

Bespoke Visual Assistance: What and How do Blind and Low-Vision People Create with Agentic Programming?

Ellie Seehorn
seehorn@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Gene S-H Kim
genekim@mit.edu
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA

Aziz Zeidieh
azeidi2@illinois.edu
University of Illinois
Urbana-Champaign
Champaign, Illinois, USA

Ather Jammaa
atherjammaa@yahoo.com
Independent Consultant
Detroit, Michigan, USA

Kun Lee
klee746@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Aditi Shah
shah.aditi@microsoft.com
Microsoft Corporation
Redmond, Washington, USA

Jaylin Herskovitz
jaylin.herskovitz@tufts.edu
Tufts University
Medford, Massachusetts, USA

Anhong Guo*
anhong@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Venkatesh Potluri*
potluriv@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

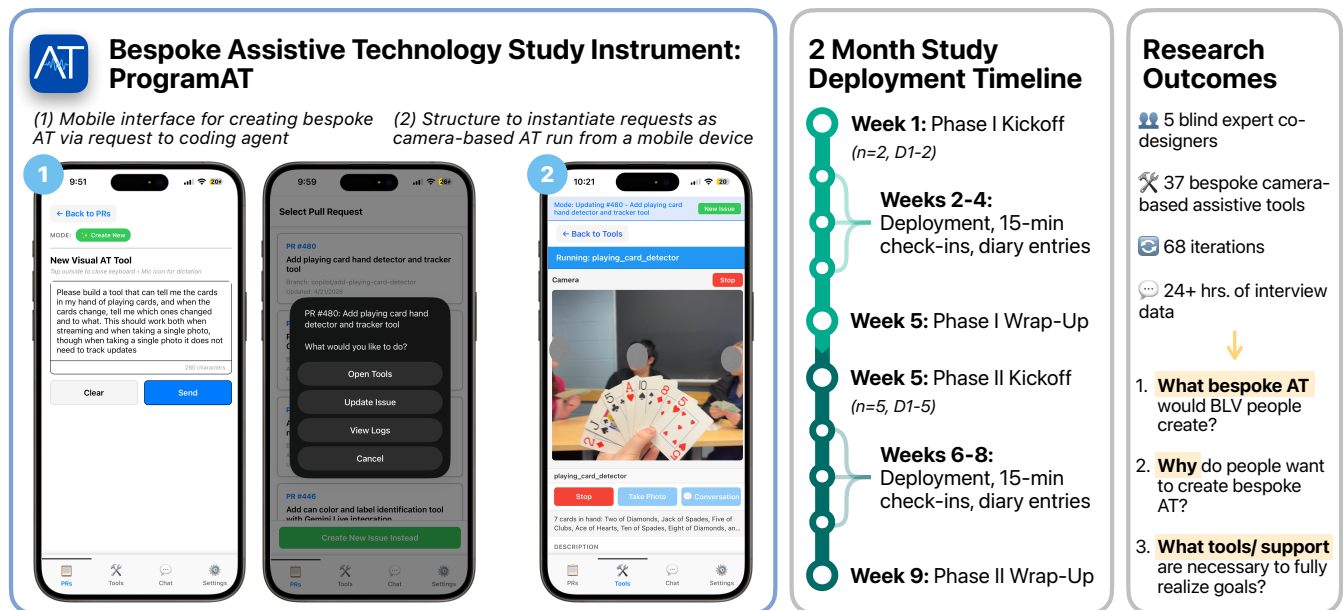


Figure 1: This paper explores how agentic programming might be used to create personal, bespoke assistive technologies. We created a mobile agentic programming interface (ProgramAT) as a means to study this through a longitudinal deployment and co-design. Through a two month study, co-designers created 37 bespoke assistive technologies. We analyze what was created, why, and how we might better support assistive technology creation in the future.

*Equal supervision.



This work is licensed under a Creative Commons Attribution 4.0 International License.
ASSETS '26, Vila Nova de Gaia, Portugal
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2521-0/2026/10
<https://doi.org/10.1145/3797867.3828973>

Abstract

AI-powered assistive technologies have long supported blind and low vision (BLV) people in everyday tasks, but they are general-purpose and often fall short of meeting complex, individualized, in-situ accessibility needs. Though agentic programming tools, like GitHub Copilot, have the potential to bridge this gap by lowering the technical barriers to building personal AT using natural language, the practical applicability of this creation paradigm has been

unknown. We address this knowledge gap through a two-phase longitudinal co-design study with five tech-savvy BLV users using ProgramAT, an agentic programming tool that supports the creation, iteration, and testing of camera-based AT. Overall, co-designers created over 37 tools, with some addressing needs unmet by any existing commercial AT—such as identifying Uber rides or interpreting hand gestures. Qualitative feedback from our co-designers and analysis of development logs surface BLV strategies for tool creation, along with key challenges including model capability limits, specification conflicts, and barriers to successful creation. We discuss recommendations to provide appropriate conversational scaffolding, community tool sharing capabilities, and support for specialized models and personal datasets for future agentic programming environments to empower BLV users to create bespoke visual assistance for themselves.

CCS Concepts

• **Human-centered computing** → **Accessibility**; **Empirical studies in accessibility**; **Accessibility technologies**; *Natural language interfaces*; **Empirical studies in HCI**; • **Software and its engineering** → **Automatic programming**; *Software prototyping*.

Keywords

Do-It-Yourself, Generative AI, Visual Assistive Tools, Accessibility

ACM Reference Format:

Ellie Seehorn, Gene S-H Kim, Aziz Zeidieh, Ather Jammoo, Kun Lee, Aditi Shah, Jaylin Herskovitz, Anhong Guo, and Venkatesh Potluri. 2026. Bespoke Visual Assistance: What and How do Blind and Low-Vision People Create with Agentic Programming?. In *The 28th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '26)*, October 25–28, 2026, Vila Nova de Gaia, Portugal. ACM, New York, NY, USA, 24 pages. <https://doi.org/10.1145/3797867.3828973>

1 Introduction

AI-powered assistive technologies (AT) such as SeeingAI [74], Be-MyAI [31], and OKO [5] have long supported everyday tasks for blind and low-vision (BLV) users¹. These tools, however are designed to be general purpose and to fit simple, common use cases [46], leaving them insufficient for many complex or individualized accessibility needs. For example, these tools do not cater to tasks that involve multiple steps (e.g. checking into a hotel), require individualized context (e.g. finding personal objects) and related to personal interest (e.g. applying makeup). [46, 54, 64, 67, 68]. The one-size-fits-all approach to AT has detrimental consequences to completing desired tasks efficiently [46, 51] and with confidence [8], contributing to tool abandonment [80]. Recent work investigates DIY approaches for AT creation, such as structured authoring of formulaic-purpose tools [47], surfacing augmentative functionality on top of commercial tools [45], or using LLMs as support for physical making processes [58]. These efforts highlight a promising trajectory toward end-user empowerment. However, a gap remains between the desire for customization and the practicality of broad

implementation. While many users prefer bespoke AT tailored to their specific needs [30], realizing such solutions depends on intensive collaboration with expert designers, making routine personalization impractical. BLV users, however, have clear ideas of what they would want to accomplish if tools allowed it [34, 46]; thus, the core problem is not in users not knowing what they need, but rather, in their lacking accessible means to author AT in-situ.

Agentic coding tools, such as GitHub Copilot, have advanced possibilities by integrating natural language with code generation, thus lowering the technical barrier to building complex systems. This paradigm promises a future where anyone can be a programmer: able to build customized tools and workflows for any use case they desire. For BLV users, this offers the ability to author bespoke AT themselves, in-situ, without needing to wait for a commercial tool to address their challenges, or for their environment to become more accessible. However, the extent to which such tools enable the creation of useful custom-camera based AT for BLV users remains unknown. Since BLV users often cannot confirm visual results, the obfuscation of how tools actually operate makes identifying tools appropriate for a desired workflow exceedingly challenging [99].

To probe into the emerging opportunities and challenges of adopting agentic programming for creating bespoke visual assistance, we seek to answer three guiding research questions:

RQ1: If the barriers to creating bespoke visual assistive technologies were lessened, what would BLV people be interested in creating? By offering a paradigm that widens the range of possibility for what can be created, we aim to achieve a clearer sense for what challenges and desires exist in the long tail of user needs for camera-based AT. In answering this question, we aim to provide insights both into pervasive unmet needs within the BLV community and into the diverse design space required to support more niche, individual desires. **RQ2: Why do people want to create bespoke, camera-based assistive technology?** Beyond the specific tools that BLV users may choose to build, we examine the motivations underlying bespoke creation in the camera-based AT domain—including the workflows it enables, and the affordances it provides beyond the resulting tools themselves. By addressing this question, we aim to better understand if and how the value of “bespoke” in assistive technology contexts extends beyond the commonly cited potential to address individualized challenges. **RQ3: What structures and supports are necessary for end-users to realize the tools they envision using agentic programming?** Agentic programming positions itself as offering a wider range of creation possibilities than previous approaches to end-user programming. Yet, AI, both in code authoring contexts and when used in the actual functionality of tools, is imperfect—especially in response to ambiguous natural language. We provide insights into the creation practices people adopt to navigate this precarity, as well as present potential design supports to better empower bespoke creation.

To investigate these questions, we design ProgramAT, an agentic programming study instrument that enables BLV users to author bespoke, camera-based assistive technologies by specifying behavior in natural language and iteratively refining executable artifacts, with code and associated edits performed by GitHub Copilot (Figure 1). ProgramAT supports the integration of vision models, task-specific logic, and contextual prompts, allowing users to create, test,

¹In this work, we use a mixture of identity-first and person-first disability terminology, to reflect the diversity of disability identity and language preferences across sub-communities.[90]. We use identity-first language to refer to the BLV community, as our co-designers prefer.

and adapt custom tools in situ. We use ProgramAT to approach our research questions through a two-stage co-design process with tech-savvy BLV end-users (two in phase I, five in phase II) over two months. In Phase I, two participants engaged in exploratory authoring to provide preliminary insights and inform system refinement to best support intended use cases. In Phase II, five participants created and iterated on multiple custom AT artifacts over time. Across both phases, participants created a total of 37 camera-based AT prototypes, through a total of 68 iterations. Throughout this process, participants also made per-tool diary entries and participated in weekly check-ins, providing longitudinal insight into the authoring practices and challenges, as well as situations where AI was more likely to succeed or fail.

Overall, this work makes three primary contributions:

- (1) An open source study instrument, ProgramAT², that enables the creation, iteration, and testing of bespoke camera-based assistive tools through agentic programming;
- (2) Understanding of the types of tools that BLV people may want to create, contextualized with their motivations and strategies to creating these tools; and
- (3) Recommendations to design agentic coding environments to support conversational programming of AT.

2 Related work

While commercially available tools are widely used, they leave several gaps, particularly with respect to nuanced, individual use cases. We make these gaps explicit and discuss open problems in the area in AI in visual assistive technology (Section 2.1). Addressing such gaps in accessible, personalized assistive technology demands approaches that simultaneously support customization, expand participation in creation, and foreground disabled perspectives. We position our work within this intersection, drawing on research in bespoke assistive technology (Section 2.2), end-user programming (Section 2.3), and disability-led design (Section 2.4).

2.1 AI-powered Visual Assistive Technology

Commercially available AI-powered AT (e.g. SeeingAI [74], BeMyAI [31], OKO [5]) receive widespread use within the BLV community; however, recent research documents that for many complex or individual needs, these tools are insufficient [46].

One area of insufficiency surrounds tools that require multiple steps: often requiring frequent tool switches to achieve a multi-step goal [46]. Commercial AI tools, even when successful in supporting one step of a user’s goal, routinely fail to support them throughout the course of the interaction [109]: failing to suggest or consider proactive next steps in favor of treating only the moment at hand. Further, across multi-step interactions, assistive AI regularly experiences task-drift—reducing its guidance’s efficacy and increasing cognitive load [116].

Another major area in which commercially available AI-AT faces challenges is in tasks related to personal interests. Without knowing a user’s personal standards or expectations, generalist tools cannot provide reliable, adequate, contextualized guidance. This leads to answers that are inconsistent on a single input (e.g. reporting if “enough” of a makeup product has been applied [66]) or too

general for use in a precise context (e.g. describing the piece needed to complete a puzzle section only as “square” [109]). This failure is not unique to assistive AI: general purpose AI, like ChatGPT’s voice mode, also tends to fall back on general, non-situationally grounded knowledge when asked about niche or individual scenarios [19]. Further, in contexts where immersion and interpretation, rather than task-completion, are the aim, the goal-driven model of assistance falls short. This challenge presents barriers to using commercial AI AT in situations where there is not one “correct” answer (e.g. art interpretation [68]), or the desired answer is informed by the user’s own perspective and context (e.g. taking in one’s surroundings while exploring nature [97]).

A third area where commercial AI AT historically struggles is in cases requiring individual context, such as finding personal objects [54, 64]. Recent commercial AI AT has made progress in this area: Find My Things [106] has recently been integrated as a SeeingAI mode [74], empowering users to leverage generalist AT towards individual purposes in some cases.

However, advances to personalizing widespread AI-based AT come with new privacy and security risks. Users have little control over what models are used and what data is received and retained in commercial tools. Prior work surfaces an expectation of limited data retention that by tools that require permanent upload of data to achieve personalization cannot meet [94], and highlights anxieties around using AI AT in private settings for fear of capturing sensitive content [6, 91]. Recent work supports obfuscating private information to lessen these concerns [114]. Yet, BLV users cannot readily verify obfuscation results [7]: limiting obfuscation alone as risk-mitigation strategy in privacy-aware AI AT.

These concerns are relevant to any use of AI in visual assistance, not just generalist, commercially available tools. Still, one major design guideline for privacy-protecting GenAI use, as proposed in Sharma *et al.* [91], is explicit user consent when data processing occurs over-network or off-device. While many AT may require the use of off-device models, users can more readily give informed consent when the models in use are 1) transparent and 2) substitutable such that if a user does not consent to a particular model or service, they can change the model rather than disqualifying use of the tool as a whole.

2.2 Bespoke Assistive Technology

In response to the barriers presented by general-purpose AT, disabled users—both within and beyond the BLV community—often attempt to combine [46] and augment existing tools [45, 103], or develop narrowly-scoped custom tools [47]. Still, these solutions are labor intensive, and as a product of being retrofit rather than a native solution, often still have gaps that prevent fully addressing accessibility needs [108]. The usability problems that follow lead to many being stuck with AT that is inadequate for their needs, or having a long list of ATs that have been tried, most of them abandoned [51]. Thus, many end-users prefer bespoke AT [30]: expert-built tools tailored to their needs. However, making such systems demands close collaboration with designers, rendering everyday customization impractical.

²ProgramAT is open-sourced at: github.com/program-at/ProgramAT-opensource

Supports to build bespoke AT vary in the types of creation and customizations that are supported. Historically, bespoke AT manifests in the physical domain [16, 43], driven by the maker movement [93]. Clinical professionals [3, 88, 92] and disabled end-users [3, 17, 100] alike have embraced digital fabrication as a means to produce custom assistive devices [16, 89].

Another, emergent, line of bespoke AT research leverages generative AI to enable custom accessibility experiences across a range of applications [58]. One application is augmentation, where GenAI supports goals beyond basic access for users of existing AT such as enabling social integration through humor and personal voice [86, 103, 104] in use of AACs—tools that support non-speech communication. Other applications enable the creation of entirely new experiences, like natively nonvisual online shopping for BLV users [96] and rapid prototyping interfaces that allow occupational therapists to build tools that meet diverse access needs [69]. Generative AI is uniquely powerful in the context of bespoke AT, as it has rapidly expanded the range of assistive tasks that are perceived as plausible to address with technology [21]. In this work, we leverage this broadened possibility space to gain greater insights into the true interests BLV users have in creating bespoke AT.

2.3 End-User Programming

In the software domain, the majority of bespoke AT creation falls into one of two categories. The first is hyper-personalization, generally done by very technically savvy disabled people by standard development methods [52, 103]. The other—which this work focuses on—is end-user programming, defined by Ko *et al.* as programming done by people who may or may not have technical expertise in order “to support some goal in their own domains of expertise”, further, “to achieve the result of a program primarily for personal, rather [than] public use” [57]. Though used by people of varied levels of programming skill, end user programming generally tries to streamline the programming experience [107] to lower the floor (reduce barriers to entry [85]), raise the ceiling (expand what is possible [72]), or both. Our work deepens understanding along this line of work, aiming to both lower the floor and raise the ceiling by making end-user programming more accessible while expanding the range and complexity of assistive tools that BLV users can create, relative to prior work [47].

End-user programming for accessibility takes many forms [44]. Examples include simple block based programming for accessible classrooms [83], a templated domain-specific language for finding specific visual information [47], leveraging on-device automations to surface novel, relevant functionality [45], and customizable AI interfaces used by people with disabilities [63] and medical practitioners [113] alike for just in time adaptive communication. The advent of agentic programming opens up new possibilities for conversational programming, a type of end-user programming. Conversational programming predates LLMs [20], yet, the prevalence of AI coding agents, like Copilot [38], Cursor [10], and Claude Code [9], reveal the possibility for this natural language form of programming to be usable by the general populace [4]. These agents have already been used in several accessibility contexts: from web development [77] to sound-accessible AR [62].

Though code agents have begun to be applied in accessibility contexts, it is still not well understood how they support end-to-end creation of bespoke assistive tools in practice. This work addresses this gap by examining how technically experienced BLV users create camera based assistive tools with agentic programming systems, focusing on what they create, how they navigate the development process, and where their processes succeed or breaks down.

2.4 Disability-Led Design

The existence of conversational programming as an approach to authoring bespoke AT does not, by itself, make it a suitable solution for addressing long-tail accessibility needs. Indeed, recent work indicates that the types of problems visual accessibility research often tackles are misaligned with the problems the BLV community places value in solving [34].

Disabled users are not waiting idle for research to meet their needs, but often trailblaze innovation based in lived experience in their own right [12, 27]. By including voices of these innovators explicitly in the research process and allowing lived experience to drive design decisions, the tacit knowledge of lived experience can be translated into material scientific knowledge.

Such an approach is critical in accessibility contexts, where needs are shaped not just by functional requirements, but by community-specific practices. Prior work shows that disabled communities often develop distinct cultural norms and interaction practices, such as differences in communication styles among AAC users [105] and linguistic and interactional preferences within the DHH community [56]. At the same time, accessibility needs are highly context-dependent, shifting across environments and social situations [49, 53]. Questions of discretion and self-preservation further shape how and when assistive technologies are used [24]. Co-design helps surface these cultural and contextual factors, supporting the development of assistive tools that better align with users’ lived experiences, priorities, and forms of self-representation [81].

To simultaneously address the need for community participation and faithfully answer the target research questions, we engage in the established strategy of involving working prototypes in participatory design [42]. This practice is standard in technically driven co-design in accessibility [45, 48, 61, 110] and HCI more broadly [40, 79, 102]. We choose co-design here because the lead author is not part of the BLV community: this method ensures the design is driven by community perspectives.

3 Study Instrument: ProgramAT

To support answering our research questions, we developed ProgramAT: a study instrument that allows users to engage in natural-language creation, refinement, and use of bespoke camera-based assistive tools. It consists of a mobile app, an agentic coding environment (GitHub Copilot retrofitted with scaffolds and tools), and a compute server. With ProgramAT, users can prompt for the creation of a desired tool using natural language—a format demonstrated to be highly expressive [59]. Created tools use the camera as input and output speech, audio tones, haptics, or some combination thereof at the user’s specification. ProgramAT differs from existing AT apps (e.g. BeMyAI) through its ability to generate, refine, and execute task-specific code rather than relying on predefined functionality

or thin LLM calls. These functionalities broaden the opportunities for what bespoke AT could accomplish, bolstering our ability to answer the question of what and how BLV people want to create.

3.1 Usage Walkthrough

We demonstrate the ProgramAT’s creation and iteration capabilities through a walkthrough of creating an Uber finder tool (a use case for an unmet need expressed by our co-designers) (shown in Figure 2). To create such a tool, a user starts by prompting for the tool, either through the mobile app using natural language or through the GitHub site, using the ‘visual assistive technology’ template. If using natural language, the user’s prompt is mapped to the fields of the template, and if prompting from the web into the template, the user’s prompt is used directly. This request is then assigned to GitHub Copilot, a coding agent that proceeds to implement the user’s requested tool, following the guidelines described in Appendix A. Once generation is complete, the user runs the tool from the mobile app, pointing the camera at the street and receiving real-time feedback about the presence of cars and potential Ubers. If they want to make any changes to the tool, for instance, reducing the verbosity of responses, they may do so in the mobile app by choosing to update issue, or on the GitHub site by leaving a comment on the pull request and tagging Copilot. This prompts a code update, which the user can resume testing once implemented. Once satisfied with the tool, the user visits the GitHub site to merge the pull request—transitioning the tool from development to production. On production, the tool is runnable for everyday use and its components can inform the implementation of future tools.

3.2 Key Interaction Components

3.2.1 Modular building blocks. We instruct Copilot to generate code in a modular, building-block style, where each tool is composed of reusable components that encapsulate specific functionality. Rather than directly modifying existing tools, ProgramAT instructs Copilot to compose existing tools’ components and augment them with new code only when none yet exists to satisfy a particular purpose, enabling tools to be developed incrementally without introducing dependencies on prior implementations [111]. Individual tools remain self-contained while still integrating with shared system infrastructure. As a result, users can build increasingly complex tools by composing modular building blocks across successive pull requests.

3.2.2 Intelligent model selection. ProgramAT selects models based on task type rather than applying a single model uniformly. Object detection defaults to YOLO11 (COCO dataset), with YOLOWorld used for objects outside COCO’s class set. Text extraction uses the Google Cloud Vision API. General reasoning and vision-language tasks use Gemini models. When a user requests a Custom-GPT-esque [78] interaction, ProgramAT uses Gemini Live and automatically reissues queries at regular intervals, overcoming the cumbersome reprompting present in standard video-AI interactions [19].

3.2.3 Mobile and desktop integration. ProgramAT’s agentic coding environment can be accessed from the mobile app, or directly from the GitHub web desktop interface, where users can create tools by creating standard GitHub issues [37] and iterate by commenting

on pull requests. An example of the tool creation process from the web is shown in Appendix B, Figure 4. This flexibility can support a variety of creation workflows.

3.2.4 Iterative workflow. After testing a tool, users can iterate by submitting a natural-language update request from the app or leaving a comment on the GitHub pull request on the web. Copilot applies changes under the same architectural constraints as the original generation, and users can re-run and re-evaluate tools as soon as re-generation concludes, supporting a tight loop between real-world use and tool refinement.

3.2.5 Development and production modes. ProgramAT distinguishes between *Development Mode*—where users create and refine tools via pull requests—and *Production Mode*, where finalized, merged tools are available for streamlined use. This separation lets users freely experiment without disrupting tools they already rely on.

3.3 User Interface

The ProgramAT mobile app is organized around two primary views corresponding to its two modes. In *Development Mode*, the *PRs* tab lists all tools currently under development. Each entry displays the tool’s name and current status (e.g., “WIP” while the agent is generating code). Users can tap a pull request to select from tools on the associated branch, submit an iteration request, or view associated logs. Selecting a tool on the branch launches it, transitioning the user into a camera interface where the tool can be run. This camera interface supports two interaction modes. In *Stream* interactions, the tool continuously processes live camera frames and delivers outputs—spoken aloud, as haptic feedback, or as on-screen text—as the scene evolves. This is suited for tasks requiring continuous awareness, such as tracking moving objects or monitoring a changing environment. If the tool is built to work similarly to a Custom GPT—a customized version of an off-the-shelf LLM designed to handle a specific task or query [78]—, this mode also allows the user to ask live follow up questions as the tool runs, a *Live* interaction style within the standard *Stream* way of running tools. By contrast, in *Take Photo* interactions, the user captures a single image and receives a response tied to that moment, better suited to one-off queries. After a photo capture, users can ask natural-language follow-up questions; the visual context is retained, so users can progressively build understanding without recapturing input. In all modes, images are discarded from the server post-processing to protect users’ privacy.

3.4 System Implementation

ProgramAT is implemented in React Native and deployed via iOS TestFlight. Each participant is assigned a personal GitHub repository forked from a shared base, providing an isolated development environment. Tool creation is mediated by a GitHub issue template tailored for camera-based assistive technology, which structures proposals around problem descriptions, example usage, and implementation details—reducing ambiguity and ensuring agent-generated code remains grounded in the user’s professed needs. When a user submits a prompt from within the app, it is first parsed by Gemini-3-Flash-Preview to populate the template fields; if any required fields cannot be inferred, the app prompts the user for

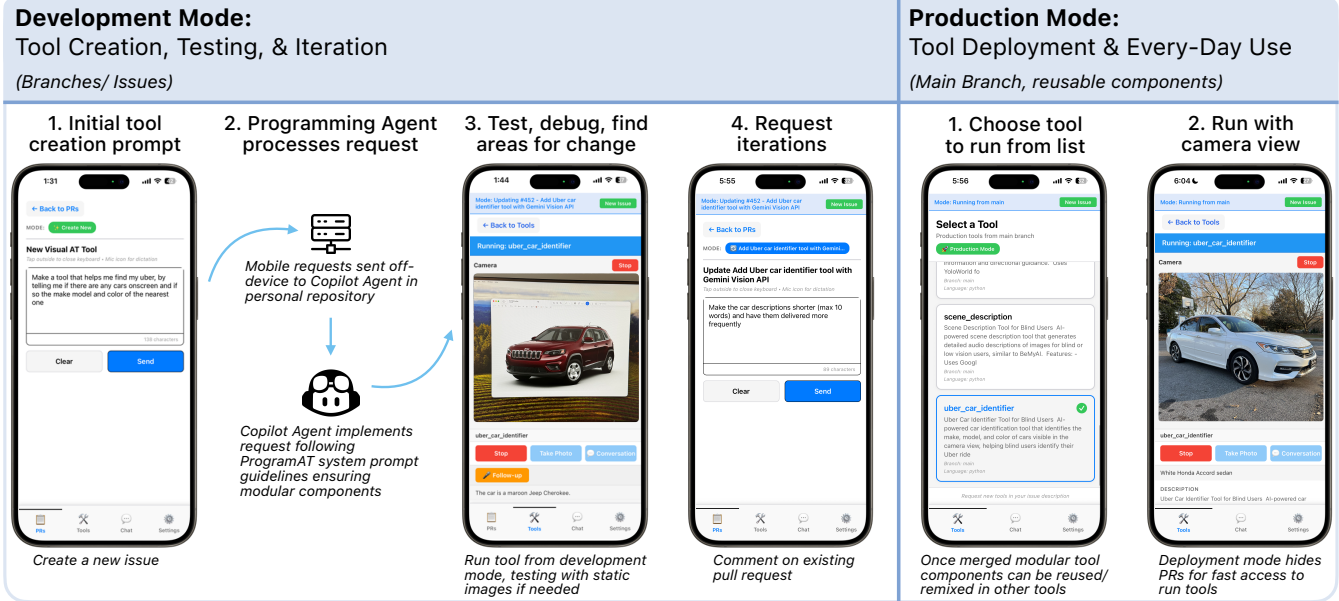
ProgramAT (Mobile App) AT Creation Workflow:


Figure 2: A walkthrough of how users create and use AT tools through the ProgramAT mobile app interface. In deployment mode, users create, test, and iterate on tools. (1) A request for a new tool is made, opening a GitHub Issue. (2) The programming agent processes the issue, following ProgramAT structure to create modular tools that run in a mobile camera-based environment. (3) Users test the created tool using the camera, observing issues. (4) Users request iterations to the tool by commenting on the existing pull request. Then, in production mode, users access a simplified interface for every day use, hiding pull requests and incomplete tools. They (1) choose the tool they want to run from a list, and (2) run it from the camera view.

clarification before submitting. This structured handoff reduces underspecified requests reaching the coding agent.

The coding agent (GitHub Copilot) operates under a detailed instruction file (see Section A) that enforces the modular architecture and model selection logic described above, as well as a set of starter tools. All generated tools conform to a standardized JSON output schema, which the mobile client parses to route outputs to the appropriate feedback channel—speech, earcons, haptics, or on-screen text. This shared schema makes multimodal output possible without tool-specific client logic.

A backend server, written in Python, coordinates agent sessions, manages GitHub API calls, handles tool execution by forwarding camera frames to the appropriate model endpoint, and returns structured results to the app. It also surfaces lightweight agent progress updates to the mobile client via WebSocket.

4 Co-Design Study Method

Our study’s objectives are two-fold: first, to understand *what* visual assistive tools people who are blind or low-vision build, and second, *how* BLV users may approach building these tools. We utilize co-design—defined in Ellis *et al.* as “a subset of participatory design which is used to understand individual usage scenarios” [30]—to explore these objectives. The study involved five tech-savvy co-designers (two in phase I and three additional co-designers in phase II) building custom visual tools with ProgramAT, making diary entries, and participating in semi-structured interviews over a span

of two months. Phase I ($n=2$) was a smaller, exploratory study aimed at generating preliminary insights into what BLV users wanted to create and how they approached creation, while also informing iterative refinements to ProgramAT to help meet these goals. Phase II built on the refinements and initial findings as a larger study ($n=5$) conducted with a more mature, stable version of ProgramAT, enabling a deeper examination of users’ creation intent and the extent to which AI can meaningfully support the development of bespoke camera-based AT. Each phase lasted approximately one month and consisted of a kick-off meeting, three weekly check ins, and a wrap-up meeting, with each co-designer building and iterating on at least four tools with ProgramAT throughout the phase. By the end of phase II, our co-designers built a total of 37 tools, with 68 total iterations.

To answer our research questions, we analyze qualitative data from interviews and diary entries, and conduct AI-assisted code reviews on the code for these tools.

4.1 Co-Designers

Across both phases, we engaged with five co-designers, all of whom are authors on this paper. Two co-designers (D1, D2) were involved in both phases, while the remaining three co-designers (D3, D4, D5) engaged with phase II only. General demographic information for all co-designers is described in Appendix E, Table 4.

Our use of the term “co-designer” refers to 1) participants’ collaboration with the research team in shaping ProgramAT (the primary

Research Questions:

1. **What bespoke AT** would BLV people create?
2. **Why** do people want to create bespoke AT?
3. **What tools/ supports** are necessary to fully realize goals?

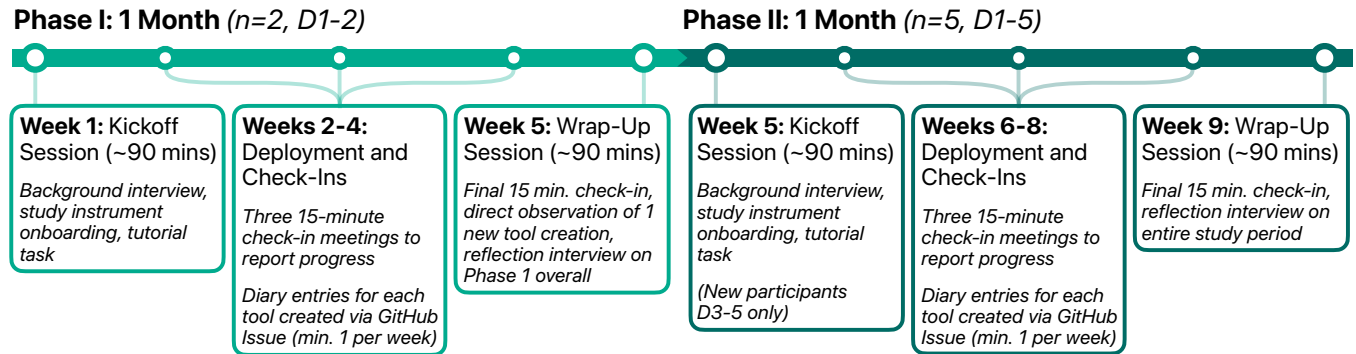


Figure 3: Timeline of the study methods. Phase I: A preliminary deployment with two co-designers to refine the study instrument. Phase II: A deployment of the final study instrument with five co-designers for further data collection. Each phase included a kick-off interview with a system tutorial, a four-week deployment with three 15-minute progress check-in meetings and diary entries, and a wrap-up reflection interview.

focus of Phase I, where the individual usage scenario is the creation process), and 2) use of ProgramAT to make custom tools (the primary focus of Phase II, where the individual usage scenario is the visual access need a tool addresses). [30] This relationship is both human-human and human-AI.

Our participant selection strategy leverages the strong influence of technical experience beyond AI literacy alone on effective AI use while being grounded in the lived experiences of people who may need visual assistive tools, enabling exploration of the upper bounds of AI capabilities to create bespoke visual AT [71, 112]. Essentially, working with technical users helps distinguish fundamental AI limitations from issues of inexperience [71] so we adopt the established practice of recruiting experts when relevant to the research questions [45, 55]. We recruited BLV co-designers who were tech-savvy, in particular with regards to AI and basic development experience. Technical experience was materially necessary for this study, since effective ProgramAT use requires comfort with development flows (e.g. Git). Each co-designer’s self-reported fluency and confidence across several relevant technical areas is described in Table 1.

Our focus on tech-savvy BLV co-designers reflects observations that users with disabilities often also represent early adopters of AI based technology [12]. D1 and D2, both graduate students in computing and information sciences, were recruited for the first phase of our co-design study. Their overlapping experiences as researchers and visual assistive technology users yielded valuable *cripepistemologies*, defined in Desai *et al.* as the way “innovators” lived experiences and the corresponding crip knowledge shape both the high-level idea behind their innovations and the smallest of design choices” [27]. Their *cripepistemologies* provided simultaneous familiarity with early prototypes and intuition for the types of AT affordances that best support BLV creation and agency: a relevant combination of experiences for phase I’s goal of informing system iteration. For phase II, we added three additional co-designers, with similar technical experience but working in non-research careers

to improve population coverage. We recruited these co-designers through personal networks and snowball sampling.

4.2 Semi-Structured Interviews

Across both phases, our semi-structured interviews onboarded co-designers, and gathered insights when wrapping up phases of the study. These interviews approximately lasted between 90 minutes to 2 hours, were audio-video recorded, and transcribed using the automated transcription features of the videoconferencing platform. The lead researcher memoed after each interview, and corrected transcription errors by hand during analysis. Protocols are available in Supplementary Material.

The onboarding interview captured our co-designers’ technical expertise, along with details about their use of AI for accessibility. We then introduced ProgramAT by asking our co-designers to build an empty seat finder and clothing recognizer—pre-determined tools that address overlooked needs [34, 73, 95] and ensure consistent tool behavior. Co-designers then familiarized themselves with ProgramAT by creating a camera-based tool of their choosing, testing and iterating on it.

The phase I wrap-up interview asked co-designers to reflect on the previous four weeks, with questions capturing where they had found success and friction so far, and how those experiences compared to those they had with pre-built AI AT [5, 31, 74]. This culminated with co-designers creating a tool of their choosing to facilitate observation of their creation and iteration strategies. Finally, the phase II wrap-up interview reflected on these strategies longitudinally, and highlighted co-designers’ perspectives on the future of creating and using bespoke AT.

4.3 Weekly Check-ins and Diary Entries

After the onboarding interview, co-designers incorporated ProgramAT into their daily lives, using it to create tools for their own

ID	Profession	General Tech Literacy	Off-the-Shelf AI	Development Flows (ex: Git)	Code Agents
D1	CS PhD student	7	6	5	5
D2	Informatics PhD student	6	5	6	5
D3	AT Instructor	7	7	4	5
D4	Accessibility Engineer	6	7	5	6
D5	Software Engineer	7	7	5	6

Table 1: Co-designer experience and self-reported technical skills (1-7), where 1 is totally novice and 7 is totally expert.

purposes, with the expectation of producing at least four tools over the four-week period of each phase. To understand their motivation and approach to tool creation, we conducted weekly 15-minute check-ins where we asked a standardized set of questions, exploring what participants had built or iterated on, what was working, what was not, and—particularly for Phase I—how ProgramAT could better support their goals.

Check-ins provided insight into both the barriers co-designers faced in creating useful, performant tools and the approaches that led to success. They also supported collaborative ideation with the research team and iteration: giving co-designers an opportunity to better understand how ProgramAT operates in order to guide iteration and future creation, while offering the research team clearer visibility into key pain points and how they might be addressed.

In addition to weekly check-ins, when a co-designer finishes work on a tool—whether they are satisfied to call it complete and merge it to main, or they have devoted substantial effort but remain unsatisfied and elect to abandon it—they “close” the associated pull request by leaving a diary entry as a comment on the PR. This comment describes their idea and prompting strategy, any iterative steps taken after the initial attempt, unexpected system behavior (positive or negative), and how the tool performed. Notably, to the authors’ knowledge, using GitHub comments as diary entries may offer a novel and promising method for coding-based qualitative studies; we will explore this further in future work.

Where check-in meetings provide in-the-moment insights into tool performance and the co-designer’s creation process, diary entries offer a post-hoc reflection on the tool-building experience. This reflection enables clearer distinction between barriers that were surmountable through iteration and those that persisted as salient long-term challenges, supporting a more precise thematic understanding of limitations and opportunities in tool creation.

4.4 Data Collection and Analysis

Through the two-phase study, we collected interview data, diary entries, and development activity including original prompts, pull request comments that informed iterations, and the code for each tool that our co-designers built.

The onboarding, phase two kickoff, and wrap up interview data was analyzed by two authors generally following the six phases of thematic analysis [14]. We used thematic analysis as it best fits our research questions; it is not only useful for understanding themes in participants’ behaviors and perspectives [15], but also widely recognized as fitting for applying qualitative insights across multiple forms of data [26, 29, 33]. Two members of the research team (a lead and supporting author) performed the analysis. The supporting author performed the first three phases of analysis on a

portion of the 90-minute sessions, generating memos [13], initial codes, and potential themes. The lead author compared this with their own analysis of the full data to ensure agreement and coverage. The lead author completed the analysis, with the supporting author reviewing the final themes and report. Additionally, the lead author coded the 15-minute check-in data in conjunction with the final tool diary entries.

These were analyzed in tandem because of their overlapping purpose: weekly check-ins were intended to provide insight into the intention and experiences behind building tools as they were being built, whereas a diary entry collects the same information post-hoc, providing a higher-level overall view. By considering these pieces together, we gain a more comprehensive view of the creation experience.

Notably, we also conduct a post-hoc code review, assessing the code created at each step, for each tool, by the end of the study. Following the standard code review practice of tailoring to organization-specific requirements [11], we develop spec-driven code review criteria [82]. To establish specs, we reference the agent instruction file (available in Appendix A), the instructions from the co-designer’s initial prompt, and, in the case of iterations, any additional or changed instructions provided by co-designers in follow ups. Follow up instructions are included alongside co-designer’s initial requests because conversational programming often operates under *progressive specification* [32, 98], a practice where developers iteratively refine outputs rather than specifying complete tasks upfront [98]. The resulting review focuses on whether generated code satisfies the provided specs.

Though agentic programming tools support AI code reviews, there are often gaps in their understanding of user’s intent [41] which lead to poor comprehension of code’s meaning in context [22]. However, AI’s code-review performance improves when provided example human reviews [70]. Thus, we adopt a human-AI approach to our code reviews. The lead author manually reviewed the commits associated with the first Copilot session for each tool as a group—representing Copilot’s first attempt at meeting specs—and provide this review along with the earlier established spec-driven guidance to facilitate the AI code review for a tool’s remaining Copilot sessions. To account for model diversity, we use Claude Sonnet 4.6 for these AI code reviews: Claude is not used by default in Copilot’s generation, and our tools use Gemini when a vision-language model is needed.

Finally, we analyze both initial and iterative prompts alongside the results of code reviews to identify patterns underlying AI successes and failures in tool creation, with the goal of informing future structural supports that better enable BLV end users to author bespoke camera-based assistive technologies. We consider several

metrics of “prompt quality” as established in Torka *et al.* [101], including general features such as length, presence of example usage, provision of situational context, specification of technical detail, and, in the case of iteration prompts, distance from previous output to desired new output. Further, ProgramAT offered both mobile and desktop methods of input; we also consider which input modality was used, in order to judge the tradeoffs between the expressiveness mobile’s pure natural language interface is theorized to offer [59] and the precision afforded by a more structured, templated form of input offered in the desktop experience [1].

5 Findings

Our research questions seek to understand what BLV people would create if barriers to bespoke visual AT were lessened, why they are motivated to create bespoke camera-based AT, and what supports are necessary to realize those tools through agentic programming. To address these questions, we conducted co-design activities with five BLV co-designers who built tools using ProgramAT over the course of two months. We highlight findings from these activities: we first present the range of tools co-designers envisioned and built—spanning common accessibility needs to highly individualized desires—and examine the motivations underlying their creation choices (RQ1, RQ2); we then turn to the creation process itself, surfacing the strategies co-designers developed to navigate the inherent ambiguity of agentic programming and the design supports that helped them realize their goals (RQ3).

5.1 What Motivates Tool Creation?

Co-designers build a broad range of bespoke assistive tools, initially prioritizing everyday accessibility needs before progressively exploring more personal, speculative applications as they became comfortable with ProgramAT.

5.1.1 What tools did co-designers create? Our five co-designers built 37 tools in total (see Table 2) to receive visual assistance with navigation, retrieve information from physical objects, and for social factors including self-presentation and to understand physical social cues. We asked our co-designers for a specific number of tools (at least four per phase), but did not constrain what they built beyond being camera based. Co-designers had many ideas they were eager to try, often exceeding what time allowed for: making a total of 37 tools, exceeding the minimum expectation of 28 tools. We present their motivations for building these tools, and discuss their own considerations to assess tools’ success.

Some tools correspond to common, well documented use cases where the motivation was straightforward: they assisted with an unavoidable task, or there is either no pre-existing, efficient, commercially available AT for the task. For example, tools for outdoor navigation included tools for solving well-studied accessibility problems of finding bus-stops and room numbers, and for specific needs to follow a straight path when walking. Tools for information access followed similar motivations: two co-designers built tools to extract information from business cards—a ubiquitous part of networking. For these types of tasks, even while building bespoke tools, co-designers often aimed to solve similar problems. All five co-designers expressed an interest to build tools to find an Uber, and two co-designers attempted to build this.

Other cases were more niche: examples include tasks already addressed by mainstream AT (e.g., D3’s light detector, which has an analogous SeeingAI mode), tools related to longer-term social understanding over immediate use (e.g. D1’s hand gesture interpreter) and tools for tasks that are nominally already accessible (e.g. D3’s Braille reader). These tools often share the motivation of lacking a pre-existing AT to address it, but are more diverse in their alternate motivations. D3’s tool, for example, were often motivated by needs outside his own: while SeeingAI already has a light detector, it uses tones only, which is adequate for him but unusable for someone who is deafblind, prompting D3 to make one that offered spoken descriptions as well because those could be rendered by a Braille display. Similarly, though D3 reads Braille fluently, he was motivated to build a Braille reader to support Braille learners, or even sighted people navigating the estate of a blind loved one.

5.1.2 How do tools of interest change over time? Co-designers’ creation trajectories began with highly pragmatic, need-driven tools, but diversified over time as they became more comfortable with ProgramAT and its capabilities. Early tools tended to focus on immediate, functional challenges reflecting a prioritization of utilitarian value. Examples of first tools include a tool to locate cooking ingredients made by D2 and a tool to locate commonplace physical items made by D4. However, throughout the study’s duration, the tools co-designers were drawn to became more varied, expanding into contexts of entertainment—as found in D4’s final tool, a chess move validator—or even proof-of-concepts for AI workflows that were not themselves AT. D2’s second-to-last tool tested a pipeline for generating an image description that a ProgramAT tool could then use as a personal object recognizer. D1 explains this evolution arises from exhausting some more basic use cases, saying: “*you know, most of those basic things, like, the more fundamental things, I feel like I’ve gotten a good grip on in my life. So, like, what are other, like, areas where I feel like I might not necessarily need AT, but my life could potentially be enhanced by AT, which ended up being a lot of ... info that I didn’t necessarily ‘need-need’, but could benefit from.*” As time went on, and the floor of needs could be addressed, co-designers felt comfortable to explore niche, nice-to-have use cases for which they may not have even considered AT in the past.

5.2 How Are Tools Built?

There was no single path to successfully building bespoke assistive tools: co-designers found success through different prompting and testing strategies, including both carefully engineered one-shot prompts and progressive iteration. Across these approaches, failures were largely explainable, arising from technical limitations, AI capabilities, or misalignment between specifications and implementation.

5.2.1 Characterizing prompt strategies. All five co-designers indicated that their strategies for approaching the initial creation of tools, as well as their iterations influenced the tool’s ultimate success. Yet, all indicated different strategies that corresponded to “best” results, according to each co-designer’s definition of success. D5 found the most success when treating her prompts for tool creation as prompts to the code agent directly rather than requests for a given tool, generally being explicit about technical details

Tool	Creator	Motivation	Alt. Approach	Mode	App/Web
Car identifier	D2 *, D4	Support more reliably finding Uber	Calling Aira or a sighted friend	Live	App
Mail sorter	D5	Distinguish between relevant and junk mail	Take repeated photos with BeMyAI	Live	Web
Hand gesture interpreter	D1	Better understand how hand gestures are used in practice	Call AIRA	Streaming	App
Playing card reader	D1	Understand a changing hand of playing cards	Use a tactile set of cards	Live	App
Braille translator	D3	Support non-Braille readers in interpreting Braille content	N/A, he personally already reads Braille	Streaming	App
Sauce and seasoning finder	D2	Properly identify and locate sauces, seasonings in his kitchen	Repeated BeMyAI descriptions	Streaming	App
Business card reader	D5 *, D2	Read only the relevant text on a business card	SeeingAI	Streaming	Web
Chess move validator	D4 *, D1	Validate moves made in a game of chess	Streaming board to sighted friend	Live	App
Vacuum assistance	D1	Determine if a spot is clean or has obstacles in it while vacuuming	Touch spot with hand and feel it	Take photo	App
Light detector	D3	Communicate the level of light in a room verbally and via earcons	For him, SeeingAI. For a deafblind person, no solution	Streaming	App
Bus stop detector	D1	Help identify if there is a bus stop and where	Use maps to get 95% of the way, from there, ask someone	Streaming	App
Scale reader	D2	Read the weight on a kitchen scale in grams	Talking kitchen scale	Streaming	App
Room name identifier	D2	Read aloud the names on room signs	Run hand along wall to read braille room signs	Streaming	App
Straight line walker	D5	Help guide to walk in a straight line	N/A, veers when walking	Streaming	Web
Makeup checker	D5	Check makeup is applied correctly	Use BeMyAI	Take Photo	Web
Clockface food description	D3	Describe where food is on a plate using clockface descriptions	Poke around with fork	Live	App
Waffle plushy finder	D2	Find plush dog (Waffle) if in frame	N/A, exploratory not assistive	Streaming	App

Table 2: A sample of tools created in our study. Table includes a description of the tool created, the co-designers’ alternative or current approach to accomplish the same task (i.e., what strategy they used without the bespoke AT, if any), the mode of the tool (live, streaming, or photo), and how the tool was created (via the app or web interface). If multiple co-designers made the same tool, details correspond to the co-designer marked with asterisk. Tools built but not listed here can be found in Table 3

and desired input and output, and adopting lengthy, highly structured initial prompts: consistent with standard prompt engineering practices in software engineering environments [28]. Through this practice, D5 had multiple tools achieve success with one iteration or fewer: an accomplishment not paralleled by any other strategy she tried. D4, however, found more success through abandoning this approach. Though he started with long and detailed prompts, and even manually selecting the agent’s model, lengthy prompt-crafting and code generation times urged him to try something simpler, and

he ultimately found shorter, and action-oriented prompts to be more successful: “Well, [initially] I did, like, a long prompt, and then also I used, I changed the model from Auto to GPT, and it took forever to [generate the code]... also, even though it took such a long time, it didn’t work too well, and and so, I changed it to auto at the end, and then just gave it a very short prompt on what I actually want, and then it kind of worked after that. So my initial assumption is that long, detailed prompts with a good model will work, but ... that wasn’t the case ..., that short comment, it just started working way better.” He

notes that if this abbreviated approach lacked any specific detail, “*I can keep on reiterating with short comments, rather than making a long prompt with everything I want, and then all the edge cases I could think of.*” This approach aligns with the practice of progressive specification [98].

The preferred approaches of D1 and D2 are similar to that of D4 in their brevity and avoidance of hyper-specification, but motivated differently. For D1, successful tool prompts often started broad, in an interest to see the possibility space of what the models underlying ProgramAT could provide, and using iteration to optimize for where the model was strong. In creating his most successful tool, the hand gesture describer, he described his process as “*initially... somewhat broad... like here’s the motivation, here’s the challenge, help me solve it. And then, after testing it, ... I got a sense of what the descriptions might look like... and the rest of the iterations were latching on to that detail.*” When tailoring his tools to leverage what the models could accomplish rather than fighting a model’s weakness, D1 found tools exhibited less friction.

For D2, by contrast, avoiding hyperspecification was motivated by being selective on what the model prioritized. His prompts early in the study were typically done via dictation in the mobile interface, producing a stream-of-consciousness specification containing all thoughts pertaining to the tool without an implicit sense of prioritization, contributing to a very low success rate for phase I tools (0/5). For his most successful tool—the car finder—, however, D2 instead used a text file to author a detailed, but intentional prompt, which he then pasted to the mobile interface. He described his rationale for doing so as “*I don’t want to follow the template, that’s just too many steps... but I also don’t want to dictate, because this has very specific things that I want to hit, and ... if I’m just talking a lot to the AI in the form of a dictation I might include something that, to me, is just me running my mouth but to [the agent] is a Bible verse.*” Though D2 found the web interface’s templated structure tedious, favoring the expressiveness of pure natural language, being judicious with included information yielded a tool that met his expectations. This result is consistent with literature indicating that, while appropriate context is vital to successful AI interactions, providing too much detail dilutes model focus [36].

For D3, the favored strategy was explicitly tied less to what to include than how to include it: creation often happened quickly via natural language whereas iteration was a more deliberate process that invited closer scrutiny and sustained attention. In effect, prompting from mobile, but iterating on the website to take advantage of the real time logs became his preferred workflow; D3 describes “*I love the mobile because... I was able to get my thoughts down ... fresh. But, like, when I make iterations to it ... I will do them from the GitHub website, so that I can actually monitor what it’s doing, and if I see something that it’s doing that I’m like, maybe it should do something else. I could just send a follow-up prompt.*” Materially, this aligns with the tendency towards progressive specification [98] highlighted by D4 and D1, but also highlights the relevance of input modality in surfacing these strategies.

Across creation strategies, co-designers placed strong emphasis on testing and iteration as core parts of their tool-building process, repeatedly trying tools in real-world contexts and using those experiences to guide refinements. Rather than treating evaluation as a separate phase, co-designers folded it into an ongoing loop—testing,

reflecting, and updating prompts to better align system behavior with their needs. D1 refers to this strategy as “experience reports”—a process by which “*I try to give it specific ... reports ... of what went wrong to try and correct the functionality... just, like, how it failed, basically, ‘hey, like, I tried this, this is what the output was, it’s not what I wanted. I was expecting this, go fix it.’*” Experience reports were a common practice amongst co-designers: efficiently surfacing breakdowns which informed concrete changes in subsequent tool versions. In doing so, participants effectively enacted a full human-centered design cycle, where evaluation and iteration were continuous and user-driven rather than discrete steps [60].

Testing practices were also frequently shaped by the context of the tool itself. In particular, for tools that processed information considered private by co-designers, privacy often informed testing practices. One example of this is D5’s mail sorter tool (described in Table 2). Since mail, junk or otherwise, generally contains sensitive information such as one’s home address, D5 was very particular about what criteria should be used in analysis, and what information should be read to perform such analysis. Her tool prompt made this criteria and scope explicit. While her hope for the tool was eventually that it would help her bulk process mail she had not seen, she first tested it with a few known pieces of mail, to see how it would respond and verify it honored by her criteria. Testing first with known examples was a frequent technique across many tool contexts, but D5 felt it specifically important here because it offered observability; she notes “*I am a security person, so, like, what are the security privacy implications of this... if I’m kind of running something that I don’t fully understand, I’m running an AI authored tool, and it is also sending my picture to some other website behind the scenes? ... That’s where, like, adding more transparency and observability controls really helps, right? Like, anything AI that you build, it’s good to have, like, more observability controls so that your user can see what exactly happens, and can actually build trust with the tool.*” Though testing with known examples is not always possible, nor is it a foolproof answer to questions of observability, it served as a valuable step towards building trust with tools, especially those with clear potential for privacy risks.

5.2.2 Examining contributors to tool success. Tools that stood out most positively in our study did not simply succeed because they “worked.” Instead, they exceeded a baseline expectation of functionality by shifting what was possible or practical in participants’ everyday workflows—either by introducing new capabilities or by dramatically reducing friction in tasks they already perform.

Some standout tools addressed needs that had no accessible or reliable implementation through existing consumer technologies. Participants described these less as incremental improvements and more as qualitatively new forms of access. D1 noted that, while his hand gesture tool was one of several successes, “*what made it different for me was if this tool didn’t work then... I could ask sighted people or call AIRA, but realistically, I’m not gonna do that.*” D2 drew a similar distinction between his scale reader—a successful but replaceable tool—and the Uber car identifier: “*I could just get a talking kitchen scale for 20, 30 bucks, whatever. I can’t get a \$20, \$30 car identifier off of Amazon. And that’s why this tool stands out, because it is so unique.*” In cases like these, even imperfect

performance was often tolerated, as the alternative was having no comparable option.

Other tools derived their value not from novelty but from reducing cumulative friction in the tasks co-designers already performed frequently. D5, for instance, could sort mail using BeMyAI, but doing so in bulk was cumbersome. A custom tool transformed the experience: *“I think [mail sorting is] one of those things which I generally do in bulk...I’m sitting with a whole load of things, and just trying to sort through that, so I feel like success feels even more profound.”* Novelty, in other words, is not a prerequisite for impact.

Across both categories, definitions of success were pragmatic rather than absolute. Tools were considered successful when they were reliably good enough to support a real-world goal, even if imperfect. Reaching that threshold, however, rarely came from a single “correct” prompt. Instead, it emerged through iteration: adjusting specificity, structure, and constraints in response to observed failures. D1 rescoped a general video description tool into one focused on hand gestures, finding a niche where it fit. D5 dug into the codebase to discover that her rationale for what made mail worth opening had never been included in the repeated query because it was omitted from the `custom_gpt` template field.

For co-designers initially resistant to iteration, eventual engagement proved transformative. D2 describes his scale reader as a *“turning point...at which I started to approach toolmaking differently, where I’m like, okay don’t expect perfection from one shot. You need to meet it in the middle.”* The tools he built afterward—the Waffle plushy finder and the car identifier—were rated among his most successful. Working tools, then, reflected not just a final prompt but the process through which it was shaped: experimentation, observation, and incremental refinement.

5.2.3 Examining tool failures. Tool failure generally comes from one of three places: requiring a technical approach ProgramAT does not support, failure of a model a tool utilized, and misalignment between specifications for a tool’s behavior—provided by both the co-designer and by ProgramAT’s structure—and how a tool ultimately performs.

The first case is simple to identify: tools that hinged on a functionality not yet addressed by ProgramAT tended to fail. For instance, D5’s straight line guide relied on the ability to cache a starting photo and use it as a comparison for all future image inputs; without explicit support for persistent memory and a cache, this was unlikely to succeed. Co-designers avoided this pitfall through two primary techniques. The first is referencing the repository’s README and architecture files to get a sense for how ProgramAT works under the hood and adjusting their requests accordingly. D4 took this approach in iterating on his stain finder tool, describing that he *“changed the prompt based on the README and architecture description. So it made it more specific, because I think the prompts before I wrote were saying some, very unnecessary things that might not be actually possible with this”*. The second technique is creating tools in a way that circumvents a technical limitation entirely. For example, since uploading one’s own data is not yet supported, D2 asked ChatGPT to describe a specific object—a plush dog named Waffle—and used that description in a prompt to ProgramAT, building a personal object recognizer without the need for local training that would typically underpin such a tool.

The second case, failure of a model that underpins a tool, proved harder to pinpoint as a source of blame for a tool’s failure since there is no visible line between where the code stops and the models it calls begin in the mobile interface. Co-designers who identified this as a source of tool failure typically did so by trying to accomplish a similar task through other, general-purpose AI platforms like ChatGPT and seeing if the problem persists: if it did, they attribute the source of failure to AI, and if it did not, they attribute failure to some characteristic of the tool itself. For instance, in his door finder, D2 found that even when the tool built with ProgramAT successfully identified whether a door’s hinges were visible, it could not reliably map that to the direction the door opens. D2 found this debugging strategy useful because *“it’s easier for me to...understand the boundaries before I can understand the capabilities...left-right is already a difficulty of AI so I’m not going to fault it.”* D3 had a similar experience while debugging his Braille translator. After the tool hallucinated a Braille translation, he tried ChatGPT and found that it would not even make an attempt, saying *“It refuses. Straight up refuses. ChatGPT will not mess with Braille. It’ll give me braille symbols...but that’s all it will do.”* Though different from D2’s experience, this also serves as evidence that tools’ failure is often a result from asking AI to do a task it is not trained well or at all on.

The final case, misalignment between specifications for tool construction and behavior and the tool’s actual implementation and performance, was often the most subtle: arising not from clear technical limitations or model failures, but from gaps between how co-designers intended a tool to behave and how those intentions were interpreted by ProgramAT. Our code-review analysis reveals that clearly missing or misinterpreted user specifications were somewhat infrequent: the code’s documentation after initial prompting reflected user-provided specifications in all but five tools. Instead, violations of user specifications tended to occur due to *spec conflict*: where two specs for a tool cannot mutually co-exist. One example of this was found in D3’s food finder tool, which aimed to use clock-face directions to describe where on a plate different types of food were located: while the documentation reflected clock-face directions, Copilot’s instruction file in the ProgramAT repository bounds clock-face directions to only values on the upper half of the clock, because in the classic navigation use case, values on the bottom half of the clock are nonsensical output because they are behind the user’s camera. This conflict between user specs and repository level specifications worked to the tools’ detriment.

Spec conflict also manifests in iterations that provide new criteria for a tool with implementation that violates existing specifications. For example, when D4 made his car finder tool, he was happy with its overall output and accuracy but found it slow, prompting him to iterate to improve the tool’s speed while holding the rest constant. Copilot latched on to the speed request, refactoring the tool to utilize lower-latency, lower-generalty models like YOLO over the existing model in use, Gemini. He described the outcome, saying *“After that comment, it got really fast at announcing whether a car is visible but the accuracy got really bad”*, conceding that while Copilot did comply with the new specification provided, it contributed to the tool’s failure by doing so without consideration to the other specifications at play.

5.3 Perceived Value of ProgramAT

Overall, co-designers viewed ProgramAT as more than a development environment: it served as a testbed for experimentation, a catalyst for re-imagining accessibility, and an accessible entry point to creating custom technology.

While co-designers found value in the tools ProgramAT facilitated, this was not the only benefit of ProgramAT. Beyond a development environment, co-designers thought of ProgramAT as a “testbed”, a “thought exercise”, and a “bridge” to creating for oneself.

As a testbed, co-designers framed ProgramAT not just as a way to build custom tools, but as a means of testing what is possible. D2 frames this as *“a sanity checker... Like, okay I have this idea, lets put it into practice immediately, a reality check of sorts, what’s possible and what’s not”*. This perspective persists even when what’s possible does not translate to a fully idealized tool; he notes *“you still have to start somewhere, and ProgramAT is the start.”* Even for co-designers whose sense of value comes primarily from the utility of created tools, the availability of seamless, rapid testing was prized. As D5 puts it, *“testing was the most delightful aspect, how easily I was able to just load a PR”*. The ease of initial creation, and low time investment in creation serves as a low-risk way to test interesting and challenging ideas; as D4 notes *“I mean, it’s just, so, so easy, right? Opening up a computer and, writing an actual program, compared to just, like, pull out the phone and typ[ing] a few sentences”*, which he found de-risked ambitious creation.

In contrast, the framing of ProgramAT as thought exercise is reflective—rooted in the ability to positively disrupt habit. D1 notes that, prior to becoming involved in the project, he was not always thinking about places AT would fit, because he already had workarounds in place: he notes that *“a lot of blind people, including myself, end up getting very used to the way that we live currently, especially if we’ve, you know, figured out non-visual workflows for things. So it’s... hard to kind of be like, oh, imagine a world where you could make improvements to your life or to your workflow.”* The study offered an opportunity for D1 to *“exercise [his] imagination”* and explore *“what if, like, there was a different approach to this, or what areas in my life could I maybe try to make things even more efficient?”* Notably, this application had value even when technology was not ultimately the answer, as occurred while making a vacuum assistance tool, which he *“ended up not needing to use... doing this my own tactile way was more efficient than aiming the camera, waiting for the description, etc.”* There was value in the exploration and probing the what-if, as he puts it *“the more valuable part is to start getting out of the ... I don’t want to call it subpar, but I’m so used to the techniques that I’m using right now that it’s hard to, like brainstorm or imagine.”*

For others, the added value was not just in the freedom to imagine, but the freedom to do. While all co-designers were recruited for their technical experience, not all are formally trained programmers, leading some, like D3, to view ProgramAT as a bridge to creating for oneself. D3, not a programmer by trade, noted that ProgramAT offered support for the entry into customizing code to meet their own access needs, in particular because the underlying structure was already established, saying *“especially for somebody who is new to coding, if they were working with this app, getting*

experience in developing their own kind of setup with a mainframe that’s already there.”

5.4 Supports and Structures to Create Bespoke Visual AT

Overall, while co-designers saw conversational programming as lowering barriers to creating bespoke assistive technology, they emphasized that democratization requires more than natural language alone. They identified conversational guidance, collaborative tool sharing, and support for user-provided data as critical structures for enabling broader participation and higher-quality tools.

Co-designers unilaterally saw ProgramAT as something that democratizes creation. D2 expressed a desire to show ProgramAT to his non-technical friends, believing it would empower them to create for themselves: *“I have some average Jill and Joe friends. Blind people that we do our research for, not academics. I would like to show them ProgramAT, and sit them down, have them give a one paragraph description, and see what they make... I think even that audience can... really make their own proof of concept. Most of these people don’t spend much time on a computer, they use their phones... so that experience alone makes programming accessible to all BLV people, whatever they do for work.”* D5 echoed this sentiment, expressing that *“the more democratized we can make it, the more people can actually leverage... I feel what ProgramAT does well is democratize the experience of building assistive technology... I’m pretty sure it would feel amazing if I wasn’t a developer to get access to something like this, because, oh my god, I suddenly become a creator, which most people can’t.”* D1 extends this line of thought, adding that systems like ProgramAT could support less technical people in feeling comfortable with technology; he imagines *“people might be able to get a sense of, like, I’m able to leverage this, and I’m not as scared about AI and technology as maybe I used to be.”* While future work is necessary to validate the utility of conversational programming for bespoke camera based AT in less technically savvy BLV populations that our co-designers predict, these perspectives nonetheless point to its potential as a pathway toward broader participation in assistive technology creation and a shift in how users engage with and feel empowered by AI-driven tools. We present findings that discuss appropriate supports to engage technical and non-technical users alike in building their own bespoke AT.

5.4.1 Creation as a conversation. Multiple co-designers noted that expanding the notion of conversation to the creation step could empower more intentional tool creation and be a valuable scaffold for those who may not know precisely what they want. Currently, conversation begins mostly at the iteration step, implicitly expecting the user to know precisely what they want; D2 describes, *“You kind of have to come with a full idea. It makes us more of the thinker, and it’s just the tool.”* The ability to sit, think this through, and arrive at a conclusion is well suited to more formal prompting, as the desktop experience enables: D2 notes that *“[he does] really enjoy sitting than writing it down and having time to think ... and actually come up with things that are worth doing”*. Yet, especially for mobile prompting, where the experience is more on-the-go and flexible, some of this structure for thinking through precisely what is desired can be lost. To this, D1—who prompted almost exclusively

from the mobile interface throughout the study—recollects that *“especially with the initial stages, for me, it was like, well, I have this idea, and then I’m not entirely sure, like, how the AI system is going to, like, respond...it could have been very nice to, before AI goes and runs off and does its thing, check in with me and kind of set up this mutual understanding [and] give me a chance to riff off of whatever the AI is, thinking about ... before I actually test out the tool.”* Bringing a conversational flow to the creation step could support end-users in both refining what they want out of a tool and optimizing their request to simultaneously meet their needs and align with what the underlying AI is capable of. However, adjusting this structure would come with tradeoffs. Simplicity of expression is one of the most appreciated aspects of the ProgramAT creation experience, and introducing a conversational layer could add friction. For D4, *“even though it’s just answering it back, it’s still not as simple...I’m not sure how people would feel about that.”*

5.4.2 Bespoke, yet collaborative. While all co-designers found being able to create unique tools valuable, some also mentioned a desire to be able to share the tools they made with others around them. While there are many needs not covered by commercial AT, the challenges they experience are typically not so unique others would not benefit from the solutions they build. This intuition was validated by the tools co-designers made over the course of the study: over 20% (8/37) of tools created were shared across participants. D1 expressed that while he could try to support someone he knew in building a tool for themselves by describing the steps he took to do it, *“you’re still gonna have to do those steps for yourself, and maybe that’s a good thing, so they can personalize it, maybe my tool is not exactly what they’re looking for, but at the same time, I feel like there’s still this issue of people duplicating efforts, and, like, because of maintenance and scalability and just, like, sustainability.”* D5 echoed this sentiment and advocates for a path to share the progress she has made with others, saying *“leveraging the power of community can be a very ideal next step, like how do we build on each other? Like today I built the mail sorter tool, let 10 people use it and iterate on it and then we all benefit. We don’t just build for ourselves.”* D5, as the co-designer who most frequently used the GitHub web interface, would occasionally see activity in other co-designers’ repositories, and found that this could offer a source of inspiration for things she would find useful herself, further grounding her collaboration would add value to a platform like ProgramAT, noting that *“as I was seeing other things that others have created, then I thought, ‘hey, that can be useful to me.’”*

5.4.3 From do-it-yourself tools to bring-it-yourself data. A recurring theme across co-designers was the desire to bring their own datasets into the tool creation process, particularly for tasks that require a high degree of specificity. While participants generally felt that existing tools performed well on broad, general-purpose tasks, they noted clear limitations when attempting to address more niche or context-dependent needs. In these cases, relying solely on non-specialized language models often resulted in outputs that were too generic to be useful, thus, co-designers expressed interest in augmenting the system with their own data, enabling tools to better reflect the particular environments, objects, or patterns that mattered in their daily lives. D2 references this as a struggle of tools based on generalized LLMs, and finds navigational tasks to be an

area where this is particularly salient, expressing that *“If it’s based on an LLM, it’s really cool, but I always get at least one hallucination. I much rather prefer if there’s something that I’m...using the AI for be trained for that specific purpose...with like OCO: it’s simply just trying to tell you whether or not a pedestrian signal is telling you to cross or not. When it’s LLM-based, I find it struggling in trying to do a lot of things at once. Jack of all trades, master of none.”*

Beyond simply improving performance, this desire points to a broader distinction between generalist and specialist tool use. Participants recognized that the current infrastructure excels at quickly scaffolding widely applicable functionality, but saw opportunities to extend this capability by incorporating more tailored sources of knowledge. This included not only uploading custom datasets, but also the possibility of integrating or switching between models better suited to specific domains. For example, in trying to build a braille reader tool D3 found general purpose models prone to egregious hallucination, and hoped to use Braille specific models to remedy this. Likewise, D4 hoped to provide a dataset of photos of chess moves to support a chess move tracker that could better understand what constituted a valid move. Co-designers attempted workarounds to this limitation, such as attaching photos to the GitHub issue for the tool to use as reference, as demonstrated in D2’s car finder, which made some progress, but served more as a primer to imagining what the bring-your-own-data experience could look like than a functional stand in. This was expressed by D2, who notes *“It would be nice if I could just, like, sideload a model, like, for example, like, color reading, let’s put a model, QR code reading or even image description, you can do it in ProgramAT...All possible.”* Such flexibility was framed as a way to bridge the gap between rapid prototyping and sustained, high-quality use, allowing tools to evolve from general approximations into more precise and reliable supports for specialized tasks and enabling utilization of ProgramAT as an ecosystem that can address a wide and comprehensive variety of needs, rather than a singular AT artifact.

5.4.4 AT co-use. All co-designers, prior to the study, used existing AI-powered AT such as BeMyAI and SeeingAI, and articulated the cases in which they would reach for one type of tool over another. Generally, co-designers agreed that ProgramAT tools were preferable for specific, repeatable tasks. Likewise, pre-built tools were preferred when generality was desirable, or when the task at hand would benefit from the shared experience of others, since the pre-built tools have thousands of users, while any ProgramAT tool is personal to the individual.

While, today, these use cases are largely separate, co-designers also speculated on futures for what co-use of bespoke tools like those made with ProgramAT and pre-built tools like BeMyAI or SeeingAI might look like. One prominent perspective envisioned a complementary relationship, where general-purpose tools and custom solutions work together to balance breadth and specificity. In this view, pre-built systems would continue to handle a wide range of common tasks, while custom tools would fill in gaps or provide more tailored functionality when needed. Participants described several ways this synergy might manifest. D1 proposed a model where general systems can infer user intent based on visual context and dynamically route tasks to the most appropriate tool, including custom ones. D5 visualized something similar, but where

an interface akin to ProgramAT was integrated in more generalized tools, allowing for the creation of bespoke tools as plug-ins as a way to address that no general tool can possibly meet every need. She argues *“BeMyEyes or SeeingAI should have, like, these kind of concept of plugins, right? Like, most AI tools are now letting you define your own tools, or define your own custom agents and things like that. ... In accessibility you cannot always check all the boxes. Every person is different ... and people know what works for them, so let them do that.”* In contrast to this fully integrated approach, D3 envisioned a future where pre-built and bespoke tools still exist separately, but can be freely surfaced on top of each other using mobile automations, similar to the concept presented in Herskovitz *et al.*’s A11yExtensions [45].

Other co-designers anticipated a more competitive trajectory, where advances in one category eventually render the other less relevant. D2 remarked that while ProgramAT is not yet advanced enough to unseat SeeingAI, when *“ProgramAT gets to the point that it has a sleek, sexy, straightforward interface, similar to those of Seeing AI and Lookout ... I will drop seeing AI, because I can make what Seeing AI does in ProgramAT ... like, ProgramAT is not one assistive technology. It is an AT ecosystem.”* Under the ecosystem framing, D2 believes that with time and technical development, ProgramAT could supplant existing tools by allowing end-users to reproduce the functionality of pre-built tools in a way that works best for them. While not every co-designer envisioned a future of co-use, some also speculated that mainstream AT could render the need to create AT conversationally irrelevant, or vice-versa. D4 noted that *“If one gets good, then I think the other will probably lose the market share”*, which could come either from the refinement of systems for custom tool creation as D2 describes, or from model improvements resulting in pre-built tools being more capable of addressing niche use cases.

Still, even with a future including these supports and structures in mind, some co-designers called into question whether democratizing the initial act of creation would successfully empower more people to create useful, usable tools. D1 expresses that though the benefits of more people being able to prototype and tailor tools to their needs are clear, the promise of this ultimately yielding more high quality tools is not. He remarks *“I don’t know, it’s a little bit unsure, still, if, prototyping easier and faster just means that we get more quality final systems. ... It’s easy to quickly iterate on stuff ... but I just don’t know high the ceiling can be raised. I think people would just kind of tend to use janky-ish technology, like their own personal tools and ... once people get into a rhythm of this is how I’m using AI in my life ... then you just kind of get stuck there, and you might not be able to have newer ideas with newer technologies.”* Even in finding ProgramAT valuable as a thought exercise, it too runs the risk of forming suboptimal habits rather than challenging them. He notes the risk of forming suboptimal habits and suboptimal tools would likely be more pronounced in the same community it stands to empower most—users without extensive development experience, because *“after you make a tool, and then try to iterate on it, but then if it’s not working, then what happens? And I feel like especially for people who aren’t ... used to thinking about programs, I think that’s gonna be more frustrating and more challenging in a lot of ways.”* In order to take advantage of a system like ProgramAT’s anticipated

power to democratize creation, future work must better understand and support the needs of novice creators.

6 Discussion

We examined how BLV co-designers engage in the creation of bespoke, camera-based assistive tools through an agentic programming system, focusing on both the conditions under which tools succeed and the sources of their failure. Our findings highlight that while such systems can enable rapid and flexible tool creation, success depends on a combination of factors, including alignment with system capabilities, the reliability of underlying models, and the co-designer iteratively refining prompts. Participants were broadly optimistic about the potential of ProgramAT to expand access to personalized assistive technology, while also surfacing key challenges in usability, transparency, and robustness. In this section, we consider these barriers and opportunities, and translate them to recommendations for future platforms, grounded in co-designers’ experiences (Section 6.1, Section 6.2, Section 6.3). Then, we discuss the limitations of these findings (Section 6.4).

6.1 Expand Creation and Access Modalities

ProgramAT supports multiple modes of creation and iteration of visual assistive tools, yielding nuances in how these modalities influenced our co-designers’ approach and motivations to building their tools. The templated interfaces available via GitHub enabled co-designers like D5 to deliberately compose and version tools outside of an immediate use context—affording time for reflection and structured refinement—whereas the free-form interface on the app supported a more in-situ and unstructured approach to quickly converge a situationally-inspired visual access need into a functional, testable tool. Though powerful, ProgramAT only supports testing on mobile devices, limiting in-situ inspiration to situations that do not expand to hands-free use. Expanding ProgramAT’s testing capabilities to support smart glasses could inspire a wider range of visual tools for hands-free scenarios such as cooking [50, 65], where visual assistance was desired by multiple participants (D4, D2), but since the hands were occupied with the task itself, it was impractical to use mobile assistance.

Further, supporting testing through smart glass form-factors warrants the need for intent specification modalities that are also hands-free. We look to making AT creation more conversational as one potential avenue. While iteration proved effective, the current workflow assumes that users begin with a well-formed idea of what they want to build. Several co-designers noted that this places a cognitive burden on the user, particularly in early stages when goals may still be evolving. Enabling back-and-forth interaction before a tool is generated could help users clarify their intent, explore alternatives, and align their requests with what the system is capable of, rather than committing prematurely to a fixed specification. When working from a hands-free, keyboardless interface like glasses, the challenges to appropriate specification via dictation mentioned by D2 are magnified, further motivating the need to support users in clearly thinking through what they want.

At the same time, integrating conversation into creation introduces relevant design tradeoffs. In particular, the current experience is valued for its immediacy and low overhead, allowing users to

quickly express an idea and see results. Adding a conversational layer could risk increasing friction. Future work should explore how to balance these dynamics, for instance, by supporting lightweight, optional dialogue that scaffolds early ideation without disrupting ProgramAT’s appealing speed and simplicity.

6.2 Collaborative Testbeds to Support Verification, Knowledge and Tool Sharing

The popular framing of ProgramAT as “testbed” by our co-designers points towards a broader opportunity to not just probe the capabilities and limits of AI systems, but to scaffold collective knowledge about how, when, and why they succeed and fail. In our study, co-designers already treated tools as informal testbeds: systematically evaluating them on inputs with known answers before deploying them in higher stakes contexts. This practice reflects a critical need among BLV users to understand system behavior, especially given the limited non-visual explainability of many AI systems [2] and the difficulty of diagnosing inconsistent outputs [8]. However, these investigations currently remain individual and ad hoc, placing the burden of learning system behavior on each individual user.

We argue for collective testbeds that externalize and aggregate this process. Rather than each user independently rediscovering the same limitations, such infrastructures could support the sharing not just of tools as discussed in Section 5.4.2 but also the strategies, assumptions and empirical insights that emerge through their use. Collaborative testbeds conceptually extend the more conventional forms of sharing wherein users exchange tools or configurations [18] by also capturing the reasoning and experimentation that make shared artifacts effective. Our co-designers developed tacit knowledge about which models were most reliable for particular tasks, how to structure prompts, and how to interpret uncertain outputs; making this knowledge visible and reusable could reduce duplicate efforts and accelerate effective tool creation, especially when the suite of available models in ProgramAT expands.

This vision parallels model evaluation dashboards in non-accessibility domains, which allow developers to compare model performance across task types [39, 84]. A similar paradigm for accessibility could answer questions about model suitability for different tasks, or under what environmental conditions certain approaches break down. Importantly, these insights would be grounded not in abstract benchmarks, but in situated use cases.

Enabling this form of sharing raises important design challenges. Tools must be represented in ways that make their assumptions, appropriate contexts of use, and limitations legible. Mechanisms for versioning [75], attribution, and iterative refinement are also necessary as tools evolve across contributors. At the same time, systems must preserve the flexibility for personalization that motivates bespoke tools in the first place. Future work should explore lightweight models of collaboration that balance supporting reuse and collective intelligence without constraining individual adaptation.

6.3 Agentic Support for Scalability and Sustainable AT Creation

Finally, we argue for strengthening agentic supports as a path towards scalable and sustainable AT creation. A first key step is to

expand ProgramAT’s set of reusable building blocks. Current components are motivated by core functionalities modeled in existing AT (e.g., scene description [31], OCR [74]) or functionalities that prior literature suggests would be valuable (e.g., camera aiming [45]). These provide a strong foundation, yet, the tools of interest in our study suggest the need for additional primitives such as teachable object recognizers or parameterizable components. Broadening these building blocks would better support diverse tool creation while reducing the effort required of end-users.

Yet, for these tools to scale in utility, we must also consider how ProgramAT might scale to the broader BLV population, necessitating explicitly addressing the cost structures underlying these systems. ProgramAT currently relies on a host of paid architectures (e.g., server hosting, API calls, Copilot subscriptions) which limits scalability, particularly within disability communities that disproportionately face economic barriers [25]. Supporting open-source development, self hosting, and lower-connectivity use cases are important steps towards equitable access.

We recommend the adoption of small language models (SLMs) to address equitability and sustainability challenges as well as extend agentic support. Deploying SLMs on-device or in lightweight local environments can significantly reduce dependence on paid APIs and robust server architectures [87], while simultaneously introducing new opportunities for tools that require lower latency than LLM-enabled tools, but still need more generality than scoped models like YOLO offer [118]. This shift positions SLMs as both a legitimate technical alternative and a key enabler of scalable, sustainable, and community-accessible AT ecosystems.

6.4 Limitations

6.4.1 Sample size. While our co-designers were intensively involved in the research process, generating rich insights and data, a sample of five remains insufficient to safely generalize to the BLV community at large. The technical experience of our five co-designers may not generalize to all BLV users; non-technical users would likely require supports beyond those surfaced here to achieve similar results [71]. However, rather than claiming generalizability, we offer rich insights into the practices of technically experienced BLV co-designers, surfacing a grounded understanding of emerging practices, breakdowns, and support needs that can inform future system design and broader investigations.

Still, while five participants is not large for an empirical study, we maintain it is not remarkably small for co-design. Methodologically similar works draw on three [23, 117], two [35, 45, 115], or even one [30, 48, 76] co-designer. These works often propose design guidelines: common practice in our community.

6.4.2 Hallucinations and AI reliability. ProgramAT relies on generative AI for tool creation and to support several tools’ intelligence. As such, there is an inherent risk of hallucinations. Literature suggests that trust and accuracy are especially vital for systems used by BLV users: making these challenges especially critical to address [2]. Neither code review nor interviews with co-designers revealed hallucinations in ProgramAT more pronounced than those present in existing forms of AI-AT such as BeMyAI. Still, this does not negate the inherent risk, which we aim to minimize by supporting users in building validation into their tools. Co-designers approached

risk mitigation multiple ways: from building confidence-reporting into tools (D1, D2) to designing control flows in which the model would not answer visual questions prior to validating environmental factors like lighting (D4). Through these affordances, we hope to engender agency in building tools users can trust, while still recognizing the potential for inadvertent harm.

6.4.3 Design Considerations for ProgramAT Study Instrument. Though ProgramAT proved useful in supporting development of a variety of assistive tools, its capabilities are limited.

Firstly, ProgramAT makes the privacy-minded choice to discard images passed by the user once they have been processed. However, this means ProgramAT supports only stateless tools: ruling out functionalities such as personal object recognizers (though D2 achieved a similar result through providing an object description in the tool prompt in his Waffle Plushy Finder (see Table 2), and hindering the success of tools that aimed to surface change over time, like D5's Straight Line Walker (see Table 2).

Furthermore, while ProgramAT is distinct from commercial AI tools in many ways—most notably in its generation of real code rather than just wrapping LLM calls and in its ability to iterate and refine existing tools—it still has many things in common with existing AI tools. Indeed, several features of ProgramAT were directly inspired by commercial offerings. The conversation component of *Take Photo* mode was modeled after the BeMyAI follow up question pattern [31]. Futher, *Live* mode was directly inspired by OpenAI's CustomGPT [78]: being introduced to the study instrument after D2 experimented with it as an alternative way of making custom tools, but found it cumbersome to constantly need to reprompt.

7 Conclusion

We examine how technically experienced BLV users engage with agentic programming to create camera based assistive tools, focusing on their motivations and strategies to create and iterate on tools. Our two-month long study, facilitated by ProgramAT, reveals that generative AI delivers on the promise to meaningfully lower barriers to prototyping and rapid iteration. Successful tool creation, however, remains an inherently collaborative process: reliant on users correcting, shaping, and extending AI behavior. These results highlight both the promise and current limitations of AI-supported AT creation—despite reducing the barriers to initial creation, AI does not eliminate the need for technical judgment and iterative engagement. By exploring where users find value and friction in authoring bespoke assistive tools with agentic code, we provide design considerations for how future systems could support BLV user-led creation in the future, including supporting collaborative dynamics, emphasizing transparency, controllability, and mechanisms for grounding tool behavior in users' lived contexts.

Acknowledgments

First and foremost, we thank our co-designers and co-authors Gene, Aziz, Ather, Kun and Aditi for their time and commitment to this work. This research would not have been possible without them. We also thank our reviewers for their time and feedback. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE 2241144, the National Science Foundation CAREER Award No. 2442243, and the

National Science Foundation HCC Award No. IIS-2516629. Any opinions, findings, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the National Science Foundation. This work was also supported in part by a Google Research Scholar Award.

References

- [1] Sushma Adepu and Rachel F Adler. 2016. A comparison of performance and preference on mobile devices vs. desktop computers. In *2016 IEEE 7th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. IEEE, 1–7.
- [2] Rudaiba Adnin and Maitraye Das. 2024. "I look at it as the king of knowledge": How Blind People Use and Understand Generative AI Tools. In *Proceedings of the 26th International ACM SIGACCESS Conference on Computers and Accessibility (St. John's, NL, Canada) (ASSETS '24)*. Association for Computing Machinery, New York, NY, USA, Article 64, 14 pages. doi:10.1145/3663548.3675631
- [3] Leila Aflatoony, Su Jin Lee, and Jon Sanford. 2023. Collective making: Co-designing 3D printed assistive technologies with occupational therapists, designers, and end-users. *Assistive Technology* 35, 2 (2023), 153–162.
- [4] Mehmet Akhoroz and Caglar Yildirim. 2025. Conversational AI as a Coding Assistant: Understanding Programmers' Interactions with and Expectations from Large Language Models for Coding. *arXiv preprint arXiv:2503.16508* (2025).
- [5] Gus Alexiou. 2023. OKO App Leverages AI To Help Blind Pedestrians Recognize Traffic Signals. <https://www.forbes.com/sites/gusalexioiu/2023/08/10/oko-app-deploys-ai-to-make-crossing-the-street-safer-for-blind-pedestrians/>
- [6] Rahaf Alharbi, Robin N. Brewer, and Sarita Schoenebeck. 2022. Understanding Emerging Obfuscation Technologies in Visual Description Services for Blind and Low Vision People. *Proc. ACM Hum.-Comput. Interact.* 6, CSCW2, Article 469 (Nov. 2022), 33 pages. doi:10.1145/3555570
- [7] Rahaf Alharbi, Angela D Cheong, Jaylin Herskovitz, Robin N. Brewer, and Sarita Schoenebeck. 2025. "Trying to Piece It Together": Exploring Accessible Error Detection in Emerging Privacy Techniques With Blind People. In *Proceedings of the 27th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '25)*. Association for Computing Machinery, New York, NY, USA, Article 18, 20 pages. doi:10.1145/3663547.3746376
- [8] Rahaf Alharbi, Pa Lor, Jaylin Herskovitz, Sarita Schoenebeck, and Robin N. Brewer. 2024. Misfitting With AI: How Blind People Verify and Contest AI Errors. In *Proceedings of the 26th International ACM SIGACCESS Conference on Computers and Accessibility (St. John's, NL, Canada) (ASSETS '24)*. Association for Computing Machinery, New York, NY, USA, Article 61, 17 pages. doi:10.1145/3663548.3675659
- [9] Anthropic. 2026. Claude Code | Anthropic's agentic coding system. <https://www.anthropic.com/product/claude-code>
- [10] Anysphere. 2024. Cursor - The AI Code Editor. <https://cursor.com/>
- [11] Tobias Baum, Olga Liskin, Kai Niklas, and Kurt Schneider. 2016. Factors influencing code review processes in industry. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (Seattle, WA, USA) (FSE 2016)*. Association for Computing Machinery, New York, NY, USA, 85–96. doi:10.1145/2950290.2950323
- [12] Jeffrey P Bigham and Patrick Carrington. 2018. Learning from the front: People with disabilities as early adopters of AI. *Proceedings of the 2018 HCIC Human-Computer Interaction Consortium* (2018).
- [13] Melanie Birks, Ysanne Chapman, and Karen Francis. 2008. Memoing in qualitative research: Probing data and processes. *Journal of research in nursing* 13, 1 (2008), 68–75.
- [14] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.
- [15] Virginia Braun and Victoria Clarke. 2021. Can I use TA? Should I use TA? Should I not use TA? Comparing reflexive thematic analysis and other pattern-based qualitative analytic approaches. *Counselling and psychotherapy research* 21, 1 (2021), 37–47.
- [16] Erin Buehler, Stacy Branham, Abdullah Ali, Jeremy J. Chang, Megan Kelly Hofmann, Amy Hurst, and Shaun K. Kane. 2015. Sharing is Caring: Assistive Technology Designs on Thingiverse. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) (CHI '15). Association for Computing Machinery, New York, NY, USA, 525–534. doi:10.1145/2702123.2702525
- [17] Erin Buehler, Amy Hurst, and Megan Hofmann. 2014. Coming to grips: 3D printing for accessibility. In *Proceedings of the 16th International ACM SIGACCESS Conference on Computers & Accessibility* (Rochester, New York, USA) (ASSETS '14). Association for Computing Machinery, New York, NY, USA, 291–292. doi:10.1145/2661334.2661345
- [18] Yoonha Cha, Victoria Jackson, Karina Kohl, Rafael Prikladnicki, André van der Hoek, and Stacy Branham. 2025. The Dilemma of Building Do-It-Yourself (DIY) Solutions For Workplace Accessibility. In *Proceedings of the 2025 CHI Conference*

- on *Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 47, 17 pages. doi:10.1145/3706598.3713302
- [19] Rueli-Che Chang, Rosiana Natalie, Wenqian Xu, Jovan Zheng Feng Yap, and Anhong Guo. 2025. Probing the Gaps in ChatGPT's Live Video Chat for Real-World Assistance for People who are Blind or Visually Impaired. In *Proceedings of the 27th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '25)*. Association for Computing Machinery, New York, NY, USA, Article 71, 14 pages. doi:10.1145/3663547.3746319
 - [20] Parmit K Chilana, Celena Alcock, Shruti Dembla, Anson Ho, Ada Hurst, Brett Armstrong, and Philip J Guo. 2015. Perceptions of non-CS majors in intro programming: The rise of the conversational programmer. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 251–259.
 - [21] Jeanne Choi, Dasom Choi, Sejun Jeong, Hwajung Hong, and Joseph Seering. 2025. How Far I'll Go: Imagining Futures of Conversational AI with People with Visual Impairments Through Design Fiction. arXiv:2510.12268 [cs.HC] <https://arxiv.org/abs/2510.12268>
 - [22] Md Shain Shahid Chowdhury, Md Naseef Ur Rahman Chowdhury, Fariha Ferdous Neha, and Ahshanul Haque. 2024. AI-powered code reviews: Leveraging large language models. In *2024 International Conference on Signal Processing and Advance Research in Computing (SPARC)*, Vol. 1. IEEE, 1–6.
 - [23] Rodolfo Cossovic, Steve Hodges, Jin Kang, and Audrey Girouard. 2023. Co-designing new keyboard and mouse solutions with people living with motor impairments. In *Proceedings of the 25th International ACM SIGACCESS Conference on Computers and Accessibility (New York, NY, USA) (ASSETS '23)*. Association for Computing Machinery, New York, NY, USA, Article 98, 7 pages. doi:10.1145/3597638.3614549
 - [24] Humphrey Curtis, Zihao You, William Deary, Miruna-Ioana Tudoreanu, and Timothy Neate. 2023. Envisioning the (In)Visibility of Discreet and Wearable AAC Devices. 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 52, 19 pages. doi:10.1145/3544548.3580936
 - [25] Rakhi Dandona and Lalit Dandona. 2001. Socioeconomic status and blindness. *British journal of ophthalmology* 85, 12 (2001), 1484–1488.
 - [26] Melissa DeJonckheere, Lisa M Vaughn, Tyler G James, and Amanda C Schon-delmeyer. 2024. Qualitative thematic analysis in a mixed methods study: Guidelines and considerations for integration. *Journal of Mixed Methods Research* 18, 3 (2024), 258–269.
 - [27] Aashaka Desai, Jennifer Mankoff, and Richard E. Ladner. 2025. Exploring Disability Culture Through Accounts of Disabled Innovators of Accessibility Technology. In *Proceedings of the 27th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '25)*. Association for Computing Machinery, New York, NY, USA, Article 78, 14 pages. doi:10.1145/3663547.3746330
 - [28] Sophia DiCuffa, Amanda Zambrana, Priyanshi Yadav, Sashidhar Madiraju, Khushi Suman, and Eman Abdullah AlOmar. 2025. Exploring Prompt Patterns in AI-Assisted Code Generation: Towards Faster and More Effective Developer-AI Collaboration. In *2025 IEEE 4th International Conference on Computing and Machine Intelligence (ICMI)*. IEEE, 1–7.
 - [29] Patricia A Dwyer. 2020. Analysis and synthesis. In *A step-by-step guide to conducting an integrative review*. Springer, 57–70.
 - [30] Kirsten Ellis, Ross de Vent, Reuben Kirkham, and Patrick Olivier. 2020. Bespoke Reflections: Creating a One-Handed Braille Keyboard. In *Proceedings of the 22nd International ACM SIGACCESS Conference on Computers and Accessibility (Virtual Event, Greece) (ASSETS '20)*. Association for Computing Machinery, New York, NY, USA, Article 13, 13 pages. doi:10.1145/3373625.3417019
 - [31] Be My Eyes. 2024. Introducing: Be My AI. <https://www.bemyeyes.com/blog/introducing-be-my-ai>
 - [32] Ahmed Fawzy, Amjed Tahir, and Kelly Blincoe. 2025. Vibe Coding in Practice: Motivations, Challenges, and a Future Outlook – a Grey Literature Review. arXiv:2510.00328 [cs.SE] <https://arxiv.org/abs/2510.00328>
 - [33] Cristina Fona. 2023. Qualitative data analysis: Using thematic analysis. In *Researching and analysing business*. Routledge, 130–145.
 - [34] Bhanuka Gamage, Thanh-Toan Do, Nicholas Seow Chiang Price, Arthur Lowery, and Kim Marriott. 2023. What do Blind and Low-Vision People Really Want from Assistive Smart Devices? Comparison of the Literature with a Focus Study. In *Proceedings of the 25th International ACM SIGACCESS Conference on Computers and Accessibility (New York, NY, USA) (ASSETS '23)*. Association for Computing Machinery, New York, NY, USA, Article 30, 21 pages. doi:10.1145/3597638.3608955
 - [35] Bhanuka Gamage, Nicola McDowell, Dijana Kovacic, Leona Holloway, Thanh-Toan Do, Arthur James Lowery, Nicholas Price, and Kim Marriott. 2025. Smart Glasses for CVI: Co-Designing Extended Reality Solutions to Support Environmental Perception by People with Cerebral Visual Impairment. In *Proceedings of the 27th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '25)*. Association for Computing Machinery, New York, NY, USA, Article 2, 24 pages. doi:10.1145/3663547.3746383
 - [36] Vladimir Geroimenko. 2025. Key principles of good prompt design. In *The Essential Guide to Prompt Engineering: Key Principles, Techniques, Challenges, and Security Risks*. Springer, 17–36.
 - [37] Github. 2025. About issues - GitHub Docs. <https://docs.github.com/en/issues/tracking-your-work-with-issues/learning-about-issues/about-issues>
 - [38] GitHub. 2025. GitHub Copilot · Your AI pair programmer. <https://github.com/features/copilot>
 - [39] Pareesa Ameneh Golnari, Adarsh Kumarappan, Wen Wen, Xiaoyu Liu, Gabriel Ryan, Yuting Sun, Shengyu Fu, and Elsie Nallipogu. 2026. DevBench: A Realistic, Developer-Informed Benchmark for Code Generation Models. arXiv preprint arXiv:2601.11895 (2026).
 - [40] Diana Grüter, Kevin Krings, Sven Hoffmann, Nino Bohn, and Thomas Ludwig. 2025. Enhancing Craftsmanship: Evaluation of AR-Assisted Learning of Work Steps in Woodworking: Enhancing Craftsmanship: Evaluation of AR-Assisted Learning of Work Steps in Woodworking. *Comput. Supported Coop. Work* 34, 4 (Dec. 2025), 1159–1220. doi:10.1007/s10606-025-09532-2
 - [41] Md. Asif Haider and Thomas Zimmermann. 2026. Understanding Dominant Themes in Reviewing Agentic AI-authored Code. arXiv:2601.19287 [cs.SE] <https://arxiv.org/abs/2601.19287>
 - [42] Brandon Haworth, Muhammad Usman, Melanie Baljko, and Foad Hamidi. 2016. The use of working prototypes for participatory design with people with disabilities. In *International Conference on Computers Helping People with Special Needs*. Springer, 134–141.
 - [43] Liwen He, Yifan Li, Mingming Fan, Liang He, and Yuhang Zhao. 2023. A Multi-modal Toolkit to Support DIY Assistive Technology Creation for Blind and Low Vision People. In *Adjunct Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (San Francisco, CA, USA) (UIST '23 Adjunct)*. Association for Computing Machinery, New York, NY, USA, Article 3, 3 pages. doi:10.1145/3586182.3616646
 - [44] Jaylin Herskovitz. 2024. DIY assistive software: End-user programming for personalized assistive technology. *ACM SIGACCESS Accessibility and Computing* 137 (2024), 1–1.
 - [45] Jaylin Herskovitz, Ellie Seehorn, Ather Jammoo, Jason Meddaugh, and Anhong Guo. 2026. A11yExtensions: Accessibility Extensions to Augment Mobile AI Assistive Technology In-Situ. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems (CHI '26)*. Association for Computing Machinery, New York, NY, USA, Article 204, 23 pages. doi:10.1145/3772318.3791559
 - [46] Jaylin Herskovitz, Andi Xu, Rahaf Alharbi, and Anhong Guo. 2023. Hacking, Switching, Combining: Understanding and Supporting DIY Assistive Technology Design by Blind People. 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 57, 17 pages. doi:10.1145/3544548.3581249
 - [47] Jaylin Herskovitz, Andi Xu, Rahaf Alharbi, and Anhong Guo. 2024. ProgramAlly: Creating Custom Visual Access Programs via Multi-Modal End-User Programming. 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 85, 15 pages. doi:10.1145/3654777.3676391
 - [48] Jeremy Zhengqi Huang, Jaylin Herskovitz, Liang-Yuan Wu, Cecily Morrison, and Dhruv Jain. 2025. Weaving Sound Information to Support Real-Time Sensemaking of Auditory Environments: Co-Designing with a DHH User. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 75, 18 pages. doi:10.1145/3706598.3714268
 - [49] Shuxu Huffman, Robin Angelini, Raja Kushalnagar, and Katta Spiel. 2025. "It only needs to work for one of us": Rethinking DIY Deaf Tech Through Situated Co-Design. In *Proceedings of the 27th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '25)*. Association for Computing Machinery, New York, NY, USA, Article 7, 7 pages. doi:10.1145/3663547.3749828
 - [50] Mina Huh, Zihui Xue, Ujjaini Das, Kumar Ashutosh, Kristen Grauman, and Amy Pavel. 2025. Vid2Coach: Transforming How-To Videos into Task Assistants. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*. 1–24.
 - [51] Amy Hurst and Jasmine Tobias. 2011. Empowering individuals with do-it-yourself assistive technology. In *The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility*. 11–18.
 - [52] Dhruv Jain, Audrey Desjardins, Leah Findlater, and Jon E. Froehlich. 2019. Autoethnography of a Hard of Hearing Traveler. In *Proceedings of the 21st International ACM SIGACCESS Conference on Computers and Accessibility (Pittsburgh, PA, USA) (ASSETS '19)*. Association for Computing Machinery, New York, NY, USA, 236–248. doi:10.1145/3308561.3353800
 - [53] JiWoong Jang, Yunzhi Li, and Patrick Carrington. 2022. "I Should Feel Like I'm In Control": Understanding Expectations, Concerns, and Motivations for the Use of Autonomous Navigation on Wheelchairs. In *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility (Athens, Greece) (ASSETS '22)*. Association for Computing Machinery, New York, NY, USA, Article 44, 5 pages. doi:10.1145/3517428.3550380

- [54] Hernisa Kacorri, Kris M. Kitani, Jeffrey P. Bigham, and Chieko Asakawa. 2017. People with Visual Impairment Training Personal Object Recognizers: Feasibility and Challenges. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) (CHI '17). Association for Computing Machinery, New York, NY, USA, 5839–5849. doi:10.1145/3025453.3025899
- [55] Sanchita S. Kamath, Aziz N. Zeidieh, Venkatesh Potluri, Sile O'Modhrain, Kenneth Perry, and JooYoung Seo. 2026. Three Modalities, Two Design Probes, One Prototype, and No Vision: Experience-Based Co-Design of a Multi-modal 3D Data Visualization Tool. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems* (CHI '26). Association for Computing Machinery, New York, NY, USA, Article 1641, 23 pages. doi:10.1145/3772318.3791272
- [56] Elizabeth Keating and Gene Mirus. 2003. American Sign Language in virtual space: Interactions between deaf users of computer-mediated video communication and the impact of technology on language practices. *Language in Society* 32, 5 (2003), 693–714.
- [57] Amy J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. 2011. The state of the art in end-user software engineering. *ACM Comput. Surv.* 43, 3, Article 21 (April 2011), 44 pages. doi:10.1145/1922649.1922658
- [58] Ben Kosa, Hsuanling Lee, Jasmine Li, Sanbrita Mondal, Yuhang Zhao, and Liang He. 2026. Not Seeing the Whole Picture: Challenges and Opportunities in Using AI for Co-Making Physical, DIY-AT for People with Visual Impairments. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems* (CHI '26). Association for Computing Machinery, New York, NY, USA, Article 286, 20 pages. doi:10.1145/3772318.3791815
- [59] Kevin Krings, Nino S Bohn, and Thomas Ludwig. 2025. (R) evolution of Programming: Vibe Coding as a Post-Coding Paradigm. *arXiv preprint arXiv:2510.12364* (2025).
- [60] Richard E Ladner. 2015. Design for user empowerment. *interactions* 22, 2 (2015), 24–29.
- [61] Cheuk Yin Phipson Lee, Zhuohao Zhang, Jaylin Herskovitz, JooYoung Seo, and Anhong Guo. 2022. CollabAlly: Accessible Collaboration Awareness in Document Editing. 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 596, 17 pages. doi:10.1145/3491102.3517635
- [62] Jaewook Lee, Davin Win Kyi, Leejun Kim, Jenny Peng, Gagyem Lim, Jeremy Zhengqi Huang, Dhruv Jain, and Jon E Froehlich. 2025. SonoCraftAR: Towards Supporting Personalized Authoring of Sound-Reactive AR Interfaces by Deaf and Hard of Hearing Users. In *2025 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. IEEE, 536–540.
- [63] Jong Ho Lee and Stephanie Valencia. 2026. Do-It-Yourself AAC: Co-Designing User-Programmable AI Communication Tools with People with Aphasia. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems* (CHI '26). Association for Computing Machinery, New York, NY, USA, Article 206, 20 pages. doi:10.1145/3772318.3790321
- [64] Kyungjin Lee. 2020. Teachable object recognizers for the blind: using first-person vision. *ACM SIGACCESS Accessibility and Computing* 123 (2020), 1–1.
- [65] Franklin Mingzhe Li, Jamie Dorst, Peter Cederberg, and Patrick Carrington. 2021. Non-Visual Cooking: Exploring Practices and Challenges of Meal Preparation by People with Visual Impairments. In *Proceedings of the 23rd International ACM SIGACCESS Conference on Computers and Accessibility* (Virtual Event, USA) (ASSETS '21). Association for Computing Machinery, New York, NY, USA, Article 30, 11 pages. doi:10.1145/3441852.3471215
- [66] Franklin Mingzhe Li, Akihiko Oharazawa, Chloe Qingyu Zhu, Misty Fan, Daisuke Sato, Chieko Asakawa, and Patrick Carrington. 2025. More than One Step at a Time: Designing Procedural Feedback for Non-visual Makeup Routines. In *Proceedings of the 27th International ACM SIGACCESS Conference on Computers and Accessibility* (ASSETS '25). Association for Computing Machinery, New York, NY, USA, Article 32, 17 pages. doi:10.1145/3663547.3746366
- [67] Franklin Mingzhe Li, Franchesca Spektor, Meng Xia, Mina Huh, Peter Cederberg, Yuqi Gong, Kristen Shinohara, and Patrick Carrington. 2022. "It Feels Like Taking a Gamble": Exploring Perceptions, Practices, and Challenges of Using Makeup and Cosmetics for People with Visual Impairments. 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 266, 15 pages. doi:10.1145/3491102.3517490
- [68] Franklin Mingzhe Li, Lotus Zhang, Maryam Bandukda, Abigale Stangl, Kristen Shinohara, Leah Findlater, and Patrick Carrington. 2023. Understanding Visual Arts Experiences of Blind People. 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 60, 21 pages. doi:10.1145/3544548.3580941
- [69] Mixuan Li and Leila Aflatoony. 2024. Exploring the Potential of Generative AI in DIY Assistive Technology Design by Occupational Therapists. In *Proceedings of the 26th International ACM SIGACCESS Conference on Computers and Accessibility* (St. John's, NL, Canada) (ASSETS '24). Association for Computing Machinery, New York, NY, USA, Article 109, 6 pages. doi:10.1145/3663548.3688506
- [70] Hong Yi Lin, Patanamon Thongtanunam, Christoph Treude, and Wachiraphan Charoenwet. 2024. Improving automated code reviews: Learning from experience. In *Proceedings of the 21st International Conference on Mining Software Repositories*. 278–283.
- [71] Qianou Ma, Kenneth R Koedinger, and Tongshuang Wu. 2026. Not Everyone Wins with LLMs: Behavioral Patterns and Pedagogical Implications for AI Literacy in Programmatic Data Science. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems* (CHI '26). Association for Computing Machinery, New York, NY, USA, Article 139, 22 pages. doi:10.1145/3772318.3791283
- [72] Yoshiaki Matsuzawa, Takashi Ohata, Manabu Sugiura, and Sanshiro Sakai. 2015. Language migration in non-CS introductory programming through mutual language translation environment. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. 185–190.
- [73] Alexander J. Medeiros, Lee Stearns, Leah Findlater, Chuan Chen, and Jon E. Froehlich. 2017. Recognizing Clothing Colors and Visual Textures Using a Finger-Mounted Camera: An Initial Investigation. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility* (Baltimore, Maryland, USA) (ASSETS '17). Association for Computing Machinery, New York, NY, USA, 393–394. doi:10.1145/3132525.3134805
- [74] Microsoft. 2021. Seeing AI. <https://www.microsoft.com/en-us/ai/seeing-ai>
- [75] Farhani Momotaz, Md Ehtesham-Ul-Haque, and Syed Masum Billah. 2023. Understanding the Usages, Lifecycle, and Opportunities of Screen Readers' Plugins. *ACM Trans. Access. Comput.* 16, 2, Article 17 (July 2023), 35 pages. doi:10.1145/3582697
- [76] Cecily Morrison, Edward Cutrell, Martin Grayson, Anja Thieme, Alex Taylor, Geert Roumen, Camilla Longden, Sebastian Tschatschek, Rita Faia Marques, and Abigail Sellen. 2021. Social Sensemaking with AI: Designing an Open-ended AI Experience with a Blind Child. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) (CHI '21). Association for Computing Machinery, New York, NY, USA, Article 396, 14 pages. doi:10.1145/3411764.3445290
- [77] Peya Mowar, Yi-Hao Peng, Jason Wu, Aaron Steinfeld, and Jeffrey P Bigham. 2025. Codea11y: Making ai coding assistants useful for accessible web development. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–15.
- [78] OpenAI. 2023. Introducing GPTs. <https://openai.com/index/introducing-gpts/>
- [79] Hyanghee Park and Joonhwan Lee. 2021. Designing a Conversational Agent for Sexual Assault Survivors: Defining Burden of Self-Disclosure and Envisioning Survivor-Centered Solutions. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) (CHI '21). Association for Computing Machinery, New York, NY, USA, Article 634, 17 pages. doi:10.1145/3411764.3445133
- [80] Betsy Phillips and Hongxin Zhao. 1993. Predictors of assistive technology abandonment. *Assistive technology* 5, 1 (1993), 36–45.
- [81] Patricia Piedade, Nikoleta Matur, Catarina Alexandra Rebelo Rodrigues, Francisco Cecilio, Afonso Marques, Rings Of Saturn, Isabel Neto, and Hugo Nicolau. 2022. Co-designing a Bespoken Wearable Display for People with Dissociative Identity Disorder. In *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility* (Athens, Greece) (ASSETS '22). Association for Computing Machinery, New York, NY, USA, Article 48, 4 pages. doi:10.1145/3517428.3550369
- [82] Deepak Babu Piskala. 2026. Spec-Driven Development: From Code to Contract in the Age of AI Coding Assistants. *arXiv preprint arXiv:2602.00180* (2026).
- [83] Lauren R. Milne and Lucy J Rubin. 2025. Blocks4All and Beyond Demonstration: Making Block-Based Programming More Universally Accessible. In *Proceedings of the 27th International ACM SIGACCESS Conference on Computers and Accessibility*. 1–4.
- [84] Muhammad Shihab Rashid, Christian Bock, Yuan Zhuang, Alexander Buchholz, Tim Esler, Simon Valentin, Luca Franceschi, Martin Wistuba, Prabhu Teja Sivaprasad, Woo Jung Kim, et al. 2025. Swe-polybench: A multi-language benchmark for repository level evaluation of coding agents. *arXiv preprint arXiv:2504.08703* (2025).
- [85] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.
- [86] Ayon Roy, Enamul Karim, Minhaz Bin Farukee, and Fillia Makedon. 2024. ChatGPT as an Assistive Technology: Enhancing Human-Computer Interaction for People with Speech Impairments. In *Proceedings of the 17th International Conference on Pervasive Technologies Related to Assistive Environments* (Crete, Greece) (PETRA '24). Association for Computing Machinery, New York, NY, USA, 63–66. doi:10.1145/3652037.3663956
- [87] Neelamani Samal, Suresh Kaswan, Reshma, Rabina Bagga, Parul Goyal, and Lisha Yugal. 2025. Small but Mighty: A Comparative Review of Small Language Models and Their Advantages. In *International Conference on AI Systems and Sustainable Technologies*. Springer, 293–303.

- [88] Saquib Sarwar and David Wilson. 2025. "Some are junk, but some are brilliant"-Occupational Therapist's Perspectives on 3D Printed Assistive Technology. In *Proceedings of the 27th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '25)*. Association for Computing Machinery, New York, NY, USA, Article 152, 5 pages. doi:10.1145/3663547.3759756
- [89] Saquib Sarwar, David Wilson, and Khairul Mahbub. 2026. Grassroots Maker Perspectives on Participation in Do-It-Yourself Assistive Technology Development. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems (CHI '26)*. Association for Computing Machinery, New York, NY, USA, Article 1605, 16 pages. doi:10.1145/3772318.3790477
- [90] Ather Sharif, Aedan Liam McCall, and Kianna Rocas Bolante. 2022. Should I Say "Disabled People" or "People with Disabilities"? Language Preferences of Disabled People Between Identity- and Person-First Language. In *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility (Athens, Greece) (ASSETS '22)*. Association for Computing Machinery, New York, NY, USA, Article 10, 18 pages. doi:10.1145/3517428.3544813
- [91] Tanusree Sharma, Yu-Yun Tseng, Lotus Zhang, Ayae Ide, Kelly Avery Mack, Leah Findlater, Danna Gurari, and Yang Wang. 2025. "Before, I Asked My Mom, Now I Ask ChatGPT": Visual Privacy Management with Generative AI for Blind and Low-Vision People. In *Proceedings of the 27th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '25)*. Association for Computing Machinery, New York, NY, USA, Article 17, 14 pages. doi:10.1145/3663547.3746335
- [92] Karin Slegers, Kristel Kouwenberg, Tereza Loučova, and Ramon Daniels. 2020. Makers in healthcare: The role of occupational therapists in the design of DIY assistive technology. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–11.
- [93] Michael Smyth, Ingi Helgason, Frank Kresin, Mara Balestrini, Andreas B. Unteidig, Shaun Lawson, Mark Gaved, Nick Taylor, James Auger, Lone Koefoed Hansen, Douglas C. Schuler, Mel Woods, and Paul Dourish. 2018. Maker Movements, Do-It-Yourself Cultures and Participatory Design: Implications for HCI Research. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems (Montreal QC, Canada) (CHI EA '18)*. Association for Computing Machinery, New York, NY, USA, 1–7. doi:10.1145/3170427.3170604
- [94] Abigale Stangl, Emma Sadjó, Pardis Emami-Naeini, Yang Wang, Danna Gurari, and Leah Findlater. 2023. "Dump it, destroy it, send it to data heaven": Blind people's expectations for visual privacy in visual assistance technologies. In *Proceedings of the 20th International Web for All Conference*. 134–147.
- [95] Lee Stearns, Leah Findlater, and Jon E. Froehlich. 2018. Applying Transfer Learning to Recognize Clothing Patterns Using a Finger-Mounted Camera. In *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility (Galway, Ireland) (ASSETS '18)*. Association for Computing Machinery, New York, NY, USA, 349–351. doi:10.1145/3234695.3241015
- [96] Kaiyuan Tang, Kerui Chen, Tian Qing, Zhou Jiang, Qiuyu Lin, and Tianming Xu. 2025. EchoMall: Prototyping GenAI-powered Vision-Free Online Shopping Experience for Blind and Low-Vision Customers. In *Proceedings of the 27th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '25)*. Association for Computing Machinery, New York, NY, USA, Article 133, 5 pages. doi:10.1145/3663547.3759758
- [97] Mengjie Tang, Xinman Li, Juxiao Zhang, Franklin Mingzhe Li, and Zhuying Li. 2026. Understanding Nature Engagement Experiences of Blind People. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems (CHI '26)*. Association for Computing Machinery, New York, NY, USA, Article 466, 21 pages. doi:10.1145/3772318.3790515
- [98] Ningzhi Tang, Chaoran Chen, Zihan Fang, Gelei Xu, Maria Dhakal, Yiyu Shi, Collin McMillan, Yu Huang, and Toby Jia-Jun Li. 2026. Programming by Chat: A Large-Scale Behavioral Analysis of 11,579 Real-World AI-Assisted IDE Sessions. *arXiv preprint arXiv:2604.00436* (2026).
- [99] Xinru Tang, Ali Abdolrahmani, Darren Gergle, and Anne Marie Piper. 2025. Everyday Uncertainty: How Blind People Use GenAI Tools for Information Access. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 63, 17 pages. doi:10.1145/3706598.3713433
- [100] Rune Thorsen, Federico Bortot, and Antonio Caracciolo. 2021. From patient to maker-a case study of co-designing an assistive device using 3D printing. *Assistive technology* 33, 6 (2021), 306–312.
- [101] Simon Torka and Sahin Albayrak. 2024. Optimizing AI-Assisted Code Generation. *ArXiv abs/2412.10953* (2024). <https://api.semanticscholar.org/CorpusID:274777259>
- [102] Kellie Vella, Madeleine Dobson, and Margot Brereton. 2025. "Hello, Mr Tree": Toying with Playful Conversational AI in the Early Years. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA '25)*. Association for Computing Machinery, New York, NY, USA, Article 8, 6 pages. doi:10.1145/3706599.3719878
- [103] Tobias M Weinberg, Ricardo E. Gonzalez Penuela, Stephanie Valencia, and Thijs Roumen. 2026. I, Robot? Exploring Ultra-Personalized AI-Powered AAC; an Autoethnographic Account. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems (CHI '26)*. Association for Computing Machinery, New York, NY, USA, Article 209, 16 pages. doi:10.1145/3772318.3790310
- [104] Tobias M Weinberg, Kowe Kadoma, Ricardo E. Gonzalez Penuela, Stephanie Valencia, and Thijs Roumen. 2025. Why So Serious? Exploring Timely Humorous Comments in AAC Through AI-Powered Interfaces. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 759, 19 pages. doi:10.1145/3706598.3714102
- [105] Tobias M Weinberg, Claire O'Connor, Ricardo E. Gonzalez Penuela, Stephanie Valencia, and Thijs Roumen. 2025. One Does Not Simply 'Mm-hmm': Exploring Backchanneling in the AAC Micro-Culture. In *Proceedings of the 27th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '25)*. Association for Computing Machinery, New York, NY, USA, Article 49, 14 pages. doi:10.1145/3663547.3746381
- [106] Linda Yilin Wen, Cecily Morrison, Martin Grayson, Rita Faia Marques, Daniela Massiceti, Camilla Longden, and Edward Cutrell. 2024. Find my things: personalized accessibility through Teachable AI for people who are blind or low vision. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. 1–6.
- [107] Jeffrey Wong and Jason I Hong. 2007. Making mashups with marmite: towards end-user programming for the web. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 1435–1444.
- [108] Di Wu. 2021. Crippling the history of computing. *IEEE Annals of the History of Computing* 43, 3 (2021), 68–72.
- [109] Jingyi Xie, Rui Yu, He Zhang, Syed Masum Billah, Sooyeon Lee, and John M. Carroll. 2025. Beyond Visual Perception: Insights from Smartphone Interaction of Visually Impaired Users with Large Multimodal Models. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 62, 17 pages. doi:10.1145/3706598.3714210
- [110] Kedi Ying, Mingzhe Tao, Ruize Dai, Ruiping Liu, Karin Müller, Gerhard Jaworek, Jiaming Zhang, and Rainer Stiefelhausen. 2025. DishDetect: What's on my plate? – Co-Designing A Mobile App for Clockwise Food Description. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA '25)*. Association for Computing Machinery, New York, NY, USA, Article 193, 7 pages. doi:10.1145/3706599.3719689
- [111] Nasif Zaman, Venkatesh Potluri, Brandon Biggs, and James M Coughlan. 2025. WhatsAI: Transforming Meta Ray-Bans into an Extensible Generative AI Platform for Accessibility. *arXiv preprint arXiv:2505.09823* (2025).
- [112] J.D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (Hamburg, Germany) (CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 437, 21 pages. doi:10.1145/3544548.3581388
- [113] Cynthia Zastudil, Christine Holyfield, Christine Kapp, Kate Hamilton, Kriti Baru, Liam Newsam, June A. Smith, and Stephen MacNeil. 2025. Helping or Homogenizing? GenAI as a Design Partner to Pre-Service SLPs for Just-in-Time Programming of AAC. In *Proceedings of the 27th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '25)*. Association for Computing Machinery, New York, NY, USA, Article 47, 18 pages. doi:10.1145/3663547.3746384
- [114] Lotus Zhang, Abigale Stangl, Tanusree Sharma, Yu-Yun Tseng, Inan Xu, Danna Gurari, Yang Wang, and Leah Findlater. 2024. Designing Accessible Obfuscation Support for Blind Individuals' Visual Privacy Management. 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 235, 19 pages. doi:10.1145/3613904.3642713
- [115] Zhuohao Zhang, John R Thompson, Aditi Shah, Manish Agrawal, Alper Sarikaya, Jacob O. Wobbrock, Edward Cutrell, and Bongshin Lee. 2024. ChartA11y: Designing Accessible Touch Experiences of Visualizations with Blind Smartphone Users. In *Proceedings of the 26th International ACM SIGACCESS Conference on Computers and Accessibility (St. John's, NL, Canada) (ASSETS '24)*. Association for Computing Machinery, New York, NY, USA, Article 55, 15 pages. doi:10.1145/3663548.3675611
- [116] Yi Zhao, Siqi Wang, Qiqun Geng, Erxin Yu, and Jing Li. 2025. "Less is More": Reducing Cognitive Load and Task Drift in Real-Time Multimodal Assistive Agents for the Visually Impaired. *arXiv preprint arXiv:2511.00945* (2025).
- [117] Feng Zhou, Steven D Benford, Sarah Whatley, Kate Marsh, Ian Ashcroft, Tanja Erhart, Welly O'Brien, and Paul Tennent. 2023. Beyond Skin Deep: Generative Co-Design for Aesthetic Prosthetics. 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 128, 19 pages. doi:10.1145/3544548.3580803
- [118] Zhengping Zhou, Lezhi Li, Xinxin Chen, and Andy Li. 2023. Mini-giants: "small" language models and open source win-win. *arXiv preprint arXiv:2307.08189* (2023).

A Copilot instructions

- From the GitHub agent, tools should be written inside the tools subdirectory which is on the same level as backend and ProgramATApp folders. This is where the code will be drawn from. From VSCode, this does not apply.
- Tool code generated by the GitHub agent in the tools folder should be written in python, not in Typescript. It should run on server, and have results delivered to device. It should return string or dict (not None, not binary).
- From the GitHub agent, write all code in a building block pattern where future generated code can leverage and augment it. Also, use any available building blocks in the generation of the requested code rather than writing everything from scratch. This may involve library functions, which is fine, but each tool should be runnable in its own right as well: it must have a main(), run(), or process_image() function, which take (image, input_data) as parameters. Do not assign these names to any helper functions. Only use them if that is the intended entry point.
- When using building blocks, use their content but do not directly import them as modules because this will not work in the exec context. Build them off each other but have them ultimately be standalone.
- From the GitHub agent, tools must NOT connect to the backend server or use WebSockets. Tools execute ON the backend server and receive data as function parameters. The mobile app handles all network communication. Tools should only process the image and input_data parameters passed to their main() function and return results as strings or dictionaries. From the GitHub agent, ALL tools MUST return audio-friendly output. Tool results are automatically spoken aloud via text-to-speech on the mobile device unless another form of audio is specified. Return values should be:
 - Natural language strings that sound good when spoken (not JSON, not code, not cryptic abbreviations)
 - Concise - Long outputs will take too long to speak
 - Descriptive - User cannot see the screen, audio is the primary interface
 - Action-oriented - Tell the user what was found/detected/processed
- From the GitHub agent, tools can return simple strings OR dictionaries with 'audio' and 'text' keys for advanced control.

Listing 1: Tool structure sample prompt portion.

```
# SIMPLE (recommended for most tools):
def main(image, input_data):
    # Process image...
    return "I found 3 red stop signs ahead" # Spoken via TTS

# ADVANCED (for custom audio behavior):
def main(image, input_data):
    # Process image...
    return {
        'audio': {
            'type': 'speech',          # 'speech', 'beep_high',
            'beep_low', 'success', 'error', 'warning'
```

```
'text': 'Warning! Obstacle 5 feet ahead!', #
What to say
'rate': 1.5,          # Speech speed multiplier
(optional)
'interrupt': True     # Stop current audio (
optional)
},
'text': 'Object detected at 5ft - WARNING' # Visual
display text
}
```

EXAMPLES of good audio-friendly returns:

```
return "No objects detected"
return "Found 5 people in the frame"
return "Text says: Welcome to the building"
return "Warning: Low light conditions"
return "Scanning complete, 12 items found"
```

EXAMPLES of bad returns (not audio-friendly):

```
return {"count": 5, "type": "person"} # Don't return raw
JSON
return "obj_det_conf_0.95" # Don't use abbreviations
return None # Don't return None
return "" # Don't return empty strings
```

- From the GitHub agent, when tools detect critical information (obstacles, warnings, errors), use the advanced dictionary format with appropriate audio types:
 - Urgent warnings: 'audio': 'type': 'beep_high', 'text': '...', 'rate': 1.5'
 - Errors: 'audio': 'type': 'error', 'text': '...'
 - Success confirmations: 'audio': 'type': 'success', 'text': '...'
 - Normal information: Just return a string (auto TTS)
- When writing code from the GitHub agent, avoid GPU based packages unless strictly necessary.
- For certain types of tasks, there are preferred models unless the user has specified otherwise. For object detection generally, this is to use Yolo11 and COCO. For detecting a specific object, it depends on if the object is in the COCO classes, as described below.

Listing 2: COCO Classes

```
COCO_CLASSES = [
    'person', 'bicycle', 'car', 'motorcycle', 'airplane', '
    bus', 'train', 'truck',
    'boat', 'traffic light', 'fire hydrant', 'stop sign', '
    parking meter', 'bench',
    'bird', 'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant
    ', 'bear', 'zebra',
    'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', '
    suitcase', 'frisbee',
    'skis', 'snowboard', 'sports ball', 'kite', 'baseball
    bat', 'baseball glove',
    'skateboard', 'surfboard', 'tennis racket', 'bottle', '
    wine glass', 'cup',
    'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple', '
    sandwich', 'orange',
    'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake
    ', 'chair', 'couch',
    'potted plant', 'bed', 'dining table', 'toilet', 'tv', '
    laptop', 'mouse',
```



```
'remote', 'keyboard', 'cell phone', 'microwave', 'oven',
'toaster', 'sink',
'refrigerator', 'book', 'clock', 'vase', 'scissors', '
teddy bear', 'hair drier',
'toothbrush'
```

- If the object they are looking for is in the COCO classes, still use Yolo11 and COCO. If it is not, use YoloWorld. For OCR or text extraction, use the Google Cloud Vision API, whose key can be accessed in environment as GOOGLE_APPLICATION_CREDENTIALS For general LLM tasks not described above, Gemini is available, and its API key can be accessed in environment as GEMINI_API_KEY Never use gemini-2.0-flash, if you would use that model, instead use gemini-3-flash-preview.
- For tools where the custom GPT field is filled to indicate that is what they want, prioritize Gemini live over anything specified above. For these tools, arrange for the specified query to be re-pushed without the user needing to say anything new every 5 seconds.
- From the GitHub agent, generate all code to be runnable within the app, which is described in the ProgramATApp subfolder.
- From the GitHub agent, all code should be able to use the camera feed provided by the app for visual processing.
- For any tools that ask for clock face navigation, output should be restricted to 1-3 and 9-12 (others cannot reasonably be captured as they would be behind the camera).
- Optimize for doing your generation quickly. Extensive documentation after the code is written and tested is not wanted or needed.

B GitHub Web Interface

Users can also create and iterate on bespoke AT from the GitHub web interface, as shown in Figure 4.

C Example prompts and iterations

C.1 D1's hand gesture interpreter

C.1.1 Initial prompt. This prompt was sent from mobile:

I'd like a tool to give me basic scene descriptions and text to speech for videos i'm watching. I want to point my phone at a laptop screen and have the tool speak in real-time what is happening in the video. Keep the descriptions to a minimum and prioritize text that appears on the screen. I don't want the tool to read subtitles. This is for text onscreen that I would not be able to hear from the spoken portion of the video.

C.1.2 Iteration prompts.

- (1) I changed my mind about a couple things. I want you to make it support live mode like a custom gpt, and i want you to trigger haptic feedback whenever there is something interesting happening visually that I might be missing. So rather than be told basic visual descriptions, I want haptic feedback to play whenever there is information i might be missing without even knowing it, so that i can pause the video and get the basic visual description that way.

- (2) The on-text reading does not trigger often enough.
- (3) Currently, the tool reads all text aloud on screen. Ignore parts of the screen that are not part of the main video frame. Use edge detection to find the portion of the feed that is the main video and only read aloud text that's part of the video.
- (4) Remove all functions and code related to the text to speech portions of the tool. Now i only want the tool to provide haptic feedback when there is something of interest happening onscreen and allowing me to ask the tool what is happening onscreen and get a basic visual description.
- (5) Remove the haptic feedback conditional. Just provide basic on screen visual descriptions - real-time.
- (6) Restart the whole project. I want a tool that describes basic hand motions that is being used by a person in a video. I want to point my phone at the computer screen and have the tool in real-time describe hand gestures a person is using while speaking.
- (7) This tool is working well so far. When describing the hand motions, also include information about the position of the wrist, palm, and arms.

C.2 D5's business card reader

C.2.1 Initial prompt. This prompt was sent from the web:

Feature Description

This tool extracts key pieces of information from an image of a card.

Problem It Solves Blind individuals often need to identify cards whether financial cards like credit card, debit card, etc. or cards for stores, transport, etc.

Proposed Solution

Given an image first detect if it is a card or not. If it is not a card, inform the user that No Card was Found If it is a card, check if the text on it is readable, if not readable, give guidance to the user on how to take a better picture i.e. go to better lighting, left part is cutoff so move phone to left, etc. If text is readable, extract the brand and key details. Be careful to not output sensitive information i.e. CVV number, SSN, etc. Decide what is safe to output like last 4 digits of card number, expiry, person's name, etc.

Implementation details

If not specified otherwise above, tools for object detection should either utilize Yolo11 and COCO or YoloWorld, based on the conditions described in the copilot instructions. Tools involving text extraction should utilize the Google Cloud Vision API.

Alternatives Considered

Example usage

User sends an image of a credit card

Output:

- Bank of America Card
- Card Holder: John Smith
- Card Number ends with 1234

Custom GPT No

GPT Query

Additional Context

Unless otherwise specified, in streaming mode, any verbal/text response should be limited to 15 words. No such limit applies to one-shot output.

Desktop AT Creation Workflow:

Development Mode:

Tool Creation, Testing, & Iteration

(Branches/ Issues)

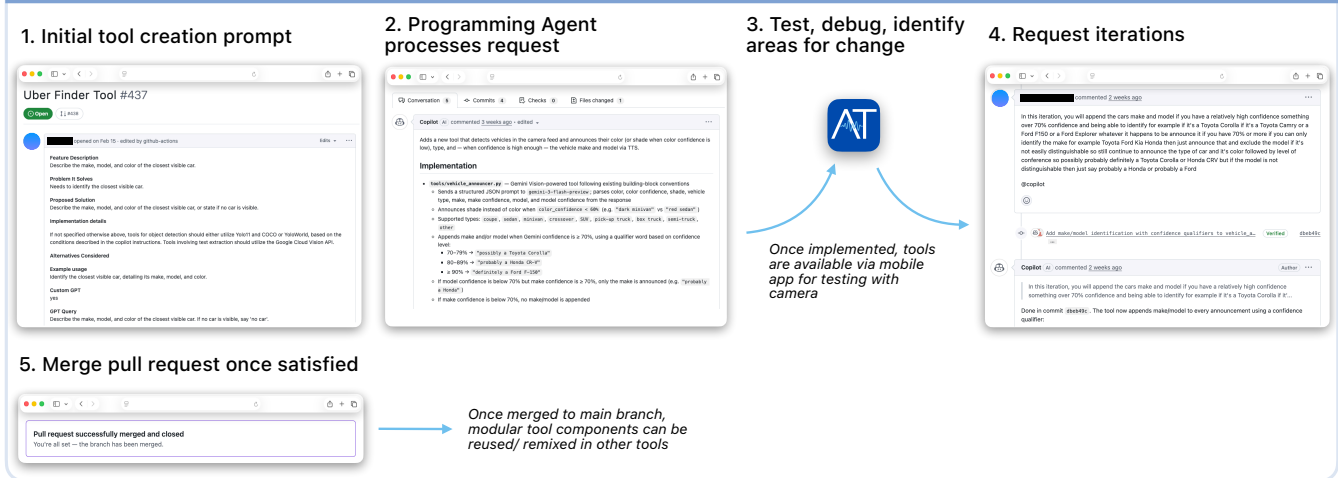


Figure 4: Bespoke AT creation process from the GitHub web interface. (1) Open an issue to create the initial prompt. (2) Programming agent processes request by opening a pull request. (3) Test the implemented tool from the mobile app to use the camera. (4) Request iterations by commenting on the open pull request. (5) Once satisfied, merge the pull request to main.

Write the code for this tool inside the tools folder

C.2.2 Iteration prompts.

- (1) Tested the tool but it keeps saying No Card Detected, may be along with no card detected message it should describe what it sees instead in 5-10 words for user to know what was captured and can adjust the camera accordingly. @copilot please address this.
- (2) Your notes say "Card detection — Uses YoloWorld (yolov8s-world.pt) with custom classes ['credit card', 'debit card', 'card', 'bank card', 'id card'] since none of these are in the COCO dataset. If no card is detected the user is told to hold a card in front of the camera.", that might be the reason for too many no card detected, can we make the tool flexible for lots of card types instead of a fixed set of known categories?
- (3) I still don't see scene description when card detection fails. I get "No Card Found ...in front of the cameara" generic message. When this message is reached, scene description should be appended.
- (4) Still doesn't work, lets start over by just using the same model as the make up checker tool. Remove the 55 category dataset, custom lighting checking algo, etc. just leverage the vision API with the prompt
- (5) I am getting a message "Unable to ... at this time". Can you find bugs and address them?

C.3 D2's kitchen scale reader

C.3.1 Initial Prompt. This prompt was sent from mobile:

You are a helpful, visual assistant with the purpose of reading the current value and measurement on a kitchen scale so it might be in

ounces. It might be in grams. It might be in pounds. You will first read the number for example 1 lbs. 1 oz. 1 g or 0.1 pound or whatever it happens to be one detected you will only read the number and you will only read the number followed by the measurement amount, whether that's gram ounce or pound you will not read anything else for example the names of the buttons or anything else if it reads zero you read zero followed by the measurement for example 0 ounces 0 g or 0 pounds.

C.3.2 Iteration Prompts.

- (1) Add Gemini Live support it should act like a custom GPT so minimize the number of prompting and all that good stuff
- (2) Simply have the tool look for a number assume that the number is in grams so read the number followed by the word grams for example 0 g 100 g 1 g 5 g 20 g etc. It should pronounce grams not just say g
- (3) Remove the Gemini life support and redo this app entirely by making it assume that what we're looking at is a scale set to grams so any number you see just append the word grams to it so for example 0 g 10 g 100 g 1000 g so on and so forth make sure to pronounce the word grams instead of just G

D Tools created

Description of all created tools not described in Table 2 are available in Table 3.

E Co-designer demographics

Co-designer age and vision demographics are available in Table 4.

Tool	Creator	Motivation	Alt. Approach	Mode	App/Web
Physical UI Reader	D1	Get assistance navigating a physical UI, such as buttons on a microwave	Memorize layout of personal appliances	Streaming	App
Vlog frame analyzer	D1	Help achieve a well framed, clutter-free, camera shot	Sighted assistance	Streaming	App
Rubiks cube guidance	D1	Understand the current face of a Rubiks cube to solve it	Call an Aira agent	Streaming	App
Cursor finder	D1 *, D4	Understand the position of a cursor on computer screen, especially if screen reader freezes	Click and see what opens	Live	Web
Dial reader	D1	Interpret the position of a dial on an appliance such as an oven	Call sighted assistance (BeMyEyes or Aira)	Take photo	App
Stain finder	D1	Identify stain or grime on walls while cleaning	Combination of tactile interaction and sighted assistance tools	Streaming	App
Washing machine dial reader	D2	Interpret the position of a dial on a washing machine	Use sighted assistance or image descriptions	Take photo	App
Radio screen reader	D2	Read the content of a hobbyist radio's digital screen	Use sighted assistance or image descriptions	Streaming	App
Video describer	D3	Describe a video's content including action, subtitles, and content like release date	Use sighted assistance	Streaming	Web
Object distance for indoor navigation	D3	Determine the distance from large physical objects while navigating indoors	Physical tools (e.g. cane)	Streaming	App
Outfit match validator	D3	Determine if an outfit matches and is situationally appropriate for the days events (e.g. casual vs business)	Sighted assistance	Take photo	Web
Clothes stain checker	D4	Determine if a clothing item has a stain	Use BeMyAI description	Take photo	App
Object locator	D4	Identify common physical objects and help a user navigate to them	Sighted assistance	Streaming	App
Sock matcher	D5	Determine if two socks from the laundry match	Ask BeMyAI	Take photo	Web

Table 3: All tools created in our study not referenced in Table 2. Table includes a description of the tool created, the co-designers' alternative or current approach to accomplish the same task (i.e., what strategy they used without the bespoke AT, if any), the mode of the tool (live, streaming, or photo), and how the tool was created (via the app or web interface). If multiple co-designers made the same tool, details correspond to the co-designer marked with asterisk

ID	Age	# Phases	Visual acuity	Age of vision loss
D1	25	2	One eye totally blind, light perception in other	14
D2	28	2	One eye totally blind, some usable vision in other	birth
D3	42	1	Minimal residual vision, some light perception	birth
D4	27	1	Minimal residual vision, some light perception	1
D5	33	1	Totally blind	11

Table 4: Co-designer demographics