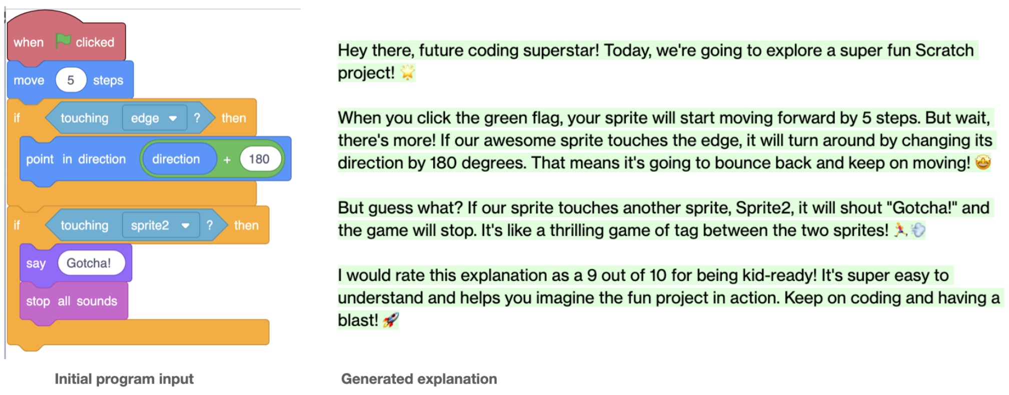
Scratch Copilot Evaluation: Assessing AI-Assisted Creative Coding for Families

Stefania Druga
Information School, University of Washington
Seattle, Washington, United States
st3f@uw.edu

Nancy Otero
Kitco
San Francisco, California, United States
nancy.otero.o@gmail.com



```
when 🏁 clicked
move (5) steps
if <touching (edge ▾)?> then
    point in direction ((direction) + (180))
if <touching (sprite2 ▾)?> then
    say (Gotcha!)
    stop all sounds
```

Hey there, future coding superstar! Today, we're going to explore a super fun Scratch project! 🌟

When you click the green flag, your sprite will start moving forward by 5 steps. But wait, there's more! If our awesome sprite touches the edge, it will turn around by changing its direction by 180 degrees. That means it's going to bounce back and keep on moving! 🤩

But guess what? If our sprite touches another sprite, Sprite2, it will shout "Gotcha!" and the game will stop. It's like a thrilling game of tag between the two sprites! 🏃💨

I would rate this explanation as a 9 out of 10 for being kid-ready! It's super easy to understand and helps you imagine the fun project in action. Keep on coding and having a blast! 🚀

Figure 1: Examples of code explanation provided by the GPT4 model.

ABSTRACT

How can AI enhance creative coding experiences for families? This study explores the potential of large language models (LLMs) in helping families with creative coding using Scratch. Based on our previous user study involving a prototype AI assistant, we devised three evaluation scenarios to determine if LLMs could help families comprehend game code, debug programs, and generate new ideas for future projects. We utilized 22 Scratch projects for each scenario and generated responses from LLMs with and without practice tasks, resulting in 120 creative coding support scenario datasets. In addition, the authors independently evaluated their precision, pedagogical value, and age-appropriate language. Our findings show that LLMs achieved an overall success rate of more than 80% on the different tasks and evaluation criteria. This research offers valuable information on using LLMs for creative family coding and presents design guidelines for future AI-supported coding applications. Our evaluation framework, together with our labeled evaluation data, is publicly available [1].

KEYWORDS

AI Assistant, Children, Families, Creative Coding

[1]https://github.com/stefania11/ScratchCopilot-Evaluation

ACM Reference Format:
Stefania Druga and Nancy Otero. 2023. Scratch Copilot Evaluation: Assessing AI-Assisted Creative Coding for Families. In *Proceedings of (Arxiv).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Computer Science (CS) education faces a critical bottleneck. The need for more trained teachers and curriculum designers stifles progress in this field [46]. Supporting project-based learning for youth and families, particularly creative coding, could be a potential solution. Prior work shows that engaging youth and families in creative coding has been advocated to promote more inclusive and accessible learning experiences [36]. The limited availability of support and expertise in computer science education also calls for innovative technological solutions, similar to Github Copilot, which show considerable promise [15].

Parents often need more technical knowledge for effective coding instruction despite their expertise in engaging their children. In this regard, models have been proposed to complement joint creative coding between children and parents by providing timely suggestions, questions, ideas, and tips. It is worth noting, however, that the introduction of external support can both positively and negatively impact youth motivation and learning.

Studies have shown that creative coding can significantly enhance student motivation and bolster confidence in their knowledge and technical abilities compared to traditional CS programs (Rittenhouse, C. S. Scholarship, Research, and Creative Work at Bryn Mawr College). Furthermore, creative coding might allow students to develop a more immersive and experiential relationship with digital processes, providing them with hands-on experiences and theoretical frameworks [10].

Young people have lauded experiences that enable them to express their ideas, foster relationships, assist others, and discover new perspectives about themselves. This emphasis extends beyond the common focus of coding initiatives on computational thinking and problem-solving skills to support social, leadership, and identity development [37].

Large Language Models (LLMs), such as OpenAI's Codex and GPT-3, have demonstrated potential in aiding tasks related to explaining, ideating, and debugging creative coding projects. However, their current performance may fall short of fulfilling the unique needs of middle school families [34]. While these models have achieved some success in generating novel and meaningful content, the necessity for human oversight to ensure the quality and accuracy of the generated content remains [22, 38].

Despite the potential of such tools, several potential disadvantages exist when utilized for family creative coding. For instance, young learners may need to be more responsive to these tools, impairing their ability to create similar code independently. Other challenges include formulating their intentions to generate the desired code and understanding the code produced by AI for subsequent modification if needed [43]. Moreover, a recent study assessing a new creative coding integrated development environment (IDE) revealed students' concerns about the trade-off between improving their abilities and facilitating the development of their skills through IDE syntax templates and autocomplete coding features [29].

Involving parents as learning partners in the creative process is paramount. Past research has shown that parents can act as mentors and co-tinkerers when families engage in game programming [7] or AI literacies tinkering [8, 24].

Given these findings, our research aims to answer the following question:

- RQ: How well do large-language models support explaining, ideating, and debugging Scratch projects for middle school families?

This paper explores the potential of LLMs in aiding families interested in learning creative coding together. We focus on the applicability of LLMs for generating Scratch program explanations, debugging, and ideation support. Our previous user study identified these three areas as primary needs for family AI-assisted creative coding [9].

Our findings reveal that LLMs achieved an overall success rate of over 80% across the various tasks and evaluation criteria. This study further contributes a public dataset of Scratch programs, complemented by the code explanations, debugging, and ideation support tasks we employed for the LLM evaluation.

Based on these findings, we discuss potential scenarios for designing inclusive LLM support for family creative coding. Moreover, we propose a series of design guidelines that could inform the development of future AI-supported coding applications. This exploration thereby provides insights into the effectiveness and potential of LLMs as supportive tools for families engaged in creative coding, offering a promising avenue for inclusive and accessible computer science education. Our evaluation framework, together with our labeled evaluation data, is publicly available here: github.com/stefania11/ScratchCopilot-Evaluation.

2 RELATED WORK

2.1 LLMs in Computing Education

Large language models (LLMs) have demonstrated potential in numerous fields, especially education and programming. In addition, the influence of LLMs on novice learners, particularly in introductory programming environments, has garnered scholarly interest.

Kazemitabaar et al. studied the effects of OpenAI Codex on middle school learners within a self-paced learning setting. Their findings suggest that Codex significantly enhanced code-authoring performance without negatively impacting manual code-modification tasks [19]. However, it was observed that performance differences in post-tests conducted a week later were not statistically significant, underscoring the necessity of further research.

Leinonen et al. explored the use of LLMs in generating code explanations, comparing GPT-3-generated explanations with those created by students in an introductory programming course. Their findings highlighted that LLM-generated explanations were perceived as significantly easier to comprehend and more accurate than those produced by the students [22].

Turning to debugging tasks, Chen et al. introduced the concept of Self-Debugging, which trains LLMs to debug their predicted programs using few-shot demonstrations. Their study established that Self-Debugging surpassed performance standards on code generation benchmarks, improving the baseline accuracy by up to 12% and demonstrating notable sample efficiency [5]. Similarly, Madaan et al. examined LLMs' capacity to suggest performance-improving code edits, establishing that tools like CODEGEN and CODEX could generate such edits for C++ and Python programs [25].

The potential of LLMs extends to computer science education, where AI code generators can offer substantial support to learners and educators alike. For example, they can automatically rectify semantic bugs and syntax errors, allowing learners to concentrate more on theoretical and problem-solving aspects of computational thinking. Additionally, these tools can assist educators in developing curriculum by creating programmatic exercises and explaining solutions [38].

Guo's study offers further insight, introducing an interactive web-based tool, the online Python Tutor, which aids students in understanding Python programming through visualization of code execution [14]. This supports novice learners in comprehending complex computer programming concepts and is an effective debugging aid.

In the realm of creative coding, it has been observed that media arts-related coding education attracts a diverse range of students who might not otherwise engage with CS in a formal setting [13, 26, 45]. Studies such as those by Sáez-López 2016, MacNeil 2022, and Sarsa 2022 indicate the potential of LLMs in this sphere, particularly for middle school families [38]. Furthermore, they demonstrate that LLMs, like GPT-3 and OpenAI Codex, can generate helpful code explanations and programming exercises, providing potential value in creative coding projects.

The current work highlights the potential of Large Language Models (LLMs) in various aspects of coding education, including enhancing code-authoring performance, generating understandable code explanations, and assisting in debugging tasks. Moreover, the promising role of AI code generators and AI-enhanced visualization tools in supporting learners and educators in computer science education has been underscored. However, despite these advancements, a gap persists in understanding LLMs' effectiveness and potential limitations, particularly concerning youth and families engaged in creative coding.

This study seeks to fill this gap by investigating the utility of LLMs in the context of middle school families engaging in creative coding. We focus on the potential of LLMs for generating Scratch program explanations, debugging, and ideation support. In doing so, we aim to contribute to the growing body of research on using LLMs in coding education and provide valuable insights into their utility for this demographic.

2.2 Family Creative Coding

Creative coding, distinct from traditional Computer Science (CS) education, often adopts a *bricolage approach* [28]. This concept, introduced in the programming context by Turkle and Papert [42], paints the coder as a bricoleur, akin to a painter contemplating their canvas between brushstrokes. This approach casts programming as a collaborative venture with the machine, a conversation rather than a monologue, wherein mistakes are not missteps but opportunities for navigation and mid-course corrections.

However, it is crucial to note that the benefits of creative coding extend beyond motivation; they also challenge the assumption that creative coding inherently develops computational thinking and problem-solving skills. For example, a recent study found that novice students often need help using optimal strategies to create animations, even with explicit instruction [44]. This finding highlights the importance of pedagogical approaches in promoting computational thinking in the context of creative coding.

A growing body of research supports the idea that collaborative creative coding, mainly when supported by AI, can effectively engage both children and parents in learning and creating with technology. Prior studies have detailed successful programs where families participate in creative coding workshops, sparking interest and activity in computing among parents and children alike [4, 36].

For instance, Druga et al. delved into how parents can aid their children in developing AI literacies through learning activities, emphasizing the benefits of parent-child partnerships [8]. Another study by Zhang et al. introduced StoryBuddy, an AI-enabled system designed for parents and children to create interactive storytelling experiences. This system caters to dynamic user needs and supports various assessment and educational goals [48].

In summary, collaborative creative coding, bolstered by AI, can be a significant avenue to involve children and parents in learning and technological creation. The existing literature provides different approaches to designing and implementing such programs, informing our current study. Our research extends this work by focusing on the potential of Large Language Models (LLMs) to support such collaborative creative coding experiences, particularly for middle school families. This emphasis on LLMs in the family creative coding context adds a new dimension to the existing discourse, potentially expanding and enhancing these collaborative learning experiences.

2.3 Culturally-Responsive Computing Education

The advent of large language models (LLMs), capable of human-like language generation, has ushered in a new era of technological interaction, which can shape user behavior and opinions. Jakesch [16] suggests that when LLMs express certain viewpoints more frequently than others, they may inadvertently influence user perspectives. The potential bias in LLMs is further substantiated by studies such as those by Gaci [12] and Nadeem [32], which report that pre-trained LLMs often reflect and perpetuate societal stereotypes and biases.

Addressing this concern, several studies have proposed measures to mitigate social biases in LLMs. For instance, Liang et al. [**?**] introduced new benchmarks and metrics for identifying and reducing these biases. In contrast, Mattern et al. [27] proposed a robust framework for quantifying biases exhibited by LLMs. Thus, it becomes crucial to confront and alleviate these biases, especially within the context of computing education.

Culturally Responsive Computing Education (CRC) is an evolving field emphasizing integrating students' identities and experiences into learning. This approach, as championed by Solyst [40] and Morales-Chicas [31], is recognized as key to fostering equity and justice in K-12 education. Moreover, Solyst et al. underline the challenges in fostering a sense of connectedness in online CRC programs and propose strategies to address them [40]. Araujo [3] further advocates for an intercultural approach, promoting relationship-building across differences.

The importance of culturally-responsive approaches is also underscored in the context of AI education for K-12 students [11]. These approaches include personalizing the learning experience, promoting AI ethics understanding among middle school students, and using cultural artifacts to reinforce computing concepts [2]. Moreover, the role of collaborative engagement between schools and communities in fostering equity-oriented CS education is highlighted [21].

In summary, the research underscores the significance of culturally-responsive approaches in computing and AI education and the necessity to address and mitigate the potential biases in LLMs. This current study aims to extend this discourse by exploring the potential of LLMs in a culturally-responsive, family-based creative coding context. Furthermore, we aim to contribute to the ongoing discussion on how to best leverage these advanced tools in a way that respects and integrates diverse cultural perspectives, ultimately promoting an inclusive and effective computing education.

3 METHOD

3.1 Development and Analysis of Scratch Projects

In our study, we curated a collection of 22 Scratch projects (see examples in Figure 2). These projects were selected by referencing popular Scratch community projects and salient examples from a previous study that analyzed 250,000 projects from the Scratch public repository [1]. These projects primarily aimed to evaluate the capabilities of a language learning model (LLM) in supporting code explanation, code debugging, and code ideation. In addition, these three tasks were identified as critical areas of focus based on findings from our previous user study on AI assistants for family creative coding.

We utilized the 22 Scratch projects as inputs for an LLM (GPT4), generating responses with and without practice tasks. This resulted in a pool of 120 creative coding support scenarios. These scenarios were evaluated independently by the first two authors, focusing on precision, pedagogical value, and age-appropriate language. In cases where the two authors disagreed on the evaluation, they engaged in a discussion until a consensus was reached. Our evaluation framework, together with our labeled evaluation data, is publicly available [2].

3.2 Leveraging OpenAI's GPT4 Model

OpenAI's GPT-4, much like its predecessor GPT-3, can be interacted with through an API or a web interface. Users provide GPT-4 with a prompt, which the model uses as a foundation to generate content in alignment with the input. For example, upon receiving a natural language description of desired functionality, GPT-4 often generates corresponding source code.

We can guide the model's content generation by specifying a "stop sequence" to halt generation upon reaching a particular sequence. Our study also employed options such as maximum token count for controlling the content length and "temperature" for influencing the model's level of creativity or randomness. Lower temperature values decrease randomness by reducing the likelihood of generating less probable tokens. However, the model remains non-deterministic regardless of temperature, with variations in content across different runs, especially at higher temperature values.

After conducting several experiments, we settled on the following model parameters for our final evaluation: a temperature of 0.7, a maximum token count of 1024, and a maximum penalty P of 1. These parameters yielded the best results during our prompts testing.

The provided prompt significantly influences the generated content of GPT-4. Therefore, we prompted the model with an existing Scratch program and context-specific natural language instructions to guide GPT-4 in explaining, debugging, or ideating Scratch programs. For example, our prompts would reference the Scratch input program, ask the model to explain it to a middle school child, and evaluate its answer:

> "You are an expert in creative coding for kids in middle school. Explain the following Scratch project {scratch_code} in an accessible and fun way. Provide first a global overview of the project.
> Rate your global response and show a score for how kid-ready your response was." (prompt used for code explanation task without practice)

The complete list of prompts for creating our evaluation dataset is in the appendix.

[2]https://github.com/stefania11/ScratchCopilot-Evaluation

4 EVALUATION

In the following section, we present the main findings of our LLM evaluation organized in the following creative-coding support scenarios: code explanation, debugging support, and ideation support.

4.1 Code Explanation Evaluation

When evaluating the code explanations, we studied whether all parts of the code were explained and whether each line was correctly explained. Of the 40 code explanations, 90% explained all parts of the code (see Table 1).

The LLM generated explanations to help middle schoolers understand their Scratch projects. In addition, TheLLMassistant provided engaging, age-appropriate explanations that were easy to understand, making the Scratch projects more enjoyable for the young coders. For instance, in the example illustrated in Figure 1, the LLM explained the project as "a thrilling game of tag between the two sprites," making it relatable and exciting for the students.

The LLM successfully explained the Scratch code in various projects, breaking down each block's steps and purpose in a way middle schoolers could comprehend. In other instances, the LLM explained a game where the character collects coins and avoids obstacles, detailing the code's structure and the logic behind each section. This helped students grasp the concepts more effectively.

Although the LLM provided accurate explanations, the tone of language was sometimes overly enthusiastic, using phrases like "Hey there, Star coder" or "Super coder, let's look at this program."

In several instances, the model explained more complex games where users collect coins and avoid obstacles to increase their scores. The model broke down each section of the code, explaining the start of the game, the continuous loop, the conditions when the sprite touches a coin or an obstacle, and the game's conclusion when the score exceeds 100. It provides a systematic and detailed breakdown, clearly understanding each function.

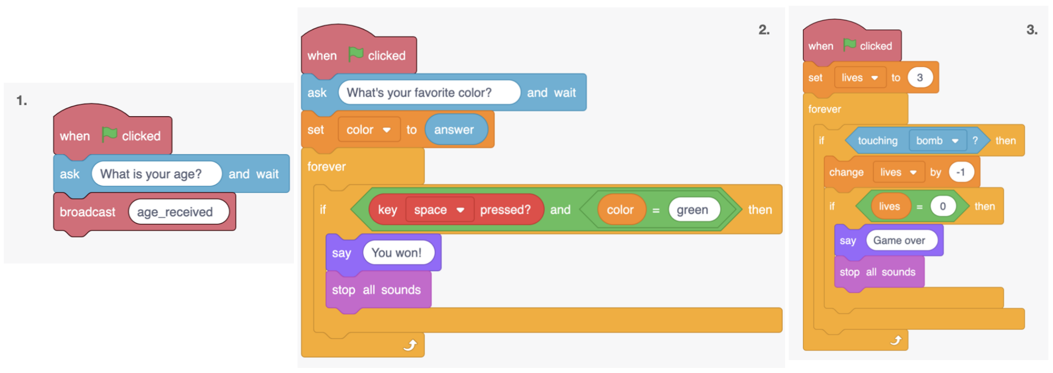
In other examples, the LLM created appropriate metaphors for explaining more complex computational concepts such as variables or loops. For example, in one instance where the sprite's size was changing continuously, the model likened the sprite to a balloon that inflates and deflates, making the code's dynamics easier to understand. It explains how the sprite grows until it reaches a specific size, then starts to shrink, creating a continuous cycle. The model's imaginative and engaging language makes the code's purpose clear and appealing, encouraging students to explore further.

Overall, the LLM demonstrated its ability to support middle schoolers in understanding their Scratch projects through engaging, informative explanations.

4.2 Code Debugging Evaluation

When evaluating the code debugging, we studied whether GPT4 could identify the correct bug and provide adequate support. Of the 40 code debugging examples, 80% correctly identified the introduced bugs.

The LLM showed promise in aiding middle schoolers to debug their Scratch projects. However, although the LLM correctly identified bugs 80% of the time, it occasionally suggested creating variables when the actual issue was related to conditionals. These



```
when [flag] clicked
ask (What is your age?) and wait
broadcast (age_received)
```

```
when [flag] clicked
ask (What's your favorite color?) and wait
set [color] to (answer)
forever
  if <<key [space] pressed?> and <(color) = (green)>> then
    say (You won!)
    stop all sounds
```

```
when [flag] clicked
set [lives] to (3)
forever
  if <touching [bomb] ?> then
    change [lives] by (-1)
    if <(lives) = (0)> then
      say (Game over)
      stop all sounds
```

Figure 2: Examples of input Scratch programs provided as input to the LLM:1.Asking player age, 2. Asking the player to guess a favorite color, 3. Game for avoiding bombs

Coding Task	Correct suggestions	Notes
Explain code	100%	No errors, but the tone of language was overly enthusiastic at times (i.e., “Hey there, Star coder,” “Super coder let’s look at this program”)
Explain code with learning	100%	More line by line explanations in this mode.
Debug code	80%	The model would find bugs even in correct programs, in two examples, it suggests creating variables instead of finding conditionals errors
Debug code with learning	90%	Did not detect “set score” instead of “change score” and play concurrent sounds bugs
Code ideas	100%	When the initial program was generic, it would suggest similar suggestions such as adding counters, timers, levels, multimodality
Code ideas with learning	100%	Whenever the initial program had something more specific (i.e., reference to favorite color) the model would develop that in creative ways otherwise, it would default to suggesting common game mechanics prevalent in existing scratch projects

Table 1: summary of model evaluation

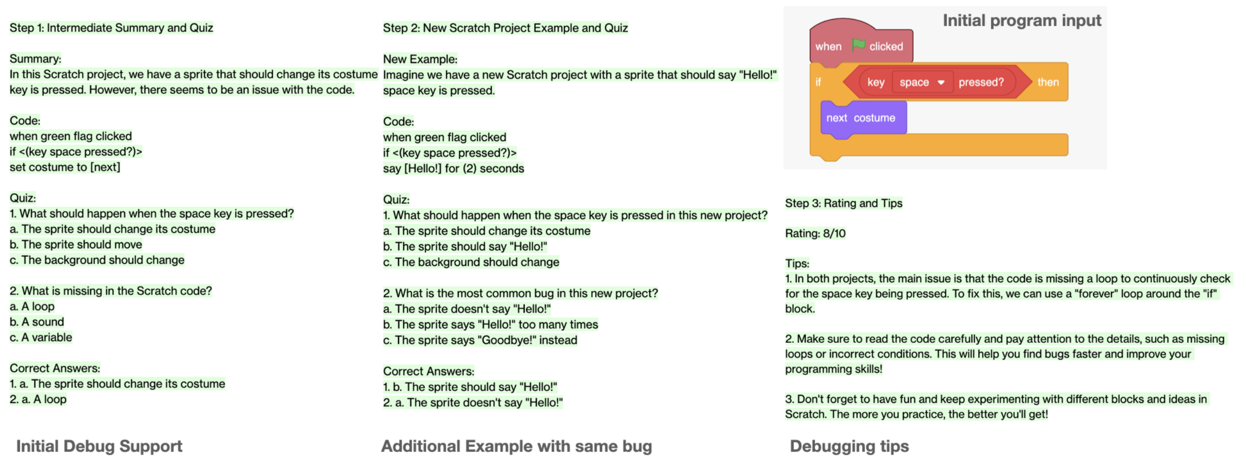
debugging errors were primarily due to the lack of context for the Scratch project input, as the model did not have access to prior code explorations or code edits from the same project.

A noteworthy example of the debugging support proposed by the model involved the model using an interesting superhero analogy to elucidate the necessity of a “forever” loop in a student’s code, thereby enhancing the functionality of the project (see Figure3).

In another scenario, the LLM identified that a student had not created a “lives” variable, confusing their project. The model guided the student through creating this variable, thus resolving the problem and making the project work as intended. However, the model did have some shortcomings; for instance, it failed to detect bugs, such as using “set score” instead of “change score” and suggested adding a “play sound” block rather than addressing the actual issue.

In the provided examples, the LLM generated interactive quizzes and practical tips as part of its pedagogical approach to facilitating debugging in Scratch projects. The quizzes were structured to summarize the project and then ask targeted questions about the intended behavior and the potential bug in the code. This approach emphasizes the comprehension of the code and encourages the learners to think critically about potential issues. For instance, in the first example, the LLM introduced a quiz highlighting the need for a loop to continuously check the space key press event (see Figure3).

Furthermore, the model provided tips after each interactive quiz session, offering advice on improving coding skills and debugging more effectively. These tips ranged from technical guidance, such as adding necessary loops or paying attention to missing elements in the code, to fostering a positive learning mindset, such as encouraging experimentation and maintaining a fun approach to coding. For example, in the second example, the model advised learners to provide clear instructions, experiment with different code blocks, and use simple language and engaging analogies. Combined with the



Step 1: Intermediate Summary and Quiz

Summary:
In this Scratch project, we have a sprite that should change its costume key is pressed. However, there seems to be an issue with the code.

Code:
when green flag clicked
if <(key space pressed?)>
set costume to [next]

Quiz:
1. What should happen when the space key is pressed?
a. The sprite should change its costume
b. The sprite should move
c. The background should change

2. What is missing in the Scratch code?
a. A loop
b. A sound
c. A variable

Correct Answers:
1. a. The sprite should change its costume
2. a. A loop

Step 2: New Scratch Project Example and Quiz

New Example:
Imagine we have a new Scratch project with a sprite that should say "Hello!" space key is pressed.

Code:
when green flag clicked
if <(key space pressed?)>
say [Hello!] for (2) seconds

Quiz:
1. What should happen when the space key is pressed in this new project?
a. The sprite should change its costume
b. The sprite should say "Hello!"
c. The background should change

2. What is the most common bug in this new project?
a. The sprite doesn't say "Hello!"
b. The sprite says "Hello!" too many times
c. The sprite says "Goodbye!" instead

Correct Answers:
1. b. The sprite should say "Hello!"
2. a. The sprite doesn't say "Hello!"

Step 3: Rating and Tips

Rating: 8/10

Tips:
1. In both projects, the main issue is that the code is missing a loop to continuously check for the space key being pressed. To fix this, we can use a "forever" loop around the "if" block.

2. Make sure to read the code carefully and pay attention to the details, such as missing loops or incorrect conditions. This will help you find bugs faster and improve your programming skills!

3. Don't forget to have fun and keep experimenting with different blocks and ideas in Scratch. The more you practice, the better you'll get!

Figure 3: Examples of debugging code support provided by the model.

quizzes, these insights establish an effective learning environment that fosters understanding and creativity in the coding process.

4.3 Code Ideation Evaluation

We studied whether LLMs could suggest relevant ideas for Scratch projects when evaluating the code ideations. All the suggestions made by the model were correct and written in kid-friendly language.

The LLM model suggested various ways to enhance Scratch projects for middle schoolers, focusing on interactivity, challenges, and visual appeal. In one example, the model proposed adding a sprite that follows the mouse pointer, introducing a timer for a time-based challenge, and altering the sprite's costume to increase visual appeal. Additionally, the model recommended incorporating levels with increasing difficulty and sound effects and music to create a more engaging and entertaining experience.

Another example showcased the AI's ability to offer creative suggestions for code modification, such as changing the sprite's appearance and color with each loop iteration, creating a visually pleasing effect. The model also suggested incorporating user input to control the number of loop repetitions, making the project more interactive (see Figure 4). Furthermore, the LLM proposed transforming the project into a game by adding collectible objects, a scoring system, and advancing levels with increasing difficulty.

In other instances, the LLM model focused on making the game more engaging and educational by adding hints based on color guesses. For instance, if a child guesses red, the hint informs them that their favorite color is of a cooler tone than red. This approach makes the game more engaging and promotes learning through feedback. The model also recommended adding a timer and associating sound effects with user actions to enhance the gaming experience (see Figure 4).

5 DISCUSSION

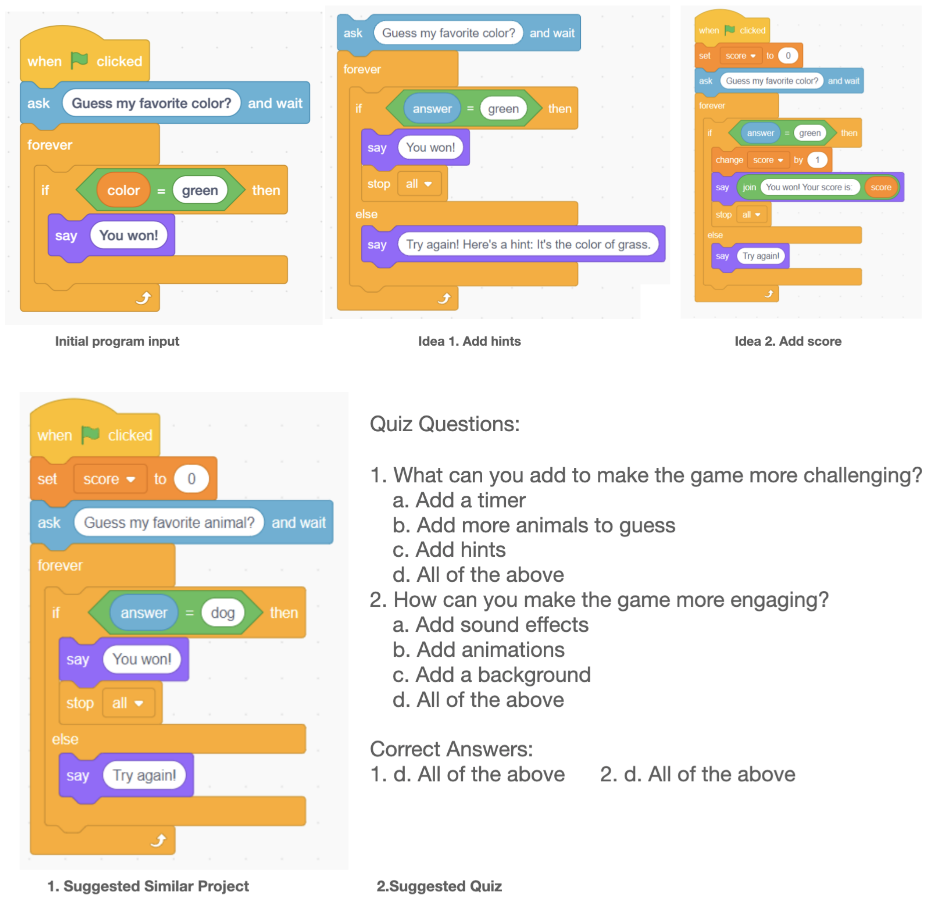
Our work asked: *How well do large-language models support explaining, ideating, and debugging Scratch projects for middle-school families?* Our study revealed that in the case of simple Scratch programs, LLMs such as GPT4 can achieve high precision and accuracy when generating code explanations, debugging supports, and code ideas. Moreover, we found that despite the model being overly enthusiastic sometimes, the language used in the support scenarios generated was child appropriate.

Our findings illuminate several implications for applying LLM models in supporting children's creative coding. For instance, the model's capacity to introduce new ideas and modifications often assumes a certain level of knowledge among children. Although this can benefit those with some experience, it may create difficulties for beginners. Therefore, future models must be designed with mechanisms to assess a child's knowledge level and adjust their support accordingly. Also, these models should be equipped to identify and rectify any incorrect assumptions about the child's knowledge or skill level.

Despite the LLM's ability to generate suggestions, these were sometimes off-topic or made assumptions about Scratch's capabilities that did not align with the child's project goals. In addition, as in previous studies [1, 35], the LLM sometimes prompted children to write complex code, resulting in "code smells" or bad programming practices. This suggests that future LLMs should aim to restrict overly complex or off-topic suggestions, thereby providing more personalized and accurate support.

Our findings also pointed to the need for LLMs to assess a child's knowledge level and adjust their support accordingly. For example, the LLM often introduced new ideas that required a certain level of understanding, which could be challenging for beginners. Similarly, it should be able to identify incorrect assumptions about the child's knowledge or skill level and rectify them, ensuring the child is not overwhelmed or misinformed.

Interestingly, the LLM was found to repetitively suggest the same ideas, limiting the scope for creative thinking. Therefore, future iterations of LLM models should aim to generate broader suggestions to encourage diverse creative thoughts. This aligns



```
when ⚑ clicked
ask Guess my favorite color? and wait
forever
    if color = green then
        say You won!
```

```
ask Guess my favorite color? and wait
forever
    if answer = green then
        say You won!
        stop all
    else
        say Try again! Here's a hint: It's the color of grass.
```

```
when ⚑ clicked
set score to 0
ask Guess my favorite color? and wait
forever
    if answer = green then
        change score by 1
        say join You won! Your score is score
        stop all
    else
        say Try again!
```

```
when ⚑ clicked
set score to 0
ask Guess my favorite animal? and wait
forever
    if answer = dog then
        say You won!
        stop all
    else
        say Try again!
```

Quiz Questions:

1. What can you add to make the game more challenging?
   a. Add a timer
   b. Add more animals to guess
   c. Add hints
   d. All of the above
2. How can you make the game more engaging?
   a. Add sound effects
   b. Add animations
   c. Add a background
   d. All of the above

Correct Answers:
1. d. All of the above 2. d. All of the above

Figure 4: Examples of code ideas provided by the model.

with the need for LLMs to stimulate creativity by suggesting non-conventional or niche project ideas, broadening the child's exposure to different concepts and genres.

While the LLM proved helpful in debugging, it sometimes failed to identify specific bugs, such as the local "change lives" bug. This highlights the need for more context awareness in the model to understand the overall goal of the program better. In addition, future models should be designed to express uncertainty when appropriate, enabling children to consider a broader range of possibilities and make more informed decisions about their projects.

Finally, regarding communication style, LLM models should balance their tone to avoid overwhelming young learners with overly enthusiastic responses. Instead, they should foster an environment that encourages exploration and learning at the child's own pace.

In conclusion, our study underscores the potential of LLMs in enhancing creative coding for children and the need for future iterations to address the highlighted areas of improvement. Our findings contribute to the growing discourse on using LLMs in coding education and align with prior work advocating for culturally-responsive, family-based creative coding contexts. These insights will inform the design of future LLMs, ultimately promoting inclusive and effective computing education.

5.1 Design Guidelines for AI-Enhanced Creative Coding Tools

Our current and previous research [9] suggests some guidelines for designing AI-enhanced creative coding tools. The tools should not answer the learners but guide them through their creative process. The fact that in our outputs, the LLM gives options and different suggestions for the youth to evaluate and pick from is a first step in this direction. The LLM should adapt the output to the right learning level based on the youth's age, prior experience, and reactions to its previous suggestion. The LLM's diversity of answers in our study showcases the possibility of reaching different learning levels, but testing that LLM accurately delivers them is still needed. The following list enumerates more of our design suggestions:

Promote Agency and Self-expression. To foster creativity and self-driven learning, LLM tools should stimulate children's thinking by posing strategic questions instead of providing direct answers [17]. This support style enhances the Scratch coding experience and aligns with the learners' preference for agency and self-expression.

Experience Influences Support Needed. LLM support should be tailored to the coder's experience level to maximize learning outcomes. For example, novice coders often require more assistance with coding game ideas, whereas intermediate coders may benefit more from ideation and debugging support [17].

Explain the Provenance of Suggestions. LLM tools should provide transparency about how they generate a suggestion and offer information about the source of the code example. This prevents misconceptions and enhances understanding of the AI's suggestions, which is particularly useful in Scratch projects [47].

Multimodal Debugging Support. LLM tools should offer visual elements alongside text to clarify complex instructions and aid in locating specific programming blocks, especially given the visual nature of creative coding [29]. This approach aligns with previous research indicating that augmenting text with visuals provides a more natural coding specification method.

Voice Input as "Third Hand." Voice input can provide a valuable, hands-free interaction method with the LLM tool, especially when children and parents are collaboratively working on their Scratch project [23, 30, 39]. However, designing this feature for diverse programmers, including children and parents, requires overcoming challenges such as recognizing children's speech or foreign accents [20].

Live Code Execution Incorporating *liveness* in the coding platform allows for auto-execution of code, helping users quickly identify non-functioning scripts and offering immediate debugging opportunities. This feature aligns with research on the benefits of immediate feedback in education. It can be especially beneficial in family creative coding scenarios where multiple users may collaborate on a single project [14, 18, 41].

Support Diverse Ideas and Projects LLM tools should encourage various project types, including art projects, story-based experiences, and projects that encourage collaborative mechanics, going beyond mainstream or competitive games. This aspect aligns with the broader goal of fostering creativity and diversity in the Scratch coding environment.

5.2 Future Work

In our future work, we primarily aim to delve deeper into the capabilities of LLM Companions in facilitating joint family engagement in creative coding. Understanding how LLM can foster shared learning experiences and promote collaboration among family members is still an open question. This will necessitate evaluating LLM models in multi-turn conversation scenarios involving children and parents, allowing us to comprehend better how LLM can support diverse family learning contexts. We also plan to evaluate LLM models on more complex Scratch programs. This will help us cater to a range of programming competencies and extend the utility of LLM in creative coding. Additionally, we aim to explore the potential of LLM models in providing asynchronous support on multiple projects. Finally, drawing inspiration from research on novice design [6], we hope to empower young coders to develop better programming skills and foster creativity by exploring multiple ideas before receiving feedback.

5.3 Limitations

While our study has provided valuable insights into the potential of LLM models in enhancing creative coding for families, it is not without its limitations. First, the list of input programs we used for evaluating the LLM model was not exhaustive. The Scratch projects we used were a representative sample, but they do not capture the entire range of programs kids create on Scratch. This vast diversity in creative coding ranges from simple animations to complex games, and our sample may not fully represent this spectrum.

Second, our evaluation scenarios focused primarily on code explanation, debugging, and ideation. While these are critical aspects of creative coding, they do not encompass all possible scenarios kids might need support. There are other areas, such as program design, structuring code, or even specific topics like working with clones and lists in Scratch, where LLM assistance could be beneficial but were not included in our study. Future research should include broader coding scenarios and challenges to assess better LLMs' potential in supporting creative coding for families.

6 CONCLUSION

This study explored the potential of large language models (LLMs) in enhancing creative coding experiences for families using Scratch. Building upon our previous user studies on AI-Assisted family creative coding, we conducted an extensive evaluation to determine how effectively LLMs could assist in understanding game code, debugging programs, and generating innovative ideas for future creative coding projects. Our research involved meticulously analyzing 120 creative coding support scenarios, incorporating LLMs' responses with and without practice tasks. In addition, our authors independently assessed each scenario on critical criteria, such as accuracy, pedagogical value, and age-appropriate language.

Our findings revealed that LLMs consistently achieved an impressive success rate of over 80% across different tasks and evaluation criteria, signifying their considerable potential in supporting family-based creative coding. However, as with any emerging technology, there are areas for refinement and improvement. Our research highlighted the need for more context awareness, diversified suggestions, adaptive communication styles, and improved debugging support in future iterations of LLMs. In conclusion, our research contributes valuable insights into the potential of LLMs in family creative coding. It provides a robust foundation for future research and development in AI-supported coding applications. These findings inform the design of more effective, engaging, and inclusive tools for creative coding education, paving the way for more families to experience the joy and learning opportunities that creative coding can provide.

REFERENCES

[1] Efthimia Aivaloglou and Felienne Hermans. 2016. How kids code and how we know: An exploratory study on the Scratch repository. In *Proceedings of the 2016 ACM Conference on International Computing Education Research.* 53–61.

[2] Ebenezer Anohah and Jarkko Suhonen. 2020. *Conceptual Model of Generic Learning Design to Teach Cultural Artifacts in Computing Education.* IGI Global, 279–294. https://doi.org/10.4018/978-1-7998-0423-9.ch015

[3] Ian Arawjo and Ariam Mogos. 2021. Intercultural Computing Education: Toward Justice Across Difference. *ACM Transactions on Computing Education* 21, 4 (oct 25 2021), 1–33. https://doi.org/10.1145/3458037

[4] Nina Bresnihan, Glenn Strong, Lorraine Fisher, Richard Millwood, and Áine Lynch. 2019. OurKidsCode: Facilitating Families to Be Creative with Computing. In *Proceedings of the 11th International Conference on Computer Supported Education*. SCITEPRESS - Science and Technology Publications. https://doi.org/10.5220/0007729405190530

[5] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128* (2023).

[6] Steven P Dow, Alana Glassco, Jonathan Kass, Melissa Schwarz, Daniel L Schwartz, and Scott R Klemmer. 2010. Parallel prototyping leads to better design results, more divergence, and increased self-efficacy. *ACM Transactions on Computer-Human Interaction (TOCHI)* 17, 4 (2010), 1–24.

[7] Stefania Druga, Thomas Ball, and Amy Ko. 2022. How families design and program games: a qualitative analysis of a 4-week online in-home study. In *Interaction Design and Children*. 237–252.

[8] Stefania Druga, Fee Lia Christoph, and Amy J Ko. 2022. Family as a Third Space for AI Literacies: How do children and parents learn about AI together?. In *CHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/3491102.3502031

[9] Stefania Druga and Amy J. Ko. 2023. AI Friends: Designing Creative Coding Assistants for Families. *Proceedings TOCE* (5 2023). https://arxiv.org/submit/4898318/view

[10] Tomi Slotte Dufva. 2021. Creative coding as compost (ing). *Post-digital, post-internet art and education: The future is all-over* (2021), 269–283.

[11] Amy Eguchi, Hiroyuki Okada, and Yumiko Muto. 2021. Contextualizing AI Education for K-12 Students to Enhance Their Learning of AI Literacy Through Culturally Responsive Approaches. *KI - Künstliche Intelligenz* 35, 2 (6 2021), 153–161. https://doi.org/10.1007/s13218-021-00737-3

[12] Yacine Gaci, Boualem Benatallah, Fabio Casati, and Khalid Benabdeslem. 2022. Masked Language Models as Stereotype Detectors? https://doi.org/10.48786/EDBT.2022.26

[13] Ira Greenberg. 2007. *Processing: creative coding and computational art*. Springer.

[14] Philip J Guo. 2013. Online python tutor: embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM technical symposium on Computer science education*. 579–584.

[15] Saki Imai. 2022. Is GitHub copilot a substitute for human pair-programming? An empirical study. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*. 319–321.

[16] Maurice Jakesch, Advait Bhat, Daniel Buschek, Lior Zalmanson, and Mor Naaman. 2023. Co-Writing with Opinionated Language Models Affects Users' Views. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–15.

[17] Dhanya Jayagopal, Justin Lubin, and Sarah E Chasins. 2022. Exploring the Learnability of Program Synthesizers by Novice Programmers. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–15.

[18] Hyeonsu Kang and Philip J Guo. 2017. Omnicode: A novice-oriented live programming environment with always-on run-time value visualizations. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. 737–745.

[19] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/3544548.3580919

[20] James Kennedy, Séverin Lemaignan, Caroline Montassier, Pauline Lavalade, Bahar Irfan, Fotios Papadopoulos, Emmanuel Senft, and Tony Belpaeme. 2017. Child speech recognition in human-robot interaction: evaluations and recommendations. In *Proceedings of the 2017 ACM/IEEE international conference on human-robot interaction*. 82–90.

[21] Michael Lachney, Audrey G. Bennett, Ron Eglash, Aman Yadav, and Sukanya Moudgalya. 2021. Teaching in an open village: a case study on culturally responsive computing in compulsory education. *Computer Science Education* 31, 4 (feb 2 2021), 462–488. https://doi.org/10.1080/08993408.2021.1874228

[22] Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A Becker. 2023. Using large language models to enhance programming error messages. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 563–569.

[23] Phoebe Lin, Jessica Van Brummelen, Galit Lukin, Randi Williams, and Cynthia Breazeal. 2020. Zhorai: Designing a conversational agent for children to explore machine learning concepts. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 13381–13388.

[24] Duri Long, Anthony Teachey, and Brian Magerko. 2022. Family Learning Talk in AI Literacy Learning Activities. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–20.

[25] Aman Madaan, Alexander Shypula, Uri Alon, Milad Hashemi, Parthasarathy Ranganathan, Yiming Yang, Graham Neubig, and Amir Yazdanbakhsh. 2023. Learning performance-improving code edits. *arXiv preprint arXiv:2302.07867* (2023).

[26] Mihaela Malita and Ethel Schuster. 2020. From drawing to coding: teaching programming with processing. *Journal of Computing Sciences in Colleges* 35, 8 (2020), 245–246.

[27] Justus Mattern, Zhijing Jin, Mrinmaya Sachan, Rada Mihalcea, and Bernhard Schölkopf. 2022. Understanding Stereotypes in Language Models: Towards Robust Measurement and Zero-Shot Debiasing. (2022). https://doi.org/10.48550/ARXIV.2212.10678

[28] Alex McLean and Geraint Wiggins. 2012. Computer programming in the creative arts. *Computers and Creativity* (2012), 235–252.

[29] Andrew M Mcnutt, Anton Outkine, and Ravi Chugh. 2023. A Study of Editor Features in a Creative Coding Classroom. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–15.

[30] Microsoft. 2023. GitHub Next | Hey, GitHub! https://githubnext.com/projects/hey-github/. (Accessed on 02/15/2023).

[31] Jessica Morales-Chicas, Mauricio Castillo, Ireri Bernal, Paloma Ramos, and Bianca Guzman. 2019. Computing with Relevance and Purpose: A Review of Culturally Relevant Education in Computing. *International Journal of Multicultural Education* 21, 1 (mar 4 2019), 125–155. https://doi.org/10.18251/ijme.v21i1.1745

[32] Moin Nadeem, Anna Bethke, and Siva Reddy. 2021. StereoSet: Measuring stereotypical bias in pretrained language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.acl-long.416

[33]]PaulTowards Paul Pu Liang, Chiyu Wu, Louis-Philippe Morency, and R. Salakhutdinov. [n. d.]. Towards Understanding and Mitigating Social Biases in Language Models.

[34] Natasha Pearce, Helen Monks, Narelle Alderman, Lydia Hearn, Sharyn Burns, Kevin Runions, Jacinta Francis, and Donna Cross. 2022. 'It's all about context': Building school capacity to implement a whole-school approach to bullying. *International Journal of Bullying Prevention* (2022), 1–16.

[35] Gregorio Robles, Jesús Moreno-León, Efthimia Aivaloglou, and Felienne Hermans. 2017. Software clones in scratch projects: On the presence of copy-and-paste in computational thinking learning. In *2017 IEEE 11th International Workshop on Software Clones (IWSC)*. IEEE, 1–7.

[36] Ricarose Roque. 2016. Family creative learning. *Makeology: Makerspaces as learning environments* 1 (2016), 47–63.

[37] Ricarose Roque and Natalie Rusk. 2019. Youth perspectives on their development in a coding community. *Information and Learning Sciences* (2019).

[38] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*. 27–43.

[39] Serenade. 2023. Serenade | Code with voice. https://serenade.ai/. (Accessed on 02/15/2023).

[40] Jaemarie Solyst, Tara Nkrumah, Angela Stewart, Amanda Buddemeyer, Erin Walker, and Amy Ogan. 2022. Insights from Virtual Culturally Responsive Computing Camps. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2*. ACM. https://doi.org/10.1145/3478432.3499136

[41] Steven L Tanimoto. 2013. A perspective on the evolution of live programming. In *2013 1st International Workshop on Live Programming (LIVE)*. IEEE, 31–34.

[42] Sherry Turkle and Seymour Papert. 1990. Epistemological pluralism: Styles and voices within the computer culture. *Signs: Journal of women in culture and society* 16, 1 (1990), 128–157.

[43] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*. 1–7.

[44] Karen Woo and Garry Falloon. 2022. Problem solved, but how? An exploratory study into students' problem solving processes in creative coding tasks. *Thinking Skills and Creativity* 46 (2022), 101193.

[45] Zoe J Wood, Paul Muhl, and Katelyn Hicks. 2016. Computational art: Introducing high school students to computing via art. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. 261–266.

[46] Aman Yadav, Sarah Gretter, Susanne Hambrusch, and Phil Sands. 2016. Expanding computer science education in schools: understanding teacher experiences and challenges. *Computer science education* 26, 4 (2016), 235–254.

[47] Weixiang Yan and Yuanchun Li. 2022. WhyGen: explaining ML-powered code generation by referring to training examples. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*. 237–241.

[48] Zheng Zhang, Ying Xu, Yanhao Wang, Bingsheng Yao, Daniel Ritchie, Tongshuang Wu, Mo Yu, Dakuo Wang, and Toby Jia-Jun Li. 2022. StoryBuddy: A Human-AI Collaborative Chatbot for Parent-Child Interactive Storytelling with Flexible Parental Involvement. In *CHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/3491102.3517479