Check for updates

Development of an LLM-Based Physical Computing Programming Tool through Camera-Based Synchronization

Sangwoo Kim[1], Wonsup Kim[2*]

[1]Department of Industrial Design, Master's Student, SEOULTECH, Seoul, Korea

[2]Department of Industrial Design, Professor, SEOULTECH, Seoul, Korea

Abstract

Background In physical computing, repeated circuit construction and programming demand technical knowledge. Synchronization, the matching of circuit configuration information between circuit construction and programming, requires users to convey information already present in the physical circuit and can involve repeated description and omission. Circuit and behavior changes lead to code revision, so the burden arises repeatedly. In programming with large language models (LLMs), users convey the information to the AI in natural language, so synchronization depends on re-encoding the circuit into language. Prior tools acquire information from the physical circuit for construction or verification, while programming-oriented tools acquire it from the user's language. Support for synchronization during programming, with acquisition from the physical circuit, is needed.

Methods This study designed CircuitPrompter, which recognizes input/output (I/O) devices in a camera-captured circuit image through an LLM and retains circuit configuration information as interactive bounding boxes (IBBs) once the user registers pin numbers, so image recognition and interface interaction replace writing the information in natural language. Six Industrial Design and Computer Science majors with physical computing coursework performed a three-stage programming task in a within-subjects design comparing a baseline (Arduino IDE with a general-purpose conversational AI) with CircuitPrompter. The evaluation measured task completion time, prompt input volume, NASA-TLX responses, and post-task interviews.

Results All six participants completed all stages in both conditions and achieved the same intended functionality. With CircuitPrompter, mean task completion time and prompt morpheme count were 20.9% and 43.4% lower, respectively, with both decreasing for all participants. Reductions in prompt input were concentrated in I/O device names and measurement-procedure vocabulary, while behavior-directing vocabulary was similar across conditions. Circuit-configuration description errors decreased from three cases in the baseline to none with CircuitPrompter. Composite NASA-TLX workload was lower for five participants, while self-rated performance was similar. In the stage involving sensor measurement and circuit-operation verification, all participants used retained information, and CircuitPrompter resulted in lower prompt morpheme counts and implementation times, with no revisions.

Conclusions CircuitPrompter's acquire-retain-reuse approach reduces repeated natural-language description and circuit-configuration omissions while preserving task completion in the evaluated tasks. CircuitPrompter is particularly useful when physical measurements had to be incorporated into subsequent programming and verified through circuit operation. The findings are limited to six participants using one unchanged, preassembled eight-device circuit. Within this scope, the study provides empirical evidence for retaining circuit-configuration information as context shared by users and AI to support synchronization between a physical circuit and LLM-based programming.

Keywords LLM-Based Physical Computing Programming, Circuit Image Recognition, Human-AI Interaction Design, Cognitive Load, Physical Prototyping

Copyright : This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/3.0/), which permits unrestricted educational and non-commercial use, provided the original work is properly cited.

*Corresponding author: Wonsup Kim (wskim@seoultech.ac.kr)

Citation: Kim, S., & Kim, W. (2026). Development of an LLM-Based Physical Computing Programming Tool through Camera-Based Synchronization. *Archives of Design Research, 39*(4), 71-92.

http://dx.doi.org/10.15187/adr.2026.11.39.4.71

Received : Aug. 04. 2026 ; **Reviewed :** Aug. 31. 2026 ; **Accepted :** Sep. 14. 2026
pISSN 1226-8046 **eISSN** 2288-2987

1. Introduction

Designers have used AI to broaden the range of prototype implementation they carry out themselves (Designer Fund, 2026). The broadening has centered on software and can extend to physical computing prototyping, which combines hardware with software.

Physical computing, a field that developed from interaction design, is the building of tangible objects that interact with the physical world by combining sensors and actuators with a microcontroller such as an Arduino (Przybylla & Romeike, 2014). Building prototypes by repeating circuit construction and programming (Dougherty, 2012; O'Sullivan & Igoe, 2004) demands technical knowledge of makers, and the demand is a burden where proficiency is low (H. Kim, 2020). During programming, users must synchronize circuit configuration information between the physical circuit and the code. Planning how the devices should behave constitutes part of the intrinsic cognitive load of the physical computing task, whereas the way users convey circuit configuration information induces extraneous cognitive load (Chandler & Sweller, 1991; Sweller, 1988). This extraneous load arises as users identify the types, placement, and wiring relationships of components in the circuit and transfer this information into the code on the screen. This is because moving back and forth between the two spatially separated representations adds to working-memory load (Mayer, 2001). As circuit modification and code revision are repeated, the user's burden increases.

Tools in prior research retain circuit configuration information, and the user makes use of the retained information while assembling the circuit or checking whether the circuit works as designed (e.g., Y. Kim et al., 2020; Wu et al., 2017). Most tools that recognize circuits require dedicated hardware, while the tool that operates during programming receives the circuit configuration information from the user in language (Y. Kim, Choi, Kang, et al., 2019). Programming with large language models (LLMs) has raised code-writing performance (Kazemitabaar et al., 2023), but the means of synchronization has shifted to natural language. The shift makes the generated result sensitive to the wording of the description (Jansen, 2023), and code generation depends on the system information the user provides (Englhardt et al., 2024).

This study aims to reduce the extraneous cognitive load on makers by supporting the synchronization of circuit configuration information in LLM-based physical computing programming. Simplifying cumbersome work in physical computing prototyping shifts the focus from how to make to what to make (C. Kim, 2024), and this study pursues the same kind of shift, from describing the circuit to deciding the behavior. Replacing the process of writing and conveying circuit configuration information in natural language with an image of the actual circuit reduces the burden of writing and the ambiguity of conveying. CircuitPrompter was therefore designed and implemented as a tool in which the LLM recognizes I/O devices in a circuit image and treats the devices as interactive objects so that the user can reuse the objects during programming.

CircuitPrompter targets the Arduino and breadboard environment, where a circuit is easy to reconfigure repeatedly. A within-subjects evaluation then compared CircuitPrompter with

the Arduino IDE and a general-purpose conversational AI in efficiency, effectiveness, and satisfaction.

2. Related Work

2. 1. Prior Research on Circuit Configuration Synchronization

Synchronization between circuit construction and programming is directional. The virtual-to-physical direction (V2P) is the process of reflecting a circuit designed in software in a physical circuit. The physical-to-virtual direction (P2V) is the process of conveying the configuration information of a physical circuit into virtual space. The burden of synchronization in programming arises after a circuit is assembled, when the configuration information is required for writing code, so the burden belongs to the P2V direction. Support for the P2V direction during programming requires acquisition of the type, placement, and connection relations of the components from the physical circuit, retention of the acquired information within the tool, and reuse of the retained information.

Prior research proposes tools for the synchronization of circuit configuration information in breadboard environments, where a circuit is reconfigured repeatedly (Table 1).

Table 1 Scope of Synchronization Support in Prior Research

Tool	Dir.	Acquire	Retain	Reuse	Required Device
CircuitSense (Wu et al., 2017)	P2V	Tool recognizes components on the breadboard	Tool shows recognized information on a virtual breadboard	User checks schematic and simulation	Sensor–embedded breadboard, waveform generator, PC
HeyTeddy (Y. Kim, Choi, Kang, et al., 2019)		User enters pin numbers and component information	Tool keeps command history as conversation state	Tool runs commands on the board and exports a sketch	SBC–MCU–touchscreen unit, PC or smart speaker
CreativeBreadboard (Park & Choi, 2022)		Tool detects resistors and wiring from a circuit image	Tool outputs the result to the web once	User checks circuit diagram and simulation result	Camera, PC
WireWay (Lertjaturaphat, Rhee, You, & Bianchi, 2026)		Tool measures voltage of each breadboard row	Tool adds measurements to the conversation context	Agent suggests error diagnosis and correction	Augmented breadboard, PC
	V2P	User draws a schematic; tool converts it to circuit configuration data	Tool keeps configuration data and conversation history on a server	Agent suggests placement, wiring edits, and component additions	
VirtualComponent (Y. Kim, Choi, Lee, Lee, & Bianchi, 2019)		User places virtual components and sets values	Tool keeps placement and values in the app	User adjusts and measures component values	Crosspoint–switch breadboard, module–board, tablet
VirtualWire (W. Lee et al., 2021)		User designs component placement and wiring	Tool stores placement and wiring in a design file	User edits wiring; tool applies it to the breadboard	Switching–matrix breadboard, PC
SchemaBoard (Y. Kim et al., 2020)		User draws a schematic; tool computes the layout	Tool stores schematic and layout	User selects a component; tool lights its position	LED–backlit breadboard, tablet, PC

The purpose for which the retained information is used divides the tools further (Table 1). Tools that support circuit construction are the most numerous (Y. Kim, Choi, Lee, et al., 2019; Y. Kim et al., 2020; W. Lee et al., 2021; Lertjaturaphat et al., 2026) and correspond to V2P synchronization, where the user enters the information while designing in software and reuses it when assembling the physical circuit. Tools that support circuit verification (Lertjaturaphat et al., 2026; Park & Choi, 2022; Wu et al., 2017) correspond to P2V synchronization, and most of them acquire the configuration information or the electrical state of the physical circuit through installed dedicated hardware, so the user reads the retained state instead of conveying the information.

The tool whose purpose of information use is programming (Y. Kim, Choi, Kang, et al., 2019) also corresponds to P2V synchronization. When the user conveys component information by voice or text input, the system retains the input as a conversation log and reflects the request on the microcontroller in real time. What the user conveys to the tool is the information of the assembled physical circuit re-encoded into language, so the burden of synchronization remains with the user.

The user revises the code when a change in the circuit alters the type or the pin connection of an I/O device. The user revises the code again when the intended behavior changes while the circuit stays unchanged. Reaching and refining an intended behavior takes repeated code revision as well, since embedded development proceeds through cycles of development and verification (Englhardt et al., 2024). The burden of entering circuit configuration information therefore arises repeatedly during programming. In prior research, however, tools that acquire circuit configuration information from the physical circuit reuse the information for circuit construction or circuit verification, while the tool that reuses the information during programming acquires it from the language of the user (Table 1). Research that supports the synchronization of circuit configuration information during programming, with acquisition from the physical circuit, is needed.

2. 2. LLM-Based Programming and Camera-Based Circuit Synchronization

In conventional programming the user writes circuit configuration information directly into code, while in LLM-based programming the user writes it in natural language and conveys it to the AI. In LLM-based embedded programming, code generation depends on the context the user provides about the system (Englhardt et al., 2024), while small changes in the wording of a device description can produce large changes in the generated result (Jansen, 2023). When the user conveys circuit configuration information to the AI in natural language, the user re-encodes the physical circuit into linguistic symbols, converting the information into a different symbolic system. The AI receives circuit configuration information only through the text input of the user, so the AI identifies the object the user intends only through the re-encoded expression.

In human communication the sender and the receiver carry out referring together, and their expressions grow shorter as they establish a shared reference (Clark & Wilkes-Gibbs, 1986). Visual information shared between the parties supports grounding in collaborative tasks and lowers the effort of referring in language (Gergle, Kraut, & Fussell, 2013). A camera image

of the circuit serves as visual information shared between the user and the AI, so conveying circuit configuration information can depend less on re-encoding. This study designs a tool that acquires circuit configuration information through camera capture, retains it as visual information shared between the user and the AI, and lets the user reuse the retained information in the programming process.

3. Design of CircuitPrompter

The user assembles a circuit with general-purpose electronic components, a breadboard, and an Arduino board, and then carries out programming with CircuitPrompter (Figure 1). The tool acquires information on the input/output (I/O) devices connected to the pins of the board.

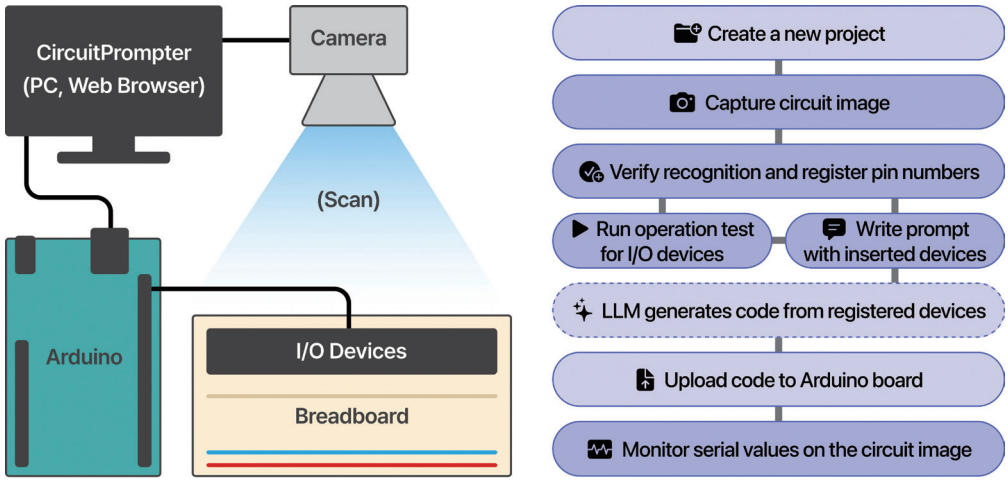

Figure 1 Configuration and Usage Process of CircuitPrompter

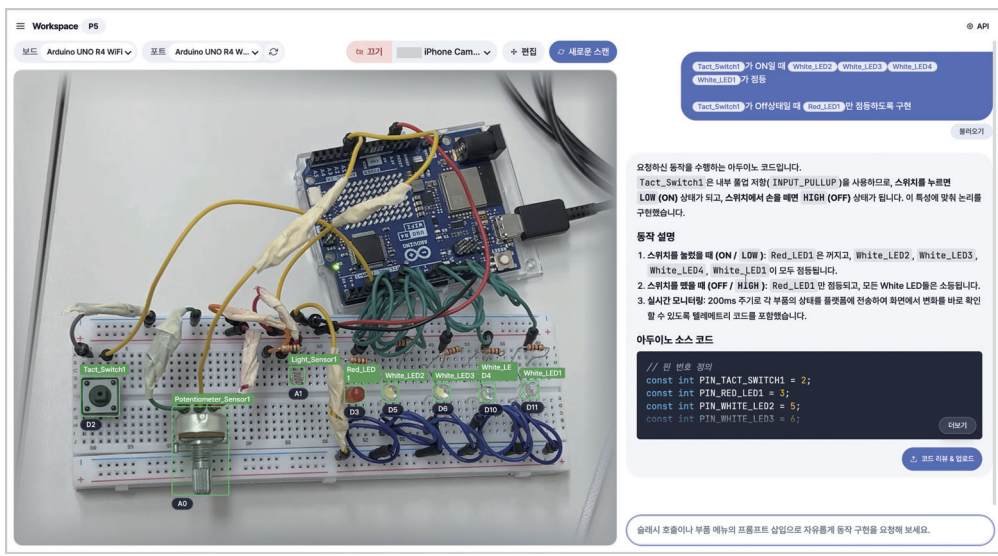

Figure 2 Screen of CircuitPrompter Prototype

The screen consists of a toolbar, an image panel that displays the circuit image, and a prompt chat panel (Figure 2). The tool recognizes the circuit from a single camera image and identifies the type, the number, and the relative placement of the I/O devices. Twisted jumper wires and occluded pin holes lower the accuracy of recognizing pin connections, so the user enters the pin connection information directly.

3. 1. Acquire: Circuit Recognition and I/O Device Registration

CircuitPrompter separates the registration of circuit configuration information from prompt writing. Registration establishes visual sharing of the circuit between the user and the AI, and the user then states the intended behavior in a prompt.

When the user captures the circuit with the scan button on the image panel, the LLM recognizes the I/O devices. The tool marks each recognized I/O device on the image with a bounding box, a rectangular marker of the position and extent of an object. The bounding boxes function as interaction objects that the user selects and manipulates, and are named Interactive Bounding Boxes (IBBs). Editing functions adjust the position of an IBB or delete an IBB when recognition is wrong.

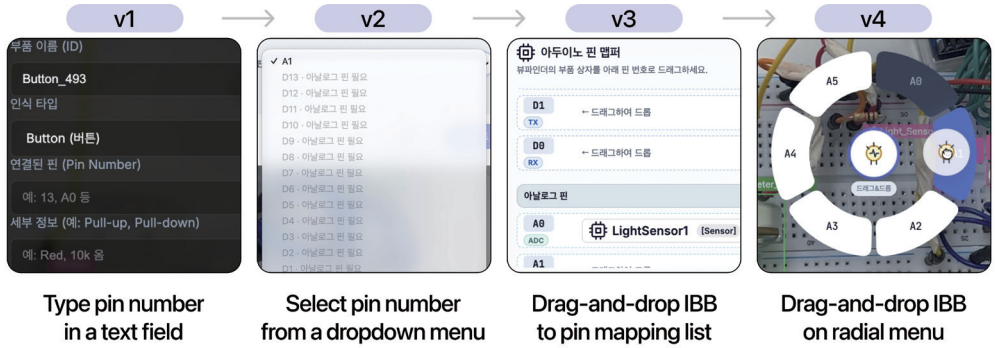

Figure 3 Iterative Design of Registration Input Methods

The user registers the pin number of each recognized I/O device. Dragging an unregistered IBB opens a pin number menu radially around that IBB in the form of a pie menu (Callahan, Hopkins, Weiser, & Shneiderman, 1988), and dropping the IBB on the intended pin number completes the registration. Iterative design across four versions produced the form of the input, replacing text entry with recognition and drag-and-drop selection (Figure 3).

3. 2. Retain: Registered Circuit State

A recognized I/O device first appears as an unregistered IBB with a pink label showing the device type. When the user sets the pin connection information by drag-and-drop, the IBB turns green and shows a name label assigned automatically from the device type together with a badge for the pin connection (Figure 4). Selecting a registered IBB by dragging or through a slash command turns the label purple, so the target of the current operation stays visible.

Once registration is complete, the tool retains the IBB in the image panel as the registered circuit state while the conversation proceeds, and as long as the circuit remains unchanged the user can reuse the registered information repeatedly.

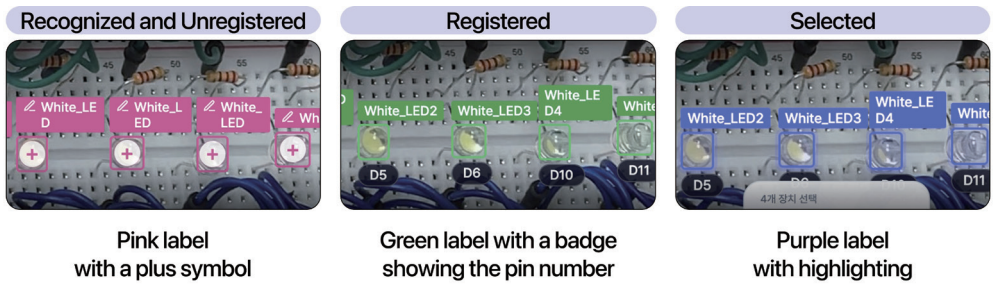

Figure 4 Visual Indicators by IBB State

3. 3. Reuse: Prompt Insertion and Operation Test

The reuse functions insert I/O device information into a prompt and test the operation of an I/O device (Figure 5). Dragging a registered IBB opens a radial menu for the functions, and a slash command inserts a registered IBB in the input field without interrupting text entry. An inserted IBB appears in the prompt as a tag that states the type, the name, and the pin connection information of the device, so the user need not state them separately.



Figure 5 Access Methods for Reuse Functions

The operation test checks whether the I/O devices in the circuit work. The user ordinarily writes test code suited to the device type to check the operation of an I/O device. When the user opens the operation test panel on a registered IBB, however, the LLM reuses the retained information and generates the test code, and the tool uploads the code to the Arduino board immediately. The tool then visualizes the input or output values of the device under test as a real-time graph and as serial values, with a separate graph for each device tested at once (Figure 6).



Figure 6 Operation Test of I/O Devices

When the user writes and sends the intended behavior, the LLM reuses the information on the registered IBBs to generate code. Code generation includes serial monitoring code for every registered device, so the user need not write serial output code separately. The user uploads the code to the board with the upload button, and the tool then displays the serial value of each I/O device on its registered IBB as a Serial-Overlay-Tag (SOT), so the correspondence between value and device stays apparent (Figure 7).

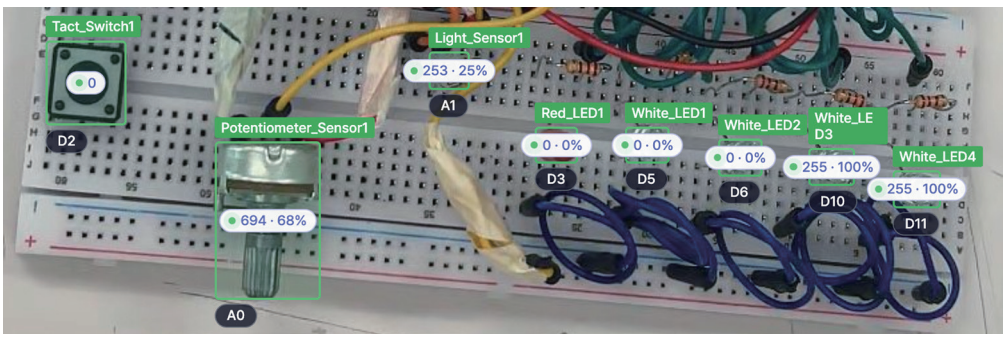

Figure 7 SOT (Serial-Overlay-Tag)

3. 4. Prototype Implementation

This study implemented CircuitPrompter as a web application with OpenAI GPT Codex 5.5. Image capture uses either a webcam connected to a PC or the Continuity Camera function of an iPhone connected to a MacBook, so the prototype operates without dedicated hardware. The prototype uses the Google Gemini-3.5-flash API for image interpretation and code generation and the Arduino CLI API for board connection and code upload, and connects to boards in the Arduino UNO family. The prototype does not include a function for editing code directly, so the LLM generates every code change and this study observes prompt writing without confounding by direct editing.

4. User Evaluation

4. 1. Experimental Design

The user evaluation examined whether CircuitPrompter reduces the burden of synchronizing circuit configuration information (Table 2).

Table 2 Overview of the User Evaluation

Item	Detail
Period	June 1 to June 19, 2026
Design	Within-subjects design comparing the baseline condition (Arduino IDE with a general-purpose conversational AI: Google Gemini) with the CircuitPrompter condition
Participants	Completers of university-level physical computing coursework (4 Industrial Design majors, 2 Computer Science majors)
Assignment	Order of conditions counterbalanced across participants (P1, P3, P5 with the baseline first; P2, P4, P6 with CircuitPrompter first)
Task	Three-stage LED programming that accumulates functions on a single circuit (S1: tactile switch, S2: potentiometer, S3: photoresistor)
Using hardware	Arduino Uno R4 WiFi; a circuit of eight I/O devices assembled on an Arduino Starter Kit breadboard
Measures	Effectiveness (whether the task was completed, whether errors occurred), efficiency (task completion time, prompt input volume), satisfaction (NASA-TLX responses, interview statements)
Data collection	Screen recording, prompt logs, NASA-TLX questionnaire, post-task interview
Analysis	Wilcoxon signed-rank test, Benjamini–Hochberg correction (Benjamini & Hochberg, 1995), Spearman correlation, thematic analysis

4. 1. 1. Participants

This experiment recruited participants who had completed university-level physical computing coursework and used a conversational AI, and every participant reported experience in AI-assisted coding. Physical computing is taught in both Industrial Design and Computer Science, and recruitment from the two majors covers design and engineering backgrounds (Table 3).

Table 3 Participant Profiles

Participant	Gender/ Age	Major / Occupation	Physical computing skill level (1 low – 5 high)	Frequency of conversational AI use (1 low – 5 high)
P1	M / 27	Industrial Design / master's course researcher	3	4
P2	M / 21	Industrial Design / undergraduate	2	4
P3	M / 29	Computer Science / iOS app developer	4	5
P4	F / 22	Industrial Design / undergraduate	2	5
P5	M / 25	Computer Science / web front-end developer	5	4
P6	M / 28	Industrial Design / master's course researcher	2	5

4. 1. 2. Conditions and Task Scenario

The baseline condition represented LLM-based physical-computing programming with a general-purpose conversational AI familiar to participants, in which written natural-language prompts served as the primary means of conveying circuit-configuration information (Table 2). To reflect the range of input behaviors that may occur when such a

service is used in practice, the study did not explicitly restrict alternative input modalities supported by the service, including image and voice input. Both conditions used the same Gemini-3.5-flash model to control for differences in AI code-generation capability, and the memory function was disabled to prevent performance improvements resulting from accumulated conversation history. To focus the experiment on the physical-computing programming process, the researcher assembled the circuit with the wiring arranged in advance and provided each participant with the Arduino board and completed circuit before the experiment.

The task scenario used the red LED as a standby indicator so that a code-driven off state remained distinguishable from a circuit with no code running. The programming task was based on LED control, an introductory task in the Arduino Projects Book (Fitzgerald & Shiloh, 2012), and presented the required behavior for each stage in a structured format. This design was intended to keep the intrinsic task demands associated with understanding and implementing the intended behavior equivalent across conditions, allowing the evaluation to focus on differences in extraneous load arising from how circuit-configuration information was conveyed.

The scenario matched programming in which code is revised while the circuit stays unchanged, and the input devices differed in signal type so the evaluation observed how circuit configuration information was conveyed for each signal type (Table 4).

Table 4 Task Stages

Stage	Task	Observation focus
S1	Toggle four white LEDs with a tactile switch, lighting the red LED when they are off; digital input	First conveyance of circuit configuration information
S2	Adjust the number of lit white LEDs by potentiometer value while keeping the Stage 1 function; analog input	Conveyance on top of an earlier implementation
S3	Adjust white LED brightness by photoresistor value while keeping the Stage 2 function; analog input requiring measurement	Checking and conveyance of sensor measurement values

4. 1. 3. Measures and Analysis

Synchronization in the baseline condition proceeds through writing in natural language, so the measures covered work time and input volume alongside task outcome and subjective workload (Table 5). Prompt input in the CircuitPrompter condition consisted of written sentences and inserted I/O device invocations, so prompt input volume comprises two counts. Participants wrote prompts in Korean, where word and character counts shift with spacing habits while morpheme segmentation stays fixed, so the analysis counted morphemes. The evaluation measured subjective workload with NASA-TLX (Hart & Staveland, 1988) and collected qualitative data through semi-structured post-task interviews.

Table 5 Operational Definitions of Measurement Items

Item	Definition
Total task completion time	From the start of the task to the confirmation of operation 10 s after the final code upload, measured identically in both conditions and rounded to 5 s
Implementation(Impl.) time	Within task time, the portion spent implementing the intended behavior at each stage
Revision(Rev.) time	Within task time, the portion spent correcting errors or mismatches with intent
Prompt morpheme count	Morphemes in the prompt text, counted with Kiwi 0.23.2 (M. Lee, n.d.), excluding symbols and punctuation
I/O device invocation count	Number of IBBs inserted into prompts
Prompt writing time	From the start of prompt entry to submission, summed over all prompts
AI request count	Number of prompts submitted to the AI
Response latency	From prompt submission to the completion of the AI response output
NASA–TLX item score	Score on each of the six NASA–TLX items
Composite workload	Mean of the six NASA–TLX items with Performance reverse–coded
Interview statement	Sentence unit segmented from post–task interview speech

Because individual differences are large in a small-sample experiment, the analysis centered on within-participant differences between conditions. The comparison of task completion time between conditions used the two-tailed paired Wilcoxon signed-rank test with the Benjamini-Hochberg correction (Benjamini & Hochberg, 1995), and the effect size $r = Z/\sqrt{N}$. Segmentation of the interview speech into sentence units made omitted subjects and objects explicit, and thematic analysis identified the experiences that recurred across participants.

4. 1. 4. Procedure

The experiment took about 90 minutes per participant and followed a fixed procedure (Table 6). The settings were a laboratory (P1, P4, P6), an office (P2, P5), and a cafe study room (P3), each a quiet, well-lit indoor space. Each participant performed both conditions in the same setting, and each session was one-on-one. An iPhone on a tripod faced the circuit and served as the camera for a MacBook through Continuity Camera (Figure 8).

Table 6 Experiment Procedure

Phase	Step	Detail
Preparation	Study briefing and consent	Study description, consent form, pre–survey
	Circuit familiarization	Provided circuit configuration
	Tool trial	Basic use, pin registration, operation test
	Task review	Programming task instructions
Task (Condition 1)	Programming task	Three stages in sequence
	Subjective workload response	NASA–TLX
Task (Condition 2)	Programming task	Three stages in sequence
	Subjective workload response	NASA–TLX
Wrap–up	Post–task interview	Semi–structured

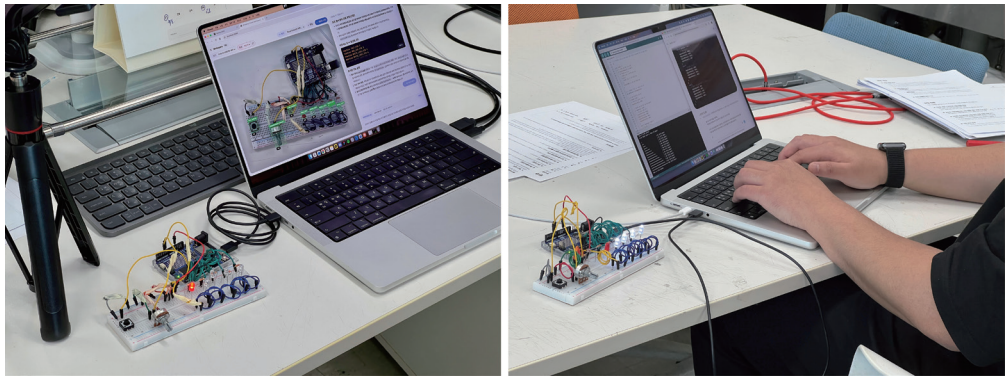

Figure 8 User Evaluation Environment

4. 2. Results

4. 2. 1. Efficiency: Task Time and Prompt Input

Task completion time was shorter in the CircuitPrompter condition for every participant, by a participant mean of 20.9% (Figure 9, Figure 10). Among the stages and work categories, S3 was the only unit in which time was shorter for every participant (Table 7). Prompt morpheme count was also lower for every participant, by a mean of 43.4%, and prompt writing time was shorter for four (Figure 11).

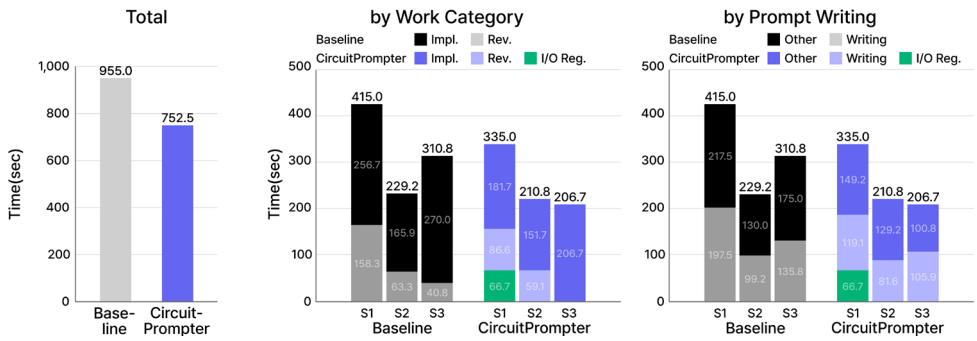

Figure 9 Mean Task Completion Time by Work Category and Prompt Writing

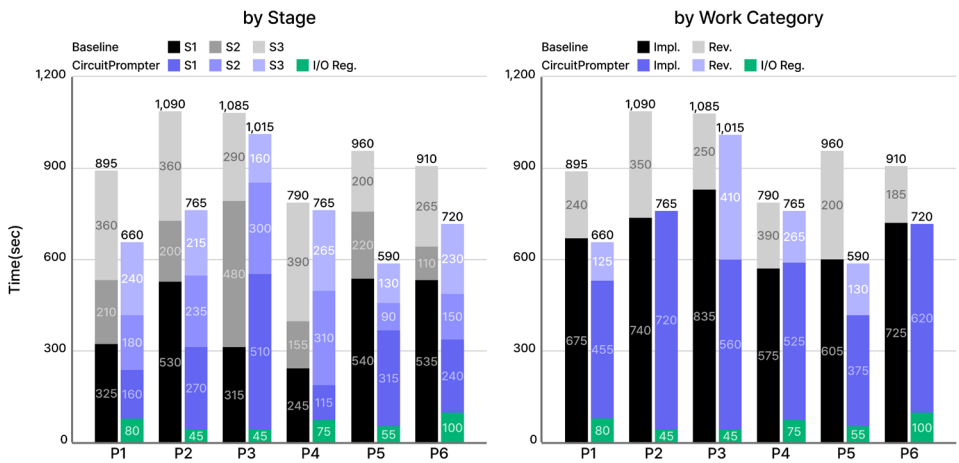

Figure 10 Task Completion Time by Participant

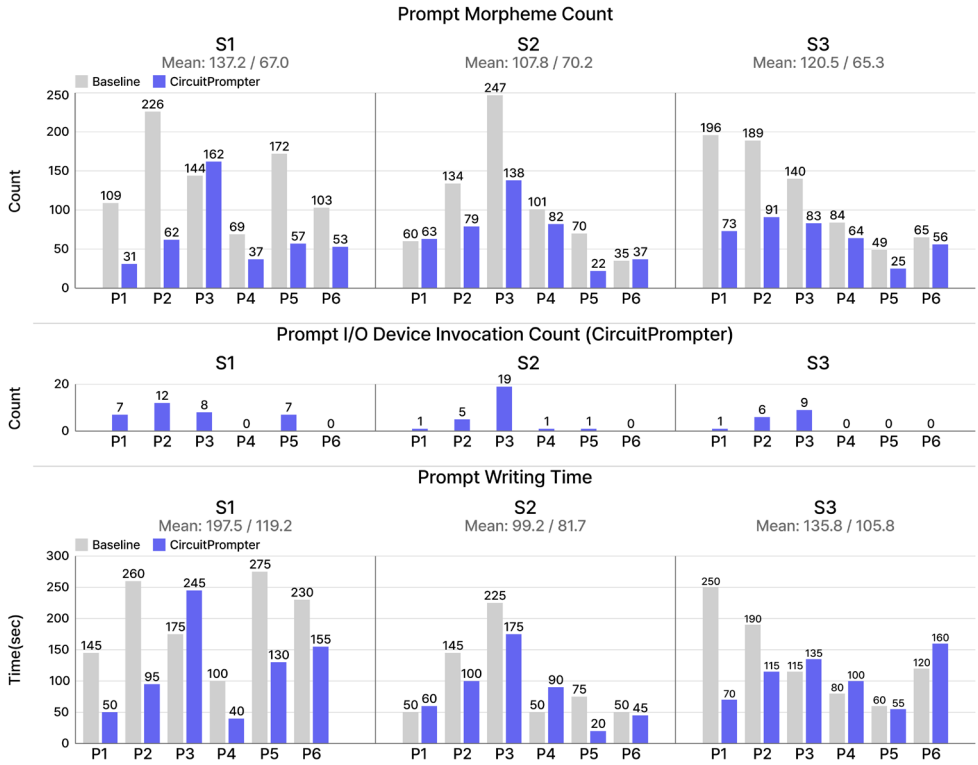

Figure 11 Prompt Input and Writing Time by Participant and Stage

Table 7 Wilcoxon Signed-Rank Test of Task Completion Time between Conditions

Unit	Shorter (CircuitPrompter)	W	p	Adjusted p	r
Total	6/6	0	.031	.093	−1.00
S1	5/6	3	.156	.234	−.71
S2	3/6	10	1.000	1.000	−.05
S3	6/6	0	.031	.093	−1.00
Impl.	4/6	5	.313	.376	−.52
Rev.	5/6	3	.156	.234	−.71

Table 8 Top Morphemes in Prompts by Stage

Stage	Condition	Top morphemes, Korean: English (frequency)
S1	Baseline	led: *led* (35), 흰색: *white* (17), 누르: *press* (14), 상태: *state* (12), 코드: *code* (11), 스위치: *switch* (11), 때: *when* (8)
	CircuitPrompter	때: *when* (9), 누르: *press* (9), on: *on* (8), led: *led* (8), 켜: *turn on* (6), 되: *become* (6), off: *off* (5)
S2	Baseline	가변저항: *potentiometer* (15), led: *led* (14), 켜: *turn on* (13), 조절: *adjust* (8), 상태: *state* (8), 되: *become* (7), 수: *be able to* (7)
	CircuitPrompter	led: *led* (10), 조절: *adjust* (10), 켜: *turn on* (8), 추가: *add* (8), 값: *value* (8), 때: *when* (7), 코드: *code* (7)
S3	Baseline	조도: *illuminance* (25), 센서: *sensor* (21), 코드: *code* (10), 추가: *add* (10), 되: *become* (8), 측정: *measure* (8), 값: *value* (8)
	CircuitPrompter	led: *led* (9), 밝기: *brightness* (9), 때: *when* (8), 조절: *adjust* (6), 조도: *illuminance* (6), 밝: *bright* (5), 코드: *code* (5)

Among the top morphemes by stage, I/O device names and measurement procedure vocabulary appeared at high frequency in the baseline condition and at low frequency in the CircuitPrompter condition, while vocabulary that directs behavior was similar across the conditions (Table 8). The reduction rate was highest in S1, where circuit configuration information was conveyed for the first time, and lowest in S2, where little new information was conveyed (Figure 11). In S3, where a measured value was needed, participants requested photoresistor serial output code from the AI in the baseline condition and read measured values with the SOT or the operation test in the CircuitPrompter condition.

To examine whether more frequent I/O device invocation was associated with greater reductions in prompt amount and writing time, participants with contrasting invocation patterns were first compared. P3 invoked I/O devices 36 times, the highest count among the participants, whereas P6 did not invoke them at all. Nevertheless, their morpheme reduction rates were similar at 27.9% and 28.1%, respectively. These two cases show that similar reductions in prompt morphemes occurred despite markedly different invocation frequencies and that frequent I/O device invocation was not necessary for such a reduction. The participant-level relationships were then examined separately across all six participants using Spearman correlations. Invocation count was not significantly correlated with either the morpheme reduction rate ($\rho = .43$, $p = .397$) or the writing time reduction rate ($\rho = -.09$, $p = .872$). Because each correlation was based on six participant-level observations, the coefficients describe the relationships observed within this sample and are presented as supplementary results.

Registration preceded prompt writing in the CircuitPrompter condition and was included in implementation time (Figure 9). Registration took approximately 67 s on average, comprising 17 s for image recognition and 50 s for pin mapping. Mapping the eight I/O devices therefore required approximately 6.3 s per device on average. During pin mapping, participants identified the Arduino pin connected to each I/O device by examining the physical circuit and then registered the connection in CircuitPrompter. P1, P4, and P6 spent additional time checking pin connections or interacting with the tool during this process, averaging approximately 8.5 s per device after excluding image recognition time. In contrast, P2, P3, and P5 proceeded continuously from identifying each pin connection to registering it, averaging approximately 4.2 s per device.

AI requests averaged 7.8 per participant in the baseline condition and 5.3 in the CircuitPrompter condition. All participants made one initial request in each of the three stages under both conditions, resulting in the same mean of 3.0 initial requests per participant. The difference therefore arose entirely from follow-up requests, which averaged 4.8 in the baseline condition and 2.3 in the CircuitPrompter condition. The largest stage-level difference was observed in S3, where follow-up requests averaged 1.7 and 0.3, respectively, and were fewer with CircuitPrompter for every participant. Despite the difference in request frequency, all participants completed all three task stages and produced the same functional outcome under both conditions. Mean response latency per AI request was 10.7 s in the baseline condition and 17.5 s in the CircuitPrompter condition. CircuitPrompter was a prototype that called the Gemini API, resulting in longer response latency than the commercial Gemini service used in the baseline condition.

4. 2. 2. Effectiveness: Task Completion and Errors

All participants implemented the functions required in all three stages and completed the tasks under both conditions. In the CircuitPrompter condition, the system correctly identified all eight I/O devices in the preassembled circuit for every participant. Revision time was shorter with CircuitPrompter for five participants and longer only for P3. Although P3 had the same number of errors in both conditions and a shorter implementation time with CircuitPrompter, correcting an AI-generation error in S1 took longer (Figure 10). No participant performed revision work in S3 with CircuitPrompter (Figure 12).

Table 9 Revision Cases and Time by Cause

Cause	Baseline cases	Baseline time (s)	CircuitPrompter cases	CircuitPrompter time (s)
Circuit configuration description error	3	180 (11%)	0	0 (0%)
AI generation error	9	1,060 (65%)	6	586 (64%)
Intent description error	4	261 (16%)	3	247 (27%)
Feature improvement	2	130 (8%)	1	82 (9%)
Total	18	1,631 (100%)	10	915 (100%)

Mean revision time per case was similar across the two conditions, at 90.6 s in the baseline condition and 91.5 s in the CircuitPrompter condition, whereas substantially fewer revision cases occurred in the CircuitPrompter condition (Table 9). Thus, the reduction in total revision time primarily reflected a decrease in the number of cases requiring revision rather than faster correction of individual errors. In particular, no circuit-configuration description errors occurred in the CircuitPrompter condition because the system required participants to enter pin-connection information during registration, making the circuit configuration available before prompt construction.

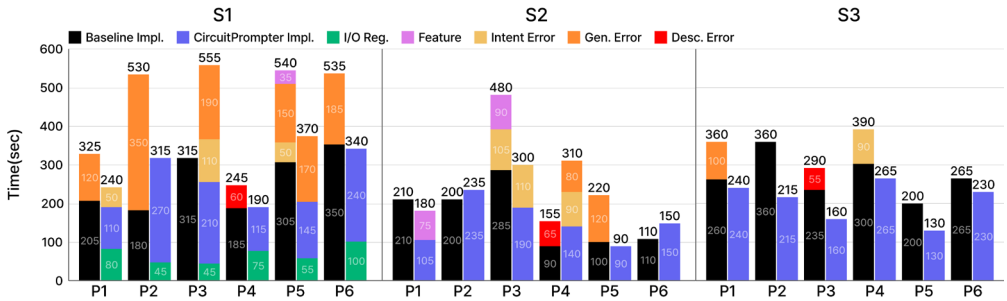

Figure 12 Revision Time by Cause, Participant, and Stage

Revisions caused by AI generation errors accounted for the largest part of revision time in both conditions, and most of that time was in S1 (Table 9, Figure 12). The number of cases was lower in the CircuitPrompter condition for every cause, and intent description errors took a similar time with one case fewer (Table 9).

4. 2. 3. Satisfaction: Subjective Workload and Interviews

Composite NASA-TLX workload was lower in the CircuitPrompter condition for five participants and higher only for P3 (Figure 13). The means for Mental Demand, Effort, and

Frustration were 49%, 40%, and 59% lower in the CircuitPrompter condition, and the mean for self-rated performance was similar across the conditions. In the post-task interview, P3 attributed the higher composite to having done less of the work, to learning a new tool, and to resolving the AI generation errors.

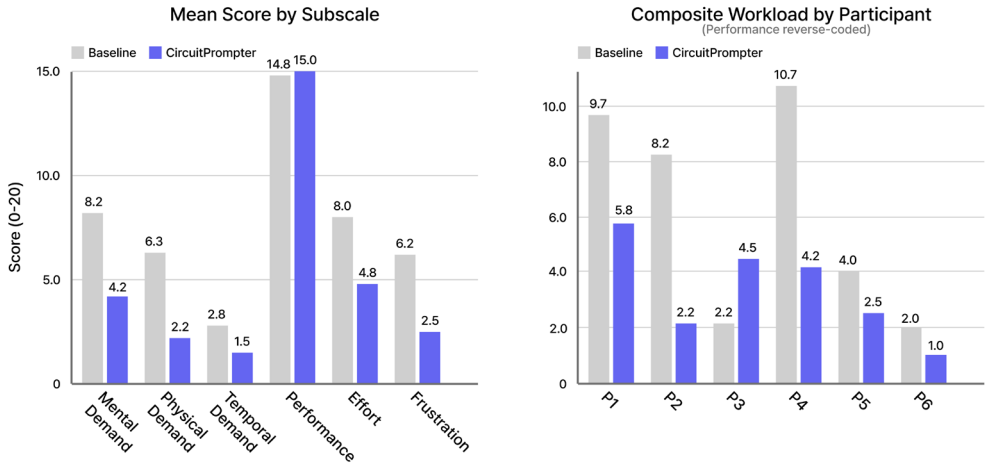

Figure 13 NASA–TLX Subscale Means and Composite Workload by Participant

The analysis drew sixty-five statements about the experience of using the tool from the post-task interviews and classified them into five categories and sixteen themes (Table 10). Among the themes, automation of pin connection entry was the improvement that the most participants raised (P1, P2, P4, P5, P6). Themes in the anticipated usefulness category state what participants expected of the tool beyond the task they performed, and a lowered prior knowledge requirement for novices who cannot distinguish device characteristics is one of them (P1, P6).

Table 10 Post–Task Interview Themes and Key Findings by Category

Category	Theme	Key Finding
Experienced usefulness	Reduced input work based on circuit recognition	Recognition of several I/O devices at once, replacing text entry of device names (P1, P6).
	Trust in retention of registered information and visual sharing	IBB retention lowering the risk of the user forgetting or the AI omitting device information; perceiving that the user and the AI refer to the same circuit configuration information (P1, P4, P5).
	Ease of invocation and convenience of selective reuse	Ease of drag selection and the slash command; flexibility of selecting a particular device to mention in a prompt (P1, P2, P4, P6).
	Visualization of device measurements and operation checks	Checking sensor values without writing serial output code; graph visualization in the operation test; visual confirmation of circuit state (P2, P3, P4, P5, P6).
Experienced inconvenience	Burden of manual pin registration	Assigning pin numbers one by one (P1, P2, P6).
	Mismatch between automatic device labeling and expectation	Some automatically assigned IBB name labels not matching what users expected (P3, P4).
	Constraint on code editing	Even a simple code change requiring an AI request, and the resulting delay in the work (P3, P5).

Anticipated usefulness	Expectation for complex circuits	Reduced burden in grasping and designating I/O devices in complex circuits (P1, P3).
	Convenience in sensor measurement	The operation test and the SOT useful for measuring various sensors (P1, P2, P5).
	Lowered entry barrier for novices	Lowering the prior-knowledge barrier for novices who cannot distinguish device characteristics (P1, P6).
Anticipated limitations	Constraints of the camera recognition environment	Limits on recognition with untidy wiring, in environments where capture is impossible, and with occluded I/O devices (P2, P3, P4, P5, P6).
	Low reliability of camera recognition	No recovery other than recapture when a component is unrecognized; cumbersome adjustment of a misrecognized result (P4, P5).
	Limits of the application scope	Usability at the initial construction stage where no circuit exists (P3, P4); longer pin connection entry as devices increase (P2, P5).
Requests for improvement	Automation of pin connection information entry	Automated entry of pin connection information through wiring recognition (P1, P2, P4, P5, P6).
	Error diagnosis and consistency advice	Identifying whether a behavior failure originates in the code or the circuit; warnings about unsuitable connections and recommendations of components suited to the configuration (P1, P2, P6).
	Extension of work management	Constant access to the code being worked on; account-based saving of the work environment and customization (P1, P3, P5, P6).

4. 3. Discussion

The reduction in prompt input was concentrated in vocabulary describing the circuit, while the amount of vocabulary specifying circuit behavior was similar across the two conditions. In the baseline condition, participants directly described circuit-configuration information in their prompts. In the CircuitPrompter condition, they used information acquired and retained during registration. The correspondence between the reduced vocabulary and the registered information shows that retaining circuit information reduced the need for repeated natural-language description. Subjective workload was lower with CircuitPrompter for five participants, while task outcomes and self-rated performance were similar across the conditions. Taken together, these results show that CircuitPrompter reduced the effort required to convey circuit-configuration information while participants addressed the same behavioral requirements and achieved comparable task outcomes. However, NASA-TLX does not distinguish extraneous cognitive load from other types of cognitive load. The interpretation that extraneous cognitive load decreased is therefore based on the combined patterns in vocabulary use, task performance, and subjective workload rather than on a direct measure of extraneous load.

Participants found it useful that they and the AI could refer to the same circuit-configuration information. They also reported that retaining the registered information reduced their concerns that they might forget a device or that the AI might omit one (P1, P4, P5). CircuitPrompter displayed the registered information to users and provided the same information to the AI, allowing participants to identify which circuit information informed the AI's responses. These comments indicate that a shared visual reference and persistent information supported trust in the AI's use of circuit information. This trust may reduce uncertainty about whether the relevant circuit information has been conveyed to the AI and help users continue interacting with the AI on the basis of a shared circuit context.

The acquired and retained circuit information was also used to inspect photoresistor measurements and verify the operation of I/O devices. In S3, participants checked sensor values through the SOT and used the operation-testing function to verify circuit behavior. Using a device such as a photoresistor requires users to inspect values measured in the physical environment and incorporate them into conditions in the code. Because CircuitPrompter retained the registered photoresistor information, participants could inspect the measured values and apply them to subsequent implementation without re-entering the device information. S3 was the only stage in which prompt morpheme count and implementation time were lower with CircuitPrompter for every participant. This pattern indicates that retained circuit information can support continuity from prompt writing to physical measurement and operation verification. More broadly, the reuse of circuit information can be designed to connect the provision of information to the AI with the measurement and verification of the physical circuit.

Assigning circuit-configuration values and behavioral intent to different input processes reduced omissions of circuit information. In the baseline condition, participants described both the intended behavior of each device and its connected pin number in a single prompt. In three cases, participants specified the intended behavior but omitted the pin number (P4-S1, P4-S2, P3-S3). A pin connection is a discrete value that can be identified from the assembled circuit, whereas behavioral intent is determined by the user and can be expressed in various ways. CircuitPrompter applied structured registration to pin-connection information and natural-language input to behavioral intent. After entering pin numbers during registration, participants described the intended behavior in subsequent prompts. No circuit-configuration description errors occurred in the CircuitPrompter condition (Table 9). These results support assigning input methods according to the characteristics of the information and show that separating discrete circuit-configuration values from behavioral intent can reduce omissions during information synchronization between users and AI.

Unlike circuit-configuration description errors, AI-generated code errors and intent-description errors occurred in both conditions. AI-generated code errors have also been reported in LLM-based embedded programming (Englhardt et al., 2024), and intent-description errors can arise from differences between user intent and the specific expressions required by an LLM (Liu et al., 2023). Structured registration provides device types and pin-connection information, but it does not control the LLM's code-generation process or how users express their intent. Mean revision time per case was similar across the conditions, while the number of revision cases was lower with CircuitPrompter. The reduction in total revision time therefore resulted primarily from fewer cases requiring revision rather than faster correction of individual errors (Table 9). Participants requested support for determining whether remaining errors originated in the code or the circuit (P1, P2, P6). This request suggests a direction for extending the design by connecting registered circuit information with generated code and observed device behavior to support diagnosis of error causes.

Registering circuit information introduced an initial interaction cost. Even when registration time was included, total task completion time was shorter with CircuitPrompter for every

participant (Figure 10). Nevertheless, participants found entering pin numbers individually burdensome (P1, P2, P6), and automating pin-connection entry was the most frequently suggested improvement (P1, P2, P4, P5, P6; Table 10). The system correctly identified all eight I/O devices in the preassembled circuit, but participants entered the pin-connection information manually. This recognition result is limited to one preassembled circuit captured under indoor lighting conditions that did not interfere with camera-based recognition. Automatically acquiring pin connections would require the system to identify connections even when wires overlap or become tangled and when devices or wires obscure connection points. The circuit configuration also remained unchanged throughout the evaluation, so the study did not measure the time required to recapture the circuit image and update the registered information after circuit modifications. Future development should automate the acquisition and updating of pin-connection information and evaluate recognition performance with more diverse circuit configurations and capture conditions.

Although image input was available in the baseline condition, all participants conveyed circuit information through text. Participants reported that they had not considered using a circuit image. The structured programming instructions may have limited exploration of alternative input methods and directed participants toward text input. The comparison in this study is therefore limited to direct natural-language description of circuit information and the provision of information identified from a camera image and registered in CircuitPrompter. Comparisons with other multimodal input methods require separate investigation.

The evaluation involved six Industrial Design or Computer Science majors who had completed university-level physical computing coursework and had experience using conversational AI for coding. Each participant used each condition once. Task completion time was shorter with CircuitPrompter both for participants who used CircuitPrompter first and for those who used the baseline first. However, the corrected comparison between conditions did not reach the conventional level of statistical significance (adjusted $p = .093$). The reported reduction rates therefore describe the pattern observed among these six participants rather than estimate a population-level effect. Evaluating each condition once captured participants' initial experience but did not determine how familiarity and stable usage patterns might develop through repeated use. Future repeated and longitudinal evaluations with users from different backgrounds and experience levels should examine whether the observed patterns persist after users become familiar with the tool.

5. Conclusion

This study aimed to reduce the burden of synchronizing circuit-configuration information between users and AI in LLM-based physical-computing programming. It divided synchronization into the acquisition, retention, and reuse of circuit information and designed and evaluated CircuitPrompter based on this framework. CircuitPrompter identifies I/O

devices in a camera-captured circuit image, retains circuit-configuration information as Interactive Bounding Boxes after users register pin connections, and makes the information available throughout programming. Through this process, image recognition and interface interaction reduce the need to repeatedly describe circuit-configuration information in natural language.

In the within-subjects evaluation, all six participants completed the three task stages in both conditions and produced final outcomes with the same intended functionality. Compared with the baseline condition, mean task completion time was 20.9% shorter and prompt morpheme count was 43.4% lower with CircuitPrompter. Both measures decreased for all six participants. Circuit-configuration description errors decreased from three cases in the baseline condition to none with CircuitPrompter. Mean NASA-TLX ratings for Mental Demand, Effort, and Frustration were 49%, 40%, and 59% lower, respectively. Composite workload was lower for five participants, while self-rated performance was similar across the conditions.

These results show that separating circuit-configuration information from repeated natural-language description reduced the required prompt input and omissions of circuit information while preserving task completion within the evaluated scope. Retained circuit information also supported sensor measurement and circuit-operation verification. When physical measurements had to be incorporated into subsequent programming, every participant showed lower prompt morpheme counts and implementation times with CircuitPrompter, and no revision work occurred.

The findings are limited to six participants performing incremental programming with one unchanged, preassembled circuit consisting of eight I/O devices. Within this scope, this study presents a Human-AI Interaction design approach that supports information synchronization between a physical circuit and LLM-based programming by retaining circuit-configuration information as context shared by users and AI and making it available throughout programming. The evaluation empirically documented reductions in the repeated input and omission of circuit-configuration information when this approach was applied to the examined tasks. By reducing the work and time required to convey circuit-configuration information during programming, this approach is expected to contribute to the rapid-prototyping activities of designers and makers using physical computing.

References

1. Benjamini, Y., & Hochberg, Y. (1995). Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society: Series B (Methodological), 57*(1), 289–300. https://doi.org/10.1111/j.2517-6161.1995.tb02031.x
2. Callahan, J., Hopkins, D., Weiser, M., & Shneiderman, B. (1988). An empirical comparison of pie vs. linear menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 95–100). New York: ACM. https://doi.org/10.1145/57167.57182
3. Chandler, P., & Sweller, J. (1991). Cognitive load theory and the format of instruction. *Cognition and Instruction, 8*(4), 293–332. https://doi.org/10.1207/s1532690xci0804_2
4. Clark, H. H., & Wilkes-Gibbs, D. (1986). Referring as a collaborative process. *Cognition, 22*(1), 1–39. https://doi.org/10.1016/0010-0277(86)90010-7

5. Designer Fund. (2026). *AI in design report 2026*. Retrieved July 31, 2026, from https://stateofaidesign.com
6. Dougherty, D. (2012). The maker movement. *Innovations: Technology, Governance, Globalization, 7*(3), 11–14. https://doi.org/10.1162/INOV_a_00135
7. Englhardt, Z., Li, R., Nissanka, D., Zhang, Z., Narayanswamy, G., Breda, J., Liu, X., Patel, S., & Iyer, V. (2024). Exploring and characterizing large language models for embedded system development and debugging. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems* (Article 150, pp. 1–9). New York: ACM. https://doi.org/10.1145/3613905.3650764
8. Fitzgerald, S., & Shiloh, M. (Eds.). (2012). *Arduino projects book*. Torino, Italy: Arduino LLC.
9. Gergle, D., Kraut, R. E., & Fussell, S. R. (2013). Using visual information for grounding and awareness in collaborative tasks. *Human–Computer Interaction, 28*(1), 1–39. https://doi.org/10.1080/07370024.2012.678246
10. Hart, S. G., & Staveland, L. E. (1988). Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *P. A. Hancock & N. Meshkati (Eds.), Advances in psychology: Vol. 52. Human mental workload* (pp. 139–183). Amsterdam: North-Holland. https://doi.org/10.1016/S0166-4115(08)62386-9
11. Jansen, P. (2023). From words to wires: Generating functioning electronic devices from natural language descriptions. In *Findings of the Association for Computational Linguistics: EMNLP 2023* (pp. 12972–12990). Stroudsburg, PA: Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.findings-emnlp.864
12. Kazemitabaar, M., Chow, J., Ma, C. K. T., Ericson, B. J., Weintrop, D., & Grossman, T. (2023). Studying the effect of AI code generators on supporting novice learners in introductory programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Article 455, pp. 1–23). New York: ACM. https://doi.org/10.1145/3544548.3580919
13. Kim, C. (2024). Educational framework for IoT product and service prototyping. *Archives of Design Research, 37*(2), 103–119. https://doi.org/10.15187/adr.2024.05.37.2.103
14. Kim, H. (2020). Learning from two types of class projects in interactive physical prototyping: Comparison between technology-driven and experience-driven project results. *Archives of Design Research, 33*(3), 75–87. https://doi.org/10.15187/adr.2020.08.33.3.75
15. Kim, Y., Choi, Y., Kang, D., Lee, M., Nam, T.-J., & Bianchi, A. (2019). HeyTeddy: Conversational test-driven development for physical computing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 3(4), Article 139*, 1–21. https://doi.org/10.1145/3369838
16. Kim, Y., Choi, Y., Lee, H., Lee, G., & Bianchi, A. (2019). VirtualComponent: A mixed-reality tool for designing and tuning breadboarded circuits. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Article 177, pp. 1–13). New York: ACM. https://doi.org/10.1145/3290605.3300407
17. Kim, Y., Lee, H., Prasad, R., Je, S., Choi, Y., Ashbrook, D., Oakley, I., & Bianchi, A. (2020). SchemaBoard: Supporting correct assembly of schematic circuits using dynamic in-situ visualization. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (pp. 987–998). New York: ACM. https://doi.org/10.1145/3379337.3415887
18. Lee, M. (n.d.). Kiwi (Version 0.23.2) [Computer software]. Retrieved July 31, 2026, from https://github.com/bab2min/Kiwi
19. Lee, W., Prasad, R., Je, S., Kim, Y., Oakley, I., Ashbrook, D., & Bianchi, A. (2021). VirtualWire: Supporting rapid prototyping with instant reconfigurations of wires in breadboarded circuits. In *Proceedings of the Fifteenth International Conference on Tangible, Embedded, and Embodied Interaction* (Article 4, pp. 1–12). New York: ACM. https://doi.org/10.1145/3430524.3440623
20. Lertjaturaphat, P., Rhee, J., You, J., & Bianchi, A. (2026). Wire Your Way: Hardware-contextualized guidance and in-situ tests for personalized circuit prototyping. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems* (Article 1471, pp. 1–19). New York: ACM. https://doi.org/10.1145/3772318.3791371

21. Liu, M. X., Sarkar, A., Negreanu, C., Zorn, B., Williams, J., Toronto, N., & Gordon, A. D. (2023). "What it wants me to say": Bridging the abstraction gap between end-user programmers and code-generating large language models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Article 598, pp. 1–31). New York: ACM. https://doi.org/10.1145/3544548.3580817
22. Mayer, R. E. (2001). *Multimedia learning*. Cambridge, UK: Cambridge University Press. https://doi.org/10.1017/CBO9781139164603
23. O'Sullivan, D., & Igoe, T. (2004). *Physical computing: Sensing and controlling the physical world with computers*. Boston: Thomson Course Technology.
24. Park, S. H., & Choi, S.-H. (2022). Web-based breadboard electrical circuit analysis using object detection for educational purpose. *Journal of Digital Contents Society, 23*(9), 1775–1785. https://doi.org/10.9728/dcs.2022.23.9.1775
25. Przybylla, M., & Romeike, R. (2014). Physical computing and its scope – Towards a constructionist computer science curriculum with physical computing. *Informatics in Education, 13*(2), 225–240. https://doi.org/10.15388/infedu.2014.14
26. Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science, 12*(2), 257–285. https://doi.org/10.1207/s15516709cog1202_4
27. Wu, T.-Y., Wang, B., Lee, J.-Y., Shen, H.-P., Wu, Y.-C., Chen, Y.-A., Ku, P-S., Hsu, M-W., Lin, Y. C., & Chen, M. Y. (2017). CircuitSense: Automatic sensing of physical circuits and generation of virtual circuits to support software tools. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (pp. 311–319). New York: ACM. https://doi.org/10.1145/3126594.3126634