How to Support ML End-User Programmers through a Conversational Agent

Emily Arteaga Garcia
Oregon State University
Corvallis, OR, USA
arteagae@oregonstate.edu

João Felipe Pimentel
Northern Arizona University
Flagstaff, AZ, USA
joao.pimentel@nau.edu

Zixuan Feng
Oregon State University
Corvallis, OR, USA
fengzi@oregonstate.edu

Marco Gerosa
Northern Arizona University
Flagstaff, AZ, USA
marco.gerosa@nau.edu

Igor Steinmacher
Northern Arizona University
Flagstaff, AZ, USA
igor.steinmacher@nau.edu

Anita Sarma
Oregon State University
Corvallis, OR, USA
anita.sarma@oregonstate.edu

ABSTRACT

Machine Learning (ML) has become an indispensable part of several End-User Programmers' (EUPs) daily work, which increasingly includes software engineering tasks. Machine Learning End-User Programmers (ML-EUPs) without the right background face a steep learning curve and an increased risk of errors and bugs in their ML models. In this work, we designed a conversational agent named "Newton" that can act as a 24/7 expert to support ML-EUPs. The design of Newton was informed by reviewing the existing literature and identifying six challenges that ML-EUPs face and five strategies to help them. We evaluated Newton's design by conducting a Wizard of Oz within-subjects study with 12 ML-EUPs. We found that Newton proved to help support the ML-EUPs, irrespective of whether participants finished their tasks or not. Still, Newton's design, which featured the identified strategies, effectively mitigated the challenges from the literature. Based on participants' interactions with Newton, we proposed six design guidelines for future conversational agents in this domain.

CCS CONCEPTS

• **Computing methodologies → Machine learning**; • **Human-centered computing → Human computer interaction (HCI)**.

KEYWORDS

End-user programming, Conversational Agent, Wizard of Oz

ACM Reference Format:
Emily Arteaga Garcia, João Felipe Pimentel, Zixuan Feng, Marco Gerosa, Igor Steinmacher, and Anita Sarma. 2023. How to Support ML End-User Programmers through a Conversational Agent. In *Proceedings of 46th International Conference on Software Engineering (ICSE 2024).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Machine learning (ML) has become widespread in research and commercial software development. For example, ML drives data-driven user experience and decision-making in both software research and development, as evidenced by the dedicated research topics at ICSE'2024. ML is used to analyze patterns in large datasets and lies at the intersection of computer science, mathematics, and statistics [9, 13, 15, 19, 43]. A broad spectrum of businesses has embraced ML, and its adoption has been growing each year [9]. ML has also caught the interest of business leaders, governments, and the general public [15, 43]. This has resulted in a large class of users who use ML for their work or to improve their careers, whom we term ML End-User Programmers (ML-EUP).

Understanding the workings of ML models requires a thorough comprehension of programming and mathematical concepts such as linear algebra and probability, which can be challenging for ML-EUPs without a strong background [6, 18, 22, 27–30, 32, 34, 46].

Empirical studies have reported challenges ML-EUPs face when developing ML software. For example, Martínez-Fernández et al. [29] reported that not having end-to-end pipeline support can be challenging, especially when deciding which algorithm to use [8, 18, 37]. Similarly, understanding what the ML model has done and why and evaluating whether the prediction is correct is difficult [13]. Even experienced ML-EUPs encounter challenges when working with complex datasets or novel problems and need to invest considerable time and effort to refine and process the input to create and execute the models [4, 13].

Therefore, it is challenging to start using ML, as it involves extensive time and effort from ML-EUPs [6, 44]. In software engineering, ML models are used to detect bugs, perform code repair, and facilitate DevOps, to name a few applications [21, 41]. As more and more software development tasks depend on ML, a larger population of software engineers are using ML in their daily tasks. Incorrect ML models can lead to inefficiencies and errors [4]. While automating parts of the ML pipeline can help, the large variety of ML-EUPs with varying levels of experience makes it difficult to serve the needs of all users (i.e., solutions for advanced users do not match the needs of ML novices) [13].

Researchers recommend learning from an expert while collaborating on a task as a strategy to overcome challenges [13, 16, 18]. However, not all ML-EUPs have access to ML experts, and many ML experts do not have the time to teach ML novices.

To bridge this gap, in this paper, we explore how a conversational agent can serve the role of an expert and scaffold ML-EUPs in their tasks. To role-play as an expert, the agent needs to give advice tailored to the specific task and explain what each ML action is doing

ICSE 2024, April 2024, Lisbon, Portugal

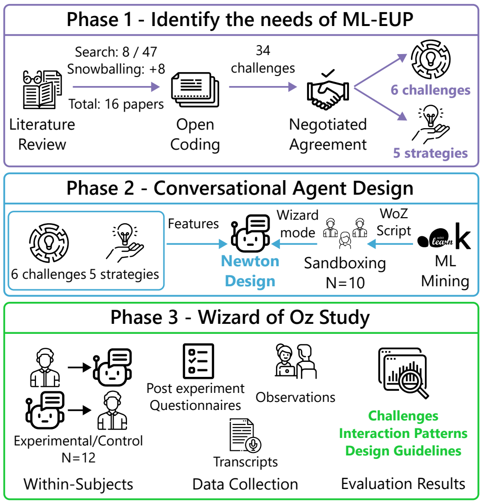


Figure 1: Study Overview

when asked. This would enable individuals without ML expertise or a solid background to receive contextualized guidance as needed.

To create an effective conversational agent, the first key step is to comprehensively understand ML-EUPs' challenges, needs, and interaction patterns. Thus far, to the best of our knowledge, no research has investigated the use of conversational agents to support ML-EUPs in completing ML tasks, and there is a lack of guidance on how to design such conversational agents. In this sense, this paper seeks to answer: *How a conversational agent can support ML-EUPs?*

Toward this goal, we followed a systematic approach to design such a conversational agent to assist ML-EUP (see Figure 1). In Phase 1, we determined the key challenges (and recommended solutions) researchers have identified for ML users. We reviewed existing work discussing the challenges in using ML, which we qualitatively analyzed using open coding and negotiated agreement. In Phase 2, we designed a conversational agent (Newton) to incorporate the recommended solutions as a plugin for Python Jupyter lab. In Phase 3, we conducted a Wizard of Oz (WoZ) lab study. In WoZ, human experts simulate all or parts of the system responses. The WoZ method offers a cost-effective, flexible, and user-centered approach to system design and evaluation when the technology is not evolved enough to provide the desired behavior [12, 14]. The experiment was designed as a counterbalanced, within-subjects study, where participants were asked to solve a classification problem. The classification problems were selected from two Kaggle competitions. In the Experimental group, participants used Newton; in the Control group, they could use any online resources they found relevant. We collected the interactions with Newton's features/strategies that helped participants overcome challenges in building their ML model.

Our findings show that fewer participants faced challenges when using Newton (17% as compared to 58%). We found that features such as decomposing into a set of steps and presenting them as dynamic checklists, generating code snippets, and providing help through predetermined help buttons, on-demand documentation, and chat responses are helpful in successful task completion. However, in some cases, these features backfired when participants got impatient and tinkered around while waiting for the agent's reply, which resulted in out-of-context responses. Based on these findings, we also derived a set of *6 design guidelines* for conversational agents that scaffold software development for ML-EUPs.

2 CHALLENGES AND STRATEGIES IN ADOPTING ML

As a first step to designing the conversational agent, we reviewed the literature to identify the challenges that ML-EUPs face when applying ML and recommended strategies. The process was conducted by the first three researchers. We surveyed IEEExplore and ACM digital libraries using the following set of keywords related to the phenomenon under research: "Challenges in Machine Learning," "Challenges in applying Machine Learning," "Challenges in Machine Learning for Software Engineers," and "Challenges and Strategies for learning ML." Our initial search gave us 47 publications.

We filtered out papers with fewer than eight pages because having more than eight pages is a typical requirement for full papers [45]. Then, based on reviewed abstracts and titles, we filtered out publications that did not explicitly focus on investigating challenges and strategies in learning and using ML. Finally, we selected eight papers and then performed an iteration of backward snowballing as suggested by Wohlin [47], resulting in eight additional papers. Our final list was composed of 16 papers.

To identify the challenges in ML, the same three researchers independently analyzed the papers following the open coding protocol [17]. We held weekly meetings to present our findings and discuss the challenges until reaching an agreement. In the first open coding iteration, we extracted 34 challenges from the papers. After the meetings, we agreed on classifying them into ten categories of challenges. We then filtered out four challenges as these were not focused on challenges faced by ML-EUPs when starting to use ML, namely: ethics, different disciplines, project management, and security. The final categorization considered challenges that could be implemented as a single conversational agent, and consisted of six challenges that are described below.

2.1 Challenges in Using ML

These challenges are further divided into two groups. The first, overarching challenges, include three challenges that occur through different stages of using ML: decision-making, programming, and explainability. The second, pipeline-specific challenges, refer to specific steps of the ML pipeline, including: data wrangling, modeling, and quality evaluation.

C1. Decision-Making is challenging because developing a ML model requires a slew of decisions that extend beyond programming expertise. For instance, users must decide which algorithm to use, what hyperparameters to tune, and how to preprocess/clean data before training ML models [8, 18, 37].

C2. Programming is challenging since ML modeling differs from traditional software development [8]. This is because the performance of each model highly depends on the quality, quantity, and variability of data [3, 4, 8, 18, 37]. For example, feature engineering in ML can be challenging when data is high-dimensional, noisy, and unstructured, which may require particular programming expertise such as deletions, additions, combinations, or mutations [22, 28].

C3. Explainability poses a challenge for large and complicated models, especially for users who lack background knowledge/expertise [6, 44]. ML models are complicated to understand because these are often used as black-boxes, with users only tuning the parameters to optimize the model's performance on a specific task. This black-box nature makes it hard for ML-EUPs to understand why the model did what it did and how to interpret the results [18, 27, 28].

C4. Data Wrangling is one of the first steps in ML pipeline and can be challenging due to the uncertainty in input data preparation. Lacking knowledge of data cleaning and preprocessing steps during data-wrangling can be challenging, complex, and difficult due to the variability of the data [4, 8, 27]. Reviewed studies also agree that a lack of preparation and manipulation before training a model is a major pain point [13, 18, 22, 28, 29, 37].

C5. Modeling can be challenging for end users as it requires understanding how to build the model and involves various decisions (crosscuts **C1**), such as capturing relevant variables and using the right functions [32]. If model development is not performed well, it can cause overfitting giving incorrect or poor predictions [13]. Modeling is also tied with programming (**C2**), since it requires orchestrating different functions together, with the right number of parameters and dependencies across functions [13].

C6. Quality of Evaluation. Understanding the quality of the evaluation requires users to understand how the model was trained, tested, and measured [6, 13, 28, 37]. ML-EUPs may have unrealistic expectations of the model's performance, such as expecting 100% accuracy. It can be challenging for these ML-EUPs to accept the imperfections of ML models, leading to disappointment, distrust, and frustration [18, 32, 44].

2.2 Recommended Strategies

Next, we reviewed the 16 publications to identify their recommended strategies for addressing these challenges. We identified five strategies, three of which are related to guidance and documentation (**S1**, **S2**, **S3**). The other two referred to technical and efficiency optimization (**S4**, **S5**).

S1. Using Checklists is a strategy for guiding users through their decisions (**C1**) while training ML Models. Checklists can also ensure particular ML stages are not skipped or overlooked [13]. For example, a detailed checklist of steps can provide conceptual tutorials and examples beyond conventional API documentation that can serve as a reference book for engineers to troubleshoot issues and optimize performance [6, 8, 37].

S2. 24/7 Expert Availability can mitigate the following challenges: (1) Data wrangling (**C4**): an expert can provide valuable and instructive insights into data extraction and pre-processing [8]; (2) Explainability (**C3**), experts can help interpret the output and guide ML-EUPs to understand the background working of ML models [8, 13]; and (3) Quality of evaluation (**C6**), an expert can help users validate, assess the results, and guide them tuning ML models appropriately to optimize performance [8]. Although, having 24/7 availability to an expert can mitigate these challenges, such access to an expert in real-life is infeasible.

S3. On-hand API Documentation that is provided in the same page as the editor can help in data wrangling (**C4**), decision-making (**C1**), and explainability (**C3**). On-hand documentation reduces the need for context switching and can be more efficient. Documentation can help ML-EUPs understand the data formatting requirements (e.g., a need for continuous and factorized variables) for specific ML models [8] and how to transform data into such formats. Documentation can provide details of different ML models regarding their computational complexity, accuracy, and context of use, which can help ML-EUPs select appropriate functions and algorithms [8]. Finally, documentation can improve explainability by explaining the meaning of different metrics/results (e.g., F1, Recall, ROC). Technical documentation that discusses the mathematical foundation and mathematical solution samples can help users with the appropriate background understand the different models and the approaches to optimize model performance [27].

S4. Code generation can mitigate challenges in data wrangling (**C4**) and programming (**C2**). Code generators can contribute to better programming practices by generating code that adheres to established standards for reproducibility and maintainability (e.g., including comments within the code)[28, 48]. Studies have shown that using code generators can enhance the overall quality of ML projects and contribute to their success [28, 48].

S5. Automated Features that automate parts of the ML pipeline can alleviate challenges related to programming (**C2**) and quality of evaluation (**C6**). Automation can be useful in reducing the amount of programming necessary for data preprocessing or feature engineering. This is achieved by automated default data preprocessing or feature engineering, as noted in L'heureux et al. [28]. Automated features can also play a vital role in identifying significant characteristics from raw data, which can be time-consuming when coding manually, as highlighted in L'heureux et al. [28], Martínez-Fernández et al. [29]. By automatically identifying relevant features from raw data to be used in the model, automated features can help instill confidence in the output and reduce the potential for human-introduced errors or biases [18].

3 CONVERSATIONAL AGENT: NEWTON

In this section, we present the design of a conversational agent that supports the strategies described in Section 2.2. We chose to design a conversation agent, Newton, to support the selected strategies, as it can offer expert advice 24/7, can be adapted to different platforms, and integrates development environments as an extension (i.e., plugin) [12]. Additionally, conversational agents have also been effectively used in the field of education to support novices [25].

Newton was conceived as a plugin for Jupyter Lab [20]—an interactive computational notebook widely used by ML-EUPs [11, 38]. Newton incorporates the selected strategies through a set of features (Sect. 3.1). Newton also provides a wizardmode to handle User-Newton interactions used in our WoZ experiment (Sect. 3.2).

3.1 Newton Features

S1. Dynamic checklist: Newton presents a checklist with the stages of the ML pipeline to guide ML-EUPs. Each step is clickable to allow ML-EUPs to interact with the agent. The checklist also includes an option to `explain the steps`. A dynamic checklist, contextualized to the task, provides a clear view of the critical steps needed [33].

S2. 24/7 Expert availability: Cerezo et al. [10] recommend a conversational agent to improve communication quality by giving ML-EUP the ability to contact an expert at any time. To support this, we incorporated three features in Newton: input text, `help me decide` buttons, and three convenient ways to reply to previous questions (text field, reply-box after a giving answer, and reply button to highlight the question the user wants to reply to).

S3. On-hand API documentation: Newton displays documentation by (1) providing links to the methods webpage, and (2) opening a panel in the notebook with the relevant portion of the documentation. As Mehrpour et al. [31] proposed, on-hand documentation helps ML-EUP implement code faster, learn the design behind code, follow examples, and receive immediate feedback.

S4. Code Generation: As per Kirwan et al. [23] guidelines, Newton provides auto-generated code associated with the different stages of ML contextualized to the task. Newton has access to all the context of the notebook, including existing variables in memory, previously executed code in order of execution, and previous chat messages. All this information is used to generate the contextualized code.

S5. Automated features: L'heureux et al. [28] posit that automated features help reduce the number of manual steps needed by ML-EUPs by quickly providing relevant, common features. We incorporated two automated features in Newton. The first allows autocompleting suggestions when typing a query. The second allows multiple ways of copy-pasting Newton's provided code into a notebook (e.g., in a cell above, in a cell below, clipboard, etc.).

3.2 Wizardmode

To incorporate WoZ support, we designed a wizardmode in which a human expert can access and role-play as a conversational agent. This wizardmode allows the "wizard" to access Newton's inspection features, reply to user queries with different types of messages, and open side panels with custom documentation. To establish communication with the user, the wizardmode leverages Jupyter Lab collaborative mode, which allows different people to connect and work on the same notebook simultaneously.

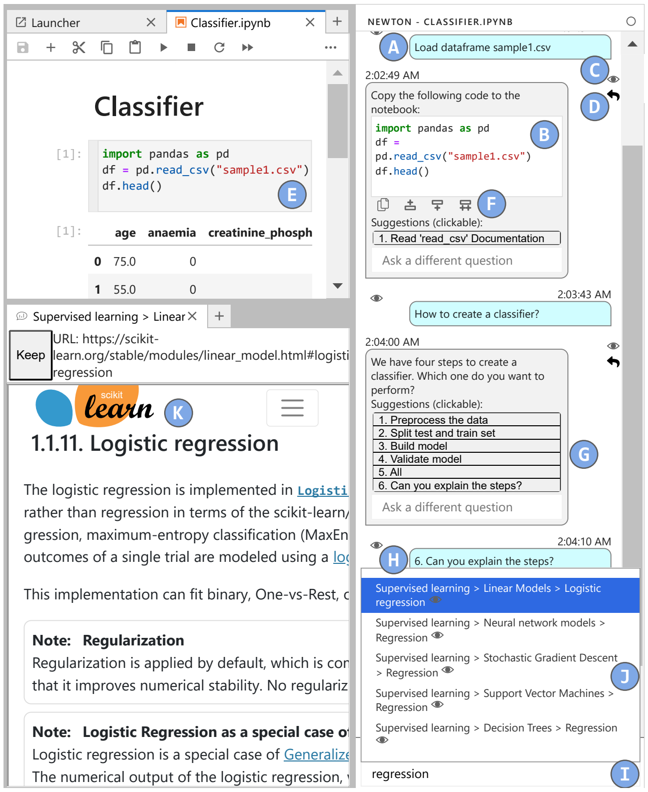
In our experiments, the wizard was the second author of this study, with expertise in Python programming and Machine Learning. The wizard followed a script to keep consistent replies among participants, reduce response times, and be perceived as a conversational agent, as suggested by Klemmer et al. [24], Setlur and Tory [42].

We prepared and refined the (wizard's) script based on the two user study tasks, which were selected from two Kaggle competitions. These competitions were about creating classification models with heart disease and heart mortality data. The outputs of the competitions became the script's core by following the steps various competitors used to tackle the problem. Additionally, we mined scikit-learn documentation related to the different classification algorithms used by the competitors. This documentation contained definitions and examples of how to use certain scikit-learn functions. We tested and refined the script by sandboxing it with ten colleagues from the authors' research labs.

The final script [5] contained a total of 110 messages, without counting the ones that were built into the autocomplete engine as those were generated from scikit-learn documentation.

3.3 Newton Walkthrough

Let us consider a scenario where a ML-EUP, Danny—a professional Python developer—wants to build a ML classification model. Figure 2 presents a snapshot of Danny's interactions with Newton and tags Newton features with letters (e.g., A).



Classifier

```
[1]: import pandas as pd
     df = pd.read_csv("sample1.csv")
     df.head()
```

[1]:

	age	anaemia	creatinine_phosph
0	75.0	0	
1	55.0	0	

Supervised learning > Linear

URL: https://scikit-learn.org/stable/modules/linear_model.html#logisti regression

1.1.11. Logistic regression

The logistic regression is implemented in Logisti
rather than regression in terms of the scikit-learn/
gression, maximum-entropy classification (MaxEn
outcomes of a single trial are modeled using a log

This implementation can fit binary, One-vs-Rest, c

Note: Regularization
Regularization is applied by default, which is cor
that it improves numerical stability. No regulariz

Note: Logistic Regression as a special case of
Logistic regression is a special case of Generalize
The numerical output of the logistic regression, v

Load dataframe sample1.csv

2:02:49 AM

Copy the following code to the notebook:

```
import pandas as pd
df =
pd.read_csv("sample1.csv")
df.head()
```

Suggestions (clickable):
1. Read 'read_csv' Documentation
Ask a different question

2:03:43 AM
How to create a classifier?

2:04:00 AM
We have four steps to create a classifier. Which one do you want to perform?
Suggestions (clickable):
1. Preprocess the data
2. Split test and train set
3. Build model
4. Validate model
5. All
6. Can you explain the steps?
Ask a different question

2:04:10 AM
6. Can you explain the steps?

Supervised learning > Linear Models > Logistic regression
Supervised learning > Neural network models > Regression
Supervised learning > Stochastic Gradient Descent > Regression
Supervised learning > Support Vector Machines > Regression
Supervised learning > Decision Trees > Regression

regression

Figure 2: Newton with tagged features.

» At the beginning of the chat, Danny asked Newton to load a file A. Newton replied to this message with the generated code to load the file B. Most messages in the chat are responses to previous questions. Clicking on the eye button C highlights the question that was answered. Users can respond to an earlier question by clicking the back-arrow button next to the question D, which can create parallel conversations.

» Danny added the code to the notebook E using the "insert cell at the end" button F and executed it. The user can copy the generated code into the notebook by: copying it to the clipboard, and clicking the option to put it directly in a new cell above, in a new cell below, or in a new cell at the end of the notebook.

» After the execution, Danny asked Newton how to create a classifier, and Newton provided a list of steps to build a classifier model G. These steps are clickable and provide paths to different flows of dialog. They also include options such as "explain the steps," which can help users understand what they are doing and build confidence in the output.

» Danny clicked on the "Can you explain the steps?" option, sending it to Newton as a chat message Ⓗ. Newton replied with a follow-up list, asking which step Danny wanted to know about [Not visible in the figure]. Clicking on a button has the same effect as replying to the message that had the button. Users can use the eye button to see the message that was answered. Similarly, Danny could have used the "Ask a different question" text input to reply directly to a message and start a natural language conversation with Newton.

» In the current snapshot, Danny is using the main text field Ⓘ to find out about "regression" algorithms. He uses Newton's autocomplete Ⓙ to see a list of autocomplete and the eye buttons in the "Logistic regression" autocomplete option to load the documentation in the browser Ⓚ. The auto-complete feature gives suggestions while the user is typing in the text field. This feature can be turned on and off based on user preference. Similarly, users can close and open multiple documentation panels at anytime. Besides presenting documentation through autocomplete options, Newton can send messages with links to open documentation panels inside the notebook.

» Danny continues to interact with Newton to build the model.

4 NEWTON EVALUATION

We evaluated Newton through a counterbalanced, within-subjects study, where participants were asked to solve a classification problem. The following evaluation questions guided our study: **Q1**. How do ML-EUPs perceive the challenges when performing a ML task? **Q2**. How do ML-EUPs interact with a conversational agent to solve a ML task? **Q3**. What common patterns emerge when ML-EUPs perform ML tasks?

4.1 Method

Recruitment. We recruited computer science students via email using a list from the university, recruiting from CS classes, and snowballing sampling. Interested participants answered a survey about their self-perceived confidence in Python (since Newton is a Jupyter Lab plugin), Machine Learning, and general programming. A total of 48 people answered the questionnaire.

Participants. From the responses, we selected 12 participants with medium to high confidence in Python and Programming and very low to medium confidence in ML.

Table 1: Demographics of Participants

ID	Gender	Education *	Preferred Language	Prog.	Confidence Python	ML	Tasks **
P1	Woman	MSc [IP]	Python, JS, C++	High	High	Low	🟩C 🟢E
P2	Man	MSc [IP]	C, CPP, Python, C#	High	High	Low	🟢E 🟩C
P3	Man	MSc [IP]	Python, C++, TS	Very High	Very High	Medium	🟢C 🟩E
P4	Woman	MSc [IP]	Python, Java, C++, R	Medium	High	Medium	🟢E 🟥C
P5	Man	MSc [IP]	Python, Java	Medium	High	Low	🔴C 🟥E
P6	Man	MSc [IP]	Python	Medium	High	Medium	🟥E 🔴C
P7	Man	MSc [IP]	Java, C, Python	High	High	Medium	🟥C 🟢E
P8	Man	MSc [IP]	Java, Python, C	Very High	High	Low	🔴C 🟥E
P9	Man	Late PhD	R	Medium	Medium	Medium	🟥E 🔴C
P10	Man	Bachelor	JS, Python	High	High	Medium	🟩C 🟢E
P11	Woman	MSc [IP]	Python	High	High	Medium	🔴E 🟥C
P12	Woman	MSc [IP]	Angular, Python	Medium	Medium	Low	🟥E 🔴C

* [IP] indicates "in progress"
** The letters indicate the use of Newton (E: experiment with Newton, C: control without it), the colors indicate the completion (green: success), the shapes indicate the dataset used in the task, and the tasks are in order.

Table 1 summarizes our participants' demographics and the experiment task order and completion. All participants reported having a CS background. Ten were Master's students, one was a Ph.D. candidate, and one was a professional with a bachelor's in CS. As a token of appreciation, students received a $20 gift card, while the professional received compensation of $50 in gift cards.

Study Protocol. Once the participants were selected, we emailed the informed consent document. The studies were conducted remotely and followed the university IRB protocol. The experiment sessions were recorded with the participant's consent and lasted around 70 minutes each.

The sessions consisted of two classification tasks to be performed with and without Newton in a within-subjects design, a questionnaire after each task, and a post-study questionnaire. We defined the classification problems based on two Kaggle competition scripts (see section 3.2). Moreover, we made small non-breaking changes to the datasets to ensure both tasks were equivalent, had the same complexity, and could be completed within 25 minutes. For instance, the categorical columns of the heart mortality dataset were originally encoded as numeric columns with `0` and `1`. We changed these values to `N` and `Y` to match the notation in the heart disease dataset, which requires an explicit encoding step.

We counterbalanced the tasks, as we show in the tasks column in Table 1. Half of the participants started the experiment with the heart mortality dataset (represented as a circle), and the other half started with the heart disease dataset (represented as a square). This counterbalancing also considered the division of control and experiment tasks, represented by the letters C (control—without Newton) and E (experiment—with Newton) in the table. After each task, participants answered questions regarding challenges they faced in completing the task, about the task, and interactions with Newton (in the Experimental condition).

Each task was time-boxed to 25 minutes to allow participants to complete both treatments; in both tasks, participants were asked to think aloud. The Control participants could use any online tools or help to complete the task. For the Experimental task, we introduced Newton to the participants. To familiarize them with Newton, we showed them its different features and let them practice with a warm-up task (e.g., asking Newton to plot a normal distribution). After completing the warm-up, the participants started the task. Participants were asked to only use Newton.

Analysis. To analyze the results, three authors qualitatively analyzed (open coding) Newton's log messages and identified five categories of user-agent interactions: (a) Newton Hint: the features in which Newton displays a hint (i.e., auto-complete messages, and documentation panels), (b) Enacted Suggestion: interactions in which the participant clicked on a suggestion given by Newton, (c) Newton Reply: Newton messages in the chat, including the ones that contain text, options, forms, and code suggestions, (d) User Input, and (e) Submission of form elements by participants.

Using these categories, we analyzed the pattern of user-agent interactions to understand how ML-EUPs would interact with a conversational agent when building a ML model. The audio files of the study sessions were transcribed by the first and third authors and analyzed using an inductive, open coding process. First, we assigned a code to the different patterns the participants applied during the study (e.g., how the participants interacted with Newton, did they use one feature more than others?). These were then merged or split as necessary to denote descriptive interaction types.

Next, we analyzed the different answers to each task. We grouped similar responses to identify more in-depth interactions between the participants and Newton. We also used the questionnaires to verify if the challenges were reduced while using Newton.

4.2 Results

The study's primary goal was not task completion but rather to observe how participants interacted with the tasks and how Newton's features helped participants in their tasks. Participants were asked to rate the tasks on a Likert scale from very bad (1) to very good (5); All participants rated the tasks as 3 or above, indicating that they generally thought the tasks were good, despite some participants not being able to finish them.

Participants who completed the task without Newton also completed the task with Newton: P1, P2, P3, P10. Two participants could not finish the task in the Control condition, but could do so with Newton (P4, P7). Finally, six participants could not complete the task in either condition (P5, P6, P8, P9, P11, P12).

In the following section, we present participants' perceptions of challenges across both conditions (with and without Newton), categorized based on the challenges identified in Section 2.1. Next, we show which of Newton's features were useful in mitigating those challenges, from which we derive a set of design guidelines (**DG**). Finally, we describe some common patterns among the participants when performing the tasks.

4.2.1 Q1. How do ML-EUPs perceive the challenges? Figure 3 presents the post-task questionnaire responses about participants' perception of challenges (with and without Newton).

The results of the Control condition show that, except for Programming, more participants in the Control condition found the different aspects of ML modeling challenging. For Programming, participants were split with 33% on either side (challenging vs. not challenging). This could be because participants were confident in programming in Python (see Table 1). This is in line with existing studies [4, 8, 27] that shows ML-EUPs need better support to help them face challenges related to Data Wrangling (reported as very/strongly challenging by 42% of the participants), Decision Making (50%), Explainability (42%), and Quality of Evaluation (50%).

In contrast, when using Newton, very few participants' perceived the ML steps as challenging. Interestingly, the fact that people did not finish the task did not impact these results. This suggests that incorporating the strategies identified in Section 2.2 in a conversational agent helps reduce the perception of challenges by ML-EUPs.

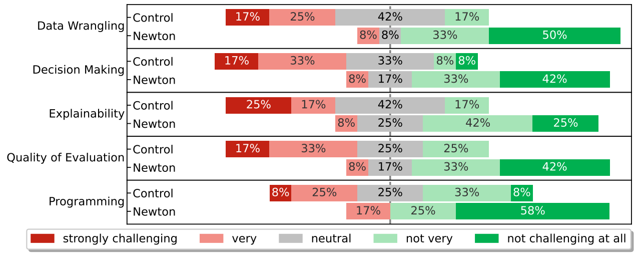


Figure 3: Challenges perception

In addition to filling out the Likert scale questions about the challenges, participants could also report other challenges they experienced through open-text responses.

Four participants indicated experiencing other challenges during the task without Newton, which we grouped into two categories.

Feeling overwhelmed: This challenge corresponds to users being overwhelmed with the amount of information that is available online and the difficulty in finding the right resource to build the model. P7, P9, and P12 experienced this challenge. P12 mentioned: "*. . . too consuming to search for data and understand stuff since too many options.*" P9 experienced similar issues: "*infinite recursive googling for syntax or function using [model parameter].*"

Feeling inadequate: P1 and P5 reported potential issues with self-confidence while developing the model. For example, P1 said: "*I would also add that I felt low confidence. I wasn't super sure what I was doing but I tried to fill my knowledge gaps by looking up tutorials.*" Similarly, P5, who stayed on the same step (loading data) and after trying different methods for a long time, expressed their frustration: "*I had an issue in loading the dataset. I don't know why!.*"

Only one participant reported an additional challenge (P1) for the Experimental condition (with Newton). They mentioned their lack of self-confidence when working on their task "*I think the only other challenge was again low self-confidence. Newton helped me figure out the series of steps I should take to build the model, but I was still unsure of how to correctly interpret the results . . . .*"

4.2.2 Q2. How do ML-EUPs interact with the different features of Newton? Here we analyze how participants interacted with the different Newton features, which serves as an evaluation of the strategies identified from the literature as discussed in Section 2.2. Figure 4 presents a visual overview of the different interactions participants had with Newton. We will use this figure to guide our analysis of participants' interaction patterns.

We first discuss the features that incorporate strategies related to guidance (**S1**, **S2**, **S3**). Then, we discuss the ones that refer to technical and efficiency optimization (**S4**, **S5**).

S1 *- Checklist.* All participants trusted Newton at some point. P1 trusted completely and completed the task without spending effort trying to figure out the next steps. As P1 indicated, "*the ability to see an overview of the steps and keep clicking continue were helpful.*" P1 began the task by typing a query (orange dot in Figure 4) asking "How to perform classification", Newton responded by giving a list of steps (checklist) explaining the process of building a classification model. P1 followed all the suggestions, leading to task completion (green dot in Figure 4). This shows that dynamic checklists can help participants overcome decision-making challenges (**C1**).

However, some participants did not use Newton's suggestions at first. Instead, they typed their own code, refined their queries by asking Newton to try to get different answers, or tried their own steps. For instance, P4 began by asking queries (orange dots in Figure 4), but rejected the suggested steps. Only after they got errors and could not continue with the task, they started to follow Newton's recommendations (region Figure 4:r4b) which helped them to complete the task. On the contrary, P5 did not heed Newton's recommendations and skipped important steps by typing new queries (Figure 4:r5b). This led P5 to errors and an unfinished task.

We realized that the agent (wizard) needs to reiterate prior steps if a user gets stuck in a step or faces an error. For instance, Newton

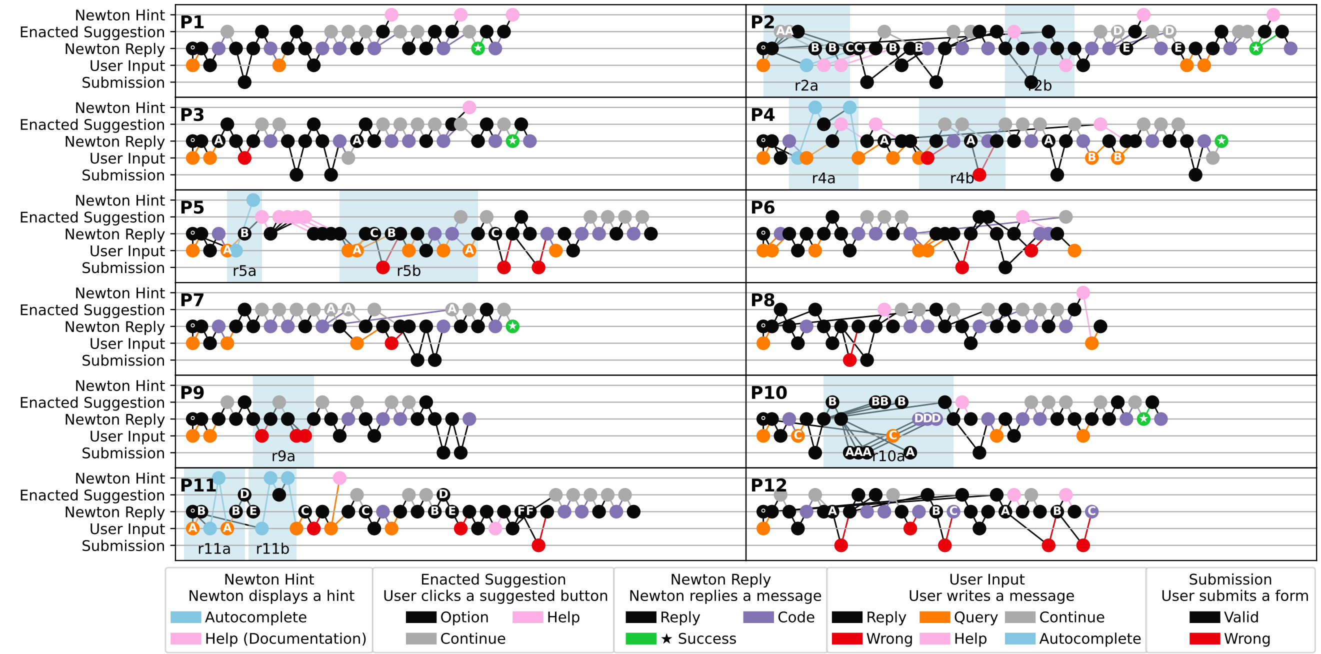


Figure 4: Participants' interactions with Newton. Each participant interaction is represented as a dot in a specific color (e.g., orange: creates a query; pink: help). The vertical axis in each block shows different categories of user-agent interactions (as presented in Section 4.1). The horizontal axis indicates the order of interactions between users and Newton. The lines between each dot indicate the interaction between the participants and Newton. Capital letters inside the dots signify a step that is repeated. (P10 repeated steps denoted by 'A' consecutively). For each scenario, we mark in the graph whether the participant successfully finished the task (P7 completed successfully, but not P9).

helped P6 to fix an execution error related to data wrangling at the end of the split data step, but did not say that the participant had to go through the previous checklist steps before proceeding to the next ones. The participant kept trying to proceed with the execution using outdated values, which led to more errors.

DG 1: Provide insights into what is currently needed when performing a task. A conversational agent should guide the users through the task, giving information on what happened and what is coming next, and not rely on a dynamic checklist alone.

S2 *24/7 Expert Availability.* Newton provides contextualized help to participants based on the step in the task that they were performing or having difficulty with. P5 described Newton as "*an online chatbot which helps us with coding and documentation info to clear the doubts*," suggesting that such help is valuable to get "*unstuck*."

Participants asked for contextualized help in one of two ways. Some participants (P2, P4, P5, P6, P8, P10, P12) used help buttons, such as "help me decide." Others (P2, P11) preferred open-ended text to ask for help or ask for additional information about how to perform the steps (e.g., "How can I know that?"). For example, during the data wrangling step (**C4**), P2 clicked on "why is encoding important" suggestion (Figure 4:r2b). Only after understanding the need for data encoding (changing categorical data to numerical) by reviewing Newton's response, P2 proceeded to complete this step. After several completed steps, P2 then asked Newton "can you help me out with some suggestions?" (Figure 4:r2b) referring to which columns could be classified. Newton indicated that all the columns in the dataset could work as a classifier output. The participant then selected the column that was given for this task and kept following Newton's suggestions until completing the task.

On the other hand, some participants, despite having the opportunity to ask Newton questions, eschewed doing so, and proceeded to execute steps on their own. For instance, among the data-wrangling steps, participants were expected to *check* for invalid zero values and remove them if they exist (they do not exist for the user study tasks). P12 decided to remove the zero values without checking for validity and asked Newton for the code to do so, which Newton provided. The participant executed this code, which made it impossible to correctly complete the classification task, since this operation removed valid categories from the dataset.

When participants asked Newton how to perform a ML task, Newton, serving as an expert, reminded participants about the required steps. For example, P10 asked Newton to perform "data scaling" before encoding categorical values. Newton gave P10 the option to either proceed with the scaling for numerical columns or encode categorical ones and scale all at once. P10 chose the latter option and was able to complete the task.

DG 2: Evaluate the output of current steps and remind users of missed steps based on the context of the workflow.

S3 *On-hand API Documentation.* Newton provides relevant documentation about ML libraries for the code it generates; taking the user to a specific method or function call. P1 and P2 were the most motivated to read about ML functions. P1 opened the documentation panel three times, first to read about `StandardScaler` after splitting data into testing and training. The second time they read about "Linear Regression," and the third time – after completing the task – they wanted to understand more about the `classification_report` function. Similarly, P2 opened the documentation panel twice, the first time to get insights from the `train_test_split` function, and the second time to look at the `predict` function. In the post-task form, P2 indicated that they liked the "*guidance when I am stuck and providing documentation of all the things that were used in the suggested code snippet.*" Other participants (P3, P8, and P11) also used the documentation panel.

We designed Newton WoZ to provide documentation in response to participants' queries or user actions. However, there were cases where proactively providing documentation would have been useful. For example, P4 faced an exception when trying to build the model. The wizard noted the exception and found a guide (external resource) to help. But, as per our WoZ script, the wizard had to wait for a user action, and, in the meantime, P4 fixed the error themselves by repeating the data-wrangling steps. In the post-task form, P4 indicated that Newton was missing "*error handling.*"

DG 3: Guide users proactively. A conversational agent should integrate output monitoring to be able to anticipate user actions.

S4 *Code Generation.* Auto-generated code by Newton helped participants to reduce programming efforts (**C2**). As P4 stated: "*it made coding easier, write efficient code fast*". Participants P4, P6, and P10 also pointed "code generation" as the most helpful feature in Newton in the post-task form: "*pre-written code,*" "*giving the code,*" "*code generation,*" respectively. These participants had high confidence in Python (see Table 1), suggesting that code generation can be useful even for experienced developers.

To reduce effort reduction and help participants avoid errors, Newton provided the code in the right formatted structure, contextualized to the task, which means users could use the code as is. The agent (wizard) had access to the notebook session to know the variable names and types to provide the correct code. For example, P11 used a dataframe with a different name (`data`) from that provided in Newton's script (`df`). So, when P11 requested a code in a subsequent step, the code had to be adjusted.

Besides contextualizing the code to the participants' notebook (current task), we *enriched* the code generation with code comments on complex operations, and the aforementioned documentation on all the invoked functions (**S3**). The goal was to help users understand the generated code and be able to maintain it in the future.

Most participants trusted the auto-generated code and copied it to the notebook. P1 was the only participant who changed the code to move an import statement to the first cell — which is a good practice according to Pimentel et al. [38]. After executing the code, half of the participants attempted to see what happened in the data by checking the output (P1, P2, P3, P4, P6, P10). They did not change the code after visualizing the results, indicating that their confidence (**C6**) in the results was high. Hence, we can further enrich future code generation by also including functions that display the output.

DG 4: Enrich code generation for understanding. The agent should include code comments, display the results, and give the option of accessing the documentation of generated functions to improve the understanding of the generated code.

S5 *Automated features.* While code generation reduced participants' effort, the auto-complete features did not help. Since the autocomplete was a pre-built Newton feature that was not controlled by the wizard, the list of suggestions may not have been presented in the best possible way, leading participants to disable the feature. For instance, P4 and P5 only used this feature at the beginning of the conversation (regions Figure 4:r4a and r5a), but disabled it after a few interactions that did not lead to the solution of the problem. P11 used this feature while typing to receive information about support vector machine classifiers, but did not follow that path and decided to disable the function (regions Figure 4:r11a and r11b). The rest of the participants deactivated the feature even before they started typing ("*How do I close this thing*", P8). Autocomplete turned out to be the least-used feature.

On the other hand, the copy-paste buttons were widely used. They allowed the users to automatically paste the code into notebook cells in the desired order. All participants widely used this feature after receiving some code from Newton. P1 stated, "*I really liked the ability to click on a button to add the recommended code snippets into a new cell at the bottom of the notebook for each step.*"

DG 5: Contextualize auto-complete features. Auto-complete functions need to be contextualized to the task to be useful.

4.2.3 Q3. What common patterns emerge when ML-EUPs perform ML tasks?

We analyzed participants' interactions in both Control and Experimental conditions to identify patterns or common behaviors by analyzing the video transcripts and observation logs.

(1) Backtracking: The most common interaction pattern was backtracking. Even though the situations were different, participants returned to a previous step in both conditions.

Without Newton: participants performed backtracking by searching, copy-pasting, testing, erroring, and going back to searching. Backtracking without Newton was time-consuming for participants as expressed by P8 "*I looked frequently and referred to websites for my solutions*". Similarly, P10, P11, and P12 searched online for code solutions using queries like "ML classifier in Python" or "classify a column using Python". After getting the search results, the participants went directly to the first three pages that popped out from the browser. From all the pages open, they just skimmed the tutorials trying to find keywords. Then they copied the code they thought could work into the notebook and tested it. They repeated this process until they encountered an error or the code showed completely different results from the ones they wanted.

With Newton: in this case, backtracking occurred when the user went to a previous Newton's reply—either by replying to it directly using the arrow icon Ⓓ or enacting a suggested option by clicking on its button Ⓖ—to explore alternative paths and topics in the conversation. In Figure 4, backtracking usually appears as edges that run across the different interaction nodes. With the exception of P9, all participants had an episode of backtracking. Figure 4 shows several examples of backtracking. It occurred when participants took steps to read some documentation and then went back to the ML steps. For instance, P1 and P3 opened documentation, then continued performing the next ML steps until successfully completing the task. On the other hand, backtracking did not work as well for some participants. For example, the interactions of participants P8 and P12 show how they had to backtrack several steps, resulting in errors and ultimately leading to an incomplete task.

(2) Tinkering: Another common pattern observed is when participants tinker with many options to find or get a desired answer. We describe how this happened in both conditions.

Without Newton: to find a desired explanation or code, some participants (P2, P5, P6, P10, P11) tinkered on as many links as their search result list showed. They used the open pages as a type of external cognition of useful resources to follow. However, these participants did not do a comprehensive review. Instead, they acted on the first relevant information source they identified. Therefore, some pages remained unopened, and once the participants completed the ML step they were working on, they closed all the pages and repeated this pattern when they searched for other code functions or other explanations. While this strategy worked in most cases, in others, participants selected the wrong resource and had to backtrack (P5, P6, P11). For instance, P11 opened several pages, such as TensorFlow tutorials, W3schools, Thispointer tutorials, and GitHub, in search of training code. However, after trying the code from each page, P11 realized it was not suitable and had to refine the search. This process was repeated twice until P11 found the desired code, but unfortunately, they ran out of time and were unable to complete the task. However, this behavior worked for participants P2 and P10.

With Newton: This pattern was common when participants received a checklist from Newton. For instance, when Newton provided a list of steps to perform classification, P2 clicked twice on the option `All` on the list (Figure 4:r2a). Then, they changed their mind before Newton's reply and tinkered with the first option from the checklist. By doing so, they created multiple conversation threads since the answers from Newton occurred in the order each option was tinkered. The participant then received a slew of answers to different questions, the participant then had trouble deciding which one to choose. P10 also had a similar situation, where they were impatient and clicked on other options before Newton could respond (Figure 4:r10a). Such (mindless) tinkering in quick succession caused errors in some cases. For instance, P11 tinkered under submission many times due to impatience, which brought errors (submission forms were not filled correctly) and finally to a not completion of the task. A large part of this issue can be attributed to the WoZ study setup, where the wizard had to send the response. Nevertheless, even in a fully automated conversation agent, mindless tinkering on the UI, can lead to multiple conversation threads that might cause user confusion.

DG 6: Manage multiple conversation threads clearly A conversational agent should efficiently manage multiple conversation threads and clearly show which path is currently being followed.

5 DISCUSSION

Overall, Newton proved to be helpful, irrespective of whether participants finished their tasks or not. Our findings provide evidence that incorporating the identified strategies as features into a conversational agent can help mitigate the challenges for ML-EUPs.

Challenges of ML-EUPs. The challenges the participants found during the study align with those we found in the literature. For instance, P7, P9, and P12 found themselves in a loop, searching for the perfect outcome. This loop challenge is very similar to the ones mentioned in this study, such as explainability **C3** and quality of evaluation **C6** (P12 when asked Besides these challenges, did you find others? If yes, please describe: " *(...) consuming to search for data and understand stuff since too many options*"). This triangulation between the types of challenges found in the literature and those found during the study can provide a foundation and motivation for finding strategies that can help mitigate these challenges.

All challenges, except for programming, could be mitigated using the Newton features. Although only half of our participants were able to finish the task using Newton, all participants found Newton's features helpful and expressed confidence in the responses. P1 stated: "*I think for me I'm still in the beginner stage of looking up tutorials and trying things out, but I liked having the support of Newton right there in the notebook. I felt like at least I could rely on Newton's answers a bit more than more random answers off the internet, which is what I use for doing other ML-related tasks . . . .*"

Participants who could not complete the task without Newton made progress by at least taking one step forward with the help of Newton. For example, P5, who had Medium programming experience, and low ML knowledge, could not finish even the first step without Newton (data loading). But with Newton's help, P5 progressed to data wrangling and mentioned "*I had a lot of choices and saw things I didn't know.* On the other hand, P6, with medium experience in programming and ML mentioned "*Newton helped me. From scratch, I was unable to perform the classification using my own resources, but with Newton, it was a lot easier.*" Note, P6, started with Newton and then moved to without-Newton condition, so any learning effects that might have occurred would have transferred to the Control condition.

Social characteristics of a conversational agent. A conversation is a social act, and a conversational agent's design needs to encode social characteristics. Chaves and Gerosa [12] elicit characteristics related to conversational intelligence, social intelligence, and personification to enrich the human-conversational-agent interaction. We attempted to include several of these characteristics in our experiment design and execution.

We personified the agent by calling it Newton and writing answers as a knowledgeable expert that adapts their writing to how the user interacts with it. For example, when the user sent a query with a greeting, Newton replied with a variation of the planned answer to include a personalized greeting. When the user sent informal messages, Newton adjusted its tone to use contractions and appropriate slang (e.g., "Got it"). These characteristics are related to the social intelligence of an agent.

We planned Newton's script by considering human conversation tone: all buttons were designed as replies that humans would give in a normal chat. For instance, the button for users to ask for help is labeled "Can you explain the steps?", and when the user clicks it, it shows up as a new message.

We also applied conversational intelligence characteristics to Newton's responses. In situations where the participant typed a confusing question, Newton rephrased the question and asked if the expressed intention was correct. In other situations in which the message was completely out of the scope of ML and programming, Newton managed the users' expectations by indicating that the specific subject was not in its database.

Despite our efforts, there is still room for improvement. First, multiple participants had an exception when they were applying some algorithms because their datasets had categorical variables as strings, and the algorithms required numeric variables. In these cases, Newton waited for the participant to interact (e.g., ask for help, send a query, click on a different button). A better solution would be to identify the exception from a catalog of known exceptions, and proactively send a message to warn the user.

Second, owing to the nature of WoZ, we had to deal with unexpected situations, and Newton lacked the knowledge to respond. In many situations, the wizard attempted to reply to unexpected questions by drafting responses in real time. Because of this, participants got impatient with the delay and started clicking on other buttons and typing new questions. According to Nielsen [35], the limit for keeping the user's attention focused on the dialogue is approximately 10 seconds. In a WoZ study, crafting human responses within this time is difficult to achieve in unexpected situations. We attempted to mitigate this problem by activating a loading icon when the message preparation was taking too long, but it was not enough, as users got impatient. In a fully automated conversation agent, the response time will not be an issue.

Finally, Newton also had limitations in keeping multiple conversations on track. The possibility of backtracking and replying to previous messages made some interactions with Newton confusing and intertwined. For instance, P2 wanted to advance on a task when he mistakenly clicked on the help button of a previous checklist, leading to an unhelpful reply from Newton (for the task at hand), instead of advancing to the next task.

Integrating existing Generative AI agents into Newton. The recent advances in Generative AI [1, 7, 36] that are trained on Large Language Models (e.g., GPT-4, LLaMA, PaLM) can be incorporated into Newton to facilitate natural language conversation with users. We plan to have Newton's UI as the way to interact with the user in the notebook environment, and the backend would be in charge of generating the right prompts to get the most appropriate outputs from generative AI. Newton would serve as a mediator, receiving the queries, creating the prompts automatically, and returning the information to the user.

Preliminary tests on GPT-4 from OpenAI indicate that strategies and guidelines related to code generation and documentation can be implemented with one-shot prompting [39]. For instance, we were consistently able to ask GPT-4 to enrich Newton's code generation (DG 4) by generating a parseable list of all functions in the generated code, adding code comments, and including functions that display the results. However, one-shot prompting is insufficient in creating an agent that provides step-by-step guidance for full ML workflow or in giving insights on what is needed next (DG 1). Few-shot prompting shows potential for some workflows, but we hit the 8,096 token limit in our attempts, as the generative AI needs extensive prior information to produce the desired answer.

As a workaround for this issue, we propose to use Newton to decompose the ML workflow for a task into smaller sets of steps (the way we did it in our WoZ script (see Section 3.2)). Newton could then act as a dialogue management system [26, 40] using the generative AI to interpret user intentions that are expressed in natural language.

Similarly, providing full contextualized support with proactive help (DG2, DG3) would require an additional management system to trigger the generative AI for specific user actions, with plans on what to observe and how to build the prompt with the context.

Finally, a drawback in the use of current generative AI is the possibility of hallucinations [2] (i.e., the AI provides an incorrect answer, but makes it sound correct so people believe it). This could be minimized by an implementation of a dialogue management system that covers most concepts in the usual workflow, but only fallbacks to answers from the generative AI for explanations that are not planned.

6 THREATS TO VALIDITY

All studies have risks, we have taken steps to minimize them and will describe them in detail below.

Construct validity: We acknowledge that the literature review process may have been affected by selection bias—the literature may only reflect some relevant work—and publication bias—positive results are more likely to be published. This may lead to an inaccurate understanding of the phenomena. To address these challenges, we piloted the queries and discussed all the steps of the analysis as a group through a negotiated consensus protocol.

We acknowledge another potential threat which is the possibility of participants misinterpreting the questions in the questionnaires. To minimize this potential issue, we piloted the questionnaires with developers of varying levels of expertise before administering them to the study participants.

Exhaustion and distraction may have affected the WoZ in remote settings. To minimize it, each session lasted at most 70 minutes. We also made an effort to reduce disruptions by giving developers (through the virtual machine testing environment) and facilitators (via the wizards) undisturbed simulated settings (Newton).

External validity: Our study was structured using Python and Jupyter Lab, the most commonly used tools in data science and machine learning. Therefore, we trade off generalizability for depth in these specific settings, and our results may not apply to other programming languages or environments.

We used the Wizard of Oz technique inspired by previous studies on chatbots [12, 25]. However, the use of human wizards instead of automated processing results in a time delay as wizards cannot replicate computational power. To mitigate this threat, we used our refined script to avoid delayed answers to the participants' questions. On top of that, since we were capturing ML-EUPs' interactions, these were not related to response time.

7 CONCLUSION

"*Newton is incredibly helpful for anyone who even has a rudimentary understanding of math and a few machine learning algorithms. Actually, they don't even need to be aware of that.*" — P3

In this work, we designed a conversational agent–Newton–that incorporates a set of five strategies recommended to help ML-EUPs. We evaluated the agent (Newton) via a WoZ experiment with 12 participants. In total, six participants were able to build the ML model with Newton. Two of them were not able to build the model without Newton, indicating that the guidance provided by Newton helped them. Moreover, even though some participants could not solve the tasks, they perceived the tasks as "less challenging" when using Newton as compared to when they did not use Newton.

We also observed how participants interacted with Newton. We noticed that participants liked to follow checklists with predefined actions, used Newton's assistance features, and trusted automated code generation. Two patterns emerged from the interactions with and without Newton: backtracking and tinkering. Backtracking occurred when participants explored alternative paths (with and without Newton), was time-consuming, and required multiple search refinements (without Newton), or when they wanted to continue the steps from a checklist (with Newton). Tinkering occurred when participants clicked on several links one after the other, trying to find information (without Newton), when they got impatient with wizard's slowness in response (with Newton), or when they wanted to explore different paths in checklists (with Newton).

The results reported in this work lay the foundation for future conversational agents that can support ML-EUPs, and form a stepping stone toward a low-code approach to ML. We plan to use the lessons learned in this study to implement and evaluate an actual conversational agent by prompt engineering a large-language model, such as GPT-4. We also foresee using the infrastructure we built for the Woz experiment in other contexts, such as designing conversational agents for programming education.

The replication package for this study with the WoZ script, forms, Newton implementation, and analysis is available at [5].

ACKNOWLEDGMENTS

REFERENCES

[1] Generative AI. 2023. https://generativeai.net/.

[2] Hussam Alkaissi and Samy I McFarlane. 2023. Artificial hallucinations in ChatGPT: implications in scientific writing. *Cureus* 15, 2 (2023), 1–4.

[3] Hamza Hussein Altarturi, Keng-Yap Ng, Mohd Izuan Hafez Ninggal, Azree Shahrel Ahmad Nazri, and Abdul Azim Abd Ghani. 2017. A requirement engineering model for big data software. In *2017 IEEE Conference on Big Data and Analytics (ICBDA)*. SciTePress, Prague, Czech Republic, 111–117. https://doi.org/10.1109/ICBDAA.2017.8284116

[4] 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 / ACM, Montreal, QC, Canada, 291–300. https://doi.org/10.1109/ICSE-SEIP.2019.00042

[5] Anonymous. 2023. *Replication Package for ML-EUP Conversational Agent Study*. Anonymous. https://doi.org/10.5281/zenodo.7780223

[6] Anders Arpteg, Björn Brinne, Luka Crnkovic-Friis, and Jan Bosch. 2018. Software engineering challenges of deep learning. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, IEEE Computer Society, Prague, Czech Republic, 50–59.

[7] Bloomberg. 2023. A Cheat Sheet to AI Buzzwords and Their Meanings: QuickTake — https://news.bloomberglaw.com/tech-and-telecom-law/a-cheat-sheet-to-ai-buzzwords-and-their-meanings-quicktake.

[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, Memphis, USA, 25–34.

[9] Arif Cam, Michael Chui, and Bryce Hall. 2019. *Global AI survey: AI proves its worth but few scale impact.* Technical Report. McKinsey Analytics.

[10] Jhonny Cerezo, Juraj Kubelka, Romain Robbes, and Alexandre Bergel. 2019. Building an Expert Recommender Chatbot. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. IEEE / ACM, Montreal, QC, Canada, 59–63. https://doi.org/10.1109/BotSE.2019.00022

[11] Souti Chattopadhyay, Ishita Prasad, Austin Z Henley, Anita Sarma, and Titus Barik. 2020. What's wrong with computational notebooks? Pain points, needs, and design opportunities. In *Proceedings of the 2020 CHI conference on human factors in computing systems*. ACM, Honolulu, HI, USA, 1–12.

[12] Ana Paula Chaves and Marco Aurélio Gerosa. 2019. How Should My Chatbot Interact? A Survey on Social Characteristics in Human-Chatbot Interaction Design. *Int. J. Hum. Comput. Interact.* 37, 8 (2019), 729–758. https://doi.org/10.1080/10447318.2020.1841438 arXiv:1904.02743

[13] João Lucas Correia, Juliana Alves Pereira, Rafael Mello, Alessandro Garcia, Baldoino Fonseca, Márcio Ribeiro, Rohit Gheyi, Marcos Kalinowski, Renato Cerqueira, and Willy Tiengo. 2020. Brazilian data scientists: revealing their challenges and practices on machine learning model development. In *19th Brazilian Symposium on Software Quality*. ACM, São Luís, Brazil, 1–10.

[14] N. Dahlbäck, A. Jönsson, and L. Ahrenberg. 1993. Wizard of Oz studies — why and how. *Knowledge-Based Systems* 6, 4 (1993), 258–266. https://doi.org/10.1016/0950-7051(93)90017-N Special Issue: Intelligent User Interfaces.

[15] Rajeev Davenport, Thomas H.and Ronanki. 2018. *Artificial intelligence for the real world.* Technical Report. Harvard business review.

[16] Zixuan Feng, Amreeta Chatterjee, Anita Sarma, and Iftekhar Ahmed. 2022. A case study of implicit mentoring, its prevalence, and impact in Apache. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, Singapore, Singapore, 797–809.

[17] Barney G Glaser. 2016. Open coding descriptions. *Grounded theory review* 15, 2 (2016), 108–110.

[18] Fuyuki Ishikawa and Nobukazu Yoshioka. 2019. How Do Engineers Perceive Difficulties in Engineering of Machine-Learning Systems? - Questionnaire Survey. In *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*. IEEE / ACM, Montreal, QC, Canada, 2–9. https://doi.org/10.1109/CESSER-IP.2019.00009

[19] Michael I Jordan and Tom M Mitchell. 2015. Machine learning: Trends, perspectives, and prospects. *Science* 349, 6245 (2015), 255–260.

[20] Project Jupyter. 2022. Jupyter. https://jupyter.org/

[21] Ioannis Karamitsos, Saeed Albarhami, and Charalampos Apostolopoulos. 2020. Applying DevOps practices of continuous automation for machine learning. *Information* 11, 7 (2020), 363.

[22] Anuj Karpatne, Imme Ebert-Uphoff, Sai Ravela, Hassan Ali Babaie, and Vipin Kumar. 2019. Machine Learning for the Geosciences: Challenges and Opportunities. *IEEE Transactions on Knowledge and Data Engineering* 31, 8 (2019), 1544–1554. https://doi.org/10.1109/TKDE.2018.2861006

[23] Ryan Kirwan, Javihn Che, Woo Jia Le, and Stefan Sarin. 2021. A Visualization and Analysis tool for VCL Auto-generation Code Framework. In *2021 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*. IEEE, Singapore, 1–5. https://doi.org/10.1109/SOLI54607.2021.9672425

[24] Scott R. Klemmer, Anoop K. Sinha, Jack Chen, James A. Landay, Nadeem Aboobaker, and Annie Wang. 2000. Suede: A Wizard of Oz Prototyping Tool for Speech User Interfaces. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology* (San Diego, California, USA) *(UIST '00)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/354401.354406

[25] Mohammad Amin Kuhail, Nazik Alturki, Salwa Alramlawi, and Kholood Alhejori. 2022. Interacting with educational chatbots: A systematic review. *Education and Information Technologies* 28, 1 (2022), 1–46.

[26] Jonáš Kulhánek, Vojtěch Hudeček, Tomáš Nekvinda, and Ondřej Dušek. 2021. AuGPT: Auxiliary tasks and data augmentation for end-to-end dialogue with pre-trained language models. *arXiv preprint arXiv:2102.05126* 2102, 05126 (2021), 1–13.

[27] Fumihiro Kumeno. 2019. Sofware engineering challenges for machine learning applications: A literature review. *Intelligent Decision Technologies* 13, 4 (2019), 463–476.

[28] Alexandra L'heureux, Katarina Grolinger, Hany F Elyamany, and Miriam AM Capretz. 2017. Machine learning with big data: Challenges and approaches. *Ieee Access* 5 (2017), 7776–7797.

[29] Silverio Martínez-Fernández, Justus Bogner, Xavier Franch, Marc Oriol, Julien Siebert, Adam Trendowicz, Anna Maria Vollmer, and Stefan Wagner. 2022. Software Engineering for AI-Based Systems: A Survey. *ACM Trans. Softw. Eng. Methodol.* 31, 2, Article 37e (apr 2022), 59 pages. https://doi.org/10.1145/3487043

[30] Satoshi Masuda, Kohichi Ono, Toshiaki Yasue, and Nobuhiro Hosokawa. 2018. A Survey of Software Quality for Machine Learning Applications. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE Computer Society, Västerås, Sweden, 279–284. https://doi.org/10.1109/ICSTW.2018.00061

[31] Sahar Mehrpour, Thomas D. LaToza, and Rahul K. Kindi. 2019. Active Documentation: Helping Developers Follow Design Decisions. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE Computer Society, Memphis, Tennessee, USA, 87–96. https://doi.org/10.1109/VLHCC.2019.8818816

[32] Dane Morgan and Ryan Jacobs. 2020. Opportunities and challenges for machine learning in materials science. *Annual Review of Materials Research* 50 (2020), 71–103.

[33] Shan Nan, Pieter Van Gorp, Hendrikus H. M. Korsten, Uzay Kaymak, Richard Vdovjak, Xudong Lu, and Huilong Duan. 2015. DCCSS: A meta-model for dynamic clinical checklist support systems. In *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. SciTePress, Angers, Loire Valley, France, 272–279.

[34] Anh Nguyen Duc, Ingrid Sundbø, Elizamary Nascimento, Tayana Conte, Iftekhar Ahmed, and Pekka Abrahamsson. 2020. A Multiple Case Study of Artificial Intelligent System Development in Industry. In *Evaluation and Assessment in Software Engineering*. ACM, Trondheim, Norway, 1–10. https://doi.org/10.1145/3383219.3383220

[35] Jakob Nielsen. 1994. *Usability engineering*. Morgan Kaufmann, San Francisco, CA, USA.

[36] OpenAI. 2023. GPT-4. https://openai.com/product/gpt-4.

[37] Andrei Paleyes, Raoul-Gabriel Urma, and Neil D Lawrence. 2020. Challenges in deploying machine learning: a survey of case studies. *ACM Computing Surveys (CSUR)* 55, 6 (2020), 114:1–114:29.

[38] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2021. Understanding and improving the quality and reproducibility of Jupyter notebooks. *Empirical Software Engineering* 26, 4 (2021), 65.

[39] Laria Reynolds and Kyle McDonell. 2021. Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI EA '21)*. Association for Computing Machinery, New York, NY, USA, Article 314, 7 pages. https://doi.org/10.1145/3411763.3451760

[40] Mahdin Rohmatillah and Jen-Tzung Chien. 2021. Corrective Guidance and Learning for Dialogue Management. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (Virtual Event, Queensland, Australia) *(CIKM '21)*. Association for Computing Machinery, New York, NY, USA, 1548–1557. https://doi.org/10.1145/3459637.3482333

[41] Dhia Elhaq Rzig, Foyzul Hassan, and Marouane Kessentini. 2022. An empirical study on ML DevOps adoption trends, efforts, and benefits analysis. *Information and Software Technology* 152 (2022), 107037.

[42] Vidya Setlur and Melanie Tory. 2022. How Do You Converse with an Analytical Chatbot? Revisiting Gricean Maxims for Designing Analytical Conversational Behavior. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 29, 17 pages. https://doi.org/10.1145/3491102.3501972

[43] Royal Society. 2017. *Machine Learning: The Power and Promise of Computers that Learn by Example: an Introduction.* Technical Report. Royal Society.

[44] Andreas Vogelsang and Markus Borg. 2019. Requirements Engineering for Machine Learning: Perspectives from Data Scientists. In *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*. IEEE, Jeju Island, South Korea, 245–251. https://doi.org/10.1109/REW.2019.00050

[45] George Vrettas and Mark Sanderson. 2015. Conferences versus journals in computer science. *Journal of the Association for Information Science and Technology* 66, 12 (2015), 2674–2684.

[46] Zhiyuan Wan, Xin Xia, David Lo, and Gail C. Murphy. 2021. How does Machine Learning Change Software Development Practices? *IEEE Transactions on Software Engineering* 47, 9 (2021), 1857–1871. https://doi.org/10.1109/TSE.2019.2937083

[47] Claes Wohlin. 2014. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering* (London, England, United Kingdom) *(EASE '14)*. Association for Computing Machinery, New York, NY, USA, Article 38, 10 pages. https://doi.org/10.1145/2601248.2601268

[48] Frank F. Xu, Bogdan Vasilescu, and Graham Neubig. 2022. In-IDE Code Generation from Natural Language: Promise and Challenges. *ACM Trans. Softw. Eng. Methodol.* 31, 2, Article 29 (mar 2022), 47 pages. https://doi.org/10.1145/3487569