Learning Agent-based Modeling with LLM Companions: Experiences of Novices and Experts Using ChatGPT & NetLogo Chat

John Chen
Northwestern University
Evanston, IL, United States of America
civitas@u.northwestern.edu

Xi Lu
University of California, Irvine
Irvine, CA, United States of America
xlu30@uci.edu

David Du
Northwestern University
Evanston, IL, United States of America
duyuzhou2013@gmail.com

Michael Rejtig
University of Massachusetts Boston
Boston, MA, United States of America
michael.rejtig001@umb.edu

Ruth Bagley
Northwestern University
Evanston, IL, United States of America
ruth.bagley@northwestern.edu

Michael S. Horn
Northwestern University
Evanston, IL, United States of America
michael-horn@northwestern.edu

Uri J. Wilensky
Northwestern University
Evanston, IL, United States of America
uri@northwestern.edu

ABSTRACT

Large Language Models (LLMs) have the potential to fundamentally change the way people engage in computer programming. Agent-based modeling (ABM) has become ubiquitous in natural and social sciences and education, yet no prior studies have explored the potential of LLMs to assist it. We designed NetLogo Chat to support the learning and practice of NetLogo, a programming language for ABM. To understand how users perceive, use, and need LLM-based interfaces, we interviewed 30 participants from global academia, industry, and graduate schools. Experts reported more perceived benefits than novices and were more inclined to adopt LLMs in their workflow. We found significant differences between experts and novices in their perceptions, behaviors, and needs for human-AI collaboration. We surfaced a knowledge gap between experts and novices as a possible reason for the benefit gap. We identified guidance, personalization, and integration as major needs for LLM-based interfaces to support the programming of ABM.

CCS CONCEPTS

• **Human-centered computing** → **Empirical studies in HCI**; **Natural language interfaces**; • **Computing methodologies** → *Simulation support systems.*

CHI '24, May 11–16, 2024, Honolulu, HI, USA

ACM ISBN 979-8-4007-0330-0/24/05
https://doi.org/10.1145/3613904.3642377

KEYWORDS

Agent-based Modeling, NetLogo Chat, ChatGPT, Programming Assistant, LLM Companion, Learning with LLMs

ACM Reference Format:
John Chen, Xi Lu, David Du, Michael Rejtig, Ruth Bagley, Michael S. Horn, and Uri J. Wilensky. 2024. Learning Agent-based Modeling with LLM Companions: Experiences of Novices and Experts Using ChatGPT & NetLogo Chat. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24), May 11–16, 2024, Honolulu, HI, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3613904.3642377

1 INTRODUCTION

The advent of coding-capable Large Language Models (LLMs) has the potential to fundamentally change the way people engage in computer programming[25]. As LLM-based programming interfaces (e.g. GitHub Copilot; ChatGPT) become increasingly popular[46], some studies started to study their user perceptions[84]. However, the research on their potential learning impacts is still limited. Many prior studies only focus on impressions of educators[46] or students[100], with little empirical data on the actual learning usage of these tools. On the other hand, a few studies started to explore how LLM-based interfaces can be designed to facilitate programming education, indicating potential advantages for learners. Notably, these studies suggest that learners with more prior programming experience tend to benefit more[42, 57]. While a recent study identifies some challenges for novice learners with LLM-based interfaces[101], there is a gap in understanding why experienced programmers seem to gain more learning benefits from these tools.

In this paper, we present the design of a novel LLM-based interface, NetLogo Chat, for the learning and practice of NetLogo. NetLogo is a widely used programming language for agent-based

modeling (ABM), which applies simple rules on multiple individual agents to simulate complex systems[94]. It is particularly powerful in capturing emergent phenomena, e.g., the spread of viruses or predator-prey systems[93]. It is an important methodology in computational modeling across scientific disciplines and education from K-12 to postgraduate levels[88], where scientists and educators are highly in need of LLM-based interfaces[21, 59]. As an important part of computational modeling, the priorities of ABM differ from general programming[65]. A modeler needs to verify that their conceptual design of individual rules matches the real-world patterns (e.g. a predator needs food to survive), the code matches the design (i.e. there are no unexpected or implicit assumptions), and the aggregated outcome matches real-world phenomena (e.g. if all prey die out, predators die too)[28]. As most LLM-related studies on computer programming work on general-purpose languages that LLMs perform best (e.g. Python or Javascript), no LLM-related studies have explored ABM or other forms of computational modeling at this point.

NetLogo Chat was designed with constructionist learning principles and incorporated known best practices for ABM and computer programming. Constructionism advocates for the design of learning experiences where learners construct their understanding of the world (e.g. knowledge of ABM) through building personally meaningful artifacts (e.g. an agent-based model around learners' interests)[61]. Similar to GitHub Copilot Chat[1], NetLogo Chat was integrated into an integrated development environment (IDE). Different from previous designs, it aims to give users more control over the human-AI collaboration processes, strives to incorporate authoritative sources, and tries to provide more support for troubleshooting.

Using both ChatGPT and NetLogo Chat as a probe[101], we conducted a qualitative study to highlight the different perceptions, behaviors, and needs of experts and novices during open-ended modeling sessions. We interviewed 30 expert and novice participants from academia, industry, and graduate schools worldwide. Participants proposed diverse NetLogo tasks from their disciplines and worked toward their modeling goals. We asked interview questions before, during, and after their interaction with each design. We answered the research questions:

(1) What perceptions - strengths, weaknesses, and adoption plans - do expert and novice users perceive LLM-driven interfaces to support their NetLogo learning and practice?
(2) How do expert and novice users use LLM-driven interfaces to support their NetLogo learning and practice?
(3) What are expert and novice users' needs for LLM-based interfaces to support their NetLogo learning and practice?

Learners generally agreed with our design principles and suggested additional features for future designs. As in other studies, experts reported more perceived benefits than novices. Comparing the different interaction patterns between experts and novices, our study reveals a behavioral gap that might explain the gap in benefits. We found that experts collaborated with LLM-based interfaces with more human judgment in all activities than novices, helping them overcome AI hallucinations, while novices struggled with evaluating and debugging AI responses. From there, we identified components of a knowledge gap between novices and experts. We reported experts' and novices' needs in LLM-based interfaces in three key themes: guidance (from LLMs); personalization (of LLMs); and integration (into modeling environments), many of which confirm and develop the design decisions of NetLogo Chat. The contributions of this paper include:

(1) The design and implementation of NetLogo Chat, an LLM-based system that supports learning and practice of NetLogo, a widely-used programming language for ABM;
(2) An empirical study that contributes to the understanding of how novices and experts perceive, use, and express needs for LLM-based programming interfaces in different ways;
(3) A theorization of the knowledge gap between experts and novices that might lead to the behavioral gap, and suggestions of potential design interventions;
(4) The design discussion and suggestions for building LLM-based programming interfaces that benefit both experts and novices in agent-based modeling more equitably.

2 RELATED WORK

2.1 LLMs for Computational Programming and Modeling

Researchers have been exploring natural-language-based interfaces for programming for decades, yet early attempts were mostly exploratory and limited in capabilities. NaturalJava[64] required users to follow a strict pattern when prompting, while later systems (e.g. NaLIX[47] or Eviza[73]) asked for a specific set of English expressions. This created difficulties for users and system designers, as they felt "a main challenge of NLP interfaces is in communicating to the user what inputs are supported."[73] Without the capability to generate natural languages, those interfaces were also constrained to one-off interactions.

Recently, a new generation of LLMs demonstrated the capability to understand and generate natural and computer languages. GPT-3 was examined in writing code explanations[52], documentation[44], and providing feedback for assignments[3]. Soon, educators started to believe that Codex could be used to solve simple programming problems[27, 90]. Embedded in ChatGPT, GPT-3.5-turbo and GPT-4 demonstrated even stronger capabilities in programming. More and more LLMs have gained the capability of coding (e.g. PALM 2; Claude 2; CodeLLaMA 2), ushering in a new era of natural language interfaces for programming.

Even the most powerful LLMs suffer from hallucinations and may misunderstand human intentions. Early users of ChatGPT complained about incorrect responses and struggled to prompt ChatGPT for a desired output[74]. While LLMs might outperform average humans in specific, structured tasks[58], the evaluation criteria might have been flawed[50], as LLMs struggled to combine existing solutions for a novel challenge[23]. A study suggested developers should not rely on ChatGPT when dealing with new problems [81].

LLMs are naturally less prepared in low-resource programming languages (LRPL). Here, our working definition for LRPL is similar to that of natural languages: with relatively scarce online resources and have been less studied by the AI field[53]. LRPLs are not less important: NetLogo, the most widely used programming language for agent-based modeling (ABM)[80], is used by hundreds of thousands

of scientists, educators, and students for computational modeling. ABM could simulate complicated emergent phenomena using simple computational rules for individual agents. It has been frequently used in different scientific disciplines[93] and science education[35] for recent decades. With considerably fewer online resources to train on, LLMs are much more prone to errors and/or hallucinations with LRPLs[79].

A few studies attempted to improve LLMs' performance with LRPLs in two directions. First, some studies fine-tuned foundational LLMs with LRPL datasets[12]. While this approach demands considerable datasets and computational power, it has not been applied to generative tasks yet[31]. Second, some studies used prompt engineering techniques. For example, aiming at simple tasks, a study creates grammar rules for LLMs to fill in[86]. Another study leveraged compiler outputs, allowing LLMs to improve their Rust code iteratively, but was only tested in a smaller number of fixed tasks[97]. The potential of LLMs in scientific disciplines, including in computational modeling, is rarely explored. Currently, the only study targeted at STEM helps with a very specific engineering task [45].

2.2 User Perception and Behaviors with LLM-based Programming Interfaces

Two strands of user perception and behaviors studies informed our design and study: studies of conversational agents (CAs); and LLM-based programming interfaces. For education, CAs were used to develop learners' writing[85], self-talk[29], and programming skills[95]. Many of them are pedagogical conversational agents (PCA) to mimic the behaviors of human tutors adaptively[96]. PCAs could serve in multiple roles, such as tutors[85], motivators[11], peer players[30], or learning companions[29].

Prior research of CAs underscored the importance of understanding user perception and behaviors[30], yet the technical boundaries of the pre-LLM era limited designers' freedom. Previous studies have explored aspects such as trust, mutual understanding, perceived roles[18], privacy[69], human-likeness[36], utilitarian benefits, and user-related factors[49] to understand users' acceptance and willingness to use CAs. However, many CAs before LLMs had to use pre-programmed responses[87], and simply emulating functional rules from human speech failed to deliver people's high expectations of CAs[19]. Without the capability to read or write code, pre-LLM CAs for computing education were largely limited to providing relevant knowledge[95] or supporting conceptual understanding of programming[48].

Recent studies have started to understand user perception and behaviors with LLM-based programming interfaces. In education, early studies focused on instructors' and students' perceptions of LLM-based interfaces for programming. Computer science students self-reported many potential benefits of using ChatGPT and were less inclined to report potential drawbacks[100]. On the other hand, computer science instructors were significantly concerned about students' widespread usage of ChatGPT[46]. While some instructors went as far as banning ChatGPT altogether, others suggested exposing students to the capabilities and limitations of AI tools, leveraging mistakes in generated code for learning opportunities. Both instructors and students expressed the need to adapt to a new, LLM-era way of teaching and learning[102].

For professionals, challenges and opportunities co-exist with LLM-based programming interfaces. Recent studies found programmers preferred to use Copilot[84] and finished tasks faster with Copilot[62]. Yet, Copilot struggled with more complicated problems, providing buggy or non-reproducible solutions[23]. Professional programmers faced difficulties in understanding and debugging Copilot-generated code, which hinders their task-solving effectiveness[84]. Programmers who trusted AI were prone to write insecure code with AI[63]. For conversational interfaces, despite inputs being in natural languages, users felt that they needed to learn LLM's "syntax"[26, 37].

Our understanding of user perception and behaviors with LLM-based interfaces during (the learning of) computer programming is still very limited. As the field just started exploring this direction, previous studies mostly focused on general user impressions[102], or conducted behavioral tasks on pre-scripted, close-ended tasks[62]. While close-ended settings made it easier to assess objective metrics[7], open-ended contexts open a wider window to understanding users' learning patterns, behaviors, perceptions, and preferences[8]. For example, a recent study observed two modes that professional programmers interact in open-ended tasks with Copilot: acceleration, where the programmer already knows what they want to do next; and exploration, where the programmer uses AI to explore their options[4]. Another study on professionals' prompt engineering shed light on their struggles, challenges, and potential sources of behaviors[101].

Still, we noticed two gaps in previous studies. First, a majority of studies chose professional programmers or computer science instructors/students as participants, while millions of people also use LLM-based interfaces without a CS background for programming tasks. Second, as HCI studies mostly focus on languages that LLMs are known to perform best, e.g. Python or HTML, little is known about user perceptions and behaviors when computational modeling or LRPLs are involved.

2.3 LLM-based Interfaces for Learning Programming and Modeling

While LLMs have shown promising potential in supporting human-AI collaboration in programming, most design studies were preliminary, and LLM-based interfaces for computational modeling remained understudied. For example, the Programmer's Assistant integrated a chat window into an IDE[68]. Beyond simple integrations, GitHub Copilot Chat[1] provided in-context support within code editors, yet its user studies were still preliminary[10]. A similar design was done on XCode without a user study[78]. Another study explored the integration between computational notebooks with LLMs and emphasized the role of the domain (in this case, data science) on LLM-based interface design[55].

LLMs have gained much attention among programming educators, but the design study is insufficient. Recent studies tested LLMs on introductory programming tasks and achieved unsurprisingly high scores[16, 70]. This prospect leads to great concerns among computer science instructors as they observed the widespread usage of ChatGPT among students[46]. Yet, only a few LLM-based

design studies targeted programming learning. Using a Wizard of Oz prototype, a study underscored the importance of supporting students' varied degrees of prior expertise[67]. A design study reported positive short-term performance gains when young, novice programming learners engaged with Codex[42]. Another study also found LLMs' benefits for novice programmers[57]. Both studies found that more experienced programmers tended to benefit more, yet the reason was still unclear.

In this study, we invoke the learning theory of Constructionism[61] to inform our LLM-based system and empirical study design. While Constructionism has no rigid definition, it argues that learning happens most felicitously when learners "consciously engage in constructing a public entity"[61]. In the context of computer programming, it means learning happens naturally through programming computers, as it iteratively externalizes learners' internal understanding of the world in code, and then allows learners to improve their understanding through watching how the code runs[60]. Moreover, it argues that computer programming is not as abstract or formal as it appears; individual programmers' approaches are often concrete and personal, in pluralistic ways[83]. However, the pluralism in thoughts is more difficult to capture by close-ended tasks (such as a problem set) and objective metrics (such as completion rate/time)[8]. As such, constructionist learning studies often prefer open-ended tasks (e.g. making games[40], designing instructional software[32], creating agent-based models in NetLogo[8]) and qualitative studies, as they open windows into the nuances of learners' perceptions and behaviors in more natural and realistic settings.

The Logo programming language and its descendants (e.g. Scratch; Alice; NetLogo) succeeded in supporting multiple ways of knowing and thinking in computing education and scientific research[75], yet to our knowledge, no published studies have explored their synergy with LLMs. Many prominent constructionist design principles could be applied to AI-based interfaces[41] and inspired the design of NetLogo Chat. For instance, "low floor, high ceiling, wide walls" asks learning environments to provide 1) an easy entrance for novices (low floor); 2) the possibility for experts to work on sophisticated projects (high ceiling); 3) the support of a wide range of different explorations (wide walls); 4) the support of many learning paths and styles[66]. We also learned from previous design studies that stress the importance of adaptive scaffolding[14, 72] and support debugging[9] for novices to learn NetLogo. Hence, we contributed to the field one of the first design studies of LLM-based interfaces for learning programming that follow the constructionist tradition.

3 NETLOGO CHAT SYSTEM

NetLogo Chat is an LLM-based system for learning and programming with NetLogo. It comprises two main parts: a web-based interface integrated with Turtle Universe (a version of NetLogo)[13] (See 3.1); and an LLM-based workflow that improves the quality of AI responses and powers the interface (See 3.2). We iteratively designed the system by:

(1) Based on authors' experiences in teaching NetLogo, we created a design prototype based on the constructionist learning theory (see 2.3), with a focus on supporting users iteratively build up their prompts and smaller code snippets before working on entire models. We developed a proof-of-concept system, using prompt engineering techniques to interact with GPT-3.5-turbo-0314.
(2) We internally evaluated the proof-of-concept with a group of NetLogo experts. During this process, we encountered frequent hallucinations with NetLogo (grammatical or conceptual mistakes; inventing keywords that do not exist; etc). For the system to provide guidance, we realized that authoritative sources are necessary for LLMs' performance;
(3) We incorporated the official NetLogo documentation and code examples into the system using prompt engineering techniques (see 3.2), evaluated other LLMs' potential, and then conducted pilot interviews to evaluate the system with three external NetLogo experts invited from NetLogo's mailing lists. The interviews used a protocol similar to the one we formally used (see 4.2), with more flexibility and open-endedness;
(4) Based on the external feedback, we identified the need for supporting troubleshooting, leading to the design decision 3.1.3. We upgraded the underlying LLM to GPT-3.5-turbo-0613, fixed many minor usability issues, and finalized the prototype we used in the empirical study.

3.1 Design Overview

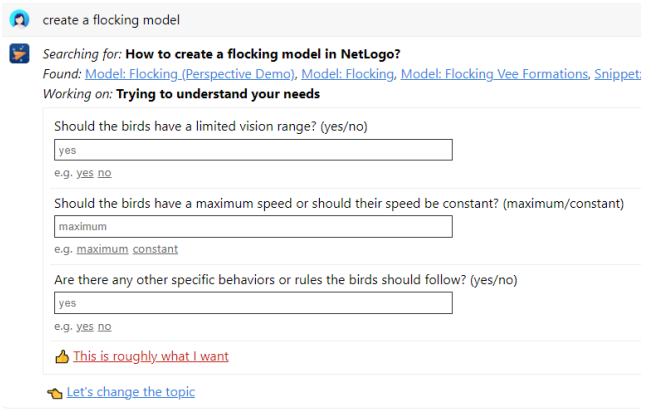

Figure 1: NetLogo Chat asking for details about human's needs.

3.1.1 Enable users to program the computer, rather than being programmed by the computer. Over-reliance on LLM-based interfaces has become a major concern among educators and some learners, where students blindly follow the instructions given by LLMs without attempting to construct their representations of knowledge. Such a scenario is antithetical to the constructionist learning tradition, where Seymour Papert's fear of "computers program children" comes back to life again[60].

Inspired by the Logo language, the design of NetLogo Chat aims to give control back to learners: to suppress LLMs' tendency to give a quick response that often assumes too much about the learner's inclination, we force it to ask clarification questions more often. Fig

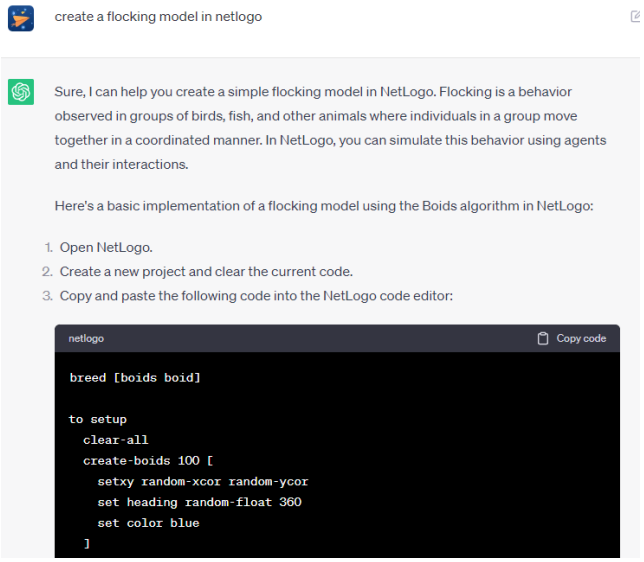

Figure 2: ChatGPT assuming details of human's needs.

1 and Fig 2 provide an exemplary comparison between NetLogo Chat and ChatGPT's reaction to a simple modeling request. Here, ChatGPT immediately assumes details of the user's needs and generates an entire model for the user to copy and paste. Meanwhile, NetLogo Chat attempts to first clarify the user's needs by asking follow-up questions and suggesting exemplary answers. The suggestions in Fig 1 serve as both an inspiration, in case learners get confused about what to write; and a shortcut, in case learners find any suggestions immediately usable.

For this feature to work effectively, it is essential to ask questions with quality. To achieve this, we used a few-shot approach and crafted templates for LLMs to follow. We conducted an informal evaluation of LLM's generated questions during our development process and empirical study. Across the board, the LLM we used was able to generate questions with acceptable quality, similar to the one demonstrated in Fig 1. A future design could embed a larger set of templates and retrieve a few relevant templates when needed.

3.1.2 Invoke Authoritative Sources Whenever Possible. Hallucination is another major concern for LLMs, particularly in an LRPL like NetLogo. For example, the code generated by ChatGPT in Fig 2 contains multiple syntax issues and requires human experts to address them. More powerful LLMs suffer from the same symptoms. We submitted similar sample requests to GPT-4, PaLM2, Anthropic Claude 2, and Falcon-180B: none could produce syntactically correct code for a classical NetLogo model.

Following previous examples in related tasks[38], we integrated NetLogo's official documentation and model examples to help improve LLMs' and human performance. Different from previous studies, we not only provided related examples to LLMs, but also revealed them to users. By doing so, we seek to improve the transparency of LLM's mechanism, foster trust in the LLM-driven system, and provide authoritative guides and examples for users even when LLMs might fail to provide precise support.

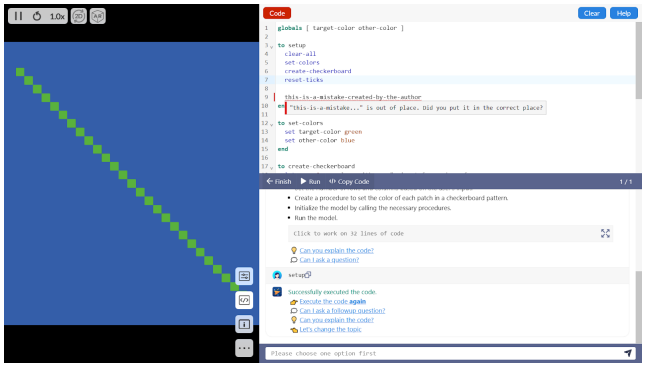

Figure 3: NetLogo Chat's embedded editor for generated code.

3.1.3 Integrate with the IDE and Enhance Troubleshooting. We seek to integrate NetLogo Chat into NetLogo's IDE beyond integrating a conversational assistant parallel to the code editor. To facilitate a constructionist learning experience, the code editor needs to be integrated into the conversational interface, where learners can work with smaller snippets of code with more ease. Thus, the design might lower the threshold for learners to tinker with the code, a key learning process advocated by the constructionist literature [61, 83].

Fig 3 provides a concrete example, where the embedded editor displays a piece of generated code. Instead of having to copy and paste the piece back into the main editor, the user could first see if any syntax issues exist in the code; run the code within a conversation; and ask follow-up questions or raise additional requests, before putting back a working code snippet into their projects.

To further support the user's troubleshooting, in addition to error messages, NetLogo Chat will display extra debugging options for users. Users could choose to look for an explanation, or ask the LLM to attempt fixing the issue on its own, or with the user's ideas. During the process, the system will attempt to find documentation and related code examples to reduce hallucinations. Building on the literature on error messages' impact on learning[5], we also clarified many messages to provide a better context for humans and both LLM-based systems used in the study.

3.2 Technical Implementation

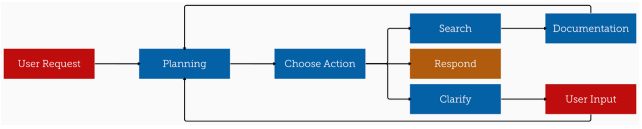

Figure 4: A brief outline for NetLogo Chat's LLM workflow.

Since OpenAI started to provide fine-tuning on GPT-3.5-turbo (the version also used in ChatGPT Free) only after we concluded the main study in July, NetLogo Chat was implemented with prompt engineering techniques. We built our project on ReAct[99], a prompt-based framework that could reduce hallucination, improve human interpretability, and increase the trustworthiness of LLMs. By requiring LLMs to generate an action plan and delegate the action to

a third-party conventional agent (e.g. search for documentation, ask clarification questions, conduct a static syntax check, etc.) before composing the final response, the framework provides a promising pathway to integrate external inputs (e.g. human input, official documentation) into LLM workflows. Fig 4 depicts a rough outline of NetLogo Chat's workflow. Imagine a user requests to "create a predation model":

(1) The LLM is instructed, in the prompt, to first elaborate on the request (planning): "The user intends to create an agent-based biology model related to predation. However, it is unclear what exactly the user wants. We need to ask follow-up questions."
(2) Next, the LLM is instructed to choose an action from the list: Ask clarification question(s); Search for documentation; Write a response; Say sorry. Here, imagine the LLM chooses "Ask clarification question(s)" based on the planning.
(3) Then, the LLM needs to generate some questions based on the request. Because LLMs are trained on real-world data, it is not difficult for them to come up with some ideas. For example, "What species do you want to put in the model?" The LLM is also instructed to provide some examples, e.g. "Wolf", "Sheep".
(4) When the user replies to the questions, the loop restarts from step (1). Since there is sufficient information about the request, the LLM decides to search for information, and also generates keywords for the search, e.g. "Wolf-sheep predation model in NetLogo".
(5) The system conducts a semantic search on a pre-assembled database of NetLogo's official documentation and code examples. The system returns the search result, use it as a new round of input, and restarts from step (1).
(6) With inputs from both the user, who clarified the request; and the database, which supplies the example; the LLM plans again, chooses to write a response, and generates its final response.

In the example, we initiated three requests with the LLM, each with a prompt template that results in a structured response[99] (e.g. any response needs to have a Plan, an Action, and a Parameter). Each request could use a different LLM that works best for the specific request. Using this approach, the system can potentially balance cost, performance, speed, and privacy. For example, a future iteration of NetLogo Chat could leverage a fine-tuned local LLM to probe the user's intentions and search for documentation. Then, with any personal or sensitive information stripped away, the system could forward the compiled request to a powerful online LLM (e.g. GPT-4).

For the empirical study, we chose GPT-3.5-turbo-0613 as NetLogo Chat's LLM backend. First, we expect most participants to be using the free version of ChatGPT, driven by the same LLM. This way, we would have a fair playing field for the empirical study, where both systems will be used. Second, at the time of our study, the response time for GPT-4 was too long to sustain a real-time experience, while we had no access to other NetLogo-capable LLMs' APIs. Although we did observe some remarkable improvement when internally evaluating the system (e.g. ChatGPT has trouble answering questions for lesser-known NetLogo keywords, while NetLogo Chat does not), a more systematic evaluation rubric is needed for future research.

Table 1: Overview of Participant Demographics (n=30)

Gender	Females: 10 (33%); Male: 19 (63%); Non-binary: 1 (3%)
Geography	Africa: 1 (3%); Asia and Oceania: 5 (17%); Europe: 8 (27%); Latin America: 2 (7%); North America: 14 (47%).
Occupation	Academics: 14 (47%); Professionals: 12 (40%); Students: 4 (13%)

4 EMPIRICAL STUDY

4.1 Participants

For the empirical study, we recruited 30 adult participants through NetLogo's official Twitter and mailing lists; and through the Complexity Explorer, a website run by Santa Fe Institute (SFI) to distribute learning resources of agent-based modeling (ABM). The exact breakdown of participants' demographic data can be seen in Table 1. The participant pool largely represented the scientific modeling community in NetLogo's main audience, with a majority of participants coming from STEM disciplines. Many participants were also related to the educator sector. 6 participants (20%) were instructors who teach or are interested in teaching NetLogo in classrooms; 4 (13%) were graduate-level students interested in learning NetLogo, making up a third of the population. Participation in the study was voluntary. All participants signed an online consent form on Qualtrics.

Building on the tradition of understanding the difference between experts and novices[17], we separated the participants into experts and novices using self-reported survey data. To mitigate the effect of inaccurate responses, NetLogo experts in the team, who have been core developers and instructors of NetLogo, watched every video and decided if a participant greatly overestimated or underestimated their capabilities. We considered the participant's discussions with the interviewer, the think-aloud process, and the coding behaviors. A vast majority of users' reports correspond with the experts' judgment. Then, to simplify the analysis, we separated participants (Table 2) by their levels into two main categories: experts, who are either experts in NetLogo or programming in general; and novices. In the study, we denote experts by the prefix E (E01-E17) and novices by N (N01-N13). 13 experts had previous experience with ChatGPT (76%), including programming (65%, n=11). 11 novices (85%) also used ChatGPT before, but much less for programming (38%, n=5).

4.2 Interviews

Our study was conducted in 3 phases:

(1) We pilot interviewed 3 experts invited from NetLogo's online community. Each was asked to comment on LLMs for NetLogo learning, as well as on ChatGPT and an early prototype of NetLogo Chat.

Table 2: Participant Information

ID	Region	Level (NetLogo)	Level (Programming)	Occupation
E01	North America	Expert	Expert	Professional
E02	Asia and Oceania	Expert	Intermediate	Academic
E03	Latin America	Intermediate	Expert	Academic
E04	North America	Expert	Expert	Academic
E05	Europe	Intermediate	Expert	Academic
E06	North America	Intermediate	Intermediate	Academic
E07	Latin America	Intermediate	Intermediate	Professional
E08	Asia and Oceania	Intermediate	Intermediate	Professional
E09	Asia and Oceania	Intermediate	Expert	Professional
E10	North America	Intermediate	Intermediate	Academic
E11	Africa	Intermediate	Expert	Academic
E12	North America	Intermediate	Intermediate	Academic
E13	Europe	Expert	Novice	Academic
E14	Europe	Intermediate	Intermediate	Academic
E15	Asia and Oceania	Expert	Expert	Student
E16	Asia and Oceania	Novice	Expert	Professional
E17	Europe	Intermediate	Expert	Academic
N01	North America	Novice	Novice	Professional
N02	North America	Novice	Novice	Academic
N03	North America	Novice	Novice	Professional
N04	North America	Novice	Intermediate	Student
N05	Europe	Novice	Intermediate	Student
N06	Europe	Intermediate	Novice	Student
N07	North America	Novice	Intermediate	Professional
N08	North America	Novice	Intermediate	Professional
N09	North America	Novice	Novice	Professional
N10	North America	Novice	Intermediate	Professional
N11	Europe	Novice	Intermediate	Academic
N12	Europe	Novice	Novice	Academic
N13	North America	Intermediate	Novice	Professional

(2) We improved the design of NetLogo Chat based on what we learned from the pilot interviews and revised the interview protocol accordingly.
(3) We conducted formal interviews with 27 online participants (30 in total).

Each semi-structured interview lasted between 60-90 minutes and was video recorded. Prior to each formal interview, participants were asked to come up with a short NetLogo task that they were interested in working on. Almost every participant brought forward a modeling task from their career domain or personal interest, e.g. to model "how honeybees decide to regulate the temperature of the hive", or "the spread of conflicting ideas". Only once, when the task scope was too complicated for the session, did we ask the participant to bring another. During any part of the interview process, interviewers generally followed the protocol, asking follow-up questions when needed. Specifically:

(1) We asked baseline questions, e.g., "What do you think are the potential advantages / disadvantages of using LLMs in supporting your learning and programming of NetLogo?" (in 2 separate questions)
(2) We asked the participant to work on their task with the help of ChatGPT. Then, we asked the same baseline questions again, then asked "What do you like or dislike about the interface". Repeat the procedure with NetLogo Chat;
(3) If time permitted, we further asked about their preferences for learning and/or programming with NetLogo and asked which feature they wanted to add/remove from either system. Here, the objective was not to strictly compare the two systems, but to elicit more in-depth discussions over LLM-based interfaces.

Since almost all users have already engaged with ChatGPT, we did not randomize the order of ChatGPT/NetLogo Chat. Also, 3 participants used the paid version (GPT-4) during the task with ChatGPT. While much of the generated data comes from the inevitable comparison between the two systems, we chose not to interpret them as objective comparisons. Instead, the different design principles underpinning the systems presented two objects to think with[60], that our participants drew on during their reflections and discussions of LLM-based programming interfaces.

4.3 Data Analysis

Our interviews resulted in around 40 hours of video data. Around half of our data is behavioral in nature, where participants worked on their tasks and were encouraged to think aloud; the other half is more verbal, where participants answered questions. As such, each interview was not only transcribed verbatim, but also watched by a researcher to create observational notes. The two streams were then combined into a single archive for analysis.

Based on our research questions, we iteratively applied the grounded theory approach[22] to analyze our data. During each step, the research team fully discussed the discrepancies between each researcher and iteratively refined the codebook to improve consistency. The analysis reached theoretical saturation at around 50% of interviews, when additional interviews no longer revealed unexpected major insights for our research questions. Then, we finished the rest of qualitative coding with the finalized codebook (Table 3).

(1) Four researchers open-coded 2 interviews, one from a novice and one from an expert, to summarize the topics mentioned by participants. During this process, researchers coded in different tabs to avoid interference. Three broad themes emerged from this phase: participants' approaches to programming; participants' interactions with AI systems; and their comments on AI systems.
(2) Taking notes of the emerging themes, the first author created a preliminary codebook that categorizes dozens of codes into themes. Each researcher coded another 2 interviews in different tabs. In this phase, we refined the themes into approaches to programming (which also helps to separate experts and novices); perceptions and observed behaviors related to AI systems; and comments on AI systems' abilities.
(3) Based on the coding results, the first author created a formal codebook, with definitions clarified based on the discrepancies between researchers (Table 3). To reduce the unbalanced influence of subjective interpretation, researchers only coded explicit behaviors; or direct comments. To avoid missing insights, researchers were instructed to highlight places where existing codes cannot cover the topics. During the first two weeks, a few codes were created or merged as a result of discussions. We retrospectively revised our coding.

Based on the codebook, the first author iteratively incorporates themes into an outline. To further mitigate individual differences, researchers were asked to include as many codes as possible for each quote or observation.

5 FINDINGS

5.1 Perception: Before and After Interaction

5.1.1 Before Interaction: Positive Expectations. Prior to the tasks, both novices and experts had positive expectations of LLM-based interfaces for NetLogo, with novices holding higher expectations than experts.

Both novices and experts expected LLM-based interfaces to save human time and support human effort, especially compared to other help-seeking activities. With LLMs, human time and energy could be liberated for more high-level tasks (E12 , N03). Educators felt that LLMs could facilitate more efficient teaching, allowing students to "more complicated things with relative ease", spiking "their imagination." (E02) LLMs can also bring emotional benefits by reducing the fear of "bothering the teachers or the experts" (E14) or asking "stupid questions" (N06).

Most participants highlighted AI's potential to help them with NetLogo's syntax. For most participants, NetLogo is not the main programming language they used. Before the advent of ChatGPT, N06 felt that she needed to "recite the words (syntax of NetLogo)". Yet, the need was eliminated when "AI can teach you very quickly". Many experts also needed support, as NetLogo "has very strict syntax rules" (E07) which makes writing more difficult.

Novices, in particular, expected that AI could be helpful for troubleshooting. N08 , for instance, felt that LLMs could help him through the troubleshooting process by describing "what I'm trying to do and get a snippet of code that helps get me past that block". For novices without a background in programming, this future looks promising. N12 is interested in the potential to "make programming more approachable to students".

5.1.2 Before Interaction: Negative Expectations. Almost every participant expressed concerns or reservations about LLM-based interfaces. Yet, the concerns of novices and experts were conspicuously different.

Experts focused on preserving human judgment. E01 believed that AI should not "replace human judgment and ability". Similarly, E06 insisted that "(human) has to do the main thinking and ideas and all of that." E17 felt that humans cannot let AI "take over the main reasoning and emotions, the emotions intervening in the decisions." Many educators were also "concerned about learning" (E13), fearing the tendency to "default to the AI system to come up with the answers instead of working through it ourselves" (E12). Many experts explicitly explained their rationales. For example, E08 was concerned that "if a model points me to a suboptimal direction, I will have no idea, because I haven't considered alternative structure". E15 feared that relying on AI responses might "make your horizon narrow" because she would miss learning opportunities when browsing through the models library. For computational modeling, AI also might lack "in-depth knowledge in a specific field" to create an entire model (E05). As such, E05 would only trust AI to "finish a specific task".

Novices were more optimistic and more concerned with their capabilities of understanding AI's responses or making AI understand them. For example, while N04 thought "one of the hypothetical drawbacks" to LLMs being "confidently incorrect", they added that "people are like this too". On the other hand, N03 feared that she would waste more time with AI if "it didn't understand me, or if I had difficulty expressing". N02 acknowledged that "there is a limitation to not knowing how to code (on how much AI could help)." Without knowledge of NetLogo, N11 felt difficult to spot LLM-generated mistakes.

5.1.3 After Interactions: Different Impacts of Hallucination. All participants encountered AI hallucinations throughout the sessions. While some participants rated NetLogo Chat higher than ChatGPT's

Table 3: An Overview of the Codebook

Code	Definition
Approaches	User's perceptions about their approach to programming tasks, e.g. planning, separating into smaller pieces, or working on it as a whole.
Learning	How users learn NetLogo or programming in general, or think that people should learn.
Coding	How users organize or write their code, or think that people should organize or write.
Help-seeking	How users seek help in general, or think that people should seek help.
Human-AI	User's perception and behaviors related to Human-AI relationship.
Prior	Users' prior experiences with ChatGPT or other AI-based interfaces.
Attitude	Users' attitudes toward AI in general, or specific AI-based systems.
Effort	AI's influence on how much, and what kind of, efforts that humans made or need to make.
Abilities	User's perception related to AI's abilities.
Response	AI's ability to provide desirable responses for humans.
Support	AI's ability to support learning/coding of NetLogo.
Interactivity	AI's ability to facilitate helpful interactions with humans.

Table 4: Novices and Experts' Perceptions on LLM-based Interfaces for NetLogo

	Experts	Novices
Before, Positive	LLMs could save human time and effort, especially in syntax.	LLMs could save human time and effort, especially for syntax, and provide emotional benefits. LLMs could help troubleshooting.
Before, Negative	LLMs could mislead humans to suboptimal directions. LLMs could hinder learning processes. LLMs could only work on smaller tasks.	While LLMs may make mistakes, it is no worse than humans. LLMs may not understand human intentions. LLMs' responses are difficult to understand.
After Interaction	LLMs supported learning or practicing by saving time. Will continue to use LLMs for learning or practicing NetLogo.	LLMs supported learning or practicing by saving time. Will seek alternative learning resources before continuing to use LLMs.

free version, most participants had similar changes in perceptions: experts, in general, reported more benefits from LLMs than novices.

Some participants reported more positively about NetLogo Chat's capabilities. Several experts questioned ChatGPT's training in NetLogo, yet they trusted more in NetLogo Chat, for it incorporates authoritative sources (see 3.1.2). E16 believed that NetLogo Chat "understands your NetLogo syntax" and "the basic aspects of NetLogo". N02 thought NetLogo Chat still had bugs but was "much more informative and precise than ChatGPT." As NetLogo Chat is designed to support troubleshooting (see 3.1.3), E04 thought NetLogo Chat "was able to kind of do some better troubleshooting to a certain extent, for it clarifies error codes".

In both cases, experts understood hallucinations as an inevitable part of human-AI collaboration and reacted with more leniency. When E03 first encountered an incorrect response, he exclaimed: "Very interesting! You're mistaken." E05 felt that LLMs helped him "finish most of the code", though he still needed to "debug and see if the code makes sense logically." As experts did not rely on LLMs to resolve issues but mostly leveraged them as a shortcut, E06 stated that hallucinations were instances "where the programmer needs to use own experience and discretion", as risks would escalate if one extrapolates "what ChatGPT provides you in a wrong manner".

Novices, on the other hand, reported more obstacles and frustration, as they relied more on LLMs for their tasks. N07 emotionally responded to a hallucination that ChatGPT "apparently made that shit up". N01 had difficulties to "fix the bugs that were in it (the generated code)." N08 's session ended up "hitting a dead end", with the frustration leading him to "go consult other resources".

Most novices and experts still thought that LLM-based interfaces supported their learning or practicing by saving time. Even though N03 had "low trust" in ChatGPT, she still felt more confident after collaboration, for it "narrowed down the stuff I have to figure out myself and has made me much faster already." As an educator, N12 felt that LLMs facilitated a constructionist learning experience in which "you're being thrown into the culture and have to learn it on the fly." E13 thought he learned a syntax from ChatGPT that would "save me time in the future" and the learning process was "a lot faster than if I were doing it by hand".

As experts reported more perceived benefits, they predominantly intended to continue using LLM-based interfaces for NetLogo. After

the task, E11 felt confident that "I can write anything I want to write". Yet, many novices, driven by their frustration with LLMs, sought alternative learning resources before considering a return. N04, for instance, had a 180-degree turn: expressing great hope before the tasks, they now inclined to "build more by myself with my own code, without AI." N13 thought that she would prefer to work with "someone who is familiar with the programming language" together with LLMs.

5.2 The Behavioral Gap Between Novices and Experts

5.2.1 Behavioral Gap in Planning and Prompting. While experts' and novices' tasks were similar in terms of complexity, we observed differences between how novices and experts plan out their tasks. Since most participants gradually adapted their prompting styles, we focused on participants' first-round prompts.

Two initial prompting patterns, one emphasizing modeling the entire system and another focusing on smaller, initial aspects of the task, emerged from our interviews. Most novices adopted the first pattern (11/13, 85%), while many experts adopted the second pattern (9/17, 53%). Below, we introduce one vignette for each pattern:

(1) N05 started by asking: "I need to make a model of the bunch of agents who are trying to promote political views to other people (...)". Although he used GPT-4, the returned code still had several syntax errors. N05 then spent the next 20 minutes trying to ask GPT-4 to fix issues without success. He expected to "put the idea into it and we'll run the code", but in the end "it didn't happen."
(2) E07 started by asking ChatGPT to "write code for drawing a rectangle". When GPT-3.5 failed to divide the rectangle further, E07 instantly pivoted to another strategy: "I have the following code that draws a rectangle. I want you to modify it so the rectangle is divided by two". GPT-3.5 still failed, yet it produced working code and did "something close to it".

The second prompting pattern involved remarkable mental efforts to decompose and plan out the task. For example, E07 described his approach as "separate into small, general tasks you want to do." E04 explained that he "just likes to iteratively build (the code)". On the other hand, in the first pattern, many participants attempted to shortcut the efforts by delegating the tasks to AI, as N05 said: "I just want to ask it (ChatGPT) to just directly make a code for this task and that's it."

By the end of the task, most participants had realized the importance of breaking tasks into smaller pieces for coding with AI. Naturally, when an LLM-based interface generated code with mistakes, a participant would be (implicitly) guided to ask smaller follow-up questions. Soon, many of them realized the benefits. N01 thought it would be better if one "works through real small problems first, before getting to more complicated problems." N10 would "start with something really basic." Experts using the first pattern had similar ideas. For example, E12 decided to restart "with something simple and just work with it."

5.2.2 Behavioral Gap in Coding and Debugging. As most participants engaged with an agent-based modeling task that they never worked on, both experts and novices learned some aspects of NetLogo with the help of AI - although, in different ways. Experts usually took a much more measured, prudent, and critical approach during coding and debugging, while novices mostly followed AI's instructions.

Most novices focused on reading AI's explanations and followed AI's instructions during their coding processes. ChatGPT often gives instructions like "You can copy and paste this code into NetLogo and run it". Even without this hint, almost all novices would copy and paste the generated code without much reading. The tendency worried some novices, but they had no choice: "I feel like I'm waiting for someone to tell me the answer, rather than learning how to solve it." (N11)

Experts put more emphasis on the code, often ignoring the explanations provided by AI. During their reading, experts evaluated and often criticized the responses, planning their next steps along the way. Only a few experts tried copying and pasting the code to see if they worked out of the box. Other experts selectively copied and pasted parts of the code into their programs, or wrote their programs with generated code on the side. Even when they copied and pasted the code, experts were more cautious. For example, while E04 decided to "just take this and see what this does", he also realized that AI-generated code would override his ideas and manually edited the code.

All participants inevitably had to debug parts of the generated code. Yet, novices sought support from AI more frequently and often struggled with AI responses. For example, N12 would regularly "copy the code that doesn't make sense and go back to AI to see if it can help me." N09 complained that while ChatGPT gave suggestions, "it obviously requires fiddling around with it." As she had little idea about NetLogo, it became a purely trial-and-error experience. Even when AI did solve some errors, it was challenging for novices to learn from the process. For example, N04 commented that while NetLogo Chat provided an automated process, it was still difficult for him to get the lesson, "since I didn't write it myself."

5.2.3 Behind the Behavioral Gap: The Knowledge Gap. We identified a knowledge gap that may lead to the behavioral gap. When novices realized they needed to spend more effort decomposing the task or vetting AI responses, they found themselves lacking the necessary knowledge. In participants' own words, we summarized the four components of a knowledge gap that novices need to overcome when working with AI.

Novices reported the need for conceptual knowledge of modeling. For example, N07 described his experience as "like being adrift on an ocean. Without a compass, and without a map." With only a basic understanding of agent-based modeling, N11 felt compelled to accept ChatGPT's response as "I don't really know how to interpret some of the output from it." Such feelings correspond with novices' tendency to skim through AI responses. Whereas, some novices asked for help from LLMs with different degrees of success. N04 first asked: "(...) Can you tell me what I will need to do before we begin?" With AI's suggestions, N04 had some more success asking follow-up questions.

Table 5: Novices and Experts' Behaviors During Human-AI Collaboration

	Experts	Novices
Planning & Prompting	Many start by asking LLMs for a smaller aspect of the task. *"NetLogo, I would like to spawn 50 turtles"*	Most start by asking LLMs to work on the entire task. *"I want to use netlogo to help me model how honeybees regulate the temperature in their hive. What should I do?"*
Evaluating	Focus more on the generated code. *"Talks too much. I want the code, not the explanation yet."*	Focus more on the generated instructions. *"I am reading the text a little bit and it spits out a bunch of code. So it did give me steps, which is nice."*
Coding	Most selectively copy and paste code, or write code on their own. *"It'd be that I just take this and see what this does. "*	Most start by copying and pasting LLM-generated code. *"This time it gives me.. two boxes to copy."*
Debugging	Debug themselves, or with help from AI. *"Oh, I didn't ask him to move. That is my problem."*	Debug with (more) help from AI. *"I'm going to ask it the same question, but I'm confused why it said something about patches."*

The unfamiliarity with the basic concepts of NetLogo and/or coding in general further adds to the difficulty in prompting and understanding. After reading a guide suggested by NetLogo Chat, N07 realized that he "probably wouldn't have chosen NetLogo to ever begin with" for his database-related task. Other novices were often confused by NetLogo's terms, even when they were mostly in plain English. N03 was confused about "why (ChatGPT) said something about patches" (note: patches are static agents that form NetLogo's modeling world), and that deepened her reliance on ChatGPT. N10 realized that she "only understand 20% of what I am reading, so I can't vet it myself." When the interviewer asked about adding comments into code, N03 replied that while it might be helpful, she was still missing "the high-level understanding of how it comes together."

Many novices also lack the experience for debugging, leading to more unsuccessful attempts and more frustrations. Participants, in particular novices, were often confused by error messages from NetLogo. N01 acknowledged that "without background knowledge, it is hard to figure out what the bugs are, if (LLM) gives you information that is inaccurate." Without experience in debugging, many novices felt frustrated and helpless as previously reported. On the other hand, E12 noted that his students "might not be comfortable with the idea that debugging is a normal part of the process." E01 believed that "the user needs a little practice in debugging their own code" before working with LLM-based interfaces.

Most novices felt a need to learn to interact with LLMs. After repeated failures, N01 felt that he did not "even know what questions to ask to get it to, because it is not doing the right thing." N06 thought AI would help a lot if she could "learn more about how to use AI." N05 realized that he needed to use the correct keywords, for otherwise it "will never generate a good model." This knowledge is relatively easier to acquire though: while N09 felt that "how to ask questions is very important", she believed that "you learn by actually doing it."

5.3 Needs for Guidance, Personalization, and Integration

5.3.1 "Good" Responses, "Bad" Responses. Participants generally appreciate and expect less technical, clear instructions. Many of them appreciate NetLogo Chat's design decisions that include authoritative sources in responses (see 3.1.2) and ask back clarification questions (see 3.1.1). However, participants' preferences are also highly personal and situational.

For both designs, some participants explicitly went against excessive or unnecessary explanations, particularly when the goal is primarily to accomplish a task at hand. For instance, E09 complained that GPT-4 "talks too much. I want the code, not the explanation yet." E14 complained that while related code samples provided by NetLogo Chat could "contain a lot of good suggestions", she wanted to move them to "another box or an expandable line".

Some participants appreciated and hoped that LLMs could stay on topic and give smaller pieces of information at a time. E01 thought NetLogo Chat would be more helpful if it only attempted to solve a bug "one at a time", for users "always overfill their buffer". Novices, in particular, prefer concrete, step-by-step responses, given their focus on AI-generated instructions. N04 wanted to "test one by one if (LLM) gave me multiple suggestions." Going beyond text responses, N03 hoped that there could be "a visual to help me better understand, or internalize what different elements of the code are", so her learning could move to a higher-level understanding of the code's intention.

For NetLogo Chat, most participants reacted positively to the reference to authoritative sources (see 3.1.2), the usage of NetLogo's language, and the provision of links to sources. E03 believed that "the possibility to go directly from this AI to the documentation" would be helpful for his students. N10 "automatically like (NetLogo Chat's response) better" because it used "NetLogo's kind of turtle and patch language." E12 felt "a little bit more confident in the information I was getting because it seemed to be coming

Table 6: Users' Needs for LLMs: Guidance, Personalization, and Integration

Guidance	Personalization	Integration
Should provide clear, less technical responses, stay on topic, and give smaller pieces of information at a time.	Should provide responses based on users' preferred styles.	Should provide better support for coding chunks and iterative modeling.
Should provide responses based on authoritative sources and in NetLogo's language.	Should provide responses based on the knowledge levels and interests of users.	Should support working on existing modeling code.
Should assume less, clarify more, and stick to user intentions for modeling.	Should support human help-seeking preferences in different ways.	Should support input and output of computational modeling.

from inside of the application." However, sticking too much to authoritative explanations might have a downside. E10 complained that NetLogo Chat gave him "dictionary reference", and "dictionary definitions are not especially helpful."

Many participants, in particular experts, reacted positively when NetLogo Chat assumed less about and stuck more to their intentions (e.g. asking questions back, see 3.1.1). For example, E09 commented that ChatGPT (GPT-4) "assumed what I wanted it to do, whereas this one makes you specify your assumptions." He prefers NetLogo Chat's approach, because "it makes you think about the code more." E12 felt that NetLogo Chat's clarification of intention was akin to "progressively guiding me towards a better prompt." As transparency is a key factor in computational modeling, N11 feared that if "anyone can produce an agent-based model, but without actually understanding all the parameters", hidden assumptions introduced by ChatGPT could be detrimental.

5.3.2 Need for Personalization. In this section, we break down the strong needs of experts and novices for more personalization, besides response styles, into two themes: knowledge levels and help-seeking needs.

Novices, in particular, felt a strong need for LLM-based interfaces to acknowledge their knowledge levels and produce responses accordingly. N07 gave a stringent critique of both systems, feeling both systems were "not useful at all", for both "presumes you know something about NetLogo". N08 felt that "ChatGPT has no idea of how much or how little I know about how to code in NetLogo, or how to code in general." Solving this issue would require more personalized approaches. Coming from an educational background, both N02 and E03 suggested that LLMs should first probe users' knowledge level before providing answers.

Participants gave a variety of suggestions that were sometimes conflicting:

(1) **Some participants prefer a guided walkthrough.** N08 hoped that LLMs could walk him through the process and provide starting points. Both E14 and N03 hoped that LLMs could be used alongside video tutorials, where they could first see a successful example of human-AI collaboration and then ask follow-up questions.
(2) **Some participants prefer contextual recommendations.** N11 hoped that LLMs could show related code examples and provide "two or three other ways that you might look with". E10 suggested that LLMs provide in-context explanations if "you don't remember the definition or explanation of a particular command".
(3) **Some participants hope that LLMs could support help-seeking from humans.** E01 hoped that LLMs could help novices "explain my situation so that I can paste it to the user group", so human experts could intervene more easily when AI fails to unstuck novices. Similarly, E17 suggested that AI could be combined with "peer to peer answers and collaboration".
(4) **Some participants believed that incorrect responses could become a learning opportunity.** E02 was concerned that students might be "exposed to fewer options" with AI, compared with "coding from scratch". E03 feared that a system capable of directly producing solutions might deprive students of the debugging process, where they would have learned." Novices also had similar feelings. After many hallucinated responses, N08 thought that ChatGPT "forces me to learn as opposed to just getting code that's ready to go." To fully transform the moment of mistake into learning opportunities, educators suggest the design not to frame mistakes as failures, but rather "as a learning moment" (E12).

5.3.3 Need for Integration. Compared with ChatGPT in a separate browser window, most participants appreciated the NetLogo Chat interface being an integrated part of the modeling environment. They particularly favor the deep integration in NetLogo Chat's design that goes beyond placing a CA and an IDE side-by-side. We further identified many participants' need for a deeper integration.

Many participants appreciated the integration of a sandbox-like code editor in NetLogo Chat, where they can tinker with smaller, AI-generated code chunks and execute them on the fly (see 3.1.3). N12 "definitely liked this feature of being able to go easily between the code and see what was changed and what was added." N04 appreciated that one can "see the code run" in the NetLogo IDE, which ChatGPT could not do. N13 thought while some code generated by ChatGPT was "so comprehensive", NetLogo Chat was able to break it down and make them "more conducive". Participants also expressed further needs for iterative modeling. E13 hoped that NetLogo Chat could help him "modularize all of my commands" by splitting the code into many smaller, more manageable chunks. E12 asked for a comparison feature between versions of code chunks that could help him "iterative changes quickly".

In addition, participants also hoped that LLMs could help them reflect on longer pieces of (existing) code. N02 and E02 wanted AI to support the combination of multiple, smaller code chunks into a single, coherent code. As such, LLM-based interfaces should be able to work with longer pieces of code. Both N06 and E08 hoped that NetLogo Chat could "look at my code and make suggestions based on my code".

Many participants needed adaptive support for modeling more than just coding. Many requested AI support in building model interfaces that could be used to take in inputs or send out outputs. For example, N06 needed NetLogo for her academic paper, hence plotting became "very important". For educators like E13, while the canvas output was "good for the three-quarters of a project", it hid "the real power of agent-based modeling - tracking the emergent properties of the model, rather than simply making bits run around the screen." During the modeling processes, many interface parts could become necessary or unnecessary depending on situational needs. Integrated LLM-based interfaces need to go beyond a "side chat window" and support various spatial configurations for advanced users to decide on.

6 DISCUSSIONS

Our study first reported, in detail, how novices and experts perceive and use LLM-based interfaces (ChatGPT & NetLogo Chat) differently to support their learning and practice of computational modeling in an open-ended setting. Most participants appreciated the design direction NetLogo Chat is heading toward. However, they also expressed their needs for improved guidance, personalization, and integration which opens up huge design spaces for future improvement.

6.1 Guidance: Bridging the Novice-Expert Gap

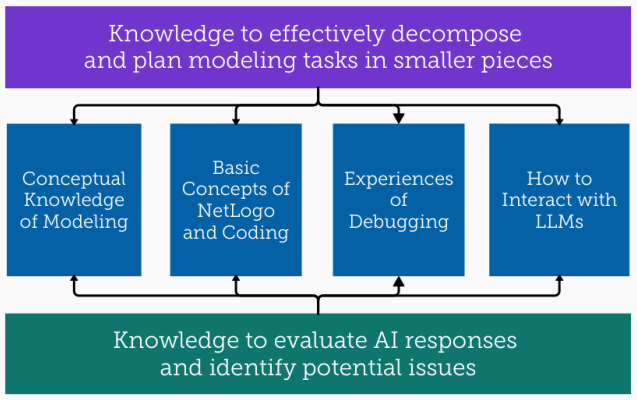

Figure 5: A preliminary theorization of the novice-expert knowledge gap.

For most participants, guidance is what they need most from LLMs in programming. While **hallucinations** from LLMs constantly present a challenge to everyone, with a higher frequency to evaluate and debug AI responses, experts suffered less negative impact than novices. As a result, experts reported higher levels of perceived gains and more optimistic adoption plans than novices. While novices in our study also attempt to evaluate and debug AI responses, they are ill-equipped for these tasks. Without understanding the knowledge gap between experts and novices, it becomes impossible to design effective guidance.

Based on our empirical findings, we theorize the two types of knowledge novices might need when collaborating with AI in computational modeling (Fig 5). First, the knowledge to effectively decompose and plan modeling tasks. Second, the knowledge to evaluate AI responses and identify potential issues. We further identified four components of knowledge that both novices and experts reported to be essential: conceptual knowledge of modeling; basic concepts of NetLogo and coding; experiences of debugging; and how to interact with LLMs. To mitigate the impact of currently inevitable hallucinations of LLMs, it is essential to help novices get over the knowledge gap.

We propose three learning moments where design intervention might work best. **The first moment** is when users plan their next steps. While most novices started by delegating the planning process to AI, most of them eventually planned on their own. Here, we follow the constructionist learning theory for a broader understanding of planning that includes both rigid, formal plans and "softer", ad-hoc exploration of problem spaces[83]. Both planning styles should be recognized as legitimate in learning and supported by the design [83]. With our current design, most novices reported positive feelings when NetLogo Chat attempted to clarify their intentions and produce a plan for their task. Since this phase does not involve any generated code, more support could be provided, as novices may have fewer problems reading and evaluating natural language responses. They may also feel more comfortable asking questions about modeling or programming ideas, relating them to the generated code later, without fearing that they cannot (yet) read or write code. Moreover, LLMs could expand learners' visions by suggesting new ideas, proposing new plans, or taking notes of human ideas. When novices are confused about basic concepts, LLMs could suggest video or textual tutorials and provide Q&A along the way.

The second moment is when users read and evaluate LLM-generated code. Reading and understanding code is one of the most important aspects of computing education[51]. However, novices in our study were neither confident nor equipped for reading code. As a result, they intended to skip the code section. As predicted by the interest development framework[56], the lack of skills (knowledge) and confidence (identity) may mutually enhance each other. Breaking the feedback loop requires designers to scaffold their reading experiences in both directions. By making explanations within code (as comments or tooltips) or visualizing the code structures (e.g. [76]), we might be able to help build novices' connections between code syntax and real-world meanings. To build up learners' confidence, LLMs should deliver code pieces and explanations in adaptive sizes that work for learners. For learners who still could not succeed, the interface should further provide ad-hoc support that helps novices ask follow-up questions, or lead them to appropriate learning resources.

The third moment is when users need to debug their code. Debugging is considered a rich learning opportunity in constructionist learning[39]. However, it is often associated with negative

feelings that both manifested in prior literature[91], as well as our findings. Unfortunately, cognitive science has found that negative moods may further impede debugging performance[43], enlarging the gap between novices and experts. Following the suggestions of educators in our study, we suggest that LLM-based interfaces could frame bugs in a more positive light, while providing a link to a successful human-AI collaborative debugging process for first-time learners. Both novices' and LLMs' debugging processes are often stuck in loops[92, 97]. While such situations are inevitable, some expert participants suggest that LLM-based interfaces could encourage learners to seek help from another human. Help-seeking is recognized as an important part of programming education, yet novices often struggle with it[54]. In such cases, LLM-based interfaces should further help them frame questions for human experts.

6.2 Personalization: Beyond "Correctness" of LLMs

Personalization has been identified as an essential factor for perceived autonomy when users interact with conversational agents[98], for emotional and relational connections[89], and for various educational benefits[6]. Adding to previous literature, we found personalization to be a crucial factor for LLMs to facilitate effective guidance for learning, as participants expect LLMs to recognize their knowledge levels and react accordingly.

While LLMs might have the potential to further the personalization of learning, recent research in LLMs focused on the "objective" capabilities, ignoring the personalized aspect of its evaluation. For example, technical reports of LLMs all reported benchmarks in whether they could produce functionally correct programs (HumanEval)[15]; if they could correctly answer multi-choice questions (MMLU)[33]; or if they could produce the correct answer of grade school mathematical problems (GSM-8K)[20]. While working toward such "correctness" benchmarks is certainly crucial for LLMs to reduce hallucination and produce better responses, it becomes problematic when the definition of "helpfulness" or "harmfulness" is measured with a ubiquitous scale without individual differences [2]. Unfortunately, today's major LLM players seem to have adopted a similar definition.

At least in learning and practice programming, we argue that helpfulness cannot be a singular metric, but instead varies based on many factors. Corroborating with constructionist design principles[66], we identified some potentially important factors, such as knowledge levels and help-seeking preferences, while other factors, such as culture, ethnicity, and gender, could be as important. To support human learning, the full potential of LLMs could only be achieved through the recognition of epistemological pluralism[83]: humans have different approaches toward learning, and technology needs to be tailored to human needs.

Most participants in our study expected or asked for personalization, in the sense that LLMs recognize their knowledge levels and help-seeking needs, yet today's designs are still far from that. While it is virtually impossible to fine-tune thousands of LLM variants, LLMs' role-play capabilities and novel prompt-based workflows (e.g. the one used by NetLogo Chat, or the concept of GPTs very recently released by OpenAI) have shown promising potential. As personalization requires the inevitable and sometimes controversial collection of user data, we suggest a more upfront approach: only collecting data that directly contributes to a more helpful AI (e.g. the knowledge level), only using data for this purpose, and explaining the benefits, risks, and privacy processes at the beginning. Alternatively, designers could also consider flowing the pathway of cognitive modeling, which deduces learners' knowledge levels from known interactions with the system[77]. On the other hand, our understanding of users' perceptions, behaviors, and needs for LLM-based programming interfaces has just begun, and we call on more studies to pursue this direction.

6.3 Integration: LLMs for Computational Modeling

For most participants, integration between LLM-based interfaces and modeling environments goes beyond stitching a chat window into the IDE. While most appreciated NetLogo Chat's design directions, they put forward many needs worth considering in future design. Here, we briefly discuss the two major themes: support for troubleshooting; and support for modeling. For troubleshooting:

(1) **The capability to work on smaller snippets of code, with the capability to execute, explain, and debug code in context.** For both humans and LLMs[34], debugging complicated code is known to be difficult. NetLogo Chat has made the first step in reducing the scope to smaller code chunks. As such, it becomes easier for humans to debug and LLMs to support their debugging processes. Whereas, more work is needed to bring together the code chunks into coherent full programs.
(2) **The capability to leverage authoritative NetLogo documentation in generated responses and for the user's reference.** In debugging contexts, LLMs' tendency to hallucinate becomes more frustrating. By providing users and LLMs with authoritative explanations within the debugging context, NetLogo Chat may reduce the effort for users to seek related information, which is also known to be difficult for novices[24]. More work is needed to explain in a more personalized way: for example, pure novices may need explanations for every basic term.
(3) **The capability to automatically send in contextual information (i.e. code and error messages) for LLM to troubleshoot.** Users generally appreciated NetLogo Chat's design decision to support troubleshooting. However, the convenience came with a potential price: when using NetLogo Chat, users were more likely to ask LLMs for help, which might lead to fewer human attempts and learning opportunities. Further studies are needed to understand this design balance better.

Many participants also asked for features that specifically support their computational modeling tasks, which are known to have different priorities from programming in general[65]. Here, two more capabilities are warranted:

(1) **The capability to assume less, actively probe, and stick to user intentions.** In addition to the potential learning opportunities (see Discussion 1), for participants, hidden assumptions in scientific modeling are particularly harmful.

While users appreciate NetLogo Chat's direction in having LLMs ask questions back, future interfaces should be able to facilitate the conversational build-up of plans and steps, further supporting users to program computers piece-by-piece rather than falling to hidden assumptions made by LLMs.

(2) **The capability to support modeling practices beyond coding.** Building the program is only one step; computational modeling also involves design, data visualization, and validation[88]. For LLM-based interfaces to support modeling practices, future interfaces should go beyond coding to support users' efforts throughout the modeling process.

7 LIMITATIONS AND FUTURE WORK

There are limitations to our study that warrant future work. As a widely used agent-based modeling language, a deeper understanding of user perceptions, behaviors, and needs for LLM-based interfaces around NetLogo may inform us of design choices for other modeling environments. Future work should consider computational modeling or programming environments that might have different priorities. Since the NetLogo language was designed for an audience without a computer science background[82], it becomes more important and meaningful to understand how to design for bridging the novice-expert gap in LLM-based interfaces. However, it is unclear whether our findings and suggestions would sufficiently support novices' and experts' learning and practice of NetLogo. Using a more rigid rubric to distinguish between experts and novices might improve the rigor of our study. A quantitative, controlled study in the future might further (in)validate our findings and suggestions. As such, we plan to work on a new iteration of NetLogo Chat design and empirical study to understand the design implications fully.

Although we aimed to recruit participants representative of NetLogo's global audience, our participant pool was not as representative as we hoped in two key dimensions. First, our participants were mostly professionals, academics, and graduate students. While K-12 teachers and learners are another major audience for NetLogo and agent-based modeling and may have different priorities and preferences[71], only one K-12 teacher was present in the study. More studies are warranted to further the empirical understanding of LLM-based interfaces in education contexts. Second, the demographics of our participants skewed towards North American and European, highly educated, and male. Such a group of participants, recruited voluntarily, might manifest higher than average acceptability toward novel technology, e.g. most of our participants have already engaged with ChatGPT. For future work, researchers need to recruit a more balanced and diverse group of participants, if the goal is for LLM-based programming interfaces to equitably support novices and experts throughout the world.

8 CONCLUSION

As Large language models (LLMs) have the potential to fundamentally change how people learn and practice computational modeling and programming in general, it is crucial that we gain a deeper understanding of users' perceptions, behaviors, and needs in a more naturalistic setting. For this purpose, we designed and developed NetLogo Chat, a novel LLM-based system that supports and integrates with a version of NetLogo IDE. We conducted an interview study with 30 adult participants to understand how they perceived, collaborated with, and asked for LLM-based interfaces for learning and practice of NetLogo. Consistent with previous studies, experts reported more perceived benefits than novices. We found remarkable differences between novices and experts in their perceptions, behaviors, and needs. We identified a knowledge gap that might have contributed to the differences. We proposed design recommendations around participants' main needs: guidance, personalization, and integration. Our findings inform future design of LLM-based programming interfaces, especially for computational modeling.

ACKNOWLEDGMENTS

We would like to express our gratitude to the NetLogo Online community and Complexity Explorer for their help and support. We especially thank the hundreds of NetLogo users who volunteered for the study. We would also like to thank current and former lab members and anonymous youth users of Turtle Universe, who provided valuable feedback and ideas during our design process. Specifically, we want to acknowledge the intellectual contributions of Umit Aslan; Aaron Brandes; Jeremy Baker; Jason Bertsche; Matthew Berland; Sharona Levy; Jacob Kelter; Leif Rasmussen; David Weintrop; and Lexie Zhao. Finally, we appreciate the valuable and actionable feedback from our anonymous CHI reviewers, which significantly strengthened the paper.

REFERENCES

[1] 2024. About GitHub Copilot Chat. https://docs.github.com/en/copilot/github-copilot-chat/about-github-copilot-chat

[2] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. 2022. Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback. https://doi.org/10.48550/arXiv.2204.05862 arXiv:2204.05862 [cs].

[3] Rishabh Balse, Bharath Valaboju, Shreya Singhal, Jayakrishnan Madathil Warriem, and Prajish Prasad. 2023. Investigating the Potential of GPT-3 in Providing Feedback for Programming Assessments. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 292–298. https://doi.org/10.1145/3587102.3588852

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

[5] Brett A Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, et al. 2019. Compiler error messages considered unhelpful: The landscape of text-based programming error message research. *Proceedings of the working group reports on innovation and technology in computer science education* (2019), 177–210.

[6] Matthew L. Bernacki, Meghan J. Greene, and Nikki G. Lobczowski. 2021. A systematic review of research on personalized learning: Personalized by whom, to what, how, and for what purpose (s)? *Educational Psychology Review* 33, 4 (2021), 1675–1715. Publisher: Springer.

[7] Paulo Blikstein. 2011. Using learning analytics to assess students' behavior in open-ended programming tasks. In *Proceedings of the 1st international conference on learning analytics and knowledge*. 110–116.

[8] Paulo Blikstein, Marcelo Worsley, Chris Piech, Mehran Sahami, Steven Cooper, and Daphne Koller. 2014. Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences* 23, 4 (2014), 561–599. Publisher: Taylor & Francis.

[9] Corey Brady, Melissa Gresalfi, Selena Steinberg, and Madison Knowe. 2020. Debugging for Art's Sake: Beginning Programmers' Debugging Activity in an Expressive Coding Context. (June 2020). https://repository.isls.org//handle/1/6319 Publisher: International Society of the Learning Sciences (ISLS).

[10] Christopher Bull and Ahmed Kharrufa. 2023. Generative AI Assistants in Software Development Education: A vision for integrating Generative AI into educational practice, not instinctively defending against it. *IEEE Software* (2023), 1–9. https://doi.org/10.1109/MS.2023.3300574 Conference Name: IEEE Software.

[11] Santi Caballé and Jordi Conesa. 2019. Conversational agents in support for collaborative learning in MOOCs: An analytical review. In *Advances in Intelligent Networking and Collaborative Systems: The 10th International Conference on Intelligent Networking and Collaborative Systems (INCoS-2018)*. Springer, 384–394.

[12] Fuxiang Chen, Fatemeh H. Fard, David Lo, and Timofey Bryksin. 2022. On the transferability of pre-trained language models for low-resource programming languages. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*. 401–412.

[13] John Chen and Uri J. Wilensky. 2021. Turtle Universe. https://turtlesim.com/products/turtle-universe/

[14] John Chen, Lexie Zhao, Horn Michael, and Wilensky Uri. 2023. The Pocketworld Playground: Engaging Online, Out-of-School Learners with Agent-based Programming. In *Proceedings of the ACM Interaction Design and Children (IDC)*.

[15] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, and Greg Brockman. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[16] Zhutian Chen, Chenyang Zhang, Qianwen Wang, Jakob Troidl, Simon Warchol, Johanna Beyer, Nils Gehlenborg, and Hanspeter Pfister. 2023. Beyond Generating Code: Evaluating GPT on a Data Visualization Course. https://doi.org/10.48550/arXiv.2306.02914 arXiv:2306.02914 [cs].

[17] Michelene TH Chi, Paul J Feltovich, and Robert Glaser. 1981. Categorization and representation of physics problems by experts and novices. *Cognitive science* 5, 2 (1981), 121–152.

[18] Douglas Clark, Brian Nelson, Pratim Sengupta, and Cynthia D'Angelo. 2009. Rethinking science learning through digital games and simulations: Genres, examples, and evidence. In *Learning science: Computer games, simulations, and education workshop sponsored by the National Academy of Sciences, Washington, DC*.

[19] Leigh Clark, Nadia Pantidi, Orla Cooney, Philip Doyle, Diego Garaialde, Justin Edwards, Brendan Spillane, Emer Gilmartin, Christine Murad, and Cosmin Munteanu. 2019. What makes a good conversation? Challenges in designing truly conversational agents. In *Proceedings of the 2019 CHI conference on human factors in computing systems*. 1–12.

[20] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. https://doi.org/10.48550/arXiv.2110.14168 arXiv:2110.14168 [cs].

[21] Grant Cooper. 2023. Examining Science Education in ChatGPT: An Exploratory Study of Generative Artificial Intelligence. *Journal of Science Education and Technology* 32, 3 (June 2023), 444–452. https://doi.org/10.1007/s10956-023-10039-y

[22] Juliet M. Corbin and Anselm Strauss. 1990. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative sociology* 13, 1 (1990), 3–21. Publisher: Springer.

[23] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, and Zhen Ming Jack Jiang. 2023. Github copilot ai pair programmer: Asset or liability? *Journal of Systems and Software* 203 (2023), 111734. Publisher: Elsevier.

[24] Brian Dorn, Adam Stankiewicz, and Chris Roggi. 2013. Lost while searching: Difficulties in information seeking among end-user programmers. *Proceedings of the American Society for Information Science and Technology* 50, 1 (2013), 1–10. Publisher: Wiley Online Library.

[25] Tyna Eloundou, Sam Manning, Pamela Mishkin, and Daniel Rock. 2023. Gpts are gpts: An early look at the labor market impact potential of large language models. *arXiv preprint arXiv:2303.10130* (2023).

[26] Alexander J. Fiannaca, Chinmay Kulkarni, Carrie J. Cai, and Michael Terry. 2023. Programming without a Programming Language: Challenges and Opportunities for Designing Developer Tools for Prompt Programming. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–7.

[27] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. 2022. The robots are coming: Exploring the implications of openai codex on introductory programming. In *Proceedings of the 24th Australasian Computing Education Conference*. 10–19.

[28] Kenneth R. Fleischmann and William A. Wallace. 2009. Ensuring transparency in computational modeling. *Commun. ACM* 52, 3 (March 2009), 131–134. https://doi.org/10.1145/1467247.1467278

[29] Yue Fu, Mingrui Zhang, Lynn K. Nguyen, Yifan Lin, Rebecca Michelson, Tala June Tayebi, and Alexis Hiniker. 2023. Self-Talk with Superhero Zip: Supporting Children's Socioemotional Learning with Conversational Agents. In *Proceedings of the 22nd Annual ACM Interaction Design and Children Conference*. 173–186.

[30] Katy Ilonka Gero, Zahra Ashktorab, Casey Dugan, Qian Pan, James Johnson, Werner Geyer, Maria Ruiz, Sarah Miller, David R. Millen, Murray Campbell, Sadhana Kumaravel, and Wei Zhang. 2020. Mental Models of AI Agents in a Cooperative Game Setting. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3313831.3376316

[31] Zi Gong, Yinpeng Guo, Pingyi Zhou, Cuiyun Gao, Yasheng Wang, and Zenglin Xu. 2022. MultiCoder: Multi-Programming-Lingual Pre-Training for Low-Resource Code Completion. https://doi.org/10.48550/arXiv.2212.09666 arXiv:2212.09666 [cs].

[32] Idit Harel and Seymour Papert. 1990. Software design as a learning environment. *Interactive learning environments* 1, 1 (1990), 1–32. Publisher: Taylor & Francis.

[33] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring Massive Multitask Language Understanding. https://openreview.net/forum?id=d7KBjmI3GmQ

[34] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. Large Language Models for Software Engineering: A Systematic Literature Review. http://arxiv.org/abs/2308.10620 arXiv:2308.10620 [cs].

[35] Nicole M. Hutchins, Gautam Biswas, Ningyu Zhang, Caitlin Snyder, Ákos Lédeczi, and Miklós Maróti. 2020. Domain-specific modeling languages in computer-based learning environments: A systematic approach to support science learning through computational modeling. *International Journal of Artificial Intelligence in Education* 30 (2020), 537–580. Publisher: Springer.

[36] Yuin Jeong, Juho Lee, and Younah Kang. 2019. Exploring Effects of Conversational Fillers on User Perception of Conversational Agents. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems (CHI EA '19)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3290607.3312913

[37] Ellen Jiang, Edwin Toh, Alejandra Molina, Kristen Olson, Claire Kayacik, Aaron Donsbach, Carrie J. Cai, and Michael Terry. 2022. Discovering the syntax and strategies of natural language programming with generative language models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–19.

[38] Harshit Joshi, José Cambronero Sanchez, Sumit Gulwani, Vu Le, Gust Verbruggen, and Ivan Radiček. 2023. Repair Is Nearly Generation: Multilingual Program Repair with LLMs. *Proceedings of the AAAI Conference on Artificial Intelligence* 37, 4 (June 2023), 5131–5140. https://doi.org/10.1609/aaai.v37i4.25642 Number: 4.

[39] Yasmin Kafai, Gautam Biswas, Nicole Hutchins, Caitlin Snyder, Karen Brennan, Paulina Haduong, Kayla DesPortes, Morgan Fong, Virginia J. Flood, and Oia Walker-van Aalst. 2020. Turning bugs into learning opportunities: understanding debugging processes, perspectives, and pedagogies. (2020). Publisher: International Society of the Learning Sciences (ISLS).

[40] Yasmin B. Kafai and Quinn Burke. 2015. Constructionist Gaming: Understanding the Benefits of Making Games for Learning. *Educational Psychologist* 50, 4 (Oct. 2015), 313–334. https://doi.org/10.1080/00461520.2015.1124022 Publisher: Routledge _eprint: https://doi.org/10.1080/00461520.2015.1124022.

[41] Ken Kahn and Niall Winters. 2021. Constructionism and AI: A history and possible futures. *British Journal of Educational Technology* 52, 3 (2021), 1130–1142. Publisher: Wiley Online Library.

[42] 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. *arXiv preprint arXiv:2302.07427* (2023).

[43] Iftikhar Ahmed Khan, Willem-Paul Brinkman, and Robert M Hierons. 2011. Do moods affect programmers' debug performance? *Cognition, Technology & Work* 13 (2011), 245–258.

[44] Junaed Younus Khan and Gias Uddin. 2022. Automatic code documentation generation using gpt-3. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–6.

[45] Varun Kumar, Leonard Gleyzer, Adar Kahana, Khemraj Shukla, and George Em Karniadakis. 2023. MyCrunchGPT: A chatGPT assisted framework for scientific machine learning. https://doi.org/10.48550/arXiv.2306.15551 arXiv:2306.15551 [physics].

[46] Sam Lau and Philip J. Guo. 2023. From" Ban It Till We Understand It" to" Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot. (2023).

[47] Yunyao Li, Huahai Yang, and Hosagrahar V. Jagadish. 2005. Nalix: an interactive natural language interface for querying xml. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. 900–902.

[48] 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. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 09 (April 2020), 13381–13388. https://doi.org/10.1609/aaai.v34i09.7061 Number: 09.

[49] Erin Chao Ling, Iis Tussyadiah, Aarni Tuomi, Jason Stienmetz, and Athina Ioannou. 2021. Factors influencing users' adoption and use of conversational agents: A systematic review. *Psychology & marketing* 38, 7 (2021), 1031–1051. Publisher: Wiley Online Library.

[50] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. https://doi.org/10.48550/arXiv.2305.01210 arXiv:2305.01210 [cs].

[51] Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister. 2008. Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the fourth international workshop on computing education research*. 101–112.

[52] Stephen MacNeil, Andrew Tran, Dan Mogil, Seth Bernstein, Erin Ross, and Ziheng Huang. 2022. Generating diverse code explanations using the gpt-3 large language model. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 2*. 37–39.

[53] Alexandre Magueresse, Vincent Carles, and Evan Heetderks. 2020. Low-resource Languages: A Review of Past Work and Future Challenges. http://arxiv.org/abs/2006.07264 arXiv:2006.07264 [cs].

[54] Samiha Marwan, Anay Dombe, and Thomas W Price. 2020. Unproductive help-seeking in programming: What it is and how to address it. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. 54–60.

[55] Andrew M. McNutt, Chenglong Wang, Robert A. Deline, and Steven M. Drucker. 2023. On the design of ai-powered code assistants for notebooks. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–16.

[56] Joseph E. Michaelis and David Weintrop. 2022. Interest Development Theory in Computing Education: A Framework and Toolkit for Researchers and Designers. *ACM Transactions on Computing Education (TOCE)* (2022). Publisher: ACM New York, NY.

[57] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2023. In-IDE Generation-based Information Support with a Large Language Model. https://doi.org/10.48550/arXiv.2307.08177 arXiv:2307.08177 [cs].

[58] OpenAI. 2023. GPT-4 Technical Report. https://doi.org/10.48550/arXiv.2303.08774 arXiv:2303.08774 [cs].

[59] Soumen Pal, Manojit Bhattacharya, Sang-Soo Lee, and Chiranjib Chakraborty. 2023. A Domain-Specific Next-Generation Large Language Model (LLM) or ChatGPT is Required for Biomedical Engineering and Research. *Annals of Biomedical Engineering* (2023), 1–4. Publisher: Springer.

[60] Seymour Papert. 1980. Mindstorms: Children, computers, and powerful ideas. (1980). Publisher: Basic Books.

[61] Seymour Papert and Idit Harel. 1991. Situating constructionism. *constructionism* 36, 2 (1991), 1–11.

[62] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. 2023. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. https://doi.org/10.48550/arXiv.2302.06590 arXiv:2302.06590 [cs].

[63] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. 2022. Do Users Write More Insecure Code with AI Assistants? https://doi.org/10.48550/arXiv.2211.03622 arXiv:2211.03622 [cs].

[64] David Price, Ellen Rilofff, Joseph Zachary, and Brandon Harvey. 2000. NaturalJava: A natural language interface for programming in Java. In *Proceedings of the 5th international conference on Intelligent user interfaces*. 207–211.

[65] Zenon W. Pylyshyn. 1978. Computational models and empirical constraints. *Behavioral and Brain Sciences* 1, 1 (March 1978), 91–99. https://doi.org/10.1017/S0140525X00059793 Publisher: Cambridge University Press.

[66] Mitchel Resnick and Brian Silverman. 2005. Some reflections on designing construction kits for kids. In *Proceedings of the 2005 conference on Interaction design and children*. 117–122.

[67] Peter Robe and Sandeep Kaur Kuttal. 2022. Designing PairBuddy—A Conversational Agent for Pair Programming. *ACM Transactions on Computer-Human Interaction* 29, 4 (May 2022), 34:1–34:44. https://doi.org/10.1145/3498326

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

[69] Shruti Sannon, Brett Stoll, Dominic DiFranzo, Malte F. Jung, and Natalya N. Bazarova. 2020. "I just shared your responses": Extending Communication Privacy Management Theory to Interactions with Conversational Agents. *Proceedings of the ACM on Human-Computer Interaction* 4, GROUP (Jan. 2020), 08:1–08:18. https://doi.org/10.1145/3375188

[70] Jaromir Savelka, Arav Agarwal, Marshall An, Chris Bogart, and Majd Sakr. 2023. Thrilled by Your Progress! Large Language Models (GPT-4) No Longer Struggle to Pass Assessments in Higher Education Programming Courses. In *Proceedings of the 2023 ACM Conference on International Computing Education Research V.1*. 78–92. https://doi.org/10.1145/3568813.3600142 arXiv:2306.10073 [cs].

[71] Pratim Sengupta, Amanda Dickes, Amy Voss Farris, Ashlyn Karan, David Martin, and Mason Wright. 2015. Programming in K-12 science classrooms. *Commun. ACM* 58, 11 (Oct. 2015), 33–35. https://doi.org/10.1145/2822517

[72] Pratim Sengupta, John S. Kinnebrew, Satabdi Basu, Gautam Biswas, and Douglas Clark. 2013. Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies* 18, 2 (June 2013), 351–380. https://doi.org/10.1007/s10639-012-9240-x

[73] Vidya Setlur, Sarah E. Battersby, Melanie Tory, Rich Gossweiler, and Angel X. Chang. 2016. Eviza: A natural language interface for visual analysis. In *Proceedings of the 29th annual symposium on user interface software and technology*. 365–377.

[74] Marita Skjuve, Asbjørn Følstad, and Petter Bae Brandtzaeg. 2023. The User Experience of ChatGPT: Findings from a Questionnaire Study of Early Users. In *Proceedings of the 5th International Conference on Conversational User Interfaces*. 1–10.

[75] Cynthia Solomon, Brian Harvey, Ken Kahn, Henry Lieberman, Mark L. Miller, Margaret Minsky, Artemis Papert, and Brian Silverman. 2020. History of logo. *Proceedings of the ACM on Programming Languages* 4, HOPL (2020), 1–66. Publisher: ACM New York, NY, USA.

[76] Juha Sorva, Ville Karavirta, and Lauri Malmi. 2013. A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education (TOCE)* 13, 4 (2013), 1–64.

[77] Ron Sun. 2008. Introduction to computational cognitive modeling. *Cambridge handbook of computational psychology* (2008), 3–19.

[78] Chee Wei Tan, Shangxin Guo, Man Fai Wong, and Ching Nam Hang. 2023. Copilot for Xcode: Exploring AI-Assisted Programming by Prompting Cloud-based Large Language Models. http://arxiv.org/abs/2307.14349 arXiv:2307.14349 [cs].

[79] Artur Tarassow. 2023. The potential of LLMs for coding with low-resource and domain-specific programming languages. http://arxiv.org/abs/2307.13018 arXiv:2307.13018 [cs].

[80] J. C. Thiele, W. Kurth, and V. Grimm. 2011. Agent-and individual-based modeling with NetLogo: Introduction and new NetLogo extensions. *Deutscher Verband Forstlicher Forschungsanstalten, Sektion Forstliche Biometrie und Informatik-22. Tagung* (2011).

[81] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F. Bissyandé. 2023. Is ChatGPT the Ultimate Programming Assistant–How far is it? *arXiv preprint arXiv:2304.11938* (2023).

[82] Seth Tisue and Uri Wilensky. 2004. Netlogo: A simple environment for modeling complexity. In *International conference on complex systems*, Vol. 21. Citeseer, 16–21.

[83] 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. Publisher: University of Chicago Press.

[84] 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.

[85] Thiemo Wambsganss, Tobias Kueng, Matthias Soellner, and Jan Marco Leimeister. 2021. ArgueTutor: An adaptive dialog-based learning system for argumentation skills. In *Proceedings of the 2021 CHI conference on human factors in computing systems*. 1–13.

[86] Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A. Saurous, and Yoon Kim. 2023. Grammar Prompting for Domain-Specific Language Generation with Large Language Models. http://arxiv.org/abs/2305.19234 arXiv:2305.19234 [cs].

[87] Qiaosi Wang, Koustuv Saha, Eric Gregori, David Joyner, and Ashok Goel. 2021. Towards mutual theory of mind in human-ai interaction: How language reflects what students perceive about a virtual teaching assistant. In *Proceedings of the 2021 CHI conference on human factors in computing systems*. 1–14.

[88] David Weintrop, Elham Beheshti, Michael Horn, Kai Orton, Kemi Jona, Laura Trouille, and Uri Wilensky. 2016. Defining Computational Thinking for Mathematics and Science Classrooms. *Journal of Science Education and Technology* 25, 1 (Feb. 2016), 127–147. https://doi.org/10.1007/s10956-015-9581-5

[89] Galit Wellner and Ilya Levin. 2023. Ihde meets Papert: combining postphenomenology and constructionism for a future agenda of philosophy of education in the era of digital technologies. *Learning, Media and Technology* (2023), 1–14. Publisher: Taylor & Francis.

[90] Michel Wermelinger. 2023. Using GitHub Copilot to solve simple programming problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 172–178.

[91] Jacqueline Whalley, Amber Settle, and Andrew Luxton-Reilly. 2021. Novice reflections on debugging. In *Proceedings of the 52nd ACM technical symposium on computer science education*. 73–79.

[92] Jacqueline Whalley, Amber Settle, and Andrew Luxton-Reilly. 2021. Novice Reflections on Debugging. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 73–79. https://doi.org/10.1145/3408877.3432374

[93] Uri Wilensky and William Rand. 2015. *An introduction to agent-based modeling: modeling natural, social, and engineered complex systems with NetLogo.* Mit Press.

[94] Uri J. Wilensky. 1997. NetLogo Wolf Sheep Predation model. http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation

[95] Rainer Winkler, Sebastian Hobert, Antti Salovaara, Matthias Söllner, and Jan Marco Leimeister. 2020. Sara, the lecturer: Improving learning in online education with a scaffolding-based conversational agent. In *Proceedings of the 2020 CHI conference on human factors in computing systems*. 1–14.

[96] Rainer Winkler and Matthias Söllner. 2018. Unleashing the potential of chatbots in education: A state-of-the-art analysis. In *Academy of Management Proceedings*, Vol. 2018. Academy of Management Briarcliff Manor, NY 10510, 15903. Issue: 1.

[97] Xingbo Wu, Nathanaël Cheriere, Cheng Zhang, and Dushyanth Narayanan. 2023. RustGen: An Augmentation Approach for Generating Compilable Rust Code with Large Language Models. (June 2023). https://openreview.net/forum?id=y9A0vJ5vuM

[98] Xi Yang and Marco Aurisicchio. 2021. Designing conversational agents: A self-determination theory approach. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–16.

[99] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2022. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*.

[100] Ramazan Yilmaz and Fatma Gizem Karaoglan Yilmaz. 2023. Augmented intelligence in programming learning: Examining student views on the use of ChatGPT for programming learning. *Computers in Human Behavior: Artificial Humans* 1, 2 (2023), 100005. Publisher: Elsevier.

[101] J. D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny can't prompt: how non-AI experts try (and fail) to design LLM prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–21.

[102] Cynthia Zastudil, Magdalena Rogalska, Christine Kapp, Jennifer Vaughn, and Stephen MacNeil. 2023. Generative AI in Computing Education: Perspectives of Students and Instructors. *arXiv preprint arXiv:2308.04309* (2023).