017.

[16] T. Lieber, J. R. Brandt, and R. C. Miller, "Addressing Misconceptions about Code with Always-On Programming Visualizations," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, (New York, NY, USA), p. 2481–2490, Association for Computing Machinery, 2014.

[17] M. Beth Kery and B. A. Myers, "Exploring Exploratory Programming," in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 25–29, 2017.

[18] R. DeLine, D. Fisher, B. Chandramouli, J. Goldstein, M. Barnett, J. Terwilliger, and J. Wernsing, "Tempe: Live Scripting for Live Data," in *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 137–141, 2015.

[19] Y. Wu, J. M. Hellerstein, and A. Satyanarayan, "B2: Bridging Code and Interactive Visualization in Computational Notebooks," in *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, UIST '20, (New York, NY, USA), pp. 152–165, Association for Computing Machinery, Oct. 2020.

[20] R. DeLine and D. Fisher, "Supporting Exploratory Data Analysis with Live Programming," in *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 111–119, 2015.

[21] M. Brachmann, W. Spoth, O. Kennedy, B. Glavic, H. Mueller, S. Castelo, C. Bautista, and J. Freire, "Your Notebook is not Crumby Enough, REPLace it," *Conference on Innovative Data Systems Research (CIDR)*, Jan. 2020.

[22] R. A. DeLine, "Glinda: Supporting Data Science with Live Programming, GUIs and a Domain-Specific Language," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, (New York, NY, USA), Association for Computing Machinery, 2021.

[23] G. Andrienko, N. Andrienko, P. Bak, D. Keim, and S. Wrobel, "Visual Analytics Focusing on Movers," in *Visual Analytics of Movement* (G. Andrienko, N. Andrienko, P. Bak, D. Keim, and S. Wrobel, eds.), pp. 131–207, Berlin, Heidelberg: Springer, 2013.

[24] G. Andrienko, N. Andrienko, W. Chen, R. Maciejewski, and Y. Zhao, "Visual Analytics of Mobility and Transportation: State of the Art and Further Research Directions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, pp. 2232–2249, Aug. 2017. Conference Name: IEEE Transactions on Intelligent Transportation Systems.

[25] Esri, "Arcgis Notebooks," 2022.

[26] Esri, "ArcMap - Geodatabases in PostgreSQL," 2022.

[27] L. Chittaro, R. Ranon, and L. Ieronutti, "VU-Flow: A Visualization Tool for Analyzing Navigation in Virtual Environments," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, pp. 1475–1485, Nov. 2006.

[28] N. Hoobler, G. Humphreys, and M. Agrawala, "Visualizing Competitive Behaviors in Multi-User Virtual Environments," in *IEEE Visualization 2004*, (Austin, TX, USA), pp. 163–170, IEEE Comput. Soc, 2004.

[29] W. Büschel, A. Lehmann, and R. Dachselt, "MIRIA: A Mixed Reality Toolkit for the In-Situ Visualization and Analysis of Spatio-Temporal Interaction Data," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, (New York, NY, USA), pp. 1–15, Association for Computing Machinery, May 2021.

[30] N. Ferreira, J. Poco, H. T. Vo, J. Freire, and C. T. Silva, "Visual Exploration of Big Spatio-Temporal Urban Data: A Study of New York City Taxi Trips," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, pp. 2149–2158, Dec. 2013.

[31] W. Chen, Z. Huang, F. Wu, M. Zhu, H. Guan, and R. Maciejewski, "VAUD: A Visual Analysis Approach for Exploring Spatio-Temporal Urban Data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, pp. 2636–2648, Sept. 2018. Conference Name: IEEE Transactions on Visualization and Computer Graphics.

[32] R. L. Peurifoy, C. J. Schexnayder, R. L. Schmitt, and A. Shapira, *Construction Planning, Equipment, and Methods*. McGraw-Hill Education, 2018.

[33] V. Grigoreanu, M. Burnett, S. Wiedenbeck, J. Cao, K. Rector, and I. Kwan, "End-User Debugging Strategies: A Sensemaking Perspective," *ACM Trans. Comput.-Hum. Interact.*, vol. 19, May 2012.

[34] J. Brooke, "SUS: a 'Quick and Dirty' Usability Scale," in *Usability Evaluation In Industry* (P. W. Jordan, B. Thomas, I. L. McClelland, and B. Weerdmeester, eds.), ch. 21, pp. 189–204, London, UK: CRC Press, 1996.

[35] A. Bangor, P. Kortum, and J. Miller, "Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale," *Journal of Usability Studies*, vol. 4, no. 3, pp. 114–123, 2009.

[36] J. R. Lewis and J. Sauro, "Item Benchmarks for the System Usability Scale," *Journal of Usability Studies*, vol. 13, no. 3, 2018.

[37] D. Miedema and G. Fletcher, "SQLVis: Visual Query Representations for Supporting SQL Learners," in *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, (Los Alamitos, CA, USA), pp. 1–9, IEEE Computer Society, oct 2021.

[38] T. Taipalus, M. Siponen, and T. Vartiainen, "Errors and Complications in SQL Query Formulation," *ACM Trans. Comput. Educ.*, vol. 18, Aug. 2018.