

# DiLogics: Creating Web Automation Programs With Diverse Logics

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

Jim Yang  
jima.yang@mail.utoronto.ca  
University of Toronto

Angel Yuan  
angel.yuan@mail.utoronto.ca  
University of Toronto

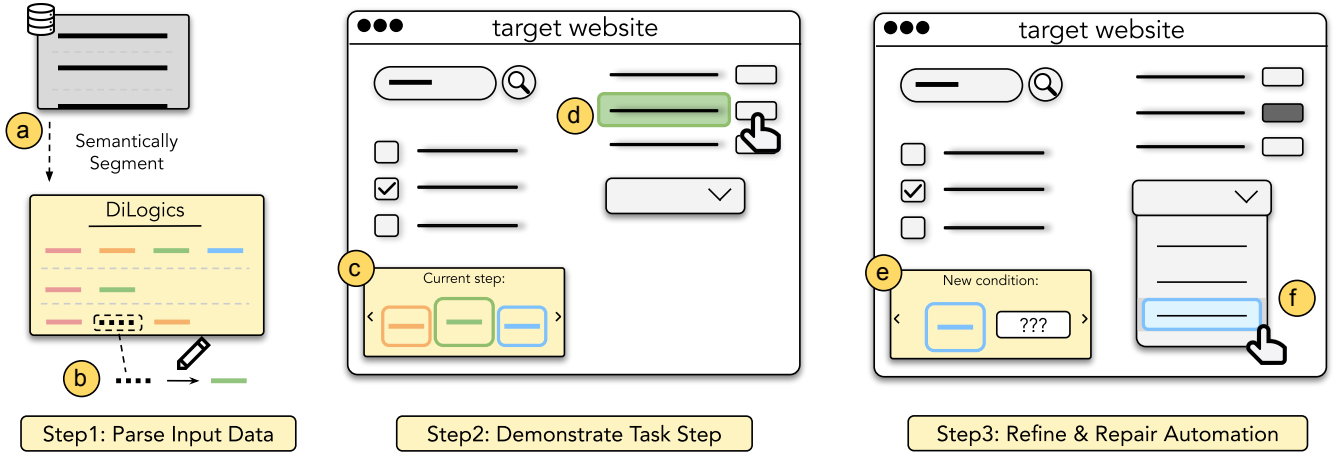
Minyi Ma  
minyi.ma@mail.utoronto.ca  
University of Toronto

Rui Dong  
ruidong@umich.edu  
University of Michigan

Xinyu Wang  
xwangsd@umich.edu  
University of Michigan

Yan Chen  
ych@vt.edu  
Virginia Tech

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



**Figure 1: The workflow of DiLogics.** (Step 1) To create web automation programs for data entry tasks, users first upload an input file, which is semantically segmented into the steps representing different task specifications (a). Users can edit inaccurately segmented steps (b). (Step 2) Users then follow a carousel of current steps (c) and demonstrate the corresponding UI actions to fulfill each specification. DiLogics highlights semantically relevant web page elements, guiding users to perform demonstrations (d). (Step 3) After two iterations of demonstrations, DiLogics learns the mappings between different steps and actions and automates the remaining task steps, generalizing GUI actions based on the specification’s semantic meaning. Users can refine the program logics (e) at any stage of execution by editing the steps or adding new demonstrations (f).

## ABSTRACT

Knowledge workers frequently encounter repetitive web data entry tasks, like updating records or placing orders. Web automation increases productivity, but translating tasks to web actions accurately and extending to new specifications is challenging. Existing tools

can automate tasks that perform the same logical trace of UI actions (e.g., input text in each field in order), but do not support tasks requiring different executions based on varied input conditions. We present DiLogics, a programming-by-demonstration system that utilizes NLP to assist users in creating web automation programs that handle diverse specifications. DiLogics first semantically segments input data to structured task steps. By recording user demonstrations for each step, DiLogics generalizes the web macros to novel but semantically similar task requirements. Our evaluation showed that non-experts can effectively use DiLogics to create automation programs that fulfill diverse input instructions. DiLogics provides an efficient, intuitive, and expressive method for developing web automation programs satisfying diverse specifications.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).  
UIST '23, October 29–November 1, 2023, San Francisco, CA, USA  
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.  
ACM ISBN 979-8-4007-0132-0/23/10...\$15.00  
<https://doi.org/10.1145/3586183.3606822>

## KEYWORDS

Web automation, PBD, neurosymbolic programming

### ACM Reference Format:

Kevin Pu, Jim Yang, Angel Yuan, Minyi Ma, Rui Dong, Xinyu Wang, Yan Chen, and Tovi Grossman. 2023. DiLogics: Creating Web Automation Programs With Diverse Logics. In *The 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23)*, October 29–November 1, 2023, San Francisco, CA, USA. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3586183.3606822>

## 1 INTRODUCTION

Interacting with web pages to complete routine data entry and migration tasks is a daily part of many occupations, from receptionists to researchers. But these tasks often require repetitive work that can be time-consuming and unfulfilling. Frequently performing these tasks manually can result in human mistakes (e.g., duplicate or missed entries) or frustration [33]. In contrast to the manual effort, web automation uses programs to simulate human interaction, creating a faster and more accurate way to complete mundane tasks. But there is a barrier to creating web automation programs for users without expertise. Through a formative analysis of web automation requests in online platforms such as StackOverflow, we found that non-experts are experiencing difficulty in creating automation programs tailored to their need. To lower the barriers in the program creation process, existing programming-by-demonstration (PBD) systems, such as SemanticOn and Rousillon [17, 20, 24, 56] allow users to manually perform a part of the task and construct an automation program based on the demonstrations.

However, while these tools can handle structured repetitive tasks that follow predetermined, uniform program logic, they are difficult to generalize when the task contains varied input data that require different page actions to fulfill. Consider the scenario where an event planner handles employee information from a spreadsheet for booking. They want to enter employee ID into a web form for every colleague traveling to a conference. While the data entry step is constant for every employee, inputting ID to the same field (i.e. uniform program logic), the subsequent steps could require different actions targeting different UI elements. For example, the planner may also need to enter information about dietary restrictions, seating preferences, and planned attendance into multiple data systems and make different selections for every employee (i.e. diverse program logic based on input). Another illustrative example is when a coordinator is placing a group lunch order for a social gathering. On the food ordering website, they need to conduct repetitive steps to search for the restaurant, click on the food item, and add it to cart. Existing tools can automate this uniform logic based on the website structure (e.g. adding each item in the order they appear), but cannot accommodate when requests don't follow such structural order. For example, two requests might order from different restaurants where items are organized differently, and one request requires side dish options while the other notes a dietary restriction. These diverse specifications would likely require the automation program to interact with distinct UI elements in different sequences, leading to a need for the program to execute a diverse set of logics depending on the input data. But the presence and content of these different types of requests might differ for

each input. Every local specification (e.g. specify attendance on an event page) might require near identical automation steps, but holistically the different requests are combined and scaled with increasing size of input data to create a hard problem, demanding system intelligence to disambiguate different requests and perform the actions accordingly. This necessitates the program to be flexible in its choice of execution in order to automate a large variety of steps based on the input data. To add to the problem, the input data, which describes task requirements, contains enormous heterogeneity in expression (e.g. multiple steps, different phrasing). A study on user commands for web actions revealed that people employ various language phenomena, often involving high-level goal description or reasoning [53]. While the user could include additional program logic in their automation script to account for diversity in input data and website UI, this extra configuration process can become laborious and error-prone. The resulting program is also task-specific, needs to be maintained, and not scalable.

In this work, we present DiLogics<sup>1</sup>, a PBD system built upon a program synthesizer [20] that assists non-experts in creating web automation programs with diverse and generalizable programming logics. The completed program can execute consistent actions for every data input; it also goes beyond symbolic inferences and dynamically executes different UI actions based on the semantic understanding of the task input and web content. To create a scalable automation program, DiLogics first semantically segments the input data to decompose the task into more tractable steps (Step 1, Fig.1). The system represents these steps in a table with a carousel widget (Fig.1.c) that informs users of the task progress. Then DiLogics elicits web demonstration for each step (Step 2, Fig.1), mapping the sequence of UI actions to the description of the step. At every step, DiLogics leverages natural language processing (NLP) models to scrape web content and locate the most relevant web page content via statistical learning. This way, UI actions (e.g., clicks and selections) are dynamically associated with semantically similar elements on the page. In addition, DiLogics employs program synthesis techniques to record the user's actions and their symbolic relationships in the web DOM structure. After a few demonstrations, the system detects the pattern in the action trace and generates an automation program.

As the user demonstrates each step, DiLogics builds a catalog of task steps to UI sequence mappings. Upon entering automation, the system matches each encountered task step to the semantically similar step in the demonstrated catalog and extends the same program logic to fulfill the current condition. When the new step is not meaningfully similar to any previous ones, DiLogics asks the user to demonstrate a new set of UI actions, and adds this step to the catalog for future generalization (Step 3, Fig.1). This approach enables flexibility in the execution of the automation program, as it will always employ the most fitting program logic based on semantic similarity, and perform the actions on the relevant element on the current page. Combining NLP models and program synthesis techniques, DiLogics can generate an automation program that consists of both rule-based structural repetitions, as well as diverse program logics based on the different input data semantics.

<sup>1</sup>DiLogics is an acronym for **D**iverse **L**ogics

We evaluate DiLogics’s usability with 10 participants using four common UI automation data entry tasks. All participants had no prior experience using web automation tools. We showed that users of DiLogics can successfully create automation programs that satisfy the input requests for every task. Despite being novices, participants were able to efficiently construct diverse programming logics by demonstrating different semantic steps. Participants also reported that unlike performing manual actions such as copy-and-paste in data entry tasks, mappings task steps to a set of UI actions via demonstration is efficient, generalizable, and reduces mental effort. Overall, they found DiLogics intuitive to use and effectively covers diverse scenarios by learning the user demonstrations. In the final section, we discuss the implications of DiLogics’ design and future works that can adapt our approach to other interactive collaborations between the human and the intelligent system.

This paper contributes the following:

- A PBD approach that assists users in creating web automation programs with diverse programming logics.
- The technique of semantically categorizing input data and mapping to generalizable UI demonstrations.
- The DiLogics system implementation and user evaluation results assess its effectiveness and usability.

## 2 RELATED WORK

Our work relates to primarily two fields in the PL and HCI communities: web automation and human-AI collaboration. In this section, we identify gaps in existing solutions and draw our design inspirations from these two areas.

### 2.1 Web Automation

The concept of web automation refers to the use of bots to perform tedious and recurring web tasks such as data entry and extraction by simulating human interactions. It is common for knowledge workers to use web automation in order to accomplish their respective tasks [31, 34, 42, 58]. For example, data entry workers may need to automate entering data into a digital system for routine tasks such as processing orders or extracting data.

Many tools have been developed to help users to create automation programs. For instance, tools like Puppeteer, Selenium, Scrapy, and BeautifulSoup allow developers to select elements and define actions to automate. Research tools like Sikuli [64] 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 these tools help lower the effort of creating programs, they all require programming knowledge and cannot disambiguate similar elements or text information.

Even for professional developers, creating automation programs is a non-trivial task. A study showed that experienced programmers have difficulty writing web macros using common web automation frameworks [25]. Participants pointed out that a primary hurdle was the labor of checking syntactical element selectors to create their programs, causing inefficiency and errors. In addition, the program might not generalize to cross-webpage selections where the elements don’t have syntactic similarities. With our work, users can specify the mappings between a task and its corresponding UI

actions via demonstrations. This saves effort on checking selectors to create a program and enables generalization of UI actions for unseen steps beyond structural similarity.

Alternatively, researchers have leveraged large datasets of UI to computationally summarize a mobile screen into a coherent natural language phrase [62], enabling conversational interaction using large-language-models (LLMs) [61], and to ground instructions to UI action sequences [39]. Commercial LLM applications also employ fine-tuned neural models for downstream activity such as UI automation [1, 11]. These tools allow users to prompt the model with high-level natural language intents, which are translated into GUI actions. However, users have limited control outside of describing the task using prompts, and cannot modify the output automation program easily. DiLogics provides a complete pipeline of web automation workflow, from processing input data, to tailoring the program to user demonstration, to refining and handling errors in automation.

### 2.2 Specifying Diverse Programming Logics

Prior works developed techniques that support users to easily express program logics to satisfy task specifications. Systems like SemanticOn and PUMICE allow users to specify conditions by demonstrating examples that are (dis)similar to a given specification (e.g., images of two people interacting, weather is hot) [24, 38]. However, they are designed to handle uniform logic – a binary conditional that determines action or no action applied universally to all content and input. Examples include downloading an image when it contains key objects, or running a macro when the weather is above a certain temperature. In this case, users need to recreate a program when there are multiple specifications that correspond to different UI actions. Commercial tools like UiPath [12] and iMacros [4] allow users to set conditional actions on specific page elements via programming, which requires expertise. But, they also lack task understanding to generalize the conditional outside of the symbolic element (i.e. HTML tag) and could not execute different actions based on input data specifications.

Alternatively, researchers designed neurosymbolic languages with both neural and symbolic elements to create programs that satisfy new specifications via approximation. Neurosymbolic programming is a generalization of classical program synthesis, bridging the gap between deep learning and program synthesis. Unlike deep learning, neurosymbolic programs can often represent long-horizon, procedural tasks that are difficult to execute using deep networks, and they are also generally easier to interpret and formalize than neural networks [18, 52]. In contrast to symbolic approaches, neurosymbolic programming does not require all specifications to be hard logical constraints.

However, this approach has been little explored in the context of web automation. For many years, ML researchers have promoted a “hybrid model” that combines the best of both worlds. As an example, WebQA developed a neurosymbolic system with domain-specific language (DSL) for extracting desired information from datasets that have similar contents but differ in the underlying structures (e.g. DOM structures) [18]. It omitted, however, user actions during upstream activities (e.g., data collection), limiting it to tasks involving a particular dataset (e.g., data extraction). For

data collection, SemanticOn was able to bridge the communication gap between users’ abstract level intent (semantic conditions) and a symbolic system using neural components without defining a DSL [24]. However, while these systems are capable of conditional behavior, their program logic is limited to binary decisions between action and no action, unable to handle diverse specifications with different corresponding actions. Therefore, current neurosymbolic approaches are either restricted to uniform program logic tasks or require the development of a domain-specific language (DSL) to encode neural model output into symbolic systems (or vice versa), making these approaches not scalable.

DiLogics also employs a neurosymbolic approach for creating programs to automate data entry tasks that involve diverse logics. Rather than following the same program logic throughout execution, DiLogics learns from user demonstration and uses statistical learning to automate the current step using the most fitting program logic based on task semantics. By categorizing users’ actions into semantic steps, DiLogics learns action patterns and logic-to-demonstration associations using both symbolic inferences and neural network approximation. The resulting programs extend beyond the uniform program logic that cannot satisfy diverse specifications due to task nature. Through this construct, DiLogics reduces the level of expertise needed by system designers in other areas to build neurosymbolic programming approaches for their tasks.

### 2.3 Programming by Demonstration

A programming by demonstration (PBD) approach has been adopted by many tools in order to further reduce the expertise required, since users only have to interact with the target applications rather than write code [15, 19, 27, 29, 43]. Among these application domains are text manipulation [16, 30, 45, 48, 63], image or video editing [28, 40, 44], and GUI synthesis [46, 47, 49, 60]. For web applications, PBD helps build automation programs without requiring users to understand browser internals or reverse-engineer target pages manually. CoScripter [33], Vegemite [41], Rousillon [17], UiPath [12], and iMacros [4] are examples of the PBD approach to web automation. The resulting programs from user demonstration are represented in visual formats such as a workflow chart [12], a for-loop [17], or in DSL code [4]. These representations require programming expertise, and users have to manually edit program logic which is often nested and convoluted.

Effectively communicating user intent is a major challenge in these PBD systems, and many systems have proposed bridging the gap between user intent and system understanding. Systems like PLOW [14] and PUMICE [38] allow users to express concepts (e.g., hot weather) in natural language and then learn the concepts to generalize the automation. ParamMacros [26] allows users to first generalize a concrete natural language question with potential values to identified parameters, and then create a demonstration of how to answer the question on the website of interest. Scout [59], Designscape [51], and Iconate [65] allow users to iteratively refine their intent by directly manipulating the AI-generated artifacts. SOVITE [35] allows users to correct system misunderstanding via direct manipulation of the highlighted entity on the GUI. Another work, APPINITE [36], also encapsulates the user’s intent in natural

language instructions and clarifies the intention in a back-and-forth conversation with the AI.

Despite promises, specifying intents to cover every case can be tedious. Furthermore, users may not know all the cases in the first place. This suggests that tools need to elicit users to better formulate their intent before creating automation programs. DiLogics addresses this challenge by parsing input data and allowing users to refine their intent continuously during automation by coordinating with our system.

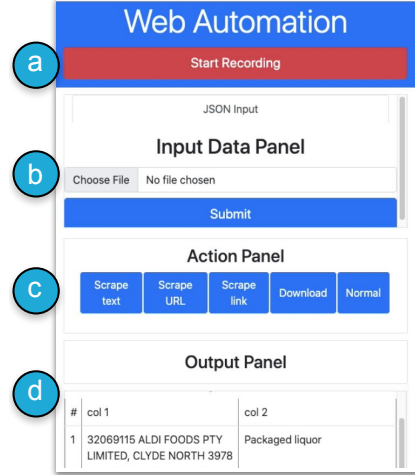


Figure 2: A screenshot of the WebRobot system UI.

### 3 BACKGROUND: WEBROBOT SYSTEM

In this section, we provide necessary information for WebRobot [20], a program synthesizer that constructs a part of the DiLogics system. WebRobot utilizes only web actions and requires no programming expertise, which is consistent with our design goals.

WebRobot utilizes a no-code approach to synthesize web automation programs based on user demonstration. To create a web automation program for a data entry or scraping task, the user first starts recording their actions (Fig. 2.a) and optionally uploads a JSON file (Fig. 2.b) if they need to input data. Then, they start demonstrating how to perform the task by choosing an appropriate action type (e.g., Scrape text) in the action panel (Fig. 2.c) followed by actually performing actions (e.g., clicking the desired text data on the website). After each scraping action, the output panel displays the extracted data (Fig. 2.d). Behind the scenes, WebRobot records every user action with its associated action type. At a very high level, WebRobot infers the user intent by generalizing a trace  $A$  of user-demonstrated actions to a program  $P$  with loops. This generalization is done by “rerolling” actions in  $A$  into loops in  $P$  – specifically, it infers inner loops first and gradually infers outer loops. In particular,  $P$  is guaranteed to not only *reproduce* the actions in  $A$  but also *generalize* beyond  $A$ . In other words,  $P$  performs more actions after  $A$ . This typically means  $P$  is a *loopy program* which “folds” actions from  $A$  into a loop that can execute for multiple iterations, essentially generalizing user-demonstrated actions based on the same program logic. Finally, WebRobot executes  $P$  to

automate the rest of the task, without users manually performing any actions. For more details on the program synthesis algorithm, please refer to the original WebRobot paper [20].

## 4 FORMATIVE STUDY AND DESIGN GOALS

### 4.1 Online Web Automation Requests

To understand the needs and barriers of web automation users, we conducted a formative study analyzing online web automation requests and derive our design goals from the results. We collected posts from developer forums like StackOverflow and sub-Reddit communities (e.g. r/automate), as well as commercial tool platforms like UiPath and iMacros forums [7, 13]. We used BeautifulSoup4 [2] and the available APIs [9] to scrape the title, content, comments/replies, and the URL of web posts. To identify relevant posts and discussions about web automation, we filtered the forums by keywords like “*UI automation*”, “*workflow automation*”, and “*web-scraping*” and ranked the results by popularity.

After data cleaning, we collected a total of 847 posts. We conducted keyword analysis within post content and identified 53% of posts as being written by non-experts without web automation or programming expertise, containing phrases like “*new to [specific tool]*” or “*beginner*”. We also found that 61% of posts were inquiries about how to perform a specific function or approach a task using existing tools. This indicates a potential barrier to usage in existing tools as they require specific domain knowledge and experience to utilize. Combined with a large number of non-expert requests, the learning curve for beginners to accomplish their automation tasks is challenging to overcome.

Our analysis also discovered examples of posts that illustrate conditional specifications. One example is when a user wanted the automation script to iterate through a list of users on a website, and conduct different actions depending on whether the user status is online [3]. In another example, the user intended to automate different UI actions based on a text element value [6]. Although the element can be easily located by the human, the user expressed difficulty in pragmatically accomplishing this behavior. In addition, we found that some users desire a simpler way to create and refine web automation programs. For example, one user expressed a need to record web macros and modify them to automate web actions that are generalizable [8].

Based on the results of our formative analysis, we identified a barrier for novice users to create web automation programs tailored to their needs. Existing tools require domain knowledge, and cannot fully satisfy conditional automation or generalize the web actions based on the page content. We also argue that current online requests are limited by the capabilities of existing tools. With higher system intelligence, users could express the need to create more generalizable web automation programs for more complicated tasks that involve conditional steps.

### 4.2 Website UI and Content Analysis

We also carried out an informal analysis to identify the common UI action sequences for completing common data entry tasks on websites. To do so, we analyzed 40 popular websites across 7 genres, including food, shopping, health, entertainment, travel, communication, and scheduling. These genres are identified by prior study [57]

and extracted from real user requests on forums such as iMacros [5] and Stack Overflow [10] that discuss the creation of web automation programs for data entry tasks. We scraped the UI widget types and analyzed the types of UI action sequences needed to complete tasks for these websites. This led us to identify 8 recurring widget categories: buttons with text (appeared in 100.0% of inspected websites), drop-down menus (77.5%), checkbox/radio buttons beside a text label (72.5%), input box for memo or special instructions (27.5%), calendar widget (20.0%), plus and minus quantity widget (20.0%), and seat map (10.0%). This investigation helped us determine which are the most common UI widgets that an automation program could encounter. We also found that websites utilize consistent GUI elements and interactions for the same categories of functionalities across all pages (e.g. navigation is often associated with buttons or links, search is often associated with an input box). Thus, we design DiLogics to automate GUI tasks under the same web domain where similar UI interactions fulfill the same semantic task. This design scopes the system generalization and assumes that a semantically similar task can be automated via the same UI actions, enhancing the accuracy within the task website domain.

### 4.3 Design goals

Based on our formative analyses and prior works, we devised three design goals to help construct our system supporting users in creating web automation programs for data entry tasks with diverse program logics.

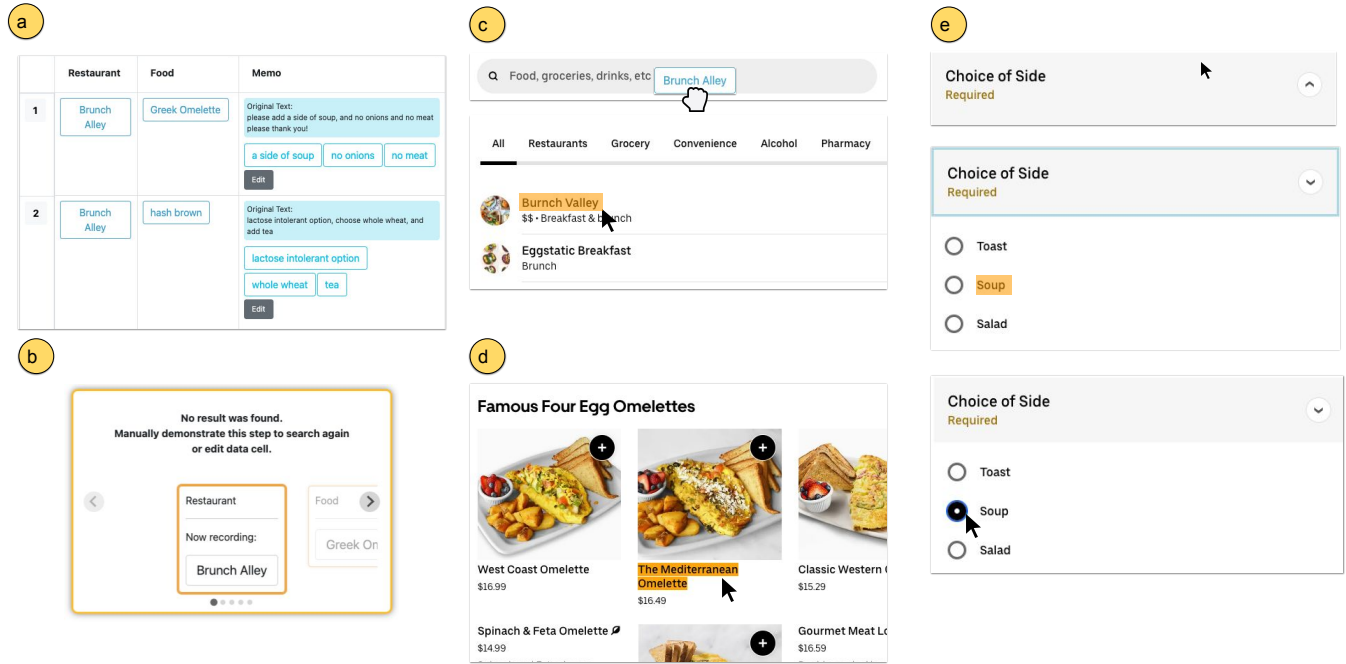
- **DG1:** Generalizable specification of the diverse mapping between task steps and user actions.
- **DG2:** Intuitive and natural interaction that constructs automation from demonstration
- **DG3:** Error-handling capability to modify automation and refine step-action mappings accessibly.

## 5 DILOGICS

### 5.1 The DiLogics User Experience

Emma, a corporation clerk, is responsible for processing food orders for all employees at a team-building event. A spreadsheet of everyone’s food orders and requests is collected through a survey. Emma could manually enter the selections for each order, but that would be time-consuming and error-prone. Instead, Emma uses DiLogics to efficiently demonstrate common categories of task steps, record her web UI actions, and synthesize an automation program that automatically completes the task for her. To begin, Emma opens the DiLogics browser extension and uploads the data sheet, displayed as a table with segmented steps for restaurant, dish, ingredient, and dietary restrictions (Fig.3.a).

**5.1.1 Manual Demonstration.** Emma begins the PBD process and moves her attention to the carousel widget on the target web page (Fig.3.b), displaying the steps to complete the current food order (i.e. data table row). Emma follows the carousel and first searches the restaurant by inputting the name into the search bar and clicking “*Search*” (Fig.3.c). Then, Emma moves to the next carousel step, which is to select the dish. DiLogics semantically searches the relevant text content on the page. Emma follows the highlight and finds the best fitting dish, then she clicks to navigate to the



**Figure 3: An example workflow of DiLogics' user demonstration.** Upon uploading the input file **a**, the carousel widget displays all task steps for the current row **b**. To demonstrate, the user first drags the restaurant name to the search bar and navigates to the intended page **c**. Then, the user moves the carousel to the next slide. DiLogics semantically searches for the dish name and the user clicks on the highlighted result to enter the detail page **d**. On this page, the user demonstrates each remaining specification. The first request is adding soup as a side item. DiLogics initially does not find any relevant option, so the user demonstrates by first clicking on the drop-down menu for sides. The system then highlights the relevant option, and the user clicks on the corresponding radio button. The user then moves on to the next specification until the end of the data row.

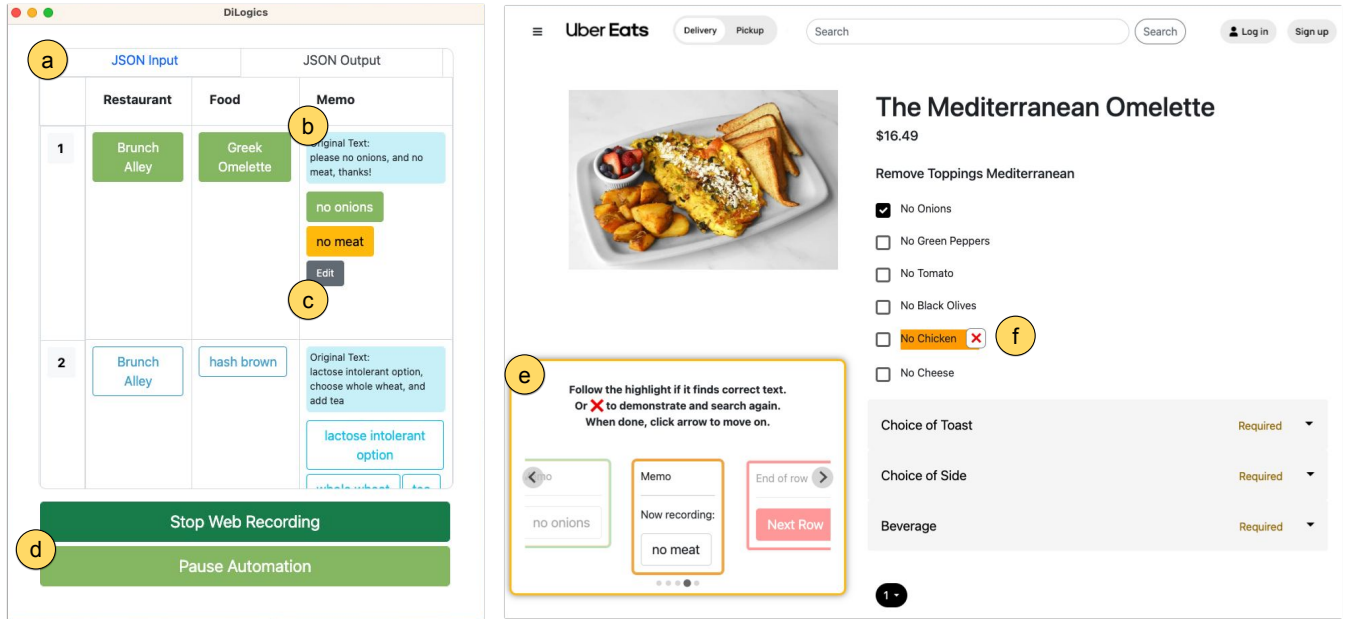
food details page (Fig.3.d). So far, the demonstrated actions are consistent for every food order, which can be handled by existing PBD tools. But they are limited when every order contains different specifications that require different program logics to fulfill, as we see in later steps.

Again advancing the carousel, the current step is a segmented user request to order “a side of soup”, however, no highlight is shown as the side item menu is not expanded. Emma opens the drop-down menu, DiLogics detects a page state change, and highlights the “Soup” menu item (Fig.3.e). This sequence of three actions (open menu, search, and click) should not be executed for orders that don't include a side, but existing PDB tools will require configuring a conditional on the symbolic element (i.e. page contains an HTML element with tag “Side”) to handle this case. Instead, DiLogics dynamically apply the best fitting program logic by storing this action trace under the category of “a side of soup” for generalization. Emma keeps demonstrating each task step following the carousel progression, until arriving at the last slide, which prompts her to complete any remaining action in this row. Emma clicks “Add To Order” to complete this order request and clicks “Next Row” on the carousel. DiLogics then generates the carousel steps for the next input data row (Fig.4.e).

**5.1.2 Semi-automation.** After Emma demonstrated the second row following the carousel, DiLogics synthesizes an automation program based on user action pattern and enters the semi-automation mode. In this stage, the system predicts the next step of action (e.g. “Next step is clicking on the highlighted element”) and prompts the user to review on the carousel widget. Emma can click “Confirm” to allow DiLogics to automate this step, or click “Cancel” for incorrect predictions and manually demonstrate the correct step. After Emma authorizes the system predictions to fulfill the third-row order, DiLogics enters full automation with the synthesized program.

**5.1.3 Full Automation.** In full automation mode, DiLogics completes the remaining orders row-by-row in the input table, and step-by-step in each row's table cells. DiLogics automates actions that are consistent for every order, such as inputting the restaurant name for search or clicking “Add to Order”. Moreover, DiLogics constructs different program logics to handle different combinations of task steps and generalize to new steps. Emma is pleased to find that DiLogics is able to correctly perform UI actions for “add a daily soup” even though it is a new condition on a different restaurant page. DiLogics achieves this by semantically matching the step to “a side of soup”, which was previously demonstrated. It can then perform the same UI action sequence but on the “daily soup” option on the current web page, despite structural differences.





**Figure 4: DiLogics UI Overview.** Left is DiLogics’ extension window. After users upload an input file ①, the data is semantically segmented into task steps ② and rendered into a table. Users can modify the inaccurately segmented step ③. They can also control the flow of the task, and pause automation ④. Right is the target website, with an overlaid carousel displaying the progress of the current data row ⑤. Steps that have been completed are marked green on the data table and carousel, and the current step is marked yellow. Users follow the carousel to start the task demonstration. DiLogics semantically searches web page and highlight the most relevant text. If the highlight is incorrect, users can cancel it ⑥, then edit the step ③ or navigate the page to reveal relevant content (e.g. expand the drop-down menu). After demonstrating the current step, users can advance to the next slide on the carousel. Users can click “Next Row” at the end to move on.

**5.1.4 Refine & Repair.** Occasionally, when DiLogics encounters a new step that does not semantically match with any previous categories (e.g. “select barbeque sauce”), the system pauses and prompts Emma to demonstrate. When DiLogics makes a mistake by highlighting or selecting the wrong element (e.g. highlight “Sauce” menu heading before the user reveals sauce options in the drop-down), Emma pauses the program to manually cancel the highlight (Fig.4.f), expand the menu, and record new demonstrations to account for new conditions. This manual effort decreases as DiLogics learns and expands its knowledge of the task semantics. Existing PBD tools require users to program the automation for conditional behavior, and could not continuously refine or repair as it executes. Using DiLogics, Emma did not have to create any conditionals to configure the automation program to handle each task step, DiLogics is able to acquire task understanding and generalize demonstrations based on the step specifications. Moreover, Emma has the ability to refine program logic and repair errors at any point during the workflow. Eventually, the program is able to efficiently execute UI actions based on this large datasheet. Emma checks the order list and the shopping cart to verify that the task has been completed, and purchases to confirm the order.

## 5.2 DiLogics’ Design Rationale and Iteration

We iteratively designed the DiLogics system based on the feedback from a 10-participant usability evaluation using the prototype. In the initial iteration, DiLogics required users to interact with the extension page to control the flow of demonstration recording, resulting in frequent attention switches between the task website and our tool. Users also needed to manually trigger a semantic search in their action trace which was inefficient. To address these issues, we made significant improvements to the user workflow and interaction process. The final version of DiLogics features a carousel widget (Fig.4.e) overlaid on the target website to guide the task progression, displaying the past, present, and next task steps, following Norman’s visibility principle [50]. The widget affords and constrains movement back and forth, giving users a sense of direction and task progress. In addition, by Horvitz’s mixed-initiative UI principles [23], we anchor the carousel on the task web page to alleviate effort and reduce context switches. In addition, DiLogics now actively searches for semantically relevant page content at every step and website state change, increasing system intelligence to simplify user’s workflow. As the task goes on, the data table provides color-coded highlighting to signal completion status. This is guided by Norman’s feedback principle [50] to help users understand DiLogics’s response and confirm their actions. Moreover,

the semi-automation mode after demonstration and before full automation corresponds to Horvitz’s principle of minimizing the cost of poor system guesses [23]. By walking through one iteration of automation with the user, DiLogics narrows the gulf of evaluation and allows users to validate system actions.

### 5.3 Design and Implementation

We implemented DiLogics as a Chrome browser extension, building upon the core program synthesis engine from WebRobot [20]. Primarily, it uses plain JavaScript for recording front-end interactions on the web page. For task step categorization and semantic similarity matching, we adopted two off-the-shelf machine learning models: Sentence-BERT [55], a pre-trained network that derives semantically meaningful sentence embeddings that can be compared using cosine-similarity, and SpaCy [22], a trained natural language processing pipeline. The system design and implementation can be separated into three parts, detailed below.

**5.3.1 Step 1: Data Input and Specification Parsing.** To process the input data into tractable steps representing different specifications, users can first upload an JSON input file to DiLogics which renders a data table (Step 1 Fig.1.a, Fig.4.a,b). While currently only supporting JSON, the input data can be easily extended to other file formats such as CSV, Excel workbook, etc. The task file could have inherent structures (i.e. columns and rows of information), but DiLogics further parses the input texts and automatically segments them into semantic steps using SpaCy [22]. The data row with the highest number of identified steps will be ranked first. This is to place most of the manual demonstration efforts at the start of the task, allowing users to record actions for most semantic categories upfront, reducing interruption in later automation. Users can inspect and edit the data if they find incorrectly parsed or ambiguous steps at any point during the demonstration or in automation by pausing the program (Fig.1.b, Fig.4.c).

The segmented task steps could also contain specifications that are too abstract or too detailed, which might be misinterpreted by the NLP model and fail to connect to web page content. For example, a user might note that they are “*lactose intolerant*” in the specification, but the web page only contains a “*No cheese*” option. A semantic search of the original step using yields no match on the page (Fig.5.a). The user can then manually rephrase this condition by editing it to “*remove dairy products*” (Fig.5.b), which DiLogics understands and highlights for demonstration (Fig.5.c). Throughout the program creation process, users have the agency to repair and refine task specifications with the help of DiLogics.

**5.3.2 Step 2: Demonstrations and Mapping.** To create an automation program, users start the web recording (Fig.4.d) and demonstrate actions for each task step from the start of the task table. DiLogics’ carousel widget organizes the current row’s steps into ordered slides, guiding the users to interact with the website to fulfill the current step as if completing the task manually (Fig.4.e).

Every web macro will be recorded and used to synthesize a repeatable program (e.g. a for-loop). The program synthesis engine based on WebRobot [20] enables inferences based on input data and website structures, such as sending each table cell data into a list of

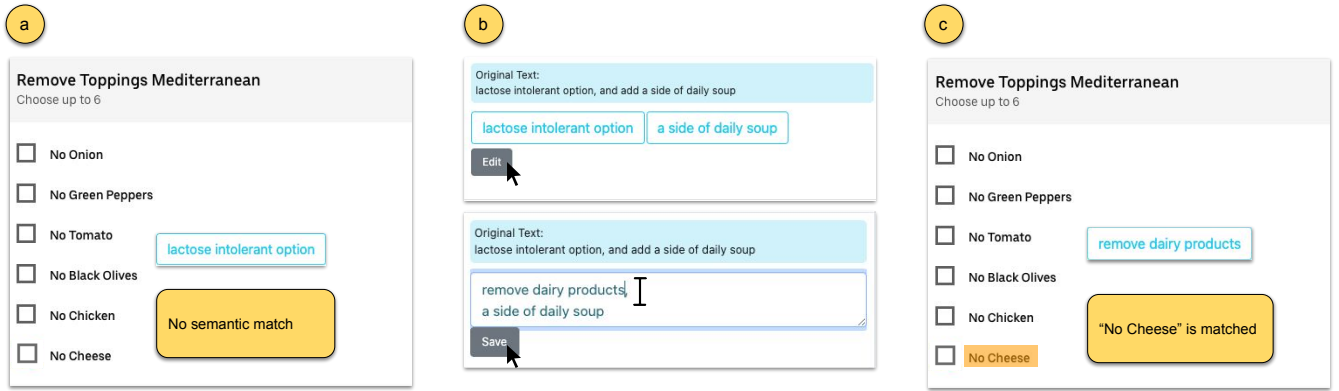
input fields in order. However, it could not generalize the automation to perform different sequences of actions based on the step specification and the page content. DiLogics extends WebRobot’s functionality by incorporating semantic search in its automation execution. After every user action, DiLogics scrapes the web page and highlights the page element that is most semantically relevant to the task step description, as determined by the cosine similarity of the two text phrases [55] (e.g. step “*remove dairy products*” relates to page option “*no cheese*”). This intelligent search feature alleviates users’ mental effort to process page information. If the highlight is accurate, users can continue to demonstrate (e.g. click the checkbox on “*no cheese*”), or they can cancel the highlight (Fig.4.f) to correct the system by editing the task step or guide the system to highlight the desired region by revealing more relevant page content (e.g. expand a menu to reveal more selections). DiLogics records this entire sequence of UI actions and maps the task step to the list of macros as a key-value pair.

As users demonstrate different steps, DiLogics constructs a catalog of step-to-UI action mappings. In later automation, to perform each task step, DiLogics first inspects the catalog to find the most similar demonstrated step via semantic matching (e.g. new step “*no meat*” is similar to “*remove dairy products*” in meaning). Then DiLogics generalizes the stored UI action sequence to the current step and automates based on the current highlighted content (e.g. highlight “*No chicken*” and click the checkbox next to the option). Note that since DiLogics constantly searches for and locates the most semantically similar element, the automation can execute macros on the correct UI regardless of website DOM structure, going beyond structural inferences and layout constraints of the task website. By recording mappings between diverse specifications and UI actions DiLogics constructs automation programs with malleable programming logic by inserting the proper UI actions for each step in real time to fulfill diverse task specifications.

**5.3.3 Step 3: Automation, Refinement, and Error-handling.** Users follow the progression of the carousel widget to record the manual demonstration for each step in a data table row, which counts as one iteration of the task (e.g. completing one person’s food order). After demonstrating for two iterations (i.e. two rows), DiLogics detects the repetitive pattern in the user action trace and generates an automation program [20]. The system then enters a semi-automation stage for the third iteration, where it prompts users with the predicted action for the current step. Users can either confirm to authorize the automation of this step, or cancel in case of incorrect prediction and redo the demonstration for this step. After confirming the synthesized program’s predicted actions in the third row, DiLogics enters full automation.

During full automation, DiLogics iterates the remaining rows of input data and perform corresponding actions based on generalization from the previous demonstration. The carousel advances with the progress of the automation, and each data table cell is marked green when that step is executed. When encountering novel cases that do not match with any steps in the catalog (Fig.5.a), DiLogics pauses the automation and elicits a demonstration to fulfill the new specification. After users demonstrate, the system appends a new step-to-macros mapping to the catalog. Through this process, the





**Figure 5: Editing data table and specifying step.** During step demonstrations, users might find that a request does not result in any semantic match on the web page content (a). This could be due to incorrect parsing, ambiguity, too high or too low a level of specificity, or limitation of the NLP model. The user can manually edit the itemized step to match the page content more closely (b), and demonstrate the UI actions for this new category of step (c).

automation is refined with added categories of demonstration, and the system enhances its capability to handle diverse specifications.

In the event of a system error during automation (e.g. highlights the wrong element or executes wrong macros), users can pause the program (Fig.4.d) to manually inspect and fix the error. They can also edit the data table cell if the step specification is vague (Fig.4.c), or re-record the incorrectly executed step with new demonstrations. DiLogics provides different error-handling techniques to address input data misinterpretations and system logic errors. Users can gradually transition from manual demonstrations, to evaluating system predictions, and finally to full automation, but always preserve the control to refine and repair the program at every stage.

## 6 SYSTEM EVALUATION

In order to evaluate DiLogics’s general usability in assisting users with diverse program logic data entry tasks across different domains and websites, we conducted an in-person user study. We used the usage evaluation strategy in the HCI toolkit to guide our study [32]. The study recruited 10 undergraduate students (6F4M, average age 21.1 y.o., average coding experience 2.3 years, denoted P1-P10) from a large public university. None of the participants had prior experience with web automation tools.

Since we are implementing a new PBD approach, a within-subject experiment would be difficult as there is no clear baseline to compare to DiLogics in solving automation tasks with diverse specifications. However, our study reveals findings on the system’s usability, coordination with AI, and error-handling in continuous programs, all of which can provide insights into future system designs.

### 6.1 Study Design

Upon signing the consent form, each participant first watched a tutorial video of DiLogics’s interface and features. Then participants performed four different task scenarios using DiLogics. For each task, an input file and a task description were provided. Each input file contains 10 rows of different requests, and each request

requires multiple steps with diverse program logics that need to be accounted for by the participants. Additionally, the specifications were intentionally designed to have varying levels of abstraction and ambiguity. This helps examine DiLogics’s refinement features for handling unseen request steps. The participants could call for 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, participants filled out an exit survey with Likert-scale and short-answer questions on system effectiveness, usability, and mental effort [21]. Each participant was compensated \$25 for their time. Each session took 60 minutes and was conducted in person on our machine. All sessions were screen- and audio-recorded. Our study is approved by the ethics review board at our organization.

### 6.2 Tasks

To design realistic tasks for users with limited experience with automation tools, we take inspiration from prior studies on common categories of web tasks, UI interactions to accomplish those tasks, and natural language commands to describe the tasks.

Based on QAWob [57], a benchmarking study that collected more than 500 website templates and sequences of GUI actions via crowdsourcing, we designed our tasks to require common UI interactions such as search, text entry, drop-down, and click. We also determined our task domains from the common website template categories, such as dining, entertainment, and shopping [57]. Then we constructed four tasks around popular websites in these domains that participants are likely to be familiar with and do repetitive work in, including UberEats, Amazon, GoodRx, and Ticketmaster. Each task involves using DiLogics to create a web automation program that inputs a list of requests to the target website (e.g. food orders with different restaurants and dishes) and fulfills the specifications (e.g. order side, remove ingredient). We limited the number of requests to 10 for each task to standardize the difficulty.

We compose the content of the task input data files following guidance from a dataset of more than 50,000 natural language commands to describe GUI actions on web elements [53]. The dataset summarizes common categories of language phenomena to express UI action goals in English. The dataset also reveals that many language commands collected from crowd workers go beyond ordinal or visual reasoning (e.g. “*click the top-most article*”) and use semantic reasoning to describe the goal and target of the GUI actions (e.g. “*change website language*” → a clickable box with text “*English*”) [53]. Following these outlines, our data files utilize five commonly classified language phenomena (Relational Reasoning, Substring Match, Paraphrase, Goal Description, and Image Target) [53] to describe the task specifications. We structured each data file to contain at least 5 diverse steps to maintain the same level of complexity and effort for demonstration. We provide the task input data files so participants can focus on experiencing the full features (e.g. condition failure, new demonstration) of DiLogics within the time constraint of the lab study. Therefore, we didn’t let participants specify task input as their instructions may not encompass all DiLogics use cases. Please see the appendix for the specification of each task (Appendix, Fig.7).

### 6.3 Results

**6.3.1 Time and Accuracy.** The user study recorded 40 task completions in total (10 participants x 4 tasks). Table 1 displays analysis for each task, including the average and standard deviation of completion time (in minute:second), task accuracy, and number of attempts to complete this task. Task accuracy is based on each row, deemed as one request (e.g. one food order with multiple specifications). Task accuracy is defined as the percentage of data rows that perfectly satisfied the specification after UI automation (e.g. selecting all the correct options in a food order). If users record a demonstration incorrectly, and the system fails to generalize this step in another data row without the user’s repair, that data row is counted as incorrect. The overall average duration to complete a task is 08:01, and the overall average task accuracy is 91.2%. In 29 of the 40 recorded task completions during the study, participants created automation programs that perfectly satisfied all the specifications in the input files. However, there are cases when users’ demonstration fails to generate a program due to human error (e.g. misclick, double click, unfamiliar with task website), reflected in the number of attempts. But errors and retries become less frequent as users learn the tool.

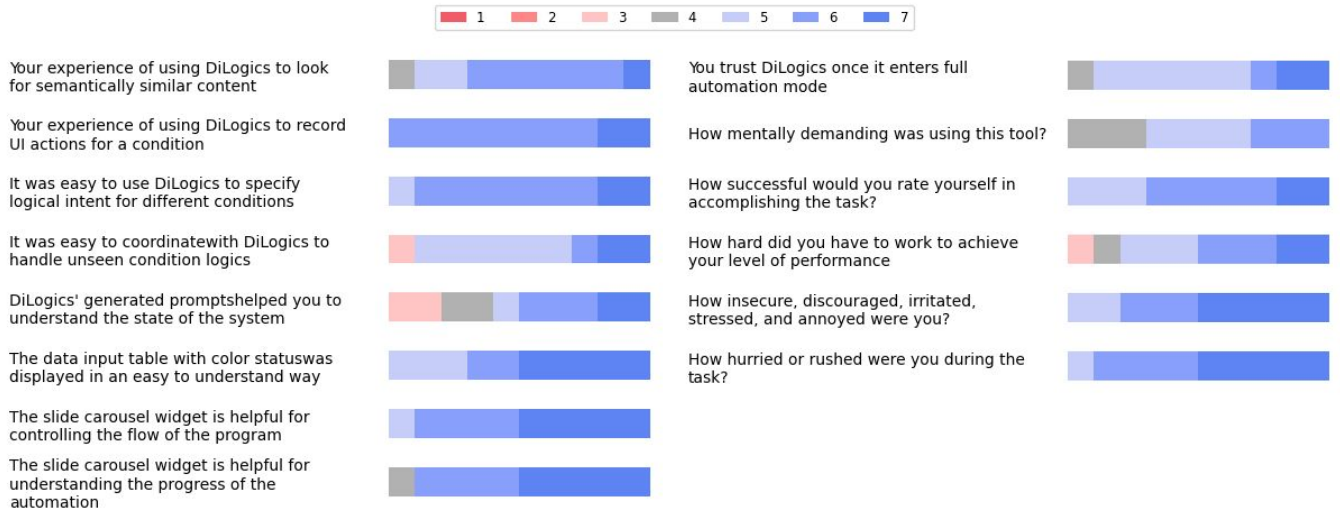
Since our study does not compare DiLogics’ approach to a baseline, we keep the task order consistent for every participant and did not conduct with-participant comparisons. However, we do observe participants requiring more attempts and longer time to complete task 1, indicating a learning curve. Some participants also noted that a “*high level of attention*” (P9) is required at the start of the study to “*be careful about the order of the clicks*” (P7, P9). However, after completing four tasks, participants rated themselves as successful in accomplishing each task on a seven-point Likert scale ( $mean=5.9$ ,  $SD=0.74$ , 7 is very successful, Fig.6). Participants also thought they did not need to work very hard to achieve the performance ( $mean=5.4$ ,  $SD=1.26$ , 7 is no hard effort at all).

Task	Time (mm:ss)	Accuracy	# of Attempts
1-UberEats	08:53 (02:17)	88.9% (16.6%)	1.8 (0.79)
2-Amazon	07:32 (02:54)	100% (0.00%)	1.6 (0.52)
3-GoodRx	08:33 (02:21)	94.0% (8.43%)	1.7 (0.48)
4-Ticketmaster	07:06 (02:33)	82.0% (31.6%)	1.5 (0.53)

**Table 1: User study results expressed in average (SD) format.**

**6.3.2 Effectiveness and Usability.** The participants also rated DiLogics’ ease of use and efficiency from “1 - very negative” to “7 - very positive”, detailed in Fig.6. They found their experience positive when using DiLogics to semantically search for content ( $mean=5.7$ ,  $SD=0.82$ ), recording UI demonstration for a step ( $mean=6.2$ ,  $SD=0.42$ ), and specifying logical intent for different specifications ( $mean=6.1$ ,  $SD=0.57$ ). Participants commented that DiLogics is helpful (P4, P5, P8), interesting (P1, P5, P7), and effective for handling tasks with batch requests (P3, P5, P8). P5 believed that DiLogics is “*good for repetitive tasks, [where] human might misclick or select wrong content due to large load [of requests]*.” The workflow from manual demonstration, to semi-automation, and eventually full-automation was also thought to be intuitive, keeping the human in the loop to aid the system’s learning process (P1, P4, P5, P7, P8). P4 commented that DiLogics is “*very intuitive, [with] easy to follow instructions, only takes two trial runs and [DiLogics] knows how to do the rest.*” Six out of ten participants noted that DiLogics is powerful at information searching and interpretation, automating UI steps across different conditions. P9 expressed that “*semantic matching works for...websites [that] have different layout and structure.*” Overall, participants recognized the system’s ability to learn a variety of GUI actions associated with the task specifications and to accurately reproduce desired interactions.

**6.3.3 Coordination and Error-handling.** Participants found DiLogics relatively easy to coordinate and straightforward, especially for the initial demonstrations to specify program logics (P4, P5, P7). Many participants (P2, P3, P4, P5, P8, P9) thought the interaction with the carousel widget provided them with a sense of control ( $mean=6.4$ ,  $SD=0.70$ ) and helped them understand the progress of the task steps ( $mean=6.3$ ,  $SD=0.95$ ). P6 also noted that the carousel, combined with the semantic highlight, can inform users of the web page content, alleviating the effort of navigating and processing the entire web page. In addition, once in automation, participants found the execution smooth (P1, P5, P7, P9). However, during the transitions between user demonstration and automation (i.e. semi-automation, or system pause to demonstrate a new step), half of the participants found themselves sometimes unsure whether it was the users’ turn to intervene or the system’s turn to automate. Therefore, they desired clearer guidance on the stage of automation and turn-taking. Participants also commented that sometimes the web macro execution response is not synchronized with the carousel progress and table status coloring, which caused confusion (P1, P3, P5, P6, P7). This is due to the fact that some UI actions are not grouped into any task step. For example, users might need to click “*Add to Order*” at the end of each request, but this implicit action is not categorized in any task description. DiLogics can improve by



**Figure 6: Survey Response.** For usability (left) and trust, 1 is very negative, and 7 is very positive. For NASA-Task load index [21] (right), 1 is very high mental demand, effort, stress, feelings of being hurried, and very unsuccessful.

representing the synthesized program with more context, visualizing past and future predicted actions in addition to the current step.

In terms of error handling, participants utilized DiLogics' error repair techniques at every stage. Nine out of ten users edited the input data table to rephrase segmented steps to the appropriate detail (e.g. "lactose intolerant" to "remove dairy products"). Seven participants rewound the carousel and re-recorded step demonstration in the event of human error (e.g. misclick or double click). In addition, eight participants manually repaired UI action errors (e.g. a wrong option is highlighted or selected) and four participants paused during automation to inspect system behavior. We observed some instances of participants noticing an error but not fixing it. In the interview, participants expressed that the automation continued (e.g. navigated to a different page) before they could take action to pause and repair. P8 and P9 suggested that a redo or undo option in the system workflow would further lower the user's effort to repair errors. Future works can make the error-handling interaction more accessible (e.g. more salient and editable execution trace) and provide more processing time or opportunities for users to react to undesired behaviors (e.g. summary of results at the end of task).

**6.3.4 Mental Effort and Trust for AI System.** Overall, participants rated relatively low mental effort ( $mean=5.0$ ,  $SD=0.82$ , 7 is not mentally demanding at all) and very low level of stress ( $mean=6.3$ ,  $SD=0.82$ ) during the study (Fig.6). Six participants rated the initial demonstration effort to be high. P7 noted that "[demonstration] is a bit heavy as [need] to worry about clicking on something wrong, and to be careful about order the clicks". As the program shifts to automation, eight out of ten participants reported decreasing mental effort. However, P5 and P8 believed the semi-automation required the most effort, as the users needed to process and react to the system prompts instead of doing intuitive manual work.

In terms of trust, participants reported a relatively high level of trust when DiLogics starts executing the program in automation

( $mean=5.4$ ,  $SD=0.97$ ). P4 suggested that "would trust the system with more learning [of the tool] and familiarity of the website" while P3 mentioned that "when the automation seems correct, [they] don't need to watch the system." Participants expressed that the stake of the tasks (P2, P4) and familiarity with the input (P1, P6) are important factors for their trust towards DiLogics. From the evaluation, DiLogics requires low mental effort after the demonstration phase. And the system elicits a general level of trust from the users. Researchers can focus on providing more guiding feedback and trust cues to lower mental effort and aid users' trust.

## 7 DISCUSSION

### 7.1 Task to Program Logic Mapping

To handle diverse program logics, DiLogics creates mappings from each task step's natural language description to its corresponding UI action sequence. This approach encapsulates the UI behaviors inside a natural language label that can be easily compared and generalized. Users define different programming logics for each category of requests. Generalization is built upon the assumption that steps similar in meaning will require similar actions on UI elements with similar affordances. From the system evaluation, participants found the task step generalizations interesting (P8), accurate (P7), and even mind-blowing (P1) in terms of capability. P5 notes that the demonstration process is important as "[the user] teach[es] the system the rules to automate the steps... helpful to keep human in the loop." DiLogics applies the same semantic intelligence to processing web page content. Once synthesized an automation program, DiLogics generalizes the UI actions to the most relevant UI elements on the current page, despite layout and content differences from the original page user demonstrated on. P1 expressed that they simply needed to "let the program search information on the website [and] do series of actions", increasing efficiency as users do not need to spend excess time to find the same information.

However, one limitation to the mapping between task steps to UI macros is that the task does not explicitly specify every required action. For example, while users specify their dietary restriction in the food order, they would not specify the need to click on “Add to Order” when ordering is done, as it is implied. DiLogics captures and automates this type of action with uniform program logic (i.e. does not change based on input) via demonstration. But participants sometimes lose track of the progress as these actions are not represented in the input data table nor the step carousel. Therefore, more signals of system state and current action could be added to improve usability.

## 7.2 Neurosymbolic Web Automation

Web automation tasks often involve the repetition of GUI actions following certain rules on the website DOM or input data structure. In symbolic systems, users have a high degree of control to define the rules based on demonstration or programming instructions, if they acquire tool and/or programming expertise [20, 24, 33]. While existing tools can establish symbolic patterns based on UI element properties and web page structures [4, 12], they do not possess the task understanding of the input data nor web page content. With an increasing volume and diversity of content on the web, symbolic program constructions are limited as the conditions do not apply to the content semantics and do not match the abstraction level of user intent. Recent advancement of LLMs leads to a rising need for high-level system understanding to provide an easier method for users to describe their intent in less specific ways that do not require programming expertise. LLM-powered tools allow intent expression using natural language and can translate abstract intents into executable steps on the web-based on powerful content understanding. But current LLM tools offer limited control of the program construction and execution [1, 11]. Users can only specify intent using prompts and examples, making these tools more similar to an API that responds to individual requests rather than large-scale data. In addition, pure statistical-learning-based tools can be inconsistent in output generation, but current tools often provide very limited or no error-handling and refinement techniques.

Neurosymbolic systems offer a hybrid model that combines symbolic inferences and similarity-based predictions. One main contribution of DiLogics is enabling web automation programs to generalize execution steps based on both symbolic and semantic learning. Mappings are generated between symbolic GUI executions and task semantics to bridge high-level user intent and lower-level web macros. Compared to LLM-based web automation tools such as Adept AI, or Taxy AI [1, 11], DiLogics provides an end-to-end pipeline from input data parsing to refinement and error repairs, making it a more complete and robust workflow for the downstream tasks of web automation. The implementation of DiLogics can adapt to evolving LLM models to harness the power of task and content understanding as the neurosymbolic approach to UI automation is generalizable.

We argue that this neurosymbolic model can be applied to other automation tasks that involve the semantic understanding of content, such as information organization, content transformation, and generative creation. Future works can explore how to ground statistical learning models, such as LLMs, in specific task frameworks

(i.e. web automation) with general rules (i.e. structural inference on DOM), and how to provide users agency to tailor the process on top of editing prompts.

## 7.3 System Scope and Limitations

The novelty of DiLogics’ design is in leveraging semantic understanding to enhance UI automation through a mapping between natural language step categories and web macros. This mapping can be established for any web automation tasks where task descriptions can be connected to symbolic UI interactions. Additionally, the set of interactions for data segmentation, programming logic demonstration, refinement, and error repair can be generalized to any other PBD systems. While the system is implemented with an existing program synthesizer [20] and off-the-shelf NLP models [22, 55], DiLogics is not dependent on any specific tool or model. For example, DiLogics could adopt the latest iteration of LLM to increase the system’s semantic understanding capability and content-matching accuracy.

However, the current implementation of DiLogics is limited to understanding text web content and does not support other modalities such as images. This is because the system extracts text-to-element relationships from the web page’s HTML to perform semantic search and UI automation. This restriction means that DiLogics cannot infer meaning from images or pure graphical UI (e.g. icons without alt-text), even though human users might express intent in relation to visual information [53].

Despite its generalizability to automate on websites with different DOM structures, DiLogics requires structured input data. The synthesized automation program needs to form a repeatable instruction set (i.e. a loopy program) grounded by symbolic structure on the input (i.e. consistent number of data columns and column ordering). This means that the input data cannot be completely unstructured like a natural language paragraph. In addition, DiLogics requires two iterations of demonstrations to form an automation program; users have to spend some manual effort to perform the first two rows at the beginning of the task. This means DiLogics cannot perform one-shot automation with a prompt, as can be done by LLM-based automation tools [1, 11].

Finally, while DiLogics’ step to UI actions mapping enables generalizability for similar task semantics, these mappings are restricted to one-to-one relationships. This was based on the assumption derived from our informal analysis of web UI and content, where for GUI tasks under the same web domain, the same semantic task is fulfilled by similar UI interactions. For tasks that span multiple domains and require different UI actions for the same semantic task, DiLogics could lead to high demonstration effort depending on the variety of UI actions and the task size. The main barrier to one-to-many mappings is the analysis of the webpage state and content. DiLogics is unable to support steps that require updating the system state and repeatedly performing actions to satisfy specifications. For example, a task step to “*remove all lactose intolerant option*” might map to a sequence of actions to search and check “*no cheese*”, but DiLogics will deem the condition fulfilled after the UI actions are executed. This means that the system will not iterate through the web content, identify the state of the condition, and find all applicable options to remove (e.g. also need to select “No

*milk*”), as it requires holistic semantic understanding of the task requirement and the UI states.

As a result of these limitations, we scope DiLogics’ capability to handle tasks where the input data is structured, the target website contains text content describing UI elements (e.g. text button, checkbox with text), and the specifications do not require repeated check on website state to fulfill. In the next section, we provide potential directions for future work to overcome these identified hurdles. We also aim to highlight insights in DiLogics’ neurosymbolic system design and implementation, as well as the interaction techniques to facilitate continuous human-AI collaboration.

## 7.4 Future Work

Based on user feedback from the system evaluation, we provide several directions to aid future system designs. First, as discussed in DiLogics’ limitations, the semantic task understanding is limited to textual web page content. Future works can leverage multi-modal neural networks, such as CLIP [54], to understand the website’s visual content as well, expanding the capability to handle user intents that involve visual reasoning (e.g. click on the dish with fish on the image). A parallel approach is to construct a knowledge graph of the website content, which could connect different contents in the form of text, image, video, and/or audio [37]. This can generate a holistic view of the web page and even relate different pages, increasing information searching capabilities during the automation. Future works could potentially achieve this using content summarization techniques to transform and embed all types of content in the same space and compare similarities. The additional high-level understanding of the web page content and the task goal can potentially expand the existing one-to-one task-UI mapping to a one-to-many relationship, enhancing generalizability.

Another limitation of DiLogics is that the automation can not perform logics that require constant analysis of the website state (e.g. check if all lactose intolerant options are chosen). Future systems can constantly analyze the website’s state and task completion status after every UI action or DOM change. This requires storing the state of the website and understanding the status of UI elements at a semantic level (e.g. the “No cheese” option is selected, but the condition is not fulfilled as “No milk” is not selected).

Finally, to further reduce user effort and make web automation programs easier and more accessible to create, future works can potentially derive patterns of execution based on previous task completion. Since every task generates an automation program, there might exist many overlaps in steps and programming logics for tasks in the same domain. Researchers can leverage the history of these completed tasks to make predictions on a new task. If the task can be processed through past similar tasks, the user might not even need to perform initial demonstrations to start program generation; the past tasks may already be capable of predicting the execution of the current one.

## 8 CONCLUSION

To support creating web automation with diverse specifications, we designed and developed DiLogics, a PBD tool that assists users in segmenting task requests and synthesizes programs based on user demonstration of example steps. The steps are mapped to

sequences of UI actions and can be generalized using both symbolic inferences and semantic similarity via statistical models. In a system evaluation, we found that participants can effectively use DiLogics to generate UI automation scripts and complete tasks with high accuracy. We propose a generalizable neurosymbolic approach that combines the advantages of rule-based systems and neural networks. Our work can offer insights into future system and interaction designs that leverage semantic understanding in traditionally symbolic automation systems.

## 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, and the National Science Foundation under grant numbers CCF-2236233 and CCF-2123654.

## REFERENCES

- [1] 2023. Adept AI. <https://www.adept.ai/blog/act-1> Accessed: March, 2023.
- [2] 2023. Beautiful Soup. <http://www.crummy.com/software/BeautifulSoup/> Accessed: March, 2023.
- [3] 2023. Help Needed with Keyword Assertion. <https://forum.imacros.net/viewtopic.php?f=7&t=30990&sid=117ef7144580822f4982137e21eddfc2> Accessed: March, 2023.
- [4] 2023. iMacro. <https://www.progress.com/imacros> Accessed: March, 2023.
- [5] 2023. iMacro Forum. <https://forum.imacros.net/> Accessed: March, 2023.
- [6] 2023. iMacros Script to Extract Specific Text From Static Position. <https://forum.imacros.net/viewtopic.php?f=7&t=28432&sid=01bf0edf807fd3d4bb507a35b87155ad> Accessed: March, 2023.
- [7] 2023. iMacros Forum. <https://forum.imacros.net/> Accessed: March, 2023.
- [8] 2023. Programatically interact with the IE browser to fill in forms and navigate etc. <https://stackoverflow.com/questions/8438782/programatically-interact-with-the-ie-browser-to-fill-in-forms-and-navigate-etc> Accessed: March, 2023.
- [9] 2023. Reddit API. <https://www.reddit.com/dev/api/> Accessed: March, 2023.
- [10] 2023. Stack Overflow. <https://stackoverflow.com/search?q=%5B selenium %5D + semantic> Accessed: March, 2023.
- [11] 2023. Taxy AI. <https://taxy.ai/> Accessed: March, 2023.
- [12] 2023. UiPath. <https://www.uipath.com/> Accessed: March, 2023.
- [13] 2023. UiPath Forum. <https://forum.uipath.com/> Accessed: March, 2023.
- [14] 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.
- [15] 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.
- [16] Alan F Blackwell. 2001. SWYN: A visual representation for regular expressions. In *Your wish is my command*. Elsevier, 245–XIII.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] Rui Dong, Zhicheng Huang, Ian Long Lam, Yan Chen, and Xinyu Wang. 2022. WebRobot: Web Robotic Process Automation using Interactive Programming-by-Demonstration. *arXiv preprint arXiv:2203.09993* (2022).
- [21] Sandra G. Hart and Lowell E. Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In *Human Mental Workload*, Peter A. Hancock and Najmedin Meshkati (Eds.). Advances in Psychology, Vol. 52. North-Holland, 139–183. [https://doi.org/10.1016/S0166-4115\(08\)62386-9](https://doi.org/10.1016/S0166-4115(08)62386-9)
- [22] Matthew Honnibal, Ines Montani, Matthew Honnibal, Henning Peters, Sofie Van Landeghem, Maxim Samsonov, Jim Geovedi, Jim Regan, György Orosz, Søren Lind Kristiansen, Paul O’Leary McCann, Duygu Altinok, Roman, Grégory Howard, Sam Bozek, Explosio Bot, Mark Amery, Wannaphong Phatthiyaphaibun,

- Leif Uwe Vogelsang, Björn Böing, Pradeep Kumar Tippa, jeannefukumaru, GregDubbin, Vadim Mazaev, Ramanan Balakrishnan, Jens Dahl Møllerhøj, wvseeker, Magnus Burton, thomasO, and Avadh Patel. 2019. *explosion/spaCy: v2.1.7: Improved evaluation, better language factories and bug fixes*. <https://doi.org/10.5281/zenodo.3358113>
- [23] Eric Horvitz. 1999. Principles of Mixed-Initiative User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Pittsburgh, Pennsylvania, USA) (CHI '99). Association for Computing Machinery, New York, NY, USA, 159–166. <https://doi.org/10.1145/302979.303030>
- [24] Rui Dong Xinyu Wang Yan Chen Tovi Grossman Kevin Pu, Rainey Fu. 2022. SemanticOn: Specifying Content-Based Semantic Conditions for Web Automation Programs. In *Proceedings of the 35th annual ACM symposium on user interface software and technology*. 563–576.
- [25] 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.
- [26] Rebecca Krosnick and Steve Oney. 2022. ParamMacros: Creating UI Automation Leveraging End-User Natural Language Parameterization. In *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 1–10. <https://doi.org/10.1109/VL/HCC53370.2022.9833005>
- [27] David Kurlander, Allen Cypher, and Daniel Conrad Halbert. 1993. *Watch what I do: programming by demonstration*. MIT press.
- [28] 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.
- [29] 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.
- [30] 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.
- [31] 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.
- [32] 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.
- [33] 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.
- [34] 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.
- [35] Toby Jia-Jun Li, Jingya Chen, Haijun Xia, Tom M. Mitchell, and Brad A. Myers. 2020. Multi-Modal Repairs of Conversational Breakdowns in Task-Oriented Dialogs. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) (UIST '20). Association for Computing Machinery, New York, NY, USA, 1094–1107. <https://doi.org/10.1145/3379337.3415820>
- [36] 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>
- [37] Toby Jia-Jun Li, Lindsay Popowski, Tom Mitchell, and Brad A. Myers. 2021. Screen2Vec: Semantic Embedding of GUI Screens and GUI Components. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM. <https://doi.org/10.1145/3411764.3445049>
- [38] 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.
- [39] Yang Li, Jiacong He, Xiaoxia Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping Natural Language Instructions to Mobile UI Action Sequences. *ArXiv abs/2005.03776* (2020).
- [40] Henry Lieberman. 1994. A user interface for knowledge acquisition from video. In *AAAI*. Citeseer, 527–534.
- [41] 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.
- [42] 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.
- [43] Karthik Mahadevan, Yan Chen, Maya Cakmak, Anthony Tang, and Tovi Grossman. 2022. Mimic: In-Situ Recording and Re-Use of Demonstrations to Support Robot Teleoperation. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–13.
- [44] 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.
- [45] 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.
- [46] 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.
- [47] 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.
- [48] 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.
- [49] 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.
- [50] Don Norman. 2013. *The design of everyday things: Revised and expanded edition*. Basic books.
- [51] 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.
- [52] 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).
- [53] Panupong Pasupat, Tian-Shun Jiang, Evan Zheran Liu, Kelvin Guu, and Percy Liang. 2018. Mapping Natural Language Commands to Web Elements. *arXiv:1808.09132* [cs.CL]
- [54] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. *arXiv:2103.00020* [cs.CV]
- [55] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).
- [56] Alborz Rezaadeh Sereshkeh, Gary Leung, Krish Perumal, Caleb Phillips, Minfan Zhang, Afsaneh Fazly, and Iqbal Mohamed. 2020. VASTA: a vision and language-assisted smartphone task automation system. In *Proceedings of the 25th international conference on intelligent user interfaces*. 22–32.
- [57] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of Bits: An Open-Domain Platform for Web-Based Agents. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 3135–3144. <https://proceedings.mlr.press/v70/shi17a.html>
- [58] 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.
- [59] 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.
- [60] 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.
- [61] Bryan Wang, Gang Li, and Yang Li. 2022. Enabling Conversational Interaction with Mobile UI using Large Language Models. *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (2022).
- [62] Bryan Wang, Gang Li, Xin Zhou, Zhouong Chen, Tovi Grossman, and Yang Li. 2021. Screen2Words: Automatic Mobile UI Summarization with Multimodal Learning. *The 34th Annual ACM Symposium on User Interface Software and Technology* (2021).
- [63] 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.
- [64] 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.
- [65] Nanxuan Zhao, Nam Wook Kim, Laura Maria 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

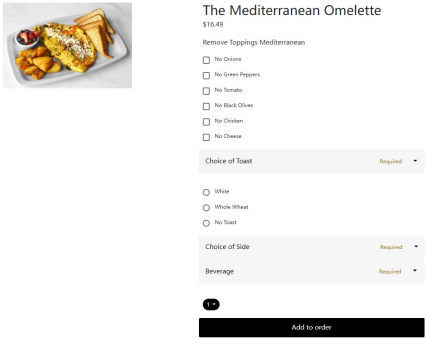
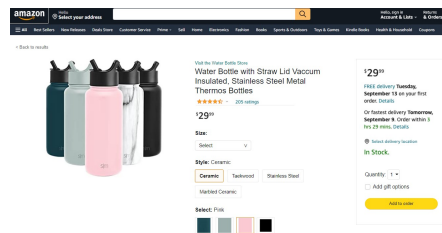
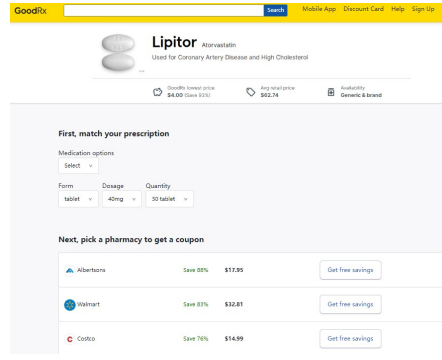
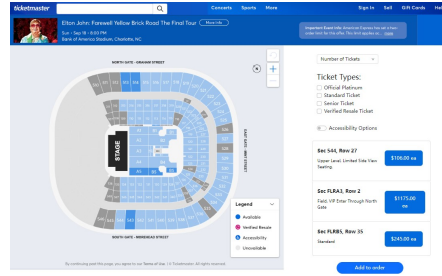
#	Task Description	Website Example																
1	<p>UberEats (Dining)</p> <p>Task Description: Order food from different restaurants with various dietary restrictions, side dish choices and other customization.</p> <p>Example Request: “Restaurant”: [“Brunch Alley”], “Food”: [“Greek Omelette”], “Memo”: [“lactose intolerant option, choose whole wheat, and add tea”]</p>	 <p>The Mediterranean Omelette \$16.49</p> <p>Remove Toppings: Mediterranean</p> <p><input type="checkbox"/> No Onions <input type="checkbox"/> No Green Peppers <input type="checkbox"/> No Tomatoes <input type="checkbox"/> No Black Olives <input type="checkbox"/> No Chicken <input type="checkbox"/> No Cheese</p> <p>Choice of Toast <span>Required</span></p> <p><input type="radio"/> White <input type="radio"/> Whole Wheat <input type="radio"/> No Toast</p> <p>Choice of Side <span>Required</span></p> <p>Beverage <span>Required</span></p> <p>1.5</p> <p>Add to order</p> <p>Order food with customization</p>																
2	<p>Amazon (Shopping)</p> <p>Task Description: Add items to order based on shopping list with different selections on size, color, style, etc.</p> <p>Example Request: “Query”: [“merchandise”], “Product”: [“water bottle”], “Condition”: [“choose stainless steel style, pick bright color if there is one, add gift”]</p>	 <p>Water Bottle with Straw Lid Vacuum Insulated, Stainless Steel Metal Thermos Bottles</p> <p>Size: <span>Select</span></p> <p>Style: Ceramic <span>Ceramic</span> <span>Teknolux</span> <span>Stainless Steel</span></p> <p>Unlabeled Ceramic</p> <p>Select: Pink <span>Pink</span> <span>Blue</span> <span>Green</span> <span>Black</span></p> <p>Shop item with selected color and material</p>																
3	<p>GoodRx (Healthcare)</p> <p>Task Description: Compare drug prices and find coupons with given list of prescription selections</p> <p>Example Request: “Drug”: [“Lipitor”], “Prescription”: [“choose brand one, pick 10mg dosage, have 60 tablets pack, use Walmart pharmacy”]</p>	 <p>Lipitor Atorvastatin Used for Coronary Artery Disease and High Cholesterol</p> <p>Medication options Select <span>▼</span></p> <p>Form: tablet Dosage: 40mg Quantity: 30 tablet</p> <p>Next, pick a pharmacy to get a coupon</p> <table> <tr> <th>Pharmacy</th><th>Save</th><th>Price</th><th>Get free savings</th></tr> <tr> <td>Albertsons</td><td>Save 85%</td><td>\$17.85</td><td>Get free savings</td></tr> <tr> <td>Walmart</td><td>Save 81%</td><td>\$12.81</td><td>Get free savings</td></tr> <tr> <td>Costco</td><td>Save 76%</td><td>\$14.89</td><td>Get free savings</td></tr> </table> <p>Find selected medicine coupon</p>	Pharmacy	Save	Price	Get free savings	Albertsons	Save 85%	\$17.85	Get free savings	Walmart	Save 81%	\$12.81	Get free savings	Costco	Save 76%	\$14.89	Get free savings
Pharmacy	Save	Price	Get free savings															
Albertsons	Save 85%	\$17.85	Get free savings															
Walmart	Save 81%	\$12.81	Get free savings															
Costco	Save 76%	\$14.89	Get free savings															
4	<p>Ticketmaster (Entertainment)</p> <p>Task Description: Book event based on provided category and requirements like number of tickets, preference for seating, etc.</p> <p>Example Request: “Type”: [“concert”], “Event”: [“Elton John”], “Specification”: [“select standard ticket, and also pick senior ticket, get 2 tickets, and select side-view seating”]</p>	 <p>Elton John: Farewell Yellow Brick Road The Final Tour</p> <p>Number of Tickets: <span>1</span></p> <p>Ticket Types: <input type="checkbox"/> Official Platinum <input type="checkbox"/> Standard Ticket <input type="checkbox"/> Senior Ticket <input type="checkbox"/> Verified Resale Ticket</p> <p>Accessibility Options</p> <p>Row 27 Upper Level Limited View From Behind \$110.00 per ticket</p> <p>Row 2 Lower Level Limited View From Behind \$117.50 per ticket</p> <p>Row 35 Standard \$24.50 per ticket</p> <p>Add to order</p> <p>Book event with preferred seating</p>																

Figure 7: User study task descriptions with example web page UI.