SemanticOn: Specifying Content-Based Semantic Conditions for Web Automation Programs

Kevin Pu
jpu@dgp.toronto.edu
University of Toronto

Rainey Fu
rainey.fu@mail.utoronto.ca
University of Toronto

Rui Dong
ruidong@umich.edu
University of Michigan

Xinyu Wang
xwangsd@umich.edu
University of Michigan

Yan Chen
yanchen@dgp.toronto.edu
University of Toronto

Tovi Grossman
tovi@dgp.toronto.edu
University of Toronto

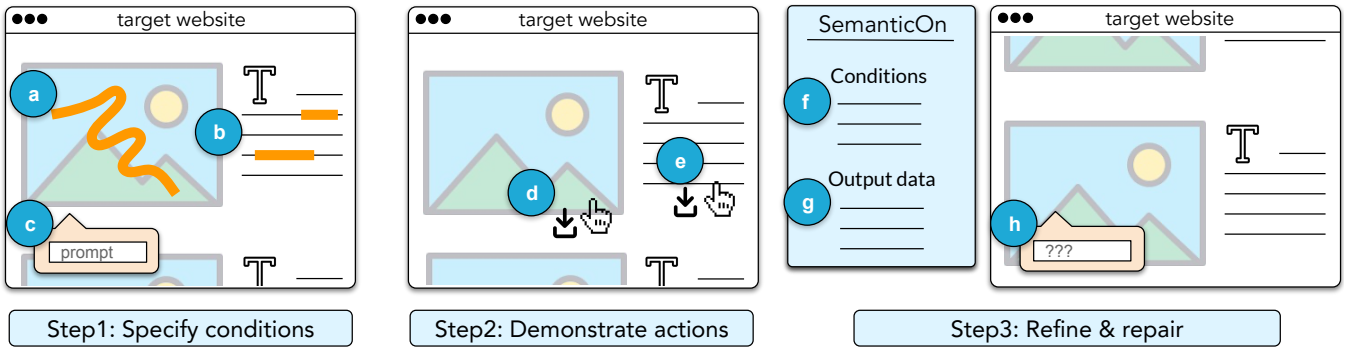


Figure 1: The workflow of SemanticOn. There are three steps to creating a web automation program with semantic conditions using SemanticOn. (Step 1) To specify semantic conditions, users can either describe their intent in text (*User Enters*, ⓒ) or indicate the section of interest by brushing through an image ⓐ or highlighting parts of a text ⓑ (*System Suggests*). SemanticOn then encodes these specifications with computer vision and natural language processing techniques into web program conditions. (Step 2) To create the intended web automation program, users demonstrate the actions on the website using WebRobot, including image downloading ⓓ and text scraping ⓔ. (Step 3) Once the program is executed, users can also easily coordinate with SemanticOn to refine the semantic conditions (ⓕ, ⓗ) or take back control to add or remove data manually ⓖ.

ABSTRACT

Data scientists, researchers, and clerks often create web automation programs to perform repetitive yet essential tasks, such as data scraping and data entry. However, existing web automation systems lack mechanisms for defining conditional behaviors where the system can intelligently filter candidate content based on semantic filters (e.g., extract texts based on key ideas or images based on entity relationships). We introduce SemanticOn, a system that enables users to *specify*, *refine*, and *incorporate* visual and textual semantic conditions in web automation programs via two methods: natural language description via prompts or information highlighting. Users can coordinate with SemanticOn to refine the conditions as the program continuously executes or reclaim manual control to repair errors. In a user study, participants completed a series of conditional web automation tasks. They reported that SemanticOn helped them effectively express and refine their semantic intent by utilizing visual and textual conditions.

KEYWORDS

Web automation, PBD, user intent, semantics

ACM Reference Format:
Kevin Pu, Rainey Fu, Rui Dong, Xinyu Wang, Yan Chen, and Tovi Grossman. 2022. SemanticOn: Specifying Content-Based Semantic Conditions for Web Automation Programs. In *The 35th Annual ACM Symposium on User Interface Software and Technology (UIST '22), October 29-November 2, 2022, Bend, OR, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3526113.3545691

UIST '22, October 29-November 2, 2022, Bend, OR, USA

ACM ISBN 978-1-4503-9320-1/22/10…$15.00
https://doi.org/10.1145/3526113.3545691

1 INTRODUCTION

Enterprises, governments, and schools often use web-based applications to manage their businesses and services. Other than information consumption, users such as clerks, data scientists, and researchers often employ these web platforms to conduct tasks that are repetitive yet essential, such as data scraping and data entry. Performing these tasks manually can often lead to human errors (e.g., data duplicates, missed entries), which can cause inefficiencies.

Web automation offers a solution that leverages bots to mimic human interactions on web applications. It assists users with tedious and recurring tasks and has proven to be faster and more accurate for various task types compared to manual effort [42].

Past research has developed techniques to help users of all expertise levels to quickly and accurately create their intended web automation programs [25–27, 44, 68]. However, these techniques are limited to creating programs with requirements at the website syntax or structural level (e.g., scraping the first two items in each row of a table). Tools capable of creating logic based on the meaning of the content (semantics) remain unexplored. For instance, commercial tools such as iMacros [2] and UiPath [8] enable users to perform record-and-replay interactions for web automation and testing. Research tools such as Helena [15] further this technique by lowering the learning curve, allowing users with little programming experience to create complex programs that can handle hierarchical data (e.g., tree-structured data) and distributed data spread across multiple websites.

We identified a need for web automation with semantic conditions through prior user studies [24] and analysis of real user requests in online forms [3, 7]. This includes vision-related semantic conditions, such as scraping images that meet specific criteria (e.g., a photography student wants to study group interaction portraits on a gallery website with thousands of photos) or text-related semantic conditions, such as scraping text only when it expresses particular sentiments (e.g., a film critic wants to evaluate positive reviews of a movie star's acting from dozens of news articles in a journal). With current techniques, users cannot specify these semantic intents in web automation programs. As noted above, semantic information often varies by content type, which makes it hard to design a universal interaction that is both easy to use and sufficiently expressive. Additionally, unlike other AI systems that provide results immediately after the provision of user inputs (e.g., chatbots), once executed, a web automation program will continuously output results as it iterates over web contents. This makes monitoring and error handling difficult, as the program may encounter unforeseen and problematic cases.

This paper explores interactive techniques to enable content-based semantic condition specification for web automation programs. We introduce SemanticOn,[1] a system that allows users to *specify*, *refine*, and *incorporate* visual and textual semantic information as conditions in web automation programs via two methods: natural language description via prompts or detailed information highlighting with system support. We define them as *User Enters* and *System Suggests*, respectively. SemanticOn combines the relative strengths of neural models (Transformer) for unstructured information and program synthesis techniques for web automation. By doing so, we introduce a new interaction paradigm for users to continuously add/refine semantic conditions in a programming-by-demonstration system. Specifically, SemanticOn builds upon WebRobot [24], a *program synthesis* system that enables users to create web automation programs by demonstrating actions on the target websites. WebRobot employs a no-code development approach that requires only web interactions in place of programming knowledge from users, which is consistent with our design goal.

[1]SemanticOn is an acronym for **semantic con**dition

Figure 1 depicts the three steps of using SemanticOn. (Step 1) To specify semantic conditions, users can either describe their intent in a sentence (Fig. 1.c), indicate their area of interest by brushing through an image (Fig. 1.a), or highlight parts of a text (Fig. 1.b). SemanticOn uses similarity-based computer vision and natural language processing techniques to encode these specifications into web program conditions. (Step 2) To create the intended web automation program, users will demonstrate actions on the website using WebRobot, including image downloading (Fig. 1.d) and text scraping (Fig. 1.e). (Step 3) Once the program is executed, users can also easily coordinate with SemanticOn to refine the semantic conditions based on the automatically detected information (Fig. 1.f, Fig. 1.h) or reclaim control to manually add or remove data if the program has misjudged (Fig. 1.g). To our knowledge, SemanticOn is the first system to explore content-based semantic specification interactions for web automation programs.

We conducted a user study with 10 participants to evaluate SemanticOn's overall usability and efficiency and to compare the semantic condition specification of each method (*User Enters* and *System Suggests*). We found that participants using SemanticOn successfully extracted 80.8% of the required data with an average time of 06:10 minute:second per task. The participants found that SemanticOn helped them effectively express their semantic intent by prompting them to consider their visual and textual perceptions of the tasks. We found a sense of control vs. effort trade-off, where participants enjoyed composing their conditions in *User Enters* but had to spend more time and mental effort devising a description to encapsulate the semantic condition. On the other hand, while participants could specify and refine conditions more easily via highlighting content details and selecting generated conditions in *System Suggests*, they had less freedom to express their intent when system suggestions were inaccurate.

In the final section of this work, we analyze the human-AI collaboration workflow in SemanticOn, discuss the implications of adding similarity-based models in a symbolic PBD system, and explore future work that can adapt our approach to other types of interactive AI systems that require semantic conditions. This work is an essential step towards the vision of natural, intent-unambiguous end-user programming with a focus on web automation creation. This paper makes the following contributions:

- The *User Enters*, *System Suggests* interaction designs, implementations, and evaluations that allow users to specify and demonstrate their intent during web automation creation,
- The refinement and error-handling techniques to clarify and improve semantic filters in a continuous human-AI collaboration process,
- SemanticOn, along with a user study showing its usability and effectiveness in helping users specify semantic conditions for web automation programs.

2 RELATED WORK

SemanticOn builds on decades of web automation systems and innovations. In this section, we draw our design goals and guidance from three areas of work: web automation, programming-by-demonstration, and user intent specification and refinement.

2.1 Web Automation

Web automation is a software technique that leverages bots to perform tedious and recurring web tasks by mimicking human interactions, such as data entry and data extraction. Data scientists, UI testers, and clerks all use web automation to help complete their domain-specific tasks [40, 43, 49, 69]. Social scientists, for example, might want to develop web data scraping programs to collect necessary web datasets. UI testers might want to create an automated browser testing program to help developers find front-end defects. Data workers might envision a data entry program for routine tasks like entering large amounts of data into a digital system (e.g., booking flights for all employees).

Creating web automation programs is a non-trivial and complex task. Many web automation tools require users to have domain knowledge (e.g., understand the Document Object Model (DOM) structure) and programming experience. Commonly used tools like Puppeteer [4], Selenium [6], Scrapy [5], and Beautiful Soup [1] require users to learn code syntax, understand the task content architecture (e.g., DOM tree hierarchy), and have software testing experience. Prior work has shown that even for professional developers, creating automation programs is time-consuming. Krosnick and Oney studied the challenges of writing web macros using common web automation frameworks for experienced programmers [35]. They found that a primary challenge for participants was the labor of checking syntactical element selectors to create their programs, which was inefficient and prone to mistakes. In addition, the program might not generalize to cross-webpage selections where the elements don't have syntactic similarity. Our work enables users to specify the semantic meaning of their target content, bypassing the issues caused by implementation.

Researchers have developed many helpful tools to reduce the effort of program creation. For desktop application automation, systems like Sikuli [76] allow users to identify a GUI element (e.g., an icon or a toolbar button) by taking its screenshot. Using computer vision techniques, it analyzes patterns in the screenshots to locate the appropriate elements when automating GUI interactions. Although this approach is promising, it requires programming knowledge and cannot disambiguate similar elements or text information. For UI testing, researchers have proposed and studied crowdsourcing and automated testing strategies to help increase the testing coverage and reduce the effort of creating programs [21, 23]. While helpful, outputs produced with these tools are hard to generalize to new UIs or contexts.

2.2 Programming-by-Demonstration

To further reduce the expertise required, many tools have used a programming-by-demonstration (PBD) approach where users only have to interact with the target applications rather than writing code [12, 36, 38]. These span a variety of application domains including text manipulation [13, 39, 54, 60, 75], image or video editing [37, 47, 51], and GUI synthesis [55, 57, 61, 71]. In the context of web applications, PBD delivers on this first design requirement, offering web automation without requiring users to understand browser internals or manually reverse-engineer target pages. The PBD approach has produced great successes in the web automation domain, most notably CoScripter [42], Vegemite [48], Rousillon [15], and iMacros [2].

Some of these systems require users to edit their traces to add parametrization. For instance, CoScripter and iMacros offer record-and-replay functionality; users record themselves interacting with the browser—clicking, entering text, and navigating between pages—and the tool writes a loop-free script that replays the recorded interaction. Because they lack support for control constructs and function composition, these systems require users to have logic skills. Other systems support iteration using program synthesis, automatically discovering loops given a demonstration of one or a few iterations. While less domain knowledge is needed, the synthesizer can make mistakes in which the user must provide more demonstrations or edit the DSL to correct it (e.g., Helena), which can be frustrating. SemanticOn instead allows users to effectively coordinate with PBD systems by smoothly switching agency and editing constraints at any time during program execution.

2.3 User Intent Specification and Refinement

User intent specification is an important and challenging component of human-AI collaboration. Ideally, users should be able to easily and naturally specify their intent to a system while understanding its states. However, given the limited capabilities of AI understanding techniques, high-level user intent can be difficult for systems to comprehend. Many systems have proposed bridging the gap between user intent and system understanding. For instance, PLOW [9] and PUMICE [46] allow users to express concepts (e.g., hot weather) in natural language and then learn the concepts to generalize the automation. Systems like Scout [70], Designscape [64], and Iconate [80] allow users to iteratively refine their intent by directly manipulating the AI-generated artifacts. Other studies have shown that this refinement interaction can even be delegated to crowd workers [18]. Another work, APPINITE [45], also encapsulates user's intent in natural language instructions and clarifies the intention in a back-and-forth conversation with the AI. While these approaches are promising, user intents can involve visual and cognitive details such as identifying visual relationships in images or parsing texts to match a high-level idea. The user's semantic level intents are often not fully or accurately expressed through natural language or limited examples only, leading to information loss during communication and rendering the communication ineffective [19].

Similar to PBD systems, programming-by-example (PBE) is another approach to facilitate program creation for various tasks such as data wrangling [29, 30, 34] and data visualization [52, 72]. Many PBE and PBD systems require users to provide additional examples to disambiguate user intent. Falx allows users to specify visualization examples using a small amount of data and then infers and transforms the data to match the design [73]. Sporq allows users to more accurately and quickly search code patterns in large codebases by prompting them to refine their intent by annotating a batch of negative examples and adding specific constraints [58]. Other works enable users to directly annotate their input examples (augmented examples) to disambiguate user intent [66, 78]. Or they employ data visualization techniques to showcase the generated

programs, allowing users to tweak the path of program generation in a tree view [77]. While promising, providing additional examples increases users' cognitive demand. In this work, we focus on addressing the ambiguous semantic conditions and designing human-AI collaboration interaction solutions to help refine the constraints based on the content.

Using machine learning (ML) models to refine intent has been a recent focus in the field of interactive ML. One common interactive ML approach allows users to offer feedback during the model training process for more effective ML model creation [16]. Work by Cai et al. allows users to adjust the search algorithm iteratively with different types of similarities at different moments [14]. Projects by Austin et al. and Jiang et al. allow users to interact with large language models to help refine their intent when writing code snippets [11, 33]. Work by Amershi et al. allows users to identify new friend groups on social media by analyzing the examples presented [10]. Software developed by Fogarty et al. helps users to create their own rules to improve the search results [28]. This research inspires our work, but instead, we focus on helping users refine their intent while interacting with continuous AI systems—web automation programs that require constant monitoring and that effectively coordinate the turn-taking.

3 BACKGROUND AND DESIGN GOALS

Our work is built upon an existing web automation system, WebRobot [24], that only uses web interactions and requires no programming knowledge from its users. This is consistent with our design goal. In addition to prior work, we derive our design goals from WebRobot's user study. In this section, we provide necessary background information on the WebRobot system and then discuss the design goals for our system SemanticOn.

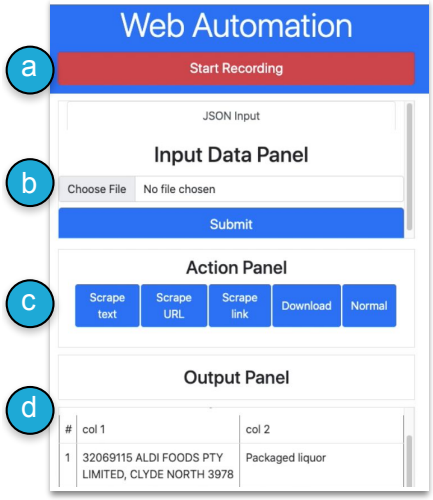


Figure 2: A screenshot of the WebRobot system UI.

3.1 The WebRobot System Workflow

WebRobot is designed to facilitate web automation program creation. Figure 2 shows the WebRobot user interface. To create a web automation program for a data entry or scraping task, a user first starts recording their actions (Fig. 2.a). Then they can either upload a `JSON` file (Fig. 2.b) if the task involves data entry, or they can choose an appropriate action (e.g., Scrape Text) in the action panel (Fig. 2.c) and perform the required actions (e.g., clicking the desired text data on the website). After each scraping action, they will see the data appended to the output panel (Fig. 2.d). Behind the scenes, WebRobot records every user action on the website and its associated action type (e.g., Scrape Text). After a few demonstrations, WebRobot synthesizes a program P from the trace A of demonstrated actions. In particular, WebRobot guarantees that P not only *reproduces* the demonstrated actions from A but also generalizes beyond A. This typically implies P would contain loops that can be used to automate the user-intended task. Finally, WebRobot executes P to automate the rest of the actions in the task.

procedure SYNTHESIZE (A)
input: $A = [a_1, \cdot\cdot, a_m]$ is a trace of user-demonstrated actions.
output: a program P that generalizes A.
1: $P_0 := a_1; \cdot\cdot; a_m$;
2: $W := \{P_0\}$; $\widetilde{P} := \emptyset$;
3: **while** $W \neq \emptyset$
4: $P := W$.remove();
5: **if** P *generalizes* A **then** $\widetilde{P}$.add(P);
6: $W' :=$ REWRITE(P);
7: $W := W \cup W'$;
8: **return** RANK($\widetilde{P}$);

Algorithm 1: Rewrite-based program synthesis algorithm.

3.2 WebRobot's Synthesis Algorithm

In a nutshell, WebRobot's synthesis algorithm (Algorithm 1) generalizes an input action trace A into a program P (with loops) by iteratively rewriting A to loops in P *from the inside out*. Initially, it creates a program P_0 with exactly those actions in A (line 1): while P_0 reproduces A, it does not generalize A (i.e., it does not produce new actions after A). Therefore, the algorithm performs iterative rewriting to gradually "compress" P_0 into more compact and general programs using a worklist algorithm (lines 2-8). The worklist W is initialized to have only P_0, and we use $\widetilde{P}$ to keep track of all programs that generalize A (line 2). Whenever W is not empty (line 3), the algorithm would remove a program P from W (line 4). It then checks to see whether P generalizes A; if so, P is added to $\widetilde{P}$ (line 5). After this, in line 6, the algorithm tries to rewrite P into more general programs, which are stored in W'. The key idea underlying our REWRITE procedure is to perform *semantic rewriting* using a methodology called *speculate-and-rewrite*. Intuitively, it inspects P, identifies repetitive patterns in P, hypothesizes potential loops that correspond to P, and finally synthesizes programs with one more level of loop. How WebRobot's speculative writing process works is beyond the scope of this work; we refer interested readers to the original WebRobot paper [24] for details. Once P is rewritten to a new set of programs W' (line 6), the algorithm simply merges W' into W (line 7). The worklist loop terminates when no programs

can be rewritten and it finally returns the smallest program in $\widetilde{P}$ using a ranking function (line 8).

3.3 User Feedback

In WebRobot's user study, participants reported that while WebRobot can help lower barriers of entry for the creation of web automation programs and handling a more comprehensive range of tasks, they wished that they could express conditions to filter the content. For instance, one participant said, *"maybe some conditional scraping [can be included], not based on whether the element exists in the webpage, but based on some other conditions."* Consistently, we found posts on forums such as iMacros [3] and Stack Overflow [7] that request the creation of web automation programs with content-based conditions. Participants also wished to refine their intent when interacting with the WebRobot system. For instance, participants reported that they wanted to *"undo my wrong manipulations"* or *"edit my history."* However, as noted in the related work, WebRobot and other systems do not effectively support actions such as undo or history manipulation.

3.4 Design Goals

Based on this prior work, we devised the following three design goals to help users easily create web automation programs with semantic conditions.

- **DG1: Ability to express content-based semantic conditions:** Users can specify semantic conditions when creating web automation programs.
- **DG2: Accessible user intent refinement:** Users can iteratively refine their semantic conditions at any time of the program creation process.
- **DG3: Responsive error handling for mistakes made by users and the system:** Users need to modify inaccurate conditions and edit scraped data easily.

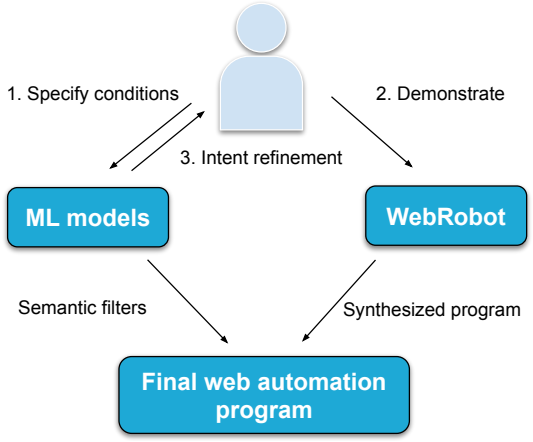


Figure 3: SemanticOn's system architecture.

4 SEMANTICON

With the three design goals above, we created SemanticOn to help users specify, refine, and incorporate semantic conditions in automated web data scraping. Figure 3 shows SemanticOn's system architecture and main user interactions at a high level. Instead of writing web macros from scratch for each website and filtering the scraped content in post-processing, users can interact with SemanticOn to compose semantic conditions on the content they want to scrape (Step 1 Fig. 3), then demonstrate actions on the target website to synthesize automation programs for different websites without writing a single line of code (Step 2 Fig. 3). Throughout this process, users can communicate with SemanticOn, which is capable of parsing text and image content through machine learning models. The users and SemanticOn work together to refine the condition set and repair errors in result selection, continuously improving both the system's and user's understanding of the filter criteria (Step 3 in Fig. 3). In this section, we first illustrate SemanticOn's user experience with a sample scenario that embodies common semantic conditions. We then detail the design and implementation of SemanticOn.

4.1 The SemanticOn User Experience

Mia, an outdoor enthusiast, wants to extract online information about travel destinations where outdoor activities are available. To help her make an informed decision, Mia wants to scrape the text description and the image for each location from an article to build potential itineraries. One option is to read through every paragraph, look at each picture, and manually copy and paste the relevant information, but that process would be tedious and repetitive. On the other hand, Mia could write a web scraping script using Python. She has some coding experience, but writing a script and filtering the results based on her preference would also be time-consuming and laborious. Instead, Mia uses SemanticOn to efficiently demonstrate her conditions and web actions and synthesize a web automation program that completes the task for her.

To begin, Mia sets the semantic conditions for the intended content (Step 1 Fig.3). She first clicks "Text Condition". She then selects *User Enters* (Fig. 4.d) to specify the semantic condition in her own words. Mia represents her high-level requirement in the system prompt by typing, *"This is a great location for outdoor activities"* (Fig. 6.c). She believes this sentence is likely semantically similar to the relevant content in this article. After clicking "Add", the condition is appended to the text condition table (Fig. 4.h) in the condition panel. Furthermore, Mia decides to add a condition to the corresponding destination image. She wants to travel to a place where hiking and water activities are accessible. To that end, she uses *System Suggests*, clicks on an ideal image, and highlights the mountain and lake in the picture (Fig. 6.a). The system detects several objects and summarizes the image content into a sentence. Mia also adds the relevant objects and caption to the corresponding tables (Fig. 4.f,g) in the condition panel.

After specifying two initial conditions, Mia decides to start the demonstration process (Step 2, Fig.3). She clicks the "Start Recording" button (Fig. 4.a) to start the web macro recording for program synthesis. Then, Mia specifies the task name as "Travel Destination Search" and sets the column number to 2, one for text descriptions and one for the associated images.

Next, Mia selects "Download Image" and hovers the mouse to highlight the desired image element (Fig. 4.j). As she clicks on the element, the image is downloaded and put into the first column

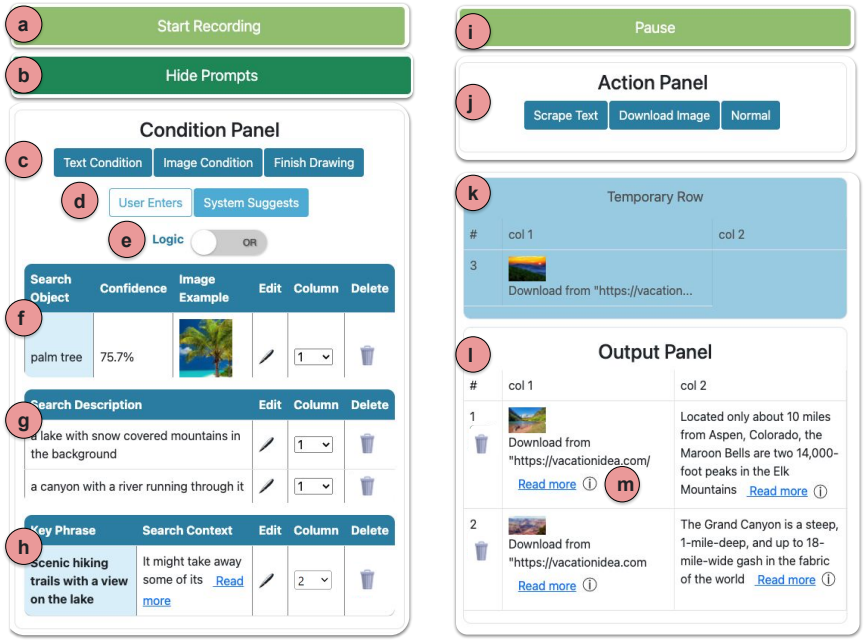


Figure 4: Overview of SemanticOn's user interface. The user begins program synthesis by clicking Start Recording ⓐ. The user can disable or enable system suggested prompts in automation ⓑ. The user can specify conditions for images or text ⓒ and choose between manually input or select system-suggested conditions ⓓ. The user can also toggle between OR and AND logic for the condition set ⓔ. A unique table displays image object conditions ⓕ, image description conditions ⓖ, and text description conditions ⓗ. The user may pause ⓘ at any time to make changes that will not affect automation. To interact with the web page and demonstrate actions for program synthesis, the user will select their action type by choosing an option in the action panel ⓙ. The system will predict the next action, shown in the temporary row ⓚ. If the semantic conditions are met, the temporary row will be appended to the output panel ⓛ. To learn more about how the content match with the conditions, the user may click on the information icon ⓜ.

of the Temporary Row (Fig. 4.k). SemanticOn evaluates conditions associated with each column before the row is added to the output panel or skipped. However, at the demonstration stage, the condition will not be triggered, as we assume users will only demonstrate macro actions on the content they want to include. Mia then selects the "Scrape Text" action and repeats the steps for the destination description. As both columns in the temporary row are filled, the entire row is added to the output panel (Fig. 4.l).

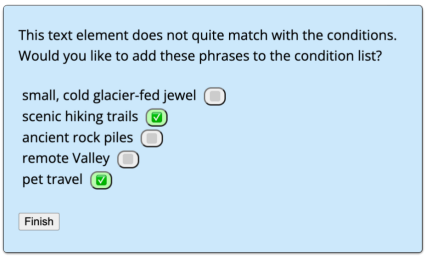


Figure 5: Example of system prompt when new content does not match with current condition.

Mia repeats her demonstration for a second destination that fits her requirement. After two rows of user demonstration, WebRobot can synthesize the web automation program and predict the user's next step, so the system now enters a guided semi-automation mode. SemanticOn scrolls into the next row of the destination and highlights the next element to be extracted. For the image, a prompt states that the generated caption matched with the initial image condition *"a lake beneath the mountain range"*. Mia confirms this selection and lets SemanticOn continue. However, Mia finds that her initial text condition is too general, as the system informs her the text condition does not match the current text and presents key phrases from this element to be potentially used as refinement (Fig.5). She realizes that *"scenic hiking trails"* would be a good textual condition to include. She also discovers *"pet travel"* as a key phrase. Mia would love to take her dog Sushi on the trip, so she also selects that key phrase. After she clicks "Finish," those two phrases are added to the text condition table, and the highlighted text is considered accepted.

After inspecting several elements in semi-automation mode and refining the condition set with system prompts, Mia feels satisfied with the system's interpretation of her requirements. She clicks on the "Hide Prompt" button (Fig. 4.b) to allow the system to automatically predict and filter the rest of the web page content without stopping for user confirmation. Mia sits back and waits for the program to finish. However, she notices that for one destination, *"Grand Canyon, USA,"* the system filters the image as it does not match *"a lake beneath the mountain range"*. However, she actually wants to include this destination in her list, as she can still engage in outdoor activities such as hiking and kayaking. To repair this error,

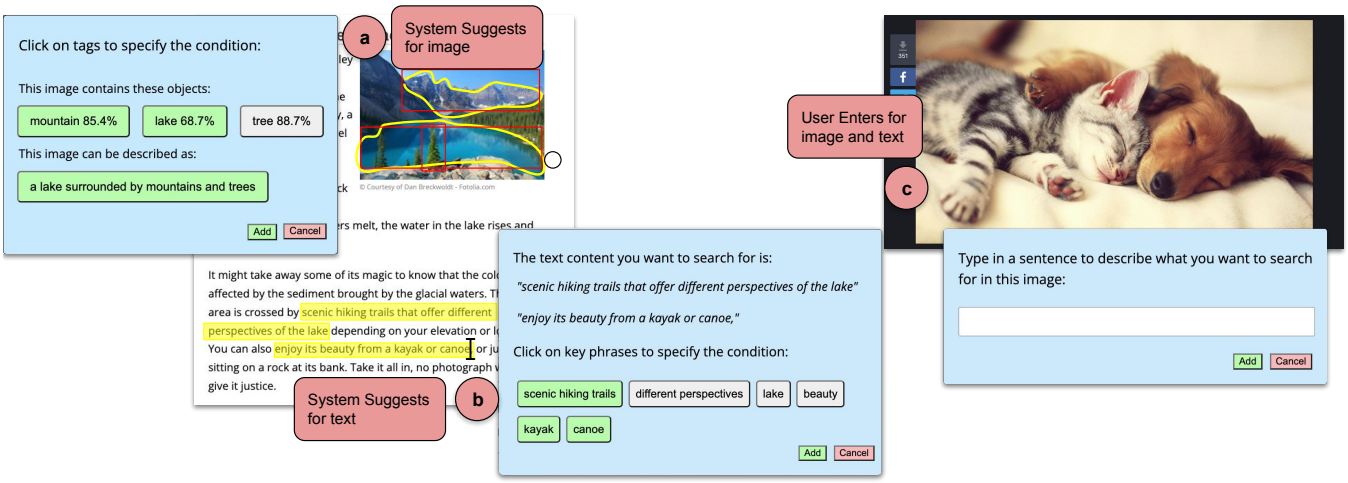


Figure 6: Two condition specification methods in SemanticOn. After entering the system suggests mode, the user may draw on images. Then, a prompt ⓐ containing detected objects within that region, and a general description of the image will be shown. Similarly, the user may highlight sections of text, and the system will prompt ⓑ important key phrases within the highlighted portion. Users may manually enter conditions by clicking on an image or section of text and entering their conditions ⓒ.

Mia pauses the automation process (Fig. 4.i, 7.b). She then manually scrapes the text description and downloads the image using the action panel (Fig. 4.j). It's worth noting that during the pause, Mia's web macros are not used for program synthesis and therefore do not affect the current automation program. Mia also adds another image condition using *User Enters*, specifying *"a canyon valley with river"* as another instance of her ideal destination. While the program is paused, Mia scrolls through, evaluates the output panel, and can delete output rows that were added by system mistake or modify/delete conditions during the pause (Fig. 7.d). After she is satisfied with the refined condition set and the current results, Mia clicks "Resume" to continue the automation. The program iterates through the rest of the web page and collects filtered data based on Mia's semantic conditions. After the program ends, Mia stops the web macro recording, and the collected results are exported as a `JSON` file. Mia uses SemanticOn to create a tailored web automation program without writing any code. She also obtains a high-accuracy result set of her ideal travel destinations by continuously working with SemanticOn to clarify and refine conditions and repair errors.

4.2 Design and Implementation

We implemented SemanticOn as a Chrome browser extension, incorporating the core program synthesis engine from an existing system, WebRobot. Primarily, it uses plain JavaScript for the front-end interactions. For semantic condition comprehension, we adopted two off-the-shelf Transformer models.

4.2.1 Step 1: Specify Semantic Conditions. To enable content-based semantic conditions specification (DG1), SemanticOn allows users to add condition criteria to specific content on the target website. Currently, the system supports conditions on text and image content (Fig. 4.c). The user can choose one of the two methods of specification: 1) *User Enters*, where the user composes the condition using natural language, or 2) *System Suggests*, a novel specification technique where the user highlights relevant content for SemanticOn to analyze and provide suggested conditions (Fig. 4.d). Upon selecting the type of condition and the specification method, the user chooses an element on the web page as the basis of the semantic condition.

Inspired by recent prompt-based interactions [33, 50, 79], a prompt displays next to the selected text or image element for *User Enters*, encouraging the user to enter a semantic description of their search criteria (Fig. 6.c). This description, along with the context of the text or image element, is added to the conditional panel on SemanticOn (Fig. 4.f,g,h). In comparison, for *System Suggests*, the user needs to highlight the crucial part of the content to set the condition on.

For images, the user brushes over an area of interest with their mouse (Fig. 6.a) to illustrate. The image is fed through an off-the-shelf pre-trained multi-layer Transformer model that learns to align image-level tags with their corresponding image region features [32]. The model detects objects for the illustrated section and generates a caption for the entire image. For texts, the user highlights relevant phrases or sentences with their mouse (Fig. 6.b). Similarly, the text is processed by an off-the-shelf unsupervised language model where the noun phrases in the input text are first detected and then ranked based on frequency and co-occurrence. The model generates key phrases in the passage highlighted by the user. The user can then pick any object, caption, or phrase tags to add to the condition table (Fig. 4.f,g,h). We used these models through the Microsoft Azure Cognitive Services and Cloud platform.[2] Guided by the design of other systems [17, 31], we also implemented edit, delete, and logical operations for multiple condition specifications, allowing the user to modify, remove, or set the `AND/OR` logic switch on the conditions (Fig. 4.e). They can also apply the condition to one or all of the output columns.

4.2.2 Step 2: Demonstrate Actions and Automate. Once the user sets initial conditions, they can start the demonstration process by clicking "Start Recording" (Fig. 4.a). This initializes the WebRobot

[2]Microsoft Azure, https://tinyurl.com/55puwcf2

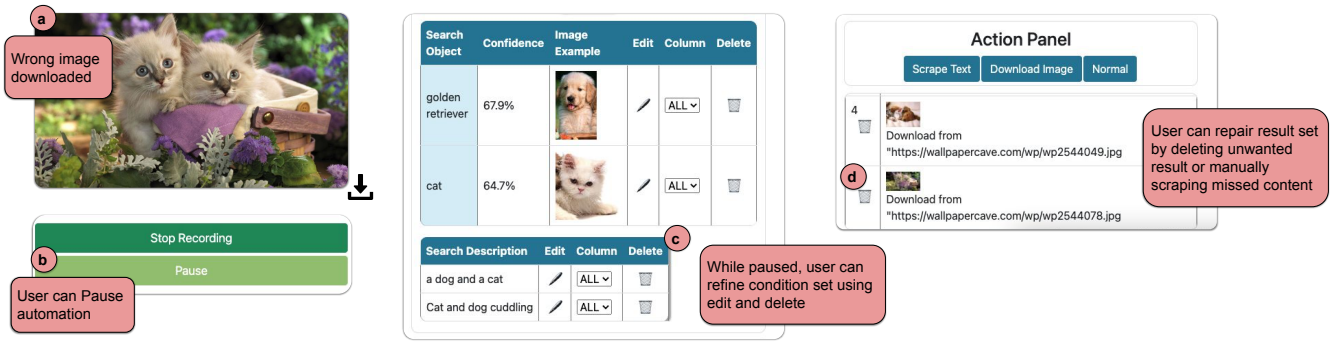


Figure 7: SemanticOn Error Repair Workflow. A user wants to download cute pictures of a cat and a dog cuddling, but they notice a photo of two cats is being scraped by mistake ⓐ. The user pauses the system ⓑ. Then, the user can edit the condition description, delete conditions ⓒ, and delete the incorrectly scraped result ⓓ.

system on the current web page. The user can then select one of the three actions under the action panel: Scrape Text, Download Image, and Normal (Fig. 4.j). The Scrape Text and Download Image actions allow users to click on the desired text or image element and extract the information into the table in output panel. In contrast, the Normal action allows the user to navigate and paginate the web page. For example, the user can use Normal to click into a web page, use Scrape Text to extract a passage, and switch back to Normal to return to the previous page. WebRobot registers this entire sequence of web macros and generates an automation program to repeat it.

After the user demonstrates their intended pattern of actions twice, WebRobot generates an automation program and predicts the next action. SemanticOn also enters a guided *semi-automation* mode. That is, for every data extraction action predicted, the system parses the content through machine learning models. A new image is considered matching with the current conditions if 1) an object specified in the image condition is detected (e.g. user specified a dog, and the new image consists of a human petting a dog), or 2) the new caption yields a similarity score above a certain threshold when compared with captions specified in the image condition. The image caption and text content similarity thresholds were set to 0.5 and 0.4, respectively, based on our benchmark testing. A new text is considered matching if it yields a similarity score above a threshold when compared with specified text conditions. We used a Sentence-Transformer to calculate the semantic similarity between captions and texts [67]. The model utilizes Sentence-BERT (SBERT), a pre-trained network that derives semantically meaningful sentence embeddings that can be compared using cosine-similarity. To avoid the low-score comparison between an entire paragraph and a key phrase, we chunk the text content and conditions into sentences and see if any pair results in a high similarity score.

4.2.3 Step 3: Refine, Repair, and Coordinate. To help users easily refine their intent (DG2) and repair mistakes (DG3), we adopted the *human-in-the-loop* approach [24, 59, 63] and introduced a *semi-automation mode* to ease the transition between manual demonstration and full system automation. First, to repair unexpected results or conditions, SemanticOn offers users a program pause function, which is important and unique in continuous AI systems. In the semi-automation mode, if new text or image content matches with the condition set, the user is notified by a prompt informing them which part of the content was matched. In contrast, if the new content is ambiguous or does not match with the condition set, the user is prompted with a set of suggested conditions (objects and captions for images, key phrases for text) generated by ML model processing (Fig. 5). The user can clarify their intent by selecting suggested conditions to append to the condition set. The new content will be accepted and added to the output results. Alternatively, the user can reject the system's prediction by clicking "Finish", in which case the condition set remains unchanged, and the new content is discarded. The semi-automation mode guides the user through condition specification and refinement. The user might begin with a high-level idea of their search query but can refine and adjust it upon seeing SemanticOn's interpretation of the result. The user can add new conditions to cover unforeseen cases or to modify/delete vague or over-specified conditions that filter results in an unintended way.

After going through predicted content, administering decisions, and refining conditions, the user might feel satisfied with the set of conditions. In that case, the user can choose to hide system prompts (Fig. 4.b) for rejected content and enter full automation. In this mode, SemanticOn continuously executes the predicted web macros and assesses each new piece of content, outputting the information accordingly. This is done by adding a condition evaluation step after executing every macro before extracting the data. However, when the user detects a filtering error, they can use the "Pause Automation" button to reclaim manual control (Fig. 7.b). When the system is paused, the user is free to modify the output result table. For each assessed piece of output, an information icon is displayed at the end of the table cell (Fig. 4.m). Once clicked, it will expand and display the ML processing results for that content. The user can repair system-made errors by deleting rows of ambiguous results accepted by the system but misaligned with the user's mental model (Fig. 7.d). They can also manually use the action panel to add content rejected by the system but matches the user's needs. In addition, the user can learn from the filtering results to add new conditions they omitted or under-specified. Likewise, condition editing and deletion are also available during the pause (Fig. 7.c). This way, the user can feel confident about the automatic selection of results, as they always have the power to reclaim manual control, repair

system mistakes, and clarify their intents. Note that during pause, none of these steps are recorded for program synthesis and do not affect the automation program.

5 SYSTEM EVALUATION

We conducted an in-person user study to evaluate SemanticOn's usability and compare the two methods for condition specifications. This evaluation was guided by the usage evaluation in the HCI toolkit evaluation strategy classification [41].

5.1 Participants

We recruited 10 people (5F5M, mean age 24.3, mean coding experience 4.6 years) at a large public university. Participants are denoted as P1-P10 in subsequent sections. Seven of the participants were graduate students, and three were undergraduates. Five participants have written web automation programs or used commercial tools to some extent. The participants were contacted by email to participate in a study where they would interact with computer software to create web automation programs.

5.2 Study Design

After signing our consent form, each participant first watched a tutorial video of SemanticOn's interface and features. Then participants performed five tasks using SemanticOn. For each task, they were given a task sheet with a semantic conditional intent. Similar to the posted tasks in online forums, the descriptions were intentionally vague, so the user could not copy the instructions verbatim and would need to formalize their own idea of how to specify and refine the conditions. The participants could request the experimenter's assistance at any time during the session. After the participants completed the tasks, we conducted a short interview with them regarding their experience. Additionally, they filled out a short survey with Likert scale and short-answer questions when they exited. The participants were compensated $25 for their time. Each session took 60-75 minutes and was conducted in person on our machine. All sessions were screen- and audio-recorded. Our study is approved by the IRB at our institution.

5.3 Tasks

Through an analysis of online web scraping request posts from iMacros Forum [3], and Stack Overflow [7], we designed five tasks that represent the challenge of creating web automation programs with semantic conditions. The Appendix (Fig 8) shows the details of the user study tasks. To evaluate the usability and utility of the system and compare the effectiveness of two intent specification methods, we designed the tasks with the following goals in mind: 1) the web content to be scraped should be realistic and easily understood by participants, 2) the tasks should extract information based on text and image content, 3) the condition criteria and the corresponding results should contain some ambiguity to allow room for refinement, and 4) the tasks should help SemanticOn to demonstrate the system's full capabilities. To achieve this goal, we adopted and piloted five web scraping requests by real users on online forums, two based on image-heavy websites (pet wallpapers[3], street photography[4]), two with text-heavy content (movie star biographies[5], list of novels[6]), and one task with a variety of both image and text content (beautiful places in the world[7]). Participants needed to extract only images or only text content for the first four tasks. In addition, they were randomly assigned to use *User Enters* and *System Suggests* as their specification method for the first four tasks. We counterbalanced the ordering of image versus text tasks and the assigned specification method to eliminate sequence and learning effects. To standardize task difficulty, participants needed to extract the top 15 text passages and/or images from the website based on a semantic condition. For the final task, participants needed to scrape both images and text in two separate columns and were free to use both condition specification methods. Task 5 served as an exploration task for participants to evaluate and compare the two specification methods and was not constrained by a result set or time. Participants engaged in this task until the end of the study. Therefore, Task 5 was not included in the quantitative analysis for accuracy and duration in the later sections.

We designed the conditions based on the content itself and established a ground truth result set that was compared with the participant's result to measure accuracy. The conditions were designed so that in the ground truth set, the number of results passing and failing the condition across all tasks was approximately the same (31 passes, 29 fails).

5.4 Results

5.4.1 Time and accuracy. The user study recorded 40 scenarios (10 participants x 4 tasks); one instance was discarded from analysis due to a recording issue, resulting in 39 total task completions. Table 1 lists the average time (in minute:second) each participant spent and the accuracy (percentage of correctly included and excluded results) on each task. The overall average duration is 06:10 (image tasks *mean*=05:55, text tasks *mean*=06:25), and the overall average task accuracy is 80.7% (image tasks $mean = 83.0\%$, text tasks $mean = 78.5\%$). We could not identify statistical significance in the difference in accuracy and duration across image and text tasks.

We also analyzed the data comparing usages of *User Enters* versus *System Suggests*. The average duration was 06:22 for *User Enters* tasks and 05:57 for *System Suggests* tasks. In terms of mean accuracy, participants achieved 83.0% for *User Enters* tasks and 78.5% for *System Suggests* tasks. Again, the differences are unable to be identified as statistically significant.

5.4.2 Overall effectiveness in intent specification. In the exit survey, participants rated the ease of use of SemanticOn overall, the ease of use of each specification method, their mental effort, and their trust in the system. Table 2 displays the average score for Likert scale questions on SemanticOn's usability. On a scale of 1 (strongly disagree, very negative) to 7 (strongly agree, very positive), participants believed their specified conditions were displayed in an easily understandable way ($mean = 6.0, SD = 1.1$) and that the

[3]Pet Wallpapers, https://tinyurl.com/47kr3mz6
[4]Street Photographers, https://tinyurl.com/bdppfa48
[5]Movie Stars, https://tinyurl.com/zsnkne5r
[6]Best Romance Novels, https://tinyurl.com/mryjs47h
[7]Vacation Destinations, https://tinyurl.com/4peucx3p

Task index	Completion time (mean, SD in mm:ss)	Accuracy
1	05:06 (01:43)	85.3%
2	06:58 (02:28)	68.9%
3	06:44 (01:50)	81.3%
4	05:56 (01:50)	86.0%

Table 1: Average time spent on each task. Tasks 1 and 3 are image tasks; tasks 2 and 4 are text tasks.

system-generated prompts in semi-automation mode helped them refine their intent ($mean = 5.6, SD = 0.97$). However, participants had polarized opinions on the statement *"it was easy to coordinate (claim controls, pause/resume, show/hide prompts) with SemanticOn to repair under- or over-specifications,"* with 5 participants agreeing and 3 participants disagreeing ($mean = 4.7, SD = 1.7$). Overall, the users widely accepted the semi-automation workflow towards automation and the opportunity to adjust conditions later on by pausing. 9 out of 10 participants added refinement conditions for Tasks 1-4, and all participants utilized the system prompts in semi-automation mode to refine their conditions in Task 5.

5.4.3 Error-handling Analysis. As per DG3, SemanticOn offers error-handling techniques for users during full automation. We measured participants' usage of three error repair methods: condition deletion, result deletion, and manual result addition. Most users opted to use the pause and repair functionalities, as 7 participants deleted their specified conditions due to inaccuracies, 6 participants removed undesired output to repair system mistakes, and 2 participants manually added desired contents missed by the filter. The utilization of the error-handling features varied based on the task type. For condition deletion, we found 16 instances across image tasks and 0 instances for text tasks. Similarly, there were 11 instances of result deletion for image tasks and 2 for text tasks. As for manual result addition, we found 3 instances for image tasks and none for text tasks. This difference can be related to the content processing speeds for visual versus textual materials. The condition specification method also plays a role in the usage of error-handling features. We observed 4 instances of condition deletions for *User Enters* and 12 for *System Suggests*. In addition, we found 4 result deletion instances for *User Enters* and 9 for *System Suggests*. This can be attributed to the difference in the required effort for each workflow. We expand on these varieties in our Discussion.

5.4.4 Comparison between User Enters and System Suggests. Regarding the difference between the two specification methods, participants found both experiences comparable but thought *System Suggest* was easier to use. P10 said, *"[My preference] really depends on the task, but I really liked the System Suggests mode, especially for words it was really easy to use and accurate."* When assessing the overall experience to specify conditions, participants rated *User Enters* slightly higher ($mean = 5.3, SD = 1.1$) than *System Suggests* ($mean = 5.1, SD = 1.2$). But when evaluating the ease of use for each interaction, users rated entering their own natural language descriptions lower ($mean = 5.1, SD = 1.1$) than brushing, highlighting, or choosing system-suggested conditions ($mean = 5.7, SD = 0.82$). We were unable to identify statistical significance in these differences. However, when measuring the average number of initial conditions set before the demonstration, participants specified an average of 2.6 conditions for *User Enters* and 4.1 conditions for *System Suggests* ($p < 0.05$). This points to a difference in ease-of-use and condition-generation capability between the two methods, which is further explored in Discussion.

Question	Likert Scale (Mean, SD)
User Enters Experience	5.3 (1.0)
System Suggests Experience	5.1 (1.2)
User Enters Ease of Use	5.1 (1.0)
System Suggests Ease of Use	5.7 (0.8)
Coordination to Refine Specifications	4.7 (1.6)
Usefulness of Generated Prompts	5.6 (0.9)
Conditions Displayed Clearly	6.0 (1.0)
Trust in Full Automation	4.5 (1.5)
Success in Completing Task	5.0 (0.9)
Mental Demand	4.6 (1.7)
Effort to Achieve Results	4.9 (1.3)
Feelings of Insecurity and Stress	5.2 (1.7)
Feelings of Being Hurried or Rushed	4.9 (1.6)

Table 2: Survey Responses. For section one (top), 1 is very negative, and 7 is very positive. For section two (bottom), 1 is very high mental demand, effort, insecurity and stress, and feelings of being hurried and rushed.

5.4.5 Mental effort and AI system trust. Participants also reported on their mental effort in completing the tasks and their trust in the AI system in the survey. Participants generally reported medium to high mental effort using SemanticOn, especially at the start of the study when they had to familiarize themselves with the interface and remember the workflow (P2, P5, P6, P7). On a scale of 1 (very demanding, very hard, not successful) to 7 (not demanding, not hard at all, very successful), participants experienced medium mental demand ($mean = 4.6, SD = 1.8$) and medium effort ($mean = 4.9, SD = 1.4$). They believed they were relatively successful in accomplishing the tasks ($mean = 5.0, SD = 0.94$).

We also found that the participants exhibited a medium level of trust ($mean = 4.5, SD = 1.6$) towards the AI system's prediction in full automation. Six users were comfortable entering full automation mode for at least one task. However, P3 and P7 expressed very low trust in the automated prediction (both rated 2 out of 7 in survey). During the interview, some participants (P5, P9) also commented that their trust in the system depended on the content type and the number of refinements required for each task. For example, P5 mentioned they *"trust this picture task a bit more...[because] text has more group[ing]s and potential variations"* when they used 11 text refinements and only 1 image refinement. We analyze the effect of content type on users' perception of the task and our system in the Discussion.

6 DISCUSSION

Based on the evaluation, we present analysis of the human-AI collaboration techniques and the role of each agent in SemanticOn. We also discuss the implications of adding similarity-based machine learning models in symbolic PBD systems. In addition, we report findings on users' trade-offs in using *User Enters* and *System Suggests*, as well as the intent specification effectiveness and system limitations to provide insights on future designs.

6.1 Human-AI Collaboration

The user workflow of SemanticOn consists of both a human and an artificial intelligence agent; they exchange control in different parts of the interaction and collaborate to build a complete and accurate web automation program. In SemanticOn, we utilize machine learning models' classification efficiency to rapidly process texts and images, summarize the content, and provide suggested semantic filters to the user. This saves the user's mental effort significantly, as it is no longer the human's role to parse content and identify an inclusion or exclusion decision. However, it also introduces a new source of error to web automation, as similarity-based models can mislabel inputs and result in false positive and false negative data. This is why human is always involved in the continuous human-AI collaboration in SemanticOn. After a couple of demonstrations, the user is prompted with potential semantic conditions for each processed content in the semi-automation mode. Additionally, even when the user feels satisfied with the set of conditions, the program can still be paused at any time if a mistake is spotted or the user wants to edit the conditions.

From the evaluation of this interaction paradigm, most participants (7/10) reported that the workflow's path from demonstration to guided semi-automation to full automation was intuitive and useful. The semi-automation phase allowed users to calibrate conditions with SemanticOn and better understand the AI system's interpretation of the conditions. This is important as users might not have had a well-defined semantic condition at the program's start. P4 believed that the *"human brain still needs to think about [how to describe the condition]...and link to all the keywords. These words might not specifically come to mind to the human."* In this case, the initial conditions may not include all of the user's needs, resulting in low filter accuracy. The semi-automation mode provides aid for this. For example, P3 mentioned that even if the results were not accurate at the start, *"I can interactively try to make [automated prediction] better as it goes."* P4 also mentioned that *"the transitional period to trust the process is good, saves a lot of fuss."*

However, the steps to refine conditions and results through system prompts and automation pauses could require too much user effort and become time-consuming. P5 commented that the entire user experience was *"a little slow...I think the fine-tuning part is slow. Once you start [full automation], then it's fine."* P2 reported that *"it took me multiple times to figure out the sequence of the buttons."* When they spotted system mistakes, P1 and P4 decided not to pause and amend the errors because it took too much effort. They believed the misclassified content was not detrimental to the automation task, as there was no requirement for accuracy. While supplying users with multiple refinement tools enhances robustness, the system must be cautious not to overload users with interaction techniques and interruptions.

6.2 Similarity-based Model in PBD Systems

A main contribution of SemanticOn is introducing a new interaction paradigm for users to continuously add/refine semantic conditions in programming-by-demonstration (PBD) systems. In particular, it enables users to express intent via similarity-based machine learning models. This specification is at a higher abstraction level than pure symbolic systems based on DOM structure. Here, we discuss the advantages and disadvantages of expressing intent using similarity-based statistical models versus pure symbolic systems, the implications of adding statistical models in PBD systems, and the generalizability and viability of SemanticOn against errors.

Traditional symbolic automation systems like WebRobot are robust at understanding specific user actions and generalizing them into repeatable steps in a synthesized program. This is achieved by iterating over the elements in the DOM structure of the websites and generalizing the pattern. Meanwhile, statistical machine learning models are useful at parsing high-level user intent and matching unstructured, but semantically similar concepts, for example extracting key entities from a natural language input. However, both systems alone have their limitations. Similarity-based models lack reasoning capability. The machine learning models we employ classify the inclusion and exclusion of content based on semantic similarity, but they cannot derive further actions based on specific criteria. For example, the models alone could not construct a program that repeatedly takes in an image URL, searches it, and downloads the image based on semantic conditions. It is efficient in the last filtering step but ineffective when structural elements need to be generalized. In contrast, symbolic models that reason about a set of user instructions over a certain website structure might fail to generalize if the query is unstructured or outside the scope of the task domain, while similarity models can cover a wider range of input content that are not constrained by the structure. SemanticOn combines these two techniques in an effective way that seals the gap between users' semantic intent and web automation.

From a practical standpoint, training an intelligent statistical model for the tasks we cover in SemanticOn's user evaluation would be effortful. Systems like Calendar.help [22] require many expert heuristics and even human workers to automate a very specific task in event scheduling across different parties. In SemanticOn, users compromise some effort by doing two rounds of demonstrations of their desired web actions. Still, the automation tasks can be generalized to a diverse set of data on many different websites. Introducing similarity-based models in symbolic systems does present a new source of error as the models can filter content incorrectly, in addition to program synthesis errors in pure-symbolic systems. To amend this, SemanticOn provides continuous condition refinement and output edit/delete options so that users can repair classification errors easily and yield more accurate results.

Our work does not rely on a specific program synthesizer to generate a web automation program or a specific machine learning model to filter content semantically. The novelty of SemanticOn is the set of condition specification, refinement, and error-correction interactions, which can be generalized to other PBD systems.

6.3 User Enters and System Suggests

One emphasis of this project is to compare the usability of the two semantic condition specification methods. To summarize the findings, participants shifted their preference between *System Suggests* and *User Enters* based on two factors: perceived effort and the type of content presented. When participants expressed their preferences and the rationale behind using each specification method, we also discovered a trade-off between their sense of control and the perceived effort.

6.3.1 Perceived effort based on content types. Perceived effort plays an important role in user preference. When it was easy to summarize the target content on which users needed to set conditions, participants preferred to use *User Enters* to express their intent instead of selecting *System Suggests*, which could be too broad or too specific. P3 remarked that *"[User Enters makes] you feel like [you have] more control because you are looking for a specific thing."*

On the contrary, when the target content was information-heavy and hard to encapsulate in a short natural language description, participants preferred to use *System Suggests* to highlight the details and let SemanticOn present potential conditions. P6 commented that *"I do like the possibility of being able to say this is what I see in the image [based on System Suggests]... and [highlight] a part of the image that I think is important."*

When participants were free to use both *User Enters* and *System Suggests* (i.e. Task5 in Appendix Fig 8), we found that text content inherently took more effort to process, and 7 out of 10 participants opted to use *System Suggests* to avoid reading a large block of text. In contrast, despite containing a variety of objects and entity relationships, the corresponding image content could be encoded and summarized more quickly, and 9 out of 10 participants chose to specify their intents with *User Enters*.

This preference based on content type persisted during condition refinement in semi-automation and full automation modes. Participants expressed difficulty in parsing textual information rapidly, as they could not distinguish whether the system made the correct selection. P3 said that *"text task in general is much more difficult,"* and P5 commented, *"when it comes to text task, I'm a little lazy to read through all this... so system can just get me relevant keywords."* Our observations support this as users were far more likely to repair errors for image tasks than text tasks by deleting inaccurate conditions and incorrect results. Participants can quickly identify whether an image fits the condition set, but they can not parse through a block of text as the automation quickly processes each row of content. Based on this finding, future designs should account for the processing effort for text content and provide users more time and aid in understanding and summarizing the information.

6.3.2 Sense of Control and mental effort trade-off. Another key factor for specification preference was the sense of control. Three participants (P3, P7, P10) listed a better sense of control as to why they preferred *User Enters*, where they could tailor the condition using their terms. In addition, in cases where *System Suggests* suggestions were inaccurate (e.g., failing to detect relevant text entities or generate image captions at the appropriate granularity), some participants (P2, P4, P6, P10) switched to *User Enters*.

However, there was a trade-off between the sense of control and the required effort. Participants rated a lower ease-of-use score on average for *User Enters* (5.1 versus 5.7 for *System Suggests)*, despite it having fewer interaction steps. Three participants (P4, P5, P10) pointed out that they preferred *System Suggests* due to the convenience and ease of mental effort. *System Suggests* is also advantageous in reporting details that escaped the user's attention. For example, P4 pointed out that they preferred *System Suggests* mode because it sometimes captured things that the user failed to recognize or think of. Additionally, P7 mentioned that they *"quite like System Suggests more, [because it is] more systematic and highlights specific things."* Participants could avoid cognitive overload by handing the processing labor to SemanticOn through *System Suggests*, but they were limited to the generated set of conditions for each item. This trade-off also affects participants' usage of error handling features during automation. With the increased cognitive load in *User Enters*, participants are much less likely to pause and repair their mistakes than when using *System Suggests*. However, this could be explained by participants adopting less accurate prompts from *System Suggests*, which leads to more errors overall for users to fix. Future work could potentially create a combined approach where users can edit system-suggested conditions, yielding a sense of control and avoiding heavy user efforts.

6.4 System Effectiveness

Most participants (8/10) agreed that SemanticOn is an effective tool for web automation and that the condition specification aspect is useful. Compared with manual effort and traditional data scraping methods such as writing automation scripts, participants found SemanticOn's no-code solution novel and easier to use. P5 mentioned that in manual scraping scenarios, *"Control-F would only get you so far"* and that the system is *"much easier than writing your own script...[and] more functionalit[ies]"* In addition, P7 thought this tool *"would be super helpful for someone not comfortable with code. [It provides] low effort but maximum reward."*

More than half of the participants (6/10) believed that SemanticOn could be applied to realistic tasks. However, some focused on the conditional selection aspect, and others emphasized on the complexity of the task. For example, P2 commented that the system is *"very good for selective tasks [when you] only want a few images from a lot of them."* Similarly, P5 suggested that SemanticOn could be used for websites without robust filtering, search engine, or categories because *"it could easily make any website sort-able."* On the other hand, P1 enjoyed using this tool on Task 5 with both image and text conditions combined and proposed that *"this type of tool would benefit... power users [for tasks] like creating a dataset...[and] dealing with complex things."* These reports provide evidence for the need of no-code solutions in conditional web automation tasks.

6.5 System Limitations

There are several limitations to our system. First is the efficiency of coordination. As mentioned, some participants found the refinement process complicated and slow and were reluctant to utilize the functionality to increase system accuracy. In future designs,

the researchers could simplify the user interface and the interaction steps and condense the system prompt information to reduce cognitive load.

Another limitation is the relatively low accuracy of the machine learning models it relies on to classify text and image content. Partially, this relates to a design decision we made when building SemanticOn. We first used two state-of-the-art models—OFA [74], which uses a sequence-to-sequence learning framework, and ImageAI, which uses a convolutional neural network [56], for image captioning and object detection, and we were able to achieve high accuracy. However, the image processing was computation-heavy and took more than 30 seconds per image on an off-the-shelf laptop. To reduce the gulf of evaluation [62], we switched to the Microsoft Azure computer vision model [32] for near-instant cloud processing but with lower accuracy and detail level for some content domains. Similarly, for key phrase extraction, we employed the Microsoft Azure language model, sending text contents via REST API calls for near-instant processing. But the extracted key phrases often have a low level of abstraction (i.e. extracting entities and nouns in a sentence without indication of interaction) and could not provide high-level descriptions like those used for image captions.

Additionally, SemanticOn builds upon an existing symbolic program synthesis system, WebRobot, which has its own constraints and requirements. And the introduction of similarity-based machine learning models also presents ambiguity to the source of error between incorrect program synthesis steps and result misclassifications. The user might be confused when encountering unexpected behavior, as it could be a product of a user demonstration error or the system's misunderstanding of user intents, which requires a different workflow to repair. However, in our user study, we emphasize repairing errors from machine learning model mislabelling as that is the novel part of SemanticOn.

6.6 Future Work

We summarized three points of feedback that are relevant for future works. First, we found that participants spent a longer time encoding and processing texts than images. This is coherent with previous studies on human capability in consuming different types of information [53]. But, this finding is significant in full automation, when the system rapidly loops through web page content, giving users little time to identify potential selection errors. Future works could enable multi-modal interaction (e.g., voice) to reduce the effort of information processing [20]. Additionally, one can provide more indicators for users to quickly recognize the information represented in text elements (e.g. highlighting matching key phrases) and decide whether the content should be included based on the semantic condition.

Second, participants separately favored one of the two condition specification methods, *User Enters* and *System Suggests*, depending on the content type and the amount of information they needed to process. Similar to the neurosymbolic program synthesis approach [16, 65], future works could create a unified technique where users could benefit from the instant and comprehensive results from machine learning models while preserving users' power to specify conditions on their own terms.

Lastly, future designs could further improve the system usability by reducing the mental effort. The dual process of collaborating with an AI on both demonstrating web macros to synthesize an automation program and specifying semantic conditions to filter web content requires a decent amount of mental effort, especially for users who are less familiar with programming or web scraping. Therefore, future designs should alleviate users' cognitive load with a more minimalist UI and more user guidance.

7 CONCLUSION

In this work, we designed and developed SemanticOn, a collaborative system that allows users to specify and refine visual and textual conditions through user-entered descriptions and system-suggested prompts in a web automation program. In a system evaluation, we found that participants can effectively use SemanticOn to create conditional filters and refine them via continuous human-AI collaboration, collecting selective web content with high accuracy. Participants' feedback also suggested that a guided semi-automation mode, where users authorize system predictions, helped clarify user intents. We also found a trade-off between *User Enters* and *System Suggests* regarding user effort and the sense of control. Our work can point directions to future system and interaction designs for user-intent specification and refinement in a continuous human-AI collaboration setting.

ACKNOWLEDGMENTS

We thank all our participants and reviewers. This research was supported in part by the National Sciences and Engineering Research Council of Canada (NSERC) under Grant IRCPJ 545100 - 18.

REFERENCES

[1] 2022. Beautiful Soup. http://www.crummy.com/software/BeautifulSoup/ Accessed: April, 2022.
[2] 2022. iMacro. https://www.progress.com/imacros Accessed: April, 2022.
[3] 2022. iMacro Forum. https://forum.imacros.net/ Accessed: April, 2022.
[4] 2022. Puppeteer. https://pptr.dev/ Accessed: April, 2022.
[5] 2022. Scrapy. https://scrapy.org/ Accessed: April, 2022.
[6] 2022. Selenium. https://www.selenium.dev/ Accessed: April, 2022.
[7] 2022. Stack Overflow. https://stackoverflow.com/search?q=%5Bselenium%5D+semantic Accessed: April, 2022.
[8] 2022. UiPath. https://www.uipath.com/ Accessed: April, 2022.
[9] James Allen, Nathanael Chambers, George Ferguson, Lucian Galescu, Hyuckchul Jung, Mary Swift, and William Taysom. 2007. Plow: A collaborative task learning agent. In *AAAI*, Vol. 7. 1514–1519.
[10] Saleema Amershi, James Fogarty, and Daniel Weld. 2012. Regroup: Interactive machine learning for on-demand group creation in social networks. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 21–30.
[11] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732* (2021).
[12] A Blackwell. 2000. Your Wish is My Command: Giving Users the Power to Instruct their Software, chapter SWYN: a visual representation for regular expressions. *M. Kaufmann* (2000), 245–270.
[13] Alan F Blackwell. 2001. SWYN: A visual representation for regular expressions. In *Your wish is my command*. Elsevier, 245–XIII.
[14] Carrie J Cai, Emily Reif, Narayan Hegde, Jason Hipp, Been Kim, Daniel Smilkov, Martin Wattenberg, Fernanda Viegas, Greg S Corrado, Martin C Stumpe, et al. 2019. Human-centered tools for coping with imperfect algorithms during medical decision-making. In *Proceedings of the 2019 chi conference on human factors in computing systems*. 1–14.
[15] Sarah E Chasins, Maria Mueller, and Rastislav Bodik. 2018. Rousillon: Scraping Distributed Hierarchical Web Data. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 963–975.
[16] Qiaochu Chen, Aaron Lamoreaux, Xinyu Wang, Greg Durrett, Osbert Bastani, and Isil Dillig. 2021. Web question answering with neurosymbolic program

synthesis. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 328–343.

[17] Yan Chen and Tovi Grossman. 2021. Umitation: Retargeting UI Behavior Examples for Website Design. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 922–935.

[18] Yan Chen, Jaylin Herskovitz, Walter S Lasecki, and Steve Oney. 2020. Bashon: A Hybrid Crowd-Machine Workflow for Shell Command Synthesis. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–8.

[19] Yan Chen, Sang Won Lee, and Steve Oney. 2021. CoCapture: Effectively Communicating UI Behaviors on Existing Websites by Demonstrating and Remixing. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*.

[20] Yan Chen, Sang Won Lee, Yin Xie, YiWei Yang, Walter S Lasecki, and Steve Oney. 2017. Codeon: On-demand software development assistance. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 6220–6231.

[21] Yan Chen, Maulishree Pandey, Jean Y Song, Walter S Lasecki, and Steve Oney. 2020. Improving crowd-supported gui testing with structural guidance. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.

[22] Justin Cranshaw, Emad Elwany, Todd Newman, Rafal Kocielnik, Bowen Yu, Sandeep Soni, Jaime Teevan, and Andrés Monroy-Hernández. 2017. Calendar.help. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/3025453.3025780

[23] Biplab Deka, Zifeng Huang, Chad Franzen, Jeffrey Nichols, Yang Li, and Ranjitha Kumar. 2017. Zipt: Zero-integration performance testing of mobile app designs. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. 727–736.

[24] Rui Dong, Zhicheng Huang, Ian Iong Lam, Yan Chen, and Xinyu Wang. 2022. WebRobot: Web Robotic Process Automation using Interactive Programming-by-Demonstration. *arXiv preprint arXiv:2203.09993* (2022).

[25] Ian Drosos, Titus Barik, Philip J Guo, Robert DeLine, and Sumit Gulwani. 2020. Wrex: A unified programming-by-example interaction for synthesizing readable code for data scientists. In *Proceedings of the 2020 CHI conference on human factors in computing systems*. 1–12.

[26] Kasra Ferdowsifard, Allen Ordookhanians, Hila Peleg, Sorin Lerner, and Nadia Polikarpova. 2020. Small-step live programming by example. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 614–626.

[27] Michael H Fischer, Giovanni Campagna, Euirim Choi, and Monica S Lam. 2021. DIY assistant: a multi-modal end-user programmable virtual assistant. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 312–327.

[28] James Fogarty, Desney Tan, Ashish Kapoor, and Simon Winder. 2008. CueFlik: interactive concept learning in image search. In *Proceedings of the sigchi conference on human factors in computing systems*. 29–38.

[29] Philip J Guo, Sean Kandel, Joseph M Hellerstein, and Jeffrey Heer. 2011. Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. 65–74.

[30] William R Harris and Sumit Gulwani. 2011. Spreadsheet table transformations from examples. *ACM SIGPLAN Notices* 46, 6 (2011), 317–328.

[31] Björn Hartmann, Leslie Wu, Kevin Collins, and Scott R Klemmer. 2007. Programming by a sample: rapidly creating web applications with d. mix. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*. 241–250.

[32] Xiaowei Hu, Xi Yin, Kevin Lin, Lijuan Wang, Lei Zhang, Jianfeng Gao, and Zicheng Liu. 2020. VIVO: Visual Vocabulary Pre-Training for Novel Object Captioning. *arXiv preprint arXiv:2009.13682* (2020).

[33] Ellen Jiang, Edwin Toh, Alejandra Molina, Aaron Donsbach, Carrie J Cai, and Michael Terry. 2021. Genline and genform: Two tools for interacting with generative language models in a code editor. In *The Adjunct Publication of the 34th Annual ACM Symposium on User Interface Software and Technology*. 145–147.

[34] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the sigchi conference on human factors in computing systems*. 3363–3372.

[35] Rebecca Krosnick and Steve Oney. 2021. Understanding the Challenges and Needs of Programmers Writing Web Automation Scripts. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–9.

[36] David Kurlander, Allen Cypher, and Daniel Conrad Halbert. 1993. *Watch what I do: programming by demonstration*. MIT press.

[37] David Kurlander and Steven Feiner. 1992. A history-based macro by example system. In *Proceedings of the 5th annual ACM symposium on User interface software and technology*. 99–106.

[38] Tessa Lau, Steven A Wolfman, Pedro Domingos, and Daniel S Weld. 2003. Programming by demonstration using version space algebra. *Machine Learning* 53, 1 (2003), 111–156.

[39] Tessa A Lau, Pedro M Domingos, and Daniel S Weld. 2000. Version Space Algebra and its Application to Programming by Demonstration.. In *ICML*. Citeseer, 527–534.

[40] Vu Le and Sumit Gulwani. 2014. Flashextract: A framework for data extraction by examples. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 542–553.

[41] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. 2018. Evaluation strategies for HCI toolkit research. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–17.

[42] Gilly Leshed, Eben M Haber, Tara Matthews, and Tessa Lau. 2008. CoScripter: automating & sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1719–1728.

[43] Ian Li, Jeffrey Nichols, Tessa Lau, Clemens Drews, and Allen Cypher. 2010. Here's what I did: Sharing and reusing web activity with ActionShot. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 723–732.

[44] Toby Jia-Jun Li, Amos Azaria, and Brad A Myers. 2017. SUGILITE: creating multimodal smartphone automation by demonstration. In *Proceedings of the 2017 CHI conference on human factors in computing systems*. 6038–6049.

[45] Toby Jia-Jun Li, Igor Labutov, Xiaohan Nancy Li, Xiaoyi Zhang, Wenze Shi, Wanling Ding, Tom M. Mitchell, and Brad A. Myers. 2018. APPINITE: A Multi-Modal Interface for Specifying Data Descriptions in Programming by Demonstration Using Natural Language Instructions. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 105–114. https://doi.org/10.1109/VLHCC.2018.8506506

[46] Toby Jia-Jun Li, Marissa Radensky, Justin Jia, Kirielle Singarajah, Tom M Mitchell, and Brad A Myers. 2019. Pumice: A multi-modal agent that learns concepts and conditionals from natural language and demonstrations. In *Proceedings of the 32nd annual ACM symposium on user interface software and technology*. 577–589.

[47] Henry Lieberman. 1994. A user interface for knowledge acquisition from video. In *AAAI*. Citeseer, 527–534.

[48] James Lin, Jeffrey Wong, Jeffrey Nichols, Allen Cypher, and Tessa A Lau. 2009. End-user programming of mashups with vegemite. In *Proceedings of the 14th international conference on Intelligent user interfaces*. 97–106.

[49] Greg Little, Tessa A Lau, Allen Cypher, James Lin, Eben M Haber, and Eser Kandogan. 2007. Koala: capture, share, automate, personalize business processes on the web. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 943–946.

[50] Vivian Liu and Lydia Chilton. 2022. Design Guidelines for Prompt Engineering Text-to-Image Generative Models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*.

[51] David L Maulsby, Ian H Witten, and Kenneth A Kittlitz. 1989. Metamouse: Specifying graphical procedures by example. *ACM SIGGRAPH Computer Graphics* 23, 3 (1989), 127–136.

[52] Mikaël Mayer, Gustavo Soares, Maxim Grechkin, Vu Le, Mark Marron, Oleksandr Polozov, Rishabh Singh, Benjamin Zorn, and Sumit Gulwani. 2015. User interaction models for disambiguation in programming by example. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. 291–301.

[53] Richard Mayer and Joan Gallini. 1990. When Is an Illustration Worth Ten Thousand Words? *Journal of Educational Psychology* 82 (12 1990), 715–726. https://doi.org/10.1037/0022-0663.82.4.715

[54] Dan H Mo and Ian H Witten. 1992. Learning text editing tasks from examples: a procedural approach. *Behaviour & Information Technology* 11, 1 (1992), 32–45.

[55] Francesmary Modugno and Brad A Myers. 1994. *Pursuit: Visual programming in a visual domain*. Technical Report. CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE.

[56] Moses and John Olafenwa. 2018–. ImageAI, an open source python library built to empower developers to build applications and systems with self-contained Computer Vision capabilities. https://github.com/OlafenwaMoses/ImageAI

[57] Brad A Myers, Dario A Giuse, Roger B Dannenberg, Brad Vander Zanden, David S Kosbie, Edward Pervin, Andrew Mickish, and Philippe Marchal. 1995. GARNET comprehensive support for graphical, highly interactive user interfaces. In *Readings in Human–Computer Interaction*. Elsevier, 357–371.

[58] Aaditya Naik, Jonathan Mendelson, Nathaniel Sands, Yuepeng Wang, Mayur Naik, and Mukund Raghothaman. 2021. Sporq: An Interactive Environment for Exploring Code using Query-by-Example. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 84–99.

[59] Julie L Newcomb and Rastislav Bodik. 2019. Using human-in-the-loop synthesis to author functional reactive programs. *arXiv preprint arXiv:1909.11206* (2019).

[60] Wode Ni, Joshua Sunshine, Vu Le, Sumit Gulwani, and Titus Barik. 2021. reCode: A Lightweight Find-and-Replace Interaction in the IDE for Transforming Code by Example. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 258–269.

[61] Jeffrey Nichols and Tessa Lau. 2008. Mobilization by demonstration: using traces to re-author existing web sites. In *Proceedings of the 13th international conference on Intelligent user interfaces*. 149–158.

[62] Don Norman. 2013. *The design of everyday things: Revised and expanded edition*. Basic books.

[63] Besmira Nushi, Ece Kamar, Eric Horvitz, and Donald Kossmann. 2017. On human intellect and machine failures: Troubleshooting integrative machine learning systems. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[64] Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. 2015. Designscape: Design with interactive layout suggestions. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. 1221–1224.

[65] Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. 2016. Neuro-symbolic program synthesis. *arXiv preprint arXiv:1611.01855* (2016).

[66] Hila Peleg and Nadia Polikarpova. 2020. Perfect is the enemy of good: Best-effort program synthesis. *Leibniz international proceedings in informatics* 166 (2020).

[67] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[68] Alborz Rezazadeh Sereshkeh, Gary Leung, Krish Perumal, Caleb Phillips, Minfan Zhang, Afsaneh Fazly, and Iqbal Mohomed. 2020. VASTA: a vision and language-assisted smartphone task automation system. In *Proceedings of the 25th international conference on intelligent user interfaces*. 22–32.

[69] Atsushi Sugiura and Yoshiyuki Koseki. 1998. Internet scrapbook: automating web browsing tasks by demonstration. In *Proceedings of the 11th annual ACM symposium on User interface software and technology*. 9–18.

[70] Amanda Swearngin, Chenglong Wang, Alannah Oleson, James Fogarty, and Amy J Ko. 2020. Scout: Rapid Exploration of Interface Layout Alternatives through High-Level Design Constraints. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.

[71] Priyan Vaithilingam and Philip J Guo. 2019. Bespoke: Interactively synthesizing custom GUIs from command-line applications by demonstration. In *Proceedings of the 32nd annual ACM symposium on user interface software and technology*. 563–576.

[72] Chenglong Wang, Alvin Cheung, and Rastislav Bodik. 2017. Interactive query synthesis from input-output examples. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1631–1634.

[73] Chenglong Wang, Yu Feng, Rastislav Bodik, Isil Dillig, Alvin Cheung, and Amy J Ko. 2021. Falx: Synthesis-powered visualization authoring. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15.

[74] Peng Wang, An Yang, Rui Men, Junyang Lin, Shuai Bai, Zhikang Li, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. 2022. Unifying Architectures, Tasks, and Modalities Through a Simple Sequence-to-Sequence Learning Framework. *CoRR* abs/2202.03052 (2022). arXiv:2202.03052 https://arxiv.org/abs/2202.03052

[75] Andrew J Werth and Brad A Myers. 1993. Tourmaline (abstract) macrostyles by example. In *Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems*. 532.

[76] Tom Yeh, Tsung-Hsiang Chang, and Robert C Miller. 2009. Sikuli: using GUI screenshots for search and automation. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*. 183–192.

[77] Tianyi Zhang, Zhiyang Chen, Yuanli Zhu, Priyan Vaithilingam, Xinyu Wang, and Elena L. Glassman. 2021. Interpretable Program Synthesis. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 105, 16 pages. https://doi.org/10.1145/3411764.3445646

[78] Tianyi Zhang, London Lowmanstone, Xinyu Wang, and Elena L Glassman. 2020. Interactive Program Synthesis by Augmented Examples. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 627–648.

[79] Zheng Zhang, Zheng Xu, Yanhao Wang, Bingsheng Yao, Daniel Ritchie, Tongshuang Wu, Mo Yu, Dakuo Wang, and Toby Jia-Jun Li. 2022. StoryBuddy: A Human-AI Collaborative Agent for Parent-Child Interactive Storytelling with Flexible Parent Involvement. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*.

[80] Nanxuan Zhao, Nam Wook Kim, Laura Mariah Herman, Hanspeter Pfister, Rynson WH Lau, Jose Echevarria, and Zoya Bylinskii. 2020. Iconate: Automatic compound icon generation and ideation. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.

A APPENDIX

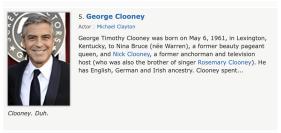
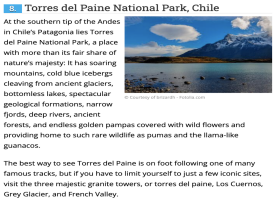
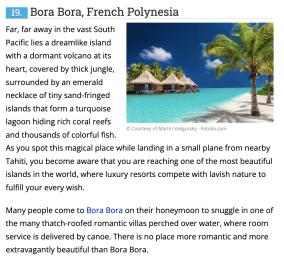
#	Task Description	Condition Type	Inclusion Example	Exclusion Example
1	Download pictures with a dog and a cat interacting	Image only	 Dog and cat interacting	 Cat only, no interactions
2	Scrape book descriptions that highlight a female protagonist's story	Text only	 Heroine perspective	 No female perspective
3	Download photos if it contains multiple people interacting	Image only	 Two people interacting	 No interactions
4	Scrape movie star biographies that writes about their acting	Text only	 Biography includes acting	 No acting in biography
5	Scrape the description and download the image if: 1) Text mentions outdoor activities and 2) Image contains mountain and water	Text and image	 Image has mountain and water and description mentions outdoor activities	 No outdoor activity description, no mountains

Figure 8: Task descriptions with inclusion and exclusion examples