











ReVISit 2: A Full Experiment Life Cycle User Study Framework

Zach Cutler , Jack Wilburn , Hilson Shrestha , Yiren Ding , Brian Bollen ,
Khandaker Abrar Nadib , Tingying He , Andrew McNutt , Lane Harrison , and Alexander Lex 

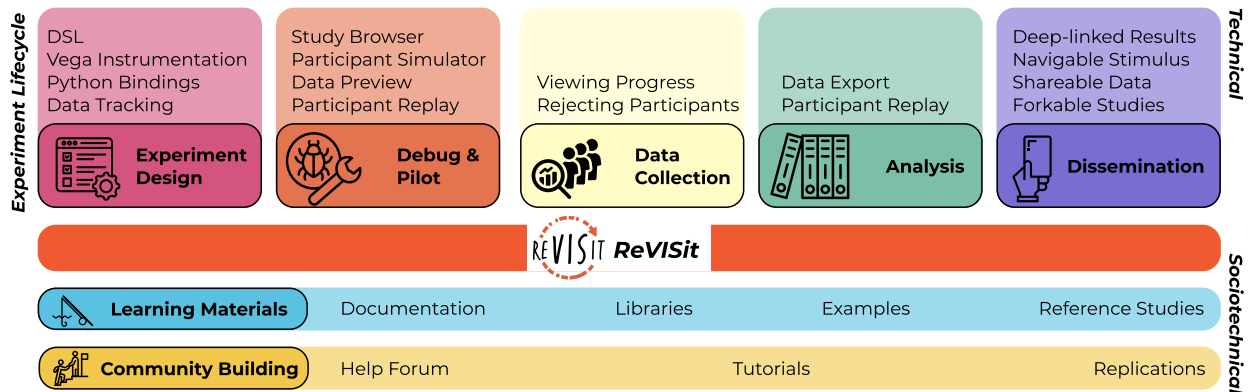


Fig. 1: ReVISit 2 supports each stage of the online user study life cycle. In addition to this linear path, the experiment life cycle is populated by internal loops between stages. For instance, issues revealed in the piloting of an experiment can lead back to the design phase to address those issues, and the process of disseminating an experiment can beget ideas for new experiments or replications.

Abstract—Online user studies of visualizations, visual encodings, and interaction techniques are ubiquitous in visualization research. Yet, designing, conducting, and analyzing studies effectively is still a major burden. Although various packages support such user studies, most solutions address only facets of the experiment life cycle, make reproducibility difficult, or do not cater to nuanced study designs or interactions. We introduce reVISit 2, a software framework that supports visualization researchers at all stages of designing and conducting browser-based user studies. ReVISit supports researchers in the design, debug & pilot, data collection, analysis, and dissemination experiment phases by providing both technical affordances (such as replay of participant interactions) and sociotechnical aids (such as a mindfully maintained community of support). It is a proven system that can be (and has been) used in publication-quality studies—which we demonstrate through a series of experimental replications. We reflect on the design of the system via interviews and an analysis of its technical dimensions. Through this work, we seek to elevate the ease with which studies are conducted, improve the reproducibility of studies within our community, and support the construction of advanced interactive studies.

Index Terms—User studies, crowdsourcing, visualization experiments.

1 INTRODUCTION

Experimental research is a mainstay method to infer causal relationships in visualization, HCI, and related areas [19]. Experiments in visualization (often called user studies) range from perceptual studies [8, 65], to studies of visualization techniques [20, 35], studies of interaction techniques [5, 38], and studies of full visualization systems [44, 52]. Experimental approaches are also used outside of quantitative, controlled research, such as in eliciting expert feedback on systems [53] or designs [26, 37, 42], or understanding insight formation [11, 40].

While these studies historically have been conducted in the lab [8, 15, 72], they are now predominantly run online through crowd work platforms [27, 52, 63], which support rapid participant recruiting and study execution. Most desktop-focused visualization tools are designed to be accessed through the browser, suggesting that web applications are an effective means to deliver study stimuli. These trends suggest that experimental research in the future will be predominantly conducted through browsers and asynchronously (as opposed to in a lab).

Despite the ubiquity of this form of inquiry, conducting these stud-

ies remains difficult for a variety of reasons. For instance, designing effective stimuli presents nuanced challenges (such as effective instrumentation of interactions) that are time-consuming to implement for experienced developers, let alone early-career researchers. Similarly, the design of experiments offers a maze of complex decisions (such as choice of factors, randomization, sampling strategies, and partitioning of conditions) that, when navigated poorly, can invalidate entire experiments. Compounding these design challenges is that making everything work as intended and ensuring that data is collected correctly can be tedious and error-prone.

To help with these challenges, researchers in visualization, HCI, psychology, and other fields draw on software frameworks to support their experiments. Whereas commercial tools, such as Qualtrics [29], excel at survey design, experimental design, and deployment, their closed-source and commercial nature hamper reproducibility, and they do not meet the more specialized needs of visualization experiments—such as sophisticated tracking of complex interactions. A variety of academic systems have been developed to help run studies; however, our discussions with stakeholders from the visualization community revealed that most are used only by individual research groups, use outdated technology, are difficult to maintain, and result in brittle deployments. Other academic systems are specific to certain visualization techniques (e.g., graphs [54]), or cover only a small slice of the experimental life cycle (such as factorial experiment design [50]).

To address these issues, we introduce reVISit 2, a software framework designed to support visualization researchers (and others) across the full life cycle of an experiment, including its design, debugging,

- Zach Cutler, Jack Wilburn, Brian Bollen, Khandaker Abrar Nadib, Tingying He, Andrew McNutt, and Alexander Lex are with the University of Utah.
- Hilson Shrestha, Yiren Ding, and Lane Harrison are with WPI.

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org.
Digital Object Identifier: xx.xxx/TVCG.201x.xxxxxxx

and deployment. ReVISit is a proven system capable of supporting complex study designs at a publication-quality level. It has been used in various studies [10, 11, 42, 46]. ReVISit 2 (referred to as simply reVISit throughout unless otherwise specified) builds on the foundation of reVISit 1 [14], but expands on our previous work in critical sociotechnical and technical dimensions. For the former we have made a concerted effort to reach out to groups who may benefit from this tool via tutorials at universities and conferences, intentionally expanding and extending our documentation, and simplifying the startup cost for experiments by making common experiment components (such as VLAT [41], Mini-VLAT [56], BeauVis [25], etc.) available as libraries.

In addition to quality of life upgrades (such as improved UIs, form elements, data download, testing), on the technical side we make a variety of novel contributions to experimentation platforms. Central among these is an enriched domain-specific language (DSL) for specifying experiments, as shown in Fig. 2. This language includes complex to implement features, such as Latin square participant distribution and randomized attention checks. Complementing this expansion, we also provide high-level Python bindings, *reVISitPY*, which offers Altair [68] style declarative specification and enables even more complex study designs (e.g., factorial designs) with little code. A natural venue for use of this library is Jupyter notebooks, where we offer an experiment prototyping pipeline wherein developers cannot only design their experiment, but see and debug the experiment within the notebook, and even access collected data. This pipeline enables researchers to design and test their study, as well as prototype analyses from within the notebook.

In contrast to other platforms, reVISit provides sophisticated tools for debugging and piloting, such as a study browser, participant view simulator, and data previews, as well as tools to manage data collection in ongoing studies. We also make strides in simplifying the specification of visualization stimuli by making Vega visualizations first-class citizens, allowing us to provide automated provenance tracking across user interactions with Vega visualizations, which supports both low-level interaction analysis and fine-grained replay (Fig. 5).

We evaluate this collection of contributions via two strategies. First, we demonstrate its expressiveness through a collection of three study replications, where each study demonstrates different capabilities of reVISit (such as dynamic sequencing, capturing speech, and analyzing provenance). Then we reflect on the design choices made in the system via three interviews with reVISit users and a close reading of the system lensed through Jakubovic et al.’s [30] Technical Dimensions of Programming Systems (TDPS). The combination of our technical design decisions and commitment to open source enables customized dissemination, which is useful during the review and reading process (reviewers and readers alike can explore studies easily and thereby build trust in results), as well as enables reproducibility, as data can be shared and studies can be forked easily. Through this work, we seek to elevate the ease with which studies are conducted, improve the reproducibility and openness of studies within our community, and support the construction of advanced interactive studies.

2 RELATED WORK VIA STAGES OF A USER STUDY

Creating and conducting user studies is a complex process with iterated phases that each present challenges, as well as opportunities, for tool support. To focus such challenges, we divide the user study process into five stages, shown in Fig. 1, which we refer to as the experiment life cycle. In practice, this process is not linear but contains loops; for example, it is expected that after a pilot the experiment will be refined at the design stage to address issues that the pilot surfaced.

We detail each stage and situate ReVISit and prior work (particularly the tools in Table 1) within that stage. We note that many tools are no longer maintained, limiting their practical utility. For a simple survey-based study, there are many easy to use open and commercial tools similar to Google Forms [21]. For custom experiments, jsPsych [13], reVISit, and Qualtrics [29] present the most viable offerings—each catering to various audiences via differing approaches, as we discuss.

2.1 Experiment Design

In the experiment design phase, study designers decide *what* participants see or do, and *when / under which conditions* they see it. We separate this phase into two activities: stimulus and experiment design. These designs are typically specified via a GUI, a library in a general-purpose programming language, or a DSL. These choices impact usability (e.g., GUI-based tools can be used by study designers with no programming skills) and expressivity (libraries and DSLs are potentially more expressive). Notably, only commercial tools (Qualtrics, Google Forms, etc.) provide the means to specify studies via GUIs, which may be due to the development effort associated with GUIs.

Stimulus Design. In most studies, a **stimulus** is presented to participants with the intent to elicit a response or behavior. In visualization research, stimuli typically take the form of images, text, video, audio (for sonification), objects (for physicalization), interactive applications, or combinations thereof. Digital stimuli are often designed to be viewed on a desktop screen, but other displays, such as AR/VR, large display environments, or mobile phones, are also used. In practice, tools that support non-desktop environments either are specifically focused on doing so (as in Flex-ER’s AR/VR support) or are commercial tools (as in Qualtrics’s support for mobile).

Existing user study tools support a range of stimulus customization and features (cf. *Stimuli* in Table 1). Several commercial tools are designed to conduct **surveys**, such as SurveyMonkey [64], and Google Forms [21]. PsyToolkit [62] is an open-source tool that also focuses on surveys. Although it is typically possible to include images and video as stimuli in a survey, they do not allow study designers to embed custom stimuli (such as web applications), and instead focus on the straightforward creation of form elements. Other tools focus on a **specific type of stimulus**, such as network visualizations (GraphUnit [54]), images (ETK [67]), 3D surfaces (EvalViz [49]) or a specific modality, such as AR/VR (Flex-ER [43]). These tools offer customization options within their domain, but forgo arbitrary stimuli. However, their focus enables rapid stimulus creation within the domain. Several tools enable study designers to integrate **custom stimuli**, typically in the form of bespoke UIs. Some tools enable combinations of custom stimuli with more structured form elements (e.g., jsPsych [13], Qualtrics [29] and reVISit), and others require designers to write custom code for form elements (e.g., experimentr [23], FROE [31], Touchstone1 [45]).

Related to stimulus choice is the decision of how to record responses and behaviors. Data is frequently collected via *form elements* (e.g., dropdowns, radio buttons, or text boxes), and is often encapsulated as common rating systems (e.g., Likert scales [34]). Recordings of user behavior during the study are also common (as interaction logs or screen recordings), as well as more explicit user measurements media (such as through video or eye tracking). Qualtrics, jsPsych, Evalbench, and reVISit automatically log **browser events**, such as mouse movements or key presses. Survey-centered and domain-specific tools make use of knowledge of their domain to automatically record data, whereas custom stimuli (typically being unconstrained HTML) require manual instrumentation. ReVISit includes automatically instrumented elements (forms and Vega programs) and means for instrumenting custom stimuli to support post-study replay and analysis. **Audio recording** is rarely supported in the study platforms we surveyed; only jsPsych, Qualtrics, and reVISit have built-in audio recording capabilities, although recording could be implemented as part of a custom stimulus in many of the tools. Only reVISit offers automatic transcription.

Factors and Sequence Design. Many studies test different conditions (independent variables), which in study design are commonly referred to as *factors*. Typical factors in visualization studies are visual encodings, datasets, or tasks [33]. For example, a study comparing node-link diagrams (NL) to adjacency matrices (AM) will vary the factors of visual encoding (NL, AM), datasets (e.g., large vs. small, sparse vs. dense), and tasks (path finding vs. cluster identification). Two common designs use factors *within subjects* (all participants see all values of a factor) and *between subjects* (participants see a subset of factor values). Studies frequently combine *within subjects* design for some factors and *between subjects* design for others, to create *mixed design* studies.

Name	Stimuli	Interaction Logging	Libraries	Basic randomization	Dynamic randomization	How to create	Study navigation	Records response data	Data preview	Video recording	Audio recording	Precise Timing	Eye tracking	Other devices	Participant replay	Open Source	Active	Specialty
Google Forms [21]	SURVEY	N	N	Y	N	GUI	Y	Y	Y	N	N	N	N	Y	N	N	Y	Images
psyt toolkit [62]	SURVEY	N	Y	Y	N	GUI	N	Y	N	N	N	Y	N	N	N	Y	Y	
ETK [67]	DOMAIN	N	N	Y	N	LIB	N	N	N	N	N	N	N	N	N	Y	N	
EvalViz [49]	DOMAIN	N	N	N	N	LIB	N	Y	Y	N	N	N	N	N	N	Y	N	
Flex-ER [43]	DOMAIN	N	N	Y	N	DSL	Y	Y	N	Y	N	N	N	N	N	Y	N	AR/VR
GraphUnit [54]	DOMAIN	N	N	N	N	GUI	N	Y	N	N	N	N	N	N	N	Y	N	Network Visualizations
Evalbench [2]	CUSTOM	Y	N	Y	N	DSL	N	Y	N	N	N	N	N	N	N	Y	N	
Experimentr [23]	CUSTOM	N	Y	Y	N	LIB	N	N	N	N	N	Y	N	N	N	Y	N	
FROE [31]	CUSTOM	N	N	Y	N	LIB	N	N	N	N	N	N	N	N	N	Y	N	
VisUnit [33]	CUSTOM	Y	Y	Y	N	LIB	Y	Y	N	N	N	N	N	N	N	Y	Y	Experimental Design
Touchstone1 [45]	CUSTOM	N	N	Y	N	GUI	N	Y	Y	N	N	N	N	N	N	Y	N	
PsychoPy [58]	C & S	N	N	Y	N	GUI+LIB	Y	Y	N	N	N	Y	N	N	N	Y	Y	
jspsych [13]	C & S	Y	Y	Y	Y	LIB	N	Y	N	Y	Y	Y	Y	Y	N	Y	Y	
Qualtrics [29]	C & S	Y	Y	Y	N	LIB	Y	Y	Y	Y	Y	Y	Y	Y	N	N	Y	Experimental Design
Sweetpea [50]	N	N	N	Y	N	GUI	N	N	N	N	N	N	N	N	N	Y	Y	
Touchstone2 [16]	N	N	N	Y	N	DSL+GUI	N	N	N	N	N	N	N	N	N	Y	N	
ReVISit 1 [14]	C & S	Y	N	N	N	DSL	Y	Y	N	N	N	N	N	N	N	Y	Y	
ReVISit 2	C & S	Y	Y	Y	Y	DSL	Y	Y	Y	N	Y	N	N	N	Y	Y	Y	

C & S – Custom and Survey, LIB – Library in General Purpose Language, DSL – Domain Specific Language, DSL+GUI – DSL and GUI

Table 1: There are a wide variety of tools for online visualization-based experiments. These range from repurposed survey tools (e.g., Google Forms, or similar tools not listed here such as SurveyMonkey [64]) to domain-specific tools (e.g., GraphUnit [54] for graphs or Flex-ER [43] for VR).

Once each factor is decided on, designers need to consider the order and conditions in which stimuli are presented—the *sequence*. The simplest design is a *fixed* sequence, wherein all participants see the same stimuli in the same order. Fixed sequencing may introduce confounders such as order effects (e.g., due to learning) [19]. To address this, it is common to (partially) *randomize* the order in which stimuli appear, or in the case of between-subjects studies, randomly show different participants different stimuli. However, purely random distribution of stimuli may lead to limited coverage when many factors are considered with few participants. To improve balance *Latin square* sequencing is often used, which ensures that stimuli will be seen equally frequently while controlling for order effects.

Some studies implement more complex ordering that is not predetermined, and instead is dependent on the answers a participant gives during the study (e.g., to make the next question harder if the previous answer was correct). We call such sequences *dynamic* sequences. For example, staircase designs [24, 70] involve iterated presentation of stimuli to find, for example, a perceptual threshold of some kind. Such dynamic sequencing is rarely supported, and often is implemented ad hoc (such as by embedding these designs into the stimuli themselves).

Some tools specialize in sequence design, while not providing support for other phases of the study life cycle. Touchstone2 [16] features a GUI for creating and sharing sequence designs. Complex sequence designs with multiple randomization strategies are creatable via GUI, and designs can be exported to configuration files that look similar to reVISit’s. Sweetpea [50] is a Python library for sequence design that takes in a series of factors and produces experimental sequences, similar to reVISitPy. Sweetpea includes a rich notion of constraints across factors that support more succinct expression of complex experimental designs than our current version of reVISitPy. VisUnit [33] supports explicitly creating sequences from specified design factors (stimulus, dataset, tasks). Sweetpea, Touchstone2, and VisUnit do not support dynamic sequences, as their sequences cannot be adjusted based on user responses. Like reVISit, jsPsych [13] supports dynamic sequencing via designer-defined functions that are called as trials get completed.

2.2 Debug & Pilot

Once an initial study has been created and a design is decided, there is a phase in which study designers ensure that their stimulus, sequence, and data collection all work as intended. Debugging is commonly done by designers taking their own study (often many times) to test that

everything behaves as intended. Efficient debugging requires being able to easily browse a study, without needing to take it from beginning to end. GUI-based platforms typically provide the means to quickly navigate to a specific stimulus or condition, while library-based tools often do not provide such support and require developers either to know a URL or to take a study from beginning to end.

Piloting is done with participants who were not involved in the study (either via colleagues in so-called “down-the-hall” testing or through recruitment of preliminary participants online), and is done to identify problems, validate data collection, and collect preliminary data. Such preliminary data is often used to conduct a power analysis to estimate the number of participants required to find statistical significance of hypotheses [9]. Most tools do not provide advanced piloting support (e.g., replays). ReVISit’s support for this phase offers an important facet of our technical contribution. Specifically, reVISit provides a study browser to navigate to different components with a participant view that shows a sequence that could be assigned, participant replays, and the ability for designers to take their own study to generate data. Some tools (e.g., jsPsych) can generate artificial data via simulation, which designers can use to identify experimental design problems and test analysis methods. ReVISit does not currently support simulation, however we believe that our DSL will make it straightforward to implement such a feature.

2.3 Data Collection

The data collection phase begins with recruiting participants whose data is planned to be used in the final analysis. Participants can come from various sources, such as crowdwork platforms (e.g., Amazon Mechanical Turk [28] or Prolific [55]), volunteer-based or gamified platforms (e.g., LabInTheWild [59]), and from networks such as mailing lists or social media. Although recruiting is largely orthogonal to study design, platforms such as Qualtrics offers (paid) access to participants, LabInTheWild provides basic study templates, and Prolific and Mechanical Turk have basic survey capabilities. During data collection experimenters must mind incoming data, which may involve rejecting fraudulent participants, identifying bugs, or examining initial data. Some tools have data previews (e.g., Qualtrics, Google Forms, and reVISit) that can be used for observability or simple analytics.

Once collected, this data needs to be stored somewhere. Commercial tools typically provide data hosting as part of their service, sometimes for a fee. In contrast, non-commercial tools leave data hosting to the

study designer. For instance, reVISit primarily uses Google Firebase (a real-time-focused document database) for storage but does not provide a hosted solution. Instead, study designers must set up their own Firebase accounts—while the specifics differ, this is broadly typical.

2.4 Analysis

There are many sophisticated methods and tools to support the analysis phase of a study. As for participant recruiting, analysis tools are largely orthogonal to study frameworks. Statistical analysis is well supported by a multitude of environments and tools. For qualitative analysis, commercial tools (e.g., MaxQDA [69]) are frequently used to help with the coding process. Various qualitative analysis tools have been created by the visualization community, such as VisTA [17] or CoUX [61]. Other tools specialize in event sequence analysis [51] (such as might be emitted by reVISit) or in analysis of eye tracking data [7]. While reVISit has some analysis capabilities, they are primarily designed to serve the debug/pilot and data collection stages—which is aligned with reVISit’s design philosophy of providing functionality that is not already covered by high-quality open tools.

2.5 Dissemination

Faithful dissemination of study procedures, data analysis, and results is crucial for making results scrutinizable, reproducible, and ultimately building trust in the outcomes. A common approach to disseminate the details of a study is to include screenshots of the procedure or exports of a survey in a read-only format (e.g., from Qualtrics). However, with web-based tools (see “Open Source” in Table 1), study designers can share both a link to the experiment as well as the code used to design it. Study data is commonly shared via hosting platforms such as OSF or GitHub. ReVISit has a unique ability to share the data with the study, so that each participant’s actions can be reviewed. For example, when including a screenshot of a response, authors can include a deep link to the stimulus page with a participant’s actions, as in Fig. 5. We emphasize that reVISit is unique in its level of commitment to and extensive support of reproducibility and transparency.

2.6 Relationship to Previous reVISit Versions

As the name would suggest, this work extends an earlier version of reVISit. A system that shares the name with reVISit focused on the analysis of user logs [51]. It did not provide study scaffolding, but instead explored topics such as event sequence analysis. It is the root of ideas for “study rehydration”, i.e., the replay of a participant’s analysis session provided in reViSit 2. The reVISit study framework we report on here was started in 2022, with funding from the National Science Foundation. A VIS 2023 short paper [14] describes the principles behind reVISit: a DSL for defining experiments, components that contain stimuli, data collection that includes provenance tracking, and a process to compile everything into deployable web-based experiments. The first version recommended for public use, reVISit 1.0, was released in June 2024, followed by a 2.0 release in January 2025. We continued to expand on reVISit with crowdsourced think-aloud studies in a CHI25 paper [11]. The key difference to prior versions is that reVISit now is a stable, well-documented, ready-to-use experimental platform, with early signs of community adoption (Sec. 5.1). In addition, we make several technical contributions (Sec. 1).

3 SYSTEM TOUR

Next, we give a tour of ReVISit, highlighting notable features. In developing these features, we centered a design goal of making experiment design and deployment as frictionless as possible (for our target audience of technical scientists who might not be software engineers), while maintaining scientific sovereignty and without redoing what others already do well (e.g., participant recruitment, data analysis).

For *sovereignty*, ReVISit is deployed as a static web page (ensuring that there is no server to maintain). Deploying a static web page avoids vendor lock-in (users can just change the website as they see fit) and supports long-term dissemination stability (studies are static and do not change as reVISit changes unless the designer intentionally does so).

```
StudyConfig =
{
  sequence      ::= Block
  components    ::= {Name →< Component >}
  importedLibraries ::= LibraryName[]
  baseComponents ::= {Name →< Component >}
  studyMetadata ::= ...

Block =
{
  order      ::= fixed | random | latinSquare | dynamicθ
  components ::= (Name | Block)[]
  numSamples ::= N+
  interruptions ::= (Deterministicθ | Randomθ)[]
  skip        ::= SkipCondition[]

Component =
{
  compType ::= Markdownθ | Reactθ | Imageθ | Websiteθ | Formθ | Vegaθ | ...
  responses ::= Response[]

Response ::= Numerical | ShortText | LongText | Likert | Dropdown | Slider
          | Radio | Video | Checkbox | Reactive | Matrix | ...

SkipCondition ::= BlockConditionθ | RepeatedBlockConditionθ | ...

Name = Symbol, LibraryName = Symbol, NewElements in for ReVISit 2
θ denotes arguments, <X> is partial (or complete) definition à la TypeScript
```

Fig. 2: The reVISit grammar with configuration details elided.

For *non-repetition*, we emphasize that ReVISit is not a database, a recruitment platform, a GUI experiment builder, or a single site analysis platform. Identifying that these are strengths of others, we design our system so that it takes advantage of those extant capacities, following recent guidance to “lean on existing technological and social infrastructures” [3]. For instance, Prolific works well for study recruitment, and so we instead support the use of any recruitment platform; statistical analysis tools in, e.g., R, are superior to anything we could provide, so we focus on compatible exports.

Complementing these intents is a commitment to sociotechnical support, which we do via a mindfully maintained collection of artifacts (including [tutorials](#), [documentation](#), and [examples](#)) as well as community efforts (such as a help forum and in-person tutorials).

3.1 The DSL

The first step in setting up a reVISit experiment, after forking the base repo, is to start designing the experiment specification using our domain-specific language (DSL) (we give the grammar for this language in Fig. 2). The root includes a list of named components that can be used in any sequence block, as well as a collection of component templates (*baseComponents*) used to partially define other components via inheritance. This language involves composing a collection of experimental “blocks” (in *sequence*). Each block can contain stimuli (*components*) or nested blocks, as well as basic logic for controlling the order of components, and more fine-grained control, such as for inserting interruptions (such as for attention checks) and whether certain blocks should be skipped (such as due to wrong answers).

The reVISit DSL is a JSON-based DSL [48], in which experiments are specified through standalone JSON files. These files are type checked through both a JSON Schema validation of the syntax as well as a secondary linter which identifies basic specification errors such as the presence of un-used components in the *StudyConfig*.

Yet, specification through JSON is sometimes noted as being undesirable or messy syntax for DSLs [48]. Moreover, reVISit programs can be enormous—with some reaching tens of thousands LOC due to repetition of structures to combine multiple factors. To address these issues, we developed a Python wrapper for the reVISit DSL called [reVISitPy](#). These Altair [68] style bindings allow study designers to make use of the full expressivity of general-purpose languages—allowing for variables, complicated looping logic, and so on. In the interest of keeping the library’s syntax familiar to visualization developers (who would likely also be study designers), we intentionally mimicked Altair’s use of structured-like method chaining.

Echoing how tools like Altair are often used in the context of Jupyter notebooks, reVISitPy is designed to work well within notebooks. To this end, reVISitPy supports in-notebook previews of the experiment and the collected data. Test data can be retrieved from the preview, so that data wrangling and analysis can also be prototyped in the same notebook—see the appendix or this [example](#). This supports rapid workflows wherein the experiment designer composes a reVISitPy program, views the effects of their design, and makes iterative adjustments.

Some extremely custom designs or those implementing high-level constraints (à la Sweetpea [50]) are more straightforward to express via direct specification of the JSON DSL. However, the Python bindings aim to simplify the process of making small prototypes and simplify experimental design more generally by keeping it centered in a single computational notebook. For example, during a team retreat, we prototyped our JND replication (Sec. 4.1), from dataset generation, to stimulus generation (via Altair), to experiment specification, testing and piloting, and preliminary analysis, all from within a single notebook.

3.2 Stimuli

To a study participant, the most evident part of an experiment is the stimuli that they interact with. ReVISit experiments can include a variety of types of stimuli, including form elements (numerical inputs, sliders, etc.), markdown files (such as for participant instructions, consent forms, and so on), images, and videos. Naturally, any list of prebuilt components will be incomplete, and so we support custom components by allowing users to supply generic web-components as well as React components which can be smoothly integrated into reVISit’s full data and provenance tracking capabilities by use of the track library [12].

However, merely exposing an endlessly customizable component takes the focus off of visualization. In experiments—and visualization practice more generally—a common way to create visualizations is through the use of DSLs, such as Vega [60]. These DSLs simplify the specification of often repetitive structures (such as data management or scaling code). Echoing this approach, we include Vega programs as first-class stimuli, allowing them to be specified directly in the reVISit DSL or imported from a separate static file. We automatically instrument these programs with provenance tracking, such that the state of the Vega signals is recorded as users interact with the programs. Fine-grained state tracking supports similarly fine-grained participant replay (as we discuss below), such as being able to view specific hover states and mouse moves. Further, consistent with other custom components, we offer a custom Vega signal callback that can be called to set the answer inside of reVISit’s reactive responses while using Vega, which allows a user to interact with the visualization and click on elements to set the answer that will be recorded by reVISit.

Many experiments use standardized surveys or other common stimuli as part of their design—e.g., VLAT. To support this usage, reVISit provides a collection of libraries that support various common tasks (as in Fig. 3), including demographics questionnaires, color vision deficiency tests, or visual literacy tests [41, 56].

3.3 Sequence

The next aspect apparent to a participant is the order in which components appear. Each *block* in our DSL has a defined *order* in which its child components are shown. ReVISit includes affordances for specifying the order and relationship between components that support rich customization and experimental designs (e.g., between-subjects, within-subjects, and mixed designs).

Fixed order shows components in the order in which they are listed, whereas random shows them in a random order (per participant). Yet, in studies with limited participants and many conditions, random ordering does not guarantee sufficient coverage of the study cases. Latin square orderings are commonly used to provide such guarantees [19], being particularly useful as a way to mitigate order effects.

While these strategies cover many different designs, they do not capture all possible sequences. Some studies rely on the answers to previous questions to determine the next stimulus that will be shown to a participant. These cannot be specified in our DSL and require custom

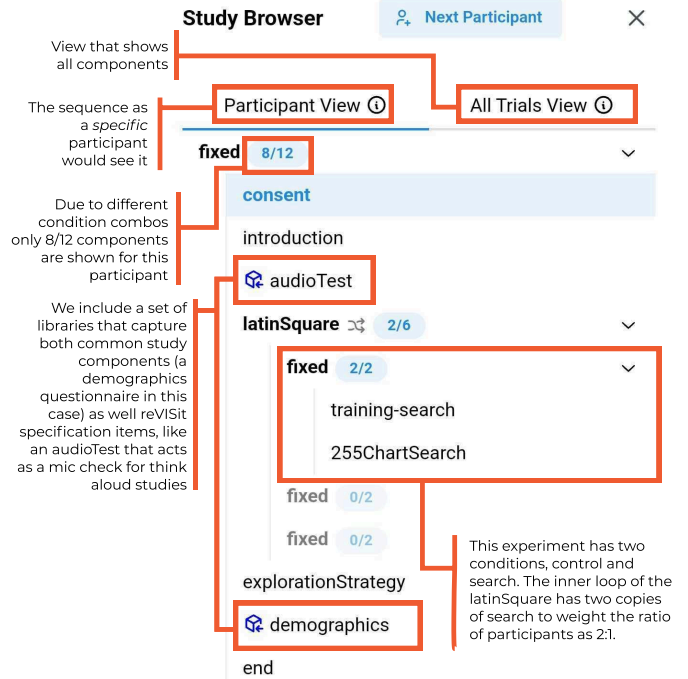


Fig. 3: The study summary for our search study replication. Each participant flows through the experiment, first seeing consent, then introduction, and then is sorted into one of three conditions (the fixed subsections of the Latin square). This summary can be seen [here](#).

logic to implement. To enable more nuanced types of designs we support dynamic ordering, in which a study designer provides a bespoke JS function that is called repeatedly as the participant progresses through a block to determine their next task. For example, consider a study design where participants see different stimuli based on the accuracy of their earlier answers—such as how US-based Graduate Record Examinations (GRE) adaptively alters the difficulty of topic sections based on the success rate of previous sections (see Sec. 4.1 for an example). Dynamic ordering could be accomplished within a stimulus; however, using dynamic functions enables various useful reVISit features (e.g., logging, participant replays, or stimuli navigation).

In addition to linear progression through an experiment, some studies may require periodically inserted components separate from the experiment logic (such as attention checks) or non-linear jumps (such as ejecting a participant if they fail a training). Each of these tasks are supported by blocks through their interruption and skip logic, respectively. Although these functionalities could be orchestrated through a collection of dynamic checks and custom components, we elevate these to language-level features to highlight their importance in study design.

We argue that reVISit can model more diverse study designs than alternatives, such as Qualtrics [29] or VisUnit [33]. For example, VisUnit lists staircases as a design form that it cannot model. ReVISit can model such designs, as in our staircase-based replication in Sec. 4.1, which is enabled by our robust native and dynamic sequencing.

3.4 Study Browser

Understanding the architecture of a study, such as which blocks are contained within which other blocks or which stimuli will be seen by what fraction of the population, as well as navigating to specific stimuli, are difficult challenges in the debug & pilot phase of an experiment. ReVISit addresses these through a *study browser*, shown in Fig. 3, which organizes experiments into a single summative view describing the mechanical architecture of the study. The table of contents-like structure is situated on the right hand side of the application while in “admin” mode, juxtaposed to the rest of the experiment (as in Fig. 5-A). Clicking on a stimulus instantly navigates to that stage of the experiment. For instance, clicking on “255ChartSearch” in Fig. 3 brings up the specific search stimuli of interest (cf. Fig. 7), thereby speeding up debugging.

The study browser has two views: a “Participant View” (active in Fig. 3), which shows the sequence just as a particular participant would see it, including the order and the selected subset of trials. Clicking on “Next Participant” rebuilds the sequence for another participant, thereby enabling experiment designers to check that all sequencing is specified correctly. The “All Trials View” makes all components immediately accessible, independent of whether they appear in the sequence of a particular participants, enabling designers to quickly navigate to each component. Finally, the study browser surfaces response data (e.g., right or wrong) when replaying a participant run.

3.5 Data

After the study is in place and launched, data collection commences. ReVISit supports several *storage engines*, including browser-storage, Google Firebase, and Supabase; custom storage engines can be implemented by interested users. We plan to expand our existing storage engines to support other common data hosting services in the future. In addition, ReVISit includes a variety of different affordances to tend to this data across the experiment life cycle. Although we refrain from re-implementing mature analysis tools, some analysis steps are better situated within the system rather than as an external analysis loop. After the experiment, data can be exported (e.g., as JSON or CSV) and used in custom analysis workflows.

The first analysis feature is **participant replay**, which allows study designers to watch individual participant trials. ReVISit’s rich notion of provenance allows for straightforward rehydration of study stimuli (such as the Vega stimuli described above), such that individual actions like keystrokes in a form can be observed, as can be seen in Fig. 5-A. The timeline and rich event logs at the bottom enable analysts to investigate each step a participant took, supporting analysis of the data—for instance in our search replication (Sec. 4.3)—as well as debugging of pilot studies. These interactions are reified as our **timeline view** (Fig. 5B), which shows a timeline of all of the tasks a participant took across an experiment, and also shows responses on tooltip. While we record mouse movement, variation in monitor sizes and aspect ratios make visualization of mouse movements difficult.

A related useful form of data for the prototyping phase is the **audio solicited through think-aloud studies**. Think-aloud can be used for usability evaluation and insight elicitation [11], such as in our search replication (Sec. 4.3). A novel usage would be to allow pilots to verbally describe their thought processes as they are doing it—rather than requiring them to recollect after the study or take notes.

Lastly, we provide a collection of simple analytic views that allow for quick sanity checking about statistics in the experiment. These include a minimalist tabular view for spot checking the data and simple analytics views that summarize participant performance by trial. The intent of these views is analogous to how Google Forms offers a collection of basic summative charts about the collected data as a way to check the distribution of things rapidly.

3.6 Post Study

After a reVISit study is complete, the work it needs to do is not entirely finished. Concerns related to ensuring the long-term accessibility, transparency, and replicability of the experiment are central to reVISit’s design and so we offer a collection of features to support these tasks.

The first focus is on the explainability of the experiment, which reVISit supports by helping reviewers and other interested parties understand the study and its results. We optionally allow non-verified visitors to navigate the experiment without having to fully take the study (as a curious reviewer might wish to do)—such as at revisit.dev/replication-studies/HeatmapJND-study. We also provide an option to deactivate data collection so as to avoid collecting data from reviewers or other unwanted sources. Study admins can also give access to the same data downloads, as well as timeline and replay visualizations, that study creators had. Similarly, reVISit enables authors to deep-link into individual participants’ trials, as we do for all reVISit figures (such as Fig. 5), enabling readers to scrutinize the context.

The second focus is long-term reproducibility. As noted above, the intended workflow for reVISit is for study creators to fork the GitHub

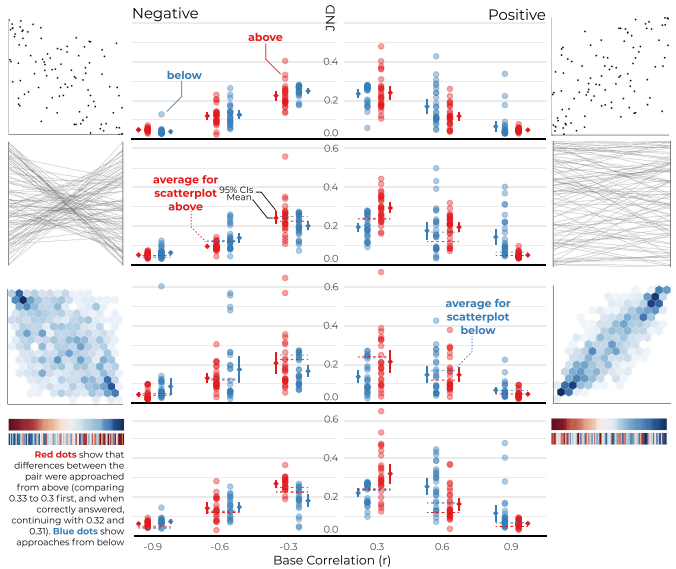


Fig. 4: Distribution of JNDs across eight conditions—four types of visualizations and positive and negative correlations for each. Scatterplots are best at showing differences in correlation. Hexbin plots show individual differences (see outliers), but are best at showing differences for low r values. Lower JNDs indicate easier detection of correlation differences. JNDs are generally lower for high correlation values.

repository, thereby creating a snapshot of the current version of the reVISit code, which will remain stable despite updates to the main repository. Additionally, having all of the code required to run the study in the same repository, including stimulus, the reVISit tool, and the configuration for experiment design, is designed to make future replications or modifications to the study simple. Although there are vulnerabilities to this architecture—such as library dependencies catastrophes [1] or changes in browser functionality—this approach offers substantially more straightforward access to experiment stimuli and architecture than comparable closed-source alternatives. We intend to continue to enrich the suite of features offered in this area, such as automatically creating archival screenshots and videos of the experiment that will be fully resilient to changes in browser technologies.

4 FIELD TEST: REPLICATIONS

To demonstrate the utility of reVISit on real-world experiments, we replicated three extant studies [18, 24, 26]. We selected these studies to demonstrate different aspects of reVISit—for instance, the first study demonstrates our dynamic sequence capability. In addition, we add new variations to each replication—such as by testing additional conditions or leveraging additional data collection modalities—so as to not merely replicate but to expand the previous studies. In total, we recruited 460 participants for the studies, of which 440 were recruited via crowdsourcing, and 20 were visualization design experts recruited via social media. All studies were pre-registered on OSF, and are available at revisit.dev/replication-studies. In this section, we sketch the studies and key results, while focusing on lessons learned as each study was executed in reVISit. More details are available in the Appendix.

4.1 Replication of Ranking Visualizations of Correlation

We replicated Harrison et al.’s [24] study on just noticeable differences (JND) of correlation values in varied visualization techniques. Harrison et al. found that different chart forms vary in how precisely participants can distinguish two similarly correlated plots of two-dimensional data (scatterplots work best), and that the JND is smaller when the correlation values r are higher. We run the study with two already examined techniques (scatterplots and parallel coordinate plots) as well as two new ones (hexbin plots and sorted heatmaps). We test each condition with different correlation values, (see Fig. 4). As in the original study, we use 100 points in each condition, except for hexbins, which are typically used for larger datasets, where we use 1000.

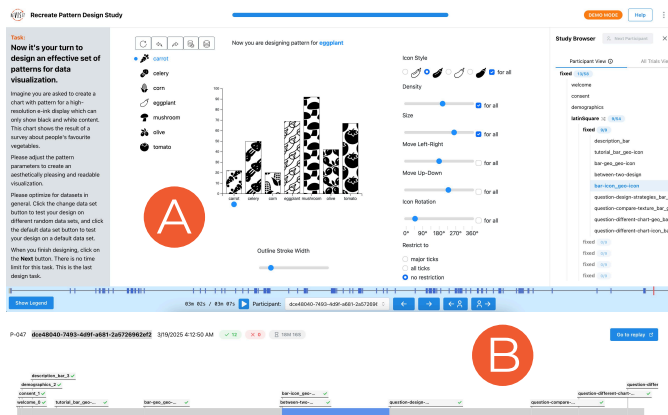


Fig. 5: A participant's result from our texture study shown in analysis mode. (A) Participant replay shows events and navigation options at the bottom. (B) The participant timeline gives summary information and serves as an entry point. [[↗ View Result](#)]

We selected this experiment because (a) it uses a challenging experimental staircase design that is not typically supported by experimental frameworks, and (b) we were interested in expanding the set of tested visualization techniques. We used reVISit's dynamic sequencing function to implement a staircase design, adjusting follow-up pairs of correlations based on previous responses. We also terminated trials early based on statistical tests of performance. We added an attention check involving an obvious pair of correlation values ($r = 0.01, 1$) placed so that we could detect thoughtless clicks. We recruited 30 participants using Prolific for each condition, totaling 240 participants.

Results. Our experiment replicates Harrison et al.'s findings for scatterplots and PCP plots (see Fig. 4). Our results show even narrower confidence intervals, which may be attributable to the improved attention check or to the other advantages of our experiment, such as more robust data quality control and a more modern interface. We find that hexbin plots are suitable to visualize correlations for large datasets, and seem to even outperform scatterplots for low correlation values. However, several poorly performing participants produced outliers, which indicates that hexbins might not be universally understood. Heatmaps perform the worst of all stimuli, although not disproportionately, suggesting they may be useful for space-restricted mediums.

Lessons Learned. For our study design, we opted to run the conditions as separate studies on Prolific, rather than assigning conditions via reVISit within a single study. An alternative Latin square design within one study can become unbalanced based on returned studies on Prolific, which we plan to address in the future with a plug-in for Prolific, so that a Latin square entry would be made available again when a participant returns a study. Separating the conditions into different studies granted us more control over participant numbers per condition, but came with tradeoffs as we had to duplicate study configurations across the conditions. Although we could have leveraged reVISitPy to generate multiple study configurations and reduce overhead, more built-in support for between-study design may be useful.

We implemented the staircase procedure using the dynamic functionality in reVISit, rather than embedding logic directly in the stimulus, as the original study did [23]. reVISit's dynamic functions allowed us to develop, debug, and pilot studies consistently across all conditions and maintain a clean and centralized implementation of the staircase logic.

4.2 Replication of Pattern Design Study

Next, we replicated He et al.'s study [26] on designing black-and-white patterns for visualization. They asked visualization experts to create and then rate designs for different patterns (geometric and iconic) and chart types (bar, pie, or map). The authors then coded participants' responses and categorized the goals into those related to readability and aesthetics, and summarized corresponding design strategies.

We chose to replicate this study to show that reVISit (a) supports a mixed study design (between-subject design on chart type and within-

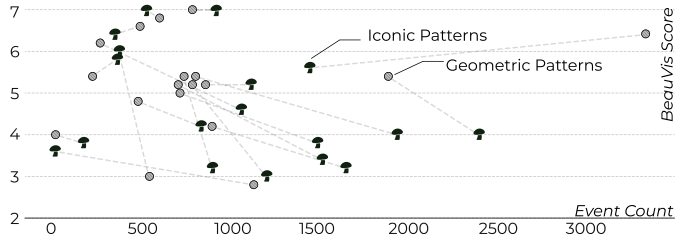


Fig. 6: Relationship between interaction events and BeauVis [25] scores in our pattern study. Each pair of points shows a single participant.

subject design on pattern order) and (b) that existing tools can be integrated into the reVISit framework. In contrast to the previous study, we fully instrumented the stimulus so we could analyze design strategies and reason about the number of interactions associated with good or bad designs. We judged each design internally (via the BeauVis scale [25]), in place of a follow-up crowdsourced study as the original did. We recruited 20 participants (design experts) from social media.

Results. Our results replicated He et al.'s in finding similar design goals and design strategies. Participants' design goals focused on distinguishability (15×), visual clarity (5×), semantic association (5×), visual pleasure (7×), and visual balance (3×)—see Fig. 6 for a distribution of scores. Two new strategies emerged: one participant attempted to create new shapes by playing with the basic geometric shapes to create new shapes, and another designer experimented with dot density to create different shades of gray, à la halftoning.

Through the reVISit replay interface, we discovered common strategies that designers use. A typical workflow involved an initial exploration of different default patterns, followed by iterative refinement of individual patterns, and finally testing and refining the design with different datasets (e.g., [↗ as this participant did](#)). For iconic patterns, a commonly observed behavior was testing different icon styles (like [↗ this participant did](#)). We noticed these design strategies only by using participant replay, highlighting the replay's value in uncovering subtle yet impactful design behaviors.

Lessons Learned. In our study configuration, we used a Latin square to order components. However, we later realized that this approach is not well-suited for studies recruiting participants via social media, as we did. A reVISit Latin square is calculated based on the participants who start the study, rather than those who finish it—each initiated study takes an entry from the Latin square. However, many social media users click on the study link but do not complete it, leading to an imbalance between conditions. Although study designers can manually reject incomplete entries to rebalance, we plan on adding a timeout mechanism that automatically rejects incomplete entries and returns their entries to the Latin square. We recommend that studies with small sample sizes or recruiting via social media use random orderings.

Instrumenting an existing interactive visualization tool with Track [12] to enable full rehydration required considerable effort. To support replay, we needed to capture every user interaction and reconstruct the visual state of the application from the captured provenance data, which required building a stable history management system. Most existing visualization libraries, however, do not support such functionality [73]. To lower the burden when using a legacy system, a combination of screen capture and logging could reduce the technical burden while achieving most of the functionality of full rehydration.

4.3 Replication of Search in Visualization Study

Finally, we replicated Feng et al.'s study [18] on search functionality in interactive visualizations. The study involved three visualization types, each tested under two conditions: (a) with search functionality and (b) without it. Participants were asked to analyze the visualization and document their findings. The study tracked their interactions, including total time spent on each visual element, whether they used the search function (if available), and whether their interactions involved searched items. The study found that the presence of text-based search influenced participants' information-seeking behavior. Specifically,

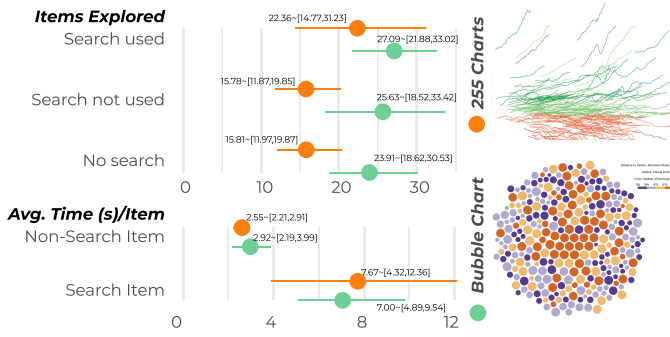


Fig. 7: Replication results from the search study align with prior findings (not shown), suggesting that while participants tend to interact with similar numbers of items, when participants engage with items highlighted during search, they tend to interact with them for longer.

search functionality encouraged users to actively seek individual data points and spend more time examining details within the data.

We chose to replicate this study because (a) it utilizes interaction logging and (b) it elicited information about participants’ process during the task, but was constrained to text-based input after the exploration task. We alternatively elicit think-aloud data from participants during the task in our replication. We follow the original study design by replicating two of the three visualizations with minor modifications—a bubble chart, and NYT’s “255 charts” [32]. The original study was implemented using vanilla JS and D3. We copied over the original code for 255 charts into reVISit using a website component, whereas we re-implemented the Bubble Chart in React. We recruited 99 and 93 participants for the Bubble and 255 charts conditions respectively after applying the original study’s minimum interaction exclusion conditions.

Results. Our experiment replicates Feng et al.’s [18] findings, in that participants in both conditions explored similar numbers of items (Fig. 7). However, the main effect of items highlighted in search being engaged with for longer also holds across both conditions. For example, in the Bubble visualization, items highlighted by search were statistically significantly viewed longer ($M(ean) = 7.00s$) than those not highlighted ($M = 2.92s$) via mouse click. Similarly, in the 255 charts visualization, search-highlighted items were also statistically significantly viewed longer ($M = 7.67s$) than non-search items ($M = 2.55s$) among search users.

Lessons Learned. ReVISit’s provenance tracking allowed us to measure where/how-long participants were interacting, but also allowed us to recreate their exploration session for deeper qualitative insights. Combined with the audio-enabled think-aloud protocol, this led to several new insights. For example, we can observe how participants “orient” themselves in the visualization: “So, this one is relatively small, the circle. And it looks like it’s an orange circle, which would appear that it’s on the higher end of earnings for people that attend this school”. It also confirmed a hypothesis from the original paper: that people would often use search to examine data that was personally relevant to them (in this case, particular industries).

5 REFLECTION

Next we reflect on the design of our system. To do so we interviewed users of reVISit and did a heuristic analysis of the system.

Interviews. We conducted a semi-structured interview study with researchers who have used reVISit in a completed study. Although reVISit has been used many times [10, 11, 42, 46], there is often an overlap between the maintainer of the system and the researchers running the study, in part because of our persistent community-centered outreach strategy. All participants were current PhD students who have run multiple user studies as part of their PhD, have run at least one study with reVISit, and are not part of the reVISit development team. We refer to participants as PX and “quote them”.

Technical Dimensions (TDPS). We conducted a close reading of reVISit via Jakubovic et al.’s “Technical Dimensions of Programming

Systems (TDPS)” [30]. TDPS is a collection of dimensions (à la Cognitive Dimensions of Notation [22]) by which to understand systems across 7 clusters of dimensions. The first author characterized how reVISit addresses each dimension, which was subsequently reviewed by two other authors. We use McNutt et al.’s [47] cluster descriptions. We elide *Conceptual* (as it is broadly covered in our discussion of the DSL) and *Complexity* (which largely overlaps with notation).

5.1 Findings

Interaction. Which loops in the system are overlapping and how far apart are the corresponding gulfs of evaluation? A common problem in developing user study prototypes is wide feedback loops during the development and testing of a study. Our study browser, as well as our development environment, is primarily aimed at tightening these feedback loops. After forking our repo, designers are met with a full development environment (via their choice of IDE, such as VSCode). P2 this found helpful: “it was also super helpful that no setup is required, [...] auto refreshes on saves and stuff: The ideal development situation that’s already there”. A common slow loop in other systems is to have to walk through the study in its entirety to debug parts of the sequence, whereas our study browser supports quick sequence navigation and therein rapid stimuli iteration.

ReVISitPy has its own unique feedback loops. When used to generate configurations, there may be a wide feedback loop between generating a configuration in a notebook, copying the generated configuration into a project, and running the project to view changes. To tighten this feedback loop, reVISitPy can preview the study within a notebook, ensuring the configuration is appropriate before leaving the notebook. However, additional exploration of desired affordances during the prototyping phase is useful future work.

An often overlooked interaction with study frameworks is with their documentation, examples, and tutorials. As part of our focus on community, we strive to have accessible documentation, along with examples covering a range of functionality (inspired by the sprawling D3 example gallery ecosystem [71]), and descriptive tutorials for more complex concepts. For instance, P2 found our data storage setup tutorial helpful, saying that they “100% had to use the guide to be able to remember what to do”; however, “I thought the guide was really good”. Similarly, P3 recalled that “I mainly learned by using the documents on the website and API documents.” No documentation is perfect, but we continue to adapt it to the needs of our user community.

A key design decision was to task study designers with using a DSL to specify experiments, in contrast to a GUI (as Qualtrics does), which likely would have radically tightened the design loop. Although there is substantially higher technical complexity in using a DSL, it makes many more designs possible than what we might have designed for in building a GUI. Echoing Alan Kay: “Simple things should be simple, complex things should be possible” [4]. While future work might explore a GUI, we suggest that centering the possible (and valuing the technical proficiencies of our design-for audience) is essential.

Notation. What notations are present and how do they interrelate? ReVISit’s primary notation is the JSON DSL (which every study interacts with), complemented by several optional ones. Our DSL is designed to be simple and highly composable, with the only syntactic abstraction being `baseComponents`, as indicated in Fig. 2.

Such simplicity can be misaligned with the expected simplicity of some experiment designs. As discussed in Sec. 3.1, config files can be very large due to repetition of structures and components for a variety of factors. P2, who needed to make a 12k LOC file, questioned “I don’t know if what I’m doing is just a really rare experimental design. It doesn’t sound it from saying it in English, but then I think it kind of ends up being fairly orthogonal to the way the sequence is set up”. ReVISitPy attempts to address this problem by offering a greater degree of abstraction than is present in the DSL. The choice of transferring complexity away from the primary notation (the DSL) and into a secondary notation (reVISitPy/Python) was intended to keep the DSL simple, as most studies would not need the complex features

that reVISitPy enables, while also empowering users who do need such functionalities. We forewent implementation of common abstractions (such as loops and variables) to avoid maintaining a full programming language. Instead, we ensure that our DSL is highly composable, allowing for complex designs via reVISitPy or other custom solutions, with the tradeoff of increased difficulty in generating the configuration.

Apart from the DSL, studies may utilize secondary notations for construction of stimuli—such as markdown or React, as well as any iframe embeddable content. Many stimuli interact with the DSL to access parameters, answers, or provenance data. To create custom stimuli, P1 used multiple notations, some of which they had no prior experience with. They pointed out that this increased the learning curve: “You’re also learning React and D3 and JavaScript and a little bit of Firebase”. The many possible notations that are usable in conjunction with reVISit widen the range of possible stimuli, but also result in a steeper learning curve and inherited complexity of such notations.

Errors. *What are they and how are they handled?* Errors within reVISit are handled differently depending on the source of the error. Errors within our primary notation, the DSL, make rendering and compiling the study impossible, increasing the importance that study designers can quickly identify and fix these errors. Warnings about DSL errors can originate from two sources. The first is from automated validation of our JSON schema. The second is application-specific errors which the system throws, such as trying to use a *baseComponent* which does not exist. In both cases, errors are immediately surfaced and shown as errors within the system with an error message. However, some errors may be difficult to identify via this message, especially errors that originated from schema validation. P3 noted, “it would be better if that was a little bit more straightforward about exactly what part is broken”. However, P3 also pointed out the increased support that code editors can give to identify such errors, saying, “I use VS Code for opening the JSON. And if there are grammar mistakes, then VS Code itself reveals which part is wrong. That was also helpful when the file was broken.” reVISit surfaces lint-like warnings for stylistic usage problems—for example, components that are defined but not used yield a warning. However, our linting is limited to programmatic problems in the DSL, and does not consider the semantics of an experimental design (e.g., confounders). Although reVISit puts a number of experimental designs within reach, it does little to help ensure that the experimental designs themselves are good outside of these practices. For instance, an experiment may unintentionally introduce learning effects by always presenting a fixed order of chart stimuli that escalate in complexity. In future work, it would be useful to explore automated experiment evaluation and the effects it would have on experimental design.

Errors that occur outside of the DSL can be more difficult to identify. reVISit surfaces errors thrown by React components to avoid complete application crashes, but does not have such functionality for iframes. Errors can be surfaced in the development environment itself, as with an error P1 ran into: “when I tried to import this [math library] into reVISit, it was telling me about some random number generating library that is outdated that revisit does not accept.” Errors such as this one, which often stem from library version conflicts, can be challenging to fix even for users with a background in web development. We could avoid such errors by dictating the development environment (e.g., by allowing only some packages or using a GUI), however this is in tension with our experimental sovereignty goal: some things may be harder, but more things will be possible.

Customizability. *How can programs be modified?* Easy replication and modification of studies is central to reVISit’s focus on improving reproducibility in user studies. Our core GitHub repository comes with a collection of example studies, such as a replication of Cleveland-McGill [8], in a manner that supports opportunistic programming-style reuse [6]. Both the experiment design (via the DSL) and the stimuli are available and modifiable. The same is true for public reVISit studies, such as those run by Lisnic et al [42], which can be reproduced or modified easily. Studies maintain the version of reVISit when forked (unless manually merged), ensuring that the study continues to operate

properly in spite of continued development on reVISit.

Additionally, as study designers fork the entire codebase, core functionality of the tool is visible and changeable. P1 used the open nature of the core code as a learning tool, observing “that’s the benefit of having all the code base up there. If there’s something that I don’t understand, I can always just try to find its source”. P2 and P3 both took this a step further, editing the core code to match certain desired behavior for their studies, with P2 saying “if I need to change something in its guts, I can do that, which is fun, as opposed to something more closed”. Lisnic et al. [42], for example, modified the core code to host a “[demonstration page](#)” for their project, including the study, its results, and a sandboxed version of their stimulus. Viewing or modifying core code was not originally intended behavior, and cannot be expected from study designers without a strong technical background. However, for those with the skills required, the core code can prove a valuable tool to accompany documentation and allow for a high level of customization.

Adaptability. *What socio-technical (e.g., learnability) dimensions are considered?* reVISit is primarily directed at the visualization community, but has been designed with the intention of being usable for a wide range of studies, including from fields with less uniform technical backgrounds (e.g., psychology). However, from our interviews (and experiences), reVISit is currently most useful for those with a programming background (inheriting learning curves from React or TypeScript). Taking advantage of many of the unique and valuable features within reVISit, such as application provenance tracking or reVISitPy, requires programming knowledge. Our choice to avoid GUI approaches (for the time being) is partially to blame for this learning curve. P3 pointed out that “there are some people that do not have the computational knowledge of web designing. It’s really difficult for me to suggest [to them that] they use revisit, because [...] you need to tune actual code, compared to Qualtrics or those web based survey systems”.

One approach we have taken to mitigate the learning curve is via community outreach, which has been a focus since the first stable release. We have held tutorials on reVISit at VIS’24, CHI’25, EuroVis’25, Georgia Tech, University of Utah, UNC Chapel Hill, and are scheduled to hold a tutorial at VIS’25. We also have an active Slack team with public channels for users to get help and make suggestions.

6 CONCLUSION

reVISit 2 seeks to simplify the process of designing, debugging, deploying, and disseminating experiments in visualization research and related fields. Through a four year cycle of community engaged work we have developed a platform that we believe drastically simplifies the experimental process, especially for the kind of sophisticated studies that are increasingly commonplace in visualization and HCI. As part of this work, we developed a range of novel contributions to experiment infrastructure, including a sophisticated DSL and a Python-based extension that can accomplish more diverse study designs than comparable frameworks, notebook-based prototypes, provenance tracking and associated participant replay, and advanced stimuli, such as Vega and React programs. To demonstrate these contributions (and the robustness of reVISit), we conducted a trio of replication studies, which we used (in tandem with a small interview study) to reflect on our design choices.

This work is not the end of reVISit’s story. There are various additional improvements to the experiment life cycle to explore, new communities to support, and limitations to address. For instance, reVISit is primarily designed for visualization research, and is focused by the particular research goals of the research team—although we made efforts to be expansive in our designs and be informed by a variety of stakeholders from around the visualization community. However, we do not address concerns held in related domains, such as games [57] or social robots [66]. Similarly, reVISit is currently limited to desktop-based web browsers, precluding studies in mediums like mobile, watchfaces, or AR/VR. While many such studies can be replicated in web browsers, native support would be preferable.

Through this work, and our continued work on reVISit, we hope to raise study quality and reproducibility for the visualization community and make it easier to investigate rich empirical phenomena.

ACKNOWLEDGMENTS

We would like to extend our thanks to the National Science Foundation (2213756, 2213757, 2402719, 2313998), to Carolina Nobre, to our users for their insightful questions and contributions, to our numerous study/interview participants, and to our community advisory board, which consists of Danielle Albers Szafr, Cindy Xiong Bearfield, Ana Crisan, Alex Endert, Jean-Daniel Fekete, Petra Isenberg, Lace Padilla, John Stasko, and Manuela Waldner.

SUPPLEMENTAL MATERIALS

The code for reVISit can be found at github.com/revisit-studies/study. All documentation, examples, and tutorials can be found at revisit.dev. A demo with example studies is at revisit.dev/study. The study replication code is at github.com/revisit-studies/replication-studies, and the full replication studies and their data are available at revisit.dev/replication-studies. We also provide an appendix with additional analysis for all replications, and a separate repository containing the data and the analysis code for the replications at github.com/revisit-studies/replication-studies-analysis. Pre registrations for each study can be found at <https://osf.io/e8anx/>.

REFERENCES

- [1] R. Abdalkareem, V. Oda, S. Mujahid, and E. Shihab. On the impact of using trivial packages: An empirical case study on npm and PyPI. *Empirical Software Engineering*, 25(2):1168–1204, Mar. 2020. doi: 10.1007/s10664-019-09792-9 6
- [2] W. Aigner, S. Hoffmann, and A. Rind. EvalBench: A Software Library for Visualization Evaluation. *Computer Graphics Forum*, 32(3pt1):41–50, 2013. doi: 10.1111/cgf.12091 3
- [3] D. Akbaba, D. Lange, M. Correll, A. Lex, and M. Meyer. Troubling Collaboration: Matters of Care for Visualization Design. In *ACM CHI Conference on Human Factors in Computing Systems*, 2023. doi: 10.1145/3544548.3581168 4
- [4] Alan Kay. Alan Kay’s adage “Simple things should be simple, complex things should be possible”. 8
- [5] J. Boy, L. Eveillard, F. Detienne, and J.-D. Fekete. Suggested Interactivity: Seeking Perceived Affordances for Information Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):639–648, Jan. 2016. doi: 10.1109/TVCG.2015.2467201 1
- [6] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer. Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1589–1598. ACM, Boston MA USA, Apr. 2009. doi: 10.1145/1518701.1518944 9
- [7] K.-T. Chen, A. Prouzeau, J. Langmead, R. T. Whitelock-Jones, L. Lawrence, T. Dwyer, C. Hurter, D. Weiskopf, and S. Goodwin. Gaze-alytics: A Unified and Flexible Visual Toolkit for Exploratory and Comparative Gaze Analysis. In *2023 Symposium on Eye Tracking Research and Applications*, pp. 1–7, May 2023. doi: 10.1145/3588015.3589844 4
- [8] W. S. Cleveland and R. McGill. Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984. doi: 10.1080/01621459.1984.10478080 1, 9
- [9] J. Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Routledge, May 2013. 3
- [10] Y. Cui, L. W. Ge, Y. Ding, L. Harrison, F. Yang, and M. Kay. Promises and Pitfalls: Using Large Language Models to Generate Visualization Items. *IEEE Transactions on Visualization and Computer Graphics*, 31(1):1094–1104, Jan. 2025. doi: 10.1109/TVCG.2024.3456309 2, 8
- [11] Z. Cutler, L. Harrison, C. Nobre, and A. Lex. Crowdsourced Think-Aloud Studies. In *ACM Conference on Human Factors in Computing Systems (CHI)*, 2025. doi: 10.1145/3706598.3714305 1, 2, 4, 6, 8
- [12] Z. T. Cutler, K. Gadhav, and A. Lex. Ttrack: A Library for Provenance Tracking in Web-Based Visualizations. In *IEEE Visualization Conference (VIS)*, pp. 116–120, 2020. doi: 10.1109/VIS47514.2020.00030 5, 7, 13
- [13] J. R. de Leeuw, R. A. Gilbert, and B. Luchterhandt. jsPsych: Enabling an Open-Source Collaborative Ecosystem of Behavioral Experiments. *Journal of Open Source Software*, 8(85):5351, May 2023. doi: 10.21105/joss.05351 2, 3
- [14] Y. Ding, J. Wilburn, H. Shrestha, A. Ndlovu, K. Gadhav, C. Nobre, A. Lex, and L. Harrison. reVISit: Supporting Scalable Evaluation of Interactive Visualizations. In *IEEE Visualization and Visual Analytics (VIS)*, pp. 31–35. IEEE, 2023. doi: 10.1109/VIS54172.2023.00015 2, 3, 4
- [15] W. Dou, C. Ziemkiewicz, L. Harrison, D. H. Jeong, R. Ryan, W. Ribarsky, X. Wang, and R. Chang. Comparing different levels of interaction constraints for deriving visual problem isomorphs. In *2010 IEEE Symposium on Visual Analytics Science and Technology*, pp. 195–202, Oct. 2010. doi: 10.1109/VAST.2010.5653599 1
- [16] A. Eismayer, C. Wacharamanatham, M. Beaudouin-Lafon, and W. E. Mackay. Touchstone2: An Interactive Environment for Exploring Trade-offs in HCI Experiment Design. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–11. ACM, Glasgow Scotland Uk, May 2019. doi: 10.1145/3290605.3300447 3
- [17] M. Fan, K. Wu, J. Zhao, Y. Li, W. Wei, and K. N. Truong. VisTA: Integrating Machine Intelligence with Visualization to Support the Investigation of Think-Aloud Sessions. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):343–352, 2020. doi: 10.1109/TVCG.2019.2934797 4
- [18] M. Feng, C. Deng, E. M. Peck, and L. Harrison. The Effects of Adding Search Functionality to Interactive Visualizations on the Web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pp. 1–13. ACM, 2018. doi: 10.1145/3173574.3173711 6, 7, 8, 13
- [19] D. Gergle and D. S. Tan. Experimental Research in HCI. In J. S. Olson and W. A. Kellogg, eds., *Ways of Knowing in HCI*, pp. 191–227. Springer, New York, NY, 2014. doi: 10.1007/978-1-4939-0378-8_9 1, 3, 5
- [20] M. Ghoniemi, J.-D. Fekete, and P. Castagliola. On the Readability of Graphs Using Node-Link and Matrix-Based Representations: A Controlled Experiment and Statistical Analysis. *Information Visualization*, 4(2):114–135, 2005. doi: 10.1057/palgrave.ivs.9500092 1
- [21] Google. Google Forms, Mar. 2025. 2, 3
- [22] T. R. Green. Cognitive dimensions of notations. *People and computers V*, pp. 443–460, 1989. doi: 10.1007/3-540-44617-6_31 8
- [23] L. Harrison, C. Gramazio, F. Yang, K. Aragam, E. Peck, and D. Schroeder. Experimentr, Feb. 2019. 2, 3, 7
- [24] L. Harrison, F. Yang, S. Franconeri, and R. Chang. Ranking Visualizations of Correlation Using Weber’s Law. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1943–1952, Dec. 2014. doi: 10.1109/TVCG.2014.2346979 3, 6, 12
- [25] T. He, P. Isenberg, R. Dachsel, and T. Isenberg. BeauVis: A Validated Scale for Measuring the Aesthetic Pleasure of Visual Representations. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):363–373, Jan. 2023. doi: 10.1109/TVCG.2022.3209390 2, 7, 13
- [26] T. He, Y. Zhong, P. Isenberg, and T. Isenberg. Design Characterization for Black-and-White Textures in Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):1019–1029, Jan. 2024. doi: 10.1109/TVCG.2023.3326941 1, 6, 7, 13
- [27] J. Heer and M. Bostock. Crowdsourcing graphical perception: Using mechanical turk to assess visualization design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pp. 203–212. ACM, 2010. doi: 10.1145/1753326.1753357 1
- [28] A. Inc. Amazon Mechanical Turk, 2025. 3
- [29] Q. I. Inc. Qualtrics, 2025. 1, 2, 3, 5
- [30] J. Jakubovic, J. Edwards, and T. Petricek. Technical Dimensions of Programming Systems. *The Art, Science, and Engineering of Programming*, 7(3):13, 2023. doi: 10.22152/programming-journal.org/2023/7/13 2, 8
- [31] Y. Jansen. FROE: Framework for Running Online Experiments, Mar. 2025. 2, 3, 13
- [32] Jeremy Ashkenas and Alicia Parlapiano. How the Recession Reshaped the Economy, in 255 Charts - The New York Times, June 2014. 8
- [33] R. Jianu, D. Laksono, A. Slingsby, and M. Okoe. VisUnit: Literate Visualisation Studies Assembled from Reusable Test-Suites. In *ACM CHI Conference on Human Factors in Computing Systems*, 2025. doi: 10.1145/3706598.3713104 2, 3, 5
- [34] A. Joshi, S. Kale, S. Chandel, and D. Pal. Likert Scale: Explored and Explained. *British Journal of Applied Science & Technology*, 7(4):396–403, Jan. 2015. doi: 10.9734/BJAST/2015/14975 2
- [35] A. Kale, F. Nguyen, M. Kay, and J. Hullman. Hypothetical outcome plots help untrained observers judge trends in ambiguous data. *IEEE transactions on visualization and computer graphics*, 25(1):892–902, 2018. doi: 10.1109/TVCG.2018.2864909 1
- [36] M. Kay and J. Heer. Beyond Weber’s Law: A Second Look at Ranking Visualizations of Correlation. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):469–478, Jan. 2016. doi: 10.1109/TVCG.2015.2467671 12
- [37] S. Kieffer, T. Dwyer, K. Marriott, and M. Wybrow. HOLA: Human-like

- Orthogonal Network Layout. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):349–358, Jan. 2016. doi: [10.1109/TVCG.2015.2467451](https://doi.org/10.1109/TVCG.2015.2467451) 1
- [38] Y. Kim, M. Correll, and J. Heer. Designing Animated Transitions to Convey Aggregate Operations. *Computer Graphics Forum*, 38(3):541–551, June 2019. doi: [10.1111/cgf.13709](https://doi.org/10.1111/cgf.13709) 1
- [39] M. S. Lam, J. Teoh, J. A. Landay, J. Heer, and M. S. Bernstein. Concept Induction: Analyzing Unstructured Text with High-Level Concepts Using LLOoM. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, pp. 1–28. Association for Computing Machinery, New York, NY, USA, May 2024. doi: [10.1145/3613904.3642830](https://doi.org/10.1145/3613904.3642830) 14
- [40] S. Lee, S.-H. Kim, Y.-H. Hung, H. Lam, Y.-A. Kang, and J. S. Yi. How do People Make Sense of Unfamiliar Visualizations?: A Grounded Model of Novice's Information Visualization Sensemaking. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):499–508, Jan. 2016. doi: [10.1109/TVCG.2015.2467195](https://doi.org/10.1109/TVCG.2015.2467195) 1
- [41] S. Lee, S.-H. Kim, and B. C. Kwon. VLAT: Development of a Visualization Literacy Assessment Test. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):551–560, 2017. doi: [10.1109/TVCG.2016.2598920](https://doi.org/10.1109/TVCG.2016.2598920) 2, 5
- [42] M. Lisnic, Z. Cutler, M. Kogan, and A. Lex. Visualization guardrails: Designing interventions against cherry-picking in interactive data explorers. In *SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2025. doi: [10.1145/3706598.3713385](https://doi.org/10.1145/3706598.3713385) 1, 2, 8, 9
- [43] M. J. Lobo, C. Hurter, and P. Irani. Flex-ER: A Platform to Evaluate Interaction Techniques for Immersive Visualizations. *Proc. ACM Hum.-Comput. Interact.*, 4(ISS):195:1–195:20, Nov. 2020. doi: [10.1145/3427323](https://doi.org/10.1145/3427323) 2, 3
- [44] S. L'Yi, A. van den Brandt, E. Adams, H. N. Nguyen, and N. Gehlenborg. Learnable and Expressive Visualization Authoring through Blended Interfaces. *IEEE Transactions on Visualization and Computer Graphics*, 2024. doi: [10.1109/TVCG.2024.3456598](https://doi.org/10.1109/TVCG.2024.3456598) 1
- [45] W. E. Mackay, C. Appert, M. Beaudouin-Lafon, O. Chapuis, Y. Du, J.-D. Fekete, and Y. Guiard. Touchstone: Exploratory design of experiments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pp. 1425–1434. Association for Computing Machinery, New York, NY, USA, Apr. 2007. doi: [10.1145/1240624.1240840](https://doi.org/10.1145/1240624.1240840) 2, 3
- [46] A. McNutt, M. K. McCracken, I. J. Eliza, D. Hajas, J. Wagoner, N. Lanza, J. Wilburn, S. Creem-Regehr, and A. Lex. Accessible Text Descriptions for UpSet Plots. *Computer Graphics Forum*, 44(3):e70102, 2025. doi: [10.1111/cgf.70102](https://doi.org/10.1111/cgf.70102) 2, 8
- [47] A. McNutt, M. C. Stone, and J. Heer. Mixing Linters with GUIs: A Color Palette Design Probe. *IEEE Transactions on Visualization and Computer Graphics*, 31(1):327–337, 2025. doi: [10.1109/TVCG.2024.3456317](https://doi.org/10.1109/TVCG.2024.3456317) 8
- [48] A. M. McNutt. No Grammar to Rule Them All: A Survey of JSON-style DSLs for Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):160–170, 2023. doi: [10.1109/TVCG.2022.3209460](https://doi.org/10.1109/TVCG.2022.3209460) 4
- [49] M. Meuschke, N. N. Smit, M. Lichtenberg, B. Preim, and K. Lawonn. EvalViz – Surface visualization evaluation wizard for depth and shape perception tasks. *Computers & Graphics*, 82:250–263, Aug. 2019. doi: [10.1016/j.cag.2019.05.022](https://doi.org/10.1016/j.cag.2019.05.022) 2, 3
- [50] S. Musslick, A. Cherkov, B. Draut, A. S. Butt, P. Darragh, V. Srikumar, M. Flatt, and J. D. Cohen. SweetPea: A standard language for factorial experimental design. *Behavior Research Methods*, 54(2):805–829, Apr. 2022. doi: [10.3758/s13428-021-01598-2](https://doi.org/10.3758/s13428-021-01598-2) 1, 3, 5
- [51] C. Nobre, D. Wootton, Z. Cutler, L. Harrison, H. Pfister, and A. Lex. reVISit: Looking Under the Hood of Interactive Visualization Studies. In *SIGCHI Conference on Human Factors in Computing Systems*, pp. 1–13. ACM, Yokohama Japan, May 2021. doi: [10.1145/3411764.3445382](https://doi.org/10.1145/3411764.3445382) 4
- [52] C. Nobre, D. Wootton, L. Harrison, and A. Lex. Evaluating Multivariate Network Visualization Techniques Using a Validated Design and Crowdsourcing Approach. In *SIGCHI Conference on Human Factors in Computing Systems*, pp. 1–12. ACM, 2020. doi: [10.1145/3313831.3376381](https://doi.org/10.1145/3313831.3376381) 1
- [53] S. Nowak and L. Bartram. Designing for Ambiguity in Visual Analytics: Lessons from Risk Assessment and Prediction. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):924–933, Jan. 2024. doi: [10.1109/TVCG.2023.3326571](https://doi.org/10.1109/TVCG.2023.3326571) 1
- [54] M. Okoe and R. Jianu. GraphUnit: Evaluating Interactive Graph Visualizations Using Crowdsourcing. *Computer Graphics Forum*, 34(3):451–460, 2015. doi: [10.1111/cgf.12657](https://doi.org/10.1111/cgf.12657) 1, 2, 3
- [55] S. Palan and C. Schitter. Prolific.ac A subject pool for online experiments. *Journal of Behavioral and Experimental Finance*, 17:22–27, Mar. 2018. doi: [10.1016/j.jbef.2017.12.004](https://doi.org/10.1016/j.jbef.2017.12.004) 3
- [56] S. Pandey and A. Ottley. Mini-VLAT: A Short and Effective Measure of Visualization Literacy. *Computer Graphics Forum*, 42(3):1–11, 2023. doi: [10.1111/cgf.14809](https://doi.org/10.1111/cgf.14809) 2, 5
- [57] N. Partlan, E. Carstensdottir, S. Snodgrass, E. Kleinman, G. Smith, C. Hartevelde, and M. S. El-Nasr. Exploratory automated analysis of structural features of interactive narrative. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 14, pp. 88–94, 2018. doi: [10.1609/aiide.v14i1.13019](https://doi.org/10.1609/aiide.v14i1.13019) 9
- [58] J. Peirce, J. R. Gray, S. Simpson, M. MacAskill, R. Höchenberger, H. Sogo, E. Kastman, and J. K. Lindeløv. PsychoPy2: Experiments in behavior made easy. *Behavior Research Methods*, 51(1):195–203, Feb. 2019. doi: [10.3758/s13428-018-01193-y](https://doi.org/10.3758/s13428-018-01193-y) 3
- [59] K. Reinecke and K. Z. Gajos. LabintheWild: Conducting Large-Scale Online Experiments With Uncompensated Samples. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW, pp. 1364–1378. Association for Computing Machinery, New York, 2015. doi: [10.1145/2675133.2675246](https://doi.org/10.1145/2675133.2675246) 3
- [60] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, Jan. 2017. doi: [10.1109/TVCG.2016.2599030](https://doi.org/10.1109/TVCG.2016.2599030) 5
- [61] E. J. Soure, E. Kuang, M. Fan, and J. Zhao. CoUX: Collaborative Visual Analysis of Think-Aloud Usability Test Videos for Digital Interfaces. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):643–653, Jan. 2022. doi: [10.1109/TVCG.2021.3114822](https://doi.org/10.1109/TVCG.2021.3114822) 4
- [62] G. Stoet. PsyToolkit: A Novel Web-Based Method for Running Online Questionnaires and Reaction-Time Experiments. *Teaching of Psychology*, 44(1):24–31, Jan. 2017. doi: [10.1177/0098628316677643](https://doi.org/10.1177/0098628316677643) 2, 3
- [63] P. T. Sukumar and R. Metoyer. Towards Designing Unbiased Replication Studies in Information Visualization. In *2018 IEEE Evaluation and Beyond - Methodological Approaches for Visualization (BELIV)*, pp. 93–101, 2018. doi: [10.1109/BELIV.2018.8634261](https://doi.org/10.1109/BELIV.2018.8634261) 1
- [64] SurveyMonkey. SurveyMonkey, Mar. 2025. 2, 3
- [65] D. A. Szafir. Modeling Color Difference for Visualization Design. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):392–401, Jan. 2018. doi: [10.1109/TVCG.2017.2744359](https://doi.org/10.1109/TVCG.2017.2744359) 1
- [66] N. Tsoi, M. Hussein, O. Fugikawa, J. D. Zhao, and M. Vázquez. An approach to deploy interactive robotic simulators on the web for hri experiments: Results in social robot navigation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7528–7535. IEEE, 2021. doi: [10.1109/IROS51168.2021.9636319](https://doi.org/10.1109/IROS51168.2021.9636319) 9
- [67] T. L. Turton, A. S. Berres, D. H. Rogers, and J. Ahrens. ETK: An evaluation toolkit for visualization user studies. In *Proceedings of the Eurographics/IEEE VGTC Conference on Visualization: Short Papers*, EuroVis '17, pp. 43–47. Eurographics Association, Goslar, DEU, June 2017. doi: [10.2312/eurovisshort.20171131](https://doi.org/10.2312/eurovisshort.20171131) 2, 3
- [68] J. VanderPlas, B. E. Granger, J. Heer, D. Moritz, K. Wongsuphasawat, A. Satyanarayan, E. Lees, I. Timofeev, B. Welsh, and S. Sievert. Altair: Interactive Statistical Visualizations for Python. *Journal of Open Source Software*, 3(32):1057, Dec. 2018. doi: [10.21105/joss.01057](https://doi.org/10.21105/joss.01057) 2, 4
- [69] VERBI Software. MAXQDA Analytics Pro, 2018. 4
- [70] F. Yang, L. T. Harrison, R. A. Rensink, S. L. Franconeri, and R. Chang. Correlation Judgment and Visualization Features: A Comparative Study. *IEEE Transactions on Visualization and Computer Graphics*, 25(3):1474–1488, Mar. 2019. doi: [10.1109/TVCG.2018.2810918](https://doi.org/10.1109/TVCG.2018.2810918) 3
- [71] J. Yang, A. M. McNutt, and L. Battle. Considering Visualization Example Galleries. In *2024 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 329–343, Sept. 2024. doi: [10.1109/VL/HCC60511.2024.00043](https://doi.org/10.1109/VL/HCC60511.2024.00043) 8
- [72] J. Zacks and B. Tversky. Bars and lines: A study of graphic communication. *Memory & Cognition*, 27(6):1073–1079, Nov. 1999. doi: [10.3758/BF03201236](https://doi.org/10.3758/BF03201236) 1
- [73] Y. Zhao, Y. Wang, X. Luo, Y. Wang, and J.-D. Fekete. Libra: An Interaction Model for Data Visualization. In *SIGCHI Conference on Human Factors in Computing Systems*, 2025. doi: [10.1145/3706598.3713769](https://doi.org/10.1145/3706598.3713769) 7