TECHNICAL REPORT

DESCRIPTION AND VALIDATION OF A NOVEL HUMAN-OPERANT RESEARCH SOFTWARE

Braden Toler[1], David Legaspi[2], Daniel Mitteer[3, 4]

[1] THE STERN CENTER FOR DEVELOPMENTAL AND BEHAVIORAL HEALTH
[2] UTAH STATE UNIVERSITY
[3] SEVERE BEHAVIOR PROGRAM, CHILDREN'S SPECIALIZED HOSPITAL—RUTGERS UNIVERSITY CENTER FOR AUTISM RESEARCH, EDUCATION, AND SERVICES (CSH-RUCARES)
[4] RUTGERS ROBERT WOOD JOHNSON MEDICAL SCHOOL

In recent years, computer models have become an indispensable tool for conducting basic human-operant research. However, one's ability to conclude meaningful relations between independent and dependent variables relies on the assumption that the computer software is reliable and operates as intended. This technical report delves into the features and validation process of a novel software entitled *VirtuOperant*; built using the Python programming language to support human-operant research. Our software validation procedure draws inspiration from the methodology developed by Smith and Greer (2022). This structured approach not only affirms the software's efficacy but also emphasizes its potential for replicability across varied research scenarios (e.g., simulations of discrimination learning, treatment relapse). The focus of this report revolves around the validation of the software and underscores the software's potential for future human-operant research. We found *VirtuOperant* to be reliable at programming various reinforcement schedules and phenomena. Given its reliability and ability to generate data automatically, we anticipate this software becoming a useful tool for future research.

Keywords: apparatus; validation; human-operant; translational; Python

The operant conditioning chamber has long served as a cornerstone for understanding behavioral mechanisms in controlled settings. Historically, this domain has been dominated by animal studies, but the shift towards human-operant research introduced a new layer of complexity and relevance (Lattal & Perone, 1998; Perone, 1985). The evolution of technology has significantly impacted this field, with computer models emerging as crucial tools to emulate and study human-operant environments (Smith & Greer, 2022). For example, research using human-operant paradigms have provided opportunities to study variables related to treatment relapse without posing risks to applied populations by increasing behavior of significance (Bolívar et al., 2017; Kestner & Peterson, 2017; Kestner et al., 2018; Ritchey et al., 2021).

Despite the advances in technology, the challenge remains to develop software that is both accurate in its representation of operant principles and versatile enough to accommodate the varied nuances of human-operant research. Saini and Mitteer (2020) noted that studies with human participants often require a more meticulous accounting of variables. Particularly, when many studies span across several days, there are potential confounds introduced by gaps between experimental sessions, which might obscure genuine instances of behavior change. When our scientific interpretation of the relations between independent and dependent variables relies on minimization of threats to internal validity (e.g., instrumentation), it is imperative to demonstrate the reliability of novel software prior to conducting research with it or interpreting results derived from the software.

The software described in this report was built using the Python programming language. It allows experimenters to (a) customize up to four response buttons, (b) program a dissimilar control activity (i.e., drawing on a portion of the

Author Note: Any correspondence or requests for access to the *VirtuOperant* software should be addressed to Braden Toler. Email: bjt0021@mix.wvu.edu.

Additional Author Information: Daniel R. Mitteer: ORCID: 000-001-8940-0657

https://doi.org/10.17605/OSF.IO/8MJU4

screen), (c) manipulate response-based or interval-based schedules of reinforcement (i.e., point gain) and punishment (i.e., point loss), and (d) graph data automatically following the experiment. While there is no explicit limit to duration, experimental phases can be programmed to be short (e.g., 120 s) and allow participants to navigate all phases of the experiment in a single iteration, thereby making data obtainment practical and reducing the inconsistencies that might arise from prolonged gaps. Therefore, the purpose of this paper is to introduce a novel software solution specifically designed to overcome the limitations with current technologies. Additionally, we provide validation for this software, employing a methodology adapted from Smith & Greer (2022). We hope this will provide researchers with a reliable and efficient tool that advances the frontiers of human-operant research, ensuring accuracy and consistency in data collection.

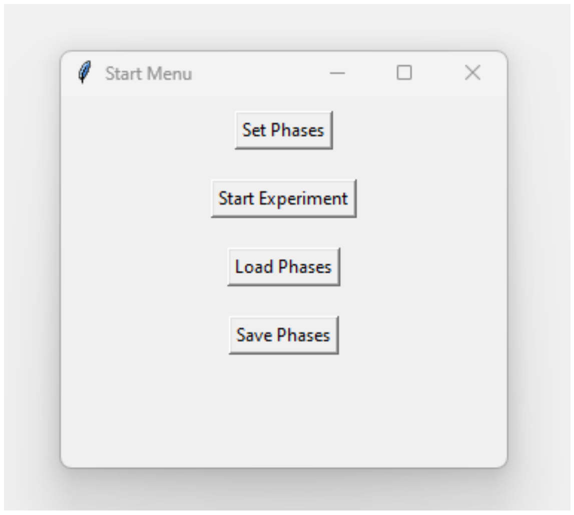


Figure 1. Setup Screen. *Note:* Set phases allows the experimenter to set up the experimental procedure; Save Phases allows the experimenter to save an experimental arrangement for future use; Load Phases allows the experimenter to load a previously existing experiment.

PROGRAM SPECIFICATIONS

Overview

The operant conditioning chamber's highly controlled environment is an integral part of basic animal research in behavior analysis. The Python program serves as a digital emulation of the environmental conditions utilized by basic researchers; creating an environment in which users can accrue points through the act of engaging in an arbitrary button pressing task as is common for human-operant research (Kestner et al., 2018; Okouchi, 2015). This program provides an interactive user interface (UI) that enables users (participants of the experiment) to interact with the experimental setup. The design and flow of the experiment are based on a series of preset experimental phases, each with its own set of reinforcement schedules. The primary functions of the program include: (a) setting up the user interface, (b) tracking and storing user interactions, (c) applying reinforcement schedules, (d) managing the different phases of the experiment, and finally, (e) analyzing and visualizing the data collected during the experiment.

Setup and Initialization

The setup and initialization process of the program involves several key steps, which prepare the application for user interaction and establish the initial state of the experiment. When the program starts, it sets up the UI. This involves creating a main application window that acts as a container for all other UI elements (Figure 1). The window is configured to be displayed in full screen mode, providing an immersive experience for the user and helping to eliminate extraneous variables. The program incorporates a drawing canvas at the bottom of the screen, outlined and accompanied by instructive text. This canvas serves as a deliberate design feature, granting users an option to disengage from the primary experiment. Drawing from insights by Bolivar et al. (2017) and Sweeney and Shahan (2016), it is crucial to provide participants with alternatives to active engagement, minimizing any coerced responses or persistent behavior in the absence of alternatives (i.e., clicking response buttons during extinction due to no other potential activities).

Above this drawing canvas, an image canvas is created to display images associated with each phase of the experiment (Figure 2b). Initialization of the experiment's parameters is the next step. Parameters such as the list of images to be displayed as stimuli, the count of button presses, the remaining time, and the reinforcement timestamps are initialized. These parameters play vital roles in the progression and operation of the experiment. The program also sets up lists to track button press timestamps and the last reinforcement timestamps for each button. Furthermore, it initializes counters for different reinforcement schedules. A list to store the different phases of the experiment is initialized, and the experiment begins with the first phase. These phases determine the duration and reinforcement schedule(s) of the experiment.

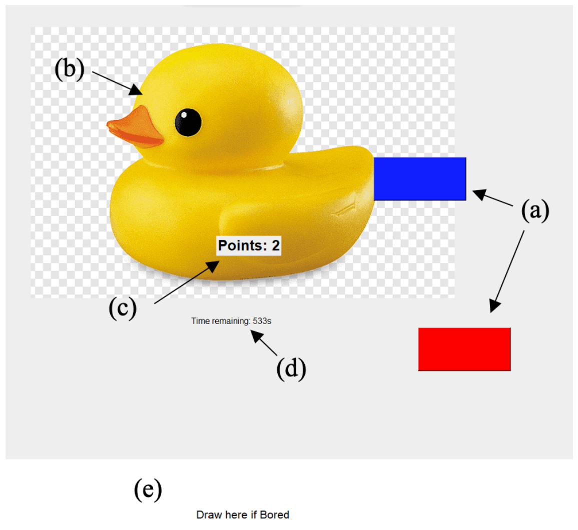


Figure 2. Experiment Screen. *Note:* The image depicts the various features of the experimental window: response buttons (a), visual stimulus (b), point counter (c), time counter (d), and drawing canvas (e). The program allows for up to four response buttons to be active at a time.

Experiment Screen

In the experimental portion of the program, the users interact with moving rectangular buttons, which are in constant motion across the screen (Figure 2a). These buttons move to a random location on the screen every 3 seconds. This creates an engaging and dynamic environment where the users need to track and press the moving buttons. This provides a consistent challenge for the user to locate and press the buttons. When a user presses a button, the program registers the event and applies the appropriate reinforcement rules depending on the current phase of the experiment. If reinforcement is available, pressing the button will produce a brief green screen flash and bell sound as the score counter increases (Figure 2c).

The program also includes aversive feedback to implement punishment procedures. If the user's button press triggers an aversive event according to the assigned schedule, the user's score is decreased. Aversive feedback is also accompanied by a brief red flash on the screen and the sound of a buzzer.

In the upper portion of the screen, the program allows the experimenter to place image files that can be varied by experimental phase. This provides experimenters with a means of creating variations in the experimental environment for the purposes of testing concepts such as renewal or discrimination (Saini & Mitteer, 2020; Bernal-Gamboa et al., 2020).

Finally, at the bottom of the screen is where the drawing canvas is placed so as to provide an alternative option to interacting with the experimental procedure (Figure 2e).

Reinforcement Schedules

In this program, the reinforcement schedules are implemented in a function called “apply_reinforcement_schedule”. This function is called each time a button is pressed, and it applies the corresponding reinforcement schedule based on the current phase of the experiment. For the fixed-ratio (FR) schedule, the function checks if the count of button presses is a multiple of the specified ratio. If it is, reinforcement is delivered, which in this context means increasing the user's points and providing positive feedback. For fixed-interval (FI) schedules, the function calculates the time elapsed since the last reinforcement. If this time is greater than or equal to the specified interval, reinforcement is delivered. The timestamp for the last reinforcement is then updated to the current global time. Variable schedules of reinforcement proved to be more complex to program. For variable-ratio (VR) schedules, the function generates a random number and checks if it is less than the reciprocal of the specified average. If it is, reinforcement is delivered. This method effectively creates a probability of reinforcement around the specified average. The variable-interval (VI) schedule functions similarly to the fixed-interval schedule, the function calculates the time elapsed since the last reinforcement. If this time is greater than or equal to a randomly generated delay (ranging from 50% to 150% of the specified average), reinforcement is delivered, and the delay is updated for the next interval. Noncontingent reinforcement (NCR) is implemented slightly differently from the others. It is controlled by a function that is called every second and checks if the time elapsed since the last reinforcement for

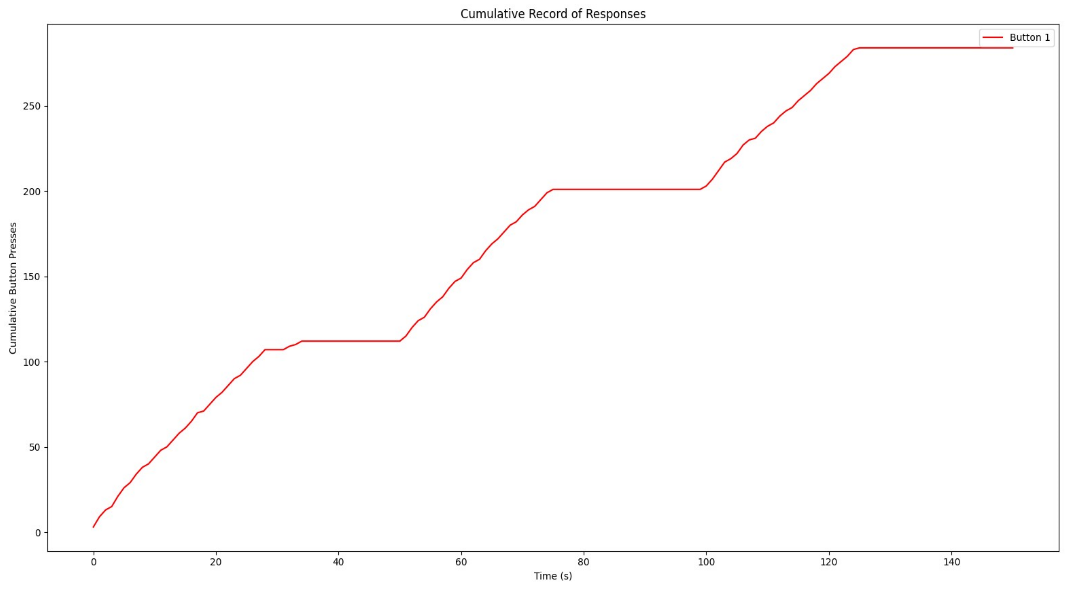


Figure 3. Sample output of cumulative record responses. *Note:* Cumulative record data paths are colored to match the corresponding response button.

an NCR schedule is greater than or equal to the specified interval. If it is, reinforcement is delivered. For a response cost, the schedule is functionally identical to the fixed-ratio schedule. However, instead of providing reinforcement, it delivers a punishment (decrementing the user's points and providing aversive feedback) when the count of button presses is a multiple of the specified ratio. The implementation of these schedules in the code accurately reflects their theoretical descriptions as will be demonstrated through validation.

Data Collection and Results

The program's data collection process is initiated the moment the user starts interacting with it. With every press of a button, a function is activated, logging the global timestamp of the event and incrementally increasing the count of button presses. The function also records the time of the corresponding reinforcement event when the function is triggered. Simultaneously, a separate function operates every second to monitor the conditions for the NCR schedule, updating the reinforcement timestamp when the condition is met. The data collected throughout the experiment is stored in real-time within the program's memory. When the experiment concludes, this collection of button-press timestamps and other pertinent data is saved in a Microsoft Excel file. This function ensures that the dataset is safely stored for further analysis post-experiment. Another function then steps in to provide a visual interpretation of the collected data. It produces two kinds of graphs: a cumulative record displaying the total number of button presses over time (Figure 3), and a rate-of-responding graph which presents the frequency of button presses (Figure 4). These visuals provide an immediate depiction of the user's interaction pattern throughout the experiment. Additionally, a summary of the results is tabulated to offer a snapshot of the key data points. This task is performed by a function that calculates the total responses and average rate of responding for each button across different phases of the experiment. The data is neatly arranged in a table, providing quick access to the phase number, button number, phase duration, total responses, and the average rate of responding for each button in each phase. This table is also saved to an Excel file, providing a reference for researchers to further analyze the data (Figure 5).

VALIDATION

Dependent-Variable Validation

To validate the efficacy of our data collection process, particularly concerning the dependent-variable values, we implemented a 25-s

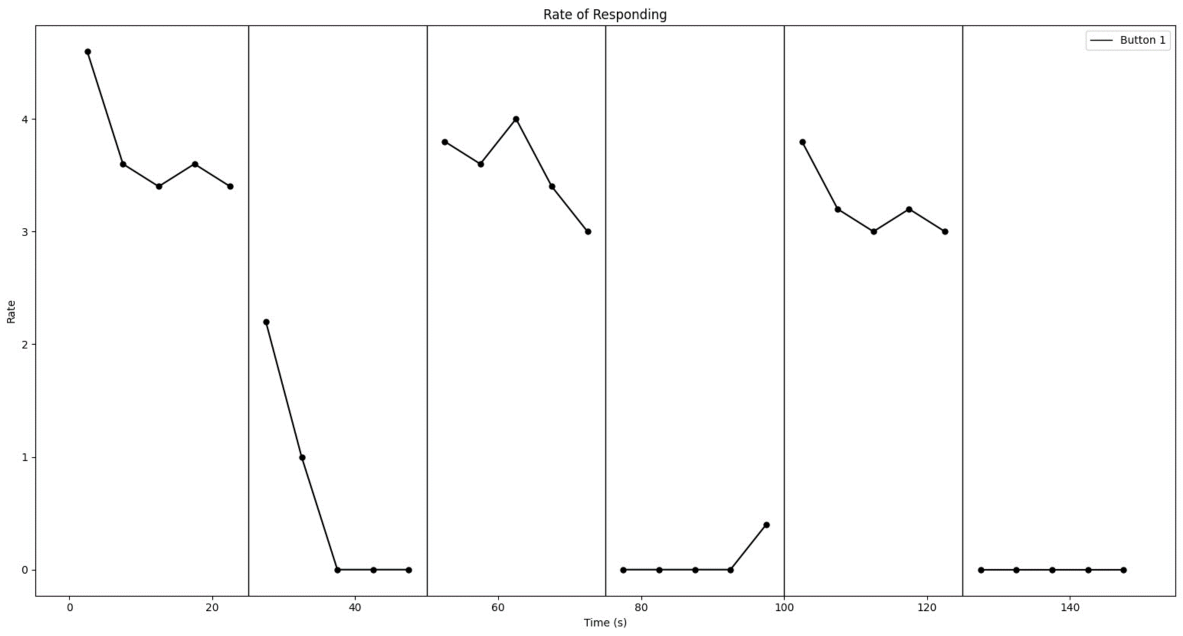
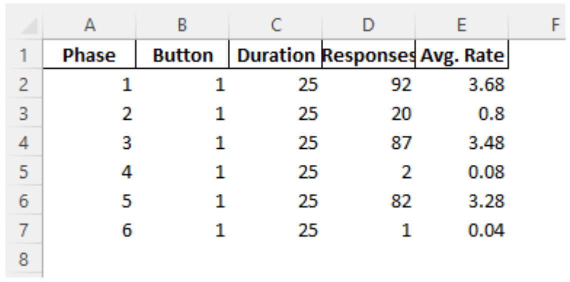


Figure 4. Sample Output of Rate of Responding. *Note:* Rate is calculated as the average number of responses per second for each 5 s interval.

experimental trial. During the initial 5-s interval of this experiment, a single response to the target button was delivered. Progressing into the subsequent intervals of 5 s each, we increased the number of responses by one response per interval, culminating in five responses in the last five-second interval. This trial design was deliberately chosen due to our program's function to track and compute response rates. It operates by aggregating the total number of responses for each 5-s interval, dividing it by five. The resultant figure is then recorded as a data point on the response rate graph. Given the approach of our testing, we predicted that the data would illustrate a stable, ascending linear progression, mirroring the systematic increments in the responses. Indeed, our expectations were met as depicted in Figure 6. Additionally, the program reported an average response rate of 0.6 responses per second, with a cumulative total of 15 responses. This aligns with our actual number of responses, further substantiating the validity of the data collection function.

Reinforcement Schedule Validation

To validate the reinforcement schedules, we tested the software across various schedule types and values, closely mirroring the method employed by Smith and Greer (2022). For each schedule, we collected data until 35 reinforcers were delivered. This approach permitted fluctuations in the trial durations, aligning with the principles of reinforcement schedules: richer schedules resulted in shorter trial durations to reach the reinforcement quota, whereas leaner schedules necessitated longer durations to achieve the same number of reinforcers. For interval schedules, we calculated the mean time between reinforcers. For ratio schedules, our focus was on determining the mean number of responses before reinforcement was delivered. Additionally, we computed standard deviations to gauge the variability of these measures. These calculated metrics, represented in Table 1, were compared with the predefined schedule values



Phase	Button	Duration	Responses	Avg. Rate
1	1	25	92	3.68
2	1	25	20	0.8
3	1	25	87	3.48
4	1	25	2	0.08
5	1	25	82	3.28
6	1	25	1	0.04

Figure 5. Sample Output of Excel Data. *Note:* Experimenters have the option to select which response options to include in the generated Excel file.

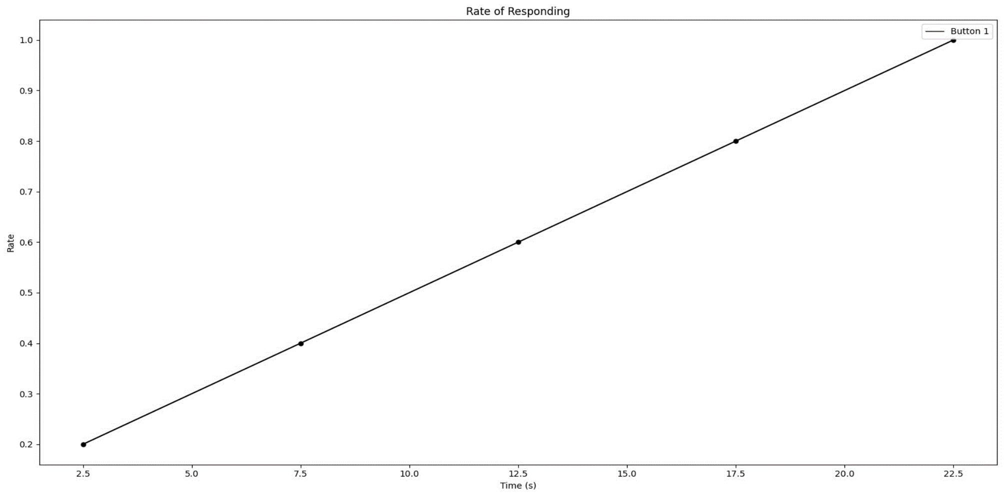


Figure 6. Dependent Variable Rate Validation. *Note:* Each data point corresponds to a single 5 s interval.

to assess the software's adherence to established operant scheduling principles. The results of this validation process, summarized in Table 1, suggest that the software closely aligns with the specified reinforcement schedules. This is particularly evident in the "Percentage within range" column, where a score of 100% represents full alignment with the set schedule values.

The alignment of the software with the theoretical reinforcement values indicates its reliability for use in human-operant research. Given the rigorous validation and its consistency with methods outlined by Smith and Greer (2022), this software is poised to be a valuable tool in the field, demonstrating its capability to yield consistent and precise outcomes.

DISCUSSION

This report introduces and validates a new software tool developed for human-operant experimental designs. As behavior analysis has advanced, particularly in the realm of human-operant research (Lattal & Perone, 1998), there is a growing demand for reliable tools. Our software, written in Python, seeks to meet this need, offering a platform for researchers to further investigate human-operant translational research. The software emphasizes ease-of-use and technical reliability. To validate the software, we adopted the validation methods from Smith and Greer (2022) with the exception of the functionality testing as the use of Python's rigorously tested and widely used library made functionality testing redundant. Our testing across various frameworks and domains indicated that the software produces accurate, replicable, and consistent results. It is important to note, however, that the software was tested by individuals who were not entirely naïve to behavior analytic procedures, which might not entirely reflect real-world research contexts.

VirtuOperant utilizes the Python programming language, which is renowned for its ability to function across different operating

	Schedule Values		
Condition	***M***	***SD***	**% within range**
Variable-interval 5 s	5.28 s	1.63 s	100
Variable-interval 10 s	9.91 s	2.81 s	100
Variable-ratio 2	1.86	1.33	100
Variable-ratio 5	5.43	3.60	100
Fixed-ratio 1	1	0	100
Fixed-ratio 5	5	0	100
Response cost 1	1	0	100
Response cost 5	5	0	100
Noncontingent reinforcement 5 s	5	0	100
Noncontingent reinforcement 10 s	10	0	100

Table 1. Validation of reinforcement schedules. *Note:* Response cost functions identically to the fixed-ratio schedule of reinforcement with the delivery of a correlated stimulus and score decrement instead of reinforcement.

systems. This cross-platform compatibility broadens the utility of our program for researchers in various settings. Moreover, once users have installed Python, the requisite libraries, and our software, continuous connection to the internet is not necessary. These attributes offer benefits that enhance the practicality and dependability of our software for human-operant research endeavors. Many contemporary studies have utilized comparable software to evaluate operant behavior in human subjects (Smith & Greer, 2023; Kimball et al., 2023). Providing open-source software that can be customized for human-operant research could significantly enhance the efficiency of translational studies and contribute to the rapid growth of scholarly work in this area.

For future development, there is potential to enhance the software's capabilities to simulate more intricate operant scenarios. The present validation results show the software's potential utility in human-operant research. On a broader scale, such tools are crucial in bridging the gap between the experimental analysis of behavior and applied behavior analysis. As highlighted by Jarmolowicz (2018), a growing disconnect exists between these two areas, which might prevent the effective translation of experimental results to applied contexts. Our software aims to offer a resource for both experimental and applied research, promoting a more integrated approach between experimental analysis of behavior and applied behavior analysis.

REFERENCES

Bernal-Gamboa, R., Nieto, J., & Gámez, A. M. (2020). Conducting extinction in multiple contexts attenuates relapse of operant behavior in humans. *Behavioural Processes, 181*, 104261. https://doi.org/10.1016/j.beproc.2020.104261

Bolívar, H. A., & Dallery, J. (2020). Effects of response cost magnitude on resurgence of human operant behavior. *Behavioural Processes, 178*, 104187. https://doi.org/10.1016/j.beproc.2020.104187

Jarmolowicz, D. P. (2018). EAB is fine, thanks for asking. *Behavior Analysis: Research and Practice, 18*(2), 169–173. https://doi.org/10.1037/bar0000116

Kestner, K. M., & Peterson, S. M. (2017). A review of resurgence literature with human participants. *Behavior Analysis: Research and Practice, 17*(1), 1–17. https://doi.org/10.1037/bar0000039

Kestner, K. M., Diaz-Salvat, C. C., St. Peter, C. C., & Peterson, S. M. (2018). Assessing the repeatability of resurgence in humans: Implications for the use of within-subject designs. *Journal of the Experimental Analysis of Behavior, 110*(3), 545–552. https://doi.org/10.1002/jeab.477

Kimball, R. T., Salvetti, E. L., Day, L. E., Karis, R., Silveira, J., & Kranak, M. P. (2023). Operant ABA renewal during dense and lean schedules of differential reinforcement. *Journal of the Experimental Analysis of Behavior, 119*(3), 529–538. https://doi.org/10.1002/jeab.840

Lattal, K. A., & Perone, M. (Eds.). (1998). *Handbook of Research Methods In Human Operant Behavior*. Plenum Press.

Okouchi, H. (2015). Resurgence of two-response sequences punished by point-loss response cost in humans. *Revista Mexicana de Análisis de La Conducta, 41*(2), 137–154. https://doi.org/10.5514/rmac.v41.i2.63744

Perone, M. (1985). On the impact of human operant research: Asymmetrical patterns of cross-citation between human and nonhuman research. *The Behavior Analyst, 8*(2), 185–189. https://doi.org/10.1007/bf03393150

Ritchey, C. M., Kuroda, T., Rung, J. M., & Podlesnik, C. A. (2021). Evaluating extinction, renewal, and resurgence of operant behavior in humans with Amazon Mechanical Turk. *Learning and Motivation, 74*, 101728. https://doi.org/10.1016/j.lmot.2021.101728

Saini, V., & Mitteer, D. R. (2019). A review of investigations of operant renewal with human participants: Implications for theory and Practice. *Journal of the Experimental Analysis of Behavior, 113*(1), 105–123. https://doi.org/10.1002/jeab.562

Smith, S. W., & Greer, B. D. (2022). Validating human-operant software: A case example. *Behavior Analysis: Research and Practice, 22*(4), 389–403. https://doi.org/10.1037/bar0000244

Smith, S. W., & Greer, B. D. (2023). Translational evaluation of on/Off Alternative Reinforcement Cycling. *Journal of the Experimental Analysis of Behavior, 120*(3), 429–439. https://doi.org/10.1002/jeab.879

Sweeney, M. M., & Shahan, T. A. (2016). Resurgence of target responding does not exceed increases in inactive responding in a forced-choice alternative reinforcement procedure in humans. *Behavioural Processes, 124*, 80–92. https://doi.org/10.1016/j.beproc.2015.12.007