Do It For Me vs. Do It With Me: Investigating User Perceptions of Different Paradigms of Automation in Copilots for Feature-Rich Software

Anjali Khurana
Computing Science
Simon Fraser University
Burnaby, British Columbia, Canada
anjali_khurana@sfu.ca

Xiaotian Su
Computer Science
ETH Zürich
Zürich, Switzerland
xiaotian.su@inf.ethz.ch

April Yi Wang
Computer Science
ETH Zürich
Zürich, Switzerland
april.wang@inf.ethz.ch

Parmit K Chilana
Computing Science
Simon Fraser University
Burnaby, British Columbia, Canada
pchilana@sfu.ca

Abstract

Large Language Model (LLM)-based in-application assistants, or *copilots*, can automate software tasks, but users often prefer *learning by doing*, raising questions about the optimal level of automation for an effective user experience. We investigated two automation paradigms by designing and implementing a fully automated copilot (AutoCopilot) and a semi-automated copilot (GuidedCopilot) that automates trivial steps while offering step-by-step visual guidance. In a user study (N=20) across data analysis and visual design tasks, GuidedCopilot outperformed AutoCopilot in user control, software utility, and learnability, especially for exploratory and creative tasks, while AutoCopilot saved time for simpler visual tasks. A follow-up design exploration (N=10) enhanced GuidedCopilot with task-and state-aware features, including in-context preview clips and adaptive instructions. Our findings highlight the critical role of user control and tailored guidance in designing the next generation of copilots that enhance productivity, support diverse skill levels, and foster deeper software engagement.

CCS Concepts

• **Human-centered computing** → **Graphical user interfaces**.

Keywords

feature-rich software; large language models; software copilots; user control; semi-automation; human-AI collaboration

ACM Reference Format:
Anjali Khurana, Xiaotian Su, April Yi Wang, and Parmit K Chilana. 2025. Do It For Me vs. Do It With Me: Investigating User Perceptions of Different Paradigms of Automation in Copilots for Feature-Rich Software. In *CHI Conference on Human Factors in Computing Systems (CHI '25), April 26–May 01, 2025, Yokohama, Japan.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3706598.3713431

CHI '25, Yokohama, Japan

ACM ISBN 979-8-4007-1394-1/25/04
https://doi.org/10.1145/3706598.3713431

1 Introduction

Human-Computer Interaction (HCI) research has long been exploring in-application AI-based assistants [5, 37, 41, 49, 55, 65] designed to assist users in navigating feature-rich software applications. With the advancements in Generative AI and pre-trained Large Language Models (LLMs) [2, 3, 12, 77], a new generation of AI assistants is emerging, capable of automating a wide range of complex tasks. These LLM-powered assistants, often branded as *copilots* [2, 74, 77], can offer targeted, flexible assistance based on users' natural language description of help needs [89]. By integrating context-sensitive automation dynamically into user workflows [9, 74, 77], these modern copilots such as Microsoft 365 Copilot [77], Adobe Firefly [2], and Figma AI [24] are exploring ways to fully automate software tasks, redefining user expectations of what generative AI can achieve and pushing the boundaries of productivity and creativity.

Despite the promise of fully automated copilots, they introduce distinct challenges due to their open-ended nature, inherent complexities, and wide-ranging failure modes [9]. Users often need to repeatedly review AI outputs and refine their inputs to align them with their intent, increasing overall effort [10, 78, 81, 87]. Addressing these challenges requires effective human-agent collaboration [9, 76], which involves providing appropriate information [6, 76] and understanding the dynamics of user interaction across diverse AI scenarios to avoid false assumptions [52, 74]. Fully automated copilots often also overlook a key principle from HCI and software learnability research: users often prefer to *learn by doing* when exploring software features [15], underscoring the need for balance between guidance and automation [42]. Research also shows that step-by-step instructions with GUI visuals [41, 49, 64] and demonstrations [32, 48, 61] are essential for improving feature discovery, retention, and learning outcomes [16, 33].

In this work, we investigate how humans and copilots can collaborate within feature-rich software and what the concept of a

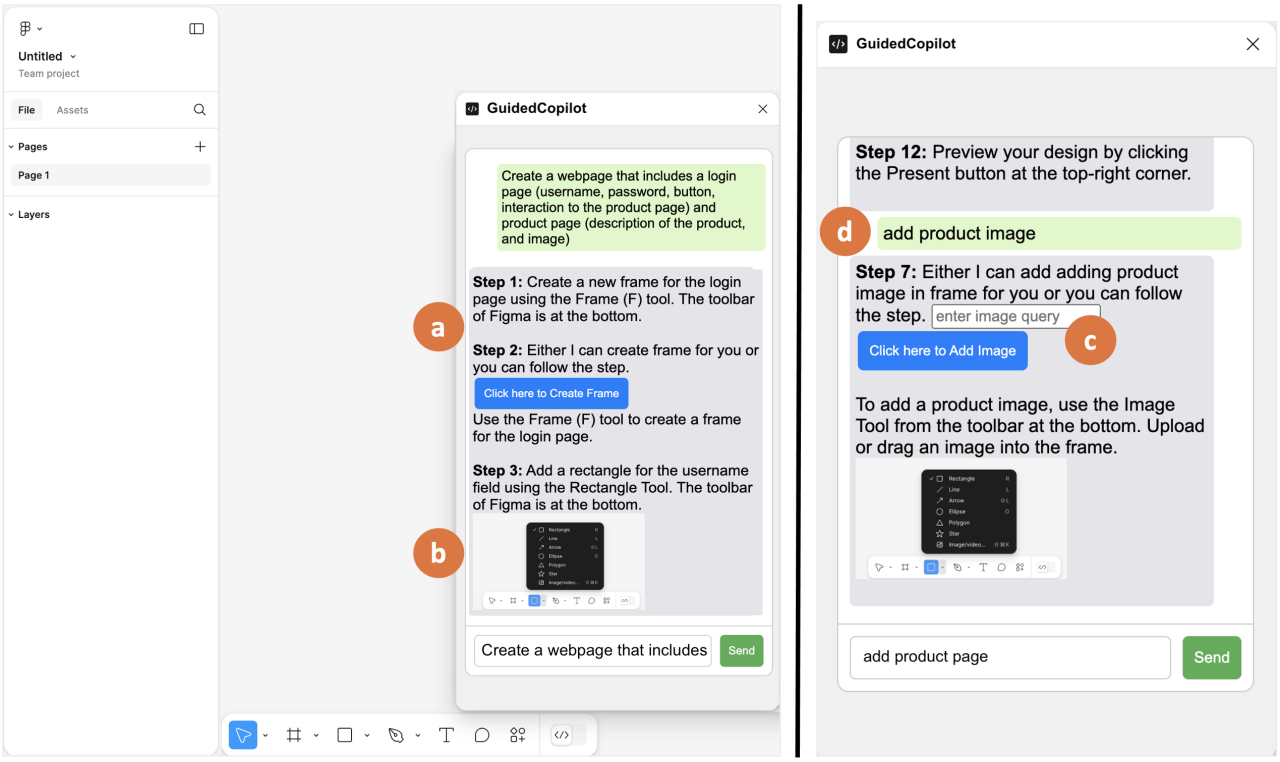

Figure 1: GuidedCopilot, a novel semi-automatic copilot: (a) Copilot assistance is structured to provide step-by-step guidance along with semi-automation for only repetitive or trivial steps in the task; (b) Visual references of the UI elements in-context to user's tasks and application are provided within the step-by-step guidance; (c) Users have control over editing the LLM extracted entities from their query before the semi-automation is performed; (d) Up-to-date mixed-medium follow-up responses are provided. To see the contrast with the fully automated copilot assistance, please see AutoCopilot in Figure 2.

"copilot" truly means to users [74]. We examine two distinct design paradigms for automation [8, 23, 37]: whether users prefer copilots that fully automate tasks ("Do It For Me"), as seen in fully automated systems, or those that serve as assistants, combining semi-automation for repetitive tasks with guided support and instructions for learning complex tasks ("Do It With Me"). We investigate how these design paradigms (semi-automation vs. full automation) impact task completion and user perceptions of software copilots, considering factors such as user expertise (novice vs. expert), familiarity with LLMs, and the nature of the tasks (fixed vs and creative, exploratory).

To study these design paradigms, we designed and implemented two in-application copilot interventions. The first, 1) AutoCopilot ("Do It For Me"), fully automates software tasks (Figure 2) based on the user's textual prompt, inspired by state-of-the-art copilot assistants (e.g., [24, 25, 77]). The second, GuidedCopilot ("Do It With Me"), is a novel semi-automatic copilot that automates only trivial or repetitive tasks while offering step-by-step visual guidance to help users locate UI elements (Figure 1). GuidedCopilot integrates visuals into its responses, allows users to initiate the automation process and enables corrections before proceeding. The research questions guiding this investigation were:

- RQ1: How do users perceive the utility, control, and potential for software learnability of in-application copilots designed using two different paradigms: full automation ("*Do It For Me*") and semi-automation ("*Do It With Me*")?
- RQ2: How can we design an in-application semi-automated copilot that provides semi-automation along with step-by-step visual guidance?

We conducted a within-subject controlled experiment (N=20) and follow-up interviews to compare the strengths and weaknesses of AutoCopilot vs. GuidedCopilot embedded in two different software applications: *Google Sheets*, an online spreadsheet application and *Figma*, an online user interface design application. While AutoCopilot was valued by some participants, particularly males with CS backgrounds, for saving time on specific tasks, GuidedCopilot consistently outperformed in user control, software utility, and software learnability. These findings were further supported by higher task completion rates and accuracy scores for GuidedCopilot. Participants expressed that GuidedCopilot provided higher user control when performing or customizing complex tasks, whereas AutoCopilot's full automation approach often did not align well with task requirements, leading to time-consuming rounds of trial-and-error and debugging.

Although participants appreciated GuidedCopilot's in-context assistance, some felt that it provided overly detailed instructions for already-familiar tasks, while others struggled to map visuals in

the chat to the corresponding UI elements. To address this, we explored design improvements to enhance semi-automatic copilots by integrating real-time context of users' progress and the application state. We developed two key features (See Figure 7): GuidedCopilotVisual (Visual Step-through), which embeds context-specific preview clips in the software interface; and GuidedCopilotADP (Adaptive Mixed-medium), which adapts and tailors instructions based on user needs and the state of their task progress. In a follow-up usability study (N=10), expert users appreciated GuidedCopilotADP's ability to adapt instructions to their proficiency levels, while novices valued the clickable navigation and the ability to skip steps. All participants found the preview clips helpful for building a mental model of the user interface, as they directly highlighted the relevant icons needed to complete each step based on users' progress and application state.

The key contributions of this paper are threefold: (1) we examine two distinct design paradigms for automation through the design and implementation of GuidedCopilot, a novel copilot that combines semi-automation with step-by-step visual guidance; and AutoCopilot, a fully automated copilot; (2) empirical insights into the strengths and weakness of AutoCopilot vs. GuidedCopilot, for completing software tasks, focusing on perceptions of software utility, user control and potential for software learnability; and, (3) a further design exploration of GuidedCopilot, introducing two new features, GuidedCopilotVisual and GuidedCopilotADP, that can enhance visual integration and deliver more targeted, task- and state-aware assistance, pushing the boundaries of in-context copilot support. We synthesized the insights from our studies into key factors (See Figure 8, Section 8.2) and levels of automation along with guidance to consider for effective human-AI collaboration in feature-rich software environments. These findings offer important implications for HCI and AI research by providing a clearer understanding of how to balance automation and user control in software copilots. Future research and applications can build on these insights to develop more adaptive and user-centered AI assistants that improve task efficiency, user autonomy, and software learning experiences.

2 Related Work

This research builds upon insights from prior work on software learnability, LLM-based in-application assistants, and challenges and opportunities in human-AI interaction in domains such as programming and software development.

2.1 Learning and Seeking Help for Feature-Rich Software

Seeking help for feature-rich software is often challenging due to scattered learning resources that require precise queries for successful retrieval [29, 33, 43]. Despite the expansion of help formats beyond traditional documentation and manuals [62, 70], novice end-users face the *vocabulary problem* [29] as they try to locate and apply relevant help from online tutorials, Q&A or FAQ sites, blogs, dedicated forums [43], and videos [44, 48]. Many resources are either outdated or too general to address specific needs [43, 83]. While online communities offer targeted and personalized help, they often involve delays and social barriers that discourage user participation [43, 83]. In-context help techniques [11, 16, 32, 34, 47, 53, 82] that assist people within their current task and/or application and step-by-step guidance with visuals and demonstrations [32, 41, 48, 49, 61, 64, 88] can be particularly useful for helping users within the context of their tasks [16, 41, 43, 53].

AI-based in-application assistants have also been explored in HCI and AI research to provide customized, context-specific solutions within software environments [19, 31, 41, 49, 58, 68]. Early systems, such as SmartAidè [68] and Crystal [58], employed Machine Learning (ML) and Natural Language Processing (NLP) to interpret user intent and offer guidance. However, unlike modern copilots powered by generative AI foundation models capable of tackling complex, open-ended tasks, the traditional AI systems based on simpler models [9, 19, 23, 31, 36, 58, 68, 75] were limited to handling narrow range of application-specific queries. Users often found the text-based assistance provided by these systems challenging to locate and use within the application's interface [19, 31, 58, 68]. Recent systems, such as Appinite [49] and ChatrEx [41] offer usability improvements by combining text and visual cues for task-specific queries. However, these systems still face challenges with training sets, limited adaptability across applications [49] and difficulties in supporting ambiguous or contextually complex queries [41]. Building on these principles and insights from prior work, our novel semi-automatic copilot, GuidedCopilot, automates only trivial or repetitive tasks while providing step-by-step guidance with visual references. In our follow-up design exploration, we further explored variations of GuidedCopilot providing assistance dynamically adapting to the user's progress and the state of the application.

2.2 Emergence of In-Application Software Copilots and LLM-based Assistants

LLM-based assistants have emerged as a "one-stop solution" [3, 84], enabling users to seek help using natural language queries with less syntactical precision [89]. However, challenges persist in crafting effective prompts [10, 42, 72, 86, 89], translating text-heavy outputs into actionable visual instructions, and in applying these LLM outputs within the software interfaces [42]. Emerging in-application LLM assistants or copilots [74, 77], such as Microsoft 365 Copilot [77], Adobe Firefly [2], etc.) offer ways to automate software tasks based on prompts. However, users face new challenges with these new LLM-based capabilities. Users have to navigate and build mental models of both the software's complex features [42, 43] and the AI capabilities of the copilots, all while maintaining control in dynamic environments [41, 42, 52]. These hurdles have led to declining adoption rates for some copilots, with many users choosing to entirely abandon them [74].

Recent efforts to optimize copilots (e.g., for productivity or creative tasks) have largely focused on technical improvements, often without adequately considering user needs or incorporating insights from user studies [80]. Research shows that users prefer *learning by doing* and resort to self-directed experimentation within an application [14, 43, 60, 70, 71]. The fully automated processes, while impressive, tend to bypass intermediate steps that prevent users from fully understanding specific software features

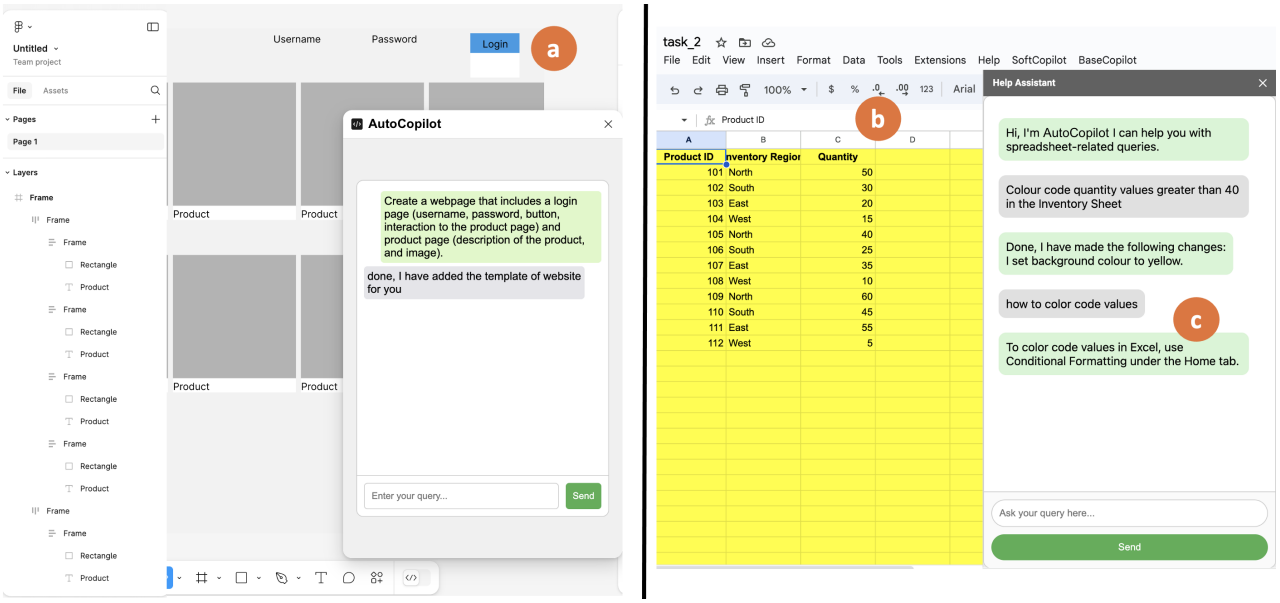

Figure 2: AutoCopilot: (a) Fully automates the user's task (e.g., creating a webpage that includes a login and product page); (b) Similar to state-of-the-art copilots, demonstrates incorrect automation (such as color coding the entire sheet instead of values greater than 40 in column C); (c) Provides follow-up textual response based on context from software documentation

and task-specific terminology [39, 74]. Recent literature also highlights concerns about balancing autonomy and agency, long debated in traditional AI systems [23, 36, 75]. However, such concerns take on new dimensions in sophisticated LLM-powered copilots embedded in complex, feature-rich software tasks. In particular, the open-ended nature of interactions and the multifaceted landscape of user behaviors and lack of accurate user mental models poses unique challenges [42]. This creates a new opportunity to provide more effective automatic and semi-automatic assistance. A critical question arises: do users prefer full automation or a balanced approach with semi-automation and guidance? Our study contributes new insights into how users perceive these two distinct design paradigms for automation in copilots (semi-automation vs. full automation) when using feature-rich software.

2.3 Human-AI interaction with LLM Assistants for Programming Tasks

Although in-application software copilots have only recently started emerging, we can draw upon empirical studies of copilots for task-based assistance in domains such as programming and software development (e.g., GitHub Copilot within VS Code). While these copilots can automate code generation, they often require users to invest additional effort in reviewing and customizing the generated output to align with their actual intent and requirements [10, 78, 81, 87]. These efforts exacerbate when the copilot generates incorrect automation (e.g., code generation) without explaining its actions, requiring users to invest time in understanding the generated output [10, 39, 81]. These findings suggest that users seek more than mere automation—they desire a deeper understanding of the syntax and steps involved in their tasks [87].

Recent work has explored ways to better interpret complex, nuanced user inputs and generate more contextually relevant responses [3, 30, 84]. However, despite spending time understanding LLM-generated code, users often find it harder to comprehend than their own code [4, 10, 50, 57, 86]. As a result, many solutions focus on enhancing comprehension and explainability of the generated code to help users handle both familiar (where programmers already have some idea how to write the code) and complex programming tasks more effectively [22, 87]. For example, the IVIE system supports in-situ understanding of code by augmenting the editor to help programmers grasp the generated content. Excessive automation can reduce user agency, while insufficient automation can make the assistant less effective or frustrating to use [39]. Our study extends these insights to the space of software copilots. Our main goal was to investigate how different levels of automated in-application help (semi-automation vs. full automation) impact task completion and user perceptions when using feature-rich software and we explored this by designing and implementing two interventions: AutoCopilot and GuidedCopilot, described below.

3 Intervention 1: Design Considerations and Implementation of *AutoCopilot*

Our first copilot intervention, AutoCopilot ("*Do It For Me*"), was based on full automation principles [8, 23, 37] to serve as a realistic baseline (See Table 1 for comparison of both copilots). Such fully automated systems [8, 23, 37] are designed to maximize efficiency and task completion, often with limited or no human interaction. Drawing inspiration from emerging automatic copilots [2, 77], AutoCopilot retrieves pertinent information from software documentation and web data to facilitate users in asking follow-up questions and presents a targeted and specific textual response to the user.

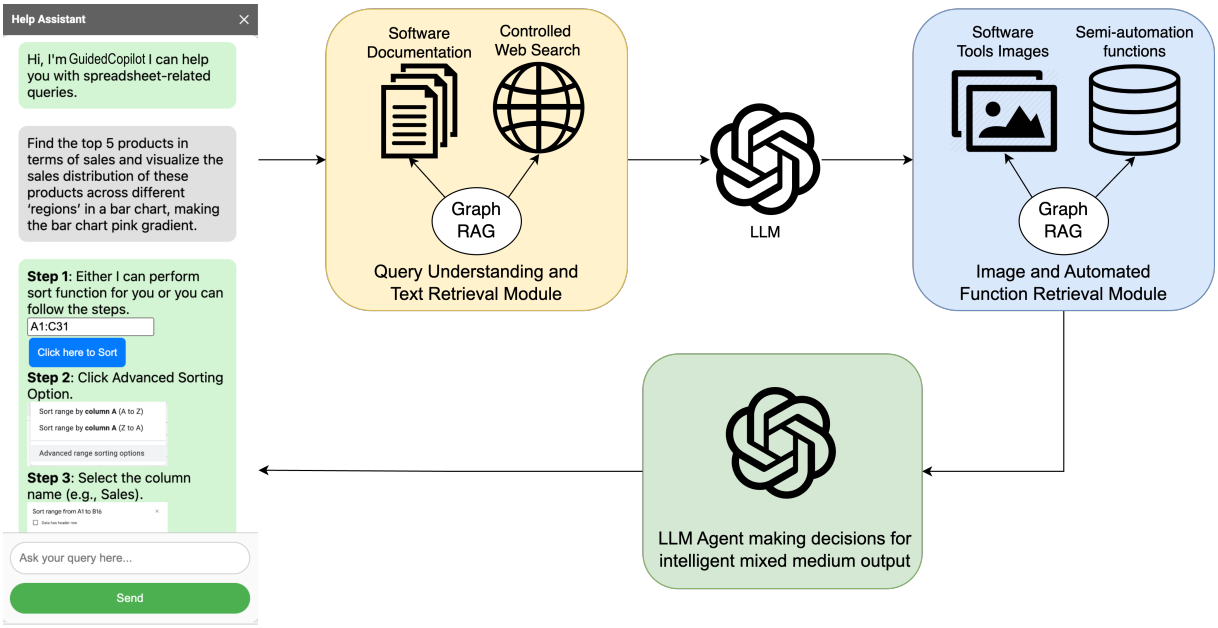

Figure 3: GuidedCopilot Architecture: The user's query is used to initiate a conversation about automating software tasks, which is then transmitted to the query understanding and text retrieval module (Section 4.1.1). This module interprets the query and performs a contextual search across documentation and web data. The extracted intent and relevant excerpts are processed by GPT-4o to generate text-based procedural steps. These steps are sent to the Image and Automated Function Retrieval Module (Section 4.1.2) for corresponding visual aids and automated functions, sourced from a curated dataset. Finally, the LLM agent integrates these text, visuals, and semi-automated functions into a cohesive, mixed-medium response tailored to the user's software-related query.

AutoCopilot sometimes also demonstrates incorrect full automation, similar to modern copilots [17, 26]. Based on these design considerations, we derived two key design goals for AutoCopilot:

(1) DG1: Provide full automation for the entire software task
(2) DG2: Offer concise follow-up responses based on context from the software documentation and web data, similar to in-application state-of-the-art copilots

3.1 AutoCopilot: User Interface Design and Implementation

To maintain consistency and control in our experimental conditions, we chose to implement AutoCopilot ourselves to facilitate an easier data collection process. Our AutoCopilot UI (Figure 2) enables users to input queries, which are then processed by GPT-4 to identify intent and extract relevant entities. We created a dataset of basic automation functions tailored to the study's tasks, using AppScript in Google Sheets and JavaScript in Figma to control UI elements. We included instructions in the prompt for AutoCopilot to always attempt full automation for user queries. We implemented a hybrid-RAG using OpenAI's text-embedding-3 model [63] and Elasticsearch [1, 73] to index the descriptions of these full automation functions and software documentation. These automation functions are then retrieved based on the user's intent and combined with extracted entities from the user's query to generate the final output. The pertinent information from software documentation is used to generate a concise textual response and facilitate users' follow-up debugging questions. We integrated LLM-based web search for up-to-date follow-up responses when existing documentation fell short. To mirror real-world automation performance, AutoCopilot occasionally encounters incorrect full automation when it fails to identify a single automation function from the multiple intents in the user's query for performing complex tasks outlined in the study, or when it cannot accurately map multiple entities from the user's query to the appropriate automation function. For example, in sorting data and applying color coding based on a condition, AutoCopilot may incorrectly map the entities to color code the entire datasheet rather than just the sorted values. Additionally, when the hybrid-RAG in AutoCopilot fails to identify relevant automation functions related to the detected intent, it shows a breakdown with an error message. To ensure consistent reliability of automation across both copilot conditions, we employed the same software documentation for RAG, ensuring both copilots generated consistent response/automation quality and accuracy while minimizing hallucinations. Additionally, the researchers conducted sanity checks for both copilots to verify response accuracy.

4 Intervention 2: Design Considerations and Implementation of *GuidedCopilot*

We implemented GuidedCopilot to reflect semi-automated systems that combine partial automation with user interaction and guidance to facilitate incremental learning [8, 23, 37, 74]. Drawing inspiration from software learnability research (discussed above), we considered how to structure the copilot assistance semi-automatically

and how to include visuals in the step-by-step guidance. We developed four key design goals to enhance copilot assistance using visuals and semi-automation.

Structuring the copilot assistance: As copilot-driven task automation increases, crucial steps for users to learn software features are often skipped, reducing user control [74]. When automation fails, these skills take control and manually perform these tasks. In such cases, GuidedCopilot should provide guided help with visual references as this approach has been shown to be beneficial for feature-rich software tasks [41, 43]. To balance user control and automation benefits, GuidedCopilot should offer: (i) semi-automation for repetitive or trivial steps in the task, (ii) user control to trigger semi-automation when needed and, (iii) human intervention to verify LLM-extracted data before automation proceeds. For example, simple tasks like "applying a sort filter in Google Sheets" or "creating a frame in Figma" could be automated, while detailed assistance could be provided for the more complex tasks.

Enhancing the step-by-step guidance with visuals: In-context GUI visuals [41, 49, 64] have shown to be effective in software help-seeking. Since users often struggle to locate and use the correct UI elements when only text-based guidance is offered [3, 12, 42, 77], copilots provide relevant in-context visual references within the chat. These visuals, tailored to the user's specific task and application, can help users locate and use software features, such as the text tool in Figma or advanced sorting option in Google Sheets.

Based on the above considerations, we derived four key design goals for building GuidedCopilot:

(1) DG1: Provide step-by-step guidance with semi-automation for repetitive or trivial steps.
(2) DG2: Provide visual references to help users locate and apply the necessary software features.
(3) DG3: Allow users to initiate semi-automation and edit LLM-extracted entities before proceeding.
(4) DG4: Offer up-to-date mixed-medium support, combining web information with software documentation.

4.1 GuidedCopilot: User Interface Design and Implementation

Based on the above design goals (Section 4), we designed and implemented GuidedCopilot (Figure 1), a novel semi-automatic copilot that automates only trivial steps while providing step-by-step visual guidance. Similar to AutoCopilot, we embedded GuidedCopilot within Google Sheets and Figma to demonstrate its feasibility and scalability across diverse feature-rich software. Figure 3 illustrates the overall architecture of GuidedCopilot, which can apply to any complex software with comprehensive software documentation, visual examples such as UI elements and screenshots, and an active user community on Q&A platforms or forums. Additionally, the underlying software should support task automation through scriptable frameworks, making it suitable for a range of softwares, from productivity suites like Office 365 or Google Workspace, as well as creative tools such as Photoshop and AutoCAD.

4.1.1 Query Understanding and Text Retrieval Module: When a user submits a query in the GuidedCopilot UI, this module interprets the user's intent and retrieves contextually relevant information for generating accurate responses by implementing the state-of-the-art Graph-based Retrieval Augmented Generation (GraphRAG) [67] using GPT-4o and the BAAI/bge-base-en embedding model [85]. GraphRAG uses NLP techniques and LLMs to construct dynamic knowledge graphs from documents, linking entities within and across sentences, outperforming traditional keyword searches and vector-based retrieval methods. To implement the GraphRAG, we used the following submodules:

Indexing: We implemented LLM-based web searches to provide up-to-date responses for follow-up user queries when existing software documentation is insufficient. Using GPT-4o, we extracted search topics from user queries and conduct searches on targeted websites, such as Google Sheets and Figma Q&A forums. The dynamically retrieved web data and documentation are segmented into discrete TextUnits or chunks, from which entities (e.g., data items, software functions, GUI elements) and their relationships are extracted to construct the knowledge graph. The chunks are then transformed into vectors and indexed in a vector database for integration with GraphRAG.

Querying for fetching relevant data: After indexing, we performed querying on the index of software documentation and web data to extract relevant entities and paragraphs based on the user's query. These extracted entities and paragraphs were subsequently used as in-context information for prompts to GPT-4o to generate stepwise textual responses to the user's query, which were further utilized in Section 4.1.2 to retrieve relevant images and semi-automation functions for integration into the appropriate steps.

4.1.2 Image and Semi-automation Function Retrieval Module. This module consists of the following key components:

Image and semi-automation function data preparation and indexing: We scraped images from software documentation showcasing tool functionalities and UI elements, extracted their descriptions and created an index using GraphRAG. We leveraged scripting capabilities in the form of plugins or extensions using HTML and JavaScript, often supported by feature-rich applications, to control UI elements and create a set of semi-automation functions (e.g., sorting data, creating charts in Sheets, or creating frames and buttons in Figma). We identified such 25 repetitive tasks for each application and created generic function templates capable of handling diverse user queries, which were then scripted to automate the tasks and indexed using GraphRAG.

Querying for relevant images and semi-automation functions: Finally, each text-based step generated by GPT-4o in response to the user's query (See Section 4.1.1) is treated as an individual GraphRAG query to retrieve relevant images and suitable semi-automation functions from the prepared indexes in Section 4.1.2. These retrieved images and semi-automation functions are then provided as input to the LLM agent.

4.1.3 LLM Agent for Mixed Medium Output. We developed an LLM agent that decides how to strategically integrate the retrieved visuals and automation functions within LLM-generated textual step-by-step response. Following the design goals (DG1 and DG2) as instructions, the LLM agent intelligently decides where to place these retrieved visuals and automation functions within the textual steps, producing a mixed-medium output presented to the user

	GuidedCopilot ("Do It With Me"): Semi-Automation Based Copilot	AutoCopilot ("Do It For Me"): Full Automation Based Copilot
Automation	Automate repetitive or trivial steps of the task (e.g., create a frame, shape, sort function, etc.), initiated by users and positioned at suitable steps within step-by-step guidance User can edit the LLM extracted entities from their query prior to performing semi-automation	Fully automate the entire software task (e.g, color code the quantities in column c greater than 40)
Step-by-step Visual guidance	Provides guidance on the task process. Includes visual references in-context of a user's tasks and application to help locate software features or functions and effectively apply them within the application	No
Software Context and Follow-up questions	Up-to-date mixed-medium response by leveraging latest relevant information on the web along with software documentation	Up-to-date textual response by leveraging latest relevant information on the web along with software documentation

Table 1: Comparison of two distinct design paradigms for automation: AutoCopilot ("*Do It For Me*") vs. GuidedCopilot ("Do It With Me")

through the GuidedCopilot UI. Similar to AutoCopilot, GuidedCopilot occasionally generated errors and exhibited uncertainty, when its GraphRAG fails to identify relevant semi-automation functions for the detected intent.

Both the copilot UIs module was built as an in-application assistant and migrated to Chrome as an extension for Google Sheets and as a plugin for Figma. To allow users freedom and control in accessing the copilot UIs, we included the close options for closing it anytime during the interaction. The code repository and prompts for both copilot interventions are available upon request.

5 Evaluation: Controlled Experiment and Follow-up Interviews

To investigate the strengths and weaknesses of our two copilot interventions, we conducted a controlled experiment and follow-up interviews with 20 participants.

5.1 Participants

We recruited 20 participants (10F|10M) for our study, focusing on non-AI expert users with little to no prior experience or knowledge of ML or NLP; however, 4/20 participants had previous experience with ML or NLP. Our participants came from both CS (9/20) and non-CS (11/20) backgrounds (including Engineering, Accounting, History, Theory, Communication, and Arts) and professions (administrative, logistics, information designers, students, researchers, professional data scientist, and software developer). Participants were familiar with LLM-based assistants like ChatGPT (15/20), GitHub Copilot (7/20), and only 5/20 had used Microsoft 365 Copilot for software tasks before this study. About half of the participants (9/20) had frequently used Google Sheets and Excel applications and the remaining were occasional users. None of the participants had used Figma before. Our participants covered a range of age groups: 18-24 (32%), 25-34 (47%), 35-44 (16%), 55-65 (5%) and had different levels of education (1 Post-Secondary, 2 Diploma, 7 Bachelor's, 6 Master's, 4 PhD). We recruited participants mainly from our university's mailing lists and found additional participants through snowball sampling.

5.2 Study Design and Procedure

We used a within-subject design to reduce participant variability, employing a Latin Square counterbalancing [69] method to randomize the order of 2 copilot conditions (4 possible orders). Each participant completed two tasks per copilot (4 tasks in total), using one feature-rich application before transitioning to a second application, with the order of the tasks and copilots randomized.

At the beginning of a session, we introduced both copilots and provided tips for interacting with the application. Participants completed a demographic questionnaire on their background and prior experiences with different software applications, LLM assistants, copilots, and chatbots. Each copilot intervention was presented in random order, and tasks were designed to be complex enough to take at least 8 minutes, regardless of participant familiarity and prior experiences. After each task, participants completed a post-task questionnaire to evaluate their overall experience with copilot assistance, focusing on utility, user control, and potential for software learnability. We encouraged participants to think aloud [59] and reminded them that the study was seeking to understand how they seek copilot assistance rather than their individual task performance. Lastly, we conducted follow-up interviews to probe into any difficulties that participants faced and their overall perceptions of different levels of automation. Each session lasted approximately one hour and participants received a $15 Amazon gift card.

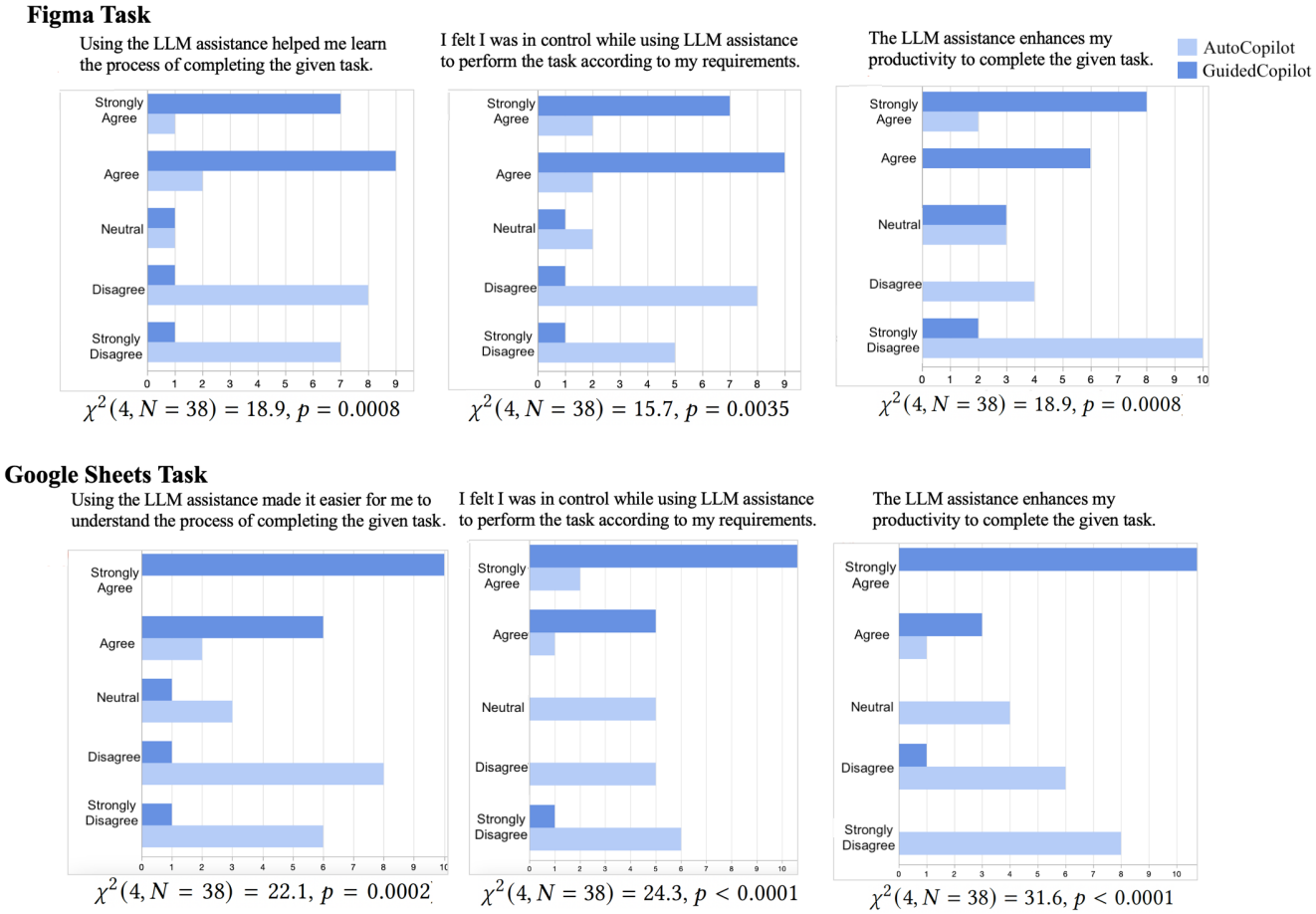

Figure 4: Overview of participants' responses to the post-task questionnaire. The Pearson Chi-Squared test showed a significant difference for each metric across both copilot interventions for completing the Google Sheets and Figma tasks. With GuidedCopilot, users demonstrated higher task completion and higher accuracy and indicated that GuidedCopilot helped them learn the software-specific steps, provided users more control and enhanced their productivity compared to AutoCopilot.

5.3 Choice of Tasks and Applications

We selected Google Sheets and Figma after exploring various other productivity and design-oriented applications (e.g., PowerPoint, Photoshop, Word), as they cover a range of tasks involving visual interactions, statistical functions, and interface design. To observe potential challenges copilot-assisted software help, we selected tasks that would require multi-stage help and prompts. For example, one of the tasks in Google Sheets was to use a copilot to analyze the top 5 products and visualize their sales across regions using advanced sorting and a custom bar chart. Similarly, in Figma, one of the tasks was to design a web page that includes a login section and displays products using copilot assistance.

5.4 Data Collection and Analysis

We recorded each participant's screen and audio recorded their interview responses. We focused on two key aspects: how they used copilot assistance and different levels of automation to complete the prescribed tasks in Google Sheets and Figma, and how they formulated and refined prompts to seek help.

We ran Pearson's Chi-square test for independence with between "Copilot Interventions" (having 2 levels: GuidedCopilot and AutoCopilot) and user responses (having 5 levels, Strongly Agree to Strongly Disagree) to quantitatively determine the significance of the results. The experimenter, in consultation with all authors, compared task completion and accuracy against the ground truth, defined as the optimal sequence of steps and ideal application of software features (See Appendix A for details). To measure the task completion, we evaluated how many task steps (e.g., sequence of features/ functions) users completed using the copilots. To measure task accuracy, we assessed their success in correctly identifying and applying the sequence of features and functions (e.g., sort or VLOOKUP function, creating buttons or shapes). To study trial-and-error strategies, we manually annotated interaction logs to analyze repeated, varied attempts within user interactions, revealing non-linear task paths, repetitive actions, and corrective sequences [13].

Finally, to complement our experimental findings, we corroborated the data with participants' think-aloud verbalizations and probed into the reasons influencing users' perception of utility, user control, potential for software learnability. We used an inductive analysis approach [18] and affinity diagrams [18] along with discussions among the research team to categorize the interview findings and identify key recurring themes.

6 Results

We have organized our results around key themes, evaluating each copilot's strengths and weaknesses in terms of utility, user control, productivity, and software learnability. We also report task completion rates, analyze user interactions to identify trial-and-error patterns, and highlight users' challenges and debugging strategies.

6.1 Task Completion, Software Utility and User Productivity with Copilot Interventions

6.1.1 Task Completion and Task Accuracy: None of the participants fully completed the tasks in Sheets or Figma using AutoCopilot. In contrast, with GuidedCopilot, 11/20 participants successfully completed the tasks in Sheets, and 6/20 completed the Figma tasks, despite Figma being new to all users. On average, participants completed 35.0% of the Sheets tasks (maximum= 50% and minimum= 0%) with AutoCopilot, compared to 88.5% (maximum= 100% and minimum= 70%) with GuidedCopilot. A paired-sample t-test further revealed a significant difference between the two copilots ($t(37.6) = 11.2$, $p < 0.0001$, two-tailed). For Figma, the task completion rate was 20.0% (maximum= 50% and minimum= 0%) with AutoCopilot and 55.0% (maximum= 75% and minimum= 30%) with GuidedCopilot, also demonstrating a significant difference ($t(37.7) = 7.2$, $p < 0.0001$, two-tailed).

Among the portion of the task completed by each participant, task accuracy was significantly higher with GuidedCopilot across both applications. In Sheets, the average task accuracy was 82.0% (maximum= 100% and minimum= 50%) with GuidedCopilot, compared to 12.0% (maximum= 50% and minimum= 0%) with AutoCopilot ($t(37.9) = 17.2$, $p < 0.0001$). In Figma, the average task accuracy was 40.0% (maximum= 60% and minimum= 20%) with GuidedCopilot, and only 5.0% (maximum= 30% and minimum= 0%) with AutoCopilot with significant difference ($t(30.7) = 10.4$, $p < 0.0001$) and no order effects were observed.

6.1.2 User Perception of Copilot Interventions on Productivity and Software Utility: Most participants (18/20) found that GuidedCopilot enhanced their perception of productivity in completing Sheets tasks compared to AutoCopilot), and these results were significant ($\chi^2(4, N = 38) = 31.6$, $p < 0.0001$), and align with task completion and accuracy scores. Even though Figma was new to all participants, most (14/20) felt that GuidedCopilot enhanced their perception of productivity over AutoCopilot with significant differences ($\chi^2(4, N = 38) = 18.9$, $p = 0.0008$). Participants reported that AutoCopilot often reduced productivity by generating excessive or irrelevant automation, requiring substantial effort to adjust: *"[Figma task]: I just need enough boilerplate to think through, but not too much. In this boilerplate, I had to spend extra time to change it...I felt like I was hitting my head against the wall...ends up wasting my time (P03)."* Conversely, 12/20 users appreciated that GuidedCopilot's user-initiated automation and visual instructions, which saved them time and enhanced productivity: *"I like the* GuidedCopilot*...the way it gives a mix of everything...text, images, and buttons...helpful and increases my productivity. It creates the frame, fields, like a built-in component interacting with the software...that's step ahead (P11)."*

Overall, most users preferred GuidedCopilot over AutoCopilot across all key measures (See Figure 4). However, a few users (6/20), primarily male computer science professionals, favored AutoCopilot due to its complete automation and perceived time savings: *"I would go with the complete automation...I am a professional and I don't care about how to make shapes in Figma. I just want to get the work done...it gives me a template I can improve and work with it (P17)."* These participants were comfortable debugging templates in Figma and felt it was easier than handling complex formulas and data manipulation in Sheets. An older participant (>55 years) also preferred AutoCopilot due to limited patience for manual experimentation: *"As an older person... my patience is not high, I just want it to do it for me (P19)."*

6.2 User Perception of Control with Copilot Interventions

We next evaluated how participants felt about their ability to control and use the copilots : (i) while performing tasks, and (ii) when handling incorrect automation generated by copilots.

6.2.1 In-Task Performance. Most participants perceived greater control with GuidedCopilot than with AutoCopilot when performing tasks in both Sheets (18/20) and Figma (16/20), with significant differences (Sheets: $\chi^2(4, N = 38) = 24.3$, $p < 0.0001$; Figma: $\chi^2(4, N = 38) = 15.7$, $p = 0.0035$). In contrast, with AutoCopilot, many participants (11/20) found full automation to be "a gamble", especially in Figma which required extensive customization for visual tasks: *"It felt like LLM 1* [AutoCopilot] *was doing it for me, which is a bit of a gamble...[as] I was little unsure of the result, whereas LLM2* [GuidedCopilot] *gave option of doing it with me, or here are the steps to do it myself (P04)."* With Sheets, many participants (13/20) also reported lower perceived control with AutoCopilot, as the lack of validation left them uncertain about the accuracy of complex tasks: *"It* [AutoCopilot] *felt like it had a mind of its own...it was doing whatever it wanted and all at once. I felt helpless and couldn't understand what went wrong. With data [tasks], things can get pretty bad (P12)."*

In contrast, GuidedCopilot offered flexibility, letting users choose between step-by-step guidance or selectively using automation at suitable steps: *"I would prefer* [GuidedCopilot]*...gave me more control to do it myself or get it done....not like a one-size-fits-all, and you can kind of fit or modify it to what you need it to do (P15)."* About half the participants (9/20) explicitly appreciated GuidedCopilot's detailed instructions that helped them verify the automation and reduce errors: *"I feel more in control with step-by-step guidance. I can understand the process of that [task]...whereas If it did everything at once, I might not like a step in the middle...that did not get the results I was expecting and I'd have to undo (P08)."*

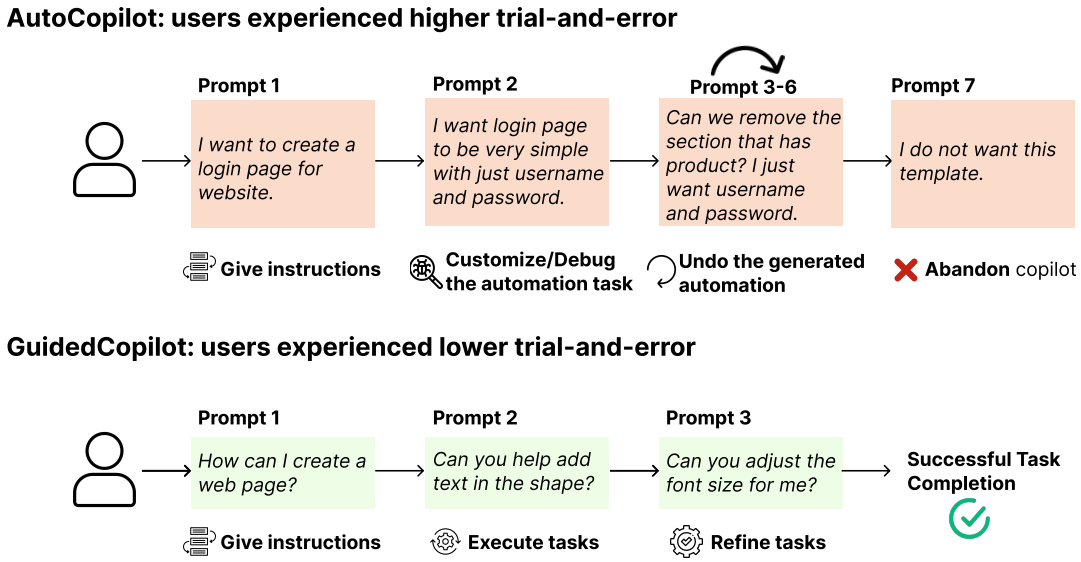

Figure 5: Trial-and-Error Differences in AutoCopilot vs GuidedCopilot: This figure illustrates the case of Participant P14, a computer science professional. Despite P14's technical expertise, they encountered higher trial-and-error with AutoCopilot, primarily focused on customizing and debugging the generated automation (e.g., a webpage template) and reversing (or undo) in case of incorrect automation, which ultimately led the user to abandon AutoCopilot. In contrast, P14 experienced fewer trial-and-error instances with GuidedCopilot, mostly related to executing and refining tasks like resizing or altering colors. The user was able to complete the task successfully with GuidedCopilot.

6.2.2 Handling Incorrect Automation. When AutoCopilot produced incorrect automation, such as generating incorrect web page templates or coloring an entire spreadsheet instead of only quantities greater than 40 in a column, users often felt derailed and resorted to trial-and-error strategies. Participants engaged in 192 trial-and-error attempts with AutoCopilot, nearly double the 89 attempts with GuidedCopilot. On average, users made five trial-and-error attempts per task with AutoCopilot (range: 3-10), compared to fewer than two with GuidedCopilot (range: 1-5) across both applications. About 75% of AutoCopilot attempts involved undoing incorrect automation, 55% focused on further debugging or editing, and 10% focused on locating and applying the assistance. These challenges led some participants (6/20) to abandon AutoCopilot entirely.

In contrast, with GuidedCopilot, 55% of 89 trial-and-error efforts focused on enhancing or customizing tasks (e.g., resizing shapes or altering text colors), with only 15% focused on correct the automation. For example, P14, faced numerous debugging challenges (Figure 5) with AutoCopilot's webpage template. However, with GuidedCopilot, they only made minor adjustments, such as resizing or altering colors: *"AutoCopilot led me into a spiral of figuring out what to do next...I just wanted a simple page with a username/password, but it kept adding things back and [would] not let me change certain things I wanted. I felt lost and caught in a spiral of bad [prompting] decisions (P14)."*

6.3 User Perception of Potential for Software Learnability

Most participants (16/20) indicated that GuidedCopilot improved their ability to learn the steps of the software tasks compared to AutoCopilot, with results significant for both Sheets ($\chi^2(4, N = 38) = 22.1$, $p = 0.0002$) and Figma ($\chi^2(4, N = 38) = 18.9$, $p = 0.0008$). Participants found GuidedCopilot's in-context visual cues particularly helpful for locating relevant menu items and navigating applications like Sheets: *"The best thing about this LLM* [GuidedCopilot] *is it showed images as a visual clue with instructions. I just had to match those little pictures with the icons in the sheets, sort of teaching and explaining the process which helped me learn how to use the software and execute the task (P16)."*

For unfamiliar applications like Figma, GuidedCopilot helped users learn basic software features, with half transferring their knowledge to new subsequent tasks (known as transfer learning, (See Figure 6). As P08 commented: *"GuidedCopilot helped me like a tutor, giving visuals and instructions that take you easily through the interface and helped me learn and gain confidence in using the software, so I learned how to create a frame, use the rectangle tool (P08)."* In contrast, many users (11/20) felt dependent on AutoCopilot, expressing doubt about whether they could perform similar tasks independently in the future: *"I don't want to rely entirely on the AI* [AutoCopilot] *to do everything; I want to learn the software while doing the task. AI should help us learn to complete tasks...not make us feel dependent on it (P05)."*

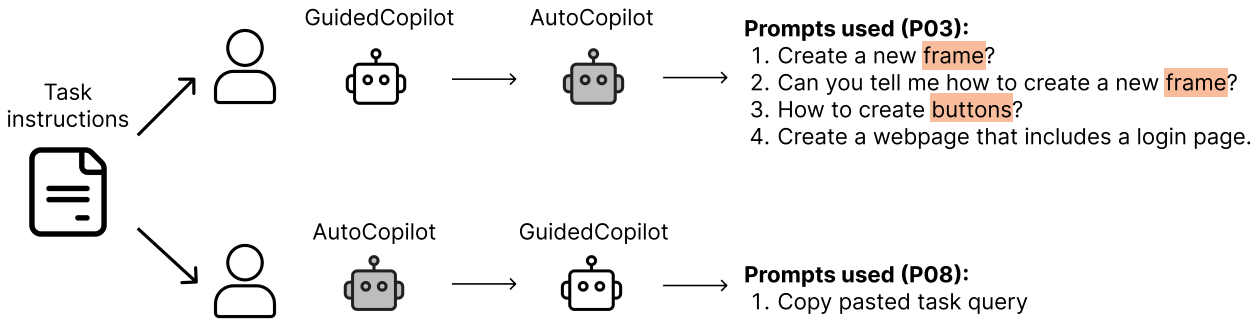

Figure 6: Illustration of Transfer Learning in Software Usage with GuidedCopilot: Participant P03 started with the GuidedCopilot condition where the step-by-step instructions helped them grasp fundamental software features (e.g., creating frames and buttons in Figma), which they effectively transferred when doing subsequent tasks. In contrast, P08, who started with AutoCopilot, did not demonstrate similar foundational knowledge and continued to rely on a repetitive prompting strategy of copy-pasting queries in follow-up tasks.

Overall, most participants (15/20) were enthusiastic about using GuidedCopilot for future software tasks: *"I like* GuidedCopilot *because it saves time for my accounting tasks to check on payments and offers both options-doing things automatically and showing how to do them myself (P16)."* Participants also expressed interest in using GuidedCopilot for other complex applications like Photoshop and SolidWorks.

6.4 Areas of improvement in GuidedCopilot

Despite the overall positive interaction with GuidedCopilot, a few participants faced minor usability issues, such as difficulty in mapping visuals within the chat to the software's UI: *"I could not...find where to locate the VLOOKUP [in Sheets]; maybe highlighting a section of the page with 'click here' would be more helpful (P11)."* A few participants (5/20) struggled with the suggested steps, especially when new to the software. Participants requested more animated instructions tailored to the real-time context of the task progress and the application's status: *"I could not use the rectangle tool...having an instructional tutorial that better understands the tasks and recognizes that I clicked on the menu bar with rectangle option and then give me instructions to learn how to use it would be helpful" (P19).* Finally, some advanced users wanted even *more* automation and fewer steps aligned with their current task progress.

7 Enhancing User Experience with Semi-Automatic Copilots: A Follow-up Design Exploration

Based on the insights from our first study, we identified key design considerations for further enhancing user experience with semi-automatic copilots that were rated as being more useful overall. We explored variations for semi-automatic copilots that build on GuidedCopilot to better integrate visuals and can provide more targeted, task- and state-aware assistance (See Figure 7). For this exploration, we selected Adobe Photoshop, as users had frequently expressed a desire to have access to semi-automatic copilots in this feature-rich application. To simulate both of these adaptive approaches, we employed the Wizard-of-Oz method [20, 66]. To capture the users' initial reaction to these designs, we conducted a usability study with 10 participants who had varying experiences with the software application.

7.1 Feature 1: GuidedCopilotVisual (Visual Step-through)

GuidedCopilotVisual (Figure 7b) offers dynamic step-by-step assistance using visual anchors and context-sensitive preview clips specific to the user's current tasks and the application state. Our design complements prior research that has shown the value of in-application instructional videos [27, 28, 40] and step-through demonstrations [32, 41, 48, 61] by incorporating users' task progress and application state. When users select "Show in the interface" (Figure 7a), GuidedCopilotVisual presents targeted preview clips that dynamically adapt to the user's progress and application state, visually demonstrating the next steps (e.g., after adding and resizing a moon, it suggests options for creating a reflection). To design this feature, we prepared the preview clips relevant to each step in advance, along with relevant images and screenshots, to support users in locating the corresponding menu items within the application.

7.2 Feature 2: GuidedCopilotADP (Adaptive Mixed-medium)

GuidedCopilotADP provides in-context mixed-medium assistance tailored to the user's current task and application state, dynamically adapting instructions based on the user's needs (Figure 7c). For example, if the user has already adjusted an element, GuidedCopilotADP skips to the next relevant step (e.g., step 3). We used GPT-4o to generate textual step-by-step instructions. The UI module is built using ReactJS and migrated to Chrome as an extension. The researcher monitored real-time task progress and the application's state and manually controlled which steps needed to be shown next, facilitating rapid prototyping and validation of various adaptive features while gaining insights into user interactions. The researcher played the preview clip based on the user's completed tasks and the next required steps, effectively mimicking a task- and state-aware adaptive system. To simulate semi-automation, the

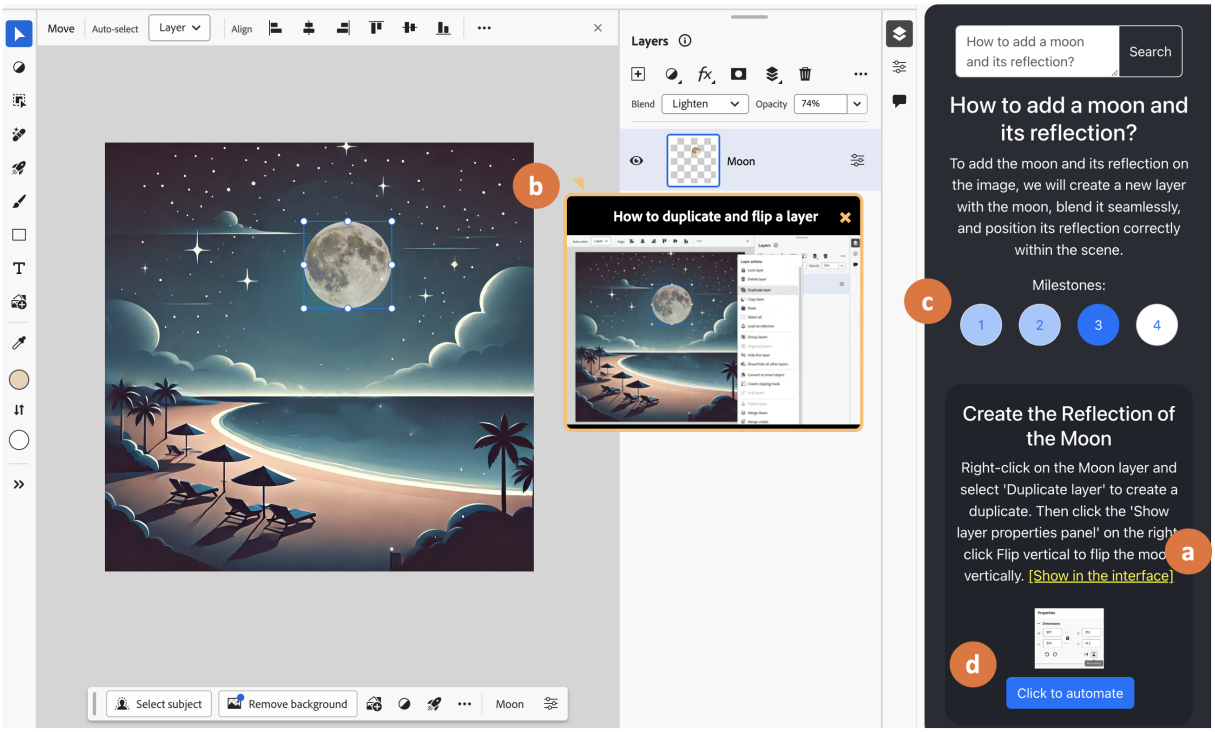

Figure 7: Design exploration building on GuidedCopilot to offer targeted, task- and state-aware assistance in Photoshop through two key features: (a, b) GuidedCopilotVisual integrates in-context preview clips within the software interface. When the user clicks "Show in the interface", the copilot detects the user's task progress and application state to display relevant preview clips for the next step (e.g., after adding and resizing the moon, the clip suggests the menu option for creating a reflection). (c) GuidedCopilotADP adapts the number of instructions based on user needs; for example, if the user has already adjusted the moon's size, it skips to step 3. Clickable milestones provide control and flexibility to navigate through instructions. (d) The user can opt for semi-automation by clicking on relevant options.

researcher executed these actions on behalf of the system whenever the user opted to automate a particular step.

7.3 Study Procedure and Participants

To study user's initial reactions, we conducted a usability study with 10 participants (4F|6M) aged 20-30, from different backgrounds (CS, Robotics, Engineering) and education levels (5 Master's, 3 PhD, 2 PostDoc). About half of the participants (6/10) were novices, while 4/10 were experienced Photoshop users. All participants were familiar with LLM-based assistants like ChatGPT, and some with GitHub Copilot (4/10).

Each session began with an introduction to both features within the copilot variation, followed by a demographic questionnaire on participants' backgrounds and prior experiences with LLM assistants and software applications, lasting 30 minutes in total. They were then asked to complete tasks using the web version of Adobe Photoshop with the GuidedCopilot plugin installed. The task involved adding a moon image and its reflection to a scene and integrating them seamlessly with the background (Figure 7), using both in-application copilot features (GuidedCopilotVisual and GuidedCopilotADP) for assistance. In follow-up interviews, we probed further into user reactions and overall experience. Participants were encouraged to think aloud throughout the session. Sessions were video and audio-recorded for transcription, including the actions on screen.

7.4 Findings

7.4.1 Users' reactions when using GuidedCopilotADP. All expert users (4/10) appreciated GuidedCopilotADP for skipping the steps they had already completed, especially in tasks where they were proficient and had clearer goals: *"The milestones are structured...it directs me from step one to three when I already completed step two. It works when you have a clear goal and you know there's one way to go...(D04)."* They suggested even more granular detection of actions to assist with option-based scenarios (e.g., opacity and blend). Experts also saw the potential for this feature in apps like Canva, Cadence, and SolidWorks.

Novices appreciated the clickable navigation of milestone steps for unfamiliar tasks and the ability to skip those steps when proficient. They found this adaptability helpful for validating the successful completion of their current step: *"I like it* [GuidedCopilotADP] *...it breaks down steps and automatically goes to the next step when I finish...I don't have to check if I completed that step. If I want to explore the other functions I can go back (D06)."* Both novice and expert users noted that GuidedCopilotADP would be increasingly

useful as novices gained proficiency with the application, for both complex and artistic tasks.

7.4.2 Users' reactions when using GuidedCopilotVisual. All participants found the task and state-aware preview clips easy to use and helpful for forming a mental model of the user interface: *"I know what is going on in my mind...is clear with these videos [preview clips] after reading the steps, it directly shows me which button to click, which parameter to modify (D04)."* Unlike traditional video demonstrations, participants liked that these clips aligned with the application's current state and task progress, saving them time and helping locate UI elements more quickly: *"I like preview clips based on my task progress because they save my time...I don't need to go to [the] beginning every time...making it more automatic was helpful (D09)."* Participants were eager to see these clips for tasks in other feature-rich applications like SolidWorks, Canva, and Lightroom.

8 Discussion

8.1 Key Takeaways

In this paper, we provide compelling evidence that users value maintaining control when interacting with software copilots, particularly for complex tasks. Semi-automation, as implemented in GuidedCopilot, was effective for repetitive and trivial steps, enhancing perceptions of productivity and user satisfaction. While AutoCopilot's full automation appealed to a few of the technical participants, it was often misaligned with tasks and increased users' debugging efforts. In contrast, GuidedCopilot's step-by-step visual guidance empowered users to learn software skills, validate automated processes, and regain control when the automation fell short, especially for unfamiliar tasks. While our study focused on GuidedCopilot's implementation in two applications, our implementation using scriptable frameworks is flexible and can be generalized to other feature-rich software with detailed documentation and visual examples. Furthermore, our follow-up design exploration highlights opportunities to enhance semi-automatic copilots with task- and state-aware features that dynamically adapt to user needs.

By challenging the trend toward fully automated copilots, our findings underscore the importance of end-user customization, guidance, and control for effective user-copilot collaboration. Our experimental and qualitative insights contribute to a deeper understanding of how to design copilots for complex workflows within feature-rich applications that accommodate different experience levels. Consistent with historical lessons in automation research [74], we argue that AI and feature-rich interfaces will coexist and evolve together as complementary tools rather than replacements. HCI and AI researchers, interface designers, developers, and others working on LLM-powered assistants leverage our framework and key factors (Figure 8, Section 8.2) to achieve more effective human-AI interaction.

8.2 Key factors to consider for levels of automation

Based on our two study findings, we identify three key factors for determining the appropriate level of automation and user control in software copilots. We propose a three-dimensional framework (Figure 8) with axes for Semi-Automation vs. Full Automation, Adaptive Step-by-Step Visual Guidance vs. Non-Adaptive Step-by-Step Visual Guidance, and three influencing factors: (i) Familiarity with the application, (ii) Task Type, (iii) User Intent in learning vs. performing tasks.

8.2.1 Familiarity with the application. ■ Our findings highlight that users benefit more from semi-automatic copilots with stepwise visual guidance, particularly when onboarding with unfamiliar software. While full automation may streamline trivial tasks, it often lacks explainability and does not help users develop a mental model of software features or learn how to manipulate automation. Future research should explore how software copilots shape user mental models, especially for novices. Though some studies have explored mental models with LLM assistants in help seeking [42, 79], more research focused on the new generation of in-application copilots is needed to foster more intuitive and effective interactions. Future work can also explore long-term deployments to investigate the impact of interaction duration (short-term vs. long-term) on user expectations and experience with automation in software copilots.

As users gain experience with a software, adaptive semi-automatic support has the potential to enhance software utility and productivity. Building on prior research on personalized and adaptive help systems [7, 21, 35, 54, 65], future copilots should be designed to offer adaptive, semi-automated approaches that balance time savings with user control, catering to varying expertise levels.

8.2.2 Task Type. ▲ Our study shows that higher levels of automation work best for simple, repetitive tasks aligning with prior works [74], but semi-automation is preferred for complex decision-making, debugging, or validation tasks (e.g., intricate data analysis or exploratory design). Copilots that offer adaptive visual guidance and previews can be particularly helpful for creative software tasks. Such copilot assistance can prevent users from becoming passive monitors — a risk often transformed with higher levels of automation [74]. Future copilots should leverage both task categories and software, moving beyond the general-purpose designs (e.g., [24, 77]) to better cater to diverse user needs. The HCI and AI communities should join forces to focus on creating copilots tailored to users' specific tasks and needs.

8.2.3 User intent in learning vs. performing tasks. ● Although we did not formally investigate gender, age, or CS background as a factor, we observed that male users with CS backgrounds preferred automation for time efficiency. However, fully automated approaches risk overreliance and deskilling [42, 74], particularly for novices. Users can face a diminished ability to perform tasks independently or intervene when automation fails. Most novices preferred our semi-automated copilot that promoted learning of software steps. This aligns with established help-seeking behaviors, such as "learning by demonstration", using visual mediums [41, 49, 64], step-by-step guidance [16, 41], adaptive help [7, 21, 35, 54, 65], in-context help systems [11, 16, 32, 34, 47], and visual demonstrations [16, 27, 28, 40, 48, 61], all proven effective for enhancing user learning and retention. Future semi-automatic copilots could go a step further by integrating Explainable AI (XAI) [51, 56] to make AI decision-making processes more understandable. Semi-automatic copilots could not only assist in task execution (as seen in GuidedCopilot) but also help users learn from the AI's reasoning [41, 45], ultimately

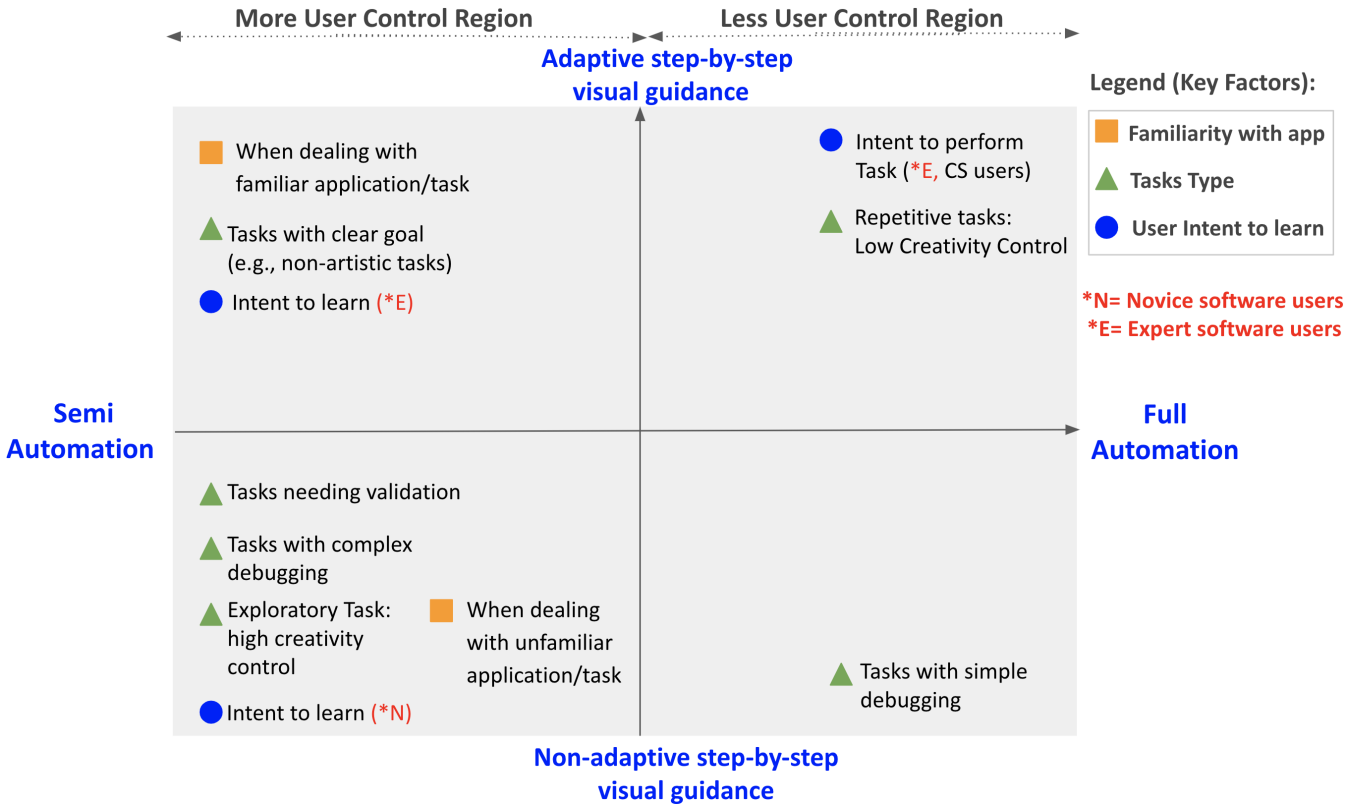

Figure 8: Dimensional framework describing key factors to consider when determining levels of automation and step-by-step guidance in copilots: (a) Familiarity with the application: When designing copilots for unfamiliar applications, a guided, semi-automatic approach with visual references can help users onboard, while more adaptive support for experts balances time savings with user control, catering to different expertise levels. (b) Tasks Type: Higher levels of automation are best suited for straightforward, repetitive, or simpler visual tasks. For more nuanced tasks involving complex decision-making, debugging, user dependencies, or creative input, semi-automatic copilots with step-by-step visuals or previews are more beneficial, allowing for greater user control. More adaptive guidance is useful for tasks with clear goals and intentions, typically non-artistic and non-exploratory, within feature-rich applications. (c) User Intent to Learn: For users with a clear intent to learn while using feature-rich software, future copilots should adopt a semi-automatic approach with step-by-step guidance—more adaptive for experts and less adaptive for novices—balancing automation with user learning. For those focused on time efficiency, such as expert software users and CS male users, a higher level of automation would be more suitable.

fostering skill development and reducing overreliance. Such copilots for complex software tasks should balance automation with learning, incorporating mixed-medium, in-context help, as seen with GuidedCopilot. Our paper provides empirical evidence that copilots should function truly as "co-pilots" [74], supporting users without diminishing their role or skills, and draw from established help-seeking strategies [14, 27, 28, 40] to maintain a balance between automation and user learning.

8.3 Prioritizing User Control When Designing Copilots

As seen in studies of LLM-powered tools for code generation [39, 74], our findings show that users preferred GuidedCopilot as its balanced automation with instructional guidance allowed users to choose their level of control. Our follow-up study further suggests that while automating certain tasks can enhance efficiency (See Section 8.2.2), users need the ability to intervene, make decisions, understand the process, and retain control. This is important in particular when dealing with complex or creative tasks where automation has higher chances of failing. This approach aligns with the concept of mixed-initiative interactions [5, 37], where users shift between manual and automated modes based on their needs and task complexity.

Although we did not focus on creativity needs, previous research suggests that visual artists and graphic designers also value control over their creative processes when using visual feature-rich software [38, 46]. Future research should explore how copilots can better support creative workflows by offering adaptable automation that enhances, rather than undermines, their creative autonomy. By allowing users to select their preferred level of automation, future copilots can better support diverse user and task needs, fostering both effective collaboration and continuous learning. Future

research should also explore multi-agent strategies to optimize when and how automation should intervene, ensuring that copilots empower users while maximizing the benefits of automation.

9 Limitations

Our study examined two distinct design paradigms in software copilots (semi-automation vs. full automation) and how they impact task completion and user perceptions. Although our findings emphasize the importance of user control and accommodating diverse experience levels, they are limited by the applications used (Figma, Sheets) and the evolving capabilities of LLMs. Although our prototypes were effective for initial insights, advanced vision-based LLMs like SORA could provide more real-time data and deeper analysis. We did not directly assess the impact of specific features (e.g., automation, step-by-step guidance), as this would compromise ecological validity. Our findings highlight the issue of full automation without transparency or guidance, suggesting that automation alone is insufficient. Future research should explore the role of transparency in both automation paradigms and quantitatively assess the independent effects of these features, along with individual differences (e.g., age, gender, expertise), using larger, more diverse samples.

10 Conclusion

We investigated two design paradigms for automation in software copilots: AutoCopilot, a fully automated copilot, and GuidedCopilot that combines automation of trivial steps with step-by-step visual guidance. Our results show that while full automation may appeal to a few users, most prefer to maintain control over complex tasks, favoring semi-automation for repetitive and trivial steps. As copilots advance toward full automation, our findings underscore the importance of offering greater user control and accommodating diverse experience levels to maximize effectiveness. These insights provide valuable guidance for HCI and AI researchers, designers, and developers in balancing automation with control and user autonomy, fostering more effective human-AI collaboration in feature-rich software.

Acknowledgments

We thank the Natural Sciences and Engineering Research Council of Canada (NSERC) for funding this research.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Adobe. 2023. Adobe unveils Firefly, a family of new creative generative ai. https://news.adobe.com/news/news-details/2023/Adobe-Unveils-Firefly-a-Family-of-new-Creative-Generative-AI/default.aspx

[3] Open AI. 2022. Introducing chatgpt. https://openai.com/blog/chatgpt

[4] Naser Al Madi. 2023. How Readable is Model-generated Code? Examining Readability and Visual Inspection of GitHub Copilot. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (Rochester, MI, USA) *(ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 205, 5 pages. https://doi.org/10.1145/3551349.3560438

[5] J.E. Allen, C.I. Guinn, and E. Horvtz. 1999. Mixed-initiative interaction. *IEEE Intelligent Systems and their Applications* 14, 5 (1999), 14–23. https://doi.org/10.1109/5254.796083

[6] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N. Bennett, Kori Inkpen, Jaime Teevan, Ruth Kikin-Gil, and Eric Horvitz. 2019. Guidelines for Human-AI Interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300233

[7] Mirjam Augstein, Eelco Herder, and Wolfgang Wörndl. 2023. *Personalized human-computer interaction*. Walter de Gruyter GmbH & Co KG.

[8] Lisanne Bainbridge. 1983. Ironies of automation. In *Analysis, design and evaluation of man–machine systems*. Elsevier, 129–135.

[9] Gagan Bansal, Jennifer Wortman Vaughan, Saleema Amershi, Eric Horvitz, Adam Fourney, Hussein Mozannar, Victor Dibia, and Daniel S. Weld. 2024. *Challenges in Human-Agent Communication*. Technical Report MSR-TR-2024-53. Microsoft. https://www.microsoft.com/en-us/research/publication/human-agent-interaction-challenges/

[10] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proc. ACM Program. Lang.* 7, OOPSLA1, Article 78 (apr 2023), 27 pages. https://doi.org/10.1145/3586030

[11] Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R. Klemmer. 2010. Example-Centric Programming: Integrating Web Search into the Development Environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 513–522. https://doi.org/10.1145/1753326.1753402

[12] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[13] Donald T Campbell. 1960. Blind variation and selective retentions in creative thought as in other knowledge processes. *Psychological review* 67, 6 (1960), 380.

[14] John M Carroll and Caroline Carrithers. 1984. Training wheels in a user interface. *Commun. ACM* 27, 8 (1984), 800–806.

[15] John M. Carroll and Mary Beth Rosson. 1987. *Paradox of the Active User*. MIT Press, Cambridge, MA, USA, 80–111.

[16] Parmit K. Chilana, Amy J. Ko, and Jacob O. Wobbrock. 2012. LemonAid: Selection-Based Crowdsourced Contextual Help for Web Applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. Association for Computing Machinery, New York, NY, USA, 1549–1558. https://doi.org/10.1145/2207676.2208620

[17] Microsoft Power Platform Community. 2024. Forums | Microsoft Power Platform Community — powerusers.microsoft.com. https://powerusers.microsoft.com/t5/General-Power-Automate/What-am-I-doing-wrong-Is-Co-Pilot-just-useless-First-time-user/td-p/2701867. [Accessed 12-09-2024].

[18] Juliet M Corbin and Anselm Strauss. 1990. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative sociology* 13, 1 (1990), 3–21.

[19] Justin Cranshaw, Emad Elwany, Todd Newman, Rafal Kocielnik, Bowen Yu, Sandeep Soni, Jaime Teevan, and Andrés Monroy-Hernández. 2017. Calendar.help: Designing a Workflow-Based Scheduling Agent with Humans in the Loop. *ACM CHI Conference on Human Factors in Computing Systems* (January 2017). https://www.microsoft.com/en-us/research/publication/calendar-help-designing-workflow-based-scheduling-agent-humans-loop/

[20] Nils Dahlbäck, Arne Jönsson, and Lars Ahrenberg. 1993. Wizard of Oz studies: why and how. In *Proceedings of the 1st International Conference on Intelligent User Interfaces* (Orlando, Florida, USA) *(IUI '93)*. Association for Computing Machinery, New York, NY, USA, 193–200. https://doi.org/10.1145/169891.169968

[21] Sylvain Delisle and Bernard Moulin. 2002. User interfaces and help systems: from helplessness to intelligent assistance. *Artif. Intell. Rev.* 18, 2 (oct 2002), 117–157. https://doi.org/10.1023/A:1015179704819

[22] Françoise Détienne. 2001. *Software design–cognitive aspect*. Springer Science & Business Media.

[23] Mica R Endsley. 2017. From here to autonomy: lessons learned from human–automation research. *Human factors* 59, 1 (2017), 5–27.

[24] Figma. 2024. Figma AI: Your Creativity, unblocked with Figma AI — figma.com. https://www.figma.com/ai/. [Accessed 11-09-2024].

[25] Adobe Firefly. 2024. https://firefly.adobe.com/

[26] Microsoft Forum. 2024. Redirecting — answers.microsoft.com. https://answers.microsoft.com/en-us/msoffice/forum/all/title-a-nightmare-experience-with-microsoft/a01f65b8-62bb-43aa-bda5-7ae48fac9095. [Accessed 11-09-2024].

[27] Adam Fourney, Ben Lafreniere, Richard Mann, and Michael Terry. 2012. " Then click ok!" extracting references to interface elements in online documentation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.

35–38.
[28] C. Ailie Fraser, Tricia J. Ngoon, Mira Dontcheva, and Scott Klemmer. 2019. RePlay: Contextually Presenting Learning Videos Across Software Applications. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300527
[29] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. 1987. The Vocabulary Problem in Human-System Communication. *Commun. ACM* 30, 11 (nov 1987), 964–971. https://doi.org/10.1145/32206.32212
[30] L Gao, S Biderman, S Black, L Golding, T Hoppe, C Foster, J Phang, H He, A Thite, N Nabeshima, et al. 2020. The pile: An 800GB dataset of diverse text for language modeling. arXiv. *arXiv preprint arXiv:2101.00027* (2020).
[31] Alyssa Glass, Deborah L. McGuinness, and Michael Wolverton. 2008. Toward Establishing Trust in Adaptive Agents. In *Proceedings of the 13th International Conference on Intelligent User Interfaces* (Gran Canaria, Spain) *(IUI '08)*. Association for Computing Machinery, New York, NY, USA, 227–236. https://doi.org/10.1145/1378773.1378804
[32] Tovi Grossman and George Fitzmaurice. 2010. ToolClips: An Investigation of Contextual Video Assistance for Functionality Understanding. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 1515–1524. https://doi.org/10.1145/1753326.1753552
[33] Tovi Grossman, George Fitzmaurice, and Ramtin Attar. 2009. A Survey of Software Learnability: Metrics, Methodologies and Guidelines. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Boston, MA, USA) *(CHI '09)*. Association for Computing Machinery, New York, NY, USA, 649–658. https://doi.org/10.1145/1518701.1518803
[34] Björn Hartmann, Daniel MacDougall, Joel Brandt, and Scott R. Klemmer. 2010. What Would Other Programmers Do: Suggesting Solutions to Error Messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 1019–1028. https://doi.org/10.1145/1753326.1753478
[35] H.W. Hastie, M. Johnston, and P. Ehlen. 2003. Context-sensitive help for multimodal dialogue. In *Proceedings. Fourth IEEE International Conference on Multimodal Interfaces*. Institute of Electrical and Electronics Engineers, 93. https://doi.org/10.1109/ICMI.2002.1166975 4th IEEE International Conference on Multimodal Interfaces, ICMI 2002 ; Conference date: 16-10-2002.
[36] Jeffrey Heer. 2019. Agency plus automation: Designing artificial intelligence into interactive systems. *Proceedings of the National Academy of Sciences* 116, 6 (2019), 1844–1850.
[37] Eric J. Horvitz. 2007. Reflections on Challenges and Promises of Mixed-Initiative Interaction. *AI Magazine* 28, 2 (Jun. 2007), 3. https://doi.org/10.1609/aimag.v28i2.2036
[38] Amir Jahanlou and Parmit K Chilana. 2024. How Example-Based Authoring of Motion Graphics Impacts Creative Expression: Differences in Perceptions of Professional and Casual Motion Designers. In *Proceedings of the 16th Conference on Creativity & Cognition* (Chicago, IL, USA) *(C&C '24)*. Association for Computing Machinery, New York, NY, USA, 347–357. https://doi.org/10.1145/3635636.3656197
[39] Majeed Kazemitabaar, Jack Williams, Ian Drosos, Tovi Grossman, Austin Henley, Carina Negreanu, and Advait Sarkar. 2024. Improving Steering and Verification in AI-Assisted Data Analysis with Interactive Task Decomposition. *arXiv preprint arXiv:2407.02651* (2024).
[40] Caitlin Kelleher and Randy Pausch. 2005. Stencils-based tutorials: design and evaluation. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 541–550.
[41] Anjali Khurana, Parsa Alamzadeh, and Parmit K. Chilana. 2021. ChatrEx: Designing Explainable Chatbot Interfaces for Enhancing Usefulness, Transparency, and Trust. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 1–11. https://doi.org/10.1109/VL/HCC51201.2021.9576440
[42] Anjali Khurana, Hariharan Subramonyam, and Parmit K Chilana. 2024. Why and When LLM-Based Assistants Can Go Wrong: Investigating the Effectiveness of Prompt-Based Interactions for Software Help-Seeking. In *Proceedings of the 29th International Conference on Intelligent User Interfaces* (Greenville, SC, USA) *(IUI '24)*. Association for Computing Machinery, New York, NY, USA, 288–303. https://doi.org/10.1145/3640543.3645200
[43] Kimia Kiani, George Cui, Andrea Bunt, Joanna McGrenere, and Parmit K. Chilana. 2019. Beyond "One-Size-Fits-All": Understanding the Diversity in How Software Newcomers Discover and Make Use of Help Resources. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3290605.3300570
[44] Juho Kim, Phu Tran Nguyen, Sarah Weir, Philip J. Guo, Robert C. Miller, and Krzysztof Z. Gajos. 2014. Crowdsourcing Step-by-Step Information Extraction to Enhance Existing How-to Videos. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 4017–4026. https://doi.org/10.1145/2556288.2556986
[45] Sunnie S. Y. Kim, Elizabeth Anne Watkins, Olga Russakovsky, Ruth Fong, and Andrés Monroy-Hernández. 2023. "Help Me Help the AI": Understanding How Explainability Can Support Human-AI Interaction. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 250, 17 pages. https://doi.org/10.1145/3544548.3581001
[46] Hyung-Kwon Ko, Gwanmo Park, Hyeon Jeon, Jaemin Jo, Juho Kim, and Jinwook Seo. 2023. Large-scale Text-to-Image Generation Models for Visual Artists' Creative Works. In *Proceedings of the 28th International Conference on Intelligent User Interfaces* (Sydney, NSW, Australia) *(IUI '23)*. Association for Computing Machinery, New York, NY, USA, 919–933. https://doi.org/10.1145/3581641.3584078
[47] Benjamin Lafreniere, Parmit K. Chilana, Adam Fourney, and Michael A. Terry. 2015. These Aren't the Commands You're Looking For: Addressing False Feedforward in Feature-Rich Software. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology* (Charlotte, NC, USA) *(UIST '15)*. Association for Computing Machinery, New York, NY, USA, 619–628. https://doi.org/10.1145/2807442.2807482
[48] Benjamin Lafreniere, Tovi Grossman, and George Fitzmaurice. 2013. Community Enhanced Tutorials: Improving Tutorials with Multiple Demonstrations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. Association for Computing Machinery, New York, NY, USA, 1779–1788. https://doi.org/10.1145/2470654.2466235
[49] Toby Jia-Jun Li, Jingya Chen, Haijun Xia, Tom M. Mitchell, and Brad A. Myers. 2020. Multi-Modal Repairs of Conversational Breakdowns in Task-Oriented Dialogs. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. Association for Computing Machinery, New York, NY, USA, 1094–1107. https://doi.org/10.1145/3379337.3415820
[50] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2024. A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 52, 13 pages. https://doi.org/10.1145/3597503.3608128
[51] Q. Vera Liao, Daniel Gruen, and Sarah Miller. 2020. Questioning the AI: Informing Design Practices for Explainable AI User Experiences. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3313831.3376590
[52] Q Vera Liao and Jennifer Wortman Vaughan. 2023. Ai transparency in the age of llms: A human-centered research roadmap. *arXiv preprint arXiv:2306.01941* (2023), 5368–5393.
[53] Justin Matejka, Tovi Grossman, and George Fitzmaurice. 2011. IP-QAT: in-product questions, answers, & tips. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. 175–184.
[54] Justin Matejka, Wei Li, Tovi Grossman, and George Fitzmaurice. 2009. CommunityCommands: command recommendations for software applications. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology* (Victoria, BC, Canada) *(UIST '09)*. Association for Computing Machinery, New York, NY, USA, 193–202. https://doi.org/10.1145/1622176.1622214
[55] Robinson Meyer. 2015. Even Early Focus Groups Hated Clippy. https://www.theatlantic.com/technology/archive/2015/06/clippy-the-microsoft-office-assistant-is-the-patriarchys-fault/396653/ [Accessed: 10-May-2020].
[56] Tim Miller. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence* 267 (2019), 1–38.
[57] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2024. Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 142, 16 pages. https://doi.org/10.1145/3613904.3641936
[58] Brad A Myers, David A Weitzman, Amy J Ko, and Duen H Chau. 2006. Answering why and why not questions in user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*. 397–406.
[59] Don Norman. 2013. *The design of everyday things: Revised and expanded edition*. Basic books.
[60] David G. Novick, Oscar D. Andrade, and Nathaniel Bean. 2009. The Micro-Structure of Use of Help. In *Proceedings of the 27th ACM International Conference on Design of Communication* (Bloomington, Indiana, USA) *(SIGDOC '09)*. Association for Computing Machinery, New York, NY, USA, 97–104. https://doi.org/10.1145/1621995.1622014
[61] David G. Novick, Oscar D. Andrade, Nathaniel Bean, and Edith Elizalde. 2008. Help-Based Tutorials. In *Proceedings of the 26th Annual ACM International Conference on Design of Communication* (Lisbon, Portugal) *(SIGDOC '08)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/1456536.1456538
[62] David G. Novick and Karen Ward. 2006. Why Don't People Read the Manual?. In *Proceedings of the 24th Annual ACM International Conference on Design of Communication* (Myrtle Beach, SC, USA) *(SIGDOC '06)*. Association for Computing

Machinery, New York, NY, USA, 11–18. https://doi.org/10.1145/1166324.1166329
[63] OpenAI. 2024. Embeddings. https://platform.openai.com/docs/guides/embeddings. [Accessed 10-09-2024].
[64] Susan Palmiter, Jay Elkerton, and Patricia Baggett. 1991. Animated demonstrations vs written instructions for learning procedural tasks: a preliminary investigation. *International Journal of Man-Machine Studies* 34, 5 (1991), 687–701.
[65] S. Pangoli and F. Paternó. 1995. Automatic generation of task-oriented help. In *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology* (Pittsburgh, Pennsylvania, USA) *(UIST '95)*. Association for Computing Machinery, New York, NY, USA, 181–187. https://doi.org/10.1145/215585.215971
[66] Martin Porcheron, Joel E. Fischer, and Stuart Reeves. 2021. Pulling Back the Curtain on the Wizards of Oz. *Proc. ACM Hum.-Comput. Interact.* 4, CSCW3, Article 243 (jan 2021), 22 pages. https://doi.org/10.1145/3432942
[67] Microsoft Graph RAG. 2024. GitHub - microsoft/graphrag: A modular graph-based Retrieval-Augmented Generation (RAG) system — github.com. https://github.com/microsoft/graphrag. [Accessed 11-09-2024].
[68] Ashwin Ramachandran and R Michael Young. 2005. Providing intelligent help across applications in dynamic user and environment contexts. In *Proceedings of the 10th international conference on Intelligent user interfaces*. 269–271.
[69] Michael L Raulin and Anthony M Graziano. 2019. Quasi-experiments and correlational studies. In *Companion Encyclopedia of Psychology*. Routledge, 1122–1141.
[70] Marc Rettig. 1991. Nobody Reads Documentation. *Commun. ACM* 34, 7 (jul 1991), 19–24. https://doi.org/10.1145/105783.105788
[71] John Rieman. 1996. A Field Study of Exploratory Learning Strategies. *ACM Trans. Comput.-Hum. Interact.* 3, 3 (sep 1996), 189–218. https://doi.org/10.1145/234526.234527
[72] Advait Sarkar, Andrew D Gordon, Carina Negreanu, Christian Poelitz, Sruti Srinivasa Ragavan, and Ben Zorn. 2022. What is it like to program with artificial intelligence? *arXiv preprint arXiv:2208.06213* (2022).
[73] Elastic Hybrid Search. 2024. Hybrid Search — Search Labs — elastic.co. https://www.elastic.co/search-labs/tutorials/search-tutorial/vector-search/hybrid-search. [Accessed 11-09-2024].
[74] Abigail Sellen and Eric Horvitz. 2024. The rise of the AI Co-Pilot: Lessons for design from aviation and beyond. *Commun. ACM* 67, 7 (2024), 18–23.
[75] Ben Shneiderman. 2020. Human-centered artificial intelligence: Reliable, safe & trustworthy. *International Journal of Human–Computer Interaction* 36, 6 (2020), 495–504.
[76] Ben Shneiderman and Pattie Maes. 1997. Direct manipulation vs. interface agents. *interactions* 4, 6 (1997), 42–61.
[77] Jared Spataro. 2023. Introducing Microsoft 365 copilot – your copilot for work. https://blogs.microsoft.com/blog/2023/03/16/introducing-microsoft-365-copilot-your-copilot-for-work/
[78] Ashley Stewart. 2024. A CIO canceled a Microsoft AI deal. The reason should worry the entire tech industry. — businessinsider.com. https://www.businessinsider.com/pharma-cio-cancelled-microsoft-copilot-ai-tool-2024-7. [Accessed 11-09-2024].
[79] Hari Subramonyam, Roy Pea, Christopher Pondoc, Maneesh Agrawala, and Colleen Seifert. 2024. Bridging the Gulf of Envisioning: Cognitive Challenges in Prompt Based Interactions with LLMs. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 1039, 19 pages. https://doi.org/10.1145/3613904.3642754
[80] Yuzhang Tian, Jianbo Zhao, Haoyu Dong, Junyu Xiong, Shiyu Xia, Mengyu Zhou, Yun Lin, José Cambronero, Yeye He, Shi Han, et al. 2024. SpreadsheetLLM: Encoding Spreadsheets for Large Language Models. *arXiv preprint arXiv:2407.09025* (2024).
[81] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, Article 332, 7 pages. https://doi.org/10.1145/3491101.3519665
[82] Laton Vermette, Shruti Dembla, April Y. Wang, Joanna McGrenere, and Parmit K. Chilana. 2017. Social CheatSheet: An Interactive Community-Curated Information Overlay for Web Applications. *Proc. ACM Hum.-Comput. Interact.* 1, CSCW, Article 102 (dec 2017), 19 pages. https://doi.org/10.1145/3134737
[83] Xu Wang, Benjamin Lafreniere, and Tovi Grossman. 2018. Leveraging community-generated videos and command logs to classify and recommend software workflows. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–13.
[84] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. 2023. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382* (2023).
[85] Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-Pack: Packaged Resources To Advance General Chinese Embedding. arXiv:2309.07597 [cs.CL]
[86] Frank F. Xu, Bogdan Vasilescu, and Graham Neubig. 2022. In-IDE Code Generation from Natural Language: Promise and Challenges. *ACM Trans. Softw. Eng. Methodol.* 31, 2, Article 29 (mar 2022), 47 pages. https://doi.org/10.1145/3487569
[87] Litao Yan, Alyssa Hwang, Zhiyuan Wu, and Andrew Head. 2024. Ivie: Lightweight Anchored Explanations of Just-Generated Code. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 140, 15 pages. https://doi.org/10.1145/3613904.3642239
[88] Tom Yeh, Tsung-Hsiang Chang, Bo Xie, Greg Walsh, Ivan Watkins, Krist Wongsuphasawat, Man Huang, Larry S Davis, and Benjamin B Bederson. 2011. Creating contextual help for GUIs using screenshots. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. 145–154.
[89] J.D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 437, 21 pages. https://doi.org/10.1145/3544548.3581388

A Additional details on Data Analysis: Controlled Experiment and Follow-up Interviews

We describe the actual software tasks users performed with assistance from copilot interventions, along with the ground truth for each task, to evaluate task completion and accuracy.

A.1 Ground truth sample for spreadsheet tasks in Sheets

- **Task 1:** Participants were asked to use copilot assistance to find the top 5 products in terms of sales and visualize the sales distribution of these products across different 'regions' in a bar chart, making the bar chart pink gradient.
  Ground Truth: The ground truth for successful task completion is defined by how many of these task steps users completed (1) using the sort function on the sheet with the data ranging from A1:C31 (2) creating a bar chart of the data ranging from A2:C6 (3) using the customize tab in the chart editor to modify the Fill color and Line color of the series bar to pink in the bar chart. The ground truth of the task accuracy reflects how accurately they applied these outlined sequence of steps and software functions to complete the given task.
- **Task 2:** Participants were asked to use copilot assistance to color-code quantity values greater than 40 in the Inventory sheet and to find and insert the corresponding product names from the 'Products' sheet into column D of the 'Inventory' sheet. They were asked to match the product IDs listed in column A of the 'Inventory' sheet with those in the 'Products' sheet under the 'Products' column.
  Ground Truth: The ground truth for successful task completion is defined by how many of these task steps users completed (1) using the Conditional Formatting on the selected data range C2:C13 (2) applying the "Greater than" format rule to set the condition for values greater than 40 (2) using the VLOOKUP formula (e.g., =VLOOKUP(A2, Products!A:B, 2, FALSE)) in column D of the 'Inventory' sheet, and (3) copying the formula down column D to apply it for all product IDs. The ground truth of the task accuracy reflects how accurately they applied these outlined sequence of steps and software functions to complete the given task.

A.2 Ground truth sample for UI design tasks in Figma

- **Task 1:** Participants were asked to use copilot assistance to create a webpage that includes a login page (username, password, button, interaction to the product page) and product page (description of the product, and image). (Participants were provided with the reference webpage to create)
  Ground Truth: The ground truth for successful task completion is defined by how many of these task steps users completed (1) using the Frame (F) tool to create two frames for login and product page function on the sheet (2) using the Rectangle tool to add input fields for username and password and create Submit button (3) using the Text Tool (T) to add a label above each field (e.g., "Username" , "Password", "Submit") and adjusting the spacing of the added rectangles (4) using Image tool to include prodcut images (already provided to the participants) (5) adding interactions to the button using the Prototype tab and dragging prototyping handle from login button to product page. The ground truth of the task accuracy reflects how accurately they applied these outlined sequence of steps and software functions to complete the given task.
- **Task 2:** Participants were asked to use copilot assistance to design an interactive tutorial that provides a comprehensive three-step interactive project timeline of your product development cycle and looks like the reference project timeline. (Participants were provided with the reference project timeline)
  Ground Truth: (1) using the Frame (F) tool to create a frame (2) using the Eclipse tool to draw an oval shape for the timeline, Rectangle tool for creating buttons and adjusting the size, spacing and shape (3) using the Fill option to color the shapes and Text Tool (T) to add a label above each field (e.g., "Step 1: Brainstorming" , "Step 2: Prototyping", "Next") (4) adding interactions to the button using the Prototype tab and dragging prototyping handle from Next button to second page The ground truth of the task accuracy reflects how accurately they applied these outlined sequence of steps and software functions to complete the given task.