Simple Human-in-the-loop System for Modern Data Science Workflow

An-Dan Nguyen[‡]
Computer Science Department
Aalto University
Espoo, Finland
dan.nguyen@aalto.fi

Linh Truong[†]
Computer Science Department
Aalto University
Espoo, Finland
linh.truong@aalto.fi

***Abstract*—Seamlessly incorporating human decisions into semi or fully automated data-centric processes has not always been easy. In recent years, data science and analytics have improved productivity in many fields through the advances in computational power. Consequently, the human-in-the-loop problem in these fields is forthcoming. Moreover, as the nature of data science and analytics is a knowledge discovery process, the characteristics of the supported human-the-loop system must cater to that nature. In this work, those characteristics of the supported human-in-the-loop in data science and analytics have been modeled. Concepts of interactions between users and the data science workflows have been derived. A simple human-in-the-loop has been developed based on these concepts. Two simple but representative scenarios are used to evaluate the human-in-the-loop support system. The results show potential in extending the system to support real-world data science workflows.**

***Index Terms*—human-in-the-loop, data science, data analytic, data science process**

I. Introduction

Human-in-the-loop (HITL) problems can arise between the automation system and the human. Recently, computational hardware and algorithmic technology innovations have helped people utilize data science and analytic process in numerous fields and industries. The interactions between the human and the automation system in data science and analytics give rise to a new class of HITL research problems.

In general, interactions between humans and automation systems can be receiving notifications, awaiting input/decisions from users, or both, depending on the interactions. A HITL system in data science will have these basic features. However, since there are unique aspects in the data science process, the supported HITL system must also have features to accommodate those needs.

Let us examine two common scenarios in the modern data science pipeline. There are the streaming and the moving data scenario. These scenarios represent interactions between users and workflows, which the proposed HITL system should accommodate. Common problems in modern data science workflows are also reflected in these scenarios.

The streaming data represents the scenario when data is moving too fast and in a large volume. It can be data received from IoT devices, data generated by users from well-known websites or social networks, or data from the financial markets. The velocity and volume of data in this scenario pose a challenge to the data science workflow to handle the data for analysis or prediction properly. Hence, user inputs are often needed to guide the system, especially in the data exploration stage when data science tasks are not clearly defined. Seamless interactions between the data science system and the users are essential in such situations.

The moving data scenario represents another scenario where the velocity and volume are not necessarily fast or large, but there is uncertainty in task completion time. The uncertainty can be due to unreliable network connections or external APIs. Uncertainty in completion time creates difficulty in planning subsequent workflow tasks, creating idle time. It is crucial in this situation to notify the users about the task's status and the following steps to take when the task is complete. A HITL system can provide frictionless communication between user and system, which can be beneficial.

Besides facilitating smooth interactions between systems and users, there is another implementation aspect that the HITL system should take into account. In the modern and in-practice data science workflows, each task such as data acquisition, data engineering, and data analysis can be too complicated to handle by one person alone. These tasks are often developed into services and maintained by a team or several developers with sophisticated workflows of their own. Hence, if the HITL systems are hard to use or drastically change the workflow within or between teams, the HITL system has a low chance of being used. Therefore, the proposed system should be easily incorporated into established workflow and systems.

The requirements of a HITL system for data science workflow can be developed from these representative scenarios and the implementation constraints outlined above. Two basic requirements of the system are outlined below.

First, the user needs a system that can provide flexible modes of interaction because the results at each step can be surprising in a data science workflow. Unexpected results are why users often have to go back at any steps to refine the methods to produce more satisfying outcomes. Users need to answer the questions that begin with "why" and "how" upon

[‡] Corresponding author
[†] Project's supervisor

receiving those unexpected results. Therefore, interactions between users and the data science systems are exploratory and investigative. Appropriate, supported tools are needed to facilitate those flexible interactions.

Second, the cost and complexity of data science tasks can be very high due to the large amount of data needed to be processed or the computational complexity. Hence, an interaction between a user and a data science workflow can be costly and complex. For example, running a complexity model such as a deep neural network or transferring big data between storage and computation server (moving data scenario) are instances of high cost and complexity originating from the data. Meanwhile, the uncertainties in choosing the data, methods, or intermediate results can complicate decisions during the data science process. Wrong decisions will lead to repeated previous or some previous steps several times, resulting in high costs. Therefore, a HITL for data science also needs to consider the complex and high-cost interactions.

One solution to address these problems can be a checkpoint system that breaks down complex and long-running time tasks into smaller parts, then saves the intermediate results and notify the users. Two basic modes of interactions in this checkpoint system are in-between processes and within the process. Figure 1 and Figure 2 shows examples of the interaction between processes and interaction within a process respectively.

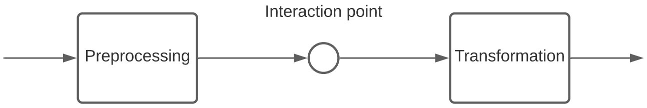

Fig. 1. An interaction between human and the workflow can happen between processes

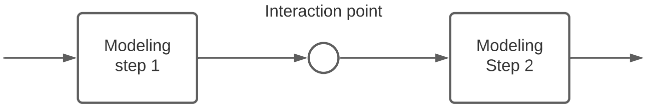

Fig. 2. An interaction between human and the workflow can happen within process

In this preliminary work, a versatile, simple HITL system for data science is developed to address the two basic requirements mentioned above. Concepts of an interaction model are presented. Later, based on the developed concepts, a simple HITL system is realized to demonstrate those concepts.

The rest of the paper will be structured as follows: Section II discusses some significant relevant works to this subject. Section III introduces the models of interactions upon which our human-in-the-loop system is based. Based on our models of interactions, concepts of action model and action plan are derived. Then, in section IV, our human-in-the-loop system implementation, based on these concepts, is presented. Section V describes in detail the scenarios to evaluate the proposed system. Then, Section VI presents the details of the two data scenarios we have mentioned and the results when applied to our HITL system. Finally, in section VII, we summarize our results, present the drawbacks, and lay out the plan for future research.

II. Related Works

Human-in-the-loop problems in machine learning and data mining have drawn the research community's attention in the last several years. Various works approached the problems from different perspectives. This section summarizes the notable works and highlights those closely related to our scenarios.

First, let us briefly mention the history of human-in-the-loop works in computer-based processes. Cranor developed one of the earliest human-in-the-loop frameworks, which can account for human behaviors given the system outputs [1]. These system outputs are called communications. Based on the type of communications the author has derived and the characteristics of the human receivers, which are a set of personal variables, intentions, and capabilities that influence a set of information processing steps, desirable behaviors of the users can be predicted.

The framework was based on the Communication-Human Information Processing (C-HIP) model developed by Wogalter [2]. A quick note here is that the C-HIP model has gone through several iterations throughout the years. There were three versions. The first one was published in 1999 [3], the second one was in 2006 [2], and the latest one was in 2018 [4]. Cranor's work was based on the 2006 version.

Two other works which inspired Cranor's paper worth mentioning here are Don Norman's seven-stage "action-cycle" [5] and James Reason's Generic-Error Modeling System [6]. Norman's framework formulated how human sets goals and perform action sequences to achieve those goals. The framework can be used to evaluate a computer system's user interface. Meanwhile, Reason's framework classified human errors into rule-based and knowledge-based errors. The framework helps gain a deeper understanding of human errors and correctly anticipate when they occur.

Although the application domains of these works mentioned above were not data-related fields (computer security (Cranor), warning system (Wogalter), design (Norman), and human errors (Reason)), these early frameworks are general enough that the principles can be applied and motivated other human-in-the-loop systems such as the one we are going to develop.

Recently, there have been attempts to address problems in human-in-the-loop in machine learning and data science. One of them is the works of Doris Xin et al. in the Helix system. [7] [8] [9] [10]. In [8], The authors listed the features that current ML systems lack, such as iterative use, introspection between iterations, leverage think-time between iterations, reducing iterative feedback latency, and automated cues for iterative change. The author also proposed the research challenges for future research in machine learning human-in-the-loop. In addition, prerequisites for such ML systems were proposed by the authors. These are declarativity and generalization. Both are tackled by modeling the workflow as a directed acyclic graph (DAG). Moreover, the authors propose Helix, a first attempt to address the prerequisites and the iterative use challenge mentioned above. Helix was compared with DeepDive [11] and KeystoneML [12] in information extraction and classification tasks. Helix helped reduce the running time substantially in both tasks compared to DeepDive and KeystoneML. Details about the Helix system can be examined in [9] or in the technical reports [10].

While Doris et al. have listed the requirements and challenges of a desirable machine learning human-in-the-loop (MLHiTL), Helix can only address one of those requirements. Moreover, the user needs to learn and be familiar with Helix's domain-specific language (DSL) before taking advantage of Helix's features which can be a steep learning curve. Nevertheless, a system such as Helix and the author's requirements serve as a strong foundation for future works.

Another framework that attempted to improve the seamlessness between the human and the data science framework is iSmartML [13]. Specifically, the framework lets the users intervene in the AutoML pipeline building process by stating their preferences to guide the search space through a configuration control panel. A recommendation engine is built to help guide the users to narrow down their search space further. In addition, a monitoring panel is provided to allow the users to track the process and send alerts when the desired results are found. Furthermore, users can also understand and diagnose the results produced by the framework. Finally, a logging mechanism stores the result between several iterations given a dataset to avoid re-exploring searched spaces to save time.

We can see the similarities between iSmartML and the Helix framework mentioned above. For example, the monitoring panel to promote the interaction between users and the framework, a logging/storing mechanism to utilize the results between iterations, and ways to specify the user preferences of the model (the DSL in the case of Helix and the configuration control panel in the case of iSmartML). These similarities reflect the basic requirements of a human-in-the-loop machine learning framework which also have been identified in Doris' works. However, the difference between iSmartML and Helix is that Helix requires more effort from the user, such as learning and specifying the pipelines through the DSL, but the user has control over the whole data science process. Meanwhile, iSmartML required only the user to state their preferences and nudge the search space. The framework performs the pipeline building process. We predict that these two frameworks do not contradict each other but represent two major camps that classify future frameworks.

There are also a set of tools that potentially be beneficial in improving the effectiveness of interaction between humans and machines. These tools are not data science frameworks themselves but can be incorporated into them.

One set of helpful tools is the interactive visualization tools which can help the users to introspect and explore the intermediate results, as stated in the second research challenge by Doris et al. The notable examples of the tools developed by the researchers at MIT Visualization Group such as Vega-Lite [14], B2 [15], and Lyra [16]. Vega-Lite took the inspiration from the Grammar of Graphics of Leland Wilkinson [17] and implemented those concepts in graphical visualization tools such as ggplot2 [18] to develop a high-level grammar of interactive graphics. Meanwhile, B2 helped bridge the gap between interactive visualization and analysis code in computational notebooks. The system links and changes the code corresponding to the performed interactions. A history of changes through interactions is also generated to keep track and serve as references for further analysis. Finally, Lyra is an interactive data pipeline interface for data transformation and visualization design using drag-and-drop interactions. The visualizations created by Lyra are easily shared and reused using the Vega [19], a lower level of Vega-Lite, specifications.

The other tools can be seen as enhancements in different aspects of the framework. For example, Spoth et al. developed a Loki system [20], which is a system consisting of dataset and mapping functions to reuse the mapping functions in a different dataset and context. Another example is the work of Zhang et al. [21]. The authors derived a general form of utility function to rank the visualizing view effectively. The view is defined to be the aggregation of measurable (numerical) attributes (features) by dimension (categorical) attributes.

This section presents notable works that show different directions in current data science human-in-the-loop research. Although the frameworks mentioned above are handy in the appropriate context, the users need to learn to use them or change their current working environments to receive their benefits. In the next section, we introduce our attempt to alleviate that problem. We present a lightweight and versatile system that users can plugin in any data science process to achieve their goals. The user only has to specify the compatible input and output formats. Next, we present our model of interactions as a basis for such a system.

III. Model of Interactions

In this section, a model of interactions is presented. This model serves as a basis on which our system operates. We go through the overview of the model. Then we detail the action model design, which is the focus of our research.

A. Overview

There are three essential components of our proposed model of interactions. They are flow, action model, and communication methods.

1) Flow: We defined *flow* as a sequence of processes that turns a question (which can accompany a data set) into answers or data products. The flow represents the actual data science process in practice. An example of the flow is the KDD process [22]. In the modern data science process, the flow can sometimes be the task involved in storing and retrieving data, especially when the data is large and has to be stored in a remote data warehouse or accessed through external APIs.

The flow we defined here is very general and can be applied in any series of processes, even when the sub-process of the flow is not clearly defined yet.

In the flow, there is a list of *interaction points* which the user wishes to interact with the data science process (the flow) to either examine the intermediate results (e.g., plot, compute statistics) or make decisions that can change the direction of the flow (e.g., go back to the previous step, add in more data, change the parameters of the subsequent process). The interaction point is where the action model is applied to provide the user with the appropriate action plans depending on the user's context and preferences. For example, an interaction point can be placed to probe the transformed data after transformation steps. Another example, the user can place an interaction point when the data have been downloaded entirely from a remote server to examine before starting analytic processes.

One minor but important note is that all the intermediate data up to the point of interaction is saved for easy exploration and investigation.

Although the concept of flow is essential, this is not our focus in this research.

2) Action Model: The *action model* is an abstract sequence of actions executed when an interaction point is reached in the flow. An *action plan* is a specific realization of an action model given the contextual/environmental variables such as the nature of the tasks, who is in charge/monitoring the execution of the process, or who will give the decision when an interaction point is reached. The action plan will consist of concrete actions predefined by the users or learned from past data. Also, the user can define an action model with the abstract action types. Our research focuses on the action model and will be presented clearly in the following sections.

3) Communication methods: The *communication methods* are how the human and data science process interact. Specifically, in our case, the user specifies and then sends the action plans given by the action model, and the data science process receives and executes those actions and returns the results. Although this is one component of our model, this is not a focus of this research. A simple message-passing paradigm will be employed in this research to demonstrate the concept.

B. Action Model, Action Plan, and Concrete Action

This section describes the action model, action plan, and concrete action. These concepts are the focus of this research.

1) Action Model: The action model is a user behavior template when encountering an interaction point. The action model serves as a template to generate action plans based on the contexts at the interaction points. At the interaction point, the context is the set of the nature of the tasks, the data, user preferences, and the goals the users want to achieve. The action plan can be learned from past data or predefined by the user based on the action model. The action plan guides the user through predefined concrete actions suitable for handling events at the interaction points.

Figure 3 summarizes the concept of flow, action model, and action plan we have devised above.

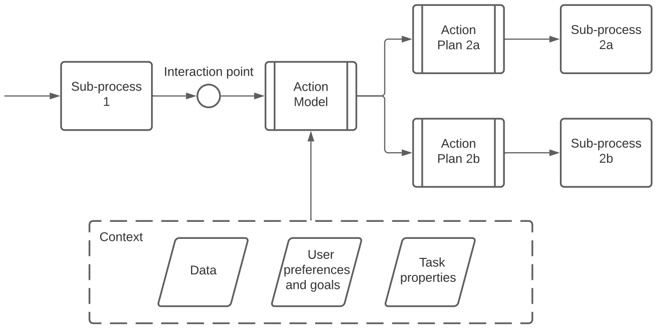

Fig. 3. Demonstration of the flow with an interaction point, action model, and action plans

Let us demonstrate the action model by the first action model we develop based on this model of interactions. Inspired by the popular internet service IFTTT[1], we derived the action model *if-this-then-that* for the data science process.

2) The if-this-then-that action model: This action model has two main elements. The cause element *if-this* and the effect element *then-that*. This action model becomes a concrete action plan when the cause and effect elements have concrete conditions and actions. For example, assume we have the concrete condition "data downloaded is completed" and the action "notify Peter," the action model becomes the concrete action plan "if data downloaded is completed, then notify Peter."

This action model is simple yet powerful because it can be applied in many situations and connect various scenarios. One problem this action model can help alleviate is the asynchronous nature of the steps in the data science process, such as waiting for the model to train or getting data from a remote server. Alternatively, the user can implement a set of strategies using this action model to prepare for unexpected events or get input from the experts on how to proceed to the next step.

The simple action model also can potentially be implemented by a wide range of APIs, libraries, and software tools. The user can easily incorporate domain-specific software into the workflow using this action model, given the appropriate input and output formats.

The action model can also be easily extended by increasing the condition in the "if-clause" or by adding a complex function "then-clause." This action model potentially be nested as *if-clause* in programming languages. However, for the sake of demonstration and simplicity, we do not intend to implement the concept of a nested action model in this research.

The action model can be represented in JSON format as follows:

```
{
    "if": <if-clause-function>,
```

[1] https://ifttt.com/

```
    "if-clause-args": {
        <args1>: <args1-value>,
        <args2>: <args2-value>,
        ...
    },
    "then": <then-clause-function>,
    "then-clause-args": {
        <args1>: <args1-value>,
        <args2>: <args2-value>,
        ...
    },
}
```

Listing 1. Conceptual *if-this-then-that* action model in JSON format

3) Action Plan: The action plan is a concrete realization of an action model. The action plan is an action model in a specific situation. Following programming terminology, an action model can be seen as a class, while an action plan is an object of that class. For example, an action plan of the action model *if-this-then-that* describe above is when the *if-statement* and the then-statement have specific values.

4) Concrete Action: Concrete action is a single action in the action plan. An action plan can have multiple concrete actions which link together in a specific way depending on the action plan. Following programming terminologies again, the concrete action of an action plan is the attribute (with value) of an object of a class. For example, in the action plan of the action model *if-this-then-that*, the *if-statement* and the *then-statement*, when they have values, are the concrete action of the action plan.

In the next section, a simple realization of the model of interactions and especially the proposed action model *if-this-then-that* is described. Then, two scenarios, streaming and moving data, are used to demonstrate the implementation.

IV. Implementation

Python has been used to implement the system. The implementation has been published as a Github repository[2]. The repository contains two main packages, the `action_model` package and the `scenario` package. The `action_model` package contains the implementation of action models. Currently, the action model *if-this-then-that* has been implemented in the module `ifttt.py`. The `scenario` package contains the realization of the streaming data and the moving data scenario with the action model *if-this-then-that*.

There is a `action_plan` folder that contains the realization of the concrete action plans of each scenario based on the action model *if-this-then-that*. The action plan file is written in JSON. The implementation of the action model *if-this-then-that* has some differences compared to the general specification described in section III due to practices constraint. Nevertheless, the overall idea is the same. Below is the JSON specification of the *if-this-then-that* action model in practice. The specification presents the parameters' name and their data types.

```
{
    "redis_server": <str>,
    "redis_port": <int>,
    "condition_clauses": [
    {
      "condition_variable": <str>,
      "condition_operator": <str>,
      "threshold_value": <numeric>
    },
    ...
  ],
  "target_module_file_path": <str>,
  "target_module_name": <str>,
  "target_function_name": <str>,
  "target_function_params": <JSON object>
  "num_repeat_recipe": <int>,
  "num_processes": <int>
```

Listing 2. *if-this-then-that* action model realization in JSON format

In the JSON configuration file, `redis_server` and `redis_port` store the connection information of the Redis server. Redis is employed to facilitate the communication between users and the system based on a message-passing paradigm.

The `condition_clauses` specifies the *if-statement*. It is an array of conditions. For demonstrative purposes, conditions are currently connected by `AND` operator. Each condition specified by three parameters `condition_variable`, `condition operator`, and `threshold value`. The `condition operator` are currently relational operators. The system can support $<, \leq, =, \geq, >$ operators. The operators are represented by the strings `lt`, `lte`, `eq`, `gte`, `ge` respectively in the configuration file.. The `condition_variable` and `threshold value` are the left hand side and the right hand side of the operation respectively. At the moment, `threshold value` only accepted numerical values.

The $target\_module\_file\_path$, $target\_module\_name$, $target\_function\_name$, and $target\_function\_params$ are the elements of the *then-statement* when the *if-statement* is satisfied. The first three elements are due to the implementation constraints in Python to load the specified function. The last element is the JSON object that stores the inputs of the that function in a key-value format.

The two last parameters, `num_repeat_recipe` and `num_processes`, specify the number of times the recipe should run (repeat) and how many Python processes are used to run the recipe each time. These are in-practice parameters. The default values of these are one. Below is the concrete example of an action plan based on the action model *if-this-then-that*.

```
{
    "redis_server": "localhost",
    "redis_port": 6379,
    "condition_clauses": [{
```

[2] https://github.com/ngndn/hitl-ds

```
        "condition_variable": "counter_1",
        "condition_operator": "gte",
        "threshold_value": 10000
    }, {
        "condition_variable": "counter_2",
        "condition_operator": "gte",
        "threshold_value": 1000
    }, ... ],
    "target_module_file_path": "./plotter.py",
    "target_module_name": "plotter",
    "target_function_name": "streaming_data_consumer",
    "target_function_params": {
        "dataset_1_path": "../../../data/streaming_data_1",
        "dataset_2_path": "../../../data/streaming_data_2",
        "result_file_path": "../../../result/streaming_plot.png""
    },
    "num_repeat_recipe": 1,
    "num_processes": 1
}
```

Listing 3. An example of a concrete action plan in JSON format based on *if-this-then-that* action model

The action plan can be described in English as: ”If $counter\_1 \geq 10000$ AND $counter\_2 \geq 1000$, run the function *streaming_data_consumer* in the module `plotter`, located in the file `plotter.py`, in the current directory, with the specified arguments and their values.”. This action plan is from the streaming data scenario example.

The implementation of the *if-this-then-that* action model is simple. The class `IFTTT` has helper methods to parse and load the parameters in the action plan file into class attributes. The method `run()` contains all the logic of monitoring the conditions and executing the target function (the then-statement) when the conditions are met. Hence, implementing and running the action plan is also very simple. Ones can create the `IFTTT` object from the configuration file, then call the method `run()`. Details can be seen in the examples on the GitHub repository.

It is worth mentioning that the *else-statement* can be easily implemented by the same programming technique as shown in the *then-statement* if needed. In order to simplify the demonstration, *else-statement* is omitted in this implementation and will be presented in future development iterations.

The streaming data scenario and the moving data scenario demonstrate the system with the *if-this-then-that* action model. The following section presents these scenarios in detail.

V. Simulation Scenarios

Since there are constraints in time, applying the proposed system in a real-world data science workflow and measuring productivity improvement is not feasible. Instead, simulated data science workflow scenarios are used to demonstrate the utilities of the implemented system. In data science workflows, handling the data properly at any stage is crucial, especially in the modern context where data volume, velocity, and complexity are high. Hence, these two scenarios, streaming data and moving data will be used to demonstrate the applicability of the proposed system in those tasks.

A. Streaming data scenario

In the streaming data scenario, simulation data was generated to simulate the data streams, such as user interactions from e-commerce websites or trading data from the stock markets. The simulated streaming data come from two bivariate normal distributions with different kernels representing the different kinds of streaming data. An action plan is triggered after a certain amount of data is accumulated in both data streams to demonstrate the system. In the first stream, the number of target data points is 10000. Meanwhile, it is 2000 data points in the second stream. Because plotting the data is a prevalent yet essential task to examine the data quality in data science workflow, it is applied in this action plan. In short, in the streaming data scenario, simulated streaming data is generated from two different bivariate normal distributions. An action plan to plot the data is executed after the data have been accumulated enough from both streams.

B. Moving data scenario

Like streaming data, the moving data scenario is also about handling data in the data science workflow. However, this scenario focuses more on the uncertainty aspect of the tasks. In modern data science, apart from getting data from the data stream, accessing data from external APIs and moving data around data centers are standard procedures. The unreliability of the APIs or network connections creates uncertainties in the completion time of those tasks. In turn, these uncertainties create unnecessary idle time. In order to eliminate the idle time, user-preferred actions should be run right after these tasks are completed. In this scenario, a dataset is created and moved from a source folder to a destination folder to simulate the above use case. The dataset was generated from a bivariate normal distribution. An action plan was placed after the event the dataset was moved entirely to the destination folder to run the subsequent task. That task is to plot data, same in the streaming data scenario. In summary, in the moving data scenario, an action plan is placed after the event the data has been completely moved to plot the data.s

In the next section, the details of how to apply our developed action model in those simulation scenarios are presented, along with the results.

VI. Experiment and Result

A. Experiment

Streaming and moving data scenarios are implemented as experiments to evaluate the developed action model *if-this-then-that*. The overview of these scenarios is described in section V.

1) Streaming data scenario: The streaming data scenario is described in the action plan specification `streaming_ifttt.json` in the folder `action_plan`. The action plan is to run a plot function based on certain criteria when there is enough accumulated data from those data streams. Data counters are stored in Redis to count the data accumulated from the stream. The data counters is the *if-statement* and running the plot function is the *then-statement*. Details are presented in the mentioned action plan.

The streaming scenario was implemented in the packages `scenario.ifttt.streaming`. In order to run the scenario, the module `setup.py` is run first to set up the environment. Then the module `run_recipe.py` is used to start the action plan. Finally, the module `run_scenario.py` runs the streaming scenario. The result is produced in the repository folder `result`.

2) Moving data scenario: This scenario simulates data moving operations between data warehouses or data getting from external APIs. A simple plot is produced when the data is moved from a source to a destination folder. A flag in Redis is set when the moving data operation is completed. Similar to the streaming scenario, the details are presented in the action plan `moving_ifttt.json` in the folder `action plan`.

The same setup is used in the moving data scenario in the packages `scenario.ifttt.moving`. An additional module `data_generator.py` is run before other modules to generate the data for moving. Then, the executing sequence is to run `setup.py`, `run_recipe.py`, and finally `run_scenario.py`. The result is also put in the folder `result`.

The details of running both the scenarios are presented in the `README.md` of the repository.

B. Result

Both results of the two scenarios are plots of the data in those scenarios. Two bivariate normal distributions with different kernels are plotted in the streaming scenario. Meanwhile, the plot of the moved data, a single bivariate normal distribution, is plotted in the moving data scenario. The results are shown in Figure 4 and Figure 5. These results show that the proposed system can be applied in these hypothetical scenarios with simulated data. However, these hypothetical scenarios are very close to the scenarios in the real-world, modern data science workflow. The plan to properly evaluate this system is laid out in the next section.

VII. Conclusion and Future Works

This pilot study has demonstrated simple concepts to realize a HITL system for modern data science workflows. The concepts of flow, action model, action plan, and communication methods are presented. Those concepts allow a simple and flexible HITL system to be implemented to support modern data science workflows easily. A preliminary system based on the *if-this-then-that* action model is implemented to demonstrate how those concepts work in practice, especially the action model and action plan. Streaming and moving data scenarios are implemented to demonstrate the *if-this-then-that* action model. Specific action plans can be easily derived from the action model specification in JSON format to fit each particular scenario. The action plan's usage is relatively easy and flexible by only specifying the details of the action plan in JSON format. The program only has to parse the JSON and execute the plan.

Nevertheless, this study has some significant drawbacks. One of them is the lack of a variety of action models. Even the *if-this-than-that* action model only supports a limited number of operations. Furthermore, due to time constraints, the experiments are only hypothetical scenarios and have not yet been implemented in a real-life modern data science workflow. Hence, subsequent iterations of the study should focus on implementing the action models that can be used in real-life the modern data science workflows, implement those action models, and extend the action models *if-this-then-that* to allow

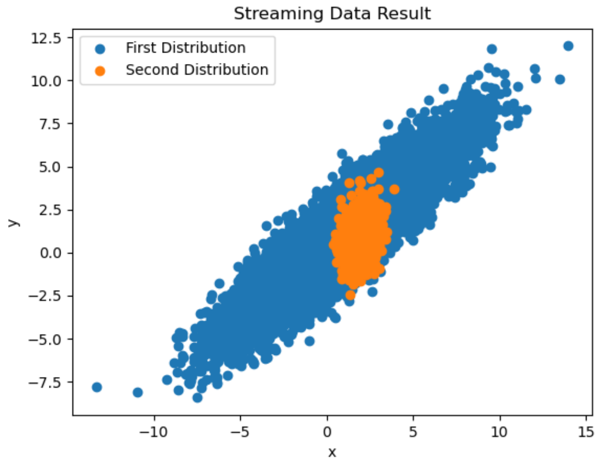

Fig. 4. Streaming data scenario result

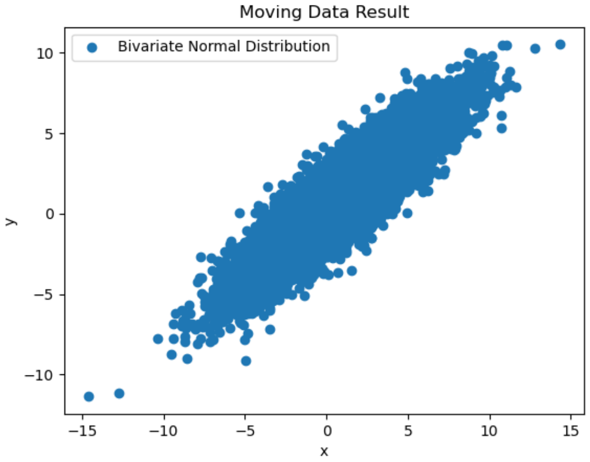

Fig. 5. Moving data scenario result

for more types of operations and conditions.

A plan very soon is to apply the current system in a real-world data science workflow, such as workflows in computational astrophysics or computational biology, to name a few. Then, users' productivity will be measured before and after applying the proposed system to quantify the system's differences and utilities. Moreover, surveying the users in these workflows before and after applying the systems will also bring invaluable perspectives.

REFERENCES

[1] L. F. Cranor, "A framework for reasoning about the human in the loop," in *Proceedings of the 1st Conference on Usability, Psychology, and Security*. USENIX Association, 2560 Ninth St. Suite 215 Berkeley, CA, United States, 2008, p. 1–15.

[2] M. S. Wogalter, *Handbook of warnings*. CRC Press, 2006.

[3] M. S. Wogalter, D. DeJoy, and K. R. Laughery, *Warnings and risk communication*. CRC Press, 1999.

[4] M. S. Wogalter, "Communication-human information processing (c-hip) model," in *Forensic Human Factors and Ergonomics*. CRC Press, 2018, pp. 33–49.

[5] D. A. Norman, *The psychology of everyday things.* Basic books, 1988.

[6] J. Reason, *Human error*. Cambridge university press, 1990.

[7] D. Xin, L. Ma, J. Liu, S. Macke, S. Song, and A. Parameswaran, "Helix: accelerating human-in-the-loop machine learning," *arXiv preprint arXiv:1808.01095*, 2018.

[8] ——, "Accelerating human-in-the-loop machine learning: challenges and opportunities," in *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, 2018, pp. 1–4.

[9] D. Xin, S. Macke, L. Ma, J. Liu, S. Song, and A. Parameswaran, "Helix: Holistic optimization for accelerating iterative machine learning," *Proceedings of the VLDB Endowment*, vol. 12, no. 4, 2020.

[10] ——, "Helix: Holistic optimization for accelerating iterative machine learning," *arXiv preprint arXiv:1812.05762*, 2018.

[11] C. Zhang, "Deepdive: a data management system for automatic knowledge base construction," Ph.D. dissertation, The University of Wisconsin-Madison, 2015.

[12] E. R. Sparks, S. Venkataraman, T. Kaftan, M. J. Franklin, and B. Recht, "Keystoneml: Optimizing pipelines for large-scale advanced analytics," in *2017 IEEE 33rd international conference on data engineering (ICDE)*. IEEE, 2017, pp. 535–546.

[13] S. Amashukeli, R. Elshawi, and S. Sakr, "ismartml: An interactive and user-guided framework for automated machine learning," in *Workshop on Human-In-the-Loop Data Analytics 2020 (HILDA*, 2020.

[14] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, "Vega-lite: A grammar of interactive graphics," *IEEE Transactions on Visualization & Computer Graphics (Proc. InfoVis)*, 2017. [Online]. Available: http://idl.cs.washington.edu/papers/vega-lite

[15] Y. Wu, J. M. Hellerstein, and A. Satyanarayan, "B2: Bridging Code and Interactive Visualization in Computational Notebooks," in *ACM User Interface Software & Technology (UIST)*, 2020. [Online]. Available: http://vis.csail.mit.edu/pubs/b2

[16] A. Satyanarayan and J. Heer, "Lyra: An Interactive Visualization Design Environment," *Computer Graphics Forum (Proc. EuroVis)*, 2014. [Online]. Available: http://vis.csail.mit.edu/pubs/lyra

[17] L. Wilkinson, "The grammar of graphics," in *Handbook of computational statistics*. Springer, 2012, pp. 375–414.

[18] H. Wickham, *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. [Online]. Available: https://ggplot2.tidyverse.org

[19] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer, "Reactive vega: A streaming dataflow architecture for declarative interactive visualization," *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2016. [Online]. Available: http://idl.cs.washington.edu/papers/reactive-vega-architecture

[20] W. Spoth, P. Kumari, O. Kennedy, and F. Nargesian, "Loki: Streamlining integration and enrichment," *Human in the Loop Data Analytics*, 2020.

[21] X. Zhang, X. Ge, and P. K. Chrysanthis, "Interactive view recommendation with a utility function of a general form," in *HILDA*, 2020.

[22] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth *et al.*, "Knowledge discovery and data mining: Towards a unifying framework." in *KDD*, vol. 96, 1996, pp. 82–88.