From Operation to Cognition: Automatic Modeling Cognitive Dependencies from User Demonstrations for GUI Task Automation

Yiwen Yin
Department of Computer Science and Technology
Tsinghua University
Beijing, China
yinyw21@mails.tsinghua.edu.cn

Yu Mei
Department of Computer Science and Technology
Tsinghua University
Beijing, China
meiy24@mails.tsinghua.edu.cn

Chun Yu*
Department of Computer Science and Technology
Tsinghua University
Beijing, China
chunyu@tsinghua.edu.cn

Toby Jia-Jun Li
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, Indiana, USA
toby.j.li@nd.edu

Aamir Khan Jadoon
Department of Computer science and Technology
Tsinghua University
Beijing, China
qinl19@mails.tsinghua.edu.cn

Sixiang Cheng
Department of Computer Science and Technology
Tsinghua University
Beijing, China
chengsx21@mails.tsinghua.edu.cn

Weinan Shi
Department of Computer Science and Technology
Tsinghua University
Beijing, China
swn@tsinghua.edu.cn

Mohan Chen
Department of Computer Science and Technology
Tsinghua University
Beijing, China
cmh21@mails.tsinghua.edu.cn

Yuanchun Shi
Department of Computer Science and Technology
Tsinghua University
Beijing, China
Qinghai University
Xining, Qinghai, China
shiyc@tsinghua.edu.cn

Abstract

Traditional Programming by Demonstration (PBD) systems primarily automate tasks by recording and replaying operations on Graphical User Interfaces (GUIs), without fully considering the cognitive processes behind operations. This limits their ability to generalize tasks with interdependent operations to new contexts (e.g. *collecting and summarizing introductions depending on different search keywords from varied websites*). We propose TaskMind, a system that automatically identifies the semantics of operations, and the cognitive dependencies between operations from demonstrations, building a user-interpretable **task graph**. Users modify this graph to define new task goals, and TaskMind executes the graph to dynamically generalize new parameters for operations, with the integration of Large Language Models (LLMs). We compared TaskMind with a baseline end-to-end LLM which automates tasks from demonstrations and natural language commands, without task graph. In studies with 20 participants on both predefined and customized tasks, TaskMind significantly outperforms the baseline in both success rate and controllability.

*Corresponding author.

This work is licensed under a Creative Commons Attribution 4.0 International License.

CHI '25, Yokohama, Japan

© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-1394-1/25/04
https://doi.org/10.1145/3706598.3713356

CCS Concepts

• **Human-centered computing** → **Interactive systems and tools**.

Keywords

Programming by Demonstration, Task Automation, Task Graph, Cognitive Dependency

ACM Reference Format:
Yiwen Yin, Yu Mei, Chun Yu, Toby Jia-Jun Li, Aamir Khan Jadoon, Sixiang Cheng, Weinan Shi, Mohan Chen, and Yuanchun Shi. 2025. From Operation to Cognition: Automatic Modeling Cognitive Dependencies from User Demonstrations for GUI Task Automation. In *CHI Conference on Human Factors in Computing Systems (CHI '25), April 26–May 01, 2025, Yokohama, Japan.* ACM, New York, NY, USA, 24 pages. https://doi.org/10.1145/3706598.3713356

1 Introduction

Repetitive and routine tasks on GUIs, characterized by fixed procedures, can be highly time-consuming. Previous studies show that a large proportion of GUI-based tasks across various domains can be automated [1, 4, 39]. Our preliminary survey, conducted with 81 participants from the local university community via a questionnaire, revealed that approximately 38.8% of daily tasks in work or study are automatable. Among the various task automation techniques, Programming by Demonstration (PBD) stands out by significantly lowering the barriers for users to create automation

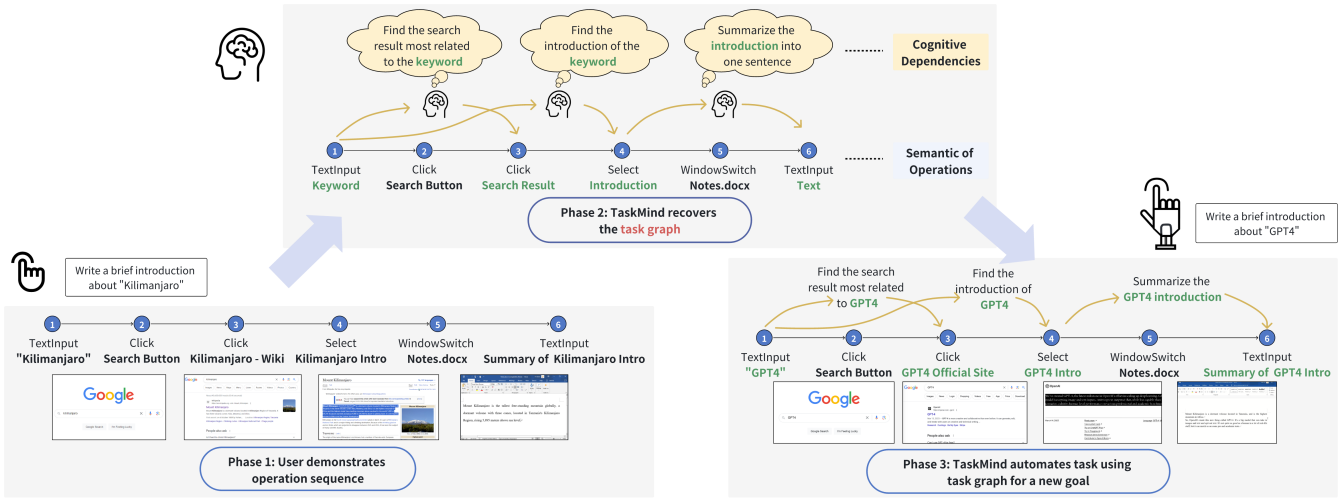

Figure 1: Using TaskMind to demonstrate and automate an example task (Task 16 in our dataset). In the demonstration phase, the user interacts with the GUI to search for and summarize a keyword's introduction. TaskMind models this task by recovering 3 cognitive dependencies between operations, which are the implicit cognitive processes users undergo when selecting parameters for operations. During automation, TaskMind executes the task graph to infer new parameters with the guidance of cognitive dependencies, generalizing the demonstration to new goals and contexts (i.e. when the search keyword and websites change).

programs [13, 29]. PBD systems enable users to manually demonstrate the task process, which contains a sequence of operations along with their parameters (i.e., the targets of operations, such as the *'Search'* button to click or the text *'Kilimanjaro'* to input in Figure 1). From the demonstration, the systems generate a task program to automate for the same or similar task goals. PBD offers an effective method for end-users without programming expertise to create personalized and customized automation [13, 29, 45].

However, existing PBD tools [25, 29, 30, 42, 45, 57] mainly focused on recording and replaying sequence of operations, overlooking the trajectory of intents that guide parameter selection, which often involve cognitively "computing" the relationships between prior parameters. These processes occur internally within users' mind and are implicit in the observed GUI operations. We refer to them as **cognitive dependencies**, which describe the cognitive processes users undergo when selecting parameters depending on prior operations. Cognitive dependencies are prevalent in various tasks, for example, as illustrated in Figure 1, consider a task of browsing and writing a brief introduction for a keyword. During the demonstration, the user first searches for a keyword, then clicks a search result with the cognitive dependency *"find the search result most relevant to the keyword"*, which guides them to select the most relevant Wikipedia link. Similarly, the user then selects a paragraph from the page with *"find the introductory paragraph of the keyword"*, and types a summary in the document with *"summarize the previously selected paragraph"*. When the task goal (the keyword) changes, subsequent parameters (the chosen search result, selected paragraph, and final input summary) should also change, following the same cognitive dependencies. Without understanding these dependencies, PBD systems cannot adjust the subsequent parameters when the initial keyword changes during automation, resulting in failures to generalize the demonstrated task program to different task goals and contexts.

To bridge this operation-cognition gap, we present TaskMind, the first PBD system that uses the reconstruction and interactive disambiguation of **cognitive dependencies** to model user intents behind operations for the generalization of learned task programs. TaskMind automatically recovers cognitive dependencies from user demonstrations by analyzing the relationship between parameters in operation sequence with the help of LLMs. These dependencies are explicitly annotated in a **task graph**, which serves as the automation program, with its dependencies used as prompts for LLMs to infer new parameters for different task goals, enhancing the system's generalizability to new contexts for complex tasks with inter-dependent parameters. TaskMind features an interactive interface allowing users to check and modify the generated task graphs with cognitive dependencies to correct errors and define new task goals before starting the automation.

Our formative study identified several challenges in building an abstract task graph from low-level GUI operations, including the gap between raw operations and their semantics, and the diversity of cognitive dependencies involved in GUI tasks. These findings led to the design of TaskMind's two new components for task analysis: a UI extractor that transforms low-level GUI events into semantically rich operations, and a dependency extractor that leverages LLMs to recognize cognitive dependencies. An operation-cognition knowledge base was also derived from the formative study to provide abstract categorization of dependencies, supporting LLMs in dependency extraction. Additionally, we introduce a self-evolving mechanism which employs a LLM-driven reflection module running in the background to take user modifications as input and update the knowledge base accordingly.

Our simulation study showed TaskMind's effectiveness in generalizing new parameters with a task success rate (SR) of 79.5% and a step SR of 93.9% before user modifications, compared to 33.3% task SR and 83.2% step SR for TaskMind without explicitly annotated cognitive dependencies as our baseline. Additional simulation studies emphasized the importance of the operation-cognition knowledge base and the reflection module. In our within-subjects user study involving 20 participants on both pre-defined and customized tasks, TaskMind significantly outperformed the baseline in both effectiveness and controllability. We discussed the effort-controllability trade-off, where using natural language (NL) instructions for the baseline is more intuitive but harder to specify intents, while modifying task graphs requires more logical thinking but offers better control through precise edits.

In conclusion, our core contributions are:

1. The design of a task graph with cognitive dependencies to enhance PBD generalizability by modeling implicit cognitive processes underlying operation sequences.
2. TaskMind, a novel PBD system that automatically reconstructs cognitive dependencies from demonstrations, supported by an operation-cognition knowledge base with a self-evolving mechanism.
3. A simulation study to assess the effectiveness of TaskMind in automation success rate compared to a version without cognitive dependencies.
4. A user study comparing the usability of modification through task graph versus NL instructions, highlighting the balance between effort and controllability in collaboration.

2 Related Work

2.1 Task Automation

Task automation employs scripts or tools to automate repetitive and tedious tasks through GUI on behalf of users, reducing manual labor and enhancing human productivity. The automation scripts typically originate from four sources: machine language programming such as Shell or Python, visual programming with drag-and-drop interfaces [43, 53, 57], NL instructions, and user demonstration, each requiring different levels of user expertise. PBD stands out as a method where the system learns tasks by observing user demonstration on the UI and can generalize the process, offers greater adaptability for personalized tasks and intuitiveness for non-programmers [3, 8, 13, 22, 26, 27, 29, 30, 32, 33, 35, 42, 47, 52]. Recently, the advent of natural language processing and pre-trained LLMs has led to the development of technologies capable of converting NL instructions into GUI operations [11, 14, 20, 24, 34, 46, 61], lowering the barrier for task automation.

Most existing automation tools focus on mobile platforms [29, 30, 33, 46], and web automation for specific use cases such as data collection and form filling [7, 10, 45]. Desktop-based systems are limited to single applications [27], or struggle with complex tasks that involve intricate intentions [21, 22]. TaskMind is a PBD-based automation tool designed for personalized, repetitive, or routine tasks. It incorporates LLMs to better understand user demonstrations with complex cognitive processes, offering a more flexible PBD solution.

2.2 GUI Understanding

Understanding GUIs is the foundation of PBD, focusing on interpreting task goals and automating operations intelligently. Previous research [50, 55, 58] used accessibility data and CV techniques to comprehend UI elements on mobile devices, while others [3, 37, 57] recorded UI elements on desktop platforms. Semantic vectors have also been employed to interpret UIs and elements [2, 15, 31]. However, these approaches concentrated on the basic information of UI elements, overlooking contextual data. A recent method [54] leverages LLMs to understand contextual information within mobile UIs, but this approach struggles with the complexity of PC interfaces and is time-consuming for automating operation sequences. TaskMind enhances the semantics of computer elements by collecting surrounding UI information. Moreover, TaskMind can identify and extract list structures from the interface, recognizing iterative tasks from a single demonstration, which is an improvement over prior works [7, 36] that focused mainly on web and specific tasks.

2.3 Task Modeling for Generalization in PBD and Integration of LLMs

The generalization of task scripts is a major challenge for PBD[13, 29, 35]. Traditional systems, which primarily rely on simple record and replay mechanisms [3, 26, 27, 47, 52], are limited in their ability to adapt recorded scripts to new, varying contexts. In contrast, abstractly modeling operations and task intentions from demonstrated sequences offers a viable solution for generalizing tasks. Existing task modeling methods can be categorized into three main approaches: (1) **Parameter extraction**, where systems identify generalizable parameters by matching voice commands with operation sequences [29, 33, 42], recognizing looping targets for repetitive tasks [21, 22], or specifying semantic filters for data scraping tasks [44]. (2) **Description of individual operations**, as seen in APPINITE [30] and PUMICE [32], which learn data descriptions of operations and task conditions from user manual input. (3) **Path Modeling**, like KITE [33], which constructs a task model from multiple demonstrations to cover all potential execution paths of a task. However, these methods typically require additional user input to clarify intentions, such as voice commands [29, 42], manual configurations of data descriptions[30], or follow-up questions[22]. In addition, they simply focus on generalizing independent operations. Therefore, more interactions are required when tasks involve multiple generalizable parameters with dependencies between them, making the creation of generalizable automation scripts more effort-intensive even than manually completing them.

Large Action Models (LAMs) leverage pre-trained LLMs or LVMs (large vision models) to generate action sequences from NL instructions [11, 14, 20, 24, 61]. They model task automation as a problem of predicting the next action based on instructions and current contexts, with their general operational knowledge embedded in the foundation models. While LAMs are not explicitly used in PBD and lack support for personalized tasks, they demonstrate the potential for using LLMs to interpret and generalize user operations. Our approach, TaskMind, extends traditional PBD by integrating LLMs to improve generalization. We use a task graph with cognitive dependencies to specify cognitive dependencies between operation

parameters, which LLMs automatically extracted from user demonstrations. During automation, these dependencies guide the use of LLM intelligence to generate parameters in new contexts. By leveraging LLMs, TaskMind enhances the traditional symbolic PBD framework, making it effective for tasks with multiple interdependent generalizable parameters.

2.4 Cognitive Processes in GUI Tasks

Cognitive dependency, a concept we introduce, describes the cognitive process user undergo when performing GUI tasks. In this process, parameters from prior operations are used to inform subsequent ones, such as summarizing previously selected text into a sentence input in the next step. Existing research on human cognitive processes, primarily in psychology and education, includes Bloom's Taxonomy [5], which categorized human cognitive skills into six levels. Marzano & Kendall's Framework[38] further improved this taxonomy. Multiple Intelligences Theory [6] modeled the cognition when humans perform information processing. Additionally, some studies [51] explored cognitive dependencies in language from a cognitive linguistic perspective. Jerry A. Fodor's work [17] advocates for an atomistic view of concepts, emphasizing simple, fundamental relationships between the mind and the world. Moreover, Fodor [16] highlights that thought is intrinsically ordered, operating through systematic, sequential, and rule-based patterns similar to a language. TaskMind transplants cognitive theories into the domain of GUI tasks, introduces an operation-cognition knowledge base concentrating on the cognitive-driven relationships between operations. With the help of the knowledge base, it recovers and models human cognitive dependencies implicit in operation sequences, structuring them into the task graph. This feature allows TaskMind to mimic human thinking by adapting cognitive dependencies to infer new parameters in new contexts, enhancing the generalization capability of PBD.

3 Formative Study

To better understand the gap between system-observed operations and users' actual intentions, we conducted a formative study focused on collecting real-world examples of repetitive and routine tasks encountered by users during computer usage. We constructed a dataset from the collected data, and the several challenges revealed by the analysis lead to the design of our task graph.

3.1 Procedure

Our study involved 21 participants aged 19-37 (μ=29.0, σ=6.6), including 8 academic researchers, 7 undergraduates from various majors, 4 corporate or government office workers, and 2 software developers. Participants installed our data collection software, which recorded their screenshots, keyboard and mouse activities using Windows hooks [40] and IUIAutomation APIs [41]. Participants were instructed to record their computer usage as usual throughout a week, with the flexibility to activate or deactivate the recording software at their discretion. They were informed that the goal of the study was to observe their daily activities in order to identify tasks suitable for automation. Additionally, they were briefed about the potential privacy implications and provided informed consent. Participants were compensated at a rate of $5 per hour for recording[1]. After the week of recording, participants engaged in informal interviews, during which they described any repetitive and tedious tasks they experienced in their computer use[2].

Consequently, only 3 participants were comfortable recording one hour per day throughout the entire week, mostly focusing on continuous tasks such as writing a paper. In contrast, 12 participants, due to privacy concerns, recorded only specific tasks they considered relevant for automation, resulting in shorter sessions about one hour per participant, involving more targeted workflows such as a secretary's work of filling out web forms. This variation in recording behavior contributed to a total of 51.3 hours of recorded data.

For data analysis, two experts with deep understanding of cognitive dependencies reviewed the recorded data along with the interview results. The review process is as follows: First, they examined the recorded videos with mouse and keyboard events, identified sequences of operations and segmented them into distinct tasks. Second, with insights from informal interviews, they identified tasks suitable for automation based on three shared criteria: a) a clear goal; b) a fixed sequence of operations; c) a repetitive, tedious nature. Third, they annotated the cognitive dependencies in these tasks, and employed the open coding method[18] to summarize the types of dependencies. Each expert individually analyzed all 51.3 hours of recorded data, then compared and merged their results to reach a consensus. By considering participants' interview results, referencing shared criteria, and merging opinions of different experts, biases from individual interpretations were minimized.

3.2 Findings

From a total of 51.3 hours of recordings, we identified 170 distinct user tasks, accounting for 96.8% of all the recording time. Among them, we selected 112 tasks that were suitable for automation, making up 65.9% of all the distinct tasks, and 59.1% of the total recording time. Tasks requiring extensive user creativity or lacking a clear goal, such as writing travel plans, replying to messages and browsing websites, were excluded. The proportion of automation tasks was higher than the 38.8% reported in the preliminary study in the introduction, likely due to participants' selective recording of tasks they considered relevant for automation. Finally, we grouped the tasks with similar procedures or goals into 39 task groups. For each group, we selected a representative task as an exemplar, constructing a dataset with 39 tasks in total (detailed in Appendix E)[3]. Among them, 10 tasks were related to student learning, 9 to academic research, 9 to clerical data processing, 6 to personal file management, and 5 to software development or testing.

The repetitive or routine tasks featured several iterations of the same operation path. To minimize the demonstration costs while achieve effective generalization, our design goal was to model the cognitive dependencies into a task graph from a single iteration of user demonstration. Our analysis revealed several challenges in reconstructing cognitive dependencies, as explained in the following sections.

[1]The compensation was consistent with the average earning of workers in the community where the study took place.
[2]The study protocol was reviewed and approved by the university ethics review board.
[3]https://github.com/Evennaire/TaskMind

Table 1: Operation-cognition knowledge base containing 5 main categories with proportion in the dataset, and their subcategories with definitions and examples of NL descriptions.

Main Category	Subcategory	Definition	Examples
Information Recall (34.0%)	Repeat	Select or input something based on prior same or similar information	Input the same text Select the element with similar text
	Format Text	Changing the format, structure, or presentation while keep content	structure previous data into specific format Replace suffix "xlsx" by "csv"
	Polish Text	Enhancing the quality or Refining the style of text	Use a more formal tone Eradicate ambiguous expressions
Information Comprehension (28.3%)	Summarize	Understanding the information and give a summary	Summarize the keywords Summarize into one sentence
	Extract Information	Understanding existing information and extract specific data	Extract student id from
Information Creation (9.4%)	Generate Text	Generating newly written content	Write a short essay based on the keyword Write an application for the post
Logical Reasoning (13.2%)	Calculate	Performing mathematical calculations based on existing data	add two numbers
	Evaluation	Analyzing data to produce a result, comparison, or categorization	compare two strings and give the result categorizing data based on certain criteria
Contextual Selection (15.1%)	Match Element	Identifying elements based on current context	Select item with the lowest price Find the search result most related to
	Match Text	Identifying text based on current context	Select the movie introduction on the webpage

3.2.1 ***Identifying iterative tasks from single demonstration.*** Our analysis revealed two types of tasks: (1) **Normal Tasks** with a linear structure users may perform periodically. (2) **Iterative Tasks** with a recurrent structure, where users repeatedly perform the same operations on a list of objects. In our dataset of 39 tasks, 23.1% (9 tasks) were identified as normal, while 76.9% (30 tasks) were iterative. In comparison, in all recorded distinct tasks during the study, normal tasks accounted for 52.9% and iterative tasks accounted for 47.1%, with a higher proportion of normal tasks such as replying to messages. A key feature of iterative tasks is that at least one operation involves an element parameter within a list on the GUI, with subsequent iterations using other elements in the list as new parameters. To identify the task type from a single demonstration, it is crucial to recognize list structures on the UI.

3.2.2 ***Enriching GUI operation semantics through contextual information.*** We identified a significant gap between raw GUI events and the semantics of operations. For instance, an input field on a webpage may not indicate its intended use due to insufficient element labels. GUI structure data provides contextual clues about surrounding elements like adjacent text labels, clarifying the purpose of user interactions with them. Integrating these supplementary data with UI analysis techniques is crucial for accurate inference of the semantics behind individual operations.

3.2.3 ***Understanding multiple sources of intent in parameter selection.*** For each operation, we explored the decision-making process involved in the selection of parameter, i.e., why users interact with specific elements or input certain text. Based on the source of intent, we identified several types of operations:

(1) **Fixed**-Parameter Operations: Operations where parameters remain constant across different contexts. In the example of Figure 1, the search button to click and the MS Word window to switch are fixed, regardless of the keyword.
(2) **Independent** Variable-Parameter Operations: Operations where parameters are directly influenced by the user's direct intent, which are also essential in determining the task goal. For example, in Figure 1, the keyword to search is a user-specified parameter. In iterative tasks, such parameters may vary across iterations and are determined by the iterative element list as mentioned in 3.2.1, and we further label them as **Iterative**.
(3) **Dependent** Variable-Parameter Operations: Operations where parameters are influenced by prior operations and the user's cognitive process. For instance, in Figure 1, the final summary depends on the selected paragraph. Each dependent operation involves a cognitive dependency, linking it to prior operations.

3.2.4 ***Recognizing diverse and task-specific cognitive dependencies.*** We categorized the cognitive dependencies of each dependent operations into distinct types, based on the cognitive processes users undergo when interacting with the UI. Firstly, these dependencies can be classified into two primary types:

- Structure Dependency, which involves understanding UI structural relationships, such as identifying the same or adjacent elements. These dependencies rely on rules with basic element information, rather than semantic understanding.
- Semantic Dependency, which involves cognitive processes that deal with semantic connections between text or element parameters. Examples include summarizing text or identifying elements with related text.

By adapting existing cognitive categorization methods, including Bloom's Taxonomy [5] and Marzano & Kendall's framework [38] to the realm of GUI operations, we divided all the semantic dependencies into five main categories. These include **Information Recall, Information Comprehension, Information Creation, Logical Reasoning, and Contextual Selection**. The main categories are designed to be both abstract and comprehensive, encompassing the full range of semantic cognitive processes during GUI operations. Furthermore, each main category contains several subcategories, which are more specific breakdowns of a main category to represent a group of similar semantic dependencies. See Table 1 for a complete list of all categories, which we term as an **operation-cognition knowledge base**. Notably, our task automation scope

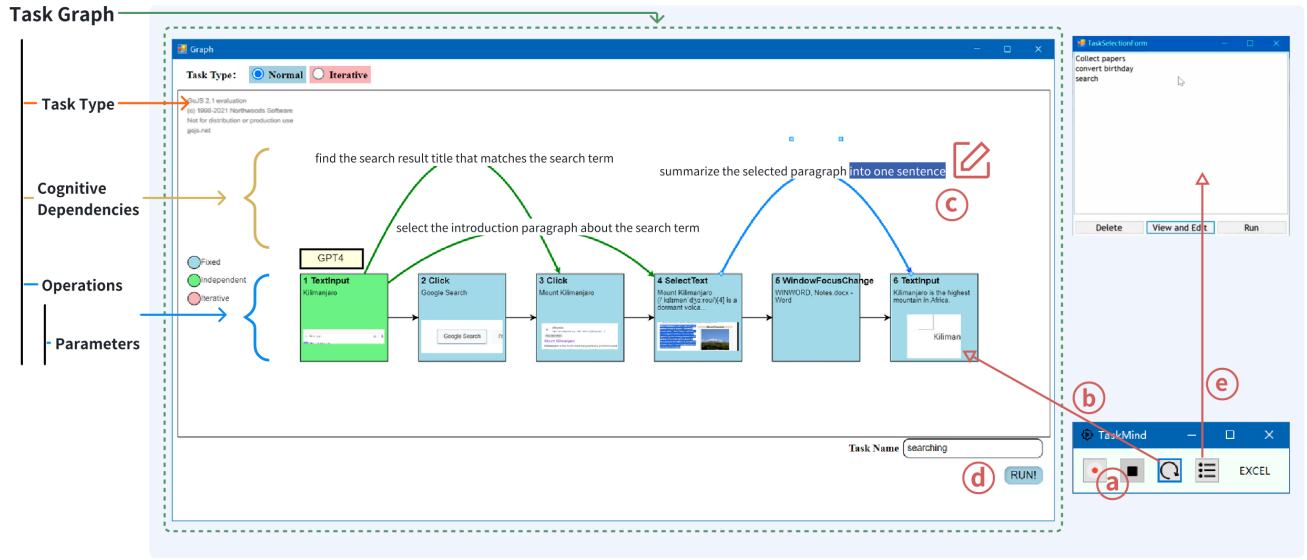

Figure 2: An example of a task graph on the UI, and the interaction steps with TaskMind to automate tasks: (a) record and stop demonstration, (b) allow TaskMind to analyze the task, (c) confirm or modify the task graph, (d) save and execute the task graph, (e) invoke task graphs in the repository.

excludes cognitive processes like image/video understanding and emotional reasoning, which are beyond automation needs.

We observed a diversity in semantic dependencies, many of which are task-specific. For instance, in our dataset, Tasks 26 and 27 both involve writing based on slides but differ in goals: Task 26 focuses on personal reflections, while Task 27 involves drafting a lecture. This highlights the necessity for a comprehensive knowledge base to capture diverse semantic dependencies. However, there is a trade-off between abstraction and specificity. The knowledge base must be abstract enough to cover many tasks, yet specific enough to address the unique cognitive dependencies within individual tasks. Thus, a mechanism is required to ensure that the knowledge base can adapt and evolve continuously to meet varied and specific demands of different tasks and users.

3.3 Task Graph Design

Based on the findings, we designed **task graph** to model cognitive dependencies within GUI tasks. A task graph serves as a task program, encompassing all the requisite information for automated execution. It comprises three components: task type, operation sequence, and cognitive dependencies. An example task graph on the UI is shown in Figure 2.

Task Type. Normal or iterative. See 3.2.1 for details.

Operation Sequences. Represented by nodes in the graph, which are sequentially connected with edges, depicting the task flow. Each operation has the following attributes:

- Index of Operation.
- Type of Operation: We defined 21 common computer operations along with their corresponding parameters, including Click, TextInput, Copy, etc. See Appendix F.
- Parameters: The target of an operation, which mainly include text, and elements on the GUI. An element is represented by a dictionary with fields including ID, Name, Type, Text, Description (contextual information to enrich the semantics, see 3.2.2) and List (the list in GUI where the element resides, if present, see 3.2.1). Operations may have no parameters or multiple parameters.
- Source of Parameter: Fixed, independent/iterative or dependent, see 3.2.3 for details.

Cognitive Dependencies. Represented by special edges in the graph. When operation B's parameter depends on operation A, a directed edge links A to B. Operation A serves as the input, and B as the output of this dependency. A cognitive dependency may have multiple inputs, but only one output. While a task might follow various dependency paths, a task graph represents just one of these paths. The text format for a semantic dependency is:

`Output=Main_Category.Subcategory<Description>(Inputs)`.

"Main Category" and "Subcategory" come from 3.2.4, and "Description" provides detailed description of the dependency in NL.

4 TaskMind

In this section, we first describe the user interaction scenario along with the system workflow of TaskMind, then present the design and implementation details of TaskMind, addressing challenges identified in our formative study.

4.1 Usage Scenario

Anna, a university student studying Economics, often encounters unfamiliar terms while reading textbooks and blogs. To build a personal glossary, she wants to search for and browse each term, then

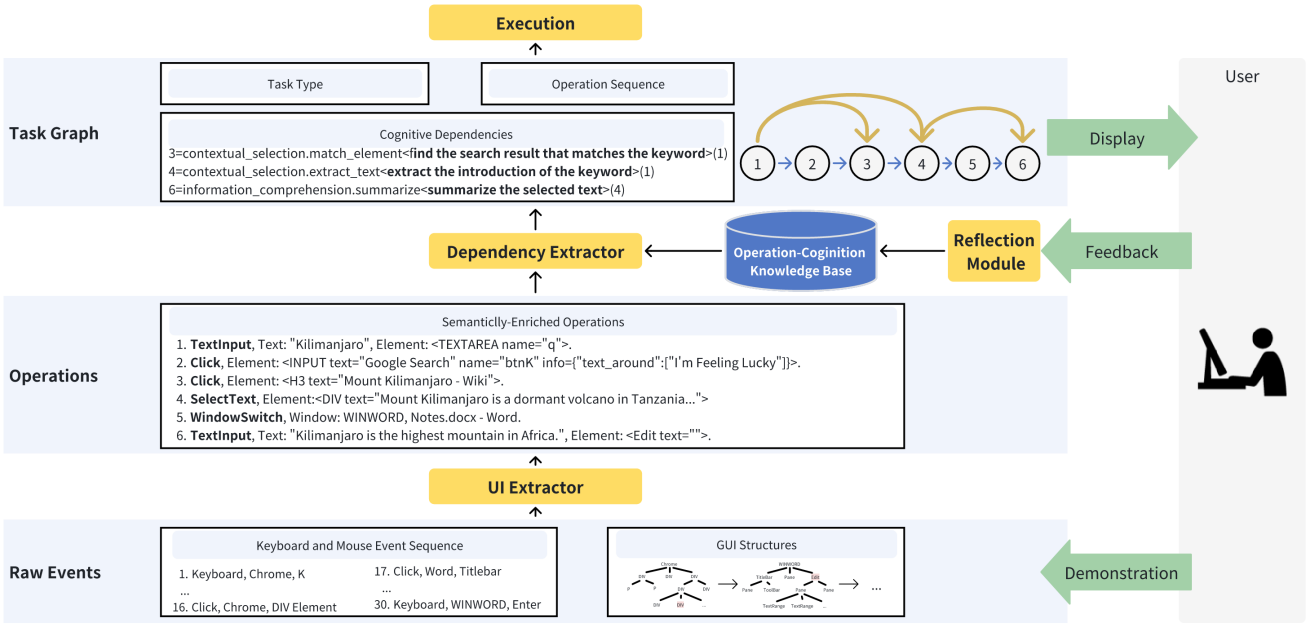

Figure 3: The overview of TaskMind system, which automatically models a task graph from the user demonstration.

write a brief summary. She decides to use TaskMind to automate this repetitive process.

Demonstration, Modification & Automation. Anna begins by demonstrating how to search the keyword *"Kilimanjaro"*. She clicks the "Start Recording" button, performs her usual process, and clicks "Stop Recording" (Fig. 2.a) and then "Analyze" (Fig. 2.b). TaskMind analyzes the recordings and generates a task graph. Anna can check the graph, which visualizes operations and their cognitive dependencies (edges with NL descriptions). She can modify the dependencies by adding, deleting, or editing their descriptions and the connected nodes (Fig. 2.c). Anna saves the graph to the task repository without modifications (Fig. 2.d). Later, Anna encounters another term *"GPT-4"*. She opens the saved task (Fig. 2.e), enters *"GPT-4"* on the first node to define a new task goal, and clicks "Run" to automate the task (Fig. 2.d). However, during automation, TaskMind gets stuck when trying to select the *"GPT-4"* introduction on the page. Anna presses "Shift + Esc" to stop the process, and re-check the task graph. She identifies an unrecognized dependency for the operation of selecting the introduction, which causes the failure, as TaskMind still tries to find the *"Kilimanjaro"* introduction on the *"GPT-4"* website. Anna adds a new dependency—*"Find the introduction of the keyword"*—and clicks "Run" to restart the automation (Fig. 2.c). With this adjustment, TaskMind successfully completes the task. From then on, whenever Anna encounters an unfamiliar term, she can simply input the new keyword, and have TaskMind automatically add its summary in her notes.

Reflection & Evolution. Once the modification is made, the reflection module analyzes it in the background, learns that *"extracting specific information"* is a type of cognitive dependency, and incorporates this knowledge into the initial operation-cognition knowledge base which supports dependency extraction. Next time when Anna creates new tasks, such as collecting abstracts of papers, TaskMind can more accurately identify the similar cognitive dependency *"find the abstract of the paper"* from the demonstrated operations.

Iterative Task. Now Anna has a list of papers to read and summarize but limited time. She creates an Excel file to track the papers and demonstrates how to search for and summarize the abstract for the first paper. TaskMind identifies the papers to iterate over, displays them in the task graph along with cognitive dependencies. Anna clicks "Run" after checking it, and TaskMind automatically collects and summarizes abstracts for the remaining papers.

4.2 Design and Implementation

TaskMind is implemented as a Windows WinForm application. During the user demonstration, it captures screen, keyboard, and mouse event data, along with UI structures using Windows hooks, accessibility APIs, and a Chrome extension for web page events and DOM trees. When analyzing the task, two key modules, the UI extractor and the dependency extractor, process the record data into a task graph. In the automation phase, TaskMind executes the task by simulating user keyboard and mouse inputs across Microsoft Office (Excel, Word, PowerPoint), Chrome browser, and the system file explorer. The system architecture is shown in Figure 3, and main modules are detailed below.

4.2.1 UI Extractor. The UI Extractor is designed to enhance the semantics of individual operations by extracting contextual information, and meanwhile identify iterative tasks by recognizing iterative UI elements.

Firstly, the UI Extractor uses a rule-based method to transform low-level GUI events into different types of operations along with their targeted parameters (element or text) (see operation types in Appendix F). The rules include predefined event patterns like grouping keyboard inputs as TextInput or recognizing shortcuts. Secondly, for each element parameter, the UI Extractor parses the UI structure to retrieves fundamental attributes such as ID, type, name, and text, with its additional information enriched through two steps: (1) *Semantics Augmentation:* It extracts text from parent and sibling elements in the UI structure as contextual information to save into the 'Description' field, enriching the semantics of the corresponding operation. For spreadsheet cells, it also captures the headers and data in the same row. (2) *Iteration Recognition:* It identifies the list on the screen where the element resides (if exists), and save necessary information into the 'List' field. This is implemented using a method inspired by Rousillon's [7] work to identify the Deepest Iterative Ancestor (DIA) of an target element in the UI structure. The siblings of the DIA are regarded as iterative. The 'List' field contains DIA's parent and sibling elements, the index within its siblings, and the path from the DIA to the target element. This list information will be used to determine new iteration goals during subsequent automation for iterative tasks, and also identify structure dependency by discerning whether two elements are in the same list.

Overall, these steps convert low-level GUI events into semantically-enriched operations, aiding in the subsequent analyzing of cognitive dependencies.

4.2.2 Dependency Extractor. The dependency extractor aims to uncover cognitive dependencies implicit in operations. This module consists of two sub-modules: **structure dependency extractor** and **semantic dependency extractor**. The structure dependency extractor utilizes augmented element information to determine UI structural relationships between elements. E.g., by comparing *ID* fields, it can deduce if there is an dependency to re-click the same element as in prior operations, and by examining *List* field of two elements, it determines their membership in the same list and assesses the spatial interval between them.

The semantic dependency extractor leverages the capabilities of LLMs for natural language semantic understanding. Initially, the semantically-enriched operations, processed by the UI Extractor, are organized into a structured textual format, as illustrated in Figure 3. Subsequently, this formatted data, in conjunction with the operation-cognition knowledge base (also formatted for standardization), was fed to an LLM. To refine the LLM's output analysis, we integrated several empirically derived rules into the prompt. The temperature of the model was set to 0.2, and instructions such as "Do not make up a MAINCATEGORY" are incorporated into the prompt, mitigating the known issue of 'hallucinating' in LLMs, where the model generates incorrect or irrelevant content [23]. We used a one-shot example to direct the output format, employing a chain of thought (COT) [56] approach to guide the model's analysis for each operation, which involves following steps: First, each operation's text is repeated to enhance the LLM's memory retention; Second, the LLM uses contextual operations to infer the user's intent; Third, operation parameters are assessed whether to be fixed or variable in new automated contexts, based on user intent; Finally, the LLM determines if parameters stem directly from user intent or depend cognitively on prior operations. In cases of cognitive dependency, it categorizes and recovers these dependencies with NL descriptions. Figure 4 outlines the workflow of the semantic dependency extractor. The entire prompt can be found in Appendix A.

Moreover, operations directly derived from user intent are analyzed using the *List* field of the element parameter (if present) to ascertain whether this operation is iterative. Ultimately, a task graph is generated, encapsulating the task type, sequence of operations with parameters, and all cognitive dependencies, saved in JSON format.

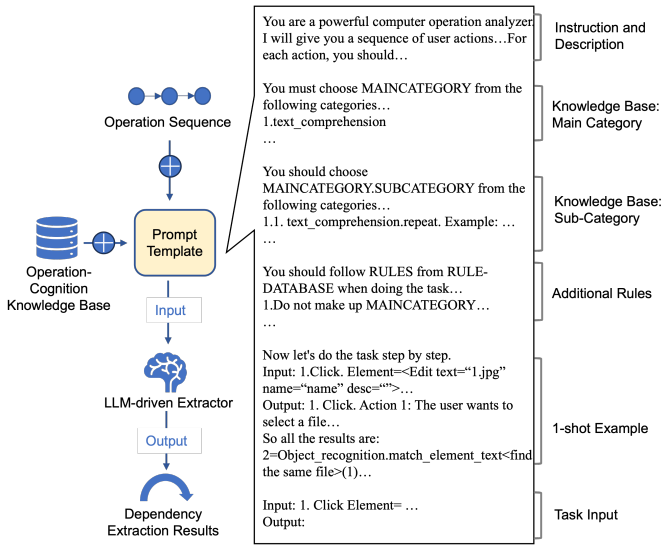

Figure 4: Implementation of the semantic dependency extractor.

4.2.3 Modification & Execution. After demonstration and task analysis by two extractors, the generated task graph will be visualized for users as an editable diagram (see Figure 2). Before starting automation, users can make necessary modifications to the graph to clarify their intents and define new task goals. The modification include adding, deleting or editing dependencies on their NL descriptions or connected operations. For iterative tasks, users can run the task graph immediately, since the remaining iterative elements represent new goals. For normal tasks, users can manually define new goals by entering text or selecting screen elements as new parameters directly from their intents (e.g., a new search keyword).

During execution, operations are carried out sequentially. For fixed operations, TaskMind uses a combination of recorded attributes (ID, type, name and text) to anchor the target element. For operations that cognitively depend on prior ones, TaskMind uses the NL description of the dependency as a prompt for LLM to inter new parameters in new contexts. To balance the generation speed and execution accuracy, the system employs gpt-3.5 for text parameter inference and gpt-4 for determining element parameters. The latter involves extracting and filtering all clickable elements on the current UI, and assigning indices for selection by the LLM.

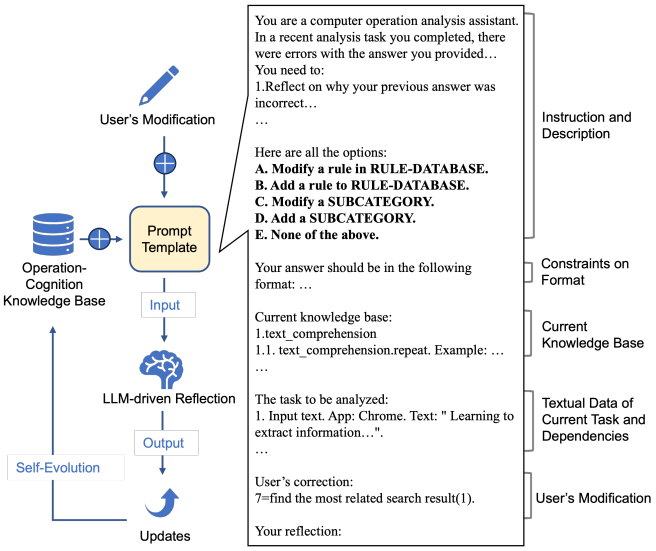

Figure 5: Implementation of the reflection module.

4.2.4 Reflection Module and Self-Evolution. The operation-cognition knowledge base helps in extracting cognitive dependencies. To achieve adaptability to diverse and task-specific needs, we introduce a **self-evolving mechanism** that employs a reflection module to digest user modifications using LLMs, enabling the continuous updating of the knowledge base over time. In this mechanism, when users modify a cognitive dependency, the reflection module—running in the background—incorporates the modification and the current knowledge base as inputs to the LLM. The reflection process is also guided by the COT approach, transforming modifications on specific dependencies into updates to the abstract knowledge base. The output includes updating existing subcategories, adding new ones, or making no changes. These updates improve the knowledge base's ability to recognize similar cognitive dependencies more accurately when users create new automation tasks in the future. Figure 5 outlines the workflow of the reflection module, and the entire prompt can be found in Appendix B.

5 Simulation Study

We conducted both a simulation study and a user study to evaluate TaskMind. The simulation study focused on technical performances of the task graph with cognitive dependencies, the semantic dependency extractor with the knowledge base, and the reflection module (see Figure 3). We compared TaskMind to the baseline version without cognitive dependencies to evaluate the impact of explicitly annotated dependencies on the success rate of generalizing automation programs learned from one demonstration to new task goals. The user study focused on usability, examining how users interact with the task graph and how they perceive the interaction, compared to the same baseline as the simulation study, where users only see a task graph without cognitive dependencies and use NL instructions to define new task goals and clarify their intents. Together, these two studies provided a comprehensive evaluation of TaskMind on both the technical performance of its pipeline and the usability of its interfaces.

The simulation study was conducted on the dataset derived from our formative study. The dataset comprised 39 tasks, each with a pre-annotated task graph containing cognitive dependencies, encompassing a total of 74 structural and 53 semantic dependencies. The average number of task steps (1 iteration) was 8.5, ranging from 3 to 19.

5.1 Evaluation of Automation Success Rate

5.1.1 ***Baseline and Implementation.*** To better understand the role of task graph with an explicit notion of cognitive dependencies, we used TaskMind without cognitive dependencies as our baseline. This was implemented using an end-to-end LLM that took user demonstration in the format after the UI Extractor (see Figure 3) as the input, and directly generated parameters for subsequent operations, without explicitly generating cognitive dependencies in the task graph as an intermediate stage. The prompt includes user-demonstrated operation sequence, already completed operations, task automation command in NL, and available elements on the current page. We provided clear instructions and few-show examples to facilitate the in-context learning, and utilized a chain-of-thought mechanism by instructing the LLM to analyze the intent of subsequent operations before generating answers to improve the performance. We controlled the type of the next operation to be the same as the demonstration in our baseline, leaving only the parameter generation to LLM, since TaskMind only use cognitive dependencies to generalize the parameters. For iterative tasks, we also designated the target element of each iteration. Detailed prompt can be found in Appendix C. We used gpt-4-1106 as the foundation model for both methods.

5.1.2 ***Procedure.*** An expert, proficient in programming and with a complete understanding of cognitive dependencies, demonstrated these tasks on a Windows 10 system. Afterward, tasks were automated with 3 methods: TaskMind using the task graph without any modification (TaskMind w/o Modification), TaskMind using the task graph with expert modifications based on the ground truth (TaskMind w/ Modification), and the baseline end-to-end LLM approach with expert-crafted automation commands in NL to describe the task goal (Baseline LLM).

We followed the metrics in Mind2Web [14] to report the task success rate (Task SR) and step success rate (Step SR) of parameter generalization and automation. To concentrate on generalizable operations that change parameters in different contexts, we calculated the success rate specifically for steps requiring parameter changes due to semantic dependencies (Semantic-Dependent Step SR). Furthermore, we considered a single automation attempt of a non-iterative task as one iteration, and reported the iteration success rate (Iter SR). Since the output parameters—suitable elements or NL text—can vary in practical settings, we conducted a manual evaluation by experts. The primary measure was whether the outcome of each step matched the task's actual intended purpose.

5.1.3 ***Results and Analysis.*** For TaskMind, 31 out of 39 tasks (79.5%) and 173 out of 204 iterations (84.3%) were successfully automated without any modifications to the task graph. After applying

Table 2: Comparison of parameter generalization success rate (SR, %) across tasks, iterations, steps, and semantic-dependent steps for 3 approaches: the end-to-end LLM approach (Baseline LLM), TaskMind without modification to the task graph (TaskMind w/o M), and TaskMind with modification (TaskMind w/ M).

	Task SR	Iter SR	Step SR	Semantic-Dependent Step SR					
				Information Recall	Information Understanding	Information Creation	Logical Reasoning	Contextual Selection	All
Baseline LLM	33.3	60.8	83.0	73.3	51.7	93.8	53.7	43.8	63.1
TaskMind w/o M	**79.0**	84.3	**98.3**	97.4	89.7	87.5	95.1	65.6	**90.9**
TaskMind w/ M	**92.3**	92.6	**99.5**	98.3	100.0	100.0	97.6	90.6	**97.7**

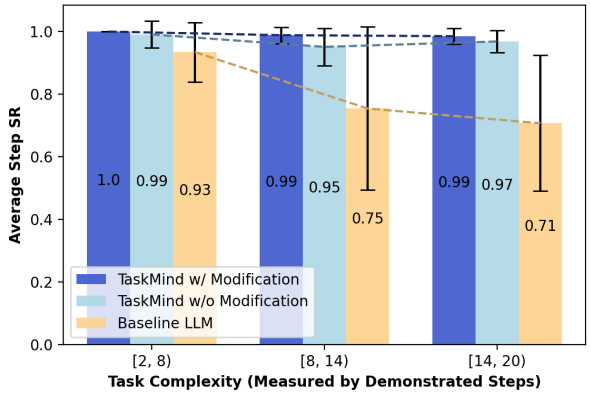

Figure 6: When task complexity increased, the step success rate of the baseline LLM declined while TaskMind maintained performance.

modifications, the success rate increased to 92.3% (36 out of 39 tasks) and 92.6% (189 out of 204 iterations). In contrast, the baseline LLM only achieved a 33.3% task success rate and a 60.8% iteration success rate. When focusing on semantic-dependent steps, the baseline LLM succeeded in only 63.1% of cases, whereas TaskMind achieved 90.9% success before modifications and 97.7% after modifications. This indicates that an explicit notion of dependencies significantly improves success rates and system reliability, particularly for steps critical to the generalization of demonstrations.

The automation failures of the baseline LLM, especially in comparison to our method, can be summarized as follows:

(1) **Tracking Long-horizon Tasks.** The LLM struggled more with complex tasks involving multiple operations, as shown in Figure 6. Results suggested the LLM often loses track of long-horizon tasks, a limitation also noted by [59], whereas a task graph is better suited for such complex scenarios. A typical example was Task 18 (T18), where the goal was to first paste and title an introduction text, then insert an image to create a slide. The baseline LLM failed to insert the image after titling, and its pre-answer analysis showed it mistakenly believed the task was complete.

(2) **Linking Task Goal to Specific Operations.** The baseline LLM often struggled to determine which operation the goal described in NL command should be applied to. As a result, the baseline LLM frequently failed to generate desired elements or text even with a clear description in the NL command. In 19 out of 39 tasks, the LLM made errors like selecting incorrect elements (e.g., failing to select an introduction paragraph in T16, selecting an irrelevant button instead of a searching result in T1), outputting blank, placeholder text, or repeating previous text (e.g., producing nothing when being told to summarize a paper in T1, generating text "sentiment" instead of the sentiment label in T35, repeating original text when writing a speech in T27). This highlights the LLM's difficulty in fully understanding the relationship between intent and operations. The task graph, however, clearly annotated dependencies for specific operation, making the task goal easier to follow.

(3) **Identifying Non-Generalizable Parameters.** The baseline LLM tended to change parameters that should remain unchanged in new contexts. For instance, in T5, it incorrectly altered the date in each birthday card's signature, and in T11, it failed to switch between two windows during a cross-application task. In contrast, the task graph explicitly indicates that operations involving no dependencies should have unchanged parameters, resulting in better performance and saving time and costs by not relying on LLM intelligence during the automation.

(4) **Generalizing Structural Positions.** The baseline LLM had difficulty inferring positional relationships between elements, often selecting incorrect cells for input. For example, in T22, it filled in the wrong rows when entering multiple items in an Excel sheet, failing to keep the entries aligned. TaskMind, with its structure dependency identified, avoided these issues.

The failure of TaskMind without modification were mainly due to incorrectly recognized semantic dependencies, which fell into four main categories: (1) **Unrecognized.** E.g., it missed the contextual selection *"find the mp4 file from current page"* in T11, leading to failures of finding new mp4 files in new contexts. (2) **Misinterpreted.** The user intent was misunderstood, e.g., "input the label for the result" with only one dependent operation, instead of "compare the proofreading result with the answer and output 'recognized' if match, otherwise 'unrecognized"' with 2 dependent operations in T6. (3) **Overly-Specific.** Dependencies were too specific to generalize in new contexts, e.g., "find the cell with the year 2019" instead of "find the cell with the same year" in T7. (4) **Overly-Abstract.**

Table 3: Recall, precision and f1-score (%) in extracting semantic dependencies for TaskMind with the simplest prompt, TaskMind without knowledge base and the full version of TaskMind.

	TaskMind Simple			TaskMind w/o KB			TaskMind Full		
	Recall	Precision	F1-score	Recall	Precision	F1-score	Recall	Precision	F1-score
Info Recall	88.9	84.2	86.5	77.8	82.4	80.0	94.4	89.5	91.9
Info Comprehension	93.3	82.4	87.5	60.0	56.3	58.1	93.3	82.4	87.5
Info Creation	40.0	100.0	57.1	20.0	100.0	33.3	60.0	75.0	66.7
Logical Reasoning	57.1	57.1	57.1	71.4	83.3	76.9	71.4	83.3	76.9
Contextual Selection	50.0	11.1	18.2	62.5	45.5	52.6	75.0	60.0	66.7
All	75.5	49.4	59.7	64.2	66.7	65.4	**84.9**	**80.4**	**82.6**

Dependencies were not detailed enough to meet task goals, especially for personalized needs, e.g., "summarize the text" instead of "write a presentation draft according to the text" in T27.

After modification, two major failures were resolved: First, task execution became more flexible instead of a rigid processes. For example, after correcting overly-specific ones, TaskMind could find cells with different years for each iteration, rather than always selecting the same year. Second, task execution better aligned with user intent, eliminating unsatisfactory outputs. For example, correcting overly-abstract dependencies allowed TaskMind to generate a presentation draft better met the task requirements. Nevertheless, even with modified dependencies, some failures persist in semantic-dependent steps. Although the types of failed dependencies vary, they all stem from the LLM's difficulty in applying semantic relationships. For example, in T6, it misjudged whether proofreading results matched the correct answer. In T19, when trying to find a cell with text similar to "duration", the LLM failed to select the correct "person-hour" cell. These errors might be addressed by introducing more sophisticated text generation or element grounding methods. Despite these issues, the introduction of task graph and cognitive dependencies still presents a significant advancement in improving success rate of automating tasks.

5.2 Evaluation of Dependency Extraction

5.2.1 Procedure. We conducted an ablation study on the dataset to validate the effectiveness of our operation-cognition knowledge base in enhancing the accuracy of semantic dependency extraction. We compared the full version of TaskMind with (1) TaskMind without the operation-cognition knowledge base provided in the prompt, (2) TaskMind with the simplest prompt, lacking of operation-cognition knowledge base and the chain-of-thought mechanism.

We used pre-annotated dependencies as the ground truth to validate the accuracy of the identification of dependencies from a operation sequence, and conducted a manual evaluation of their NL descriptions by experts due to the variability in NL expressions. Notably, although the knowledge base was derived from annotated data, it provided only an abstract categorization of semantic dependencies, with carefully designed examples included in the prompt that do not overlap with specific dependencies in the dataset. This ensured the knowledge base acted as a reference categorization, rather than a direct data source. The comparison between TaskMind with and without the knowledge base highlighted the contribution of expert-condensed categorization to the LLM's performance in identifying and describing dependencies.

5.2.2 Results. For a total of 53 semantic dependencies, the full version of TaskMind consistently outperforms the other two, achieving the highest overall F1-score of 82.6%, with particularly strong performance in Information Recall (91.9%), and Information Comprehension (87.5%), as shown in Table 3. The significant improvement observed after incorporating the operation-cognition knowledge base highlights its effectiveness in helping LLM understand task goals and the user intent of each operation.

Notably, while the simplest version showed a slight improvement in recall, it suffered a significant drop in precision, especially in predicting Contextual Selection (11.1%). The model identified many unnecessary contextual selection dependencies, such as "click the 'SUM' button" or "select the edit area", mistakenly interpreting these non-generalizable operations as requiring changes. This might be due to the prompt's simplification, which likely narrowed the LLM's focus, but also impaired its ability to understand and analyze the goal of generalization. Additionally, verbosity bias [48] might have contributed to the problem, leading to an excessive number of irrelevant dependencies.

5.3 Evaluation of Reflection Module

We further evaluated the performance of the reflection module by applying expert and non-experts modifications to the task graph for improving the dependency extractor.

5.3.1 Procedure. Our analysis included 3 groups of tasks, each comprising the original task where initial recognition of semantic dependencies requires modifications, and 2 additional tasks with similar semantic dependencies drawn from the formative study data. In addition to an expert proficient in programming and task graph modification, we also recruited 10 participants from a local university (7 males, 3 females) as non-experts, none of whom had prior PBD experience. They were first introduced about the concept of task graph and cognitive dependency, and then given time to modify the semantic dependencies in the original generated task graph for each group, with their personal interpretations. Following this, the reflection module integrated the modification and updated the knowledge base. The evolved knowledge base after modification of each individual was then applied to the 2 additional tasks in the group to test its effectiveness in recognizing semantic dependencies. Overall, this process led to 33 modifications of task graphs with

Table 4: Examples of modifications from expert and non-expert users and corresponding updates to the initial knowledge base.

Task	Original Dependency	Provider	Modification	Added Sub-category
1	extract a part of the text from	Expert	select abstract of paper based on title	select_based_on
		User 1	find the abstract of	extract_specific_information
		User 2	show the abstract of this paper	extract_specific_information
2	summarize the text inside	Expert	write summary and personal thoughts of slide content	reflect_and_summarize
		User 1	express personal feelings about the slides	express_personal_feelings
		User 2	summarize the content of the slide and write down feelings	summarize_and_comment
3	none	Expert	write the email content based on the email subject	based_on_previous_input
		User 1	write email on given subject	write_text_based_on_given_subject
		User 2	write detailed content text for the title	elaborate_text

66 test cases of evolution performance in total, encompassing 11 expert and non-expert participants. We reported the average recall, precision and f1-score before and after each individual modification.

Table 5: Average recall, precision and f1-score (%) in extracting semantic dependencies for 3 groups of tasks before and after individual modifications.

	Recall	Precision	F1-score
Before	85.7	58.3	69.4
After Expert Modifications	92.9	92.3	92.7
After Non-expert Modifications	90.3	91.6	90.9

5.3.2 Results. We noted a substantial improvement after the modification, as depicted in Table 5. The initial F1-score of 69.4% improved to 92.7% after expert modifications, and also improved to 90.9% after non-expert modifications.

Table 4 displays the original inaccurate dependency descriptions, examples of modifications from both expert and non-expert participants, and the corresponding updates to the initial knowledge base. The updates listed in the table were all proven effective in enhancing the recognition of semantic dependencies for similar tasks. Of the 33 updates, the majority (30) involved adding a new sub-category to the knowledge base, with the remainder including alterations to an existing main category or sub-category.

We further analyzed the behaviour of non-experts while modifying dependencies, who had less knowledge about PBD and cognitive dependency. There were 60 test cases of non-experts modifications in total, and 53 out of 60 test cases (88.3%) showed improved recognition of semantic dependencies post-evolution, with 45 cases (75.0%) achieving full accuracy of all dependencies. However, 15 cases exhibited inaccuracies or omissions in recognizing semantic dependencies. Analysis of these cases revealed common issues: misinterpretation or bias in understanding intent (5 cases), omission of details or use of ambiguous pronouns like "this" or "previous" (7 cases), and lack of clarity in illustrating relationships with prior operations (3 cases).

Despite these challenges, the study demonstrated the system's ability to integrate task-specific dependencies based on user modifications, even when the modification was overly specific or abstract. For instance, a general modification like "write detailed content for the title" was as effective as a more specific instruction such as "write an email based on the email title." Overall, the experimental results indicate that with the reflection module, our system has the ability to learn new cognitive processes from user feedback and generalize them into abstract knowledge, enhancing TaskMind's capability to evolve and adapt to personalized needs.

6 User Study

To evaluate the effectiveness, efficiency, and usability of TaskMind, we conducted a two-phase lab study with 20 participants. We used the same baseline setup as in the first simulation study (Section 5.1.1), where TaskMind without cognitive dependencies was employed. In this setup, users could only view a task graph without cognitive dependencies after demonstration, and therefore use NL instructions to define new task goals and clarify their intents if necessary (see Figure 7), which was the same objective of modifying the task graph. The study aimed to answer the following research questions:

RQ1. Is TaskMind more effective than the baseline approach?
RQ2. Is TaskMind more efficient than the baseline approach?
RQ3. Is TaskMind easier to interact with than the baseline approach?
RQ4. Is TaskMind capable of addressing users' real needs?

In the first phase, we used a within-subject study to compare TaskMind with the baseline using a set of predefined tasks that varied in difficulty levels and types of cognitive dependency. In the second phase, participants were encouraged to complete personalized tasks based on their actual needs and goals, using both TaskMind and the baseline method.

Table 6: 10 predefined tasks used in the user study that varied in 2 difficulty levels and 5 types of cognitive dependency.

	Easy	Difficult
Information Recall	Rename image files (Task 21)	Summarize annual total (Task 7)
Information Comprehension	Introduce scenic spots (Task 18)	Search for keywords (Task 16)
Information Creation	Write an email (Task 23)	Write a reflection (Task 26)
Logical Reasoning	Fill out a web form (Task 15)	Search for papers (Task 1)
Contextual Selection	Calculate rectangle areas (Task 39)	Calculate procurement expenses (Task 36)

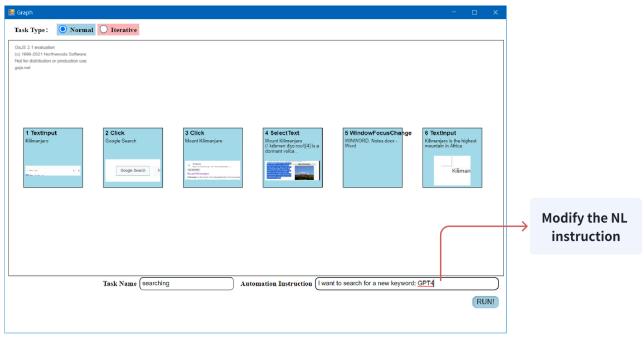

Figure 7: Interacting with the baseline LLM to automate tasks through natural language instruction, without task graph and cognitive dependencies.

6.1 Participants

We recruited 20 participants (10 male and 10 female) aged 20-36 (μ=23.1, σ=1.7), including 12 (P1-P12) university students and 8 (P13-P20) professionals. 5 participants had prior programming experience of more than 5 years, and also had used automation tools or written automation scripts, while 10 participants had prior programming experience of 1-5 years but had no automation experience, and 5 participants had no programming or automation experience. None of them had experience with visual programming tools.

6.2 Procedure

Each study started with an introduction and a consent solicitation, followed by a tutorial. During the tutorial, participants were given a document detailing necessary information and watched instructional videos on each method. They were informed about different interaction modes of the two methods. For TaskMind, participants were required to review the task graph and could modify it by adding, editing, or deleting dependencies before automating the task. For the baseline method, participants needed to edit the automation command in NL before execution, without examining the task graph. If the automation failed, they could re-modify the task graph or NL command and try the task again. The interaction interfaces are shown in Figure 2 and Figure 7. Before concluding the tutorial, they completed a simple task (T2) using both TaskMind and baseline methods to prove their understanding of the basic usage.

Following the tutorial, the formal study was divided into two stages: In the first stage, each participant was randomly assigned 2 tasks from a set of 10 predefined tasks, with 1 difficult and 1 easy in randomized order. They were asked to complete these 2 tasks using both TaskMind and baseline methods, and then assess the success based on their expectations. In the second stage, participants were encouraged to use both methods to complete a task relevant to their personal or professional contexts, utilizing their own files and familiar websites. Each participant completed a total of 3 tasks using both methods, resulting in 6 trials per participant and a total of 120 trials across the study. Each predefined task was completed 4 times using each method. To mitigate the learning effect, orders of methods and conditions in both phases were counterbalanced across tasks and participants.

After each trial, participants filled out a survey to rate aspects of workload and usability on a 7-point Likert scale based on NASA TLX model [19] with additional questions, with 1 being the most negative experience and 7 the most positive. After all the trials, participants evaluated their overall experience on the same 7-point Likert scale. All survey questions are shown in Appendix D. After completing all tasks, they engaged in a semi-structured interview, which explored their thought processes during interaction with both task graph and NL command, their perceptions of the interaction effort and utility, and whether they understood and paid attention to cognitive dependencies. Each phase took about 40 minutes, with the entire process taking about 2 hours. Each participants received $12 per hour and $2 for each personalized tasks as compensation[4].

6.3 Tasks

We selected 10 representative tasks from the dataset collected during the formative study, ensuring each type of semantic dependency was covered by two tasks: one easy (with 6 or fewer demonstrated steps) and one difficult (with more than 6 demonstrated steps). The task list is detailed in Table 6.

6.4 Results

6.4.1 Success Rate and Time (RQ1, RQ2). Participants concluded each trial either when satisfied with the automation results or upon deeming the task impossible to complete. As shown in Table 7, the average completion times were 5.1 minutes (σ=2.0) for TaskMind and 15.3 minutes (σ=8.4) for Baseline LLM, with a statistically significant difference (p=0.0001<0.05) observed through a paired-samples t-test. The step success rates were 95.3% for TaskMind and 78.0% for

[4]The study protocol was reviewed and approved by the university ethics review board.

Table 7: Comparison of efficiency between TaskMind and the baseline LLM.

	Task Complete Time (min)	Modification Time (s)	Number of Modifications
TaskMind	5.1 (σ=2.0)	102.5 (σ=38.0)	0.76 (σ=0.83)
Baseline	15.3 (σ=8.4)	23.5 (σ=10.3)	1.96 (σ=0.95)

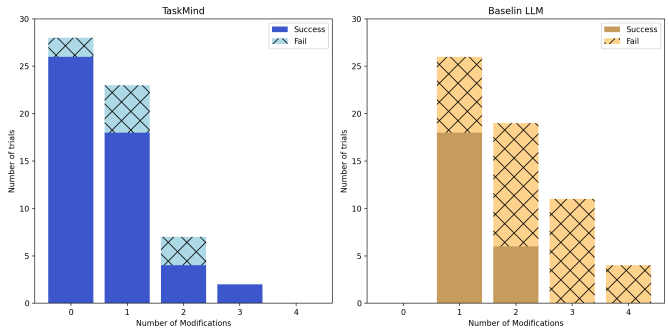

Figure 8: The effectiveness of trials based on the number of modifications to task graphs for TaskMind versus NL instructions for the baseline LLM. Since all participants performed at least one NL addition, there is no '0 number of modification' data for the baseline LLM.

Baseline LLM. These results indicate that participants were able to complete tasks more efficiently and reliably using TaskMind than with the baseline. The efficiency of TaskMind mainly comes from its improved success rate which lowers the need for further modifications, and reduced execution time achieved by using the task graph as a guide to skip non-generalizable parameter inference.

The study required TaskMind identifying a total of 61 instances of semantic dependencies, where it achieved a precision of 76.5%, recall of 79.6%, and an F1-score of 78.0%. The slightly lower performance compared to the simulation study was likely due to participants introducing additional and unintentional operations that impacted the recognition of semantic dependencies, as evidenced by the higher average number of demonstrated steps 9.6 compared to the 8.8 steps in the expert demonstrations for the pre-defined 10 tasks during the simulation.

6.4.2 Modifications and Collaborations (RQ1, RQ2, RQ3). For each trial, we recorded the total time participants spent making multiple modifications in the interfaces. On average, they spent 102.5 seconds (σ=38.0) with 0.76 (σ=0.83) modifications on TaskMind's task graphs, compared to 23.5 seconds (σ=10.3) and 1.96 (σ=0.95) modifications on Baseline LLM's NL commands. When modifying tasks graphs, participants mainly focused on correcting dependency recognition errors, categorized into 4 types discussed in 5.1: adding unrecognized (21.1%), correcting misinterpreted (7.9%), abstracting overly-specific (31.6%), clarifying overly-abstract (26.3%). The remaining changes (13.2%) involved deleting incorrectly identified dependencies. In contrast, NL command modifications primarily involved creating and expanding commands to specify task goals.

Participants spent more time on modifying the task graph (RQ2, RQ3). 7/20 participants mentioned that they needed extra time to grasp the logic of operations and cognitive dependencies. P11 noted, *"It took more time to understand the meaning of the arrows (cognitive dependencies)"*, while P4 explained, *"Most of the time was spent on creating and verifying missing dependencies"*. Additionally, we observed that participants tended to carefully check each operation when working with the task graph to validate the dependencies. For example, P7 spent much time analyzing the dependencies in T7 (Summarize annual total) which involve numerous steps and inter-dependent operations.

Despite extra time required for task graphs, modifying task graphs proved more effective than modifying NL commands with a higher success rate on each number of modifications (RQ1), as shown in Figure 8. Participants found it challenging to describe the intent of specific operations using NL commands, whereas task graphs provided more clarity (RQ3). P1 noted, *"Task graphs provide opportunities to rewrite the logic between operations, enabling targeted modifications to key steps of the task and their information sources."* P12 added, *"It's easier for task graphs to correct automation errors."* Meanwhile, when a task involved multiple dependencies, a single NL command was insufficient for accurate description. For example, facing Task 1 with 3 dependencies, P5 gradually modified the NL command from a simple description to a detailed one 3 times—"search for the title, click on a website, locate the abstract on the page, paste it into the adjacent cell, and write a summary for the abstract", until ultimately abandoning it.

6.4.3 Subjective Feedback (RQ3). We employed the Wilcoxon signed-rank test to compare the two methods on different metrics of subject ratings, as shown in Figure 9. For qualitative data from interviews, we used thematic analysis to identify key patterns related to participants' perceptions. There were no statistically significant differences in mental demand (Z=51.0, p>0.05), physical demand (Z=36.0, p>0.05), ease of learning (Z=12.5, p>0.05) and ease of using (Z=16.0, p>0.05) between TaskMind and Baseline LLM. However, 9/20 participants perceived that modifying the task graph required more mental demand and effort, especially in logical thinking, while NL instructions were more convenient. As P9 mentioned, *"The task graph requires understanding each dependency and their connections between operations, which is a bit hard to grasp."* Notably, when dealing with simple tasks, 2/20 participants (P10 and P18) preferred the modification of NL instructions for the simplicity, as P10 expressed, *"The task graph is relatively complex and requires editing of details. If the task is simple, the advantages are not as noticeable."*

Despite the effort, TaskMind significantly outperformed Baseline LLM in terms of performance (Z=12.0, p=0.0001<0.05), frustration (Z=25.5, p=0.005<0.05), satisfaction (Z=15.0, p=0.0001<0.05), controllability (Z=17.0, p=0.0001<0.05), reliability (Z=30.0, p=0.0003<0.05) and willingness to use (Z=6.0, p=0.03<0.05). Considering TaskMind's effectiveness compared to the baseline LLM, participants were willing to invest more time and effort in understanding the

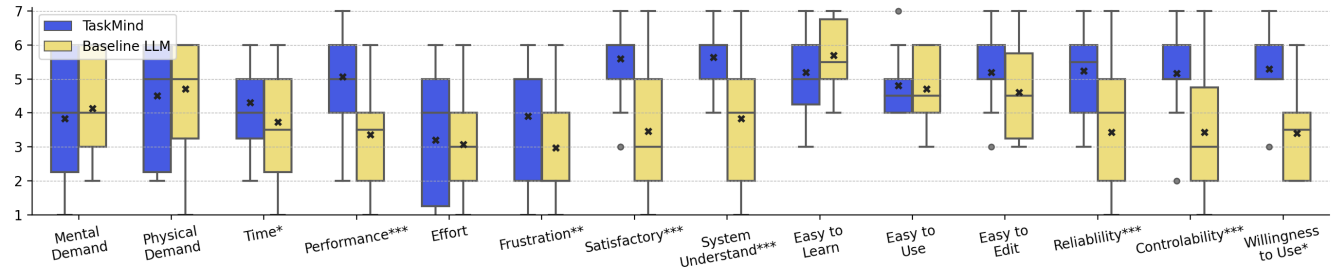

Figure 9: NASA-TLX and usability scores for each subscale with asterisks showing the statistically significant mean differences on the Wilcoxon signed-rank test (*: p<0.05, **: p<0.01, ***: p<0.001). Crosses (x) represent mean values, and horizontal lines (—) represent median values. For NASA-TLX (the first 6 scales), 7 indicates very low mental and physical demand, no time pressure, good performance, minimal effort, and no frustration. For usability (the remaining scales), 7 is very positive.

task graph to achieve better utility. P1 expressed, *"The task graph is more reliable, and after weighing the options, I'm still more willing to spend effort on modifying it."* Controllability also contributed to users' experience when interacting with TaskMind. Based on interview feedback, the sense of control was driven by two main factors: First, the specificity of granular modifications with effective feedback, as mentioned in Section 6.4.2. Second, the improved sense of mutual-understanding when collaborating with the machine. 11/20 participants mentioned that the task graph provided a more transparent way to verify the system's understanding of the task. As P7 noted, *"The task graph is easier to understand, as the arrows (cognitive dependencies) clearly show how the system processes my requirements."* P8 mentioned, *"The task graph's clear structure allows me to know whether the machine understands my requirements, making me easier to give clear instructions and ensuring we are on the same page."* This highlights the design need for a transparent, controllable interface that allows users to make targeted changes and instantly see results.

2/20 participants expressed concerns about the risks of flexible automation with Baseline LLM, as it occasionally overwrote original data row and deleted files, as P13 noted, *"it's a bit dangerous when you hand over control of the computer to the automation tool."* Concerns about execution efficiency were also raised, with 6/20 participants complaining that Baseline LLM was time-consuming.

6.4.4 Personalized Tasks (RQ4). The 20 personalized tasks provided by participants during the 2nd phase varied in difficulty, with number of operations ranging from 3 to 30 (μ=9.8, σ=7.4), and number of semantic dependencies ranging from 0 to 4 (μ=1.6, σ=1.1). These tasks included 6 related to file management, 9 to learning and academic activities, and 5 to clerical data processing. Among them, TaskMind successfully automated 15 out of 20 tasks (75.0%), while the baseline LLM automated 4 out of 20 tasks (20.0%). The F1-score for TaskMind in extracting semantic dependencies was 82.4%. In addition to the issue of incorrectly applying semantic dependnecies discussed in 5.1.3, TaskMind also encountered failures for two reasons. First, overly complex tasks increased the difficulty of interpreting and modifying the task graph, such as P2's attempt at collecting various data from a medical paper, where she failed to add unrecognized cognitive dependencies. Second, diverse operation paths led to automation failures, as seen in P11's task to collect citations from websites with varied formats and access paths, where the system failed due to its reliance on a fixed path demonstrated.

Participants believed that TaskMind could address their real-life needs. 12/20 participants expressed that repetitive tasks are common in their current jobs, with data organizing and processing tasks being the most frequently mentioned. For example, P15 noted, *"I'm currently interning at a company where we often do repetitive spreadsheet work, like filtering data from one sheet and filling it into the corresponding fields in another. Your tool would be helpful."* However, 5/20 participants felt that there are certain requirements TaskMind cannot meet, including repetitive tasks with significant flexibility or require decision-making. For example, P10 expressed his concern that processing experimental data *"is quite repetitive but requires thinking and analysis, which an automation tool would handle less effectively."* Additionally, 5/20 participants expressed interest in TaskMind's future improvements in interaction and broader application across different computer interfaces, suggesting features like *"trying NL instructions first, with the option to modify the task graph if results are unsatisfactory"* (P11) and proactive task prediction and recommendations after a few operations, without requiring manual invocation (P4). Overall, the second phase underscored TaskMind's effectiveness in automating real-scenario tasks, while also revealing limitations in handling intricate, flexible tasks or those requiring decision-making, highlighting potential areas for further improvement and expansion.

7 Discussion

7.1 Cognitive Dependency in Neuro-Symbolic Task Automation Systems

Traditional PBD methods employ predefined task scripts to guide the automation [7, 21, 27, 29, 30, 32, 37, 57], providing reliability but lacking the flexibility to adapt to diverse contexts, closely resembling a record-and-replay approach. Recent LLM-driven automation methods generate operation sequences from NL commands, relying on the pre-trained LLM's internal knowledge of task operations [11, 14, 20, 24, 60, 61]. This allows for flexible task execution but leads to lower reliability and higher risks [61]. Neuro-symbolic systems employ a hybrid model that combines both logical and neural components [9, 45], integrating symbolic inference on task

execution logic with semantic reasoning on user intents to enhance both reliability and generalizability in task automation.

Recent neuro-symbolic automation systems model tasks from various aspects including diverse program logics [45] to hierarchical task structure [28]. Upon existing work, TaskMind contributes to adding a semantic layer to represent the cognitive dependencies between operations, bridging the gap between observed operations and the user's underlying cognitive processes. The explicit notion of cognitive dependencies in a task graph enriches the task representation with additional semantic information, improving the generalizability for tasks with multiple interrelated parameters. The task graph is a hybrid representation of task knowledge, and is both user-interpretable and machine-referenceable, facilitating effective task consensus between humans and machines. We claim that this approach can be extended to include various types of task knowledge in the task graph, such as complex control logic and exception handling, providing scalability in collaborative automation for more intricate tasks in the future.

7.2 Operation-Cognition Knowledge Base for Personalized Task Modeling

Our simulation study without knowledge base reveals that while LLMs possess a foundational understanding of human cognition, they lack comprehensive knowledge about the cognitive processes specific to GUI operations. This shortfall is likely due to the scant discussion in human-generated corpora, a point P8 highlighted by noting that people seldom reflect on their intentions behind operations. To bridge this gap, we established an operation-cognition knowledge base through data collection, offering an overview of cognitive processes involved in GUI operations, and a reference for future researchers who seek to abstract operation and task intentions as a task modeling solution.

In addition, the challenges in recognizing personalized and task-specific dependencies led to our design of the hierarchical structure of the knowledge base, consisting of five closed main categories, several open subcategories for further expansion, and a self-evolving mechanism driven by LLM. The improvement in recognition after evolution highlighted TaskMind's capacity to learn from user feedback and abstract specific descriptions into generalized knowledge. However, challenges still exist with complex dependencies due to the limitation of the demonstration. For instance, the demonstration of conditional logic proved insufficient for capturing the complete intent, as evidenced by a personal task from P7 who aimed to "replace '1' with 'Male' and '2' with 'Female"', yet only the modification for the first data row was demonstrated.

7.3 Collaboration Insights for PBD

The diversity of personalized task requirements further underscores the importance of interaction in PBD to ensure alignment between human and machine in understanding tasks. Our user study compared two interactive interfaces: task graphs and NL instructions, revealing an effort-controllability trade-off. We observed that NL instructions are more accessible to non-programming users (P3, P9, P5), intuitive to modify (P8, P9), but difficult to modify with specificity (P1) and appear opaque (P2, P10). Task graphs offer better transparency, controllability and more reliability (P1, P2, P5, P10), but are initially more challenging to learn (P13) and requiring a more complex thought process to modify (P1, P8, P15). This trade-off reflects the tension between ease of use and the level of control. Existing work [49], which focuses on trust in automation agents, also mentioned controllability, reliability and some related criteria that influence human-machine collaboration in task automation. We believe that exploring improved interaction methods for task automation is essential for enhancing the effectiveness of such collaborations.

8 Limitations and Future Work

Currently, TaskMind automates tasks in limited applications, covering a substantial range of repetitive tasks encountered in professional and academic settings. However, user feedback indicated a demand for expansion to more applications such as literature management tools and alternative spreadsheet programs like WPS.

One limitation of TaskMind is that the rule-based element grounding approach for fixed operations cannot handle dynamically changing web structures. This challenge is beyond the scope of our current work, but future efforts can explore how to ground known elements, by generating flexible rules or using fast vision-based approaches.

Another limitation is that the influence of cognitive dependencies only affects the generalization of parameters. During the automation, the operation path remains fixed, which limits flexibility when dealing with complex workflows such as multiple branches or hierarchical sub-tasks. For example, this led to the failure of a personalized task that involved collecting citations from web pages with different operation paths. In the future, we will continue to explore more generalized neuro-symbolic approaches by modeling knowledge such as diverse and hierarchical workflows into the task graph, integrating LLMs to grant flexibility to more types of complex tasks.

Finally, taskMind operates on a PBD basis, requiring users to manually demonstrate the entire operation process and define task boundaries through explicit start and stop recording actions. Previous studies [7, 12] have tackled the challenge of identifying task boundaries through rule-based methods. Future versions of TaskMind aim to transcend these limitations by implementing advanced techniques for automatic and intelligent task segmentation, providing a more intuitive automation system by mining task graphs from users' natural usage.

9 Conclusion

We introduced TaskMind, an intelligent PBD system designed to automate GUI tasks by recovering cognitive dependencies from user demonstrations. Central to its design is a task graph and a comprehensive operation-cognition knowledge base developed from an extensive formative study, which effectively models computer tasks. TaskMind leverages this knowledge base, integrated with LLMs, to identify and recover these dependencies autonomously. Additionally, the user feedback for the task graph contributes to the self-evolving of the system. This innovative approach marks a significant advancement in intelligent task automation and paves the way for future developments in the field.

Acknowledgments

This work is supported by the Natural Science Foundation of China under Grant No. 62132010, Beijing Key Lab of Networked Multimedia, Institute for Artificial Intelligence, Tsinghua University (THUAI), Beijing National Research Center for Information Science and Technology (BNRist), 2025 Key Technological Innovation Program of Ningbo City under Grant No. 2022Z080, Beijing Municipal Science and Technology Commission, Administrative Commission of Zhongguancun Science Park No.Z221100006722018, and Science and Technology Innovation Key R&D Program of Chongqing.

References

[1] Reyhaneh Alimoradi, Fereidoon Shams Aliee, and Bahar Farahani. 2023. An Efficient Method For Selecting and Prioritizing Business Processes In Robotic Process Automation. In *2023 7th Iranian Conference on Advances in Enterprise Architecture (ICAEA)*. IEEE, 85–92.

[2] Deniz Arsan, Ali Zaidi, Aravind Sagar, and Ranjitha Kumar. 2021. App-Based Task Shortcuts for Virtual Assistants. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 1089–1099.

[3] Shaon Barman, Sarah Chasins, Rastislav Bodik, and Sumit Gulwani. 2016. Ringer: web automation by demonstration. In *Proceedings of the 2016 ACM SIGPLAN international conference on object-oriented programming, systems, languages, and applications*. 748–764.

[4] Maxime Bédard, Abderrahmane Leshob, Imen Benzarti, Hafedh Mili, Raqeebir Rab, and Omar Hussain. 2024. A rule-based method to effectively adopt robotic process automation. *Journal of Software: Evolution and Process* 36, 11 (2024), e2709.

[5] Benjamin S Bloom, Max D Engelhart, Edward J Furst, Walker H Hill, and David R Krathwohl. 1956. *Taxonomy of educational objectives: The classification of educational goals. Handbook 1: Cognitive domain.* McKay New York.

[6] Marc H Bornstein. 1986. Frames of Mind: The theory of multiple intelligences.

[7] 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.

[8] Jiun-Hung Chen and Daniel S Weld. 2008. Recovering from errors during programming by demonstration. In *Proceedings of the 13th international conference on Intelligent user interfaces*. 159–168.

[9] 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.

[10] Weihao Chen, Xiaoyu Liu, Jiacheng Zhang, Ian Iong Lam, Zhicheng Huang, Rui Dong, Xinyu Wang, and Tianyi Zhang. 2023. MIWA: Mixed-Initiative Web Automation for Better User Control and Confidence. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–15.

[11] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. 2024. Seeclick: Harnessing gui grounding for advanced visual gui agents. *arXiv preprint arXiv:2401.10935* (2024).

[12] Allen Cypher. 1991. Eager: Programming repetitive tasks by example. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 33–39.

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

[14] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2024. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems* 36 (2024).

[15] Shirin Feiz, Jason Wu, Xiaoyi Zhang, Amanda Swearngin, Titus Barik, and Jeffrey Nichols. 2022. Understanding Screen Relationships from Screenshots of Smartphone Applications. In *27th International Conference on Intelligent User Interfaces*. 447–458.

[16] Jerry Fodor. 1975. The language of thought.

[17] J Fodor. 1998. *Concepts: Where Cognitive Science Went Wrong.* Oxford University Press.

[18] Beverley Hancock, Elizabeth Ockleford, and Kate Windridge. 2001. *An introduction to qualitative research.* Trent focus group London.

[19] Sandra G Hart. 1986. NASA task load index (TLX). (1986).

[20] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. 2024. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14281–14290.

[21] Thanapong Intharah, Michael Firman, and Gabriel J Brostow. 2018. RecurBot: Learn to auto-complete GUI tasks from human demonstrations. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–6.

[22] Thanapong Intharah, Daniyar Turmukhambetov, and Gabriel J Brostow. 2019. Hilc: domain-independent pbd system via computer vision and follow-up questions. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 9, 2-3 (2019), 1–27.

[23] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *Comput. Surveys* 55, 12 (2023), 1–38.

[24] Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, et al. 2024. AutoWebGLM: A Large Language Model-based Web Navigating Agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5295–5306.

[25] Tessa Lau. 2009. Why programming-by-demonstration systems fail: Lessons learned for usable ai. *AI Magazine* 30, 4 (2009), 65–65.

[26] Tessa Lau, Julian Cerruti, Guillermo Manzato, Mateo Bengualid, Jeffrey P Bigham, and Jeffrey Nichols. 2010. A conversational interface to web automation. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*. 229–238.

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

[28] Lane Lawley and Christopher Maclellan. 2024. VAL: Interactive Task Learning with GPT Dialog Parsing. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–18.

[29] 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* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 6038–6049. https://doi.org/10.1145/3025453.3025483

[30] 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)*. IEEE, 105–114.

[31] 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*. 1–15.

[32] 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* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 577–589. https://doi.org/10.1145/3332165.3347899

[33] Toby Jia-Jun Li and Oriana Riva. 2018. KITE: Building conversational bots from mobile apps. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. 96–109.

[34] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping natural language instructions to mobile UI action sequences. *arXiv preprint arXiv:2005.03776* (2020).

[35] Henry Lieberman. 2001. *Your wish is my command: Programming by example.* Morgan Kaufmann.

[36] 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.

[37] 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.

[38] Robert J Marzano and John S Kendall. 2006. *The new taxonomy of educational objectives.* Corwin Press.

[39] Benjamin Matthies. 2020. Assessing the automation potentials of management reporting processes. *International Journal of Digital Accounting Research* 20 (2020), 75–101.

[40] Microsoft. 2021. Hooks. https://learn.microsoft.com/en-us/windows/win32/winmsg/hooks Accessed: 2024-04.

[41] Microsoft. 2021. IUIAutomation interface. https://learn.microsoft.com/en-us/windows/win32/api/uiautomationclient/nn-uiautomationclient-iuiautomation Accessed: 2024-04.

[42] Lihang Pan, Chun Yu, JiaHui Li, Tian Huang, Xiaojun Bi, and Yuanchun Shi. 2022. Automatically generating and improving voice command interface from operation sequences on smartphones. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–21.

[43] Blue Prism. 2023. SS&C Blue Prism. https://www.blueprism.com/ Accessed: 2024-04.

[44] 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 *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–16.

[45] 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 *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–15.

[46] Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2023. Android in the wild: A large-scale dataset for android device control. *arXiv preprint arXiv:2307.10088* (2023).

[47] André Rodrigues. 2015. Breaking barriers with assistive macros. In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility*. 351–352.

[48] Keita Saito, Akifumi Wachi, Koki Wataoka, and Youhei Akimoto. 2023. Verbosity bias in preference labeling by large language models. *arXiv preprint arXiv:2310.10076* (2023).

[49] Sivan Schwartz, Avi Yaeli, and Segev Shlomov. 2023. Enhancing trust in LLM-based AI automation agents: New considerations and future challenges. *arXiv preprint arXiv:2308.05391* (2023).

[50] 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.

[51] P.A.M. Seuren. 2006. Meaning: Cognitive Dependency of Lexical Meaning. In *Encyclopedia of Language & Linguistics*. Elsevier, 575–577. https://doi.org/10.1016/B0-08-044854-2/01056-7 Accessed: 2024-09-09.

[52] Atsushi Sugiura and Yoshiyuki Koseki. 1996. Simplifying macro definition in programming by demonstration. In *Proceedings of the 9th annual ACM symposium on User interface software and technology*. 173–182.

[53] UIPath. 2023. UIPath. https://www.uipath.com/ Accessed: 2024-04.

[54] Bryan Wang, Gang Li, and Yang Li. 2023. Enabling conversational interaction with mobile ui using large language models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–17.

[55] Cheng-Yao Wang, Wei-Chen Chu, Hou-Ren Chen, Chun-Yen Hsu, and Mike Y Chen. 2014. Evertutor: Automatically creating interactive guided tutorials on smartphones by user demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 4027–4036.

[56] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[57] 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.

[58] Xiaoyi Zhang, Lilian de Greef, Amanda Swearngin, Samuel White, Kyle Murray, Lisa Yu, Qi Shan, Jeffrey Nichols, Jason Wu, Chris Fleizach, et al. 2021. Screen recognition: Creating accessibility metadata for mobile applications from pixels. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15.

[59] Zirui Zhao, Wee Sun Lee, and David Hsu. 2024. Large language models as commonsense knowledge for large-scale task planning. *Advances in Neural Information Processing Systems* 36 (2024).

[60] Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614* (2024).

[61] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854* (2023).

A The Prompt of Semantic Dependency Extractor

system

You are a clever computer task analyzer. I will give you a sequence of user computer operations with parameters, which includes Element type, Text type, etc. You should first analyze the abstract and generalizable goal of the task, and give the global parameter of the task TASK_PARAM (If there are multiple, give the first). Then, for each operation, you should (a) repeat its type, (b) analyze the user's intention of this operation, and why the user chose this element or text, (c) determine if this operation is an operation with fixed parameters, which means the parameter should not change if the task will be executed in other context with other parameters, (d) explain why the user select this element or input this text, of which the reason should be a causal relation of this operation's parameters with other parameters of previous operations. The relation should only be between Text or Element types.
The relation should be in the following format: y=SUBCATEGORY.MAINCATEGORY<ADDITIONAL INSTRUCTION>(x). Here, y is the index of current operation, x is the index of a previous operation, MAINCATEGORY is the main category of the relation, SUBCATEGORY is the sub-category of the relation, and ADDITIONAL INSTRUCTION is a description of the relation. The description should be specific enough to be informative but abstract enough to generalize to new context.

You must choose MAINCATEGORY from the following options. Do NOT make up a MAINCATEGORY by yourself! Here are all the main categories:
1.Information recall.
2.Information comprehension.
3.Information creation.
4.Logical reasoning.
5.Contextual selection.

You can choose SUBCATEGORY from the following example options. If the SUBCATEGORY you need doesn't exist, you can create a new SUBCATEGORY. Here are examples of the sub-categories:
1.1. Information recall: repeat. Select an element or input something based on previously same or similar information. Example: repeat.information_recall<select the element with similar text>(x), repeat.information_recall<input the same text>(x).
1.2. Information recall: polish text. Polish the text into specific style. Example:y=polish_text.information_recall<use a more formal tone>(x).
1.3. Information recall: format text. Keep the original content intact while changing the format, structure, or presentation. Example: y=format_text.information_recall<replace suffix>(x), y=format_text.information_recall<structure previous data based on a specific format>(x).
2.1. Information comprehension: summarize. Understand the information and give a summary. Example:y=summarize.information_comprehension <summarize into keywords>(x).
2.2. Information comprehension: derive information. Understand the information and derive some information. Example:y=derive_information.information_comprehension <derive the student id from>(x).
3.1. Information creation: generate. Generate or expanding new content, instead of merely adjusting the format of existing content. Example:y=generate.information_creation<write a short essay based on the keyword>(x), y=generate.information_creation<write an application for the post>(x).
4.1. Logical reasoning: calculate. Performing mathematical calculations on existing data. Example: y=calculate.logical_reasoning<add two numbers>(x, y).
4.2. Logical reasoning: evaluation. Analyzing existing data to produce a result, comparison, or categorization. Example: y=evaluation.logical_reasoning<compare two strings and give the result>(x, y), y=evaluation.logical_reasoning<categorizing data based on certain criteria>(x, y).
5.1. Contextual selection: match element. Select the element from the current screen based on some criteria. Here, x can be equal to y, which means the result comes from the current context. Example: y=match_element.contextual_selection<find the movie title on the current screen>(x), y=match_element.contextual_selection<Select the item with the lowest price>(x).
5.2. Contextual selection: match text. Select the text from the current screen based on some criteria. Here, x can be equal to y, which means the result comes from the current context. Example: y=extract_text.contextual_selection<extract the text that matches a specific pattern>(x).

ATTENTION, here are some important rules you need to follow:
1. Copy, Paste or other operations without Element or Text parameter, will be fixed and not have relation with other operations.
2.Do NOT make up any MAINCATEGORY.
3.If it's an Element selection operation, you should only look for the relation between current Element's

features with previous operation's Text/Element parameter. If it's a TextInput operation, you should only look for the relation between current Text parameter with previous operation's Text/Element.
4. Usually, the index of x should be smaller or equal than y, which means the parameter must come from the previous parameters, or the current context.
5. An unfixed operation MUST have a relation.
6. Try your best to find the relation with previous operations, avoiding x = y!

Now let's do the task step by step, operation by operation. DO NOT skip any operations! DO NOT skip any operations! Follow the output format of the example!

user

1. Click. Element=<Edit text="1.jpg" name="name" desc="">.
2. Right click. Element=<Edit text="1.jpg" name="name" desc="">.
3. Click. Element=<MenuItem text="" name="rename(M)" desc="">.
4. Input text. Text="1(color printing).jpg". Element=<Edit text="1.jpg" name="1.jpg" desc="">.
5. Press.

assistant

The abstract task goal is to rename the file. The global parameter of the task, which is the target of the goal, is the file to be renamed, and the first unfixed operation is 1, so the first global parameter TASK_PARAM=1. There are 5 operations and I must analyze each operation without skipping any of them.
1. Click. Element=<Edit text=""1.jpg"" name=""Name"" desc="""">. Operation 1: The user clicks this element because it's the file that the user wants to rename. The file will be changed if the user want to rename other files, so it's unfixed. But this is the global parameter, so there is no relation. Conclusion: unfixed.
2. RightClick. Element=<Edit text=""1.jpg"" name=""Name"" desc="""">. Operation 2: The user right clicks this element because the user wants to see the context menu of the same file. It's unfixed. Because current Element's text feature ""1.jpg"" has the same text with the Element (""1.jpg"") in operation 1, So the result is 2=repeat.information_recall<find the same file>(1). Conclusion: unfixed.
3. Click. Element=<MenuItem text="""" name=""Rename(M)"" desc="""">. Operation 3: The user click this element because the user wants to rename the file. This operation will no change if the user wants to rename other file, so it's fixed. Conclusion: fixed.
4. TextInput. Text=""1(color printing).jpg"". Element=<Edit text=""1.jpg"" name=""1.jpg"" desc="""">. Operation 4: The user inputs the new name of the file. The text will be changed if the file changed, so it's unfixed. Because current Text ""1(color printing).jpg"" is a formatted string to previous text ""1.jpg"" in Operation 1, So the result is 4=format_text.information_recall<add the specific suffix ""(color printing)"" to the filename>(1). Conclusion: unfixed.
5. Enter. Operation 5: The user confirms after renaming the file. This operation has no parameter, so it's fixed. Conclusion: fixed.

So all the results are: 2=Object_recognition.match_element_text<find the same file>(1). 4=text_processing.edit_text<insert string "(color printing)" before suffix>(1).

B The Prompt of Reflection Module

system

You are a computer operation analysis assistant. In a recent analysis task you completed, there were errors with the answer you provided, and the user made corrections to it. Now, I will present the analysis task, the error in your answer, and the user's correction. You need to:
1.Reflect on why your previous answer was incorrect. Examine the specifics of the error and the root cause behind it.
2.If there is a RULE in RULE-DATABASE that causes your answer to be incorrect, modify that RULE.
3.Reflect what areas can you improve upon for the next time you do this task. (For example, paying more attention to specific aspects, etc.) Then summarize it into a new RULE. Then you should generalized the new RULE, so that the new RULE can be applied correctly to many tasks and cases.
4.Reflect whether modifying the name or example of a SUBCATEGORY can help you do the task better and fix the error. However, the new SUBCATEGORY must cover the range of the old one. If you can't ensure that, add a SUBCATEGORY instead of modifying! For instance, you must not modify SUBCATEGORY "compose text" to "write an email on", because "write an email on" does not cover the complete range of "compose text". Then generalize the name of the SUBCATEGORY.
5.Reflect whether the user feedback belongs to a new SUBCATEGORY that better discribes this dependency. If true, add the SUBCATEGORY (Remember the SUBCATEGORY should belong to an existing MAINCATEGORY.) Then generalize the SUBCATEGORY based on the following rules: The name of a SUBCATEGORY must be a general description of a class of dependencies. The name of a SUBCATEGORY should not be task-specific, it should cover many similar tasks and examples. For instance, "write an email on" can't be the name of a SUBCATEGORY because it can only cover write-email tasks. However, "write text based

on" can be a SUBCATEGORY because it can also cover write-speech tasks, write-letter tasks, etc..
6.Based on your reflection above, give suggestions on how to prevent errors in completing this task next time. You can choose one or more options from the following. Here are all the options:

A. Modify a RULE in RULE-DATABASE. Remember to generalize the new RULE.
B. Add a RULE to RULE-DATABASE. Remember to generalize the new RULE.
C. Modify a SUBCATEGORY. Remember to generalize the new SUBCATEGORY. Then modify the origin examples of the SUBCATEGORY.
D. Add a SUBCATEGORY. Remember to generalize the new SUBCATEGORY. Then give two different examples of the new SUBCATEGORY that best explains it.
E. None of the above.

Your answer should be in the following format(you can change the words inside {}):
1.why my previous answer was incorrect:{answer}
2.modify a RULE in RULE-DATABASE:{assessment and analyse}
generalize the modified RULE:{analyse and RULE generalization}
3.add a RULE to RULE-DATABASE:{assessment and analyse}
generalize the new RULE:{analyse and RULE generalization}
4.modify a SUBCATEGORY:{assessment and analyse}
generalize the name of the modified SUBCATEGORY:{analyse and name generalization}
5.add a SUBCATEGORY:{assessment and analyse}
generalize the name of the new SUBCATEGORY:{analyse and name generalization}
6.suggestion(only output the chosen suggestion):
A. RULE-DATABASE.modify({number of the origin RULE},{content of the origin RULE}).content({content of the RULE after modification}).[endA]
B. RULE-DATABASE.add({the RULE you want to add}).[endB]
C. SUBCATEGORY.modify({initial name of MAINCATEGORY}. {initial name of SUBCATEGORY}) .name ({name of MAINCATEGORY}.{name of SUBCATEGORY after modification}) .example1({example of SUBCATEGORY, in format of y= MAINCATEGORY. SUBCATEGORY <ADDITIONAL INSTRUCTION>(x)}) .example2({example of SUBCATEGORY, in format of y= MAINCATEGORY. SUBCATEGORY <ADDITIONAL INSTRUCTION>(x)}).[endC]
D. SUBCATEGORY.add({name of MAINCATEGORY}. {name of SUBCATEGORY you want to add}).example1 ({example of SUBCATEGORY, in format of y= MAINCATEGORY. SUBCATEGORY <ADDITIONAL INSTRUCTION>(x)}) .example2 ({example of SUBCATEGORY, in format of y= MAINCATEGORY. SUBCATEGORY <ADDITIONAL INSTRUCTION>(x)}) .[endD]
E. NONE.[endE]

C The Prompt for the baseline LLM

system

– Goal –
You are a task automation agent. You should generate the next operation based on the automation instruction, the already automated operations and the current screen information, referring to the demonstrated operation sequence. You should first analyze the goal of the task , and then analyze the intent of this operation before giving the answer. Attention: When automating tasks, make sure your operations are the same as the demonstrated ones, with only the parameters changing. For repetitive tasks, the demonstration is the first iteration, and you should begin from the second iteration onward. Each iteration should follow the same operations as the demonstration. You should output the operation along with its parameters. There are multiple types of parameters: Text, Element, Url, Key and Window. For parameters of the Element type, you only need to select and output the corresponding ID as a LIST from the current screen elements. For the Window type, you only need to select and output the ID from the available windows. If the task is complete, use "End" operation to terminate the automation.

– Available Operations with Parameter types –
Click(Element), TextInput(Element, Text), Copy(Element, Text), Paste(Element, Text), Save(), WindowOpen(Element), WindowClose(), MiddleClick(), DoubleClick(Element), RightClick(Element), Enter(), FunctionKey(Element, Keydata), MultiSelect(Element, Element), Drag(Element, Element?), Move(Element, Element), NewTab(), UpdateTab(Url), CloseTab(), SelectText(Element, Text), WindowFocusChange(Window), End()

– Example 1 –
Input:
Available elements on the current screen:
<DataItem id=""0"" text=""2024"" name=""A2"">
<DataItem id=""1"" text=""2025"" name=""A3"">
<DataItem id=""2"" text=""2026"" name=""A4"">
...
Demonstrated operation sequence:
1. Type: WindowFocusChange, Window: Excel.
2. Type: Click, Element: <DataItem text=""2024"" name=""A2"">.
3. Type: Copy,

Automation instruction:
I want to copy the content of each dataitem of this column.
This task is repetitive. The first iteration is already demonstrated. At this iteration, operation 2's parameters is: <DataItem text=""2025"" name=""A3"">
Already automated operations:
1. Type: WindowFocusChange, Window: Excel.
The next automated operation should be Click. You MUST provide the most likely parameters, even if the information is incomplete.
Output:
This task is to copy the content of all the cells. The next step is corresponded to operation 2 in the demonstrated sequence. I should click the target of this iteration, so the next step is:
Type: Click, Element: id=[1]

– Example 2 –
Input:
Available elements on the current screen:
<DIV id=""0"" text="""" name=""input"">
...
Demonstrated operation sequence:
1. Type: Click, Element: <Div text="""" name=""input"">.
2. Type: TextInput, Text: ""Beijing is a beautiful city"", Element: <Div text="""" name=""input"">.
Automation instruction:
Write a introduction about Shanghai.
Already automated operations:
1. Type: Click, Element: <Div text="""" name=""input"">.
The next automated operation should be TextInput. You MUST provide the most likely parameters, even if the information is incomplete.
Output:
This task is to write an introduction. The next step is corresponded to operation 2 in the demonstrated sequence. I should write an introduction about "Shanghai", so the next step is:
Type: TextInput, Text: ""Shanghai is a bustling city, known as the Pearl of the Orient."", Element: id=[0]

– Example format of other outputs –
Type: WindowFocusChange, Window: id=[0]
Type: MultiSelect, Element: id=[1,2]
Type: Move, Element: id=[1,2]
Type: End
...

user

Available elements on the current screen: ...
Available windows for WindowFocusChange on the current screen: ...
Demonstrated operation sequence: ...
Already automated operations: ...
Automation instruction: ...

The next automated operation should be .... You MUST provide the most likely parameters, even if the information is incomplete.

D Surveys of User Study

D.1 Survey of Workload and Usability

- How mentally demanding was the task?
- How physically demanding was the task?
- How hurried or rushed was the pace of the task?
- How successful were you in accomplishing the task?
- How hard did you have to work to accomplish your level of performance?
- How frustrated did you feel during the task?
- How satisfied were you with the automation result?
- To what extend do you think the system understand your task?

D.2 Survey of User Experience

- How easy was it to learn how to use the system?
- How easy was it to automate the task using the system?
- How easy was it to modify the task and interact with the system?
- To what extend do you think the system is reliable?
- To what extend do you think the system is controllable?
- How likely are you to use this system in the real life?

E The Dataset of 39 Example Tasks

The example tasks are listed in Table 8 and 9. It should be noted that the column 'Count' stands for the number of *Structure Dependency* in a single task.

F The Set of 21 Common Computer Operations

See Table 10.

Table 8: The Dataset of 39 Example Tasks (Part 1)

	Task Name	Task Description	Semantic Dependency List	Count
1	Paper Abstract	Search for the paper, paste the abstract, then translate and summarize the main points.	1.Object_recognition.match_element_text<find the search result most related to> 2.text_comprehension.extract_information<select abstract of paper> 3.text_comprehension.summarize<summarize in one sentence>	4
2	Extract Birthday from ID Card-1	Extract the birthday and fill in the new column according to the ID number in the form (first click the ID number and then enter the birthday).	1.text_comprehension.extract_information<extract birthday from ID number>	3
3	Extract Birthday from ID Card-2	Extract the birthday and fill in the new column according to the ID number in the form (directly enter the birthday).	1.text_processing.edit_text<extract date of birth from ID number>	1
4	Edit Formatted Data	Form a format string to fill in a new column based on the data in multiple columns of the table (directly enter in the new column without instruction).	1.text_processing.edit_text<format text according to template>	2
5	Generate Birthday Card	Generate or modify the birthday card according to excel sheet, which contains name and date information.	1.text_creation.compose_text<create a greeting message with the name and date from previous actions>	4
6	Character Proofreading	Select and copy the cell text in the excel sheet to the browser interface, click proofreading, and fill in the new column with results and identified problems.	1.logical_reasoning.compare_results<compare and output "identify" if same> 2.text_comprehension.repeat<input the same text>	6
7	Data Summation and Aggregation	Sum a column of data, remember the result and switch to sheet1 to input the result.	1.Object_recognition.match_element_properties<find the same year> 2.text_comprehension.repeat<input the same text>	4
8	Batch Rename .md Files	Rename multiple files from 1.txt to 1.md.	1.text_processing.edit_text<replace ".txt" with ".md" and keep the number same>	1
9	Batch Rename Paper	Rename a list of files to 'paper1', 'paper2', 'paper3', etc.	1.text_processing.edit_text<replace text with "paper" and iteration number>	1
10	Batch Rename Files	With file 'xxx-1' ready, copy more files and rename the new files to 'xxx-2', 'xxx-3', etc.	1.text_processing.edit_text<replace the number after the dash with iteration number>	1
11	Merge Folders	Click into sub-folders in folder A, copy the mp4 file, and paste it into the corresponding sub-folder in folder B.	1.Object_recognition.match_element_text<find the file with similar text>	1
12	Batch File Compression	Compress multiple folders separately and delete all the folders before compression.	None.	1
13	Batch File Decompression	Decompress multiple compressed files separately and delete all files before decompression.	None.	1
14	Batch Rename by Specific Title	Rename each file in a folder to "xx(color_printing)".	1.text_processing.edit_text<add string "(color_printing)" to the text>	1
15	Fill in Web Forms	Copy the contents of excel and fill them out in turn in the web form.	1.Object_recognition.match_element_properties<similar column header>	0
16	Keyword Searching	Type keywords in the document, switch to the browser to search for keywords, and summarize the content to the document after browsing.	1.Object_recognition.match_element_text<find the result most related to> 2.text_comprehension.repeat<input the same text> 3.text_comprehension.summarize<summarize in one sentence> 4.text_comprehension.extract_information<extract detailed description from>	1
17	PPT Remark Translation	Translate the text on the PowerPoint into English and fill in the remarks column.	1.text_processing.translate<translate into Chinese>	1
18	Make PPT in Batches according to Word	Make PowerPoint in batches according to Word, and make one page for each scenic spot in Word.	1.Object_recognition.match_element_text<find the picture with the same name> 2.text_comprehension.extract_information<extract the keyword from>	2
19	Subject Fee Summary	Copy and paste the corresponding columns "Name", "ID number" and "Fee" into the corresponding column of the master table.	1.Object_recognition.match_element_text<find the column with similar text>	3
20	Rename Download Files	Click a link in the web page, download the files on a new page and name each file the name shown on the link.	1.Object_recognition.match_element_text<find the same file> 2.text_processing.edit_text<remove string "25-" and "(2023)" from>	1
21	Rename Files with Suffix "fix"	Add -fix to the filename of all the images in a folder.	1.text_processing.edit_text<insert string "-fix" before suffix>	1
22	Expenditure Detail Form	Fill in the different items in the corresponding expense table according to the schedule.	1.Object_recognition.match_element_properties<extract property information> 2.logical_reasoning.extract_information<extract how to calculate the total fee>	8
23	E-mail Writing	Enter the subject of the email first, and then write the message.	1.text_creation.based_on_previous_input<write an email on the subject>	0

Table 9: The Dataset of 39 Example Tasks (Part 2)

	Task Name	Task Description	Semantic Dependency List	Count
24	Extract a List by Folder Names	Click on the folder and enter the name of all the folders in the root directory in word.	1.text_comprehension.repeat<input the same text>	1
25	Add Caption to Pictures	Add a common caption to two illustrations that already have a caption.	1.Object_recognition.match_element_text<find the same image> 2.text_comprehension.summarize<summarize in one sentence>	0
26	Write Thoughts according to PPT	Expand summary and thoughts to word document according to PowerPoint.	1.text_creation.reflect_and_summarize<summarize the text inside>	2
27	Write a Speech according to PPT	Write the presentation to word document according to PowerPoint.	1.text_creation.compose_presentation_draft<summarize in one paragraph>	2
28	File Movement	Move all files with names like "xx(color_printing) "into a folder.	None.	1
29	Automatic Numbering-1	Auto-number each line with [1][2][3].	1.text_processing.edit_text<add iteration number before the original text>	2
30	Automatic Numbering-2	Change the dot number from 0.1.2. to 1.2.3.	1.text_processing.edit_text<replace the reference number with iteration number>	2
31	Modify Title Format in Batches	Use the same format for a list of the same type of text, with a space between the number and the text.	1.text_processing.edit_text<insert a space after "Level">	2
32	Bank Card Number Conversion	Replace the four asterisks in the bank card number column with the number 1000.	1.text_processing.edit_text<replace "****" with "1000">	2
33	PPT Text Polishing	Polish the English text in the PowerPoint text box.	1.text_processing.extend_parameter<summarize in one sentence>	2
34	Learning Content Recording	Put the title of the video being played in the browser into the Today plan in word document.	1.text_creation.compose_text<write a sentence based on the video title>	1
35	Sentiment Analysis	Mark whether a movie review is positive or negative.	1.logical_reasoning.classify<judge if the statement is positive or negative>	2
36	Calculate the Amount of Materials Purchased	For a material in a row of the form, fill in the total amount calculation formula, and calculate the total amount.	1.text_creation.compose_text<combine the values in the "other_properties"> 2.logical_reasoning.calculate<calculate the result of the formula>	1
37	Answer Chinese Self-test Questions	Select a question and enter the answer.	1.logical_reasoning.answer_question<provide the answer to the question>	2
38	Take Multiple Choice Questions	Select a question and enter one of the four options.	1.logical_reasoning.answer_question<select the most correct answer from the options>	2
39	Calculate Rectangular Area	Calculate the area based on the length and width of the yard.	1.logical_reasoning.calculate<calculate the product of the numbers>	2

Table 10: The Set of 21 Common Computer Operations

	Operation	Parameter Types	Description
1	Click	element	Left mouse click
2	TextInput	element, text	Input text
3	Copy	element, text	Copy, including keyboard Ctrl+C and mouse click on the copy button
4	Paste	element, text	Paste, same rule as above
5	Cut	element, text	Cut, same rule as above
6	Save	/	Save, same rule as above
7	WindowOpen	element	Open application window
8	WindowClose	/	Close application window
9	MiddleClick	/	Page scroll
10	DoubleClick	element	Double-click with an interval less than a threshold
11	RightClick	element	Right-click to invoke a menu
12	Enter	/	Enter key
13	FunctionKey	element, keydata	Shortcut keys
14	MultiSelect	element1, element2	Drag to multi-select, such as Excel cells
15	Drag	element1, element2	Drag a UI element
16	Move	element1, element2	Move files
17	NewTab	/	Open a new browser tab
18	UpdateTab	url	Navigate to a URL
19	CloseTab	/	Close a browser tab
20	WindowSwitch	window	Switch to another window
21	SelectText	text, element	Drag to select text on a page