Who's the Leader? Analyzing Novice Workflows in LLM-Assisted Debugging of Machine Learning Code

Jessica Y. Bo
University of Toronto
jbo@cs.toronto.edu

Majeed Kazemitabaar
University of Toronto
majeed@dgp.toronto.edu

Emma Zhuang
University of Toronto
jianyun.zhuang@mail.utoronto.ca

Ashton Anderson
University of Toronto
ashton@cs.toronto.edu

Abstract

While LLMs are often touted as tools for democratizing specialized knowledge to beginners, their actual effectiveness for improving task performance and learning is still an open question. It is known that novices engage with LLMs differently from experts, with prior studies reporting meta-cognitive pitfalls that affect novices' ability to verify outputs and prompt effectively. We focus on a task domain – machine learning (ML) — that embodies both high complexity and low verifiability to understand the impact of LLM assistance on novices. Provided a buggy ML script and open access to ChatGPT, we conduct a formative study with eight novice ML engineers to understand their reliance on, interactions with, and perceptions of the LLM. We find that user actions can be roughly categorized into *leading the LLM* and *led-by the LLM*, and further investigate how they affect reliance outcomes like over- and under-reliance. These results have implications on novices' cognitive engagement in LLM-assisted tasks and potential negative effects on downstream learning. Lastly, we pose potential augmentations to the novice-LLM interaction paradigm to promote cognitive engagement.

CCS Concepts

• **Human-centered computing → Empirical studies in HCI**.

Keywords

Novice-LLM interactions, LLM-assisted coding, mental models, cognitive engagement, over-reliance

ACM Reference Format:
Jessica Y. Bo, Majeed Kazemitabaar, Emma Zhuang, and Ashton Anderson. 2018. Who's the Leader? Analyzing Novice Workflows in LLM-Assisted Debugging of Machine Learning Code. In *Proceedings of Tools for Thought Workshop at CHI 2025 (Tools for Thought Workshop).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

Tools for Thought Workshop, Yokohama, Japan

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Large Language Models (LLMs) have shown remarkable ability to achieve exemplary performance in a variety of tasks, including coding, writing, and standardized tests [14, 33]. However, their effectiveness in adapting to diverse end-users, ranging in their abilities in the task, still has room for improvement [11, 12, 17, 20]. In the idealized human-LLM collaboration scenario, assistance provided by LLMs should not only help with automating manual work, but also democratizes access to task-specific knowledge for novices and laypeople [29]. Despite these aspirational goals, this partnership breaks down when the users – who are supposed to be in charge – are not actually experts in the tasks. For example, prior research in the LLM-assisted coding domain has documented the meta-cognitive struggles that novices face in prompting and verifying LLMs, leading to 'rabbitholes' of over-reliance [15, 17, 21, 22, 27].

We hypothesize that novice-LLM communication barriers increase in tasks involving highly complex relationships. In domains where the solution is not as easily verifiable as an introductory programming assignment, people must remain cognitively engaged in the task even whilst working alongside powerful LLM partners. To give a concrete example, machine learning (ML) is such a domain where the relationship between input hyperparameters and output metrics is not easily predictable [1, 16]. There is not one direct path towards the optimal solution, but a complex and iterative process that involves constant verification [3, 4, 19]. Expert implementers can apply their learned domain-specific ML diagnostic skills, but novices lack this [2, 24, 30]. This opens up a space for LLMs to lend support to novices [3, 9].

In this paper, we investigate what problems ML novices, with poor mental models and little implementation experience, face in an LLM-assisted workflow. We conduct a formative study with eight participants to understand how they rely on ChatGPT in a challenging ML debugging task. We identify two types of reliance actions (*leading* and being *led-by* the LLM) that result in different reliance outcomes, including different categories of usage errors. Our results suggest that even among novices, prior ML skill levels may correlate with reliance actions and task performance. We pose discussion around the broader consequences of using LLMs to scaffold novices' learning, and how the novice-LLM interaction framework can be augmented to improve novices' mental models and prevent over-reliance on suboptimal LLM responses.

2 Formative Study

We conducted an in-person formative study at an R1-equivalent university in Canada. We recruited eight participants from the undergraduate and graduate student body through Slack channels and word-of-mouth. Participants were asked to self-filter for their level of expertise in ML, using the guideline that they should be *"familiar with implementing machine learning models but not an expert, such as if you have taken an introductory course to machine learning or you are self-taught and have implemented ML projects"*. Participants are told that they would be debugging a faulty machine learning script with the help of ChatGPT, while *thinking aloud* about their process. The study time was 60 minutes and participants were compensated $20 CAD (approximately $14.80 USD at the time of the study) for their participation. All study procedures are approved by the Research Ethics Board at the institution.

2.1 ML Debugging Task

While there are many complex LLM-assisted tasks that we can consider, ML debugging is unique for several reasons. With the growing popularity of machine learning and AI in the mainstream, more and more hobbyists and workers without formal education or training are learning ML implementation [8]. As such, the ubiquity of this growing group of ML novices makes this domain novel and impactful to examine.

Furthermore, ML debugging is also unique from code debugging, where the latter is a more structured process that tends to have a one-to-one mapping of problems and solutions. Code fixes are also generally directly verifiable through compilation and passing unit tests. Conversely, issues in ML code can compound together and interact [19]. The solution is also harder to verify, as the ideal performance of a model on a dataset is usually not known. To successfully solve them, it can involve incorporating both conceptual knowledge and looking at empirical impacts on the performance to make adjustments to the ML code. This requires a robust mental model of ML debugging, including a) diagnosing the cause(s), b) identifying potential solutions, and c) validating the improvement in performance. For a novice without sufficient expertise, this can be very challenging and warrants assistance.

For the experiment, we develop a simple ML training script using the UCI *Adult Income* dataset [6] (see Listing 1 in Appendix A), with the task of predicting if a person is high income ($y = 1$) or not ($y = 0$). The code, written with Python libraries in a Google Colab, includes processing the data, creating the train/test datasets, fitting a `scikit-learn` *Random Forest* (RF) classifier, and evaluating the performance (F1 score and other metrics). We artificially introduce three problems into the code that would cause noticeable performance degradations on the test performance, see Table 1 for their details and solutions. **E1** and **E2** reflect 'code bugs', while **E3** is a result of the distribution of the dataset. The participant is not required to solve the errors using the provided solutions, as we accept any attempts at applying *conceptually sound* methods to *correctly-identified* problems.

2.2 Study Procedure

In the **Pre-Task**, participants filled out a survey to self-rate their experience with machine learning and using LLMs, as well as five *knowledge-based* multiple-choice questions on basic machine learning concepts — this is to estimate their actual abilities for the task, as self-reports may be biased.

Table 1: Summary of the three problems introduced in the machine learning debugging task and their solutions.

ID	Error Description	Example Solution
E1	Overfitting of the Random Forest model due to complexity hyperparameters not being set.	Limit model complexity hyperparameters, such as `max_depth=5`.
E2	Data distribution shift due to the training/test datasets being unshuffled when split.	Set `shuffle=True` in `train_test_split()`.
E3	Data imbalance in the dataset which causes poor performance in the minority class.	Set model parameter `class_weight = balanced`, or resample data with SMOTE.

At the start of the **Main Task**, participants were shown a tutorial demonstrating the *think-aloud* protocol if they weren't already familiar with it. They then received an explanation of the machine learning code and the objective of finding and fixing errors, although they were not told the total number of errors or what metrics they should focus on. They were given free reign to use ChatGPT 4o-mini[1] to complete the task. The participants' workflows and queries were closely monitored and recorded by screen and audio. After the session, their modified code is scored on held out dataset and the number of problems they fixed were counted. Their transcripts with ChatGPT were also saved for further analysis.

In the **Post-Task** survey, participants rated their experience using ChatGPT for the task and any perceived improvements in their ML skills on a 7-point Likert scale. Some of the subjective perception questions are adapted from the UMUX questionnaire [13]. Lastly, participants answered semi-structured interview questions, focusing on topics such as their strategy for using ChatGPT when debugging, the quality of their workflow, and what an ideal ML debugging tool might look like.

3 Results

We describe our findings in terms of task performance, workflows (reliance actions and outcomes), and subjective perceptions.

3.1 Task Performance

Our analysis first examines the participants' ChatGPT-assisted performance in correcting the errors. Everyone identified and solved (or attempted to solve) at least one problem, with four participants solving two, and two participants solving all three. The best possible F1 score on the holdout dataset using an RF model is around 0.32 (as experimentally determined by the research team), and the participants achieved scores ranging 0-0.28, with 0.16 being the performance of the unmodified code. Due to the sensitivity of ML code, it was possible to implement a partial solution but still severely degenerate the performance.

[1]We use the *mini* version as it was available on free accounts and we did not necessarily require a state-of-art model for the study.

Interestingly, we find that participants' holdout F1 scores are correlated to their performance on the initial ML knowledge quiz (but not their self-reported ML experience), with Pearson's $r = .93$, $p < .001$. As we have a small number of participants, we do not assign any strong implications to these results. However, taken together with prior research, this suggests that less skilled users struggle with using LLMs to *accelerate* their implementation. For a task as complex as machine learning, strong mental models of the domain is necessary to appropriately oversee ChatGPT's outputs.

3.2 Reliance Actions and Outcomes

While ChatGPT helped everyone to some extent, we observed that participants' reliance *actions* and *outcomes* had some important differences. We define the relevant terms as follows:

(1) **Reliance Action:** This describes how the participant worked with the LLM – either they were *leading the LLM* through asking specific, and planned-out questions; or they were *led-by the LLM* by asking open-ended questions and following the LLM's suggestions. While subtle, this separates *who* (the user, or the LLM) is the primary driver of debugging. This dichotomy of behaviour is related to the *acceleration* vs *exploration* paradigm defined by Barke et al. and the *shepherding* vs *drifting* behaviour documented by Prather et al..
(2) **Reliance Outcome:** Using the Appropriate Reliance framework [7, 23], we define four categories of outcomes in terms of if the user relied on the LLM and if they made the correct decision – *Rely on Self* (when the LLM is wrong), *Rely on LLM* (when the LLM is correct), *Over-rely* (relying on LLM when it is wrong), and *Under-rely* (not relying on LLM when it is right). The first two demonstrate correct reliance, while the latter two are incorrect reliance.

Reliance Actions were coded for each conversation turn between the participant and ChatGPT, informed by notes and audio transcripts taken during the study session. For example, a *leading* participant independently hypothesized the causes of an error and asked, *"What are the most important hyper parameters for Random Forest?"*; while a *led-by* participant queried, *"What is wrong with this code?"* and verbalized they were unsure of how to start. **Reliance Outcomes** were coded based on what the participant did with ChatGPT's response — either applying the suggestions or moving on. A simplified diagram of the participants workflows is shown as Figure 1. For each participant, the top bar indicates their reliance actions, and the bottom bar indicates the outcomes.

Participants who asked for general guidance (*led-by*) and shifted the workload to ChatGPT generally performed worse than participants who used ChatGPT deliberately to support their debugging plans (*leading*). Four of the five top scorers on the holdout dataset were *leading* ChatGPT in half or more of the conversation turns (**P3**, **P4**, **P7**, and **P8**). Qualitatively, we observe that *under-reliance* occurred more when the participants were controlling the debugging; while *over-reliance* is more frequent when the participant blindly followed the LLM. This is likely explained by less experienced novices lacking relevant knowledge to drive decisions in the task. We also note that amongst the *led-by* participants, some seemed primarily driven by their high trust in ChatGPT – such as **P5** and **P6** who presented as power LLM users.

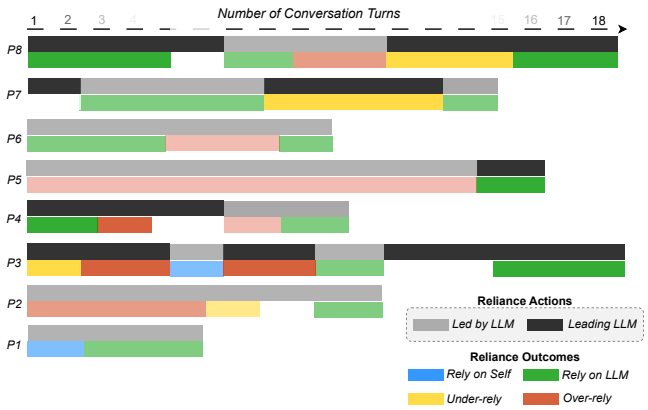

Figure 1: Workflow analysis for each participant with reliance outcomes (top bar) and corresponding LLM actions (bottom bar). To disambiguate between the *leading/led-by* paradigm, the reliance outcome colours when the user *lead* the LLM are more saturated than when *led-by* the LLM.

There are novice-specific factors that degraded interactions with ChatGPT. We label several types of meta-cognitive errors related to how a lack of robust mental models interacts with the compliant nature of LLMs:

- **Leading Query**: Participants who know what they want to explore often inject 'leads' into their queries, which was problematic if the idea was wrong — such as, *"Give me the code for feature standardization"*, where standardization is not necessary for RF models. ChatGPT would provide the code and lead the participant into an unnecessary over-reliance.
- **Filtering**: When asked broad questions, ChatGPT would generate a long list of potential solutions (sometimes more than 10 items). This was very difficult for users to filter through in the time constraint of the task, and sometimes lead to under-reliance.
- **Verification**: Even if the participant identifies the correct actions to take and ChatGPT provides reasonable advice, small differences in the suggestions can drastically impact the outcome. **P8** spent significant time searching for the best RF hyperparameter values without success because ChatGPT's suggested values were not optimal for the dataset. More generally, many participants exhibited difficulty with understanding which ML metrics to pay attention to (training vs testing performance; accuracy vs F1 score) and had a hard time with communicating changes in performance to ChatGPT.

3.3 Subjective Perceptions

Despite the difficulty of the task, overall perceptions of ChatGPT's ability to aid in the task are positive based on the post-task survey. All participants indicated they believed they learned something through using ChatGPT, although their confidence in their ML debugging skills did not improve. Based on interview results, learning mostly referred to low-level and syntactic information – such as the hyperparameters in an RF model – *not* mental models of ML

debugging. Perceived usability and success with prompting is more mixed, and several participants indicated in the post interview that they would take different prompting strategies (like adding context and specificity) if they were to redo the task (**P1**, **P2**, **P4**, **P5**, **P8**).

In terms of the generated content, participants commented that ChatGPT's responses were too broad and lacked context (**P1**, **P2**, **P7**) and the amount of information is overwhelming to filter (**P3**, **P8**) – signaling cognitive overload. However, many also liked that ChatGPT is more selective than Google Search, which helped them find relevant information more efficiently (**P3**, **P6**, **P7**). Some indicated they would like to see suggestions of common errors to debug as a general guideline (**P3**, **P4**). **P6**, who was the only one who faced issues with running the ChatGPT generated code, wanted verification that the syntax can compile. These perception demonstrate the potential for ChatGPT to guide users in complex tasks, but also the some shortcomings of vanilla ChatGPT's interaction strategies for task novices.

4 Discussion

4.1 Key Findings

We present the preliminary findings of novice-LLM workflows in a highly challenging task with high requirements for domain knowledge – machine learning debugging. Participants who were successful in the task tended to have better mental models and were able to *lead* the debugging process, while less experienced participants were more likely to delegate the bulk of the work and be *led-by* ChatGPT. We also identify some interesting modes of interaction failure. When asked open-ended queries, ChatGPT would present a breadth of ideas but not pinpoint the most salient suggestions, leading to cognitive overload and under-reliance. On the other hand, users with incorrect hypotheses about the task asked poorly phrased queries to ChatGPT, leading to 'rabbitholes' of over-reliance on irrelevant advice.

4.2 Implications on Cognition

Designing LLMs to collaborate with novices is challenging in many ways. The interactions between the user's meta-cognition, mental models, and the LLM's behaviour can result in various pitfalls, some of which were documented in the study. While an expert may be able to leverage the LLM using precise prompting and seasoned verification skills to automate the task, we observed that novices struggled with both aspects of task performance and learning. There may even be further tensions between optimizing the debugging task performance and having the novice user meaningfully engage and *lead* the task — as in, even when novices *are* engaged, their faulty mental models of the task can be reinforced through the LLM's sycophancy, resulting in poor learning outcomes. Beyond in-task over-reliance errors, the reduction of high-quality learning in the long term, especially without proper educational scaffolding, is a key concern in novice-LLM interactions.

An important question posed for future research is how the LLM interaction paradigm can be augmented to correctly engage novice users who may have insufficient experience or even incorrect beliefs about the task they are working on. Could LLMs that generate satisficing solutions lead to a culture of novice implementers with poor understanding of how and why the solution was constructed? Will there be downstream effects of improper reliance on LLMs in the educational context, where students bypass foundational learning or are reinforced with bad mental models? We propose some ways to improve novice-LLM interactions in the next section.

4.3 Improving Novice-LLM Interactions

Lastly, we identify how novice-LLM interactions can potentially be augmented to avoid the pitfalls observed in our study. We divide these ideas based on *which side* of the conversation they take effect:

(1) **LLM-side improvement**: ChatGPT's propensity to answer questions complacently and agreeably, even if the user's questions are uninformed, made it a bad guide for novices. *Sycophancy*, a trait that is learned through training by reinforcement learning on human feedback, describes how LLMs would align themselves to reflect the users beliefs, even if the beliefs are incorrect [25]. While we did not observe outlandishly sycophantic statements, ChatGPT was not firm with correcting users' faulty mental models and was overall ineffective at helping novices improve their task knowledge. Adapting LLMs to be more proactive in correcting user's faulty assumptions can help steer novices onto the right track and engage in appropriate reflection to form correct hypotheses in the future.

(2) **User-side improvement**: Users also face challenges as a result of struggling to articulate their intent in their prompts [26, 31]. Some of the participants in the study also expressed wanting to see examples of good queries. Given that we conducted the experiment with mostly Computer Science students, who should have higher technical expertise and AI literacy than the general public, we expect that lay users would need even more support. Providing a tutorial of appropriate usage and effective prompting strategies may help user ask higher quality questions and guide them away from being led into over-reliance [18].

(3) **Bidirectional improvement**: Given that both the user and LLM participate in the conversation, improvements can be made from both sides, iteratively. Since the mental model of a novice can be highly flawed due to lack of information, incorrect information, or incorrect understanding of processes, it would benefit the LLM to customize its response to tackle these misconceptions [10]. *Mutual theory of mind* is a cognitive psychology construct that has been explored in the LLM space, describes the ability of humans to infer what information another person knows. It has been applied as a framework to improving trust and perceptions in human-LLM collaboration tasks [28, 32].

We propose these future directions for the research community as ways to improve interactions between novices and their LLM assistants, with a focus on reducing over-reliance, enhancing cognitive engagement, and improving mental models of the task.

References

[1] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 291–300.

[2] Saleema Amershi, Max Chickering, Steven M Drucker, Bongshin Lee, Patrice Simard, and Jina Suh. 2015. Modeltracker: Redesigning performance analysis tools for machine learning. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. 337–346.

[3] Emily Judith Arteaga Garcia, João Felipe Nicolaci Pimentel, Zixuan Feng, Marco Gerosa, Igor Steinmacher, and Anita Sarma. 2024. How to support ml end-user programmers through a conversational agent. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–12.

[4] Rob Ashmore, Radu Calinescu, and Colin Paterson. 2021. Assuring the machine learning lifecycle: Desiderata, methods, and challenges. *ACM Computing Surveys (CSUR)* 54, 5 (2021), 1–39.

[5] 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.

[6] Barry Becker and Ronny Kohavi. 1996. Adult. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XW20.

[7] Jessica Y Bo, Sophia Wan, and Ashton Anderson. 2024. To Rely or Not to Rely? Evaluating Interventions for Appropriate Reliance on Large Language Models. *arXiv preprint arXiv:2412.15584* (2024).

[8] Carrie J Cai and Philip J Guo. 2019. Software developers learning machine learning: Motivations, hurdles, and desires. In *2019 IEEE symposium on visual languages and human-centric computing (VL/HCC)*. IEEE, 25–34.

[9] Jialun Cao, Meiziniu Li, Ming Wen, and Shing-chi Cheung. 2023. A study on prompt design, advantages and limitations of chatgpt for deep learning program repair. *arXiv preprint arXiv:2304.08191* (2023).

[10] Kartik Chandra, Katherine M Collins, Will Crichton, Tony Chen, Tzu-Mao Li, Adrian Weller, Rachit Nigam, Joshua Tenenbaum, and Jonathan Ragan-Kelley. 2024. Watchat: Explaining perplexing programs by debugging mental models. *arXiv preprint arXiv:2403.05334* (2024).

[11] John Chen, Xi Lu, Yuzhou Du, Michael Rejtig, Ruth Bagley, Mike Horn, and Uri 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*. 1–18.

[12] J Andrés Díaz-Pace, Antonela Tommasel, and Rafael Capilla. 2024. Helping Novice Architects to Make Quality Design Decisions Using an LLM-Based Assistant. In *European Conference on Software Architecture*. Springer, 324–332.

[13] Kraig Finstad. 2010. The usability metric for user experience. *Interacting with computers* 22, 5 (2010), 323–327.

[14] Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. 2013. Bigbench: Towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*. 1197–1208.

[15] Majeed Kazemitabaar, Xinying Hou, Austin Henley, Barbara Jane Ericson, David Weintrop, and Tovi Grossman. 2023. How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*. 1–12.

[16] Sanjay Krishnan and Eugene Wu. 2017. Palm: Machine learning explanations for iterative debugging. In *Proceedings of the 2Nd workshop on human-in-the-loop data analytics*. 1–6.

[17] Francesca Lucchetti, Zixuan Wu, Arjun Guha, Molly Q Feldman, and Carolyn Jane Anderson. 2024. Substance Beats Style: Why Beginning Students Fail to Code with LLMs. *arXiv preprint arXiv:2410.19792* (2024).

[18] Qianou Ma, Weirui Peng, Hua Shen, Kenneth Koedinger, and Tongshuang Wu. 2024. What you say= what you want? Teaching humans to articulate requirements for LLMs. *arXiv preprint arXiv:2409.08775* (2024).

[19] Nadia Nahar, Haoran Zhang, Grace Lewis, Shurui Zhou, and Christian Kästner. 2023. A meta-summary of challenges in building products with ml components–collecting experiences from 4758+ practitioners. In *2023 IEEE/ACM 2nd International Conference on AI Engineering–Software Engineering for AI (CAIN)*. IEEE, 171–183.

[20] Sydney Nguyen, Hannah McLean Babe, Yangtian Zi, Arjun Guha, Carolyn Jane Anderson, and Molly Q Feldman. 2024. How Beginning Programmers and Code LLMs (Mis) read Each Other. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–26.

[21] James Prather, Brent N Reeves, Paul Denny, Brett A Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. *ACM Transactions on Computer-Human Interaction* 31, 1 (2023), 1–31.

[22] James Prather, Brent N Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett A Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. 2024. The widening gap: The benefits and harms of generative ai for novice programmers. In *Proceedings of the 2024 ACM Conference on International Computing Education Research-Volume 1*. 469–486.

[23] Max Schemmer, Niklas Kuehl, Carina Benz, Andrea Bartos, and Gerhard Satzger. 2023. Appropriate reliance on AI advice: Conceptualization and the effect of explanations. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*. 410–422.

[24] Eldon Schoop, Forrest Huang, and Bjoern Hartmann. 2021. Umlaut: Debugging deep learning programs using program structure and model behavior. In *Proceedings of the 2021 CHI conference on human factors in computing systems*. 1–16.

[25] Mrinank Sharma, Meg Tong, Tomasz Korbak, David Duvenaud, Amanda Askell, Samuel R Bowman, Newton Cheng, Esin Durmus, Zac Hatfield-Dodds, Scott R Johnston, et al. 2023. Towards understanding sycophancy in language models. *arXiv preprint arXiv:2310.13548* (2023).

[26] Lev Tankelevitch, Viktor Kewenig, Auste Simkute, Ava Elizabeth Scott, Advait Sarkar, Abigail Sellen, and Sean Rintel. 2024. The metacognitive demands and opportunities of generative AI. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–24.

[27] Jiessie Tie, Bingsheng Yao, Tianshi Li, Syed Ishtiaque Ahmed, Dakuo Wang, and Shurui Zhou. 2024. LLMs are Imperfect, Then What? An Empirical Study on LLM Failures in Software Engineering. *arXiv preprint arXiv:2411.09916* (2024).

[28] 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.

[29] Rose Wang, Qingyang Zhang, Carly Robinson, Susanna Loeb, and Dorottya Demszky. 2024. Bridging the novice-expert gap via models of decision-making: A case study on remediating math mistakes. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 2174–2199.

[30] Qian Yang, Jina Suh, Nan-Chen Chen, and Gonzalo Ramos. 2018. Grounding interactive machine learning tool design in how non-experts actually build models. In *Proceedings of the 2018 designing interactive systems conference*. 573–584.

[31] JD 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.

[32] Shao Zhang, Xihuai Wang, Wenhao Zhang, Yongshan Chen, Landi Gao, Dakuo Wang, Weinan Zhang, Xinbing Wang, and Ying Wen. 2024. Mutual theory of mind in human-ai collaboration: An empirical study with llm-driven ai agents in a real-time shared workspace task. *arXiv preprint arXiv:2409.08811* (2024).

[33] Zibin Zheng, Kaiwen Ning, Yanlin Wang, Jingwen Zhang, Dewu Zheng, Mingxi Ye, and Jiachi Chen. 2023. A survey of large language models for code: Evolution, benchmarking, and future trends. *arXiv preprint arXiv:2311.10372* (2023).

A Machine Learning Debugging Code

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split

# Helper function provided to participants to display the performance metrics.
def get_performance(model, X, y):
    print("\t\t overall accuracy: {:.2f}".format(model.score(X, y)))
    print("-"*60)
    y_pred = model.predict(X)
    print(classification_report(y, y_pred, target_names=["<=50k",">50k"]))
    fig, ax = plt.subplots(figsize=(3,3))
    cm = confusion_matrix(y, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["<=50k",">50k"])
    disp.plot(ax=ax)
    plt.show()

# url is given as the url of the processed UCI Adults dataset.
dataset = pd.read_csv(url)
dataset.head()

# Convert categorical features to numbers
y = dataset['income']
X = dataset.drop(columns=['income'])

categorical_cols = ['workclass', 'marital-status', 'occupation']
for col in categorical_cols:
    X[col] = pd.Categorical(X[col])
    X[col] = X[col].cat.codes

# Split into training and testing datasets (E3).
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, shuffle=False)

# Define and fit Random Forest model (E1, E3)
model = RandomForestClassifier().fit(X_train, y_train)

# Display classification performance on training dataset.
get_performance(model, X_train, y_train)

# Display the classification performance on testing dataset.
get_performance(model, X_test, y_test)
```

Listing 1: Script of the machine learning debugging task, formatted in Google Colab.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009