

---

# PCEval: A Benchmark for Evaluating Physical Computing Capabilities of Large Language Models

---

Anonymous Author(s)

Affiliation

Address

email

## Abstract

1 Large Language Models (LLMs) have demonstrated remarkable capabilities across  
2 various domains, including software development, education, and technical assistance. However, a fundamental question remains unresolved: can LLMs reason  
3 about physical constraints—the rules governing how abstract computations must  
4 map onto real-world hardware? This capability is distinct from, and arguably more  
5 demanding than, software generation alone, as it requires grounding language  
6 understanding in spatial, electrical, and connectivity constraints. To address this  
7 gap, we introduce PCEVAL (Physical Computing Evaluation), a benchmark and  
8 evaluation protocol for physical computing education that enables fully automatic,  
9 execution-based assessment of LLMs across both logical and physical implementation  
10 artifacts, without human grading. Our evaluation framework assesses LLMs in  
11 generating circuits and producing compatible code across varying levels of project  
12 complexity. Through comprehensive testing of 14 leading models, PCEVAL provides  
13 the first reproducible and automatically validated empirical assessment of  
14 LLMs’ ability to reason about fundamental hardware implementation constraints  
15 within a simulation environment. Our findings reveal that while LLMs perform well  
16 in code generation and logical circuit design, they struggle significantly with physical  
17 breadboard layout creation, particularly in managing proper pin connections  
18 and avoiding circuit errors. PCEVAL advances our understanding of AI assistance  
19 in hardware-dependent computing environments and establishes a foundation for  
20 developing more effective tools to support physical computing education.  
21

## 22 1 Introduction

23 Physical computing is the practice of connecting software with the physical world, typically through  
24 microcontroller platforms such as Arduino that control sensors, actuators, and displays. This connection reveals how implemented code produces tangible effects in the physical world, effectively  
25 bridging abstract computation and real-world objects. This unique characteristic has made physical  
26 computing a rapidly growing element of modern STEM (Science, Technology, Engineering, and  
27 Mathematics) education, valued for fostering creativity, problem-solving skills, and computational  
28 thinking through hands-on experiences (Chung and Lou, 2021; Kastner-Hauler et al., 2022; Araújo  
29 and Saúde, 2025). Consequently, educational institutions worldwide are increasingly offering physical  
30 computing classes to help students interact with technology and understand complex concepts  
31 (El-Abd, 2017; García-Tudela and Marín-Marín, 2023; Schätz et al., 2024).

33 Despite its educational value, physical computing remains difficult to teach and assess. In physical  
34 computing education, teachers must have dual expertise in software and hardware, yet even experienced  
35 educators report challenges in evaluating physical circuits with their small components and  
36 tangled wires (Hyeon et al., 2016; Theodoropoulos et al., 2018). In addition, our interviews with

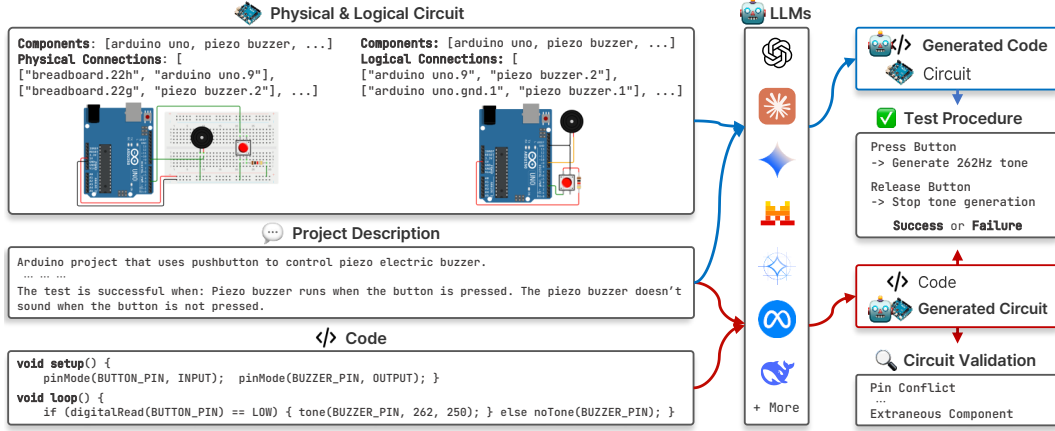


Figure 1: This figure illustrates the core protocols of the PCEVAL benchmark. The left panels show the input options for LLMs, while the right panels show the evaluation workflow. There are basically two types of tasks: code generation (blue arrow) and circuit generation (red arrow). Generated outputs undergo validation through standardized test procedures and circuit validation metrics, measuring both logical correctness and physical implementation feasibility in physical computing systems. (The circuit visualizations are shown to aid understanding. They are not used as inputs to the LLMs.)

37 eight experienced computer science educators (Appendix A) showed that individualized feedback  
 38 and circuit verification represent a widespread burden in classroom practice.

39 *In physical computing, working with circuits is absolutely essential, and that's just something AI*  
 40 *can't help with. These days, tools like Gemini in Colab can write and even suggest code, which is*  
 41 *great. But it's not like an arm can suddenly pop out and build the circuit for you.*

42 - Public High School Computer Science Teacher C, Appendix A. Q4

43 Given these persistent challenges, it is natural to ask whether LLMs can provide meaningful assis-  
 44 tance—yet this first requires understanding whether LLMs can reason under physical constraints  
 45 at all. However, despite the remarkable capabilities of LLMs in software development and coding  
 46 assistance (Annepaka and Pakray, 2024; Wang et al., 2024), their evaluation remains under-developed  
 47 in hardware-dependent settings, particularly when outputs must satisfy execution-level constraints  
 48 induced by physical artifacts. Prior efforts on LLMs for electronics (Jansen, 2023; Yang et al., 2024)  
 49 further highlight this gap, as they relied on labor-intensive manual expert evaluation and largely  
 50 overlooked the physical layout constraints central to educational breadboard implementations.

51 To address this gap, we introduce PCEVAL (Physical Computing Evaluation), the first benchmark  
 52 designed to systematically evaluate whether LLMs can reason under physical constraints—a capability  
 53 we argue is critical for assessing LLM readiness for real-world prototyping and education beyond  
 54 text-only domains. PCEVAL decomposes the challenge into four dimensions: (1) Logical Circuit  
 55 Generation, (2) Physical Circuit Generation, (3) Code from Logical Circuits, and (4) Code from  
 56 Physical Circuits. This structured decomposition enables **scalable and reproducible evaluation**  
 57 **without manual expert intervention**, overcoming the subjectivity and labor costs of prior manual  
 58 assessments.

59 Our evaluation of leading LLM architectures on PCEVAL spans projects of varying complexity, from  
 60 basic I/O operations to intricate multi-component interactive systems. The experimental results reveal  
 61 a clear disparity: while LLMs demonstrate competence in code generation and logical circuit design,  
 62 they struggle considerably with physical circuit generation, particularly in adhering to breadboard  
 63 mechanics and ensuring correct pin configurations. Although mitigation strategies such as self-  
 64 improvement and chain-of-thought prompting provided partial improvements, substantial challenges  
 65 remain, reflecting the difficulty of physical computing tasks. By illuminating these challenges,  
 66 PCEVAL establishes a critical starting point for developing models capable of truly interacting with  
 67 our physical world.

68 This research offers three key contributions:

Table 1: Comparative analysis of related benchmarks, highlighting key differentiators in task scope and evaluation. \*For VerilogEval and ResBench, code generation itself constitutes the logical circuit description (HDL). †EmbedTask’s automated evaluation via HIL testing requires physical hardware setup. §PCEVAL’s code generation assessment uniquely includes tasks for generating code from detailed physical circuit layouts.

| Benchmark                      | Target Domain      | Code Generation | Logical Circuit Generation | Physical Circuit Generation | Automated Evaluation | Validation Method    |
|--------------------------------|--------------------|-----------------|----------------------------|-----------------------------|----------------------|----------------------|
| HumanEval (Chen et al., 2021)  | General SW         | ✓               | ✗                          | ✗                           | ✓                    | Unit Testing         |
| MBPP (Austin et al., 2021)     |                    | ✓               | ✗                          | ✗                           | ✓                    | Unit Testing         |
| VerilogEval (Liu et al., 2023) | HDL Design         | ✓               | ✓*                         | ✗                           | ✓                    | HDL Simulation       |
| ResBench (Guo and Zhao, 2025)  |                    | ✓               | ✓*                         | ✗                           | ✓                    | HDL Sim. & Synthesis |
| EmbedTask (Yang et al., 2024)  | Physical Computing | ✓               | ✗                          | ✗                           | ✓†                   | HIL Testing          |
| MICRO25 (Jansen, 2023)         |                    | ✓               | ✓                          | ✗                           | ✗                    | Manual Evaluation    |
| PCEVAL                         |                    | ✓§              | ✓                          | ✓                           | ✓                    | HW/SW Sim.           |

- **New Evaluation Framework (PCEVAL):** We define and operationalize physical constraint reasoning as a measurable LLM capability distinct from code generation, and introduce an execution-based benchmark protocol establishing a reproducible framework to evaluate both functional correctness and physical constructability in hardware-dependent tasks.
- **Scalable Automated Evaluation:** Our framework eliminates subjective manual assessment through decomposed tasks and simulation-based validation, enabling reproducible evaluation without domain expert intervention.
- **Diagnostic Findings:** Across 14 models and 300 tasks, we reveal a consistent, model-agnostic gap between logical and physical reasoning—identifying pin conflict and breadboard topology violations as primary failure modes—providing a diagnostic basis for future model development.

## 2 Related Work

**AI for Programming.** Early LLM benchmarks focused on algorithmic tasks in Python, later extended to broader domains like data science and web/mobile applications (Chen et al., 2021; Austin et al., 2021; Hendrycks et al., 2021; Lai et al., 2023; Zheng et al., 2024). While these works advanced functional correctness in software, they do not address physical grounding, where correctness depends not only on execution but also on whether the hardware is physically realizable.

**AI for Hardware and Physical Systems.** Recent research has explored LLMs in hardware-related domains, including Verilog/HDL generation (Liu et al., 2023; Thakur et al., 2024), analog circuit design with SPICE-based evaluation (Lai et al., 2025), and PCB layout assistance (Liu et al., 2025). Educational contexts such as Arduino programming have also been considered (Johnson et al., 2024; Subramaniam et al., 2024), but these studies typically limit themselves to small-scale code snippets without evaluating physical circuit design. EmbedTask (Yang et al., 2024) introduced hardware-in-the-loop testing for IoT systems, yet its scope is restricted to validating code against pre-defined circuit representations, bypassing the problem of generating physically valid circuits. MICRO25 (Jansen, 2023) is closest to our work, demonstrating that LLMs can generate electronic device designs from text. However, its evaluation depends heavily on *manual expert judgment*, raising concerns about cost, scalability, and reproducibility. Furthermore, MICRO25 does not address physical circuit generation or assess code generation under physical constraints. In contrast, PCEVAL introduces two key advances: (1) a fully automated and reproducible evaluation pipeline ensuring scalability and consistency; and (2) a principled decomposition of physical constraint reasoning into four measurable subtasks, enabling fine-grained diagnosis of LLM capabilities that prior work conflates or omits.

## 3 PCEval Benchmark

### 3.1 The Interviews and Problem Definition

To develop a benchmark addressing real challenges in physical computing education, we first conducted in-depth interviews with eight experienced CS educators (Appendix A). These discussions revealed several obstacles in physical computing education. Educators consistently highlighted three

critical challenges: (1) time-intensive hardware setup and debugging (A, B, C, E), (2) difficulties providing individualized support across varying student skill levels (A, C, D, E, F, G, H), and (3) the complex interplay between circuit construction and code functionality (A, B, C). For instance, public school CS Teacher A candidly noted the inherent manageability constraints, stating, “*I think the maximum number of students one teacher can handle in physical computing class is about 15.*” These firsthand accounts indicate that while physical computing offers immense educational potential (Appendix A.Q4), its practical implementation is fraught with significant, persistent hurdles. Addressing these hurdles effectively may require new forms of assistance, potentially from AI, but this first necessitates a clear understanding of current AI capabilities and limitations within this specific context.

In particular, the interviews reveal that practical challenges in physical computing directly influence the technical design of PCEVAL — a benchmark specifically created to evaluate LLMs in this domain. While current LLMs are adept at software tasks (Chen et al., 2021), they struggle to integrate abstract instructions with tangible, constrained hardware in physical computing. Key deficiencies include reasoning about physical constraints (e.g., layouts, pinouts) and ensuring coherence between circuit designs and corresponding code. Accordingly, PCEVAL systematically assesses LLMs not only on isolated code or schematic generation but, crucially, on producing physically viable layouts and hardware-compatible code, providing guidance for AI tool development in this domain.

### 3.2 Evaluation Task Structure

Each PCEVAL project is defined by five components: **(D)escription**, a natural language specification of objectives and requirements; **(L)ogical circuit**, an abstract pin-to-pin connection map; **(P)hysical circuit**, a breadboard-level implementation with wiring constraints; **(C)ode**, an executable program aligned with the given circuit; and **(T)est procedure**, a set of automated checks that assert expected outputs under simulated inputs. This decomposition enables controlled evaluation of specific LLM capabilities while maintaining realistic educational scenarios.

### 3.3 Task Definitions

The PCEVAL benchmark is designed to rigorously assess the capabilities of LLMs in physical computing through four distinct generation tasks. As illustrated in Figure 1, each task challenges an LLM to produce a specific target artifact (either a circuit representation or code) based on a designated set of input components from our dataset structure. These tasks systematically probe different facets of an LLM’s understanding, from logical design to physical implementation and code-hardware compatibility. The four tasks are defined as follows:

1. **Logical Circuit Generation** ( $D, C \rightarrow L$ ): Given a natural language project description  $D$  and the corresponding code  $C$ , the LLM is tasked with generating a complete logical circuit specification  $L$ . This task primarily evaluates the LLM’s ability to infer necessary hardware components and their abstract, pin-to-pin logical connections based on functional requirements and program logic, without concern for physical layout details.
2. **Physical Circuit Generation** ( $D, C \rightarrow P$ ): Using the same inputs as the previous task, the project description  $D$  and code  $C$ , the LLM must generate a detailed physical circuit layout  $P$  suitable for breadboard implementation. This task assesses the LLM’s understanding of practical hardware integration, including valid wire routing adhering to breadboard mechanics and physical constraints.
3. **Code Generation from Logical Circuit** ( $D, L \rightarrow C$ ): In this task, the LLM is provided with the project description  $D$  and a logical circuit specification  $L$ . Its objective is to generate a functional code  $C$  that correctly implements the project requirements using the provided logical hardware structure. This tests the ability to translate a conceptual circuit design into working code.
4. **Code Generation from Physical Circuit** ( $D, P \rightarrow C$ ): Similar to the preceding task, the LLM receives the project description  $D$ , but in this case, it is accompanied by a physical circuit layout  $P$ . The LLM must generate the code  $C$  that is compatible with this specific physical hardware configuration, including adherence to the explicit pin assignments dictated by the breadboard layout.

For each project in PCEVAL, we evaluate all four tasks using a controlled methodology that isolates the specific generation capability being assessed. As shown in the right panel of Figure 1, we pair each type of LLM-generated artifact with one of the reference components ( $L$ ,  $P$ , or  $C$ ) to construct complete, testable systems. When evaluating circuit generation capabilities, we connect the generated circuit with the reference code. For code generation tasks, we pair the generated code with the corresponding reference circuit. This structured evaluation approach allows us to execute the Test Procedure ( $T$ ) and precisely measure performance on each distinct capability. By systematically controlling one aspect while testing another, we can accurately determine whether the integrated system functions correctly according to project requirements, ensuring a focused assessment of the LLM’s performance across the physical computing spectrum.

### 3.4 Project Design

PCEVAL contains 300 benchmark task instances derived from 75 curated projects (50 Arduino Uno, 25 ESP32). While the number of projects is modest relative to code-only benchmarks, each instance includes mutually consistent reference code, logical and physical circuits, and an automated test procedure, making each sample substantially more expensive and enabling reproducible execution-based evaluation. Each project defines one complete tuple of description, logical circuit, physical circuit, code, and test procedure. The four tasks in Section 3.3

then create four distinct generation problems from this tuple. These task instances are not repeated attempts at the same output. They evaluate different capabilities under a shared project context. The source projects cover components commonly used in introductory physical computing classes, including LEDs, buttons, buzzers, sensors, displays, motors, and RTC modules. We organize the projects into four curriculum oriented complexity levels. Level 1 covers basic single component or single module input and output. Level 2 adds controlled behavior such as thresholding, serial interaction, or sensor state processing. Level 3 introduces interaction between multiple components, such as buttons with LEDs, sensors with displays, or sensors with actuators. Level 4 covers integrated projects that combine multiple inputs and outputs with state, timing, or application logic. Table 2 summarizes the distribution of projects and task instances across these levels. It also reports descriptive complexity measures, including lines of code, cyclomatic complexity, component count, and connection count. These measures describe the programming and wiring burden of each level. They are not used as evaluation targets. For each project, the reference code, logical circuit, physical breadboard layout, and test procedure were checked for mutual consistency before model evaluation. The success criteria were fixed through the test procedure before any model outputs were evaluated. Appendix E lists all source projects and gives additional details on benchmark scale.

Table 2: Summary statistics for the PCEval benchmark dataset, categorized by complexity level, showing project counts, code and circuit complexity metrics.

| Statistic                 | Level 1 | Level 2 | Level 3 | Level 4 |
|---------------------------|---------|---------|---------|---------|
| Num. Task Instances       | 88      | 80      | 72      | 60      |
| (C) Lines of Code         | 14.27   | 18.60   | 19.41   | 28.40   |
| (C) Cyclomatic Complexity | 3.50    | 5.65    | 5.83    | 7.07    |
| (L) Num. Components       | 4.00    | 3.30    | 7.28    | 7.73    |
| (L) Num. Connections      | 8.45    | 6.60    | 17.50   | 15.86   |
| (P) Num. Components       | 5.00    | 4.30    | 8.28    | 8.73    |
| (P) Num. Connections      | 16.59   | 14.55   | 38.61   | 33.67   |

### 3.5 Evaluation

We assess LLM performance across all PCEVAL tasks using an execution-based protocol that measures both functional correctness and physical feasibility. The primary metric is simulation success. A generated artifact is paired with the corresponding reference artifact and executed against the project-specific test procedure in a simulation environment (Shaked, 2020). For circuit generation, the generated circuit is paired with the reference code. For code generation, the generated code is paired with the reference circuit. This paired reference design isolates the capability under test while still evaluating a complete hardware and software system.

This protocol is stricter than textual similarity or graph overlap. A generated output must not only resemble the reference artifact, but also work with the fixed counterpart artifact under the expected input and output behavior. Small errors such as an incorrect pin number, a missing ground path, or a mismatched device initialization can therefore cause failure. For physical circuit generation, simulation success alone is not sufficient. PCEVAL also checks whether the generated layout is physically constructible on a breadboard. These checks detect structural errors such as redundant connections, missing components, isolated components, pin conflicts, and breadboard bypasses.

Table 3: Comparative performance (success rate %) of various LLMs on PCEVAL benchmark tasks, highlighting differential capabilities across circuit generation and code generation domains.

| Model                 | Param. | Circuit Gen.         |                      |         | Code Gen.            |                      |         | Total Overall |
|-----------------------|--------|----------------------|----------------------|---------|----------------------|----------------------|---------|---------------|
|                       |        | $D, C \rightarrow L$ | $D, C \rightarrow P$ | Overall | $D, L \rightarrow C$ | $D, P \rightarrow C$ | Overall |               |
| Closed Source LLMs    |        |                      |                      |         |                      |                      |         |               |
| GPT-4o-mini           | -      | 50.9                 | 1.3                  | 26.1    | 54.4                 | 53.6                 | 54.0    | 40.1          |
| Gemini-2.0-Flash-Lite | -      | 51.2                 | 2.7                  | 26.9    | 56.3                 | 57.3                 | 56.8    | 41.9          |
| Claude Haiku 4.5      | -      | 58.1                 | 3.7                  | 30.9    | 69.1                 | 64.3                 | 66.7    | 48.8          |
| Gemini-2.0-Flash      | -      | 60.5                 | 17.9                 | 39.2    | 59.7                 | 55.7                 | 57.7    | 48.5          |
| Gemini-3.1-Flash-Lite | -      | 71.5                 | 26.9                 | 49.2    | 72.3                 | 68.8                 | 70.5    | 59.9          |
| GPT-4.1               | -      | 54.1                 | 16.0                 | 35.1    | 68.8                 | 63.2                 | 66.0    | 50.5          |
| GPT-4o                | -      | 62.4                 | 28.5                 | 45.5    | 65.3                 | 61.1                 | 63.2    | 54.3          |
| GPT-5-mini            | -      | 56.5                 | 44.8                 | 50.7    | 70.9                 | 72.5                 | 71.7    | 61.2          |
| o3-mini               | -      | 65.3                 | 46.4                 | 55.9    | 71.5                 | 71.2                 | 71.3    | 63.6          |
| Open Source LLMs      |        |                      |                      |         |                      |                      |         |               |
| LLaMA 3.1             | 8B     | 24.3                 | 1.6                  | 12.9    | 29.6                 | 25.9                 | 27.7    | 20.3          |
| DeepSeek-Coder-V2     | 16B    | 33.3                 | 0.8                  | 17.1    | 34.1                 | 21.1                 | 27.6    | 22.3          |
| Gemma 3               | 27B    | 52.8                 | 1.6                  | 27.2    | 36.0                 | 37.3                 | 36.7    | 31.9          |
| Phi 4                 | 14B    | 37.1                 | 2.4                  | 19.7    | 49.1                 | 37.9                 | 43.5    | 31.6          |
| Mistral-Small 3       | 24B    | 51.5                 | 14.1                 | 32.8    | 49.1                 | 35.7                 | 42.4    | 37.6          |

Table 4: Detailed success rates (%) of selected high-performing LLMs across project complexity levels, demonstrating performance degradation with increasing project complexity.

| Model                 | $D, C \rightarrow L$ |             |             |             | $D, C \rightarrow P$ |             |             |             | $D, L \rightarrow C$ |             |             |             | $D, P \rightarrow C$ |             |             |             |
|-----------------------|----------------------|-------------|-------------|-------------|----------------------|-------------|-------------|-------------|----------------------|-------------|-------------|-------------|----------------------|-------------|-------------|-------------|
|                       | L1                   | L2          | L3          | L4          | L1                   | L2          | L3          | L4          | L1                   | L2          | L3          | L4          | L1                   | L2          | L3          | L4          |
| Claude Haiku 4.5      | 84.7                 | 64.3        | 36.4        | 43.3        | 2.7                  | 5.1         | 2.9         | 2.4         | 78.0                 | <b>80.4</b> | 67.6        | 41.1        | 72.9                 | 65.1        | 72.6        | 39.9        |
| Gemini-3.1-Flash-Lite | <b>88.9</b>          | <b>81.5</b> | <b>51.2</b> | 51.8        | 45.1                 | 34.8        | 13.6        | 6.1         | 84.9                 | 79.2        | <b>75.9</b> | 42.3        | 74.9                 | <b>80.2</b> | 73.1        | 44.7        |
| GPT-5-mini            | 68.7                 | 57.9        | <b>51.2</b> | 43.9        | 55.3                 | 49.6        | <b>34.3</b> | <b>35.1</b> | <b>89.3</b>          | 67.9        | 73.6        | 43.5        | <b>86.7</b>          | 72.1        | <b>78.6</b> | 47.1        |
| o3-mini               | 84.9                 | 71.3        | 43.1        | <b>54.3</b> | <b>58.0</b>          | <b>66.0</b> | 28.3        | 32.0        | 86.5                 | 72.3        | 72.1        | <b>48.3</b> | 83.3                 | 74.1        | 71.7        | <b>48.3</b> |
| Phi 4                 | 62.0                 | 41.1        | 11.2        | 25.5        | 2.2                  | 7.1         | 0.0         | 0.0         | 58.2                 | 45.4        | 55.8        | 31.3        | 43.5                 | 54.1        | 32.9        | 20.5        |
| Mistral-Small 3       | 72.7                 | 57.1        | 29.7        | 38.5        | 29.1                 | 20.0        | 0.0         | 5.7         | 60.9                 | 59.8        | 32.1        | 36.2        | 49.3                 | 47.8        | 24.5        | 14.4        |

Reference implementations were checked against real hardware and simulator. This makes the benchmark scalable while grounding the target behavior in classroom hardware artifacts. We conduct five independent trials for each task-model combination and report mean success rates. We use mean success as the primary metric because classroom assistance requires reliable single-attempt outputs. Appendix D reports pass@5 as a complementary sampling-based measure. Detailed metric definitions, simulator details, and validation statistics are provided in Appendix C.

## 4 Experimental Results

### 4.1 Setup

We evaluated 14 LLMs spanning frontier closed-source systems and representative open-source models (Team et al., 2025; Grattafiori et al., 2024; Abdin et al., 2024; Jiang et al., 2023). We selected models to cover (i) widely deployed frontier APIs used in educational coding assistance, and (ii) representative open-source LLMs with strong code-generation performance at different scales (Jimenez et al., 2023; Rein et al., 2023; Chen et al., 2021). Our experimental comparison focuses on LLMs because PCEVAL evaluates an end-to-end, partially specified generation setting: the input consists of natural-language project descriptions and partial artifacts, while the output may require inferring missing hardware structure, producing a breadboard-level layout, or synthesizing executable microcontroller code. Off-the-shelf symbolic, search-based, and EDA methods typically assume formalized inputs such as truth tables, netlists, placements, or routing constraints, and therefore are not directly comparable as single-solver baselines (Brayton et al., 1984; Kirkpatrick et al., 1983; Lee, 1961; Hart et al., 1968) (Appendix F). For each of the four benchmark tasks, we used standardized prompts with explicit output specifications (Appendix M). Table 3 reports aggregate results over all 300 benchmark task instances, which are derived from 50 Arduino Uno projects and 25 ESP32 projects. Appendix G provides the platform specific breakdown.

## 4.2 Analysis of Model Performance and Task Characteristics

**Circuit Gen. vs. Code Gen.** Table 3 shows that most models achieve higher success in code tasks than in circuit tasks. For example, GPT-5-mini reaches 71.7% on code generation overall versus 50.7% on circuit generation overall, while Claude Haiku 4.5 reaches 66.7% versus 30.9%. This trend also holds for the strongest overall model, o3-mini, which obtains 71.3% on code generation overall but 55.9% on circuit generation overall. These results suggest that, although LLMs can often synthesize executable microcontroller code when given a circuit representation, generating the corresponding hardware artifact remains less reliable.

**Logical Circuit Gen. and Physical Circuit Gen.** The most striking finding, evident in Table 3, is the substantial performance gap between logical and physical circuit generation. Across all models, success rates for Physical Circuit Generation ( $D, C \rightarrow P$ ) are markedly lower than for Logical Circuit Generation ( $D, C \rightarrow L$ ). Many models achieved success rates below 10% for physical circuit generation. Even top-performing models such as o3-mini (46.4%) and GPT-5-mini (44.8%) found this task considerably more challenging than logical circuit design, where they scored 65.3% and 56.5%, respectively. This highlights the profound difficulty LLMs face in translating conceptual project requirements and code into physically valid and implementable breadboard layouts while considering pin assignments.

**Sensitivity to Project Complexity.** Project complexity generally makes the benchmark more difficult (Table 4). This tendency is especially clear in physical circuit generation, where performance on later levels is typically lower than on earlier levels. For example, Gemini-3.1-Flash-Lite declines from 45.1% at Level 1 to 6.1% at Level 4, o3-mini declines from 58.0% to 32.0%, and Mistral-Small 3 declines from 29.1% to 5.7%. Logical circuit generation and code generation also become harder at later levels, although the decline is generally less severe than in physical circuit generation. This pattern is consistent with the curriculum-oriented level design of PCEVAL, which progresses from single-component tasks to controlled single-component behavior, multi-component coordination, and integrated multi-component projects. See Appendix J and Appendix K for complementary visual analyses across complexity metrics and project levels.

**Dominant Failure Causes in  $D, C \rightarrow P$ .** Physical circuit generation errors are not merely logical inconsistencies. They reflect violations of breadboard mechanics. Figure 2 separates behavioral success from physical constructability. The blue bars count layouts that pass the functional test procedure, while the red bars additionally require the layout to avoid both breadboard bypasses and pin conflicts. The gap between these bars shows that some generated layouts are behaviorally plausible in simulation but still impossible or inappropriate to construct as breadboard circuits. Pin conflicts cause the largest drop for most selected models. Breadboard bypasses vary more by model. This pattern suggests that current LLMs can often recover the intended circuit behavior, but fail to respect occupancy and topology constraints that are specific to breadboard construction. Detailed results are provided in Appendix C.

**Implication of the task decomposition.** These results show that PCEVAL is diagnostic rather than merely comparative. The gap between Logical and Physical Circuit Generation localizes a major failure mode to breadboard-level spatial and occupancy reasoning, rather than to component inference alone. Conversely, the relatively stronger performance on code-generation tasks suggests that many models can synthesize compatible programs once the circuit representation and pin assignments are

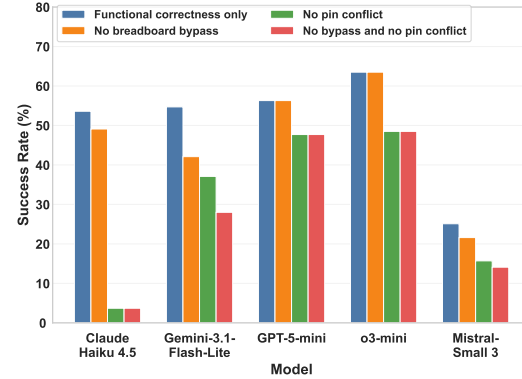


Figure 2: Impact of physical-constraint filters in  $D, C \rightarrow P$ : success rates are shown under progressively stricter criteria—**blue**: functional correctness only; **orange**: functional correctness and no breadboard bypass; **green**: functional correctness and no pin conflict; **red**: functional correctness and no breadboard bypass and no pin conflict. The drop from blue to red highlights how bypasses and pin conflicts drive failures in physical layout generation.

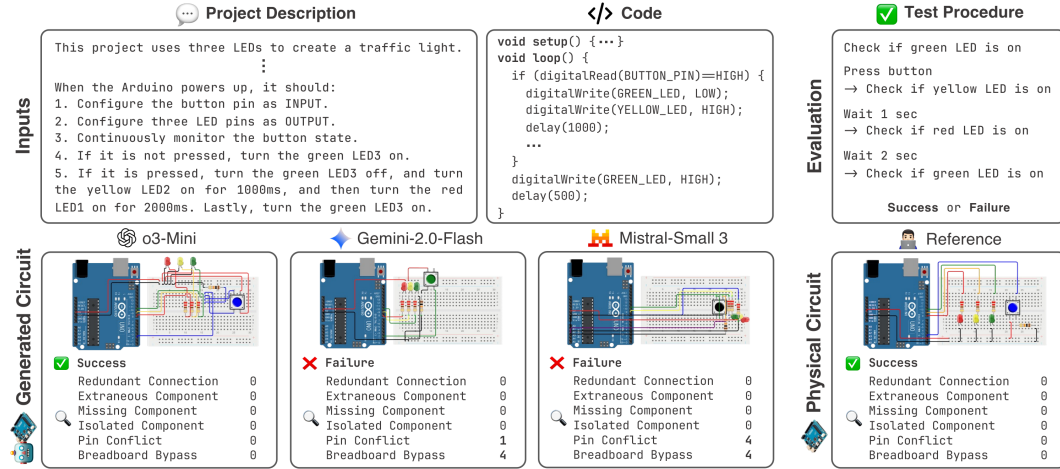


Figure 3: Qualitative examples of LLM-generated physical circuits for a traffic light project (level 4), illustrating successful and failed attempts with corresponding error analyses.

given. This decomposition therefore identifies where future systems should intervene: not only in code synthesis, but also in explicit planning, and constraint checking for physical layouts.

### 4.3 Exploratory Mitigation Strategies

We additionally tested two prompting-based mitigation strategies in the Arduino Uno setting. Detailed results are reported in Tables 7 and 8 in Appendix D.

**Self-improvement.** Our error analysis suggested that many failures arise from small but execution-critical implementation errors across both circuit and code outputs, including missing connections and incorrect pin specifications. Motivated by this, we explored a multi-turn self-improvement strategy in which the model received structured failure logs and iteratively revised its solution for up to five turns. This strategy improved overall performance by +10.3 percentage points for o3-mini and +7.1 points for Gemini-2.0 Flash, with especially large gains in code-generation tasks, where even minor pin-related mistakes often caused otherwise plausible solutions to fail.

**Chain-of-thought prompting.** For tasks which use physical circuit, we hypothesized that explicit intermediate reasoning could help bridge logical design and physical implementation. We therefore guided models to first generate an intermediate Logical Circuit representation before producing the final code or physical layout—mirroring the human process of reasoning from abstract connections to concrete breadboard placement. This chain-of-thought strategy produced notable improvements for some models including +16.8 percentage points for Mistral-Small 3, +11.6 for DeepSeek-Coder-V2, and +10.8 for Gemma 3, although its impact was not uniform across all models.

### 4.4 Qualitative Results

Figure 3 presents an example of the input, generated circuit, testing process, and results for a physical circuit generation task involving a traffic light project. The input consists of a scenario and corresponding code in which pressing a button changes the light from green to yellow to red and then back to green. The figure further illustrates which specific constraints are violated in each failed case. This scenario appears to be challenging because it requires correctly assigning the role of each leg of the button within the circuit. In addition, the example from o3-Mini suggests that even when a circuit functions correctly, it is not necessarily well-organized or easy to interpret. More qualitative results can be found in Figures 4, 5 and 6 in Appendix I.

## 5 Educational Usability and Pedagogical Implications

Our findings suggest that LLMs hold considerable promise for physical computing education, but their integration must be guided by careful pedagogical considerations. Prior studies already caution

that students may rely on LLMs for “ready-made answers,” which can hinder deeper engagement and learning (Anson, 2024; Jošt et al., 2024). This concern was also raised in our *initial interviews with eight experienced CS teachers*, which we conducted at the outset to identify the central pain points in physical computing education (Appendix A). Educators emphasized that while LLMs can generate working physical computing solutions, students may bypass essential reasoning processes if guidance is not structured. This challenge is amplified by known factors such as automation bias (Li and Wu, 2025) and the moderating role of prior knowledge (Lee et al., 2025), where learners with limited foundational skills may be less able to critically assess AI outputs.

To complement these design-stage insights, we conducted exploratory post-benchmark validation: a focus group with two physical computing educators (8 and 10 years of experience, 30 minutes each) and a survey of three pre-service CS teachers (Appendix B). Unlike the initial interviews, these sessions were conducted *with the benchmark and model outputs in hand*, enabling participants to directly assess educational potential and limitations. Both groups highlighted tangible benefits—particularly the use of automated circuit verification to reduce instructor workload and assist less-experienced teachers. At the same time, they noted practical barriers for classroom use: the lack of step-by-step assembly guidance, declining readability in complex layouts, and the absence of pedagogical conventions such as consistent wire colors. Notably, pre-service teachers rated low-complexity projects as highly usable in classrooms, but their educational value dropped sharply at higher complexity levels.

Taken together, these findings suggest two implications: (i) LLMs should serve as scaffolding tools that structure learning and support student agency, and (ii) benchmarks such as PCEVAL should extend beyond functional correctness to capture usability factors (e.g., readability, step-wise guidance). Our work provides a foundation for this direction, highlighting the need to bridge computational correctness and instructional utility.

## 6 Conclusion

We introduced PCEVAL, a benchmark for systematically evaluating LLM capabilities in physical computing tasks. Our evaluation of 14 models reveals that while LLMs perform reasonably well in code generation and logical circuit design, they struggle significantly with physical circuit generation, particularly in managing pin conflicts and breadboard layout constraints. This performance gap indicates that current LLMs have difficulty reasoning about spatial and physical implementation details that are essential for educational physical computing. While mitigation strategies like self-improvement and chain-of-thought prompting provided partial improvements, the challenges remain most pronounced in tasks requiring precise pin assignments and adherence to breadboard mechanics—areas where students typically need the most guidance. Educational validation through focus groups suggested that effective classroom deployment requires consideration of pedagogical factors beyond technical correctness, including step-by-step guidance and visual clarity. These findings suggest that AI support tools for physical computing education must balance automation with learning objectives.

## 7 Limitations

**Educational Focus and Scope.** PCEVAL focuses on introductory STEM education contexts, which represents the domain where physical constraint reasoning is most urgently needed and least studied. While this scope excludes professional embedded systems scenarios (e.g., power optimization, real-time requirements, EMI considerations), we argue that the benchmark’s primary value lies precisely in this educational focus: it is the first to reveal that LLMs systematically fail at physical layout reasoning even in simplified settings, suggesting that this gap is fundamental rather than domain-specific.

**Platform Scope and Generalizability.** PCEVAL evaluates LLM performance on two representative microcontroller platforms for introductory physical computing, Arduino Uno and ESP32 (El-Abd, 2017; García-Tudela and Marín-Marín, 2023; Schätz et al., 2024). While this reduces the likelihood that our findings are artifacts of two boards, the current benchmark is still limited to these two platforms and does not capture the full diversity of physical computing ecosystems. At the same time, this limitation is primarily one of coverage rather than benchmark design, since PCEVAL’s task decomposition is not tied to a particular board and can be extended to additional platforms by providing the corresponding board specifications, reference circuits, and evaluation procedures.

## References

- Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, and 1 others. 2024. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*.
- Yadagiri Annepaka and Partha Pakray. 2024. Large language models: A survey of their development, capabilities, and applications. *Knowledge and Information Systems*, pages 1–56.
- Daniel WJ Anson. 2024. The impact of large language models on university students’ literacy development: A dialogue with lea and street’s academic literacies framework. *Higher Education Research & Development*, 43(7):1465–1478.
- José Luís Araújo and Isabel Saúde. 2025. Exploring arduino programming in non-formal education context: enhancing middle school students’ interest and motivation. *Research in Science & Technological Education*, pages 1–18.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.
- Robert K Brayton, Gary D Hachtel, Curt McMullen, and Alberto Sangiovanni-Vincentelli. 1984. *Logic minimization algorithms for VLSI synthesis*, volume 2. Springer Science & Business Media.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- Chih-Chao Chung and Shi-Jer Lou. 2021. Physical computing strategy to support students’ coding literacy: an educational experiment with arduino boards. *Applied Sciences*, 11(4):1830.
- Mohammed El-Abd. 2017. A review of embedded systems education in the arduino age: Lessons learned and future directions. *International Journal of Engineering Pedagogy*, 7(2).
- Pedro Antonio García-Tudela and José-Antonio Marín-Marín. 2023. Use of arduino in primary education: a systematic review. *Education Sciences*, 13(2):134.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Ce Guo and Tong Zhao. 2025. Resbench: Benchmarking llm-generated fpga designs with resource awareness. *arXiv preprint arXiv:2503.08823*.
- Peter E Hart, Nils J Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107.
- Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and 1 others. 2021. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*.
- Seong-Eun Hyeon, Seungyun Lee, Hyoung Jun Kim, and 1 others. 2016. A study on the development of teacher training programme for maker education. *European Proceedings of Social and Behavioural Sciences*.
- Peter Jansen. 2023. From words to wires: Generating functioning electronic devices from natural language descriptions. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 12972–12990.
- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Léo Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.
- Donald M Johnson, Will Doss, and Christopher M Estep. 2024. Using chatgpt with novice arduino programmers: Effects on performance, interest, self-efficacy, and programming ability. *Journal of Research in Technical Careers*, 8(1):1.

- Gregor Jošt, Viktor Taneski, and Sašo Karakatič. 2024. The impact of large language models on programming education and student learning outcomes. *Applied Sciences*, 14(10):4115.
- Oliver Kastner-Hauler, Karin Tengler, Barbara Sabitzer, and Zsolt Lavicza. 2022. Combined effects of block-based programming and physical computing on primary students’ computational thinking skills. *Frontiers in Psychology*, 13:875382.
- Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. 1983. Optimization by simulated annealing. *science*, 220(4598):671–680.
- Yao Lai, Sungyoung Lee, Guojin Chen, Souradip Poddar, Mengkang Hu, David Z Pan, and Ping Luo. 2025. Analogcoder: Analog circuit design via training-free code generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 379–387.
- Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pages 18319–18345. PMLR.
- Chin Yang Lee. 1961. An algorithm for path connections and its applications. *IRE transactions on electronic computers*, (3):346–365.
- Hao-Ping Hank Lee, Advait Sarkar, Lev Tankelevitch, Ian Drosos, Sean Rintel, Richard Banks, and Nicholas Wilson. 2025. The impact of generative ai on critical thinking: Self-reported reductions in cognitive effort and confidence effects from a survey of knowledge workers.
- Rui Li and Tong Wu. 2025. Delving into the practical applications and pitfalls of large language models in medical education: Narrative review. *Advances in Medical Education and Practice*, pages 625–636.
- Bingyang Liu, Haoyi Zhang, Xiaohan Gao, Zichen Kong, Xiyuan Tang, Yibo Lin, Runsheng Wang, and Ru Huang. 2025. Layoutcopilot: An llm-powered multi-agent collaborative framework for interactive analog layout design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.
- Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. 2023. Verilogeval: Evaluating large language models for verilog code generation. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–8. IEEE.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2023. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*.
- Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*.
- Eric Schätz, Lutz Hellmig, and Alke Martens. 2024. Analysis of student-problems while working with physical computing devices. In *CSEDU (1)*, pages 322–329.
- Uri Shaked. 2020. Wokwi - online arduino and esp32 simulator. Accessed: 2025-05-09.
- Priti Subramaniam, Dr. Gajanan Uttam Patil, Gajanan Uttam Patil, and Anilkumar Dulichand Vishwakarma. 2024. Application of ai chat gpt with arduino and micropython. *International Journal of Innovations in Engineering and Science*.
- Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, and 1 others. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.
- Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg. 2024. Verigen: A large language model for verilog code generation. *ACM Transactions on Design Automation of Electronic Systems*, 29(3):1–31.
- Anastasios Theodoropoulos, Prokopis Leon, Angeliki Antoniou, and George Lepouras. 2018. Computing in the physical world engages students: Impact on their attitudes and self-efficacy towards computer science through robotic activities. In *Proceedings of the 13th workshop in primary and secondary computing education*, pages 1–4.
- Tianyu Wang, Nianjun Zhou, and Zhixiong Chen. 2024. Enhancing computer programming education with llms: A study on effective prompt engineering for python code generation. *arXiv preprint arXiv:2407.05437*.

- 471 Huanqi Yang, Mingzhe Li, Mingda Han, Zhenjiang Li, and Weitao Xu. 2024. Embedgenius: Towards automated  
472 software development for generic embedded iot systems. *arXiv preprint arXiv:2412.09058*.
- 473 Dewu Zheng, Yanlin Wang, Ensheng Shi, Hongyu Zhang, and Zibin Zheng. 2024. How well do llms generate  
474 code for different application domains? benchmark and evaluation. *arXiv preprint arXiv:2412.18573*.

## 475 A Interviews with Computer Science Educators

476 To gain comprehensive insights into the practical challenges encountered in physical computing  
477 education, interviews were conducted with eight educators. This group included six in-service  
478 computer science teachers from public middle and high schools (Teacher A, B, C, D, E, F) and two  
479 instructors with experience in private educational settings (Instructor G, H). The interviews aimed  
480 to identify common difficulties in areas such as class preparation, managing diverse student needs,  
481 providing individualized support, the inherent benefits and drawbacks of physical computing, and the  
482 need for better tools and resources. All educator names have been anonymized. Each interview lasted  
483 approximately 30 minutes. Participants were recruited by directly contacting educators with relevant  
484 teaching experience in computer science or physical computing education. Participation was entirely  
485 voluntary, and no financial compensation was provided, as these interviews were conducted as brief  
486 expert contributions rather than paid labor. At the time of recruitment and again before the interviews  
487 began, participants were informed that their responses would be used for research on AI support in  
488 physical computing education, that selected excerpts could be reported only in anonymized form, and  
489 that participation was optional and could be discontinued at any time. Consent was obtained from all  
490 participants before the interviews.

### 491 A.1 Key Interview Question Areas

- 492 • **Q1:** What are the primary challenges and time-consuming aspects encountered when preparing for  
493 and conducting physical computing or practical coding classes?
- 494 • **Q2:** How do diverse student skill levels and the need for individualized support impact the flow of  
495 physical computing classes, and what difficulties arise in providing this support?
- 496 • **Q3:** What are the perceived benefits and drawbacks of physical computing classes, particularly  
497 regarding hardware integration (like Arduino) with software?
- 498 • **Q4:** What kind of additional support, tools, or resources do educators feel are needed to conduct  
499 physical computing classes more effectively, including the potential role of LLMs?

### 500 A.2 Thematic Summary of Interview Responses with Excerpts

- 501 **Q1:** What are the primary challenges and time-consuming aspects encountered when preparing for  
502 and conducting physical computing or practical coding classes?

Educators consistently highlighted the significant time investment required for both preparing and conducting physical computing classes. Key challenges include the initial setup of hardware and software, creation or sourcing of suitable teaching materials, and managing the wide range of student skill levels.

- **Teacher A:** “Well, I think all teachers would agree that the most difficult stage is the connection stage.” “Each time, firmware and installed programs disappear, which is a challenge.” “So, I think it’s not easy for teachers to change teaching aids every year.” “During class, well, connection is something I prepare beforehand, but we always connect when class starts. So, in a way, that causes delays in the class progression.”
- **Teacher B:** “First, the initial setup takes longer than expected.” “But just as students need time to get used to new equipment or software they are handling for the first time, understanding the machine or physical computing device and how to connect it takes quite some time, which often takes longer than the essential programming part.” “But for physical computing, the order and structure need to be slightly rearranged by the teacher depending on the task.”

503

- **Teacher C:** “First, when the practical exercise begins, I try to create reference materials that are as detailed as possible for the students.” “And when we actually start, many unexpected errors appear. Among them are the students’ lack of experience with circuits, and also the stability of the boards.” “Compatibility boards often perform poorly. So, I always prepare plenty of spares.” “Driver installation and such, for compatibility boards, sometimes require installing separate drivers. There are many such environmental factors. And if students bring their own laptops, the number of things to set up increases significantly.” “So, initially, when we did the camp with the Digital Sprout (CS Camp) [Anonymous] University side, even following the manuals they sent, files wouldn’t upload at all... I remember digging through English manuals, manually setting everything up, and working on that continuously for a week before the class.”
- **Teacher E:** “Then, the initial setup actually takes the longest time. In the case of Micro:bit, the setup environment doesn’t take that extremely long, but anyway, when using any new teaching aid, I anticipate it will probably take a very long time.” “Even simple things like having to sign up for a new environment, each little thing becomes a hurdle.”
- **Teacher F:** “In the case of specialized high schools, since I’m not teaching the subjects I expected, I have to prepare for classes anew every time.” “The content isn’t light either, so I have to study while preparing for class, and I also have to create materials.” “However, for specialized high school subjects, there aren’t many materials, so I think the biggest difficulty is that I have to create everything myself.” “Even in those, as I mentioned earlier, since class materials are not widely shared, I think creating examples for the students to do is the most difficult part.”
- **Instructor G:** “Yes, robot coding was quite difficult. Since I can’t use the robot at home, it’s an item that’s only at the academy. So even if I studied a lot before going, there were times when it didn’t work well in practice.” “So, I really wanted to go to the academy a bit late, but once I went about an hour early to try it out myself.”
- **Instructor H:** “The step that took a lot of time in the preparation process was that we usually had to continuously check the mini-cars for any malfunctions or charging status before and after class.” “Also, for auxiliary teaching aids like partitions or obstacles, we had to continuously check if there were any issues, and that process seemed to take a lot of time.”

504

505 **Q2:** How do diverse student skill levels and the need for individualized support impact the flow of  
506 physical computing classes, and what difficulties arise in providing this support?

A major recurring theme was the challenge of managing diverse student abilities. Instructors find it difficult to cater to both fast learners and those who struggle, often leading to class delays and an inability to provide equal support to all students. Troubleshooting individual student issues, whether hardware or software, consumes significant class time.

- **Teacher A:** “Individual understanding varies much more with these teaching aids. So, I have to walk around a lot. (To support students)” “If a connection is lost mid-class, that team gets delayed, and all other teams have to wait for that team to reconnect.” “I think students ask more questions when their understanding of the tool is lacking.” “Yes, yes, yes, I definitely think so [that there isn’t enough time for one teacher to help all students].” “I think the maximum number of students one teacher can handle is about 15.” “If all four members lack aptitude or are all struggling and fall behind, I end up focusing on those students and can’t provide much feedback to other teams. The other teams might have finished and are just playing around.”
- **Teacher B:** “Even with the same code, the sensors and other physical computing components differ slightly, so things often don’t operate as desired.” “Motors might suddenly break down, sensors might fail, or physical connections might come loose. When such things happen, spending time addressing them means less time for the actual lesson... losing a lot of time that way happens frequently.” “The variation among students is also very large, so it’s always

507

good to prepare one or two advanced tasks or tasks that students can do according to their level.”

- **Teacher C:** “Once you exceed about 10 students, practical physical computing classes become very difficult.” “First, students find building circuits quite challenging... many kids lack a high understanding of electrical circuits... Then, we also had to check if the code was written correctly accordingly, so that was a somewhat difficult part.” “When the friend next to them has it working, but theirs isn’t... their sense of achievement can diminish quickly... feedback needs to be fast, there need to be many people available to help, and peer learning... need to be utilized much more.” “Over 60% [of class time is spent giving feedback].”
- **Teacher D:** “Since the class pace is extremely fast, students who grasp things well and those who already have some basic knowledge... follow along well. But students who are encountering the language for the first time seem to take some time to apply concepts and write code. So, there’s quite a bit of difference in level.” “It’s rather a very small minority, so those kids can’t confidently ask for help like this. During class time.” “If you start helping one student with their question, you have to keep helping that student, and then you can’t cover all the material.”
- **Teacher E:** “First, the students have a very hard time keeping up. Initially, when using something new, they really struggle.” “I do one step, then go around and check everyone, then do another step, go around and check everyone again. This keeps repeating... eventually, I have to go around and check on everyone.” “The class gets significantly delayed.”
- **Teacher F:** “The biggest thing is the difference in level among the students.” So, some students finish too quickly and are playing around after about 10 minutes, while some students don’t understand until the very end, so I have to stand next to them and help them continuously.” “As I explain it more simply and easily, the time spent with one student becomes longer. So, the overall class time becomes loose.”
- **Instructor G:** “The students’ levels are really so different.” “To match their levels, I tried many activities. First, I tried teaching them one-on-one... but that was too hard for me.” “Because the students’ levels were so different, if the number of students increased, it seemed very difficult to match each student’s progress or solve each student’s problems.”
- **Instructor H:** “Some students who didn’t listen properly to the theory class didn’t understand the meaning or main activity of the practical exercise. So, for these students, we had the difficulty of having to explain the theory class content again while helping them with the practical activity.” “Since there were 15 teams of middle school students, there were parts that students didn’t understand, and also students who couldn’t proceed properly, so we had to help them one by one. There were several times when I felt that one main instructor and two assistant instructors were not enough to support 15 teams.” “Explaining the theory to each team, explaining what the practical content was, and helping them... usually meant that the practical exercise time... would often exceed 1 minute per team. Therefore, there was a shortage of time to guide all 15 teams.”

508

509 **Q3:** What are the perceived benefits and drawbacks of physical computing classes, particularly  
510 regarding hardware integration (like Arduino) with software?

Educators see significant value in physical computing for its ability to engage students and make abstract concepts tangible. However, they also point out the inherent difficulties in debugging hardware, managing unpredictable component behavior, and the financial and maintenance burden of hardware kits.

- **Teacher A:** “Advantages, well, the kids definitely enjoy it. I think it’s one of the classes they find most fun.” “they can actually see it with their own eyes, and I think that generates a lot of interest.” “The disadvantage... Students who find it difficult find this more challenging than programming.” “First, many students struggle with the connection stage.” “If they’ve written the code perfectly, but it doesn’t run as expected, it could actually be a flaw in the

511

physical computing tool itself... I think the difficulty also lies in having to consider two things [hardware and software].”

- **Teacher B:** “For classes using physical computing, kids often only work in virtual environments like Python, so they don’t know how it’s actually used... But when they actually see something move... I think they feel, ‘Ah, this is why it’s necessary.’” The biggest disadvantage is, firstly, it’s not easy for schools to provide the budget for this.” “Secondly... managing it continuously is difficult. Especially for Arduino, sensors and similar items are often disposable.”
- **Teacher C:** “The advantage is that it seems to foster interest in the various electrical appliances and many tools in our actual daily lives.” “The problem is, they need continuous success to maintain interest.” “As for disadvantages, it consumes too much time [for teacher preparation].” “Acquiring or purchasing such hardware in Korea is quite cumbersome... things sold as kits, are often too expensive.”
- **Teacher E:** “The biggest advantage seems to be capturing the students’ concentration and interest.” “However, the disadvantages could be that teaching aids can break, and that students get too distracted by them and don’t focus properly on the lesson.” “First, there’s no equipment. Primarily, there’s no equipment.” “Last year, our school had absolutely no budget for the computer science subject.”
- **Teacher F:** “The biggest advantage is that it’s not just theory; students can try things out and internalize them.” “The disadvantage seems to be... it takes a lot of time, and the direction I planned for the class and the direction the students actually follow are different.” “And the biggest disadvantage in terms of teaching aids is that they cost a lot of money.”
- **Instructor G:** “In the case of robot coding, using LEGOs definitely sparks students’ interest. So, because it’s fascinating, they become more immersed, more focused, and more participative.” “[Disadvantage:] If students drop them, it seems to result in a very large financial loss.” “If the number of LEGOs for robot coding we are using is limited, if it breaks or an error occurs, there have been cases where that student could hardly proceed with the class.”
- **Instructor H:** “Classes using teaching aids... have the advantage of consolidating the theoretical content once more and allowing students to understand it better by directly experiencing the theory with their hands.” “When students use physical teaching aids... their concentration level increases, and they don’t get bored.” “The biggest disadvantage... is that since it’s an activity conducted after the theory class, for students who don’t understand the theoretical content, both the theory class and the teaching aid class feel rather meaningless.” “There were students who would damage the teaching aids, such as by pressing that sensor part with their hands, scribbling on the teaching aids, or trying to break them.”

512

513 **Q4:** What kind of additional support, tools, or resources do educators feel are needed to conduct  
514 physical computing classes more effectively, including the potential role of LLMs?

There’s a clear call for better support systems, including more accessible and detailed guidelines for students, improved teacher training, larger budgets for hardware, and tools that can help bridge the gap between software and hardware. The potential of LLMs is recognized, but their current limitations in addressing physical hardware challenges are also noted.

- **Teacher A:** “I wish there were more opportunities for the creators of the tools, or for places that professionally train teachers on these tools, to provide regular training.” “Sharing of teaching materials that teachers can use easily.” “The connection methods need to be developed to be easy, and those developed connection methods should be widely distributed so that students can understand them at their level.” “[Regarding LLMs for students] First-year middle school students can’t use GPT, so I’ve never tried it, and their Googling skills are also lacking... So, if there were a platform or something that summarized this kind of information, I would refer to that platform.”
- **Teacher B:** “Students learn Python, but to actually learn Arduino, they might need to be taught C++, which presents a double cognitive load. So, finding a way to reduce that seems

515

- to be the first necessity.” “Secondly, many teachers perceive physical computing as difficult... I think teacher training or similar initiatives are needed to reduce those perceptions.” “For Arduino, yes, there have been times like that [using ChatGPT]. but... often they just end up asking ‘make it for me.’... a lack of understanding of the coding itself tends to occur.”
- **Teacher C:** “Feedback needs to be fast, there need to be many people available to help, and peer learning... need to be utilized much more.” “For the code parts, naturally, these days... using various AI like ChatGPT, you can catch errors, get help with code writing for connections, etc. But in the physical aspect, giving students quick feedback on various errors is really necessary. Especially in physical computing, the circuit part is crucial, and this is something AI absolutely cannot do for you... An arm doesn’t suddenly pop out from there to build the circuit.” “Nowadays, like Collab, Jemaine just wrote the code and recommended it to me, and it’s like this.” “I wish schools had more budget to purchase such things.” “To do physical computing, you need a small makerspace-like area... but those are often lacking.”
  - **Teacher D:** “[When using LLMs for class preparation] I still don’t think the quality and completeness are high enough to be immediately used in practice... Since it can’t be used directly, I figure it’s easier to just quickly make the blanks myself.” “[When do I use GPT?] When showing short code examples for practice during class.”
  - **Teacher E:** “First, there’s no equipment. Primarily, there’s no equipment.” “Last year, our school had absolutely no budget for the computer science subject.” “The entire budget for teaching aid purchase was 1 million won.”
  - **Teacher F:** “First, I hope the Ministry of Education provides a lot of budget.” “I think it would be good if there was a tool or service that could organize information about such teaching aids or share it with teachers.” “[Regarding LLMs] since I can’t code, I always use GPT to search.”
  - **Instructor G:** “I think the guidelines given to students are too few.” “If we could distribute guidelines to each student on how they can actually use them... if detailed guidelines were available through a platform on those laptops, it would be a huge help during class.” “If they could model what they’re building with their hands at home before coming to the academy, if they could try to fit the pieces together and implement it in software beforehand, it would be a huge help.”
  - **Instructor H:** “In the case of teaching aids, malfunctions are frequent, and I believe that continuously developing new teaching aids is more helpful for students’ classes. Therefore, I think financial support is obviously necessary.” “If there was a management system or a communication tool, instructors could communicate better, and it would be convenient to have a community where we can report and inform each other about problematic teaching aids or tools.”

## B Educational Validation through Expert Focus Groups

### B.1 Participants and Procedure

We conducted two semi-structured interviews with physical computing educators (8 and 10 years of teaching experience, respectively; 30 minutes each) and collected responses from three pre-service CS teachers (senior undergraduates who had completed teacher certification) via a short online survey. These participants were recruited by direct invitation based on their relevant teaching background. Participation was voluntary, and no financial compensation was provided. Before the interviews or survey, participants were informed that their feedback would be used for research on the educational usefulness of PCEVAL and LLM outputs, reported only in anonymized or aggregate form, and that participation was optional. Consent was obtained from all participants prior to data collection.

### B.2 Key Findings from Educators

- **Step-by-step guidance:** Educators emphasized that students require sequential assembly instructions (e.g., “now connect the red wire to pin 13”), which current LLM outputs do not provide.
- **Multiple candidate solutions:** Presenting alternative layouts was highlighted as pedagogically valuable for teaching breadboard principles, compared to providing only one “answer.”
- **Readability principles:** Practical design habits (e.g., consistent wire colors, left-oriented layouts) were noted as essential for teaching, yet absent from current benchmarks.
- **Reducing capability gaps:** Educators noted that LLMs could help less-experienced teachers identify circuit problems and provide richer feedback to students.
- **Validation of automated simulation:** Simulation-based verification was recognized as a major strength, reducing teacher workload by ensuring functional correctness before classroom use.

### B.3 Findings from Pre-service Teachers

Survey participants rated educational value across project complexity levels:

- Low-complexity projects: Readability 4.67/5, Correctness 5.0/5, Educational Value 4.0/5.
- Medium-complexity: Readability 1.67/5, Correctness 4.67/5, Educational Value 2.0/5.
- High-complexity: Readability 1.0/5, Correctness 2.33/5, Educational Value 1.0/5.

They consistently identified excessive jumper wires, convoluted routing, and overlapping components as barriers to classroom usability.

### B.4 Implications

These findings confirm that while functional correctness is a prerequisite, true educational utility also requires readability, process guidance, and layout simplicity. PCEVAL provided the systematic framework to reveal these limitations, highlighting directions for future benchmarks and LLM evaluation methods.

## C Evaluation Metric Details

This appendix details the metrics used to evaluate the outputs generated by LLMs in the PCEVAL benchmark. We assess both the structural integrity of generated circuits and the quality of the generated code, providing a comprehensive view of LLM capabilities in physical computing tasks.

### C.1 Simulator

Reference implementations were validated across real hardware and multiple simulators (Wokwi, Tinkercad, Virtual Breadboard, SimulIDE), ensuring cross-platform consistency. We ultimately selected Wokwi for automated evaluation due to its API accessibility and component coverage, while the framework itself remains simulator-agnostic.

### C.2 Code Quality Validation

Beyond functional correctness determined by simulation success, we assess the quality of LLM-generated code using CodeBLEU (Ren et al., 2020). Table 5 reports aggregate CodeBLEU scores for both code generation tasks across the full benchmark, comparing outputs that led to simulation success with those that failed.

Overall, successful outputs tend to receive higher CodeBLEU scores than unsuccessful ones, but the gaps are often modest. For example, GPT-4.1 on  $D, L \rightarrow C$  shows 0.54 for successful samples and 0.50 for failed ones, while Gemini-3.1-Flash-Lite on  $D, P \rightarrow C$  shows identical aggregate scores of 0.52 for both groups. This pattern suggests that many failed outputs remain close to the reference code in surface form and structure, yet still contain errors that are decisive in physical computing settings, such as incorrect pin assignments, mismatched device initialization, or platform-specific implementation mistakes. CodeBLEU is therefore useful as a complementary quality signal, but it does not substitute for execution-based evaluation.

### C.3 Circuit Validation

Our circuit validation protocol identifies structural errors in both logical and physical circuit designs generated by LLMs. This involves six distinct error checks:

Four metrics apply to both logical and physical circuits:

- **Redundant Connection:** Duplicate connections between the same two component pins (e.g., listing both  $[A, B]$  and  $[B, A]$ ).
- **Extraneous Component:** A component included in the LLM-generated circuit that is not present in the ground truth (reference) circuit.
- **Missing Component:** A component present in the ground truth circuit but omitted from the LLM-generated circuit.
- **Isolated Component:** A component correctly included in the generated circuit but lacking any necessary electrical connections.

For physical circuit evaluation, two additional metrics specifically address implementation constraints on a breadboard:

- **Pin Conflict:** Assigning multiple connections to a single, indivisible breadboard hole or a component pin that cannot accept multiple direct connections.
- **Breadboard Bypass:** Creating direct pin-to-pin connections between components without utilizing the breadboard, violating standard physical prototyping practices for the given tasks.

Table 6 presents the comprehensive circuit validation results, showing the average frequency of these errors across all evaluated models and circuit types.

Table 5: CodeBLEU quality assessment of LLM-generated code, comparing scores between successful and unsuccessful simulation outcomes for different code generation tasks.

| Model                 | $D, L \rightarrow C$ |         | $D, P \rightarrow C$ |         |
|-----------------------|----------------------|---------|----------------------|---------|
|                       | CodeBLEU             |         | CodeBLEU             |         |
|                       | Success              | Failure | Success              | Failure |
| GPT-4o-mini           | 0.50                 | 0.40    | 0.49                 | 0.39    |
| Gemini-2.0-Flash-Lite | 0.56                 | 0.47    | 0.54                 | 0.45    |
| Gemini-2.0-Flash      | 0.54                 | 0.49    | 0.55                 | 0.48    |
| Claude Haiku 4.5      | 0.49                 | 0.45    | 0.46                 | 0.44    |
| GPT-4.1               | 0.54                 | 0.50    | 0.52                 | 0.48    |
| GPT-4o                | 0.52                 | 0.45    | 0.52                 | 0.44    |
| Gemini-3.1-Flash-Lite | 0.53                 | 0.51    | 0.52                 | 0.52    |
| GPT-5-mini            | 0.53                 | 0.46    | 0.52                 | 0.46    |
| o3-mini               | 0.52                 | 0.42    | 0.49                 | 0.45    |
| LLaMA 3.1             | 0.51                 | 0.43    | 0.53                 | 0.36    |
| DeepSeek-Coder-V2     | 0.47                 | 0.41    | 0.50                 | 0.34    |
| Gemma 3               | 0.51                 | 0.44    | 0.54                 | 0.37    |
| Phi 4                 | 0.46                 | 0.43    | 0.48                 | 0.38    |
| Mistral-Small 3       | 0.58                 | 0.46    | 0.58                 | 0.40    |

Table 6: Comprehensive circuit validation results showing average error frequency across all evaluated models, categorized by error type and circuit representation.

| Model                 | Circuit Type | Logical / Physical Errors |                      |                   |                    | Physical Only Errors |                   |
|-----------------------|--------------|---------------------------|----------------------|-------------------|--------------------|----------------------|-------------------|
|                       |              | Redundant Connection      | Extraneous Component | Missing Component | Isolated Component | Pin Conflict         | Breadboard Bypass |
| GPT-4o-mini           | Logical      | 0.06                      | 0.86                 | 0.96              | 0.24               | -                    | -                 |
|                       | Physical     | 0.01                      | 1.68                 | 1.16              | 0.73               | 4.29                 | 4.18              |
| Gemini-2.0-Flash-Lite | Logical      | 0.12                      | 1.43                 | 0.80              | 0.36               | -                    | -                 |
|                       | Physical     | 0.14                      | 2.07                 | 0.89              | 0.26               | 5.87                 | 7.68              |
| Gemini-2.0-Flash      | Logical      | 0.00                      | 0.88                 | 0.87              | 0.06               | -                    | -                 |
|                       | Physical     | 0.00                      | 1.70                 | 0.90              | 0.29               | 3.53                 | 2.19              |
| Claude Haiku 4.5      | Logical      | 0.00                      | 1.59                 | 1.40              | 0.02               | -                    | -                 |
|                       | Physical     | 0.00                      | 2.88                 | 1.47              | 0.11               | 10.28                | 0.56              |
| GPT-4.1               | Logical      | 0.00                      | 0.77                 | 0.79              | 0.02               | -                    | -                 |
|                       | Physical     | 0.26                      | 1.50                 | 0.82              | 0.11               | 7.32                 | 0.03              |
| GPT-4o                | Logical      | 0.00                      | 0.78                 | 0.74              | 0.02               | -                    | -                 |
|                       | Physical     | 0.02                      | 1.35                 | 0.70              | 0.52               | 2.10                 | 1.03              |
| Gemini-3.1-Flash-Lite | Logical      | 0.00                      | 1.44                 | 1.39              | 0.01               | -                    | -                 |
|                       | Physical     | 0.01                      | 3.17                 | 1.71              | 0.03               | 2.12                 | 2.32              |
| GPT-5-mini            | Logical      | 0.00                      | 2.73                 | 2.11              | 0.01               | -                    | -                 |
|                       | Physical     | 0.04                      | 4.16                 | 2.12              | 0.25               | 1.03                 | 0.01              |
| o3-mini               | Logical      | 0.01                      | 1.23                 | 0.83              | 0.01               | -                    | -                 |
|                       | Physical     | 0.01                      | 1.88                 | 0.98              | 0.01               | 4.17                 | 0.01              |
| LLaMA 3.1             | Logical      | 0.13                      | 1.74                 | 2.50              | 0.61               | -                    | -                 |
|                       | Physical     | 0.03                      | 1.22                 | 1.74              | 0.06               | 1.39                 | 1.54              |
| DeepSeek-Coder-V2     | Logical      | 0.04                      | 1.55                 | 2.07              | 0.43               | -                    | -                 |
|                       | Physical     | 0.01                      | 1.48                 | 3.07              | 0.24               | 1.14                 | 2.70              |
| Gemma 3               | Logical      | 0.06                      | 1.30                 | 0.93              | 0.39               | -                    | -                 |
|                       | Physical     | 0.00                      | 1.76                 | 1.02              | 0.65               | 4.75                 | 1.14              |
| Phi 4                 | Logical      | 0.05                      | 0.94                 | 0.78              | 0.03               | -                    | -                 |
|                       | Physical     | 0.04                      | 1.45                 | 1.10              | 0.23               | 4.81                 | 1.62              |
| Mistral-Small 3       | Logical      | 0.10                      | 0.84                 | 1.04              | 0.07               | -                    | -                 |
|                       | Physical     | 0.29                      | 1.57                 | 1.15              | 0.10               | 2.37                 | 0.77              |

## D Exploratory Analyses on the Arduino Uno Setting

The analyses in this section focus on the 50 Arduino Uno projects and should be interpreted as diagnostic follow-up analyses rather than additional cross-platform benchmark results. We use Arduino Uno because it is the benchmark’s canonical instructional setting, and because some of these analyses require additional inference runs or are reported for a selected set of models. Our main claims remain based on the full-benchmark results in Table 3, while Appendix G shows that the overall benchmark trends are broadly consistent across Arduino Uno and ESP32.

### D.1 Self-Improvement and Chain-of-Thought

Tables 7 and 8 report exploratory mitigation results in the Arduino Uno setting. These analyses test whether multi-turn prompting can reduce errors in the benchmark’s canonical setting; they are not additional cross-platform comparisons.

As discussed in Section 4.3, the main trends are consistent across models. Self-improvement yields robust and steady performance gains, especially in code-related tasks. CoT exhibits more variable benefits and can even degrade performance on hardware tasks.

Table 7: Full results of self-improvement experiments on the Arduino Uno setting (success rates, %).

| Model            | Score                   | $D, C \rightarrow L$    | $D, C \rightarrow P$    | $D, L \rightarrow C$    | $D, P \rightarrow C$    |
|------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|
| o3-mini          | 66.2 $\rightarrow$ 76.5 | 70.8 $\rightarrow$ 79.6 | 48.0 $\rightarrow$ 58.0 | 72.8 $\rightarrow$ 83.6 | 73.2 $\rightarrow$ 84.8 |
| Gemini-2.0 Flash | 50.1 $\rightarrow$ 57.2 | 63.2 $\rightarrow$ 73.2 | 21.2 $\rightarrow$ 16.8 | 59.6 $\rightarrow$ 72.8 | 56.4 $\rightarrow$ 66.0 |

Table 8: Full results of CoT prompting experiments on the Arduino Uno setting (success rates, %).

| Model             | $D, C \rightarrow P$    | $D, P \rightarrow C$    |
|-------------------|-------------------------|-------------------------|
| Mistral-Small 24B | 14.4 $\rightarrow$ 11.6 | 32.0 $\rightarrow$ 48.8 |
| GPT-4o            | 28.0 $\rightarrow$ 29.6 | 61.2 $\rightarrow$ 66.8 |
| GPT-4o-mini       | 1.2 $\rightarrow$ 2.4   | 54.0 $\rightarrow$ 52.0 |
| Phi-4             | 3.2 $\rightarrow$ 3.2   | 37.2 $\rightarrow$ 47.2 |
| LLaMA 3.1         | 2.0 $\rightarrow$ 5.2   | 24.8 $\rightarrow$ 22.4 |
| Gemma 3           | 2.4 $\rightarrow$ 2.8   | 34.4 $\rightarrow$ 45.2 |
| DeepSeek Coder V2 | 1.2 $\rightarrow$ 0.0   | 20.8 $\rightarrow$ 32.4 |

### D.2 Pass@5

We additionally report pass@5 for six selected models in the Arduino Uno setting. Table 9 summarizes these results. This metric is computed from five independent samples per project-task pair and is intended as a complementary view of sampling variance in the benchmark’s canonical setting, rather than as a replacement for the full-benchmark mean success rates reported in Table 3.

Table 9: Pass@5 results on the Arduino Uno setting, computed from five independent samples per project-task pair.

| Model                 | $D, C \rightarrow L$ | $D, C \rightarrow P$ | $D, L \rightarrow C$ | $D, P \rightarrow C$ | Arduino Avg. |
|-----------------------|----------------------|----------------------|----------------------|----------------------|--------------|
| Gemini-2.0-Flash-Lite | 79.2                 | 16.7                 | 70.8                 | 75.0                 | 60.4         |
| Claude 3.7 Sonnet     | 87.5                 | 50.0                 | 83.3                 | 83.3                 | 76.0         |
| GPT-4.1               | 87.5                 | 54.2                 | 83.3                 | 87.5                 | 78.1         |
| Gemma 3               | 70.8                 | 8.3                  | 41.7                 | 33.3                 | 38.5         |
| Phi 4                 | 66.7                 | 4.2                  | 62.5                 | 58.3                 | 47.9         |
| Mistral-Small 3       | 75.0                 | 13.6                 | 62.5                 | 50.0                 | 50.2         |

### 614 D.3 Temperature Hyperparameter Settings

615 All main experiments used structured-text inputs, and the complete prompts are given in Appendix M.  
 616 We used a maximum generation limit of 4,096 tokens. For temperature, the main benchmark used each  
 617 provider’s default setting rather than imposing a single value across all APIs. The robustness checks  
 618 below were then run on the Arduino Uno setting to test whether the main qualitative conclusions  
 619 depend strongly on reasonable decoding or prompt variations.

620 Table 10 shows that varying temperature between 0.1 and 0.3 leads to only modest score fluctuations  
 621 for Phi 4 and Mistral-Small 3. The overall qualitative picture is unchanged: task difficulty ordering  
 622 remains stable, and physical circuit generation remains the hardest task category.

Table 10: Temperature sensitivity analysis on the Arduino Uno setting (success rates, %).

| Temp. | Model           | Arduino Avg. | $D, C \rightarrow L$ | $D, C \rightarrow P$ | $D, L \rightarrow C$ | $D, P \rightarrow C$ |
|-------|-----------------|--------------|----------------------|----------------------|----------------------|----------------------|
| 0.1   | Phi 4           | 35.5         | 36.4                 | 2.4                  | 48.8                 | 54.4                 |
| 0.1   | Mistral-Small 3 | 41.9         | 50.0                 | 16.0                 | 52.0                 | 49.6                 |
| 0.2   | Phi 4           | 34.8         | 44.2                 | 0.8                  | 51.7                 | 42.5                 |
| 0.2   | Mistral-Small 3 | 40.1         | 57.5                 | 13.3                 | 56.7                 | 40.8                 |
| 0.3   | Phi 4           | 36.0         | 42.0                 | 2.8                  | 50.4                 | 48.8                 |
| 0.3   | Mistral-Small 3 | 40.7         | 49.6                 | 10.4                 | 51.2                 | 51.6                 |

## 623 E PCEval Benchmark Projects

624 This appendix provides a detailed overview of all projects included in the PCEval benchmark  
625 dataset. The projects are categorized by complexity level, outlining the specific tasks and hardware  
626 components involved in each.

627 While the absolute number of projects (75) may appear modest in isolation, the design of PCEVAL  
628 ensures a sufficiently large and diverse evaluation space. Each project supports four distinct generation  
629 tasks, resulting in 300 unique evaluation samples. This scale is comparable to established software  
630 benchmarks such as HumanEval, which includes 164 problems for code-only tasks (Chen et al.,  
631 2021). In physical computing, dataset construction is substantially more complex than in code-only  
632 settings, since each instance requires not only executable code but also consistent logical and physical  
633 circuit specifications, validated across both simulation environments and real hardware.

634 The most directly related prior work, MICRO25 (Jansen, 2023), introduced 25 projects with nat-  
635 ural language descriptions. However, MICRO25 provides no reference implementations (code or  
636 circuits) and relies entirely on manual expert evaluation, limiting reproducibility and scalability. In  
637 contrast, PCEVAL doubles the number of projects (75), and each project includes complete reference  
638 implementations—code, logical circuits, and physical circuits—supporting fully automated and  
639 reproducible evaluation across 300 task instances.

Table 11: Overview of PCEval Benchmark Arduino Projects by Level

| Level   | Project Name              | Description                                                                                                                                            |
|---------|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| Level 1 | 7 segment display basic   | Connect a 7-segment display to Arduino and write code to show the number 5.                                                                            |
|         | bar led basic             | Connect a bar graph LED to Arduino and write code to turn on LEDs 1 through 6.                                                                         |
|         | buzzer basic              | Connect a piezo buzzer to Arduino and write code to play a sound.                                                                                      |
|         | distance sensor basic     | Connect an ultrasonic distance sensor to Arduino and write code to measure the distance and print it to the serial monitor.                            |
|         | humidity sensor basic     | Connect a DHT22 sensor to Arduino and write code to measure the humidity and print it to the serial monitor.                                           |
|         | LCD display basic         | Connect an ILI9341 LCD display to Arduino and write code to show “Hello World!” on the display.                                                        |
|         | led blink basic           | Connect an LED to Arduino and write code to repeatedly turn the LED on and off with delays.                                                            |
|         | led RGB basic             | Connect an RGB LED to Arduino and write code to turn on only the blue segment.                                                                         |
|         | photoresistor basic       | Connect a photoresistor sensor module to Arduino and write code to detect light intensity and print it to the serial monitor.                          |
|         | RTC module basic          | Connect an RTC module to Arduino and write code to read the current time and print it to the serial monitor.                                           |
|         | servo motor basic         | Connect a servo motor to Arduino and write code to repeatedly move the servo to 0° and 180°.                                                           |
|         | temperature sensor basic  | Connect a DHT22 sensor to Arduino and write code to measure the temperature and print it to the serial monitor.                                        |
| Level 2 | 7 segment display counter | Connect a 7-segment display to Arduino and write code to show 1, 2, and 3 based on elapsed time.                                                       |
|         | accelerometer             | Connect an MPU6050 accelerometer to Arduino and write code to measure acceleration on the X, Y, and Z axes and print the values to the serial monitor. |
|         | button duration           | Connect a push button to Arduino and write code to measure how long the button is pressed and print the duration to the serial monitor.                |

Continued on next page

**Table 11 – continued from previous page**

| <b>Level</b> | <b>Project Name</b>       | <b>Description</b>                                                                                                                                             |
|--------------|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
|              | button pulldown           | Connect a push button to Arduino with a pulldown resistor and write code to monitor the button state and print it to the serial monitor.                       |
|              | button pullup             | Connect a push button to Arduino with a pullup resistor and write code to monitor the button state and print it to the serial monitor.                         |
|              | distance sensor           | Connect an ultrasonic distance sensor to Arduino and write code to measure the distance and compare it with a threshold.                                       |
|              | gyroscope                 | Connect an MPU6050 gyroscope to Arduino and write code to measure rotation on the X, Y, and Z axes and print the values to the serial monitor.                 |
|              | humidity sensor           | Connect a DHT22 sensor to Arduino and write code to measure the humidity and compare it with a threshold.                                                      |
|              | serial bar led            | Connect a bar graph LED to Arduino and write code to read serial input and light up the corresponding number of LEDs.                                          |
|              | serial LCD display        | Connect an ILI9341 LCD display to Arduino and write code to read serial input and display it on the LCD.                                                       |
|              | serial monitor            | Connect nothing to Arduino and write code to read serial input and print it back to the serial monitor.                                                        |
|              | serial RGB led            | Connect an RGB LED to Arduino and write code to read serial input and turn on the corresponding RGB segment.                                                   |
|              | temperature sensor        | Connect a DHT22 sensor to Arduino and write code to measure the temperature and compare it with a threshold.                                                   |
| Level 3      | 4 digit 7 segment display | Connect a 4-digit 7-segment display to Arduino and write code to show 1, 2, and 3 based on elapsed time.                                                       |
|              | 7 segment display serial  | Connect a 7-segment display to Arduino and write code to show serial input on the display.                                                                     |
|              | button buzzer             | Connect a push button and piezo buzzer to Arduino and write code to play a sound when the button is pressed.                                                   |
|              | button LCD display        | Connect a push button and ILI9341 display to Arduino and write code to show the button state on the LCD.                                                       |
|              | button led                | Connect a push button and LED to Arduino and write code to turn on the LED when the button is pressed.                                                         |
|              | button RGB led            | Connect three push buttons (red, green, blue) and an RGB LED to Arduino and write code to turn on the corresponding color segment when each button is pressed. |
|              | button RTC timezone       | Connect a push button and RTC module to Arduino and write code to print the time in different timezones based on the button state.                             |
|              | button servo motor        | Connect a push button and servo motor to Arduino and write code to change the servo motor angle using the button.                                              |
|              | dht22 LCD display         | Connect a DHT22 sensor and an ILI9341 LCD display to Arduino and write code to show humidity and temperature values on the LCD.                                |
|              | multiplexer photoresistor | Connect six photoresistors to Arduino via a 16-channel analog multiplexer and write code to print each sensor's value.                                         |
|              | multiplexer potentiometer | Connect five potentiometers to Arduino via a 16-channel analog multiplexer and write code to print each potentiometer's value.                                 |
|              | photoresistor bar led     | Connect a bar graph LED and photoresistor to Arduino and write code to light up the bar LEDs based on the mapped value of the photoresistor.                   |
|              | potentiometer bar led     | Connect a bar graph LED and potentiometer to Arduino and write code to light up the bar LEDs based on the mapped value of the potentiometer.                   |

Continued on next page

**Table 11 – continued from previous page**

| <b>Level</b> | <b>Project Name</b>       | <b>Description</b>                                                                                                                                                        |
|--------------|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|              | potentiometer servo motor | Connect a potentiometer and servo motor to Arduino and write code to set the servo motor angle based on the mapped potentiometer value.                                   |
| Level 4      | alarm                     | Connect a piezo buzzer and RTC module to Arduino and write code to play a sound when the specified time is reached.                                                       |
|              | binary led                | Connect five LEDs to Arduino and write code to show the binary representation of serial input using the LEDs.                                                             |
|              | calendar display          | Connect an RTC module, an ILI9341 display, and two push buttons to Arduino and write code to show the current date on the LCD. Use the buttons to navigate between dates. |
|              | clock                     | Connect a 4-digit 7-segment display and RTC module to Arduino and write code to show the current time in HH:MM format.                                                    |
|              | dday counter              | Connect an RTC module to Arduino and write code to read a date from serial input and print the number of days remaining until that date.                                  |
|              | exercise counter          | Connect an MPU6050 sensor and 7-segment display to Arduino and write code to count each time the Y-axis rotation exceeds a threshold, then show the count on the display. |
|              | multiple timezone         | Connect an RTC module and four push buttons to Arduino and write code to display different timezones depending on which button is pressed.                                |
|              | parking spot monitor      | Connect an ultrasonic distance sensor and RGB LED to Arduino and write code to measure distance and switch from green to red light if the distance is below a threshold.  |
|              | piano keyboard            | Connect four push buttons and four piezo buzzers to Arduino and write code to play a tone when the corresponding button is pressed.                                       |
|              | smart led system          | Connect five LEDs and five photoresistors to Arduino and write code to turn on each LED when its corresponding photoresistor reads below a threshold.                     |
|              | traffic light             | Connect a push button and three LEDs to Arduino and write code to turn on the green LED by default, then switch to yellow and red in order when the button is pressed.    |

Table 12: Overview of PCEval Benchmark ESP32 Projects by Level

| Level   | Project Name              | Description                                                                                                                                                  |
|---------|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Level 1 | 7 segment display basic   | Connect a 7-segment display to ESP32 and write code to show the number 5.                                                                                    |
|         | bar led basic             | Connect a bar graph LED to ESP32 and write code to turn on LEDs 1 through 6.                                                                                 |
|         | buzzer basic              | Connect a piezo buzzer to ESP32 and write code to play a sound.                                                                                              |
|         | distance sensor basic     | Connect an ultrasonic distance sensor to ESP32 and write code to measure the distance and print it to the serial monitor.                                    |
|         | humidity sensor basic     | Connect a DHT22 sensor to ESP32 and write code to measure the humidity and print it to the serial monitor.                                                   |
|         | LCD display basic         | Connect an ILI9341 LCD display to ESP32 and write code to show "Hello World!" on the display.                                                                |
|         | led blink basic           | Connect an LED to ESP32 and write code to repeatedly turn the LED on and off with delays.                                                                    |
|         | led RGB basic             | Connect an RGB LED to ESP32 and write code to turn on only the blue segment.                                                                                 |
|         | photoresistor basic       | Connect a photoresistor sensor module to ESP32 and write code to detect light intensity and print it to the serial monitor.                                  |
|         | RTC module basic          | Connect an RTC module to ESP32 and write code to read the current time and print it to the serial monitor.                                                   |
| Level 2 | 7 segment display counter | Connect a 7-segment display to ESP32 and write code to show 1, 2, and 3 based on elapsed time.                                                               |
|         | accelerometer             | Connect an MPU6050 accelerometer to ESP32 and write code to measure acceleration on the X, Y, and Z axes and print the values to the serial monitor.         |
|         | button duration           | Connect a push button to ESP32 and write code to measure how long the button is pressed and print the duration to the serial monitor.                        |
|         | button pulldown           | Connect a push button to ESP32 with a pulldown resistor and write code to monitor the button state and print it to the serial monitor.                       |
|         | button pullup             | Connect a push button to ESP32 with a pullup resistor and write code to monitor the button state and print it to the serial monitor.                         |
|         | distance sensor           | Connect an ultrasonic distance sensor to ESP32 and write code to measure the distance and compare it with a threshold.                                       |
|         | gyroscope                 | Connect an MPU6050 gyroscope to ESP32 and write code to measure rotation on the X, Y, and Z axes and print the values to the serial monitor.                 |
|         | humidity sensor           | Connect a DHT22 sensor to ESP32 and write code to measure the humidity and compare it with a threshold.                                                      |
| Level 3 | button led                | Connect a push button and LED to ESP32 and write code to turn on the LED when the button is pressed.                                                         |
|         | button RGB led            | Connect three push buttons (red, green, blue) and an RGB LED to ESP32 and write code to turn on the corresponding color segment when each button is pressed. |
|         | photoresistor bar led     | Connect a bar graph LED and photoresistor to ESP32 and write code to light up the bar LEDs based on the mapped value of the photoresistor.                   |
|         | potentiometer bar led     | Connect a bar graph LED and potentiometer to ESP32 and write code to light up the bar LEDs based on the mapped value of the potentiometer.                   |

Continued on next page

**Table 12 – continued from previous page**

| <b>Level</b> | <b>Project Name</b>  | <b>Description</b>                                                                                                                                                      |
|--------------|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Level 4      | binary led           | Connect five LEDs to ESP32 and write code to show the binary representation of serial input using the LEDs.                                                             |
|              | calendar display     | Connect an RTC module, an ILI9341 display, and two push buttons to ESP32 and write code to show the current date on the LCD. Use the buttons to navigate between dates. |
|              | dday counter         | Connect an RTC module to ESP32 and write code to read a date from serial input and print the number of days remaining until that date.                                  |
|              | parking spot monitor | Connect an ultrasonic distance sensor and RGB LED to ESP32 and write code to measure distance and switch from green to red light if the distance is below a threshold.  |

## F Why Direct Comparisons to Classical Design-Automation Methods Are Not Straightforward

An important question is whether PCEVAL should be compared against classical symbolic, search-based, or design-automation methods. Such context is useful. The key issue, however, is that PCEVAL evaluates end-to-end generation from partially specified educational intents, whereas classical methods are typically applied after the problem has already been formalized as a truth table, netlist, placement instance, or routing problem (Brayton et al., 1984; Kirkpatrick et al., 1983; Lee, 1961; Hart et al., 1968).

In other words, classical methods typically start from a formalized design instance in which the component set and constraint structure are already specified, whereas PCEVAL still requires the system to infer those structures from partially specified educational problems. This distinction affects all four benchmark tasks. For  $D, C \rightarrow L$  and  $D, C \rightarrow P$ , the system must infer the hardware specification itself from underspecified language and code. For  $D, L \rightarrow C$  and  $D, P \rightarrow C$ , the system must synthesize executable code that is consistent not only with the intended behavior but also with the specific circuit representation, including concrete pin assignments in the physical-layout case. In particular, PCEVAL’s physical-layout task is not a standard PCB or VLSI routing problem: the output is a breadboard-level educational artifact with shared-row topology, power rails, single-hole occupancy constraints, and required compatibility with Arduino code and project semantics.

Classical routing or optimization methods can still be useful after such representations exist, but they do not by themselves recover project intent, choose components, or generate microcontroller code. A fair non-LLM baseline would therefore be a pipeline rather than a single solver. It would likely need to combine semantic parsing of the project description, component retrieval, rule- or constraint-based circuit synthesis, board-aware placement or routing, code templates or program synthesis, and simulator-in-the-loop validation. We view such hybrid systems as a valuable direction for future work, and PCEVAL can evaluate them once they are extended to cover the full end-to-end task definition.

Table 13: Classical methods vs. PCEVAL.

| Aspect | Classical methods                                                   | PCEVAL tasks                                           |
|--------|---------------------------------------------------------------------|--------------------------------------------------------|
| Input  | Formal specs such as truth tables, netlists, or routing constraints | Natural-language descriptions plus a partial artifact  |
| Task   | Optimize or validate a formalized design instance                   | Infer missing structure and generate a new artifact    |
| Output | Optimized logic, placement, or routing result                       | Logical circuit, breadboard layout, or executable code |

Table 14: Platform-specific success rates (%) on the Arduino Uno subset (50 projects), corresponding to the Arduino portion of Table 3.

| Model                 | Param. | Circuit Gen.         |                      |         | Code Gen.            |                      |         | Total Overall |
|-----------------------|--------|----------------------|----------------------|---------|----------------------|----------------------|---------|---------------|
|                       |        | $D, C \rightarrow L$ | $D, C \rightarrow P$ | Overall | $D, L \rightarrow C$ | $D, P \rightarrow C$ | Overall |               |
| Closed Source LLMs    |        |                      |                      |         |                      |                      |         |               |
| GPT-4o-mini           | -      | 51.6                 | 1.2                  | 26.4    | 52.8                 | 54.0                 | 53.4    | 39.9          |
| Gemini-2.0-Flash-Lite | -      | 52.8                 | 2.4                  | 27.6    | 56.4                 | 56.8                 | 56.6    | 42.1          |
| Claude 3.5 Haiku      | -      | 51.2                 | 1.6                  | 26.4    | 66.0                 | 60.8                 | 63.4    | 44.9          |
| Gemini-2.0-Flash      | -      | 63.2                 | 21.2                 | 42.2    | 59.6                 | 56.4                 | 58.0    | 50.1          |
| Claude Haiku 4.5      | -      | 60.4                 | 4.0                  | 32.2    | 72.8                 | 65.6                 | 69.2    | 50.7          |
| GPT-4.1               | -      | 54.0                 | 13.6                 | 33.8    | 69.6                 | 67.2                 | 68.4    | 51.1          |
| GPT-4o                | -      | 62.4                 | 28.0                 | 45.2    | 65.6                 | 61.2                 | 63.4    | 54.3          |
| Claude 3.7 Sonnet     | -      | 71.2                 | 13.6                 | 42.4    | 67.6                 | 70.0                 | 68.8    | 55.6          |
| Gemini-3.1-Flash-Lite | -      | 74.4                 | 24.8                 | 49.6    | 74.0                 | 72.8                 | 73.4    | 61.5          |
| GPT-5-mini            | -      | 58.8                 | 44.0                 | 51.4    | 70.8                 | 74.0                 | 72.4    | 61.9          |
| o3-mini               | -      | 70.8                 | 48.0                 | 59.4    | 72.8                 | 73.2                 | 73.0    | 66.2          |
| Open Source LLMs      |        |                      |                      |         |                      |                      |         |               |
| LLaMA 3.1             | 8B     | 23.2                 | 2.0                  | 12.6    | 26.0                 | 24.8                 | 25.4    | 19.0          |
| DeepSeek-Coder-V2     | 16B    | 27.6                 | 1.2                  | 14.4    | 30.0                 | 20.8                 | 25.4    | 19.9          |
| Gemma 3               | 27B    | 51.2                 | 2.4                  | 26.8    | 29.6                 | 34.4                 | 32.0    | 29.4          |
| Phi 4                 | 14B    | 33.6                 | 3.2                  | 18.4    | 48.8                 | 37.2                 | 43.0    | 30.7          |
| Mistral-Small 3       | 24B    | 50.8                 | 14.4                 | 32.6    | 49.2                 | 32.0                 | 40.6    | 36.6          |

Table 15: Platform-specific success rates (%) on the ESP32 projects (25 projects), corresponding to the ESP32 portion of Table 3.

| Model                 | Param. | Circuit Gen.         |                      |         | Code Gen.            |                      |         | Total Overall |
|-----------------------|--------|----------------------|----------------------|---------|----------------------|----------------------|---------|---------------|
|                       |        | $D, C \rightarrow L$ | $D, C \rightarrow P$ | Overall | $D, L \rightarrow C$ | $D, P \rightarrow C$ | Overall |               |
| Closed Source LLMs    |        |                      |                      |         |                      |                      |         |               |
| GPT-4o-mini           | -      | 49.6                 | 1.6                  | 25.6    | 57.6                 | 52.8                 | 55.2    | 40.4          |
| Gemini-2.0-Flash-Lite | -      | 48.0                 | 3.2                  | 25.6    | 56.0                 | 58.4                 | 57.2    | 41.4          |
| Claude 3.5 Haiku      | -      | -                    | -                    | -       | -                    | -                    | -       | -             |
| Gemini-2.0-Flash      | -      | 55.2                 | 11.2                 | 33.2    | 60.0                 | 54.4                 | 57.2    | 45.2          |
| Claude Haiku 4.5      | -      | 53.6                 | 3.2                  | 28.4    | 61.6                 | 61.6                 | 61.6    | 45.0          |
| GPT-4.1               | -      | 54.4                 | 20.8                 | 37.6    | 67.2                 | 55.2                 | 61.2    | 49.4          |
| GPT-4o                | -      | 62.4                 | 29.6                 | 46.0    | 64.8                 | 60.8                 | 62.8    | 54.4          |
| Claude 3.7 Sonnet     | -      | -                    | -                    | -       | -                    | -                    | -       | -             |
| Gemini-3.1-Flash-Lite | -      | 65.6                 | 31.2                 | 48.4    | 68.8                 | 60.8                 | 64.8    | 56.6          |
| GPT-5-mini            | -      | 52.0                 | 46.4                 | 49.2    | 71.2                 | 69.6                 | 70.4    | 59.8          |
| o3-mini               | -      | 54.4                 | 43.2                 | 48.8    | 68.8                 | 67.2                 | 68.0    | 58.4          |
| Open Source LLMs      |        |                      |                      |         |                      |                      |         |               |
| LLaMA 3.1             | 8B     | 26.4                 | 0.8                  | 13.6    | 36.8                 | 28.0                 | 32.4    | 23.0          |
| DeepSeek-Coder-V2     | 16B    | 44.8                 | 0.0                  | 22.4    | 42.4                 | 21.6                 | 32.0    | 27.2          |
| Gemma 3               | 27B    | 56.0                 | 0.0                  | 28.0    | 48.8                 | 43.2                 | 46.0    | 37.0          |
| Phi 4                 | 14B    | 44.0                 | 0.8                  | 22.4    | 49.6                 | 39.2                 | 44.4    | 33.4          |
| Mistral-Small 3       | 24B    | 52.8                 | 13.6                 | 33.2    | 48.8                 | 43.2                 | 46.0    | 39.6          |

## G Platform Specific Breakdown of Main Results

Table 3 reports aggregate scores over the full PCEVAL benchmark. Tables 14 and 15 separate those scores by platform. They use the same task definitions and evaluation metrics for Arduino Uno and ESP32. The prompt structure was aligned across platforms, with small platform specific substitutions described in Appendix M.

**Notes on Model Availability** Some models reported in the main table were evaluated only on the Arduino Uno subset. In particular, Claude 3.5 Haiku and Claude 3.7 Sonnet became unavailable before the ESP32 runs were conducted. Tables 14 and 15 therefore report only their Arduino Uno results. The corresponding ESP32 entries are marked as unavailable.

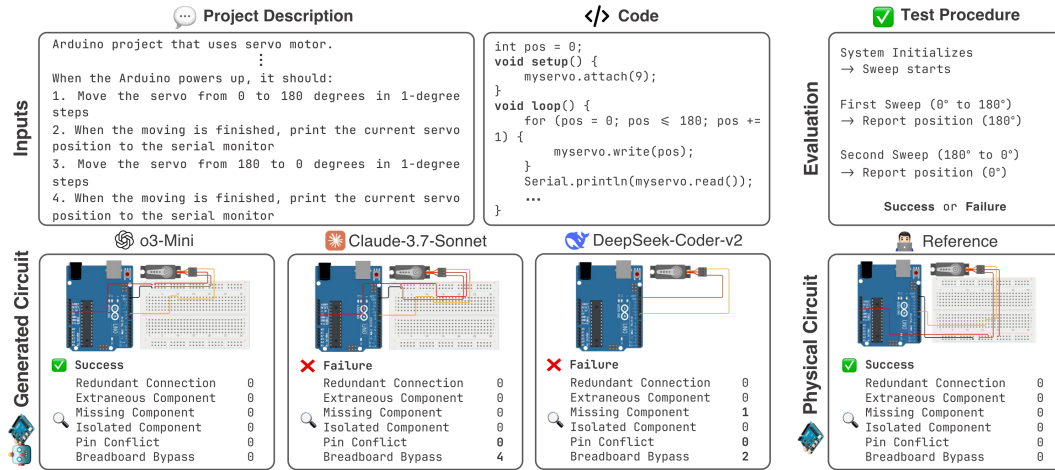


Figure 4: Qualitative examples of LLM-generated Arduino physical circuits for a servo motor basic project (level 1), illustrating successful and failed attempts with corresponding error analyses.

## H System Specifications for Evaluation

The primary evaluations of Large Language Models (LLMs) for the PCEVAL benchmark were conducted by querying their respective APIs. For any local processing, data analysis, and interaction with these APIs, the following system configuration was utilized:

- **CPU:** AMD EPYC 9354 32-Core Processor (1 Socket, 32 Cores, 64 Threads).
- **GPU:** NVIDIA RTX 6000 Ada Generation.
  - GPU Memory: 49140 MiB
  - NVIDIA Driver Version: 535.230.02
  - CUDA Version: 12.2
- **System Memory (RAM):** 125 GiB.
- **Operating System Environment:** Linux-based (x86\_64 architecture).

This configuration provided a stable and capable environment for managing the data, running evaluation scripts, and interfacing with the LLM APIs. The specific computational requirements for querying each LLM API varied depending on the model provider and were not a limiting factor in this local setup.

## I More Qualitative Results

This section presents additional qualitative results to provide a more nuanced understanding of LLM performance in physical circuit generation. The Figures 4, 5 and 6 showcase visual examples from specific PCEVAL projects, illustrating both successful circuit implementations and common failure modes encountered by the LLMs. These examples serve to complement the quantitative evaluations by offering concrete instances of the models' outputs and the challenges they face.

## J Visualizing the Impact of Project Complexity Metrics on LLM Performance

We present a visual analysis of the relationship between project complexity metrics and LLM performance across four physical computing tasks (Figure 7). The visualizations plot project-level success rates (0.0–1.0) against two complexity metrics, number of connections and lines of code. Each point shows the average performance of five selected models, namely Gemini-3.1-Flash-Lite, GPT-5-mini, o3-mini, Phi 4, and Mistral-Small 3, on one Arduino Uno or ESP32 project. Projects are color-coded by complexity level (1–4), and the linear regression lines (in red) summarize the overall trends.

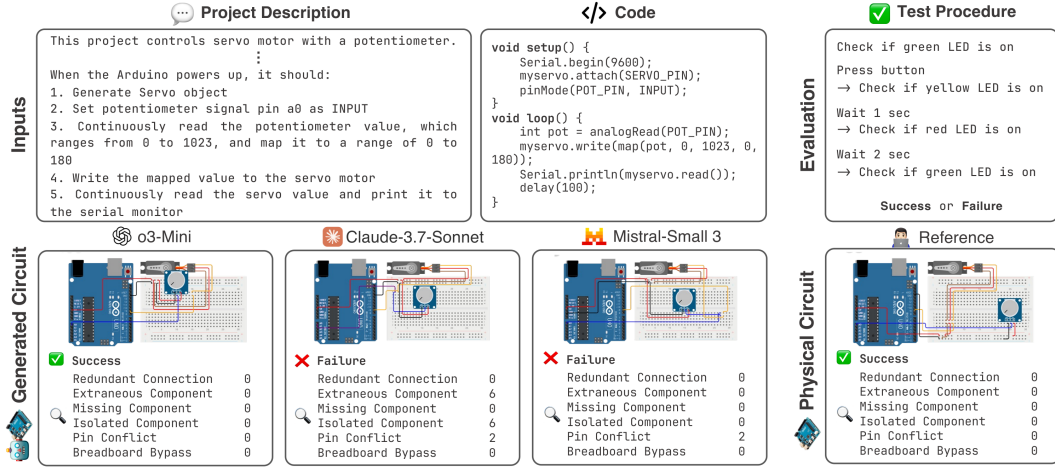


Figure 5: Qualitative examples of LLM-generated Arduino physical circuits for a potentiometer servo motor project (level 3), illustrating successful and failed attempts with corresponding error analyses.

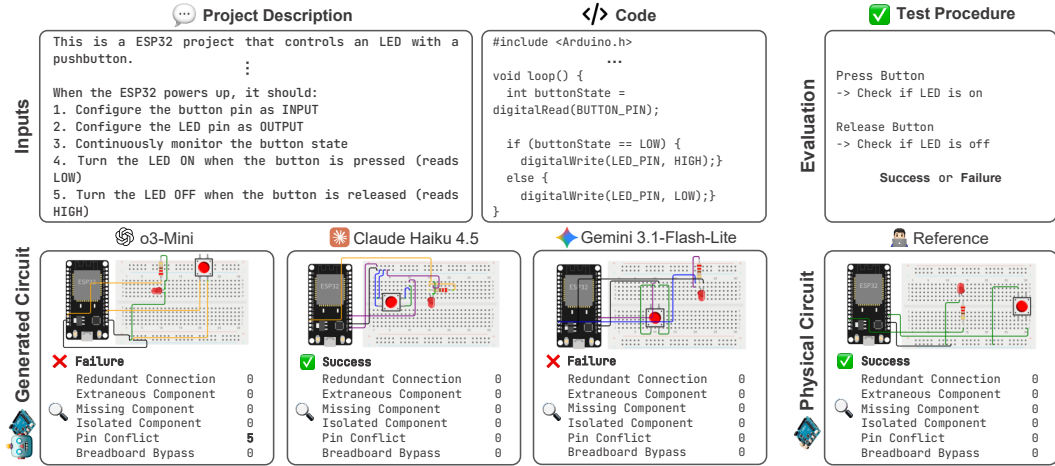


Figure 6: Qualitative examples of LLM-generated ESP-32 physical circuits for a button LED project (level 3), illustrating successful and failed attempts with corresponding error analyses.

## 704 J.1 Circuit Generation Tasks

705 Figures 7a and 7b present results for Logical Circuit Generation ( $D, C \rightarrow L$ ) and Physical Circuit  
 706 Generation ( $D, C \rightarrow P$ ), respectively. Both tasks exhibit negative correlations between success rates  
 707 and the two complexity metrics. In physical circuit generation, the decline is slightly steeper with  
 708 connection count than with code length, indicating that denser breadboard layouts remain especially  
 709 challenging. In logical circuit generation, both metrics are associated with lower success, with code  
 710 length showing a comparable and slightly stronger trend than logical connection count. Overall,  
 711 more complex projects reduce circuit-generation performance regardless of whether complexity is  
 712 expressed in wiring or code.

## 713 J.2 Code Generation Tasks

714 For Code Generation from Logical Circuit ( $D, L \rightarrow C$ ) and Code Generation from Physical Circuit ( $D,$   
 715  $P \rightarrow C$ ) (Figures 7c and 7d), the number of code lines shows a markedly steeper negative relationship  
 716 with success rates than either logical or physical connection counts. This pattern suggests that, in the  
 717 code-generation settings, longer and more complex programs are a stronger source of difficulty than  
 718 the complexity of the input circuit representation.

### 719 J.3 Summary

720 Our analysis reveals that LLMs exhibit task-specific sensitivities to different dimensions of complexity.  
721 Code-generation performance is most strongly tied to program length, whereas circuit-generation  
722 performance is affected by both wiring complexity and code complexity. Across all four tasks,  
723 however, increased project complexity is consistently associated with lower success.

## 724 K Visualizing the Impact of Project Level on LLM Performance

725 Figure 8 visualizes the distribution of project-level success rates across the four PCEVAL tasks,  
726 revealing how performance varies both within and across complexity levels after averaging over the  
727 same five selected models.

### 728 K.1 Circuit Generation Tasks

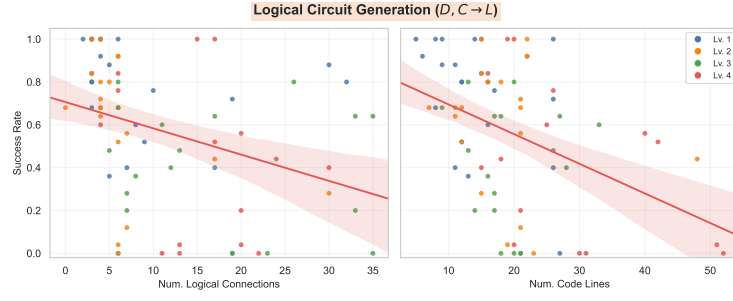
729 **Logical Circuit Generation (Figure 8a):** Level 1 shows the highest and most consistent performance  
730 (median  $\approx 0.80$ ), followed by a decline at Level 2 (median  $\approx 0.68$ ). Levels 3 and 4 both center  
731 around substantially lower medians (each  $\approx 0.44$ ), and both levels exhibit broad dispersion rather  
732 than a tight cluster of outcomes. This suggests that logical circuit generation becomes notably less  
733 reliable once projects move beyond the simpler introductory levels.

734 **Physical Circuit Generation (Figure 8b):** This task exhibits the weakest overall performance and  
735 the clearest degradation with complexity. The median drops from approximately 0.58 at Level 1 to  
736 0.44 at Level 2, 0.38 at Level 3, and 0.28 at Level 4. Although there is still substantial variation within  
737 each level, the entire distribution shifts downward as projects become more complex, indicating  
738 persistent difficulty with physically valid circuit layout.

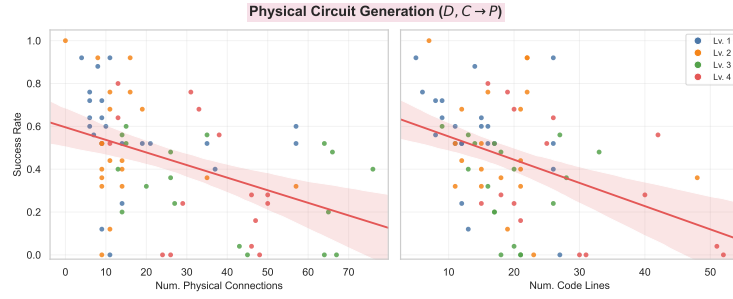
### 739 K.2 Code Generation Tasks

740 **Code Generation from Logical Circuit (Figure 8c):** The box plot shows a clear downward trend in  
741 median performance from Level 1 (median  $\approx 0.84$ ) to Level 2 (median  $\approx 0.74$ ), Level 3 (median  
742  $\approx 0.62$ ), and Level 4 (median  $\approx 0.36$ ). Levels 1 and 2 remain comparatively strong, but Levels 3  
743 and 4 show much broader spreads, especially at Level 4 where the interquartile range extends from  
744 complete failure to relatively strong outcomes. This indicates that code generation from logical  
745 circuits remains feasible on simpler projects but becomes increasingly unstable on the most complex  
746 tasks.

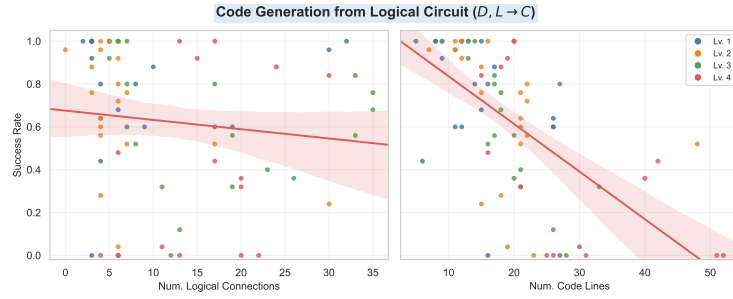
747 **Code Generation from Physical Circuit (Figure 8d):** This task follows a similar pattern, but with  
748 a slightly different profile across the easier levels. The medians are approximately 0.64 at Level 1,  
749 0.70 at Level 2, 0.60 at Level 3, and 0.32 at Level 4. Performance therefore remains relatively stable  
750 through Levels 1 and 2 before dropping more substantially at Levels 3 and 4. At the same time, the  
751 presence of failures even at lower levels indicates that interpreting physical circuits can still break  
752 down on otherwise simple projects.



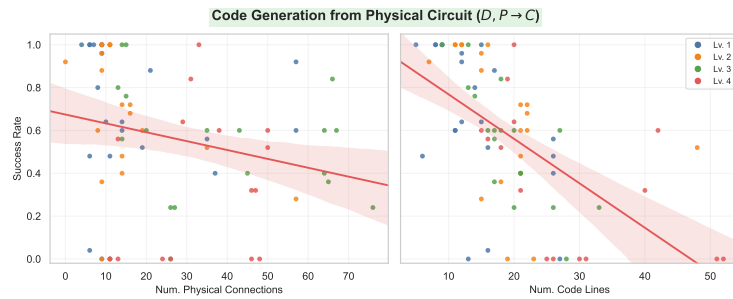
(a) Logical Circuit Generation ( $D, C \rightarrow L$ )



(b) Physical Circuit Generation ( $D, C \rightarrow P$ )

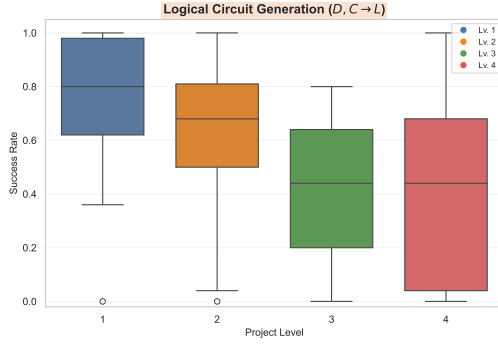


(c) Code Generation from Logical Circuit ( $D, L \rightarrow C$ )

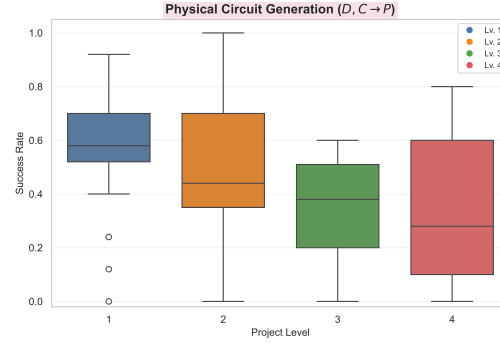


(d) Code Generation from Physical Circuit ( $D, P \rightarrow C$ )

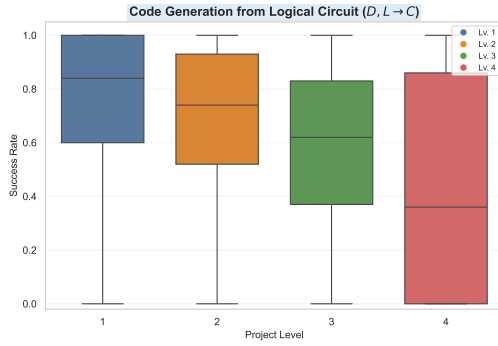
Figure 7: Test procedure success rates across different tasks and complexity metrics, arranged vertically. Subfigure (a) shows Logical Circuit Generation, (b) Physical Circuit Generation, (c) Code Generation from Logical Circuit, and (d) Code Generation from Physical Circuit. Each subplot details performance against the number of connections and lines of code relevant to the task.



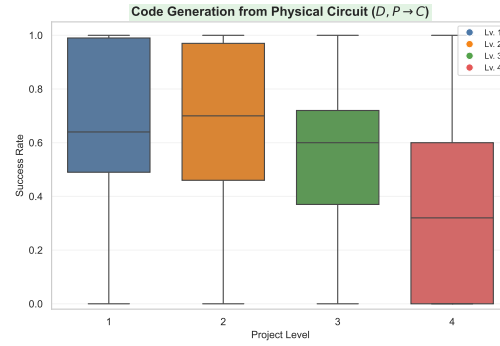
(a) Logical Circuit Gen. ( $D, C \rightarrow L$ )



(b) Physical Circuit Gen. ( $D, C \rightarrow P$ )



(c) Code Gen. from Logical Circuit ( $D, L \rightarrow C$ )



(d) Code Gen. from Physical Circuit ( $D, P \rightarrow C$ )

Figure 8: Project-level success-rate distributions across the four PCEVAL tasks by benchmark level. Subfigure (a) shows Logical Circuit Generation, (b) Physical Circuit Generation, (c) Code Generation from Logical Circuit, and (d) Code Generation from Physical Circuit. Each box plot summarizes the distribution of project-level success rates within a level after averaging over the same five selected models.

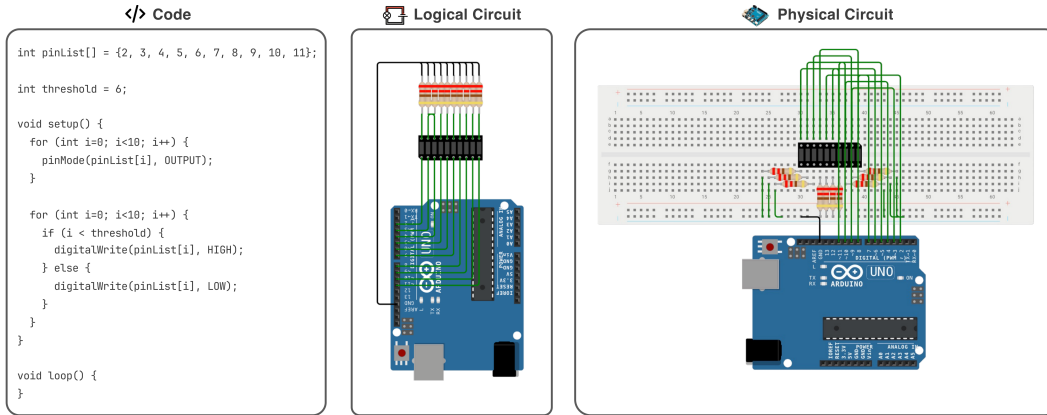


Figure 9: Reference code, logical circuit, and physical circuit for the “Bar LED Basic” project.

## L Reference Code and Circuit Examples

This section provides reference examples for two PCEVAL projects. For each project, we show the reference code together with the corresponding logical and physical circuit representations used in the benchmark. These figures are intended to make the task artifacts concrete and to illustrate the range of structures covered by the dataset.

### L.1 Bar LED Basic (Level 1)

Figure 9 shows the reference implementation for the “Bar LED Basic” project, including the code, logical circuit, and physical circuit. Although this project belongs to Level 1, it already involves a multi-pin output device with repeated LED-resistor wiring patterns.

### L.2 4 Digit 7 Segment Display (Level 3)

Figure 10 shows the reference implementation for the “4 Digit 7 Segment Display” project. This example includes a denser wiring structure and a multi-digit display configuration, providing a representative reference artifact from a more advanced benchmark level.

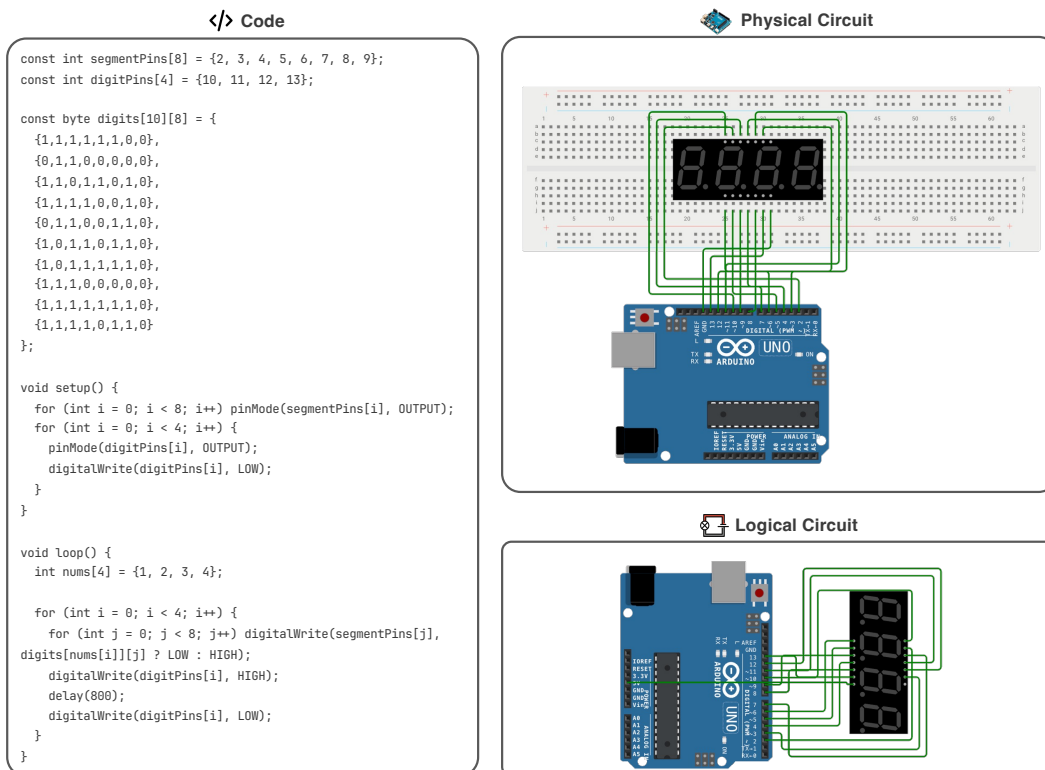


Figure 10: Reference code, logical circuit, and physical circuit for the “4 Digit 7 Segment Display” project.

## 766 M Prompt Details

767 This appendix provides the specific prompts used for each of the four tasks in the PCEVAL benchmark.  
768 Each prompt was carefully designed to clearly articulate the task objectives to the LLMs and to  
769 specify the required output format. For tasks involving physical circuit generation or code generation  
770 from physical circuits, prompts included detailed explanations of breadboard mechanics and electrical  
771 connectivity rules to ensure LLMs had the necessary contextual information. The prompt listings  
772 below show the Arduino versions used as the base templates. For ESP32, we used closely matched  
773 platform specific variants instead of identical prompt strings. These variants changed the initial  
774 task request, replaced Arduino references in the project description and component names with  
775 ESP32 specific ones, and added implementation guidance for the Arduino core such as including  
776 `Arduino.h` in code generation settings.

### Code Generation Task (Logical Hardware)

```
777 # Arduino Code Generation Task (Logical Hardware)
779 ## Task
780 Please generate Arduino code (main.ino) that will work with this logical
781 hardware configuration and produce the expected behavior.
782 The code should fulfill all requirements and pass all test steps in the
783 scenario.
784 Note that this is a LOGICAL circuit diagram, meaning it shows the direct
785 connections between components at a conceptual level, NOT physical layout with
786 breadboard positions.
787
788 ### Output Format
789 Please provide only the code without explanations or markdown formatting.
790
791 ## Project Description
792 {Project Description Text}
793
794 ## Hardware Configuration (Logical Circuit)
795 Below is the JSON diagram describing the logical hardware circuit:
796 ```json
797 {Standardized Logical Diagram JSON}
798 ```
```

### Code Generation Task (Physical Hardware)

```
804 # Arduino Code Generation Task (Physical Hardware)
806 ## Task
807 Please generate Arduino code (main.ino) that will work with this physical
808 hardware configuration and produce the expected behavior.
809 The code should fulfill all requirements and pass all test steps in the
810 scenario.
811 Note that this is a PHYSICAL circuit diagram, meaning it shows the actual
812 breadboard layout with specific component placements and wire connections
813 using breadboard positions.
814
815 ### Output Format
816 Please provide only the code without explanations or markdown formatting.
817
818 ## Project Description
819 {Project Description Text}
820
821
```

```

824 ## Hardware Configuration (Physical Breadboard Layout)
825
826 ### Pin and Position Naming Conventions
827 - Component pins: "arduino1.pin13", "led1.anode", "resistor1.pin1", etc.
828 - Breadboard positions (main area): "breadboard.COLUMN_ROW"
829     - Column: A number from 1-60
830     - Row: A letter from a-j
831     - Rows a-e are on the top half, rows f-j are on the bottom half
832     - Example: "breadboard.10a" refers to column 10, row a (top half)
833 - Breadboard power rail positions: "breadboard.RAIL_TYPE.COLUMN_NUMBER"
834     - RAIL_TYPE: tp (top positive), tn (top negative), bp (bottom positive),
835     bn (bottom negative)
836     - COLUMN_NUMBER: A number, typically referring to segments or positions
837     along the rail (e.g., 1 up to 50)
838     - Example: "breadboard.tp.1" refers to position 1 on the top positive (+)
839     rail
840     - Example: "breadboard.bn.12" refers to position 12 on the bottom negative
841     (-) rail
842
843 ### Breadboard Explanation
844 A breadboard is divided into two halves separated by a central gap:
845 - Top half: 5 rows of holes, commonly labeled a, b, c, d, e
846   (from top to bottom for each column segment).
847 - Bottom half: 5 rows of holes, commonly labeled f, g, h, i, j
848   (from top to bottom for each column segment).
849 - The columns are typically numbered 1-60 from left to right.
850   So, for each column number, there's a set of 'a-e' holes and
851   a set of 'f-j' holes.
852
853 #### Electrical Connections:
854 - Terminal Strips (the main area with rows a-e and rows f-j):
855     - Within each half (top or bottom), and for each column number, the 5
856     holes in that column segment are electrically connected vertically.
857     - Example: 'breadboard.1a', 'breadboard.1b', 'breadboard.1c', '
858     breadboard.1d', and 'breadboard.1e' are all connected.
859     - Example: 'breadboard.1f', 'breadboard.1g', 'breadboard.1h', '
860     breadboard.1i', and 'breadboard.1j' are all connected.
861     - Holes in different columns are NOT connected. (e.g., 'breadboard.1a' is
862     not connected to 'breadboard.2a')
863     - The top half (holes a-e) and the bottom half (holes f-j) are
864     electrically separated by the central gap. (e.g., 'breadboard.1e' is not
865     connected to 'breadboard.1f')
866 - Power Rails (tp, tn, bp, bn):
867     - All holes in a single power rail (e.g., the 'tp' rail) are electrically
868     connected horizontally along the length of that entire rail.
869     - Example: All positions along "breadboard.tp" (e.g., 'breadboard.tp
870     .1', 'breadboard.tp.2', ..., 'breadboard.tp.50') are connected.
871     - Typically, the positive (+) and negative (-) rails are electrically
872     separate from each other.
873     - The top set of power rails (e.g., 'tp' and 'tn') are electrically
874     separate from the bottom set of power rails (e.g., 'bp' and 'bn'), unless
875     explicitly connected by external wires.
876
877 Below is the JSON diagram describing the physical hardware circuit
878 with breadboard layout:
879 ```json
880 {Standardized Physical Diagram JSON}
881 ```
882

```

## Circuit Generation Task (Logical)

```

884
885 # Arduino Hardware Design Task
886
887

```

```

888 ## Task
889 Based on the Arduino code and project description, create a logical circuit
890 diagram that will work correctly with this code.
891
892 Focus on the direct connections between components at a conceptual level, NOT
893 physical layout with breadboard positions.
894
895 ## Output Format
896 Provide your answer in JSON format with two main sections:
897 1. "components": A list of all hardware components needed
898 2. "connections": A list of all connections between components
899
900 ### Components Format
901 Each component should have:
902 - "id": A unique, meaningful identifier with a number appended (e.g., "
903 arduino1", "led2", "resistor1"), that is consistently used in the connections
904 section
905 - "type": Component type ("Arduino Uno", "LED", "Resistor", "Button", etc.)
906 - "properties": Optional object with properties specific to the component type
907
908 ### Connections Format
909 Each connection is simply an array with exactly two elements, each being a
910 connection point formatted as "componentId.pinName".
911 For example: ["arduino1.pin13", "led2.anode"]
912
913 ### Pin Naming Conventions
914 - Arduino pins: "pin2", "pin13", "a0", "a1", "5v", "3.3v", "gnd1", "gnd2", "
915 gnd3", etc.
916 - LED pins: "anode", "cathode"
917 - Resistor pins: "pin1", "pin2"
918 - Button pins: "pin1.l", "pin1.r", "pin2.l", "pin2.r"
919
920 ### Example
921 ```json
922 {
923   "components": [
924     {"id": "arduino1", "type": "Arduino Uno"},
925     {"id": "ntc_sensor1", "type": "NTC temperature sensor",
926      "properties": {"temperature": "24"}}
927   ],
928   "connections": [
929     ["arduino1.pin13", "resistor2.pin1"],
930     ["ntc_sensor1.VCC", "arduino1.5v"],
931     ["ntc_sensor1.GND", "arduino1.gnd2"],
932     ["ntc_sensor1.OUT", "arduino1.A0"]
933   ]
934 }
935 ```
936
937 Based on the code, provide a complete JSON with all components and direct
938 logical connections to create a functional circuit.
939
940 ## Project Description
941 {Project Description Text}
942
943 ## Arduino Code
944 ```cpp
945 {Sketch Code Text}
946 ```
947

```

## Circuit Generation Task (Physical)

```

949 # Arduino Hardware Design Task
950

```

```

952
953 ## Task
954 Based on the Arduino code and project description, create a physical
955 breadboard layout that will work correctly with this code.
956
957 Include a breadboard and show how components would be physically placed and
958 connected on it.
959
960 ## Output Format
961 Provide your answer in JSON format with two main sections:
962 1. "components": A list of all hardware components needed
963 2. "connections": A list of all connections between components
964
965 ### Components Format
966 Each component should have:
967 - "id": A unique, meaningful identifier with a number appended (e.g., "
968 arduino1", "led2", "resistor1") that is consistently used in the connections
969 section
970 - "type": Component type ("Arduino Uno", "LED", "Resistor", "Button", "
971 Breadboard", etc.)
972 - "properties": Optional object with properties specific to the component type
973
974 ### Connections Format
975 Each connection is simply an array with exactly two elements, which can be
976 component pins or breadboard positions.
977 For example: ["arduino1.pin13", "breadboard1.10a"]
978
979 ### Pin and Position Naming Conventions
980 - Component pins: "arduino1.pin13", "led1.anode", "resistor1.pin1", etc.
981 - Arduino pins: "pin2", "pin13", "a0", "a1", "5v", "3.3v", "gnd1", "gnd2", "
982 gnd3", etc.
983 - LED pins: "anode", "cathode"
984 - Resistor pins: "pin1", "pin2"
985 - Button pins: "pin1.l", "pin1.r", "pin2.l", "pin2.r"
986 - Breadboard positions (main area): "breadboard.COLUMN_ROW"
987   - Column: A number from 1-60
988   - Row: A letter from a-j
989   - Rows a-e are on the top half, rows f-j are on the bottom half
990   - Example: "breadboard.10a" refers to column 10, row a (top half)
991 - Breadboard power rail positions: "breadboard.RAIL_TYPE.COLUMN_NUMBER"
992   - RAIL_TYPE: tp (top positive), tn (top negative), bp (bottom positive),
993   bn (bottom negative)
994   - COLUMN_NUMBER: A number, typically referring to segments or positions
995   along the rail (e.g., 1 up to 50)
996   - Example: "breadboard.tp.1" refers to position 1 on the top positive (+)
997   rail
998   - Example: "breadboard.bn.12" refers to position 12 on the bottom negative
999   (-) rail
1000
1001 ### Breadboard Explanation
1002 A breadboard is divided into two halves separated by a central gap:
1003 - Top half: 5 rows of holes, commonly labeled a, b, c, d, e
1004   (from top to bottom for each column segment).
1005 - Bottom half: 5 rows of holes, commonly labeled f, g, h, i, j
1006   (from top to bottom for each column segment).
1007 - The columns are typically numbered 1-60 from left to right.
1008   So, for each column number, there's a set of 'a-e' holes and a set of 'f-j'
1009   holes.
1010
1011 #### Electrical Connections:
1012 - Terminal Strips (the main area with rows a-e and rows f-j):
1013   - Within each half (top or bottom), and for each column number, the 5
1014   holes in that column segment are electrically connected vertically.
1015     - Example: 'breadboard.1a', 'breadboard.1b', 'breadboard.1c', '
1016     breadboard.1d', and 'breadboard.1e' are all connected.
1017     - Example: 'breadboard.1f', 'breadboard.1g', 'breadboard.1h', '
1018     breadboard.1i', and 'breadboard.1j' are all connected.
1019   - Holes in different columns are NOT connected.

```

```

1020         (e.g., 'breadboard.1a' is not connected to 'breadboard.2a')
1021     - The top half (holes a-e) and the bottom half (holes f-j) are
1022       electrically separated by the central gap.
1023       (e.g., 'breadboard.1e' is not connected to 'breadboard.1f')
1024 - Power Rails (tp, tn, bp, bn):
1025     - All holes in a single power rail (e.g., the 'tp' rail) are electrically
1026       connected horizontally along the length of that entire rail.
1027       - Example: All positions along "breadboard.tp" (e.g., 'breadboard.tp
1028         .1', 'breadboard.tp.2', ..., 'breadboard.tp.50') are connected.
1029     - Typically, the positive (+) and negative (-) rails are electrically
1030       separate from each other.
1031     - The top set of power rails (e.g., 'tp' and 'tn') are electrically
1032       separate from the bottom set of power rails (e.g., 'bp' and 'bn'), unless
1033       explicitly connected by external wires.
1034
1035 ### Example
1036 ```json
1037 {
1038     "components": [
1039         {"id": "arduino1", "type": "Arduino Uno"},
1040         {"id": "breadboard1", "type": "Breadboard"},
1041         {"id": "ntc_sensor1", "type": "NTC temperature sensor",
1042          "properties": {"temperature": "24"}}
1043     ],
1044     "connections": [
1045         ["breadboard1.29g", "arduino1.gnd3"],
1046         ["ntc_sensor1.VCC", "breadboard1.28f"],
1047         ["ntc_sensor1.GND", "breadboard1.29f"],
1048         ["ntc_sensor1.OUT", "breadboard1.27f"],
1049         ["breadboard1.28g", "arduino1.5v"],
1050         ["breadboard1.27g", "arduino1.A0"]
1051     ]
1052 }
1053 ```
1054
1055 ### Important Constraints
1056 - Each Arduino pin can only have ONE connection assigned to it.
1057 - Each breadboard hole can only have ONE connection assigned to it.
1058 - Direct pin-to-pin connections between components without using a breadboard
1059   are not permitted for circuit construction involving a breadboard. All such
1060   connections must be routed via breadboard positions.
1061
1062 Pay attention to the internal connections of the breadboard.
1063 Based on the code, provide a complete JSON with all components and connections
1064   to create a functional physical circuit with breadboard layout.
1065
1066 ## Project Description
1067 {Project Description Text}
1068
1069 ## Arduino Code
1070 ```cpp
1071 {Sketch Code Text}
1072 ```
1073

```

## 1075 M.1 Prompt Variants for Exploratory Mitigation

1076 In addition to the four base prompts above, the exploratory mitigation experiments in Section 4.3  
 1077 used two prompt variants in the Arduino Uno setting. Self-improvement reused the base task prompt  
 1078 on the first turn and, from the second turn onward, supplied the model's previous output together  
 1079 with structured compilation and test feedback. Chain-of-thought (CoT) prompting was applied only  
 1080 to tasks involving physical circuits and decomposed them into two sequential steps: first derive an  
 1081 intermediate logical circuit, then generate the final code or breadboard layout conditioned on that  
 1082 intermediate representation.

## Self-Improvement Follow-Up Prompt Template

```
1083 # {Artifact Type} Self-Improvement Task (Iteration {Iteration Number})
1084
1085 ## Original Task
1086 {Original Prompt}
1087
1088 ## Your Previous {Artifact Type}
1089 {Previous Generation}
1090
1091 ## Previous Results
1092 - Compilation: {Compilation Result}
1093 - Testing: {Test Result}
1094 {Error Details}
1095
1096 ## Self-Improvement Instructions
1097 Please analyze your previous {Artifact Type} and the test results,
1098 then create an improved version.
1099 Focus on fixing the issues mentioned in the test results and error
1100 messages.
1101 Make sure your {Artifact Type} works correctly with the hardware
1102 specification and fulfills all requirements.
1103
1104 Ensure your improved {Artifact Type} follows all formatting
1105 requirements from the original prompt.
1106 {Output Format Reminder}
```

## CoT Prompt for Code Generation from Physical Hardware

```
1111 # Arduino Code Generation Task (Physical Hardware)
1112
1113 ## Step 1: Derive a Logical Circuit
1114 The goal is to generate Arduino code that matches the given physical
1115 circuit, step by step.
1116 This process first defines a logical circuit diagram to outline
1117 component connections, and then generates Arduino code that interfaces
1118 correctly with that circuit.
1119
1120 Based on the physical circuit diagram, create a logical circuit
1121 diagram that specifies the logical connections between components.
1122 Focus on the direct conceptual connections between components, NOT
1123 physical layout with breadboard positions.
1124
1125 ### Output Format for Step 1
1126 Provide your answer in JSON format with two main sections:
1127 1. "components": A list of all hardware components needed
1128 2. "connections": A list of all connections between components
1129
1130 Use the same component naming conventions, example JSON structure,
1131 and breadboard interpretation rules as in the base physical-hardware
1132 code-generation prompt above.
1133
1134 ## Hardware Configuration (Physical Breadboard Layout)
1135 ```json
1136 {Standardized Physical Diagram JSON}
1137 ```
1138
1139 ## Step 2: Generate Code
1140 Now, based on the project description and the logical hardware
1141 configuration you generated, please generate Arduino code (main.ino)
1142 that will work with the circuit and produce the expected behavior.
```

```

1146 ## Project Description
1147 {Project Description Text}
1148
1149 ### Output Format for Step 2
1150 Please provide only the code without explanations or markdown
1151 formatting.
1152

```

## CoT Prompt for Physical Circuit Generation

```

1154 # Arduino Hardware Design Task
1155
1156 ## Step 1: Derive a Logical Circuit
1157 The goal is to create a physical breadboard layout that will work
1158 correctly with this code, step by step.
1159 This process first defines a logical circuit diagram to outline
1160 component connections, and then creates a physical breadboard layout
1161 that reflects these connections in hardware.
1162
1163 Create a LOGICAL circuit diagram that specifies connections between
1164 components.
1165 Focus on the direct connections between components at a conceptual
1166 level, NOT physical layout with breadboard positions.
1167
1168 ### Output Format for Step 1
1169 Provide your answer in JSON format with two main sections:
1170 1. "components": A list of all hardware components needed
1171 2. "connections": A list of all connections between components
1172
1173 Use the same component naming conventions, logical-circuit JSON
1174 format, and pin naming rules as in the base logical circuit-generation
1175 prompt above.
1176
1177 ## Project Description
1178 {Project Description Text}
1179
1180 ## Arduino Code
1181 ```cpp
1182 {Sketch Code Text}
1183 ```
1184
1185 ## Step 2: Generate a Physical Breadboard Layout
1186 Now, create a PHYSICAL breadboard layout that corresponds to the
1187 logical circuit you generated.
1188 Ensure that the pin assignments on the Arduino board remain exactly as
1189 specified in the logical circuit configuration, and that the generated
1190 breadboard layout correctly interfaces with the given code.
1191
1192 ### Output Format for Step 2
1193 Provide your answer in JSON format with two main sections:
1194 1. "components": A list of all hardware components needed
1195 2. "connections": A list of all connections between components
1196
1197 Use the same breadboard naming conventions, electrical-connection
1198 rules, and single-connection constraints as in the base physical
1199 circuit-generation prompt above.
1200
1201
1202

```

## 1204 N Use of AI Assistants in Writing

1205 In accordance with NeurIPS policy, we disclose the use of Large Language Models in preparing this  
1206 manuscript. We used LLMs to assist with grammar checking, sentence restructuring, and language

1207 polishing during the revision process. The LLM was primarily used to improve clarity and conciseness  
1208 of technical descriptions, particularly in the methodology and results sections. All scientific content,  
1209 experimental design, analysis, and interpretations are the original work of the authors. The LLM  
1210 served solely as a writing aid and did not contribute to the research conception, or core intellectual  
1211 contributions of this work.

## 1212 NeurIPS Paper Checklist

### 1213 1. Claims

1214 Question: Do the main claims made in the abstract and introduction accurately reflect the  
1215 paper’s contributions and scope?

1216 Answer: [Yes]

1217 Justification: The abstract and introduction state the benchmark scope, four evaluation  
1218 dimensions, automatic evaluation setting, evaluated models, and main empirical findings.  
1219 The claims are supported by the task definitions, dataset description, and experimental  
1220 results in Sections 3–4.

1221 Guidelines:

- 1222 • The answer [N/A] means that the abstract and introduction do not include the claims  
1223 made in the paper.
- 1224 • The abstract and/or introduction should clearly state the claims made, including the  
1225 contributions made in the paper and important assumptions and limitations. A [No] or  
1226 [N/A] answer to this question will not be perceived well by the reviewers.
- 1227 • The claims made should match theoretical and experimental results, and reflect how  
1228 much the results can be expected to generalize to other settings.
- 1229 • It is fine to include aspirational goals as motivation as long as it is clear that these goals  
1230 are not attained by the paper.

### 1231 2. Limitations

1232 Question: Does the paper discuss the limitations of the work performed by the authors?

1233 Answer: [Yes]

1234 Justification: The paper includes a dedicated Limitations section discussing the educational  
1235 scope, introductory-project focus, and platform coverage limitations across Arduino Uno  
1236 and ESP32.

1237 Guidelines:

- 1238 • The answer [N/A] means that the paper has no limitation while the answer [No] means  
1239 that the paper has limitations, but those are not discussed in the paper.
- 1240 • The authors are encouraged to create a separate “Limitations” section in their paper.
- 1241 • The paper should point out any strong assumptions and how robust the results are to  
1242 violations of these assumptions (e.g., independence assumptions, noiseless settings,  
1243 model well-specification, asymptotic approximations only holding locally). The authors  
1244 should reflect on how these assumptions might be violated in practice and what the  
1245 implications would be.
- 1246 • The authors should reflect on the scope of the claims made, e.g., if the approach was  
1247 only tested on a few datasets or with a few runs. In general, empirical results often  
1248 depend on implicit assumptions, which should be articulated.
- 1249 • The authors should reflect on the factors that influence the performance of the approach.  
1250 For example, a facial recognition algorithm may perform poorly when image resolution  
1251 is low or images are taken in low lighting. Or a speech-to-text system might not be  
1252 used reliably to provide closed captions for online lectures because it fails to handle  
1253 technical jargon.
- 1254 • The authors should discuss the computational efficiency of the proposed algorithms  
1255 and how they scale with dataset size.
- 1256 • If applicable, the authors should discuss possible limitations of their approach to  
1257 address problems of privacy and fairness.
- 1258 • While the authors might fear that complete honesty about limitations might be used by  
1259 reviewers as grounds for rejection, a worse outcome might be that reviewers discover  
1260 limitations that aren’t acknowledged in the paper. The authors should use their best  
1261 judgment and recognize that individual actions in favor of transparency play an impor-  
1262 tant role in developing norms that preserve the integrity of the community. Reviewers  
1263 will be specifically instructed to not penalize honesty concerning limitations.

### 1264 3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not present theoretical results, theorems, or formal proofs; it is an empirical benchmark and evaluation study.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

#### 4. Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper specifies the benchmark structure, task definitions, evaluation protocol, simulator-based validation, model set, prompt templates, and platform-specific breakdowns needed to reproduce the main experimental methodology

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  - (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

1319 some way (e.g., to registered users), but it should be possible for other researchers  
1320 to have some path to reproducing or verifying the results.

## 1321 5. Open access to data and code

1322 Question: Does the paper provide open access to the data and code, with sufficient instruc-  
1323 tions to faithfully reproduce the main experimental results, as described in supplemental  
1324 material?

1325 Answer: [Yes]

1326 Justification: We provide anonymized benchmark data, reference artifacts, evaluation scripts,  
1327 prompt templates, and reproduction instructions.

1328 Guidelines:

- 1329 • The answer [N/A] means that paper does not include experiments requiring code.
- 1330 • Please see the NeurIPS code and data submission guidelines ([https://neurips.](https://neurips.cc/public/guides/CodeSubmissionPolicy)  
1331 [cc/public/guides/CodeSubmissionPolicy](https://neurips.cc/public/guides/CodeSubmissionPolicy)) for more details.
- 1332 • While we encourage the release of code and data, we understand that this might not  
1333 be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not  
1334 including code, unless this is central to the contribution (e.g., for a new open-source  
1335 benchmark).
- 1336 • The instructions should contain the exact command and environment needed to run to  
1337 reproduce the results. See the NeurIPS code and data submission guidelines ([https:](https://neurips.cc/public/guides/CodeSubmissionPolicy)  
1338 [//neurips.cc/public/guides/CodeSubmissionPolicy](https://neurips.cc/public/guides/CodeSubmissionPolicy)) for more de-  
1339 tails.
- 1340 • The authors should provide instructions on data access and preparation, including how  
1341 to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- 1342 • The authors should provide scripts to reproduce all experimental results for the new  
1343 proposed method and baselines. If only a subset of experiments are reproducible, they  
1344 should state which ones are omitted from the script and why.
- 1345 • At submission time, to preserve anonymity, the authors should release anonymized  
1346 versions (if applicable).
- 1347 • Providing as much information as possible in supplemental material (appended to the  
1348 paper) is recommended, but including URLs to data and code is permitted.

## 1349 6. Experimental setting/details

1350 Question: Does the paper specify all the training and test details (e.g., data splits, hyperpa-  
1351 rameters, how they were chosen, type of optimizer) necessary to understand the results?

1352 Answer: [Yes]

1353 Justification: The paper reports the evaluated model set, benchmark composition, task defi-  
1354 nitions, prompts, evaluation metrics, simulator choice, number of trials, and hyperparameter  
1355 details including token limit and temperature settings.

1356 Guidelines:

- 1357 • The answer [N/A] means that the paper does not include experiments.
- 1358 • The experimental setting should be presented in the core of the paper to a level of detail  
1359 that is necessary to appreciate the results and make sense of them.
- 1360 • The full details can be provided either with the code, in appendix, or as supplemental  
1361 material.

## 1362 7. Experiment statistical significance

1363 Question: Does the paper report error bars suitably and correctly defined or other appropriate  
1364 information about the statistical significance of the experiments?

1365 Answer: [Yes]

1366 Justification: We report mean success rates with confidence intervals/standard errors com-  
1367 puted over independent trials, and state how uncertainty estimates are calculated.

1368 Guidelines:

- 1369 • The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

## 8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: The paper reports the local CPU, GPU, GPU memory, CUDA version, RAM, and OS used for API querying and analysis.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

## 9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines>?

Answer: [Yes]

Justification: The research was conducted on educational benchmark tasks and anonymized educator feedback, and the paper describes voluntary participation, anonymized reporting, and consent procedures for the human-subject components.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

## 10. Broader impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: Section 5 discusses positive educational impacts, such as reducing instructor workload and supporting less-experienced teachers, as well as risks such as students bypassing reasoning through ready-made LLM-generated answers.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

## 11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release a high-risk pretrained model, image generator, scraped dataset, or similarly misuse-prone asset; the introduced benchmark consists of educational physical-computing tasks and evaluation artifacts.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

## 12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cite all existing assets and provide their versions, licenses, and terms-of-use considerations in the supplemental material.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, [paperswithcode.com/datasets](https://paperswithcode.com/datasets) has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

**13. New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The released PCEVAL assets include documentation covering task structure, file formats, reference artifacts, evaluation scripts, limitations, and license information.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

**14. Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [Yes]

Justification: Appendices A and B provide participant details, interview/survey procedures, question prompts/instructions, consent information, anonymization, and compensation details.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

**15. Institutional review board (IRB) approvals or equivalent for research with human subjects**

1525 Question: Does the paper describe potential risks incurred by study participants, whether  
 1526 such risks were disclosed to the subjects, and whether Institutional Review Board (IRB)  
 1527 approvals (or an equivalent approval/review based on the requirements of your country or  
 1528 institution) were obtained?

1529 Answer: [\[Yes\]](#)

1530 Justification: We conducted interviews with IRB approval.

1531 Guidelines:

- 1532 • The answer [\[N/A\]](#) means that the paper does not involve crowdsourcing nor research  
 1533 with human subjects.
- 1534 • Depending on the country in which research is conducted, IRB approval (or equivalent)  
 1535 may be required for any human subjects research. If you obtained IRB approval, you  
 1536 should clearly state this in the paper.
- 1537 • We recognize that the procedures for this may vary significantly between institutions  
 1538 and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the  
 1539 guidelines for their institution.
- 1540 • For initial submissions, do not include any information that would break anonymity (if  
 1541 applicable), such as the institution conducting the review.

1542 **16. Declaration of LLM usage**

1543 Question: Does the paper describe the usage of LLMs if it is an important, original, or  
 1544 non-standard component of the core methods in this research? Note that if the LLM is used  
 1545 only for writing, editing, or formatting purposes and does *not* impact the core methodology,  
 1546 scientific rigor, or originality of the research, declaration is not required.

1547 Answer: [\[Yes\]](#)

1548 Justification: The paper describes LLMs as the evaluated systems in the core benchmark  
 1549 methodology and includes prompt details; Appendix N also discloses the use of LLMs for  
 1550 grammar checking, sentence restructuring, and language polishing.

1551 Guidelines:

- 1552 • The answer [\[N/A\]](#) means that the core method development in this research does not  
 1553 involve LLMs as any important, original, or non-standard components.
- 1554 • Please refer to our LLM policy in the NeurIPS handbook for what should or should not  
 1555 be described.