To Search or To Gen? Design Dimensions Integrating Web Search and Generative AI in Programmers' Information-Seeking Process

Ryan Yen
University of Waterloo
Waterloo, Ontario, Canada
MIT CSAIL
Cambridge, Massachusetts, USA
ryanyen2@mit.edu

Yimeng Xie
University of Waterloo
Waterloo, Ontario, Canada
y329xie@uwaterloo.ca

Nicole Sultanum
Tableau Research
Seattle, Washington, USA
nsultanum@tableau.com

Jian Zhao
University of Waterloo
Waterloo, Ontario, Canada
jianzhao@uwaterloo.ca

Abstract

Programmers now use both generative AI (GenAI) and traditional web search for information-seeking, yet how these tools are used individually or in combination remains unclear. To answer this, we conducted a multi-phase investigation, including retrospective interviews to identify foraging behaviours and challenges and an observational study with a technology probe to analyze how contextual information flows across tools. Our findings reveal that effective information-seeking requires adaptable strategies and varying levels of contextual detail. Building on these insights, we propose five design dimensions for developing tools that integrate web search, GenAI, and code editors. We further demonstrated the generative power of these design dimensions with a proof-of-concept prototype, validated through a user study, offering actionable design implications for enhancing integrated information-seeking workflows across web search and GenAI in programming.

CCS Concepts

• **Information systems** → Web searching and information discovery; • **Human-centered computing** → *Interaction paradigms*; **Empirical studies in HCI**.

Keywords

Information-Foraging, Information Seeking, Human-AI, Code Generation

ACM Reference Format:
Ryan Yen, Yimeng Xie, Nicole Sultanum, and Jian Zhao. 2025. To Search or To Gen? Design Dimensions Integrating Web Search and Generative AI in Programmers' Information-Seeking Process. In . ACM, New York, NY, USA, 23 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Programmers invest significant time in seeking and synthesizing external information to address programming tasks [57]. Traditionally, they have relied on web search to gather the necessary information [6, 35]. Recently, advances in generative AI have introduced an alternative approach. Programmers can type in more complex queries in natural language and generate tailored solutions [55].

Despite the convenience of GenAI-based information seeking, issues such as hallucinations, misinformation, and lack of transparency pose significant challenges that prevent programmers from fully relying on these tools [33, 37, 39, 60]. To address these issues, techniques such as retrieval-augmented generation (RAG) have been proposed to contextualize GenAI with web search results, enhancing factuality and reliability [42, 67]. At first glance, this approach appears to "solve" the problem by enabling programmers to find answers more quickly, with cited sources available for fact-checking. However, this approach overlooks a critical aspect of the *information-foraging process* [51, 64]. This process, in which programmers manually search, collect, and organize information from diverse sources, is crucial for building domain knowledge, developing problem-solving mental models, and evaluating the appropriateness of results [26, 36, 56, 64]. Thus, before automating these processes or fully integrating such features, it is crucial to first understand how programmers currently use web search and GenAI individually or in combination, and how these tools influence their information-foraging behaviours and problem-solving workflows.

We explore how programmers seek information using web search and generative AI, focusing on how they transfer *context* between these tools. By context, we mean a dynamically negotiated, relational property that programmers actively construct and sustain within each specific activity, rather than a static body of information that merely surrounds their actions [13]. Rather than comparing the two tools in isolation, we investigate how programmers weave them together, capitalizing on each one's strengths or using one as a fallback when the other falls short. Our research (see Figure 1) begins with a retrospective interview with $N = 8$ programmers to understand how programmers forage information using these tools—the extent that they use them *complementarily* or *alternately*—and to identify the challenges that arise. We present interview findings organized around three major decision-making stages that detail how programmers carry information within or across tools.

A major challenge we identified is the difficulty of *translating* information extracted from web search or GenAI into formats suitable for the other tool, largely due to their differing interaction paradigms and the varying levels of formality with which information is encoded in each medium. To explore the design requirements

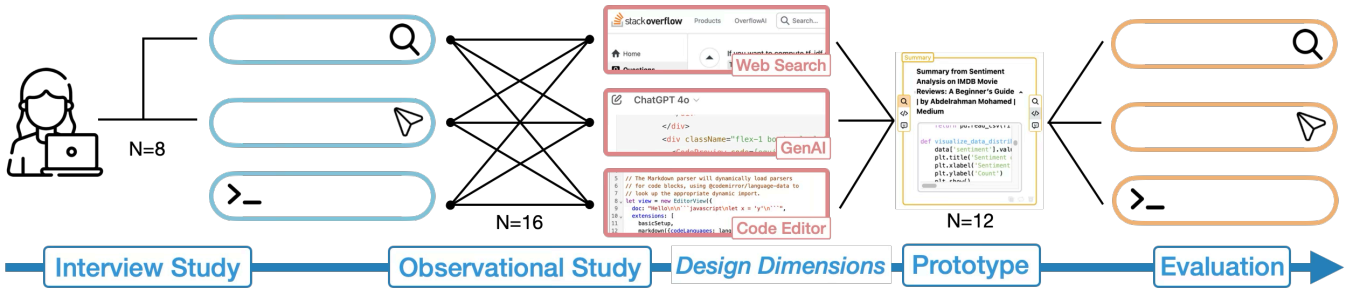


Figure 1: Overview of the study procedure. The process begins with an interview study to explore programmers' practices of using both web search and GenAI. Then, an observational study identified information flows, information types, and translation patterns. Finally, we synthesize these insights into design dimensions and validate them through a designed prototype.

for transferring information across tools, we conducted an observational study with $N = 16$ programmers using a technology probe to gain deeper insights into how information flows between tools and how information is transmitted and translated in situ. Our findings revealed diverse patterns of information flow and demonstrated that the transferred information extends beyond simple search results or generated outputs. Programmers frequently transferred high-level concepts, interaction histories, rationales, and verbatim snippets of text and code, employing various strategies to translate this information for subsequent rounds of problem-solving.

Building on these insights, we propose five dimensions for future tool design, enhancing information flowing between web search and GenAI. These dimensions include: (1) Interoperability, reducing resistance between tools through integration; (2) Adaptability, supporting dynamic navigation across linear, branching, and parallel flows of information; (3) Contextual Awareness, externalizing and preserving task context to facilitate reuse and cross-tool transitions; (4) Translatability, enabling context-centric translation with a balance of automated and manual processes; and (5) Traceability, maintaining clear provenance to support validation and workflow continuity. To explore the generative power of these dimensions, we developed a prototype and validated it through a subsequent user study with $N = 12$ participants, reporting changes in their workflows and key insights for future tool design. We conclude with design implications for developing advanced tools that integrate web search and GenAI, aiming to better support programmers in seeking information to solve complex programming problems.

2 Related Work

We reviewed the literature on programmers' information-seeking behaviours and discussed the current approaches to integrating web searches with generative AI tools. Our work builds on this foundation by investigating the complementary roles of these tools and how they can be intertwined to support programming tasks.

2.1 Information Foraging in Programming

Programmers seek information to address programming problems [57]. The process encompasses a variety of cognitive tasks, including understanding unfamiliar code segments, making high-level decisions on selecting appropriate frameworks, and refining solutions based on feedback [6]. Central to this process are concepts of information foraging and sensemaking [52], where foraging describes the strategies employed to seek out and gather information effectively. Sensemaking describes the process of schematizing curated information and dissemination, including how programmers collect and organize information, which we do not specifically investigate in this work. During the foraging process, programmers need to search, collect, and organize information to concretize their goals and determine approaches for solving specific programming problems. Kittur et al. [25, 26] further explore how structured approaches can aid developers in managing web-based information, though at the potential cost of increased cognitive load. Information foraging theory has been widely applied to different domains, such as informing web design [31] or understanding source code navigation. Here, we focused on the web foraging behaviour in web search and GenAI.

2.1.1 Information Seeking with Web Search. Conventional web search remains a foundational tool for programmers' information-seeking activities, characterized by a non-linear, iterative process known as *orienteering* [64]. Unlike the "perfect" search engine's goal of *teleporting* users directly to their target, orienteering involves gradually refining search queries and navigating through information, leveraging human intuition and contextual understanding. This approach is essential for solving complex programming problems as it allows developers to dynamically explore and adjust their search strategies[26]. In programming, web search is used for planning code structure, debugging errors, researching documentation, and understanding concepts [18, 28]. Rather than seeking direct answers, programmers often search for a particular information source that leads them to their solution (e.g., searching for a particular tutorial to set up a web socket rather than directly searching for *"setting up web socket"*) [64]. Programmers also leverage the process of web foraging to gather meta-information [68], identify potential solutions, and synthesize relevant knowledge into a structured form that meets their specific criteria [34, 36]. Web foraging's step-by-step approach provides programmers with a sense of location [64] by reflecting on what has been searched and guiding the next steps. It also helps in understanding and contextualizing the search results [32]. These studies highlight the value of web search beyond merely delivering results, demonstrating benefits to users from the process of information foraging itself. Our research

builds on this notion, exploring how to reconcile web foraging with generative AI use, rather than simply using web search results as factual context for AI generation.

2.1.2 Information-Seeking Behaviours with Generative AI-Driven Conversational Search. Generative AI tools like ChatGPT and Copilot Chat are reshaping how programmers seek and utilize information [55, 62]. While a comprehensive theory of information foraging with these tools is still developing, studies indicate that programmers increasingly rely on them to accelerate familiar tasks and explore novel solutions [4]. GenAI-driven conversational search allows users to craft more complex instructions and queries, leveraging external information to enhance the generated results. This iterative approach enables users to refine outputs dynamically, building on the conversational history for greater precision and relevance. However, challenges persist in LLM-driven conversational search, including bias [2], filter bubbles [60], and the spread of misinformation that may appear credible [37]. Several research efforts have attempted to address these issues by incorporating uncertainty expressions in generated results [23] or highlighting uncertain segments in generated code [66]. While these approaches have successfully mitigated over-reliance on generative outputs, they also emphasize the importance of consulting additional sources to ensure accuracy and reliability.

As a result, research shows that programmers often continue to consult human-vetted sources like StackOverflow for validation [65]. Programmers frequently turn to web searches to validate the correctness of information by gathering broader contextual clues [1, 35]. These clues include metrics such as the number of discussions surrounding a solution or the recency of updates to a specific GitHub repository. This behaviour highlights how programmers' evaluation processes often involve synthesizing information from multiple sources and considering cues beyond the content provided by a single tool. Our research moves beyond merely comparing or integrating GenAI and web search. Instead, we aim to investigate whether and how programmers utilize both tools across different scenarios, uncovering the interplay between these tools and their role in programmers' information-seeking workflows.

2.1.3 Tools Supporting Information Foraging in Programming. Several tools have been developed to support programmers in their information-foraging activities, from traditional web search enhancements to AI-driven solutions. For example, Unakite [34] and Crystalline [36] facilitate the organization and comparison of information gathered from the web, helping developers make informed decisions. In contrast, AI tools like GitHub Copilot integrate the information-seeking process directly into the coding environment, offering real-time suggestions that are contextually relevant to the code being written [58]. These tools provide a foundation for supporting programmers in web-based information foraging. Our research builds on their approaches and implications, investigating how programmers use both web search and generative AI together, with a particular focus on supporting their transitions between these information-seeking tools.

2.2 Techniques Supporting the Integration of Web and GenAI

Current commercial tools often integrate Information Retrieval (IR) and Natural Language Processing (NLP) to combine web search with generative AI [3, 47]. Techniques like WeKnow-RAG [67] and WebGPT [42] demonstrate this by merging retrieved web search results to enhance the factual accuracy of AI-generated suggestions, aiming to balance AI efficiency with the contextual depth of web searches. Similarly, several fully autonomous web-based agents have been introduced, leveraging large multi-modal capabilities to interact with websites [61, 70]. However, these approaches often overlook the critical role of the programmer's information foraging process during web searches and struggle to predict the outcomes of actions in the absence of user-defined goals [9]. Our research emphasizes the importance of human-centric strategies outlined in established information foraging studies [27, 52, 56, 64], which purely algorithmic methods may not fully capture. Rather than asserting the superiority of either web foraging or generative AI, we argue that both offer unique, complementary benefits due to their distinct interaction paradigms. By understanding how programmers use both tools, we aim to provide design implications for future systems that empower programmers to navigate between web foraging and AI generation, maximizing the strengths of both under users' own control.

3 Interview Study

Our overall study process began with a retrospective interview (see Figure 1). The study aims to gain insights into programmers' current practices, processes, and challenges in their decision-making when using web search and GenAI in seeking information for solving programming problems.

3.1 Participants and Procedure

We recruited eight participants (P1-P8) (five men, three women; ages $24-29, M = 26.8, SD = 1.26$) through purposive sampling [14]. In our recruitment process, we sought participants experienced in programming and using LLM-driven code generation tools through screening questions in Appendix A.1. Recruited participants reported having four to eight years of programming experience and regularly used LLM-driven tools (8-18 times/week). Participants were compensated with CAD$20 for a 45-minute interview session.

We first asked each participant to provide a minimum of three recent examples of their ChatGPT usage for programming problem-solving, including instances involving web search as part of the process, to encourage participants to reflect on their utilization of both web search and generative AI. We then asked about challenges they faced when interacting with both tools, explored cases involving their combined usage, and inquired about their thought processes throughout the information-seeking process (see the interview questions in Appendix A.2). Interviews were audio-recorded and automatically transcribed. We analyzed the interviews using Reflecting on reflexive thematic analysis [7, 8], employing both a deductive approach and an inductive approach to identify and generate codes and themes. Two researchers on the team independently conducted the initial analysis to identify themes related to challenges, decision-making stages, and knowledge extraction.

In addition to the deductive themes, two inductive themes also emerged: influential factors and knowledge translation.

3.2 Collected Data

Participants provided 28 examples where they used both web search and generative AI for information-seeking and problem-solving. We identified iterative rounds of information-seeking based on the number of instructions (e.g., prompts or search queries) they wrote to achieve a specific goal. In most cases (26 out of 28, 26/28), participants engaged in 2-5 rounds of iterations involving both web search and prompting, while two cases involved only one web search session and a single round of prompting. We classified the tasks undertaken by participants into open-ended and closed-ended categories. In 21 out of the 28 cases, participants performed open-ended tasks, which did not have a clearly defined end goal, requiring exploration and iterative refinement. These included nine cases of exploratory data analysis and modelling, five cases of front-end development, four cases of data mining and web scraping, and three cases involving server-related tasks. The remaining seven cases were closed-ended tasks, where participants had specific end goals, such as implementing a particular feature or solving a bug.

3.3 Results

Overall, participants considered both tools had different strengths, but sometimes they are interchangeable. We identified how they perceived and used both tools and described their general workflow with the three decision-making stages.

3.3.1 Three Decision-Making Stages in the Information-Seeking Process. We identified three decision-making stages in the iterative information-seeking process using both tools, encapsulated in an integrated process model (Figure 2). The model is grounded in theories such as the sensemaking foraging loop [52] and Norman's seven stages of activity [44]. The flow begins with a `Goal` , where programmers identify their tasks and problems, followed by articulating `Intentions` to address them. For example, when classifying the Iris dataset, programmers break down the task into selecting a model and defining procedures for training and evaluation. They then perform `Actions` , choosing between web search or generative AI, a decision we term the *Selection* stage. After executing these actions, programmers analyze the results, akin to the `Understanding` step in the foraging loop, extracting essential information during the *Extraction* stage. Programmers then `Organize` this information, preparing it for the *Translation* stage, where they reformulate it into new queries or prompts. This iterative process may loop back to `Goal` for broad objectives or to `Intentions` for refining problem-solving strategies, dynamically adapting to the programmer's evolving needs and understanding.

Selection. In the earliest iterations, tool choice was largely shaped by two factors: (1) Familiarity with the domain: participants turned to web search for unfamiliar topics (*"fear that I do not have the knowledge to verify correctness"* [P1]) and used GenAI for areas in which they had more expertise; and (2) Clarity of goals: vague or poorly defined problems often prompted web searches to explore potential solutions, while more well-defined tasks prompted participants to rely on generative AI for targeted support. After a few trials, participants reassessed whether to persist with the same tool or switch. This decision was based on whether both tools appeared capable of producing similar outcomes, whether one tool failed to meet expectations, or whether an alternative tool offered distinct advantages. For example, P1 began with a web search to assess trade-offs among libraries and then switched to GenAI for a tailored solution about the chosen direction.

Many participants (6/8) frequently encountered repetitive or unhelpful results, which prompted them to *"fallback on web search"* [P2] or switch from web search to GenAI. Conversely, participants stuck with GenAI when they believed that *"the problem has enough solutions in the [AI] model"* [P5] or that the generated code could be more easily customized. Throughout this stage, decisions were guided by metrics such as *credibility, diversity, and up-to-dateness*. Web searches were valued for credibility checks, while GenAI was appreciated for its *customizability and efficacy*.

Extraction. As participants iterated, they skimmed and curated results to identify which details were relevant for subsequent prompts or queries. Early iterations often focused on *"defining the problem domain"* [P1] without much extraction, whereas later iterations involved reusing either concrete elements (e.g., copied code) or abstract knowledge gained from earlier exploration. However, most participants (7/8) lacked explicit methods to track the exact sources of extracted information. All participants reported a need for an external space (e.g., comments in a code editor or a Google Doc) to jot down thoughts and knowledge gained during the foraging process. Here we define the extraction stage not merely as the explicit act of copying and pasting but more broadly as the process of distilling specific knowledge or insights from one or multiple results. This process becomes evident through participants' reuse of the information.

Translation. Finally, participants transformed their understanding or extracted snippets into actionable prompts or search queries for the next round of information seeking. This stage involved reformulating information based on the interaction format required by each tool. For example, P7 integrated steps from a web tutorial with code examples as context to guide generated code that followed a specific flow. It was rare for participants to directly reuse verbatim content without translation, although three instances occurred when searching web to clarify keywords from generation.

Participants frequently modified or supplemented existing queries or prompts based on new insights. P4 noted that *"by gradually refining instructions, I can understand which words [utterances] might steer in a certain direction"*. In some cases, participants relied on GenAI to help translate information into usable queries. For instance, P5 used a prompt: *"What search queries should I use to find related projects related to the task described in the [pasted] context below?"* Most participants (6/8) found it easier to translate information into prompts for GenAI than into web search queries, as the latter required more concise and structured phrasing. Two participants specifically mentioned turning to GenAI when they struggled to formulate well-formed search queries, often due to gaps in domain knowledge. However, four participants noted challenges when switching from extended web search sessions to using GenAI, which arose from the need to decide which extracted information to include and how to align the generation with the provided context.

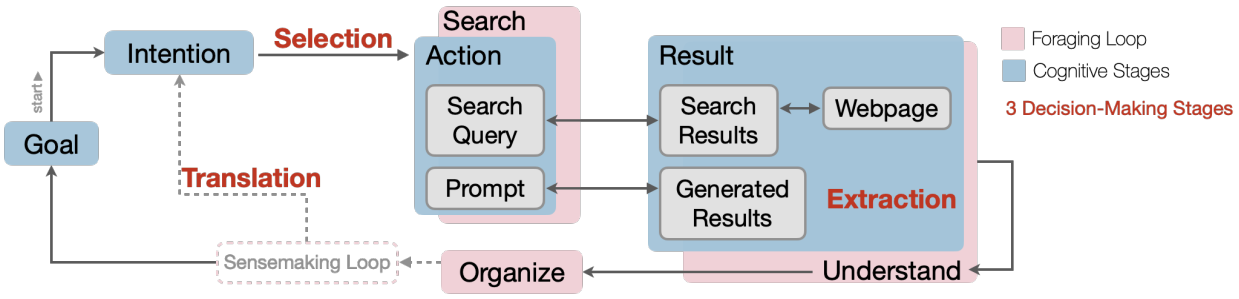


Figure 2: A process model describing interactions between programmers and search/GenAI. It incorporates cognitive stages, decision-making stages, and a foraging loop to illustrate how programmers navigate between tools.

3.3.2 Challenges in Using Both Tools for Information Seeking. Participants faced dilemmas in deciding when and how to switch tools. Some described an *exploring vs. exploiting* tension: continuing a current line of inquiry versus branching out to a new domain without fully mastering prior insights. Others wrestled with *supplementing vs. modifying* an existing query or prompt, uncertain if they should add new context or restructure the existing request.

While participants recognized the value of using both web search and GenAI, they encountered substantial hurdles when transferring information across tools. All participants described the process of copying, summarizing, and re-articulating content as labor-intensive and error-prone. P5 highlighted the difficulty of *"deriving the best outcome from both inputs,"* noting that each tool demanded a different representation of the same problem, making it challenging to maintain coherence.

Participants also pointed out that synthesizing information from multiple sources—often written in different styles or designed for different audiences—required significant cognitive bandwidth. The effort to align and reconcile these fragments of information introduced friction into their workflows and often led to delays or suboptimal decisions. A particularly common challenge was the issue of context loss during iterative tool-switching. Three participants noted that critical goal-related context often dissipated as information was transitioned between tools. As P2 remarked, they could *"go in completely different directions without realizing it."* This loss of continuity made it harder to track reasoning over time, especially in more complex or long-running tasks.

To mitigate these issues, participants expressed a desire for more integrated workflows—such as embedding web search results directly into code editors (P4) or enabling automatic transformation of curated web knowledge into AI-ready prompts. These suggestions underscore the need for systems that preserve information across tools and reduce the manual overhead of switching between fragmented information spaces.

4 Observational Study

Although we observed examples of how participants translated extracted knowledge between tools, categorizing these strategies was challenging without a complete view of the information flow and think-aloud data for each decision-making stage. As a result, we followed up with a comprehensive observational study (see Figure 1) to investigate the types of context being transmitted and how they are transformed throughout the different stages. The observational study aims to understand *how information flows* between tools, *what types of context* programmers extract and *how that context is translated* into subsequent queries, prompts, or code.

4.1 Participants

We recruited another 16 participants (P9-24; 9 identified as men, 7 as women; ages 20-32, $M = 24.8$, $SD = 2.41$) through convenience sampling methods including mailing lists, social media postings, and word-of-mouth. All participants were experienced programmers, with three to eight years of programming experience. We specifically invited individuals familiar with Python or JavaScript, as the tasks we designed were implemented in those languages. They were also familiar with utilizing both web search (6-18 times per week) and generative AI (5-12 times per week) for solving programming problems. To ensure a diverse capture of problem-solving processes, we recruited nine graduate students, four professional developers, and three undergraduate students. We organized the recruitment into batches, with each batch comprising four participants corresponding to four different study tasks (see Section 4.3).

4.2 Study Probe

We developed an integrated Chrome extension, depicted in Figure 3 to reduce the effect of tool bias [54]. The probe combines a sidebar displaying a code editor, task descriptions, and the GenAI-driven conversational interface, allowing us to focus on information context switching, then window context switching. Additionally, task descriptions within the interface were made non-selectable to prevent direct copying and pasting into the GenAI; also, all components of the interface were adjustable. System logs recorded participants' behaviour, including web search activities (e.g., query writing, tab switching), GenAI interactions (e.g., prompt writing, copy/pasting), and actions within the code editor. Detailed logs are provided in Table 2. To ensure we collected data on instances where participants were only viewing a component without interacting, we asked them to click on the component (AI chat, web page, or code editor) whenever they were looking at it, and double-click to interact with it (Figure 3 C).

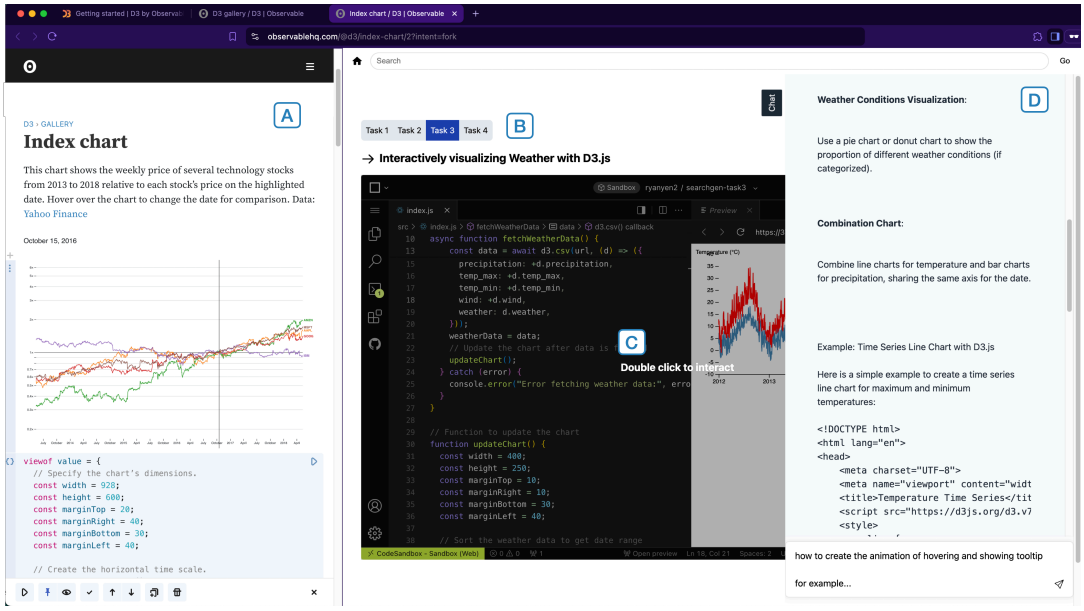


Figure 3: Our probe was implemented as a Chrome extension that appeared as a side panel (B) alongside the standard web browsing interface (A). The extension included several resizable components, such as task descriptions, a web-based code editor (C), and a GenAI-driven conversational interface (D).

4.3 Tasks

We designed four exploratory programming tasks to simulate scenarios where programmers seek information to solve problems. Each task included specific subtasks and requirements aimed at prompting participants to evaluate different approaches and to integrate their selected approach into a fully functional code editor with console access. We conducted iterative testing with $N = 3$ pilot participants to make sure tasks demanded a blend of high-level information gathering and low-level coding skills and could not be completed in a single attempt using current state-of-the-art generative AI technologies. The tasks encompassed diverse programming challenges, including implementing a browser-based code editor, performing sentiment analysis, creating visualizations with animation, and establishing a websocket connection with authentication. Full task details are provided in the Appendix B.

4.4 Procedure

After a 5-minute briefing, participants had 45 minutes to complete their tasks using our integrated probe featuring a code editor, a genAI chat interface, and web search access. Participants were assigned tasks based on their self-reported unfamiliarity, as measured by a score of ≤ 2 on a 5-point Likert scale from the screening questionnaire. This approach ensured that while participants were proficient in the programming language, they were relatively unfamiliar with the specific tasks, necessitating information seeking through generative AI or web search. Participants received a base compensation of CAD$25, with an additional CAD$5 incentive for successfully meeting all task requirements. Two experimenters observed and documented participants' behaviours throughout the study, focusing on web foraging techniques, prompt articulation based on gathered information, and integration of web search results into the problem-solving process. Following the task session, a 20-minute follow-up interview was conducted. We employed a mixed-methods approach [12] to triangulate data from interview responses, observational notes, and think-aloud protocols, which were all transcribed for subsequent analysis. Detailed interview questions and observation notes for the experimenters are available in Appendix C.

4.5 Data Collection and Analysis

All recorded logs are listed in Table 2. The system log data were further segmented into *information flow episodes*. Each episode begins when the user starts working on an information-seeking subtask and ends when they transition to another tool or complete the subtask. During the analysis, the *code editor* was considered as another important "tool" for identifying information flow, apart from the *web search* and *generative AI*. Within each information flow episode, users engaged with two or more tools, interacted with outputs, and decided on subsequent steps if the task remained incomplete. A total of 151 episodes were identified, with 13 considered outliers due to rapid tool-switching behaviour.

The segmented data were analyzed using iterative open coding, following the thematic analysis approach by Clarke and Braun [11]. Task episodes were coded into information flow types and tool usage patterns. Two researchers independently conducted open coding on the same set of information flow episodes to develop an initial codebook. The researchers then independently re-coded 20% of the episodes using the preliminary codebook. Inter-rater reliability was assessed using Cohen's kappa [30], resulting in an

Table 1: Context flow in tool switching. The types correspond to categories of context described in Section 4.6.1. Percentages indicate the proportion of each major context type among all occurrences of that specific context flow observed in the information flow episodes, with N indicating the total number of coded instances.

Context Flow	Predominant Type	Example	% of Types	#N
Web → Code	Text	P10 pasted code snippet from Kaggles' notebook to code editor	94.45%	36
Code → Web	Keyword	*"alternative methods to **SMOTE()**"* [P17].	94.74%	19
Web → GenAI	Gist/ Knowledge	*"I noticed [from the documentation] that SVG has to be declared first before the plot"* [P16].	72.73%	11
GenAI → Web	Gist	*"Can Socket.IO [library used by GenAI's results] authenticate connections?"* [P19].	92.31%	13
Code → GenAI	Keyword	*"why **scaleLinear()** not working"* [P13]?	83.78%	37
GenAI → Code	Text	P10 pasted the generated code of Naive Bayes to the code editor	95.56%	45

initial agreement of $K = 0.78$ on average. Discrepancies were resolved through discussion, and some types of flow and context are being merged. Using the finalized codebook, both researchers independently coded the entire set of episodes. The final inter-rater reliability achieved was $K = 0.93$. Any remaining disagreements were resolved through negotiation until 100% consensus. The final codebook included seven information flow types (Figure 4), four information context types and six context flow patterns (Table 1).

4.6 Results

We first describe the types of context that emerged and illustrate how programmers extract, transmit, and translate this context across tools.

4.6.1 Types of Context. Our logs, video recordings, and post-task interviews revealed four primary INFORMATION CONTEXTS (*Verbatim*, *Keywords*, *Gist*, and *Knowledge*) plus an additional ACTION CONTEXT referring to participants' prior activities and future plans.

Verbatim Snippets. Participants occasionally copied entire blocks of code or text from search results or AI-generated outputs into their code editor, or vice versa. For example, P10 imported large code segments from a Kaggle notebook directly into their local editor, while P18 pasted AI-generated code for a logistic regression pipeline. Although this verbatim reuse offered convenience, we found that it often required participants' extensive ad-hoc modifications due to partial content mismatches, such as library version conflicts or missing dependencies in the user's environment.

Keywords. When debugging or refining a solution, participants often extracted specific keywords (e.g., function names, error messages, or library tokens) and used them in another tool. For instance, P15 copied the text "`scaleLinear is not a function`" from the console and pasted it into a web search query to identify potential solutions. Similarly, P19 discovered the "`socket.emit`" syntax on one website and reused the exact token in a generative AI prompt to customize the solution further. This keyword-based approach is effective in web searches, as it enables concise articulation of queries. At the same time, it helped steer generated outputs in the desired direction.

Gist. Participants often skimmed through only a portion of a tool's output to extract the *gist*—a quick, distilled understanding of whether an approach seemed feasible. For example, P16 remarked, *"I already know it is wrong by looking at the first few lines of code since it asked me to use SMOTE."* This immediate recognition of misalignment led P16 to pivot to a web search to verify the dataset shape. During web searches, participants could observe convergence or disagreement among search results directly on the results page, benefiting from a richer information scent. This enabled them to refine their queries without needing to click on individual links.

Knowledge. This category of context represents the understanding that programmers develop over time. Participants often acquired Knowledge by reading documentation or browsing multiple web resources, then applied these insights when interacting with generative AI or coding. For instance, P16 glanced at a D3.js tutorial online, learning conceptually that an `<svg>` must be declared to place graphical elements. Although P16 never copied code verbatim, this high-level understanding shaped later AI prompts (*"I want to place circles inside an existing* `<svg>`*..."*). The presence of such knowledge was inferred from study notes and think-aloud protocols.

Action Context. Finally, the action context includes references to previous activities and plans for subsequent actions, enriching the input to better guide the search or AI interaction. We noted the relevance of ACTION CONTEXT whenever participants annotated a web query or AI prompt with details of prior steps (e.g., *"I tried searching for X and it didn't work,"* or *"I already tried following code, got a 404 error"*). For instance, P9 wrote, *"I was searching for highlighting code syntax using codemirror, but it is not working. Now I want to use Prism instead."* These meta-level descriptions allow a tool, especially GenAI, to better tailor follow-up suggestions while preventing redundant recommendations.

4.6.2 Information Flow Patterns. In this section, we focus on participants' cross-tool interaction rather than their behaviour within any single tool. We observed four primary patterns—*Singleton*, *Stash*, *Branch*, and *Merge*—collectively encompassing 7 information flows (A-G) depicting distinct strategies for weaving together web search and AI (Figure 4).

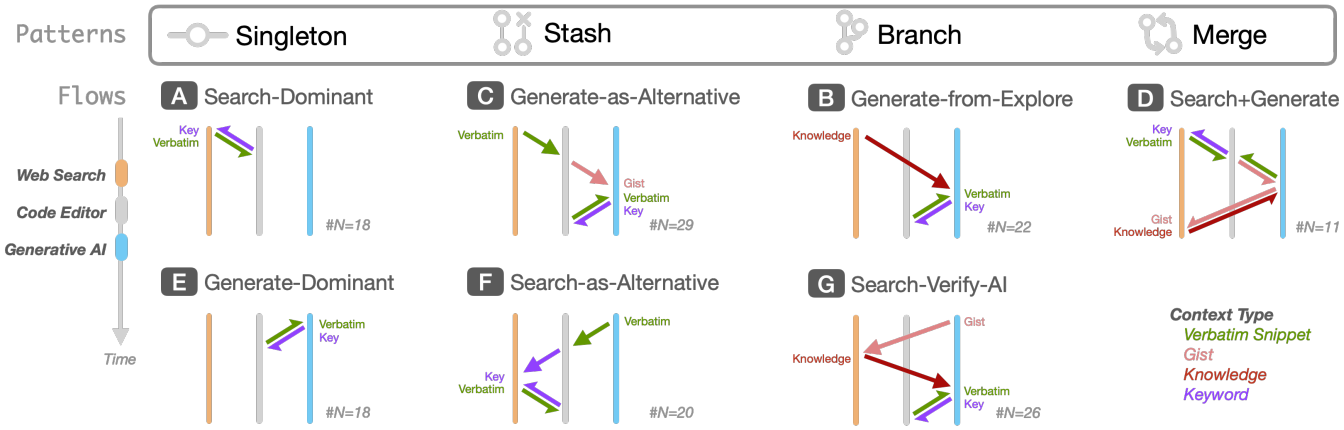


Figure 4: Four patterns and seven information flows identified from the study, illustrating how participants navigated tools and contexts. Arrow colours represent the predominant type of context transferred between tools (e.g., *Gist*, or *Verbatim Snippets*). The #*N* notation indicates the frequency of each flow, as determined from the final codebook analysis.

SINGLETON *(Flow A, E).* Some participants worked mostly within one tool until their session ended, switching only after completing a major milestone. In *Search-Dominant (Flow A)*, P11 began by gathering broad insights from multiple web pages, occasionally copying code snippets into an editor for future modifications. Conversely, in *Generate-Dominant (Flow E)*, participants like P12 relied almost exclusively on AI to debug a specific error. In both workflows, the transferred context was primarily textual, such as code syntax from the editor or verbatim snippets from search results or AI outputs. While translation between search results and the code editor was required, these interactions occurred ad hoc, typically after pasting one or more code snippets.

STASH *(Flow C, F).* Two distinct patterns emerged when participants encountered obstacles with their initial approach. In *Generate-as-Alternative (Flow C)*, participants started with web search until they failed to find a solution (e.g., P16 abandoned a WebSocket approach), prompting a sudden switch to AI. Conversely, in *Search-as-Alternative (Flow F)*, participants began with AI (e.g., for code generation) but shifted to web search when *"the AI's code kept failing"* [P13]. Some participants (5/18) noted it was harder to identify dead ends with AI compared to web search, as *"it always provides seemingly credible answers"* [P12]. This often resulted in abandoning partial progress, described by P23 as *"all the build-up is wasted,"* where they *"restart from scratch"* in a different tool with minimal carryover. While participants sometimes archived progress (e.g., leaving browser tabs open or revisiting prior chat history), the primary challenge was recalling orphaned context. As a result, participants rarely reused *Gist* or *Verbatim Snippets* directly, often starting fresh based on vague memories.

BRANCH *(Flow B, G).* Participants frequently initiated a new branch of information-seeking without abandoning the original one. In *Generate-from-Explore (Flow B)*, they first gathered high-level strategies or *Gist* from web searches (e.g., P19's conceptual understanding of D3.js) before prompting AI to transform that knowledge into workable code. This flow relied on transferring summarized insights rather than verbatim snippets. In these cases, participants maintained a new main branch while revisiting the original branch as a supplemental reference. Alternatively, in *Search-Verify-AI (Flow G)*, participants began with AI-generated code but consistently validated snippets or *Knowledge* through web searches. When verification confirmed the correctness of the AI's output, they returned to AI to refine their solution, maintaining consistency in the primary branch.

MERGE *(Flow D).* In contrast to BRANCH, participants following this flow did not maintain a single primary branch. Instead, both tools were used concurrently to complete interdependent subtasks. For example, P18 used AI for syntax suggestions and then validated the output's compatibility by consulting official documentation. In these cases, participants leveraged both tools based on their respective strengths. For instance, two participants used web search to progress while waiting for AI-generated responses to complete. While some participants (3/18) found this approach helped *"maximize the benefits of both tools"* [P24], others highlighted challenges, such as repeatedly carrying over INFORMATION CONTEXT (e.g., function names) or documenting ACTION CONTEXT to avoid redundant exploration.

4.6.3 Strategies for Context Translation. Unless the generated code could be directly used in the code editor, all outputs required translation to adapt to other tools. Effective context translation is essential for utilizing both tools complimentary while preserving information integrity. However, current tools experience challenges of context loss when adapting outputs to different information structures, and articulating vast context into actionable instructions (see Section 3.3.2). Below, we detail the common strategies observed.

Explaining Background Context. Participants often articulated their progress and prior efforts when crafting prompts for GenAI. This included describing web search outcomes, code written so far, or higher-level goals, providing the AI with necessary *action context*. For example, P11 explained, *"I've tried [X] library for syntax highlighting but need help integrating it with real-time editing."* This strategy was less explicit in web search, where participants relied

on concise terms (e.g., *"alternative methods for* `SMOTE()` *"*) to refine queries. Despite these differences, both approaches demonstrated how participants leverage background context to tools for more targeted results.

Anchoring Context Through Keywords During Transitions. Participants frequently reused specific keywords to maintain continuity across tools, assisting transitions and diverging results. These keywords often emerged from insights gained during earlier iterations. For instance, P13 repeatedly referenced `scaleLinear()`, such as "`scaleLinear()` *not working with negative values"*, across both web search and GenAI, ensuring results remained relevant to the same issue. This practice helped participants refine their exploration while minimizing redundant results or irrelevant tangents.

Preprocessing Context Before Using It. GenAI was often used to preprocess gathered information, such as combining, summarizing, or comparing results from multiple sources. For instance, P15 prompted, *"Combine [web search result] and [code snippet] to fit this specific library."* While this strategy saved time, two participants still preferred manual preprocessing, believing it to be essential for understanding and future decision-making. P24 explained, *"It's important to process the information myself to ensure I fully understand the options."* [P]reprocessing also appeared in search queries, where participants used Boolean operators (e.g., AND, OR) to integrate multiple sources systematically.

In more complex flows (B, D, G), participants combined partial solutions from different stages, weaving them back into the code editor or new prompts. For example, P15 validated one part of AI-generated code on the web, discovered a minor discrepancy, and appended that info as *"the [prior pasted search results] is not supported, can you fix the code?"* in a subsequent prompt. This incremental approach often resembled a branching search rather than a tidy linear progression.

Labelling Context. Some participants manually labelled extracted snippets to clarify their intended use or origin. For example, P22 labelled sections as *"for preprocessing only"* or *"valid for JWT authentication"* to ensure they were reused appropriately in subsequent queries or prompts. Similarly, in web search, P17 used labelling directly in queries, such as *"Django REST framework authentication best practices 2023 NOT outdated tutorials"*, to guide results toward specific needs while excluding irrelevant sources. This labelling strategy helped participants maintain clarity when navigating complex tasks, such as separating authentication logic from real-time messaging in a chat application.

Dumping Context. Occasionally, participants bypassed detailed processing and pasted raw information into GenAI prompts, relying on the tool to extract insights. For example, P13 pasted an error log from a failed WebSocket connection and asked, *"What steps should I take to resolve this issue?"* Similarly, some participants dumped raw information directly into code editors, using comments to maintain context. For instance, P14 added, *"// Issue with WebSocket connection—check JWT decoding logic here"*, to capture unresolved issues and revisit them later. Yet, this approach sometimes resulted in missed nuances, particularly in complex workflows requiring a deeper understanding of dependencies.

4.7 Summary of Empirical Insights and Design Dimensions

We propose five key dimensions for future tool design, inspired by our empirical findings on how programmers navigate web search, code editor and GenAI.

Interoperability: Reducing resistance between tools. A key consideration is the extent of integration and coupling between web search and GenAI tools, focusing not on creating an all-in-one melting pot but on facilitating information transitions across tools [21] and reducing viscosity [16]. For instance, maintaining a shared data structure for context or enabling adaptive transformations during import/export flows. Workflows such as *singleton* benefit from separate interfaces that distinguish broad exploration from targeted generation. Conversely, tightly coupled designs might embed web search into GenAI interfaces, enabling on-the-fly validation or interactive visualizations (e.g., hovering over code for AI suggestions). Prior tools like Blueprint [5] and Codelet [46] demonstrate embedding documentation and search within code editors by sharing context across and synthesizing a code-centric search view. Effective coupling should prioritize context transitions, easing *selection*, *extraction*, and *translation* processes without conflating tool modalities.

Adaptability: Supporting dynamic workflow navigation. The second dimension describes how tools support users in adapting to diverse flow patterns by providing mechanisms to switch between flows, maintain partial solutions, and merge or revisit paths as needed. Some linear workflows (SINGLETON) guide users through a single sequence of decisions, simplifying context tracking but limiting opportunities for simultaneous experimentation with multiple options. In contrast, many participants adopted branching workflows (BRANCH or MERGE), exploring alternative snippets (Flow B) or verifying AI outputs via web searches (Flow G). Tools could better support this behaviour by enabling users to spawn distinct *threads* of exploration [20], maintain multiple partial solutions, and merge or discard them as needed. Importantly, we emphasize that tools should not only support specific patterns but also afford flexibility, aligning with programming's iterative and non-linear nature.

Contextual Awareness: Externalizing and preserving context at different abstractions. The third dimension addresses how context is extracted, represented, and transmitted across tools. Both studies revealed that participants frequently struggled with context loss, particularly when transferring high-level *gist* or *knowledge* between tools. Effective designs must carefully manage diverse types of context, as programmers curate and extract context at varying levels of abstraction to facilitate transitions. Implicit approaches could automatically track user actions (e.g., search queries or copied snippets) and surface task-centric information when needed [5]. Conversely, explicit approaches might require users to label or specify context as described in Section 4.6.3, offering finer control at the expense of added effort. Since context is often reused, revisited, and pieced together, designs should reify context as a persistent, manipulable object. This would enable programmers to manipulate, organize, refine, and share ideas for reuse, as exemplified by Passages [17] or Memolet [69].

Translatability: Enabling context-aware translation across modalities. Another critical consideration is how tools facilitate the *translation* of extracted information into new prompts, queries, or code based on identified strategies. Currently, participants rely on manual reformulation strategies to adapt context across tools, such as labeling or summarizing. Tools could incorporate automation to summarize search results or generate AI-ready prompts, while still allowing users to refine or annotate outputs. Effective translation should also account for the ACTION CONTEXT, adapting outputs to generate diverse results across tools while avoiding redundant information buildup. Additionally, tools should provide curated spaces for knowledge externalization [24] and sensemaking [52], crucial for managing intensive extracted context. Prior tools like CoNotate [48], HunterGather [59], and Unakite [34] offer examples of information organization and could inform the translation stage.

Traceability: Maintaining clear provenance of actions. The final design dimension highlights the importance of preserving provenance and tracking history, particularly during the translation stage, where users transfer and adapt information across tools. This is crucial because context loss during translation can impede users' ability to make informed decisions or reflect on their current position in the information space (see Section 3.3.2). Tools should enable participants to retrace the origins of a snippet or understand the rationale behind an AI-suggested approach. For example, some users sought to verify the credibility of a library, while others wanted to recall the keywords or actions that led to discovering a valuable snippet. Tools that preserve an item's lineage can help users validate correctness, maintain orientation, and avoid redundant efforts [43, 45, 50]. Provenance tracking can range from simple source links to detailed chains of actions, offering a continuum of detail depending on user needs.

This provenance becomes particularly useful when tools aim to automate the extraction of ACTION CONTEXT or synthesize context during translation, as it provides the necessary semantic grounding for these processes. Prior work suggests capturing provenance at the action level [45], which records semantic information beyond event logs, reducing the need for additional user input, such as explicit task goals [15].

5 Towards Tool Design: A Proof-of-Concept Prototype

These design dimensions offer implications for designing tools that facilitate context transitions between web search and GenAI while adapting to programmers' workflows. To validate the generative power of these dimensions, we demonstrated a proof-of-concept prototype that operationalizes these design dimensions, addressing challenges like context management and transitions while exploring variations in programmers' information-seeking behaviours.

Reifying the Context. To operationalize the **Contextual Awareness** dimension, we introduce Browsette, an interactive object that reifies context extractions and can be manipulated, reused and shared across web search, code editor and GenAI. Programmers can create Browsette objects manually (e.g., by selecting a code snippet) or have them generated automatically. Browsette supports three key types of context (see Figure 5F): 1) *Background*: Automatically preserves high-level ACTION CONTEXT (e.g., search history or code edits); 2) *Summarization*: Stores distilled insights or overarching knowledge (e.g., a summarized pipeline). Programmers can guide summarization using specific instructions; 3) *Text*: Holds verbatim snippets or keywords. By treating context as a first-class object, Browsette ensures essential information is retained, revisited, and adapted seamlessly across tools.

Supporting the Information Flow. To address the **Interoperability** and **Adaptability** dimensions, our prototype introduces two key extensions. The first is a web-based code editor extension built on CodeMirror, enhanced with a Browsette dropbox and a conversational interface for generative AI. Users can drag and drop Browsette objects to seamlessly share context between the editor, AI interface, and browser, enabling both linear and branching workflows without losing relevant information. Additionally, the extension automatically generates *Background* Browsette objects based on code editing and AI interactions, preserving high-level context effortlessly.

The second extension is a Chrome browser add-on that augments Browsette functionality to web browsing. Users can drag selected content from web pages into a Browsette dropbox (see Figure 5A), capturing relevant snippets along with provenance metadata such as URLs and snippet locations. This supports branching exploration by allowing users to revisit, edit, or discard results as needed. Furthermore, users' foraging behaviours—such as query inputs, tab changes, and visited web pages—are synthesized into an up-to-date *Background* Browsette object with the web marked as the source (see Figure 5G). These extensions integrate disparate interfaces by providing synchronized access to the Browsette dropbox, reducing friction when switching between web search and AI assistance while preserving the unique affordances of each tool.

Translating Context Across Tools. To support **Translatability**, the system automatically adapts Browsette objects when dragged between components (e.g., from a web result to an AI prompt, see Figure 5C). For instance, dragging a `TfidfVectorizer` snippet into the GenAI tool customizes the prompt with relevant context, avoiding manual reentry of details. Similarly, dropping the same snippet into a web search box converts it into a focused query (e.g., *"movie review sentiment analysis with TF-IDF"*). Users can manually edit and save the transformed context or regenerate it based on updated information.

Preserving History and Provenance. Finally, our prototype addresses **Traceability** by logging item lineage and capturing the origins of each Browsette object. Actions such as code edits, snippet extractions, and AI interactions are recorded in a shared Firestore database, complete with timestamps and metadata. This allows users to trace a Browsette object's source (e.g., the original URL, see Figure 5B) and assess the reliability of snippets or approaches. Additionally, each Browsette object maintains a complete transaction history, enabling users to navigate across versions.

Implementation Details. The Chrome extension and the CodeMirror-based Next.js editor access a shared pool of Browsettes stored in Firestore's real-time database. To capture user interactions, the Chrome extension gathers click events, scrolling behaviours, and parsed page data; meanwhile, the CodeMirror editor logs editing

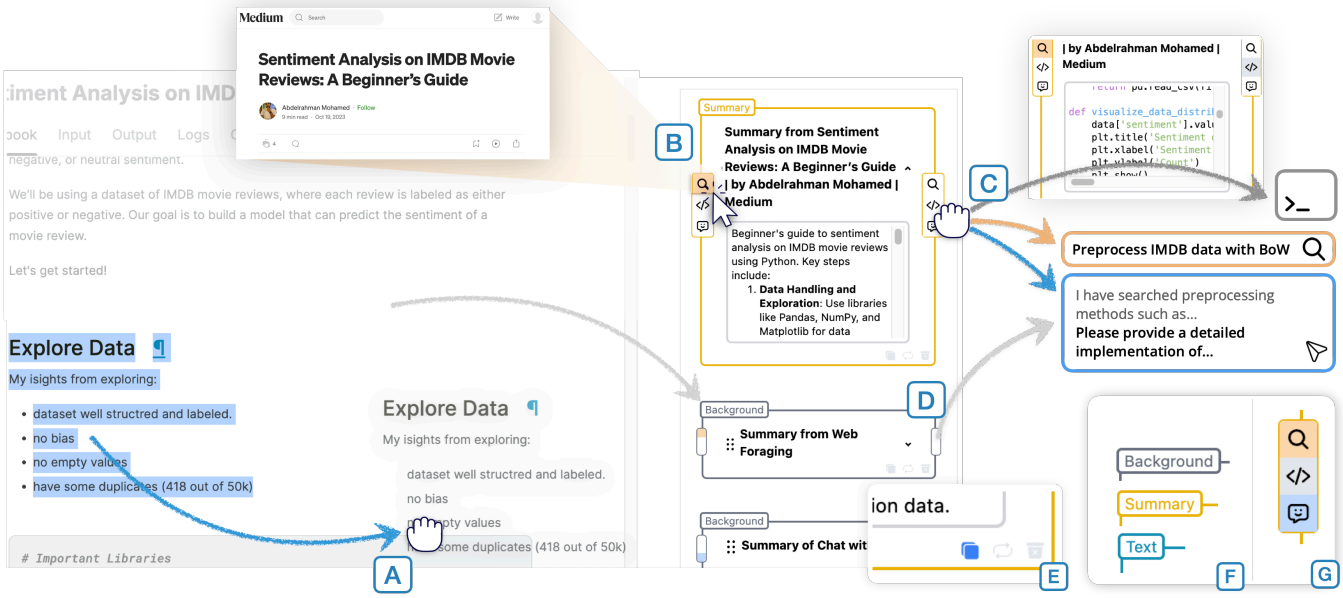


Figure 5: The overview of prototype design. (A) The user selects and drags text from a "source" (e.g., web page) into the extension, converting the context into a Browsette. (B) Clicking the source icon (e.g., 🔍) navigates back to the provenance, such as the source URL, chat ID, or code section. (C) The target icon allows users to translate the context to tailored inputs that can drop to the "target" tool (e.g., search queries, prompts, or code). (D) Automatically synthesizing users' ACTION CONTEXT as background information shared across tools, contextualizing the input. (E) Utility functions like copy, regenerate and delete provide additional control over the Browsette. (F) Three different types of Browsette. (G) Three types of source/target tools.

patterns, command usage, and concurrent code. These signals, along with the corresponding outcome of the action [9], are fed into a behavioural context pipeline that applies iterative weighted summarization with `GPT-4o`, taking into account factors such as signal importance [36] and recency. The summarized content is then stored as Browsettes, indexed by BM25 to surface the most relevant context on demand [53]. Finally, when translating context across tools, we instruct `GPT-4o` to rely on the retrieved context and user's latest query or command as context to augment the generation of code snippets, search query, or prompt.

5.1 Follow-Up User Study Setup

To examine whether and how the prototype, designed based on the proposed five design dimensions, influenced programmers' workflow patterns and the strategies they adopted, we conducted a follow-up study. We re-invited all 16 participants from the initial study, and 12 agreed to participate (7 identified as men, 5 as women; ages 22–28, $M = 23.9$, $SD = 1.87$). Participants retained their original identifiers (P9, P10, P12, P14, P15, P16, P18, P19, P21–P24). Participants were assigned an exploratory programming task similar in category to the observational study but with modified subtasks and criteria. After a 10-minute briefing using the prototype, participants had 45 minutes to complete all subtasks. System interactions were logged, and think-aloud verbalizations were recorded. A 15-minute semi-structured interview followed. We applied the same codebooks described in Figure 4 and Table 1 to ensure consistency in coding information flow and contextual factors. Two researchers independently coded the data using deductive thematic analysis.

5.2 Results

In the following, we report our results of the follow-up study, supplemented with context from our previous observational study where the same participants completed the same task without the prototype. Similar to the previous study, our focus is on reporting qualitative findings, using statistical data as supplementary evidence to provide interpretation rather than to claim statistical significance. Overall, all participants completed the task with a similar task completion time as that in the previous observational study ($M_{Browsette} = 35.92$ vs. $M_{prev} = 40.42$, $SD = 5.67$, see Figure 6A).

5.2.1 Strategic Problem-Solving and Changing Workflow. The prototype encouraged a more strategic and deliberate approach to problem-solving by fostering effective tool use and improving information flow. Most participants (11/12) reported planning their tool usage beforehand. Figure 6B also highlighted changes in information-seeking workflows. There was increased use of *Generate-from-Explore* ($Mdn_{Browsette} = 3.0$ vs. $Mdn_{prev} = 1.0$, p=0.003, r=0.781) and *Search+Generative* ($Mdn_{Browsette} = 2.5$ vs. $Mdn_{prev} = 0.5$, p=0.005, r=0.838) flows, with a corresponding decrease in *Generate-as-Alternative* ($Mdn_{Browsette} = 1.0$ vs. $Mdn_{prev} = 2.0$, p=0.008, r=0.883) , suggesting participants relied on web search and GenAI as complementary tools rather than standalone solutions. Although

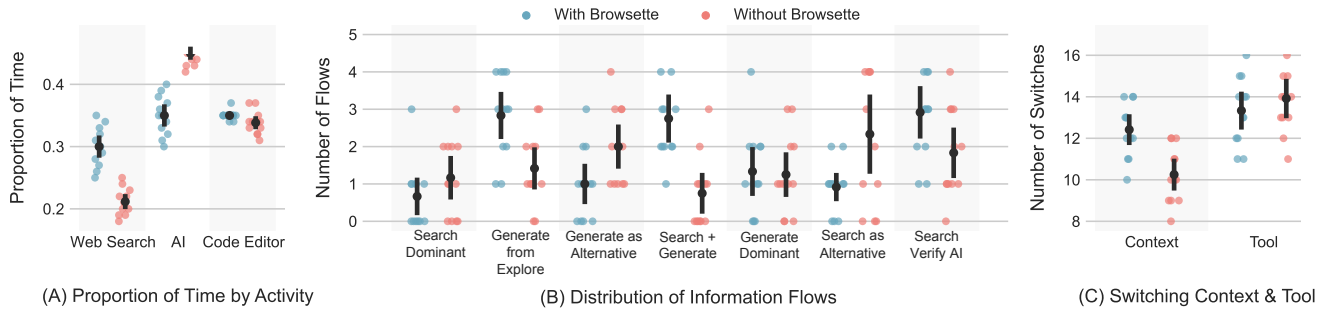


Figure 6: Comparison of (A) time proportion participants spent on three tools, information flows defined in Figure 4, and context/tool switching behaviour with and without the use of Browsette. Detailed statistical results can be found in Appendix E.

participants (9/12) occasionally switched to one tool as an alternative when encountering dead ends (STASH pattern), participants were more willing to switch tools as the prototype facilitated context translation without needing to start from scratch.

The creation and use of Browsettes enhanced reflective awareness for most participants (9/12), prompting more deliberate decision-making. P12 noted, *"The act of deciding what to capture made me more aware of the steps I was taking and why,"* while P19 mentioned becoming more critical of selected information to ensure relevance. These reflective practices enabled participants to focus on the programming task itself rather than deciding which tools to use, as noted by P22: *"I could focus more on the problem rather than switching tools."*

Additionally, participants relied more on web search when using the prototype, likely due to the automatic ACTION CONTEXT extraction feature. Some participants (3/12) highlighted this feature helped them *"reflect on what has been done"* [P22] and reuse *"trustworthy content from the web"* [P18]. This feature supported efficient transitions between tools by reducing redundant efforts and maintaining context across tools, as reflected in increased context carryover ($Mdn_{Browsette}$ = 12.5 vs. Mdn_{prev} = 10.0, p=< 0.001, r=0.883) . Although context-switching frequency (see Figure 6C) remained stable ($Mdn_{Browsette}$ = 13.5 vs. Mdn_{prev} = 14.0, p=0.035, r=0.793) , participants approached tool switching more strategically, leveraging captured context to adapt workflows dynamically.

5.2.2 New Strategies in Translation. Participants used the provided prototype to enhance their existing strategies and also developed new approaches for context translation.

Foraging for Browsette. A common behaviour observed among participants (10/12) was their active and purposeful approach to web foraging, aimed at creating more comprehensive Browsettes. For example, P15 remarked, *"I started keeping my searches more organized because I knew they would feed directly into the sidebar [Chrome extension] later."* However, the study also highlighted a need for finer control over the content captured within Browsettes. Two participants expressed a desire for greater customization, as *"some GenAI results are just for exploratory purposes"* [P9]. This feedback suggests that operationalizing the **Translatability** dimension requires a more dynamic and user-tailored approach to balance automation and control.

Translation as a Starting Point. Participants did not always apply Browsettes directly. Instead, some (6/12) often used the translated context as a starting point for further refinement. For example, P16 clicked on the GenAI icon on the Browsette from web search to transform it into a prompt. However, instead of directly dragging it into the input box, P16 returned to web search upon realizing the translated instruction lacked example usage from CodeMirror's documentation. P19 commented, *"it [translated output] was more of a launching pad to dig deeper into the problem."* It suggests that Browsettes sometimes serve as tools for ideation and exploration from the initial translations.

5.2.3 Remaining Challenges. Despite its benefits, several challenges with our prototype were noted, particularly around contextual misalignment. Four participants (4/12) observed that Browsette did not always capture the nuance or specificity needed for their tasks. P10 shared, *"[the Browsette] wasn't quite on point, and I had to adjust it manually to make it fit my needs."* Similarly, P22 remarked, *"Sometimes the context was too general, and I had to add more details to get the results I wanted."* While participants could edit Browsette, the effort required for these modifications was occasionally burdensome, highlighting limitations in the system's ability to fully encapsulate complex programming contexts. Additionally, two participants expressed that the Browsette Dropbox should afford more structured externalization strategies, such as hierarchical organization. This feedback underscores the need to explore alternative structures aligned with the **Traceability** dimension, such as version control-inspired approaches like Variolite [22].

6 Discussion

Here, we relate our findings to established HCI research on information foraging. We then discuss its connection to research on information sensemaking and discuss the applicability of our design dimensions beyond programming tasks.

6.1 Programmer-in-the-Loop Web Foraging

Many recommender systems have embedded information scent into model training [19, 63] or generation augmented through web search results [42, 67]. However, it is crucial to engage programmers "in-the-loop" beyond merely improving retrieval accuracy [10, 64].

The involvement of programmers in this process goes beyond refining recommendation or generation precision; it fosters deeper engagement in exploring, understanding, decision-making, and evaluating the presented information [35, 36]. Knowing that the context they provided directly influenced the AI's suggestions made participants (9/12) more confident in the results as they are similar to the *"solution seen on the web"* [P16]. This sense of control appeared to foster greater reliance on AI as a supportive tool, rather than a black-box solution [38].

Our findings reveal that web search plays a role far beyond fact-checking or validating AI-generated responses. The *Generate-from-Explore* and *Search-as-Alternative* flows identified in Figure 4 align with previous studies highlighting the use of web search for exploring alternative perspectives and comparing solutions [34, 56]. Through a web search, programmers navigate complex information spaces via iterative query construction [40]. This trial-and-error process is fundamental to learning and cognitive development, allowing programmers to refine their understanding through active engagement rather than passive information reception [36].

In our proof-of-concept prototype, we focused on supporting contextual buildup and information sharing across tools without automating the search process. This approach emphasizes the importance of maintaining user agency in the foraging process, ensuring that programmers remain central to information exploration and understanding [64]. While this study represents an initial exploration of this concept, future research could investigate further on balancing the automation and control, aligning with the **Translation** dimension.

6.2 Information Sensemaking

While our study did not investigate deeply into information sensemaking, a critical aspect of information-seeking, this step is essential for programmers navigating complex information spaces. Sensemaking helps when information becomes cluttered, preventing users from recognizing past activities [52, 56] and aiding in the progressive understanding of the information space [25]. Although sensemaking does not always require an external space for externalization, our designed Browsettes' dropbox in the prototype, which supports drag-and-drop for rearrangement, can offer some benefits of sensemaking. We observed participants engaging in sensemaking when organizing their lists of Browsettes. Some participants (5/12) used drag-and-drop to arrange Browsettes, placing all "Background" Browsettes upfront or categorizing them based on their source. As P23 noted, this helps *"knowing what context is being shared"* and avoids redundant contextualization.

In our follow-up user study, we found that some participants intentionally conducted searches to create more comprehensive Browsettes. This process is similar to creating summarized notes, jotting down the contexts they have encountered. This behaviour suggests that Browsettes can serve as a tool for externalizing and organizing information, supporting the sensemaking process. Future research could explore the specific structures that programmers need to effectively organize these Browsettes [26, 56]. Prior studies, such as those by Palani et al. [48, 49], have discussed recommending future search queries based on the sensemaking space articulated by users. Extending these ideas could be a promising direction, potentially enhancing the maintenance of provenance and **History**.

6.3 Beyond Programming Tasks and Future Directions

While our study focused on programming tasks, the potential applications developed based on the proposed design dimensions could extend beyond this domain. The core functionalities demonstrated in our prototype, such as supporting context transfer across tools, summarizing web foraging actions, and enabling strategic planning, can potentially benefit other complex, information-intensive activities. Tasks such as academic research [20], active reading [17], or personal information-seeking [29] face similar challenges of navigating vast information landscapes, synthesizing diverse sources, eliminating information silos, and iteratively refining understanding. These domains also require the use of several information sources and frequent context switching [41, 48] and are currently being investigated in the context of using generative AI for information seeking. However, as we consider expanding the use of AI-assisted information-gathering and processing tools, it is crucial to address privacy concerns, misinformation, potential biases, and the risk of creating echo chambers [60, 71]. Additionally, programming tasks usually come with a more structured format with predefined syntax, so the methods of extraction and translation should be fine-tuned based on the distinct nature of other scenarios. For instance, in academic research, the extraction process might focus more on capturing key arguments and methodologies from papers, and translation could involve reformulating these ideas into research questions. While the core principles remain applicable, the implementation details would need to be tailored to the requirements of each domain.

7 Limitations

Our study provides an empirical investigation into the use of web search and generative AI for information-seeking in programming. The qualitative approach enabled us to capture contextual insights into how participants interact with tools and to use the think-aloud protocol to reveal nuanced cognitive decisions influencing their workflows. While the qualitative method afforded depth and detail, it also came with certain limitations, such as potential observer effects. These limitations could be addressed in future studies by quantifying some effects through large-scale statistical tests. We also acknowledge that while our focus was on the process of information-seeking workflows, particularly the flow of context across tools, we did not compare the task outcomes across the two studies. This means we did not evaluate outputs using metrics such as precision, depth, or divergence. Future work could explore how the presence or absence of tool support affects the quality of output, as well as its impact on long-term learning and personal development. Additionally, the participant sample, drawn from a specific subset of programmers, may not fully capture the diversity of programming practices, workflows, or information-seeking behaviours across the broader programming community. Future studies could address this by including a wider range of participants from different domains, experience levels, and organizational contexts to further validate and extend our findings.

Next, the prototype supports information foraging mostly in a web-based environment, without testing its use within a separate IDE, which offers cross-window interactions more closely aligned with typical programmer workflows. Additionally, our study did not rigorously evaluate the behavioral context summarization or the retrieval-augmented generation pipeline employed to the prototype. These components were intentionally implemented as a simple but workable baseline to meet our design dimensions. Future work could explore alternative retrieval or generation approaches, leveraging advanced methods to improve the system's performance. Finally, while our findings suggest that the prototype helps context sharing across tools, we also observed occasional context misalignment between tools and the need of finer control over automation provided. Future work could focus on developing more sophisticated methods for summarizing and interpreting foraging behaviour based on the facet of **Translation**, enhancing the alignment of context and dynamically adjusting the level of automation.

8 Conclusion

This work presents a multi-step empirical investigation into how programmers integrate web searches and generative AI tools in their information-seeking workflows. Through interviews and an observational study, we synthesized five design dimensions that highlights insights into programmers' decision-making strategies, information flows, contextual details, and translation processes. Building on these design dimensions, we developed a prototype to operationalize its facets of design considerations and explored how design changes programmers information-seeking behavior through a follow-up user study. Our research contributes to the field by emphasizing the critical role of human involvement in information-foraging tools in programming, identifying five key dimensions of design for context transfer and sharing, and offering actionable design implications for future tools through the demonstration of our prototype.

Acknowledgments

This work is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant #RGPIN-2020-03966. We acknowledge that much of our work takes place on the traditional territory of the Neutral, Anishinaabeg, and Haudenosaunee peoples. Our main campus is situated on the Haldimand Tract, the land granted to the Six Nations that includes six miles on each side of the Grand River.

References

[1] Lynne M. Markus . 2001. Toward a Theory of Knowledge Reuse: Types of Knowledge Reuse Situations and Factors in Reuse Success. *Journal of Management Information Systems* 18, 1 (May 2001), 57–93. doi:10.1080/07421222.2001.11045671 Publisher: Routledge _eprint: https://doi.org/10.1080/07421222.2001.11045671.

[2] Abubakar Abid, Maheen Farooqi, and James Zou. 2021. Persistent anti-muslim bias in large language models. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*. 298–306.

[3] Perplexity AI. 2024. Perplexity AI. https://www.perplexity.ai/ Accessed: 2025-01-13.

[4] Shraddha Barke, Michael B James, and Nadia Polikarpova. 2023. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 85–111.

[5] 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*. 513–522.

[6] Joel Brandt, Philip J Guo, Joel Lewenstein, Mira Dontcheva, and Scott R Klemmer. 2009. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1589–1598.

[7] Virginia Braun and Victoria Clarke and. 2019. Reflecting on reflexive thematic analysis. *Qualitative Research in Sport, Exercise and Health* 11, 4 (2019), 589–597. doi:10.1080/2159676X.2019.1628806 arXiv:https://doi.org/10.1080/2159676X.2019.1628806

[8] Virginia Braun and Victoria Clarke. 2012. *Thematic analysis*. American Psychological Association.

[9] Hyungjoo Chae, Namyoung Kim, Kai Tzu-iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. 2024. Web Agents with World Models: Learning and Leveraging Environment Dynamics in Web Navigation. *arXiv preprint arXiv:2410.13232* (2024).

[10] Ed H. Chi, Peter Pirolli, Kim Chen, and James Pitkow. 2001. Using information scent to model user information needs and actions and the Web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Seattle, Washington, USA) *(CHI '01)*. Association for Computing Machinery, New York, NY, USA, 490–497. doi:10.1145/365024.365325

[11] Victoria Clarke and Virginia Braun. 2017. Thematic analysis. *The journal of positive psychology* 12, 3 (2017), 297–298.

[12] John W Creswell. 2021. *A concise introduction to mixed methods research*. SAGE publications.

[13] Paul Dourish. 2004. What we talk about when we talk about context. *Personal and ubiquitous computing* 8 (2004), 19–30.

[14] Ilker Etikan, Sulaiman Abubakar Musa, Rukayya Sunusi Alkassim, et al. 2016. Comparison of convenience sampling and purposive sampling. *American journal of theoretical and applied statistics* 5, 1 (2016), 1–4.

[15] David Gotz and Michelle X. Zhou. 2009. Characterizing users' visual analytic activity for insight provenance. *Information Visualization* 8, 1 (Jan. 2009), 42–55. doi:10.1057/ivs.2008.31

[16] Thomas RG Green. 1989. Cognitive dimensions of notations. *People and computers V* (1989), 443–460.

[17] Han L. Han, Junhang Yu, Raphael Bournet, Alexandre Ciorascu, Wendy E. Mackay, and Michel Beaudouin-Lafon. 2022. Passages: Interacting with Text Across Documents. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems (CHI '22)*. Association for Computing Machinery, New York, NY, USA, 1–17. doi:10.1145/3491102.3502052

[18] Jane Hsieh, Michael Xieyang Liu, Brad A. Myers, and Aniket Kittur. 2018. An Exploratory Study of Web Foraging to Understand and Support Programming Decisions. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 305–306. doi:10.1109/VLHCC.2018.8506517 ISSN: 1943-6106.

[19] Amit Kumar Jaiswal, Haiming Liu, and Ingo Frommholz. 2019. Information foraging for enhancing implicit feedback in content-based image recommendation. In *Proceedings of the 11th Annual Meeting of the Forum for Information Retrieval Evaluation*. 65–69.

[20] Hyeonsu Kang, Joseph Chee Chang, Yongsung Kim, and Aniket Kittur. 2022. Threddy: An Interactive System for Personalized Thread-based Exploration and Organization of Scientific Literature. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology (UIST '22)*. Association for Computing Machinery, New York, NY, USA, 1–15. doi:10.1145/3526113.3545660

[21] David R Karger and William Jones. 2006. Data unification in personal information management. *Commun. ACM* 49, 1 (2006), 77–82.

[22] Mary Beth Kery, Amber Horvath, and Brad A Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists.. In *CHI*, Vol. 10. 3025453–3025626.

[23] Sunnie SY Kim, Q Vera Liao, Mihaela Vorvoreanu, Stephanie Ballard, and Jennifer Wortman Vaughan. 2024. " I'm Not Sure, But...": Examining the Impact of Large Language Models' Uncertainty Expression on User Reliance and Trust. In *The 2024 ACM Conference on Fairness, Accountability, and Transparency*. 822–835.

[24] David Kirsh. 2010. Thinking with external representations. *AI & SOCIETY* 25, 4 (Nov. 2010), 441–454. doi:10.1007/s00146-010-0272-8

[25] Aniket Kittur, Andrew M Peters, Abdigani Diriye, and Michael Bove. 2014. Standing on the schemas of giants: socially augmented information foraging. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. 999–1010.

[26] Aniket Kittur, Andrew M. Peters, Abdigani Diriye, Trupti Telang, and Michael R. Bove. 2013. Costs and benefits of structured information foraging. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. Association for Computing Machinery, New York, NY, USA, 2989–2998. doi:10.1145/2470654.2481415

[27] Aniket Kittur, Andrew M Peters, Abdigani Diriye, Trupti Telang, and Michael R Bove. 2013. Costs and benefits of structured information foraging. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2989–2998.

[28] Amy J. Ko, Brad A. Myers, Michael J. Coblenz, and Htet Htet Aung. 2006. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. *IEEE Trans. Softw. Eng.* 32, 12 (dec 2006), 971–987. doi:10.1109/TSE.2006.116

[29] Andrew Kuznetsov, Joseph Chee Chang, Nathan Hahn, Napol Rachatasumrit, Bradley Breneisen, Julina Coupland, and Aniket Kittur. 2022. Fuse: In-Situ Sensemaking Support in the Browser. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology (UIST '22)*. Association for Computing Machinery, New York, NY, USA, 1–15. doi:10.1145/3526113.3545693

[30] JR Landis. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* (1977).

[31] Kevin Larson and Mary Czerwinski. 1998. Web page design: Implications of memory, structure and scent for information retrieval. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 25–32.

[32] Jimmy Lin, Dennis Quan, Vineet Sinha, Karun Bakshi, David Huynh, Boris Katz, and David R. Karger. 2003. The role of context in question answering systems. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems* (Ft. Lauderdale, Florida, USA) *(CHI EA '03)*. Association for Computing Machinery, New York, NY, USA, 1006–1007. doi:10.1145/765891.766119

[33] Fang Liu, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, Li Zhang, Zhongqi Li, and Yuchi Ma. 2024. Exploring and evaluating hallucinations in llm-powered code generation. *arXiv preprint arXiv:2404.00971* (2024).

[34] Michael Xieyang Liu, Jane Hsieh, Nathan Hahn, Angelina Zhou, Emily Deng, Shaun Burley, Cynthia Taylor, Aniket Kittur, and Brad A. Myers. 2019. Unakite: Scaffolding Developers' Decision-Making Using the Web. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST '19)*. Association for Computing Machinery, New York, NY, USA, 67–80. doi:10.1145/3332165.3347908

[35] Michael Xieyang Liu, Aniket Kittur, and Brad A. Myers. 2021. To Reuse or Not To Reuse? A Framework and System for Evaluating Summarized Knowledge. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW1 (April 2021), 166:1–166:35. doi:10.1145/3449240

[36] Michael Xieyang Liu, Aniket Kittur, and Brad A. Myers. 2022. Crystalline: Lowering the Cost for Developers to Collect and Organize Information for Decision Making. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems (CHI '22)*. Association for Computing Machinery, New York, NY, USA, 1–16. doi:10.1145/3491102.3501968

[37] Nelson F Liu, Tianyi Zhang, and Percy Liang. 2023. Evaluating verifiability in generative search engines. *arXiv preprint arXiv:2304.09848* (2023).

[38] Shuai Ma, Ying Lei, Xinru Wang, Chengbo Zheng, Chuhan Shi, Ming Yin, and Xiaojuan Ma. 2023. Who Should I Trust: AI or Myself? Leveraging Human and AI Correctness Likelihood to Promote Appropriate Trust in AI-Assisted Decision-Making. 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 759, 19 pages. doi:10.1145/3544548.3581058

[39] Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. *arXiv preprint arXiv:2212.10511* (2022).

[40] Gary Marchionini. 2006. Exploratory search: from finding to understanding. *Commun. ACM* 49, 4 (2006), 41–46.

[41] Catherine C Marshall, Morgan N Price, Gene Golovchinsky, and Bill N Schilit. 2001. Designing e-books for legal research. In *Proceedings of the 1st ACM/IEEE-CS joint conference on digital libraries*. 41–48.

[42] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332* (2021).

[43] Phong H Nguyen, Kai Xu, Andy Bardill, Betul Salman, Kate Herd, and BL William Wong. 2016. SenseMap: Supporting browser-based online sensemaking through analytic provenance. In *2016 IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, 91–100.

[44] Donald A Norman. 1986. Cognitive engineering. *User centered system design* 31, 61 (1986), 2.

[45] Chris North, Remco Chang, Alex Endert, Wenwen Dou, Richard May, Bill Pike, and Glenn Fink. 2011. Analytic provenance: process+interaction+insight. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI EA '11)*. Association for Computing Machinery, New York, NY, USA, 33–36. doi:10.1145/1979742.1979570

[46] Stephen Oney and Joel Brandt. 2012. Codelets: linking interactive documentation and example code in the editor. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. Association for Computing Machinery, New York, NY, USA, 2697–2706. doi:10.1145/2207676.2208664

[47] OpenAI. 2024. Introducing ChatGPT Search. https://openai.com/index/introducing-chatgpt-search/ Accessed: 2025-01-13.

[48] Srishti Palani, Zijian Ding, Austin Nguyen, Andrew Chuang, Stephen MacNeil, and Steven P. Dow. 2021. CoNotate: Suggesting Queries Based on Notes Promotes Knowledge Discovery. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM, Yokohama Japan, 1–14. doi:10.1145/3411764.3445618

[49] Srishti Palani, Yingyi Zhou, Sheldon Zhu, and Steven P. Dow. 2022. InterWeave: Presenting Search Suggestions in Context Scaffolds Information Search and Synthesis. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology (UIST '22)*. Association for Computing Machinery, New York, NY, USA, 1–16. doi:10.1145/3526113.3545696

[50] Beatriz Pérez, Julio Rubio, and Carlos Sáenz-Adán. 2018. A systematic review of provenance systems. *Knowledge and Information Systems* 57 (2018), 495–543.

[51] Peter Pirolli and Stuart Card. 1999. Information foraging. *Psychological review* 106, 4 (1999), 643.

[52] Peter Pirolli and Stuart Card. 2005. *The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis.*

[53] Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics* 11 (2023), 1316–1331.

[54] Miguel A. Renom, Baptiste Caramiaux, and Michel Beaudouin-Lafon. 2022. Exploring Technical Reasoning in Digital Tool Use. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 579, 17 pages. doi:10.1145/3491102.3501877

[55] Steven I Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D Weisz. 2023. The programmer's assistant: Conversational interaction with a large language model for software development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*. 491–514.

[56] Daniel M. Russell, Mark J. Stefik, Peter Pirolli, and Stuart K. Card. 1993. The cost structure of sensemaking. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems (CHI '93)*. Association for Computing Machinery, New York, NY, USA, 269–276. doi:10.1145/169059.169209

[57] Caitlin Sadowski, Kathryn T. Stolee, and Sebastian Elbaum. 2015. How developers search for code: a case study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 191–201. doi:10.1145/2786805.2786855

[58] 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).

[59] M. C. schraefel, Yuxiang Zhu, David Modjeska, Daniel Wigdor, and Shengdong Zhao. 2002. Hunter gatherer: interaction support for the creation and management of within-web-page collections. In *Proceedings of the 11th international conference on World Wide Web (WWW '02)*. Association for Computing Machinery, New York, NY, USA, 172–181. doi:10.1145/511446.511469

[60] Nikhil Sharma, Q. Vera Liao, and Ziang Xiao. 2024. Generative Echo Chamber? Effect of LLM-Powered Search Systems on Diverse Information Seeking. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 1033, 17 pages. doi:10.1145/3613904.3642459

[61] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*. PMLR, 3135–3144.

[62] Jiao Sun, Q Vera Liao, Michael Muller, Mayank Agarwal, Stephanie Houde, Kartik Talamadupula, and Justin D Weisz. 2022. Investigating explainability of generative AI for code through scenario-based design. In *Proceedings of the 27th International Conference on Intelligent User Interfaces*. 212–228.

[63] Nima Taghipour and Ahmad Kardan. 2008. A hybrid web recommender system based on q-learning. In *Proceedings of the 2008 ACM symposium on Applied computing*. 1164–1168.

[64] Jaime Teevan, Christine Alvarado, Mark S Ackerman, and David R Karger. 2004. *The Perfect Search Engine Is Not Enough: A Study of Orienteering Behavior in Directed Search.* Technical Report.

[65] 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 *Chi conference on human factors in computing systems extended abstracts*. 1–7.

[66] Helena Vasconcelos, Gagan Bansal, Adam Fourney, Q. Vera Liao, and Jennifer Wortman Vaughan. 2024. Generation Probabilities Are Not Enough: Uncertainty Highlighting in AI Code Completions. *ACM Trans. Comput.-Hum. Interact.* (Oct. 2024). doi:10.1145/3702320 Just Accepted.

[67] Weijian Xie, Xuefeng Liang, Yuhui Liu, Kaihua Ni, Hong Cheng, and Zetian Hu. 2024. WeKnow-RAG: An Adaptive Approach for Retrieval-Augmented Generation Integrating Web Search and Knowledge Graphs. arXiv:2408.07611 [cs.CL] https://arxiv.org/abs/2408.07611

[68] Ka-Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. 2003. Faceted metadata for image search and browsing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Ft. Lauderdale, Florida, USA) *(CHI '03)*. Association for Computing Machinery, New York, NY, USA, 401–408. doi:10.1145/642611.642681

[69] Ryan Yen and Jian Zhao. 2024. Memolet: Reifying the Reuse of User-AI Conversational Memories. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, USA) *(UIST '24)*. Association for Computing Machinery, New York, NY, USA, Article 58, 22 pages. doi:10.1145/3654777.3676388

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

[71] Jiawei Zhou, Yixuan Zhang, Qianni Luo, Andrea G Parker, and Munmun De Choudhury. 2023. Synthetic Lies: Understanding AI-Generated Misinformation and Evaluating Algorithmic and Human Solutions. 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 436, 20 pages. doi:10.1145/3544548.3581318

A Survey and Interview Questions for Retrospective Interview

A.1 Eligibility Screening Survey

1. **How confident are you in your overall programming experience?**
   1: Very Inexperienced
   2: Inexperienced
   3: Moderately Experienced
   4: Experienced
   5: Very Experienced
2. **How many years of programming experience do you have?** years
3. **How familiar are you with AI code generation tools (e.g., GitHub Copilot, ChatGPT)?**
   1: Not Familiar
   2: Slightly Familiar
   3: Moderately Familiar
   4: Familiar
   5: Very Familiar
4. **Over the past few weeks, how often did you typically employ AI code generation tools such as OpenAI's Codex, GitHub Copilot, or ChatGPT for your programming tasks?** (e.g., times per week)

A.2 Semi-Structured Interview Questions

Programming Workflow and Tool Integration

1. **Programming Workflow Integration:** Can you describe your typical programming workflow, particularly emphasizing how you utilize code synthesis tools like Copilot and web search in this process?
2. **Decision Making between GPT and Web Search:** How do you decide when to use tools like GPT and when to resort to web search during your coding process? Could you provide a specific example illustrating this decision-making process?

Information Seeking and Evaluation in Programming

1. **Information Requirements:** When you begin looking for information during programming, what specific types of information are you usually seeking? (e.g., syntax clarification, algorithmic approaches, best practices)
2. **Assessment of Information Quality:** What criteria do you use to determine whether a search or generated result is good enough for your needs? What factors are important to you?
3. **Determining Importance of Information:** What kind of information do you consider as most important from the generated or search results?
4. **Synthesizing Information from Multiple Sources:** Can you describe how you synthesize or combine information from different sources (like Copilot, web search, forums)? How do you resolve conflicts or discrepancies in information?
5. **Long-term Information Retention:** When you find particularly valuable information, how do you ensure its retention for future use? Do you have a system for organizing or bookmarking useful resources?

Challenges and Limitations of Both Tools

6. **Challenges and Limitations:** Could you discuss some challenges or limitations you've encountered with both tools? How does the other tool help in overcoming these challenges?
7. **Web Search Efficacy:** Can you provide an example where web search helped you gain a better understanding of a programming concept or language feature that tools like Copilot alone couldn't provide?
8. **Future of GPT and Web Search:** In your opinion, do you think the advancement of technologies like GPT-4 with internet and web scraping access could eventually replace traditional web search for programming-related queries?

B Tasks

B.1 Task 1: Implement Browser-Based Collaborative Code Editing

Task Description: You are tasked with implementing a browser-based collaborative code editing tool that allows multiple users to edit the same code document in real-time. The tool should support syntax highlighting and real-time collaborative editing. Try to gather information on libraries, frameworks, and best practices needed to complete this task. Document your process and decisions as you go.

Goals:

- Understand the requirements and components needed for collaborative code editing.
- Identify suitable libraries and frameworks for real-time synchronization and syntax highlighting.
- Implement a basic prototype demonstrating real-time collaboration between two users with JavaScript.

Steps:

1. Start by researching existing libraries and frameworks that support real-time synchronization (e.g., WebSockets, Firebase, etc.).
2. Search for documentation or tutorials on how to integrate these libraries into a web application.
3. Investigate how to implement syntax highlighting in the editor (e.g., using CodeMirror, Monaco Editor, etc.).
4. Combine the information gathered to create a basic prototype.
5. Document your search queries, sources, and any code snippets used in your implementation.

B.2 Task 2: Sentiment Analysis on Movie Reviews Using Web Scraping and Machine Learning

Task Description:

Your task is to build a sentiment analysis model to classify movie reviews as positive or negative. Start by scraping movie reviews from a website (e.g., IMDb, Rotten Tomatoes) and then use this data to train and evaluate a machine learning model. The model should achieve a reasonable level of accuracy. Try to gather information on web scraping techniques, data preprocessing, model selection, and evaluation metrics.

Goals:

- Understand how to scrape data from websites using tools like BeautifulSoup or Scrapy.
- Learn about preprocessing textual data for machine learning.
- Implement and evaluate a sentiment analysis model using libraries like Scikit-learn or TensorFlow.

Steps:

1. Research different web scraping tools and techniques suitable for scraping movie reviews.
2. Search for tutorials on text preprocessing and sentiment analysis.
3. Investigate how to implement and evaluate sentiment analysis models using Scikit-learn or TensorFlow.
4. Combine the information gathered to scrape movie reviews, preprocess the data, and implement the model.
5. Document your search queries, sources, and any code snippets used in your implementation.

Starter Code for Web Scraping:

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

def scrape_reviews(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    reviews = []
    return reviews

url = 'https://www.example.com/movie-reviews'
reviews = scrape_reviews(url)

# Save scraped reviews to a CSV file
df = pd.DataFrame(reviews, columns=['review'])
df.to_csv('movie_reviews.csv', index=False)
```

Starter Code for Sentiment Analysis:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

df = pd.read_csv('movie_reviews.csv')

# Preprocess the data
vectorizer = TfidfVectorizer(stop_words='english')

# Split the data

# Train a logistic regression model
model = LogisticRegression()

# Evaluate the model
```

B.3 Task 3: Developing a Real-Time Weather Dashboard with Web Scraping and Visualization

Task Description:

Your task is to create a real-time weather dashboard that displays weather information for multiple cities. Start by scraping weather data from a reliable website (e.g., Weather.com, AccuWeather) and then use this data to create dynamic visualizations using a library like D3.js or Plotly. The dashboard should update in real-time to reflect current weather conditions. Try to gather information on web scraping techniques, data visualization, and creating dynamic web applications. Document your process and decisions as you go.

Goals:

- Understand how to scrape real-time data from websites using tools like BeautifulSoup or Scrapy.
- Learn about creating dynamic visualizations with libraries like D3.js or Plotly.
- Implement a web application that displays real-time weather data.

Steps:

1. Research different web scraping tools and techniques suitable for scraping weather data.
2. Search for tutorials on creating dynamic visualizations with D3.js or Plotly.
3. Investigate how to integrate real-time data updates into a web application.
4. Combine the information gathered to scrape weather data, create visualizations, and implement the real-time dashboard.
5. Document your search queries, sources, and any code snippets used in your implementation.

Starter Code for Web Scraping:

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

def scrape_weather(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    weather_data = []
    return weather_data

url = 'https://www.example.com/weather'
weather_data = scrape_weather(url)

# Save scraped weather data to a CSV file
df = pd.DataFrame(weather_data)
df.to_csv('weather_data.csv', index=False)
```

Starter Code for Real-Time Visualization:

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Real-Time Weather Dashboard</title>
    <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
</head>
<body>
    <h1>Real-Time Weather Dashboard</h1>
    <div id="weather-dashboard"></div>
    <script>
        async function fetchWeatherData() {
            const response = await fetch('weather_data.csv');
            const data = await response.text();
            const rows = data.split('\\n').slice(1);
            const weatherData = rows.map(row => {
                const [city, temperature] = row.split(',');
                return { city, temperature: parseFloat(temperature) };
            });
            return weatherData;
        }

        async function updateDashboard() {}

        setInterval(updateDashboard, 10000); // Update every 10 seconds
        updateDashboard();
    </script>
</body>
</html>
```

B.4 Task 4: Building a Real-Time Chat Application with WebSockets and Authentication

Task Description:
Your task is to build a real-time chat application that supports multiple users, utilizing WebSockets for real-time communication and JWT (JSON Web Tokens) for authentication. Users should be able to register, log in, and join chat rooms. Try to gather information on setting up WebSocket connections, implementing JWT authentication, and creating a basic user interface. Document your process and decisions as you go.

Goals:

- Learn how to set up WebSocket connections for real-time communication.
- Understand how to implement JWT authentication in a web application.
- Develop a basic user interface for a chat application that supports multiple chat rooms.

Steps:

1. Research how to set up a WebSocket server and client.
2. Search for tutorials on implementing JWT authentication in a web application.
3. Investigate how to integrate WebSocket communication and JWT authentication into a single application.
4. Combine the information gathered to implement the real-time chat application.
5. Document your search queries, sources, and any code snippets used in your implementation.

Starter Code for WebSocket Server:

```
const WebSocket = require('ws');
const jwt = require('jsonwebtoken');
const wss = new WebSocket.Server({ port: 8080 });

wss.on('connection', (ws) => {
    ws.on('message', (message) => {
```

```
        const { token, data } = JSON.parse(message);
        try {
            const user = jwt.verify(token, 'your_secret_key');
            // Broadcast message to all connected clients
        } catch (error) {
            ws.send(JSON.stringify({ error: 'Authentication failed' }));
        }
    });
});

console.log('WebSocket server running on ws://localhost:8080');
```

Starter Code for JWT Authentication:

```
const express = require('express');
const jwt = require('jsonwebtoken');
const bodyParser = require('body-parser');

const app = express();
app.use(bodyParser.json());

const users = []; // In-memory user storage for simplicity
app.post('/register', (req, res) => {});
app.post('/login', (req, res) => {});

app.listen(3000, () => {
    console.log('Authentication server running on http://localhost:3000');
});
```

C Observation Notes and Interview Questions from Observational Study

C.1 Observational Notes

Reasons for Switching Tools:

- When switching tools, what is the reason?
- What challenges are prompting the context switching?

Behaviors During Web Foraging:

- Observe how participants navigate web resources (e.g., search engines, forums, documentation).
- Track interactions such as typing search queries, scrolling, clicking links, and reading content.

Knowledge Extraction:

- What information is extracted and how is it used?
  - Following search?
  - For code generation?
  - Directly to the code editor?
- Note how they integrate this information into their problem-solving approach.

Writing Prompts:

- Prompts or decisions potentially influenced by web-searched information.
- Criteria to determine the relevance and usefulness of information.

Failure & Repair:

- Note any iterative processes where participants revisit web searches or refine their generative AI outputs based on new findings.

C.2 Semi-Structured Interview Questions

General Questions:

- Can you briefly describe your workflow of problem-solving using both tools?

Reason for Conducting Web Searches:

- Can you describe specific instances during the task when you felt the need to conduct a web search?

Criteria for Information:

- What criteria did you use to determine if the information you found was relevant or useful?

Foraging Process:

- Can you walk me through your process of gathering information from the web?

Knowledge Extraction:

- What specific knowledge did you extract from your web searches that helped you with the task?
- How did you integrate the information found online into your approach to solving the task?

Articulating Prompts:

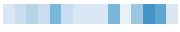
- Can you give an example of a prompt you created using the knowledge from your web searches?
- How did you decide what information to include or omit when formulating your prompts?

Extra:

- Did you find the combination of web searching and generative AI more effective than using one method alone? Why or why not?
- What improvements would you suggest for better integrating web search and generative AI in future tasks?

D Logs

Table 2: Types of logs collected from the study probe, including name, descriptions, corresponding activity frequencies visualized as heatmaps, and whether the type of logs is being used to create Browsette contained ACTION CONTEXT.

	Event	Description	Activity Frequency*	Used
Web Search	text selection	User selecting a segment of text		✔
	copy/paste	When the user copies text from the web page that is more than one word and pastes the text into the input query		✔
	click	When the user clicks on text, a button, or a link within the web page		✖
	query writing	When the user writes the search query		✔
	tab switching	When the user switches between tabs		✖
Gen AI	prompt writing	When the user types instructions in the input box of the chat		✔
	result	The Markdown result returned from the generative AI		✔
	copy/paste	When the user copies content from the result or pastes content into the input box		✔
Code Editor	focus/unfocus	When the user focuses or unfocuses the code editor		✔
	copy/paste	When the user copies or pastes content from or to the code editor		✔

* The heatmaps display user activity frequency over time intervals of three minutes, with colour intensity from light to dark blue (0 to 13) indicating activity levels.

E Final User Study Results and Stats

Table 3: Statistical Comparisons between Conditions (with and without Browsette)

Category	Comparison	Mdn (with)	Mdn (without)	p-value	Effect Size (r)
Time	Task Time	35.5	42.5	0.016	0.68
	Portion in Web Search	0.21	0.30	<0.001	0.88
	Portion in AI	0.45	0.35	<0.001	0.88
	Portion in Code Editor	0.34	0.35	0.050	0.75
Flows	Search-Dominant (Flow A)	0.5	1.0	0.196	0.68
	Generate-from-Explore (Flow B)	3.0	1.0	0.003	0.78
	Generate-as-Alternative (Flow C)	1.0	2.0	0.008	0.88
	Search+Generate (Flow D)	2.5	0.5	0.005	0.84
	Generate-Dominant (Flow E)	1.0	1.0	0.952	0.39
	Search-as-Alternative (Flow F)	1.0	2.5	0.072	0.59
	Search-Verify-AI (Flow G)	3.0	1.5	0.110	0.46
Switching	Context Switching	10.0	12.5	<0.001	0.88
	Tool Switching	13.5	14.0	0.035	0.793