CoAutoML: User Interface Framework for Machine Learning Novices using LLM-based AutoML and Test-Driven Machine Teaching

Valeria Starkova
vs9@williams.edu
Williams College
Williamstown, MA, USA

Iris Howley
ikh1@williams.edu
Williams College
Williamstown, MA, USA

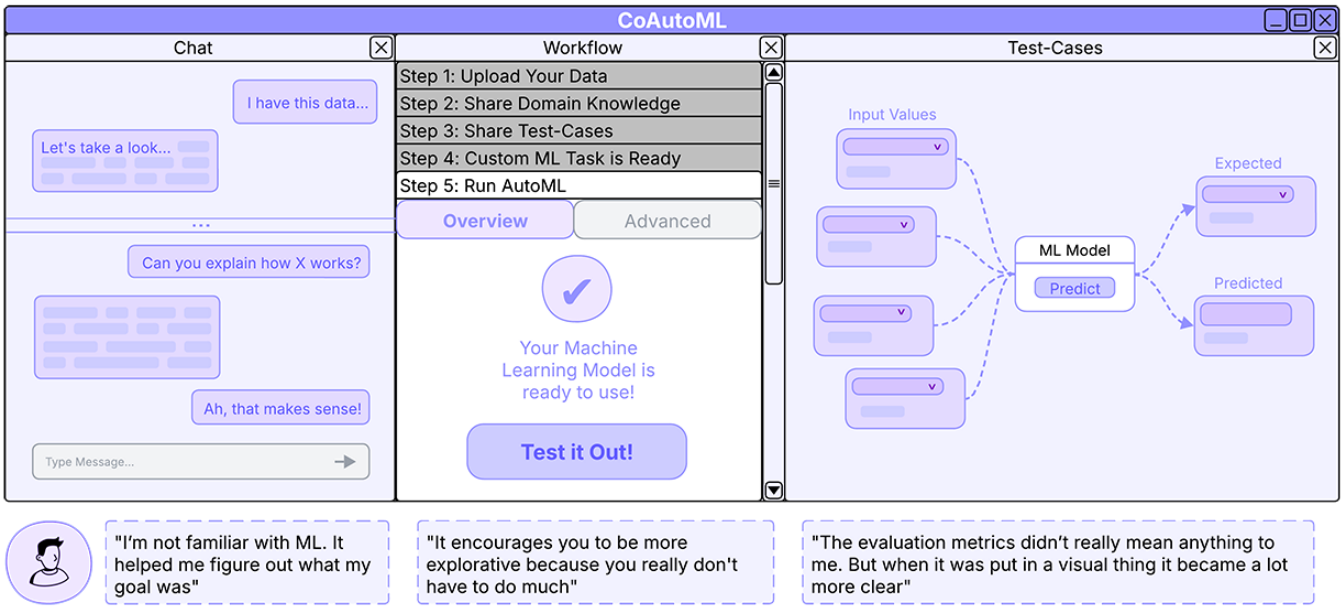


Figure 1: The CoAutoML Framework. Our system integrates three core components: (1) a conversational LLM interface (left) for ML task guidance, (2) a visual workflow panel (center) displaying step-by-step progress (data upload → domain knowledge → test-cases → ML task formulation → AutoML process), and (3) an interactive test-cases flowchart (right) aligning with novices' mental models of input-output mapping.

Abstract

An explosion in automated machine learning (AutoML) tools has led to numerous back-end frameworks enabling users with machine learning expertise to leverage the power of Machine Learning (ML). However, to ensure that ML tools are openly accessible to all who stand to benefit from their predictive and analytical powers, we must examine how true novices without ML knowledge interact with AutoML tools, perceive ML, and form their mental models of ML processes. We achieve this goal with our user-facing framework that combines the understandability of conversation with Large Language Models (LLMs) and the interface scaffolding necessary to support true machine learning novices in building their own models. We then evaluate the effectiveness of our framework in a user study. Results show that our ML-novice participants felt confident performing ML tasks independently, citing the tool's ease of use and its ability to help them formalize their ML goals.

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**; *Machine learning*; • **Human-centered computing** → *Empirical studies in HCI*; **Interactive systems and tools**.

Keywords

AutoML, Intelligent Interfaces, Machine Teaching, LLMs, User Interfaces

IUI '26,

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/3742413.3789153

ACM Reference Format:
Valeria Starkova and Iris Howley. 2026. CoAutoML: User Interface Framework for Machine Learning Novices using LLM-based AutoML and Test-Driven Machine Teaching. In *Proceedings of March 23–26, 2026 (IUI '26).* ACM, New York, NY, USA, 28 pages. https://doi.org/10.1145/3742413.3789153

1 Introduction

As Artificial Intelligence (AI) moves towards becoming a General Purpose Technology (GPT) of the 21st century [3, 25], the ability to create AI systems becomes more confined to a smaller set of people—AI/ML experts. Many steps of a machine learning pipeline are highly iterative and could be automated. As a result, Automated Machine Learning (AutoML) [32] emerged with the goal of automating parts of the ML process, including data processing, feature engineering, model selection, and hyper-parameter optimization. The ultimate goal of AutoML is to automate all steps of machine learning, from dataset processing to model deployment. This would have the effect of easing the job for AI experts, as well as democratizing AI by enabling non-experts to build their own ML models.

However, most current AutoML tools such as Auto-WEKA, Google AutoML, and H20 AutoML [4, 5, 23] primarily cater to the ML expert audience. They overlook the unique needs of non-experts, those who may possess domain knowledge but have little ML expertise. Opening up this ability to everyday users could allow citizen scientists, journalists, teachers, etc. to solve issues in their local communities. However, studies reveal a low user adoption of AutoML tools [28] even among experts. Recognizing this challenge, the HCI community has recently started to advocate for a more "Human-Centered AutoML Paradigm" to address the limitations of existing ML systems by empowering non-experts to engage directly in model development [1, 15, 19]. Furthermore, few research efforts have bridged the gap between novices and ML in practice. Existing studies primarily focus on users with prior technical backgrounds or propose conceptual guiding frameworks without building empirical tools to test their effectiveness [15, 33, 35]. To that end, we run a baseline study showing that ML novices cannot use existing AutoML systems and then build an empirical AutoML framework specifically catered to a non-expert's mental model of Machine Learning. By integrating Large Language Models (LLMs) into our AutoML tool, we enable non-experts to communicate their requirements directly to the platform, reducing reliance on intermediaries such as data scientists.

Building on principles from human-centered AutoML and recent advancements in LLM-based AutoML, we propose CoAutoML, a novel framework that integrates mixed-initiative LLM interfaces, Test-Driven Machine Teaching (TDMT), and LLM-based AutoML tools. The proof-of-concept system in this article demonstrates the feasibility of creating human-facing AutoML solutions for novice users. This project marks a move toward more accessible AI, allowing people from all walks of life to engage in ML-creation process and solve issues in their local communities without extensive ML expertise. Furthermore, by allowing non-expert users to deeply engage in and understand how an ML algorithm works in an accessible manner, we can equip them with the necessary critical lens to navigate the modern world where ML has become so ubiquitous.

2 Prior Work

2.1 Automated Machine Learning

Conventional ML processes can be divided into four primary phases [7]: data preparation, feature engineering, model creation, and model evaluation. In the first stage, data scientists carry out a number of tasks to clean and pre-process the raw data. This could involve encoding the categorical variables, normalizing, handling missing values, and more. Since the quality of the input data directly affects how well the developed models perform in the future, this is one of the most important phases in machine learning. The next step is feature engineering, which involves extracting relevant features from the data, transforming them, or creating new features. The third stage is model generation, during which users explore a range of machine learning algorithms to identify the most suitable one. This step also involves tuning the hyperparameters for some models, which cannot be learned and must be hand-picked by a human. The last step is model evaluation, when data scientists evaluate the ML models' performance using evaluation metrics such as accuracy, precision, recall, F1 score, etc. to examine how well the models perform on unseen data.

Throughout this process, data scientists frequently perform all these steps, often by trial and error. For example, hyperparameter optimization requires iteratively testing combinations of parameters to find the best-performing set. These problems make conventional machine learning processes human resource-intensive.

AutoML addresses this inefficiency by automating the complex process of hyperparameter tuning and algorithm selection [32]. The goal of AutoML is to identify the combination of algorithms and hyperparameters, referred to as the *pipeline*, that minimizes a k-fold cross-validated loss [18].

Over the years, the scope of algorithmic AutoML expanded to include automatization of other stages of the ML pipeline, such as feature engineering and model deployment, eventually leading to end-to-end automation of the entire ML pipeline [30], now present in solutions like Google Cloud AutoML, Microsoft Azure Machine Learning, and AutoKeras [10]. This leap forward, with AutoML's promise of automating all steps of an ML pipeline, allows data scientists to find optimal models much easier and non-experts to be able to engage in ML without requiring extensive expertise.

2.2 Human-Centered AutoML

Current discourse on Human-Centered AutoML primarily addresses machine learning practitioners or non-experts with some coding/ML experience [15, 22, 35]. However, they overlook the needs of *true novice users*—those with zero programming and machine learning experience.

Nevertheless, insights from these papers offer valuable perspectives on human-centered AutoML that can inform our own research. For instance, prior work identified three key challenges users face with real-world AutoML tools: *customizability*, *transparency*, and *privacy* [22]. Similarly, Lindauer et al. further emphasize that improving the interpretability of AutoML, designing better user interfaces, and extending the human-centered automation paradigm to other aspects of the data science workflow are critical next steps [15]. Through interviews with 16 ML practitioners, Xin et al. found that full automation is neither a requirement nor a

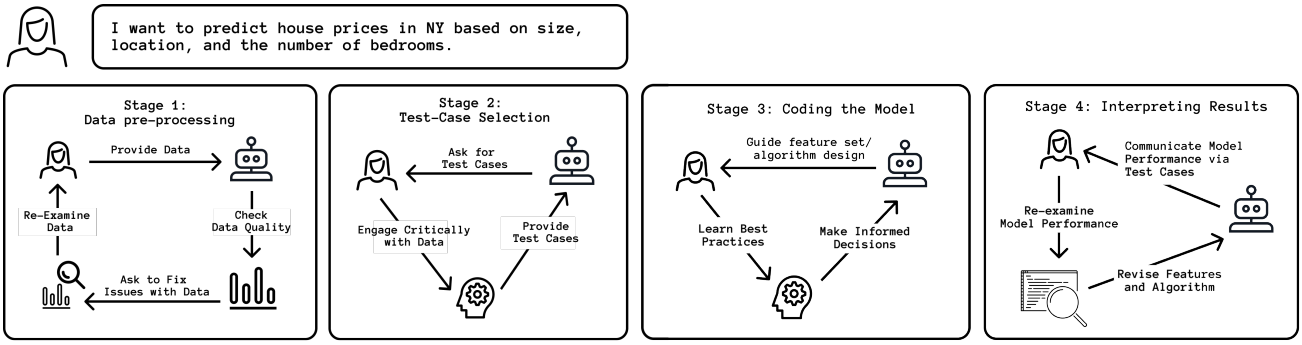


Figure 2: Test-Driven Machine Teaching Interaction Flow adapted from [35]

desired outcome; instead, the focus should be on creating "human-compatible" tools that enhance transparency and trust while giving users a sense of agency. Their findings advocate for an *interactive, human-in-the-loop approach* to strike the delicate balance between human control and automation. Additionally, they emphasize that AutoML tools need to be *adaptable* to users' expertise levels, both in terms of the UI and the jargon used. They advocate for a back-and-forth, *interactive dialog* between a human and the machine to address each others' blind spots.

Through conducting interviews and surveys with around 100 non-experts (those who lack formal ML training, but actively build ML models), Yang et al. identified several key insights [35]:

(1) Non-experts rarely start with a predefined problem but instead have data ready at hand and explore how ML can help them learn from the data.
(2) They often have a deep understanding of the problem as it relates to their work and hobbies.
(3) Their mental model of ML is largely based on input-output mapping: data goes in, predictions come out.
(4) Non-experts tend to over-rely on accuracy as the sole measurement of success, overlooking other important factors such as robustness, overfitting, etc.

Additionally, they identified several challenges faced by non-experts when using AutoML systems:

(1) Non-experts often struggle to formulate a clear and feasible ML goal.
(2) Their main strategy to improve the model's performance was to add more data; none of the interviewed non-experts attempted to perform feature engineering.
(3) Although non-experts didn't fully understand the concept of "features," they recognized their value through their impact on the model's performance.

These findings underscore the need to design accessible AutoML tools for novice users that closely match their mental models of ML and accommodate non-experts' knowledge gaps.

2.3 Test-Driven Machine Teaching (TDMT)

To address these challenges and cater to non-experts' mental model of ML, Yang et al. introduce the novel interaction flow, Test-Driven Machine Teaching (TDMT), to help guide HCI researchers in building tools suited for non-experts' needs [35]. TDMT (adapted in Figure 2) encourages users to define test cases before building an ML model. This approach encourages non-experts to think critically about their problem, fostering incremental learning both of their intentions and the mental model of the ML system. Through a hand-picked process of providing the test cases, non-experts can begin to formalize their own ML goal. After test cases are provided, the tool suggests feature engineering and machine learning algorithms based on the data and alerts users of data quality issues. Upon training the model, the tool shows predictions based on the provided test cases. This differs from other AutoML tools that usually show visualizations and statistics, which require ML expertise to interpret. Instead, the test cases users provide are personal to them, and therefore easy for them to understand and evaluate. Most importantly, this test-driven approach prevents users from over-relying on the accuracy metric to evaluate the performance of a model, a common pitfall. To our knowledge, there does not yet exist an implemented application of the TDMT approach in an interactive AutoML tool for non-experts.

The work on "Machine Teaching" does look at supporting users with minimal to no ML expertise to build supervised models, with some success [29]. However, the authors' MATE system uses a "Wizard of Oz" approach in which all the ML guidance is provided by a human expert observing a live feed of participants' behavior. This suggests that systems can support novices in building machine learning models, but with LLMs it might be possible to build real versions of working machine teaching.

These papers strengthen the need for developing accessible AutoML solutions for complete novices as a necessary and timely pursuit. By building on the extensive research conducted in these connecting areas, there is a promising opportunity to carve out a nascent branch of AutoML research specifically focused on supporting true novices so that all may access the power of ML tools.

2.4 LLMs as Interfaces for AutoML

LLMs offer fertile ground for AutoML with natural language interfaces with emerging studies exploring the synergy between LLMs and AutoML [24]. The authors identify several promising avenues, including using LLMs as a bridge between users and complex algorithmic interfaces. LLMs provide the necessary knowledge of the

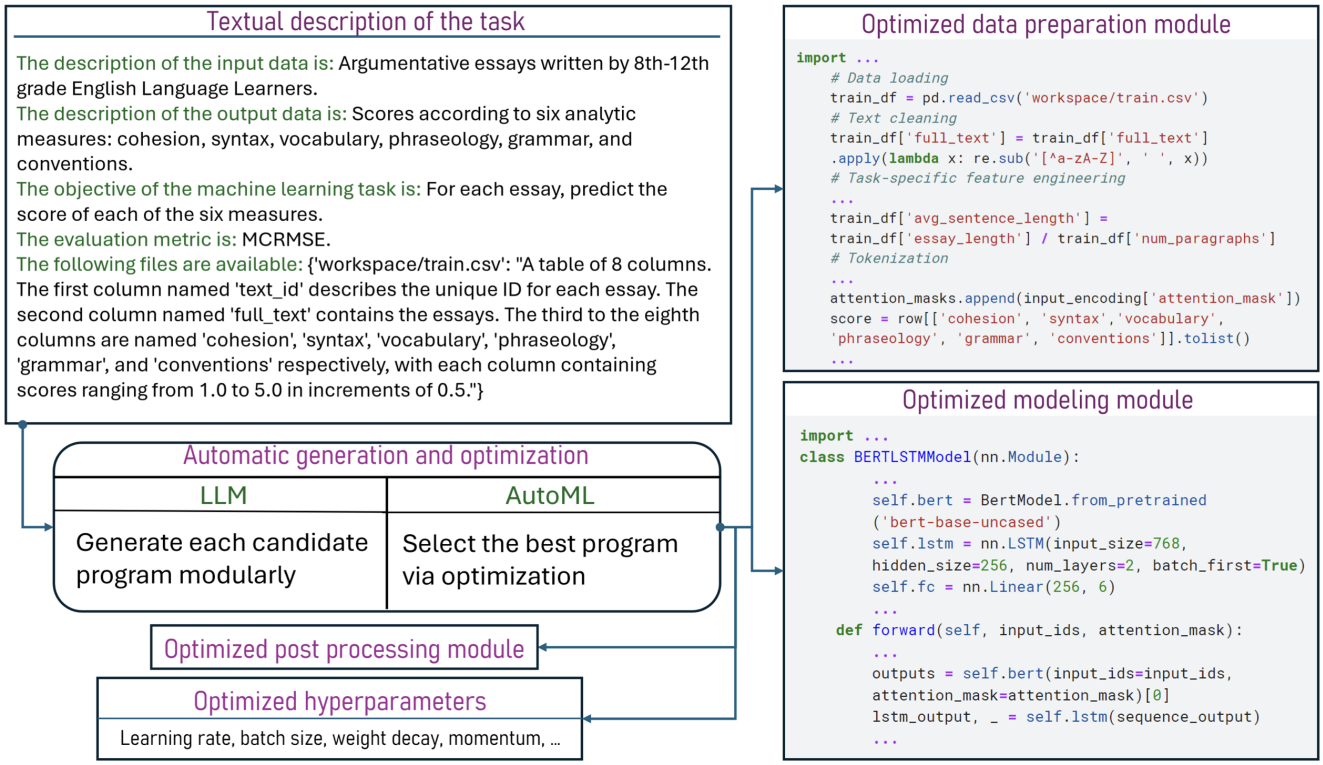


Figure 3: Text-to-AutoML tool example usage. Screenshot is from [34].

best machine learning practices embedded in the extensive data they were trained on, and AutoML helps to optimize the code by algorithmically searching for the most optimal solution.

With the increasing popularity of LLMs, there has been a notable surge in research aimed at developing LLM-based AutoML tools. For example, Xu et al. developed a promising AutoML tool using LLMs [34]. This tool performs a number ML tasks, generating optimal ML models based on textual JSON input shown in 4. Users are prompted to enter all the necessary information in text format. This response is still inaccessible to novice users, as they might not fully comprehend these important aspects of AutoML settings.

```
{ "input data": "Argumentative essays written by 8th-12th grade English Language Learners",
  "output data": "Scores according to six analytic measures: cohesion, syntax, vocabulary",

  "task objective": "For each essay, predict the score of each of the six measures",
  "evaluation metrics": "MCRMSE",
  "files": {"test_workspace/train.csv": "A table of 8 columns, The first column named 'text_i
  },
  "key": "...",
  "GPT version": "3.5",
  "workspace": "test_workspace"
}
```

Figure 4: Example JSON code the user must specify for the AutoML system in [34].

Other LLM-AutoML projects face similar shortcomings. For example, the work by Zhang et al. [36] is able to solve complex ML tasks by dynamically training models with optimized hyperparameters using LLMs. To achieve this, the user-provided input paragraph is fed into an LLM to generate a prompt that establishes the AutoML pipeline. LLMs handle all steps of data preprocessing, model architecture design, hyperparameter tuning, and the prediction log. However, the input paragraph consists of a detailed "Data Card," which includes data description such as data name, input data type, label space, and evaluation metric; a "Model Card," which contains a description of the model name, model structure, model description, architecture hyperparameters, and evaluation metric. Additional requests may also be included, such as constraints like "the inference time smaller than 10 FPS." Like Xu et al., this project and many others such as AutoM3L [17] and AutoGluon [31] lack an intuitive user interface and is designed for ML experts, requiring technical knowledge to provide extensive models and data descriptions [34].

To begin training the model, users must have clear goals from the beginning, although non-expert users rarely start with a predefined goal in mind and even have a hard time understanding the meaning of features or tend to over-rely on the accuracy metric [35]. Therefore, such extensive and accurate descriptions would alienate novice users when training the model. Instead, we need something that integrates a human-in-the-loop approach to encourage incremental learning and gently steer the user towards understanding their goal and ML task. There is a growing body of research on AutoML with LLMs that often shares the same shortcomings [8, 17, 21, 26, 27]. Therefore, the key question to consider is: how can we address this issue to bridge the gap between current LLM-based AutoML tools and a human-centered AutoML paradigm?

3 Design Motivation

3.1 Baseline Study

To motivate the development of our own system, we conducted a baseline assessment to see how effectively non-experts could independently define their ML objective and specify input/output data in the format required by LLM-based AutoML tools. As our system is intended for users who have domain knowledge, but lack awareness of machine learning, our target users for this baseline study generally do not have a deep understanding of data requirements for building ML models nor ML evaluation metrics. As part of the user study described in Section 6, we recruited 16 participants who self-identified as novice users through a pre-questionnaire (demographics details in Table 6). Before interacting with our CoAutoML system, participants completed a short form which prompted them to delineate the same categories of information required by the Text-to-AutoML system described in [34]. We adapted the phrasing for each question from the JSON description prompts used in [34] as shown in Table 1.

Table 1: Prompts for gathering baseline interactions with Text-to-AutoML System.

Input data	Please describe your input data. The input data contains the features that are the characteristics or attributes of the dataset. These features provide information that the model uses to make predictions.
Output data	Describe the expected output data, also known as 'labels' or 'targets', which represent the desired prediction or outcome associated with each input example.
ML Objective	Describe the task objective, which describes the goal of the machine learning task.
Evaluation metrics	Describe the evaluation metrics, which are numerical measures of how well the model's predictions match the desired outcome.
Files	Describe the available files given in your task.

Our baseline assessment revealed significant barriers preventing novice users from using existing LLM-based AutoML tools. For example, no participants correctly identified the evaluation metric, descriptions of the dataset lacked technical depth necessary to use the Text-to-AutoML tool, input and output data descriptions were largely overly vague and unusable with the Text-to-AutoML tool.

Table 2 further illustrates this gap by comparing expected Text-to-AutoML inputs with most representative novice responses.

3.2 Design Implications

Based on prior research and the challenges faced by novice users in using Text-to-AutoML, we categorized the following key design goals (DG) to address in our own system, summarized in Figure 5.

DG1: LLMs as a Mixed Initiative Interface Early work on mixed-initiative systems by Horvitz [9] laid the groundwork for balancing human and machine agency principles that find renewed relevance in the context of LLMs. Our system takes inspiration from Horvitz's framework by adapting its core ideas to the needs of ML non-experts. In the context of AutoML, prior research established that non-experts rarely start with a predefined ML task [35] but rather have data they're looking to learn from. Therefore, it is crucial that our platform infers user intent over time through scaffolding and engaging in iterative dialog to resolve lingering user uncertainties. This approach radically differs from other LLM-based AutoML tools that require users to define the modeling task upfront [8, 17, 21, 26, 27, 34].

Following the best Mixed Initiative principles, users of our system should be able to modify any part of the workflow at any time, whether that means editing domain knowledge items, adjusting test cases, or tweaking the generated ML task or model use. The system must update its internal state accordingly and maintain a shared context throughout these interventions.

In response to the adaptability and transparency challenges highlighted in [22, 33], our system makes the inner workings of AutoML fully transparent. However, to avoid overwhelming non-expert users, technical complexity is initially kept behind a default interface. Users can decide to reveal deeper layers of detail, including downloadable source code, enabling inspection for deeper understanding, when desired. The AI should also default to communicating in non-technical language unless the user asks otherwise.

DG2: Domain Knowledge Sharing. Kazemitabaar et al. [11] showed how editable AI-generated assumptions about data columns can improve both code generation and serve as a helpful mechanism for sharing domain knowledge from human to AI. In our system, we adapted this approach for AI-assumptions and incorporated them into the LLM-based AutoML code generation. Our baseline showed novices struggled to describe their data in depth and technical detail. The domain knowledge sharing stage addresses this by first generating a set of assumptions for each column, highlighting things such as the column's meaning, importance, and role, and providing scaffolding for novices unfamiliar with feature engineering concepts. Users can accept, modify, or delete generated assumptions, maintaining agency while reducing cognitive load.

DG3: Test-Driven-Machine-Teaching (TDMT) Integration. Yang et al. [35] identified that non-experts' mental model of ML largely centers around input-output mapping. They proposed Test-Driven-Machine-Teaching as a paradigm that matches novices' expectation of ML but did not implement the actual system. CoAutoML is the first to operationalize the TDMT framework outlined by Yang et al. While the authors did not propose a concrete UI design for TDMT, we envision TDMT as an interactive flowchart, similar to Google's Teachable Machine [2], where input nodes feed into an output node mirroring the intuitive "data goes in, predictions come out" mental model of non-experts uncovered in [35].

Table 2: Comparison of example Text-to-AutoML provided prompts from paper [34] and corresponding novice responses to the same prompts.

Question	Sample Text-to-ML Prompt [34]	Sample Novice Response
Input Data	Various features about various locations in Boston, such as per capita crime rate and proportion of residential land zoned for lots over 25,000 sq. ft.	"The input data is the information we have about each house such as its number of bathrooms or the area it's located in" (P7)
Output Data	The housing price of each location.	"A number? or like a prediction based on how good the investment would be" (P3)
ML Objective	For each location in Boston, predict the housing price of the location.	"task is to evaluate the different characteristics and predict the price/viability of investment (but I am not sure)" (P2)
Evaluation Metric	RMSE	"The evaluation metrics would be something like a difference of how close it is to an expected average" (P4)
Files	`workspace/data.csv`: A table of various features and housing prices across different locations in Boston. The table has fourteen columns. The first thirteen columns of the table are features used for prediction. The last column named 'MEDV' is the output/target of the prediction (housing price).	"a data table with different transactions" (P3)

4 CoAutoML

Building on our design principles from Section 3.2, we propose CoAutoML, a novel framework that integrates mixed-initiative LLM interfaces, Test-Driven Machine Teaching (TDMT), and LLM-based AutoML tools. The proof of concept system in this article demonstrates the feasibility of creating human-facing AutoML solutions for novice users. CoAutoML's most distinctive feature is its ability to cater to non-experts' mental models of Machine Learning to extract critical information necessary—such as input/output data, ML objective, evaluation metrics, file descriptions—to feed into LLM-based AutoML tools like [8, 17, 21, 26, 27, 34].

The user interface is structured around three primary components: chat, workflow, and test cases. The *chat section* serves as the user's main entry point and support system. The *workflow section* encapsulates the stages of Machine Learning in a step-by-step process revealed sequentially: (1) data upload, (2) domain knowledge sharing, (3) test case selection, (4) custom ML task review, (5) and a final AutoML code generation phase. These two components combined serve to address **DG1** and **DG2**.

The final phase (5) is also split into two tabs adapted for users' skill levels: "Overview" abstracts the technical details and shows informative status updates while the AutoML tool runs in the background, and the "Advanced" view presents full output from the

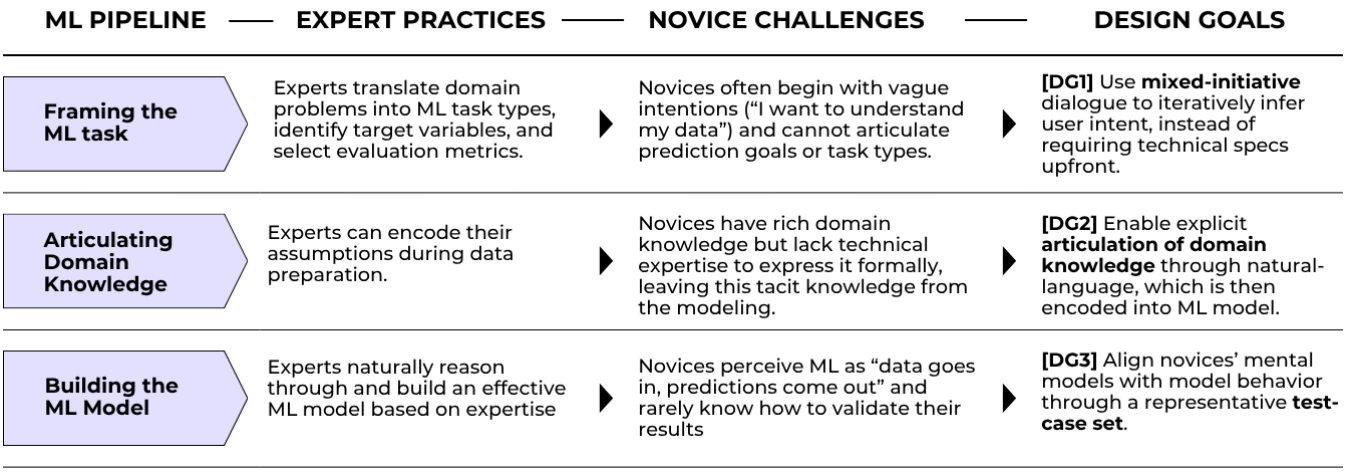


Figure 5: Summary of design goals based on ML stage, expert practices, and novice challenges.

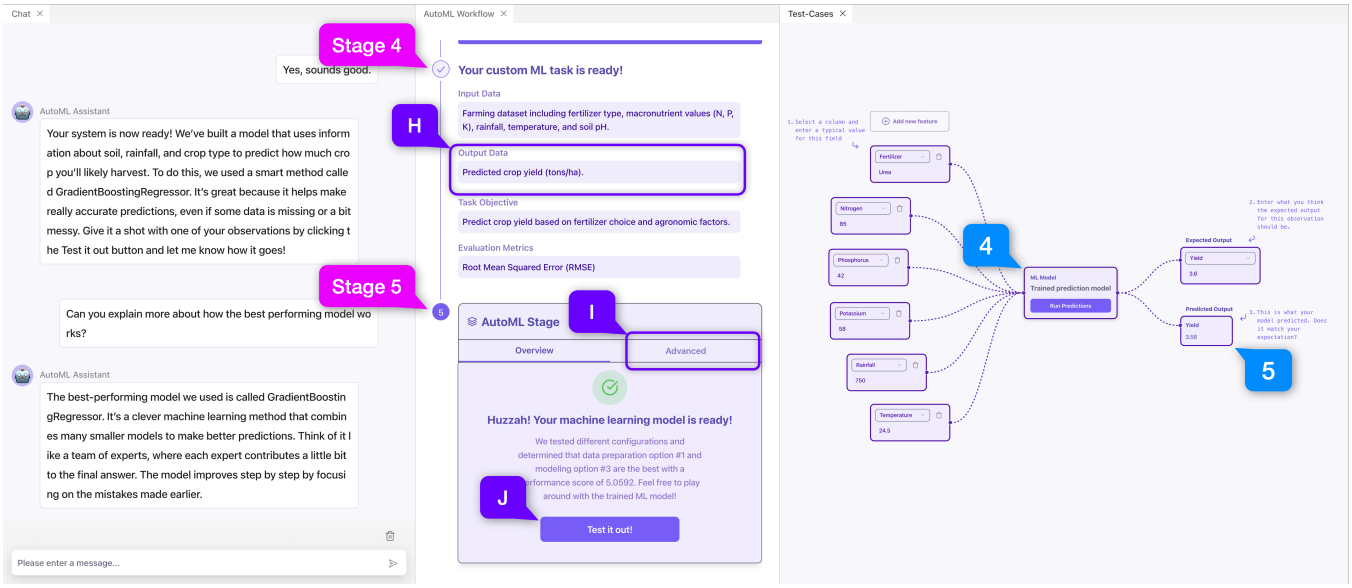


Figure 6: The CoAutoML full-scale interface showing three integrated panels. See Table 3 for detailed descriptions.

AutoML backend, shows optimization search history, highlights the best performing model and data preparation strategies, and provides access to download all generated files. The most distinctive feature of CoAutoML is its real-world implementation of TDMT as discussed in **DG3**, which is represented as a section on the right tab pictured in Figure 8.

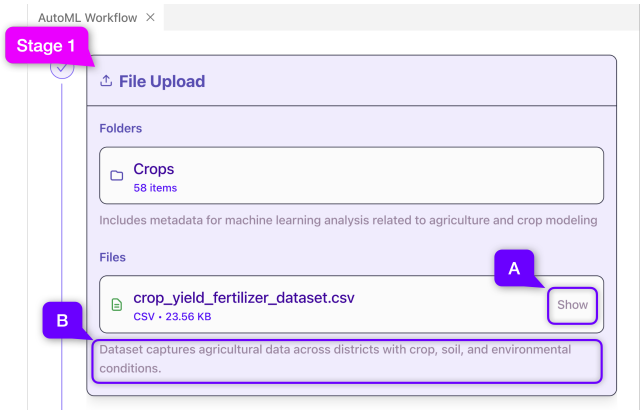
John is an experienced farmer who has grown crops for over 20 years. Recently, he started collecting structured data about his harvest—soil quality, moisture levels, crop yields, and fertilizer usage hoping to better understand how different variables affect his results. Over time, John has built up spreadsheets full of this data, but he is never quite sure what to do with it, until he hears about the CoAutoML desktop application and decides to give it a try.

4.1 Core Features and Usage Scenario

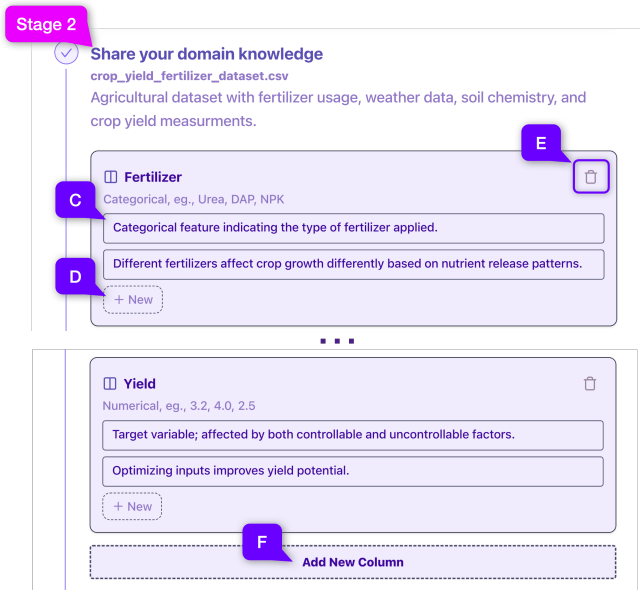
Below, we explore the core features of our tool through a typical interaction with a user persona who has no coding experience.

Table 3: Description of CoAutoML interface panels (Figure 6).

Panel	Description
Chat (Left)	Users converse with the AutoML Assistant, which explains model results in accessible language.
Workflow (Center)	Stage 1: Data Upload. For more details refer to Figure 7a. Stage 2: Domain Knowledge Sharing (Assumptions). For more details refer to Figure 7b. Stage 3: Test-Cases (Observations). For more details refer to Figure 8. Stage 4: Editable task specifications (H) including input/output data, task objective, and evaluation metrics for the Text-to-AutoML tool. Stage 5: AutoML results with Overview and Advanced tabs (I), plus a button to run the trained model against test-cases (J). Refer to Figure 9 for more details about the Advanced Tab.
Test-Cases (Right)	Implements Yang et al.'s TDMT [35] via a node-based interface, where users can: **(1)** Input test-cases during Stage 3 (e.g., Fertilizer: Urea, Nitrogen: 85) (Figure 8, 2) **(2)** Specify expected output (Yield: 3.6) (Figure 8, 3) **(3)** Run the trained ML model (4) **(4)** Compare predicted output (Yield: 3.58) against expectation (5)



(a) Stage 1: Data Upload



(b) Stage 2: Domain Knowledge

Figure 7: The workflow panel (zoomed in) guiding users through key stages. (a) Stage 1: Users can upload their data, preview the dataset using the "Show" button (A), and view the short file/folder blurb (B). (b) Stage 2: Domain Knowledge displays column cards (C) showing LLM-generated assumptions. Users can add new domain knowledge entries via the "+ New" button (D), remove columns using the delete button (E), or add new columns via the "Add New Column" button (F).

Data Input and Analysis. John downloads the app, launches it, and is greeted by a simple interface: a chat window on one side, and an initial panel for uploading his data on the other. John uploads his comma-separated values files, and the assistant immediately begins parsing the contents, generating some initial information based on the uploaded files (Figure 7, Stage 1, B). The assistant lets him know it recognized crop-related information on the chat panel and created a few assumptions. The assistant then prompts John to review and edit the assumptions based on his domain knowledge.

Input Assumptions. John reads over the generated assumptions for each column, making sure that they align with what he had in mind, and fine-tunes them drawing from his deep domain expertise (Figure 7, Stage 2). He can edit or delete generated assumptions by hovering over the field (Figure 7, Stage 2, C, D). Additionally, he may delete entire columns or add missing ones using E and F. Once he's satisfied, he lets the assistant know. The tool then prompts John to input a set of real-world examples from his farm. Throughout the process, the assistant clearly communicates what is happening, what comes next, and why, so John feels confident and in control.

Test-Cases. The app highlights the button where John can add his observations (Figure 7, Stage 3 G.1). Once clicked, a new section opens on the screen showing a diagram: input nodes (features) flow into a single output node (target), mimicking the familiar mental model of ML systems where "inputs go in, outputs come out" [35] (Figure 8). On one side, he sees the input columns tied to the assumptions he made earlier (Figure 8, 2); on the other, there is an output node inviting him to enter an expected output (Figure 8, 3). Visual labels guide John through the diagram, explaining each step (Figure 8, 1).

He pauses to reflect on what he wants to achieve with this data, then begins entering values for a typical crop that thrives under the Urea fertilizer and choosing the yield type as the expected output he wants to predict. Each time he adds a new test case, a progress bar updates on the workflow panel (Figure 8, Stage 3). When sufficient test-cases are added, John is prompted to submit his examples to the assistant.

Custom ML Task Generation. The assistant analyzes the examples and generates a custom machine learning task based on John's observations (Figure 6, Stage 4). It reiterates the machine learning objective in a user friendly way in the chat window, asking John to confirm if he wants to proceed. The assistant's message uses orienting phrases like "in the "Workflow section", or "to the left of your screen" when referring to UI components on the screen. If John wants to make adjustments, he can talk to the assistant to adjust the task. Once confirmed, the app enters the AutoML stage, where it automatically creates custom machine learning pipelines and code modules tailored to his goal (Figure 6, Stage 5). While the task is processing on the backend, a loading screen keeps John informed with friendly updates.

AutoML Process. After a few minutes, training completes, and John sees a blue button inviting him to test his observations and view the predicted output (Figure 6, Stage 5, J). The original observations on the test-cases panel are updated with a new node in the middle showing the model (Figure 6, 4), and two nodes coming out of it, representing the predicted and expected outputs (Figure 6, 5), allowing for side-by-side comparison. John can quickly assess how well the model is performing just by looking at the

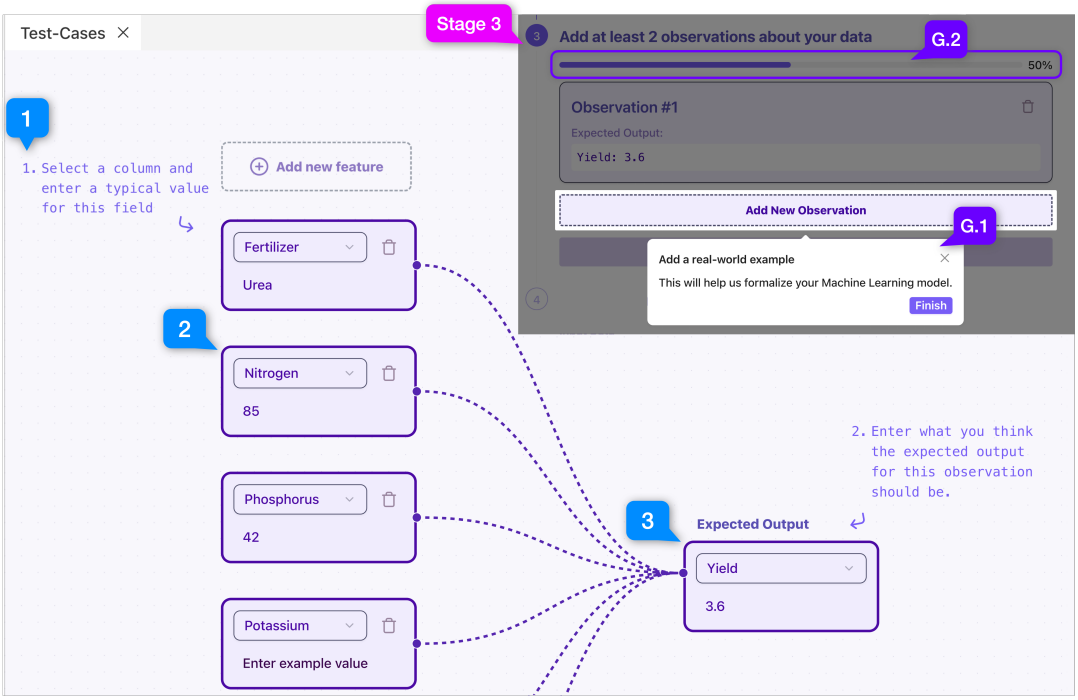


Figure 8: Test-Cases Panel implementing Yang et al.'s TDMT framework. Zoomed In. (1) Instructional labels (1) guide users in building input-output mappings, (2) Input nodes contain columns from domain knowledge items entered earlier (e.g., fertilizer usage, Nitrogen levels, so on) (2), and (3) Output node (3) capturing the target prediction (crop yield). Back in the workflow panel (Stage 3), a pop-up window appears and highlights the button to call the user to action (G.1). A progress bar (G.2) shows completion toward the minimum test-cases required and the added test-cases are presented in a compact card.

comparisons. He starts experimenting with different input values until he discovers which features most strongly influence yield prediction. John can easily adjust the inputs and re-run predictions.

He notices the "Advanced" tab and becomes curious (Figure 6, 1). He clicks on it and sees a lot of information about different models and data preparation options the system tried out, highlighting the best options and their performance (Figure 9). He doesn't quite understand what any of it means, but when he asks the assistant to explain the best-performing model, it responds with a clear breakdown. John now understands not just what worked, but why, including how many models were tested and why the current one was selected. This transparency increases his trust in the results.

5 System Implementation

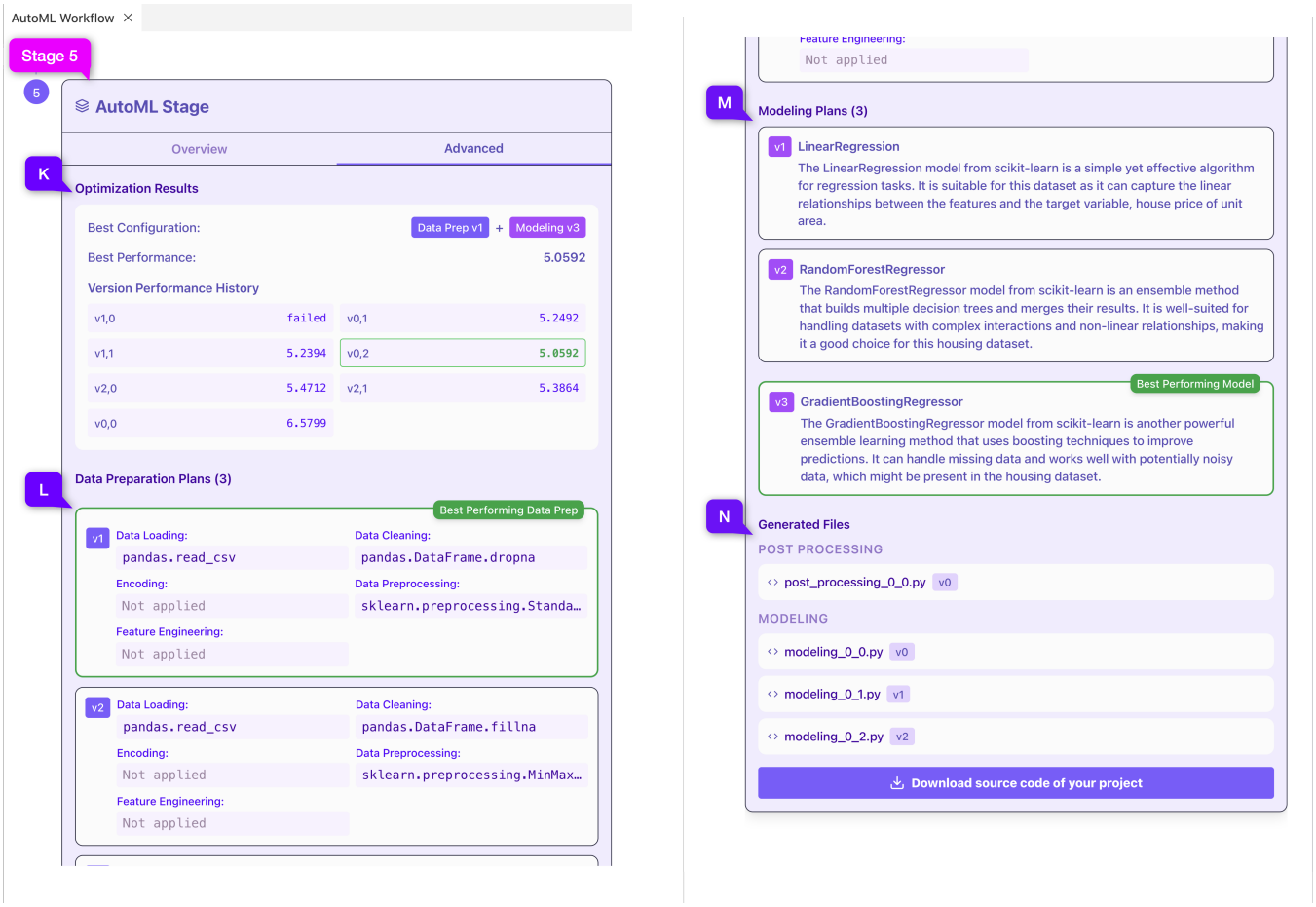
5.1 Overview.

CoAutoML is a desktop-based application written in React and supported by Electron. The backend is implemented in FastAPI and uses the new 2025 OpenAI Agents SDK, which enables a highly modular and extensible multi-agent system. The backend sends updates and receives user messages through strongly typed WebSocket messages, with HTTPS API endpoints supporting data upload, test-case submission, and prediction execution. We selected the Text-to-AutoML tool from [34] to serve as the backbone of our system for its outstanding performance compared to other LLM-based AutoML tools. CoAutoML currently supports classification and regression tasks, enabling models such as Random Forest, Gradient Boosting, and ensemble methods. The system automatically selects appropriate model families based on the identified task type. Image recognition and NLP are currently unsupported in CoAutoML, presenting a further avenue for exploration. Below, we outline the key features and architecture of our implementation illustrated in Figure 10. We provide the complete source code of CoAutoML at https://github.com/UberHowley/coautoml along with a README.md document for future reproducibility.

We used GPT-o3 for the main Orchestration Agent due to its fast performance, low latency, and strong reasoning capabilities. For each of the specialized helper agents, as well as the Text-to-AutoML tool itself, we used GPT-4o for its deeper reasoning capabilities.

5.1.1 Orchestration Agent (OA). This is the primary agent that coordinates tasks between specialized tools/agents and serves as a proxy to the user based on the outputs from the specialized agents. The Orchestrator Agent is prompt-engineered with an understanding of the general UI structure, workflow, and state updates. This enables spatial referencing in conversations:

> *"Look for the blue 'Add New Observation' button on the right side of your screen—I've highlighted it for you."*



(a) Optimization Results and Data Preparation Plans

(b) Modeling Plans and Generated Files

Figure 9: Advanced Tab of the AutoML Stage (Stage 5, zoomed in). (a) The Optimization Results section (K) displays the best configuration (combining data preparation and modeling versions), the best performance score, and a version performance history table showing all tried combinations. The Data Preparation Plans section (L) lists different preprocessing strategies (e.g., data loading, cleaning, encoding, preprocessing, feature engineering) with the best performing plan highlighted. (b) The Modeling Plans section (M) presents multiple model options (e.g., LinearRegression, RandomForestRegressor, GradientBoostingRegressor) with descriptions and highlights the best performing model. The Generated Files section (N) lists all generated Python scripts for post-processing and modeling, with a button to download the complete source code of the ML project.

Following **DG1**, the OA is instructed to help users discover their ML goal through scaffolded interactions. To that end, it was prompt-engineered to guide users through each step of the ML process, explaining what is happening, what comes next, and why it matters. It is designed to translate complex ML concepts into layman's terms and to help users stay oriented within the system. Based on user input and its own reasoning, it can call any of the helper agents described below. A key design challenge was to make the multi-agent coordination as smooth and invisible to users as possible. To that end, we employed several techniques:

For example, when the helper agents return information to the OA and that information was simultaneously updated in the UI through WebSocket, the OA is instructed to not repeat information already visible on screen. Another key challenge was making the OA synthesize information from multiple agents to keep the appearance of a "single actor" and weave a cohesive narrative. For example, the single response below integrates multiple pieces from other agents: data analysis agent, test-case integration agent, and UI element updates; however, the users perceive a single intelligent entity, not multiple agents processing data:

> "Based on the examples you provided, it looks like we'll focus on predicting crop yield using factors like soil conditions and rainfall. Does that match what you had in mind?"

Another key component was making sure the OA's responses weren't overly wordy as conventional AI agents tend to do. Thus,

the OA only introduces one concept per message and keeps a natural human-like flow:

> "Great! Your model is trained. Before we test it out, did you want to know more about how it works, or should we jump straight to trying it with your examples?"

5.1.2 Analyze Data Agent. Using both rule-based logic and output from specialized agents, this agent helps analyze the user-uploaded files and folders. This output is returned to the Orchestration Agent for reasoning about the uploaded data and for triggering the Assumptions Agent. It also sends a WebSocket message to the frontend with a signal to update the data upload component with a short blurb for each file (Figure 7, Steps 3.1 and 3.2). The generated data model is saved to the shared context for the next agent.

Example Output to OA:

```
"files": [{
  "filename": "crop_data.csv",
  "snippet":"
     | nitrogen_level | rainfall | soil_ph | crop_yield |
       |--------------|------|--------|-----------|
       | 45.3         | 120.5| 6.8    | 8.2       |
       | 32.1         | 95.3 | 7.1    | 6.7       |
       ...",
  "detailed_description": "Dataset contains
    agricultural measurements across 8 columns and 450
    rows. Features include soil composition metrics
    (nitrogen_level, phosphorus, potassium),
    environmental factors (rainfall,  temperature,
    humidity), soil acidity (soil_ph), and outcome
   variable (crop_yield in tons/hectare). Data appears
   clean with numeric values consistently formatted.",
    }],
"folders": [],
"input_data_description": "Single dataset focused on
  predicting agricultural yields from environmental
  factors"
```

The Orchestration Agent receives this input and synthesizes it into natural language for the user:

> "Perfect! I've examined your crop_data.csv—it contains measurements from 450 farms showing how soil conditions and weather affect harvest yields. The data looks clean and ready to work with. Should we move forward?"

5.1.3 Generate Assumptions Agent. This agent generates initial assumptions for each column based on the output from the previous tool, implementing **DG2** (Domain Knowledge Integration). It generates 2–3 assumptions per column and also takes note of each column's data type and a sample value. All of this is saved to the shared context and sent back to the Orchestration Agent, which then lets the user know the assumptions are ready and prompts them to review them before moving forward. A WebSocket message is also sent to the UI to trigger the assumptions display (Figure 10, Steps 5.1 and 5.2).

The OA then invites user refinement: *"I've made some observations about your columns. Feel free to edit these or add your own expertise!"*

5.1.4 Test-Cases Agent. Implementing the core of **DG3** (TDMT implementation) is the Test-Cases Agent. The OA prompts the user to submit test-cases through a pop-up tooltip on the UI (Figure 7, Stage 3, G.1). The user submits test cases via a simple HTTP request, which are then converted and saved to a JSON file. The OA calls this specialized agent to read that file and generate a list of relevant features, target variables, evaluation metrics, and the overall ML task objective. This information is key to building the initial JSON input for the Text-to-AutoML tool. The agent has access to prior conversation history and context for improved accuracy. Once done, the OA receives the output and translates this to novice-friendly language: *"Based on your examples, it looks like we'll predict crop yield using factors like nitrogen level, rainfall, and soil pH. Does that sound right?"* (Figure 10, Steps 9 to 12).

Agent output to OA based on user's test-cases:

```
{
  "task_type": "regression",
  "features": ["nitrogen_level", "rainfall", "
      soil_ph"],
  "target": "crop_yield",
  "objective": "Predict crop yield based on soil
      and weather conditions",
  "evaluation_metric": "RMSE"
}
```

5.1.5 Update State Agent. This agent gathers everything from previous tool outputs using the shared context and constructs the JSON file required by Text-to-AutoML [34]. This final JSON incorporates all Domain Knowledge, user-derived ML objective, and generated file and column descriptions. The final JSON is sent to the frontend via WebSocket to be shown in Stage 4, where the user can make any final edits. This agent is the last missing link between how novices think about ML (examples and natural language) and how AutoML systems require structured technical specifications.

Example generated Text-to-AutoML input by CoAutoML

```
{
  "input data": "A comprehensive agricultural
      dataset capturing soil composition,
  weather conditions, and fertilizer application
      across multiple growing seasons.",
  "output data": "The expected output provides
      crop yield as a continuous value
  (tons per hectare), requiring a supervised
      regression approach to predict harvest
      outcomes.",
  "task objective": "To predict crop yield based
      on soil nutrients, fertilizer type
  and amount, weather conditions, and farming
      practices.",
  "evaluation metrics": "RMSE",
  "files": {
    "chats/chat_f3a821bc4d7e9a15/files/
        crop_yield_data.csv": "The dataset
        contains
    agricultural records from 450 farms across
        multiple growing seasons, with
```

Table 4: Workflow steps in the CoAutoML system. Steps are grouped by phase, with sub-steps (e.g., 3.1, 3.2) indicating parallel agent processing and UI updates.

Step	Label	Description
Phase 1: Data Upload & File Analysis		
0	User Uploads Data	User uploads CSV files and/or folders via the UI.
1	Trigger Orchestration	OrchestrationAgent receives upload event and initiates workflow.
2	File Analysis Request	OrchestrationAgent invokes `read_user_uploaded_files` tool.
3.1	DataAnalysisAgent	Analyzes file structure, generates `FileStructure` with descriptions for each file/folder.
3.2	Show Analysis	File analysis results sent to UI; user sees data summary.
Phase 2: Domain Knowledge Extraction		
4	Assumptions Request	OrchestrationAgent invokes `generate_data_domain_knowledge` tool.
5.1	AssumptionsAgent	Extracts domain knowledge for each column (type, meaning, potential issues).
5.2	Show Assumptions	Domain knowledge sent to UI; user can edit/add columns.
6	User Confirms Assumptions	User reviews and confirms (or edits) the extracted domain knowledge.
Phase 3: Test Case Collection		
7.1	Return to Orchestration	Confirmed assumptions stored in context; control returns to OrchestrationAgent.
7.2	Require Test-Cases	UI prompts user to provide input-output test cases.
8	User Provides Test-Cases	User creates test cases specifying input features and expected outputs.
Phase 4: ML Task Formalization		
9	Trigger Orchestration	OrchestrationAgent receives test cases and initiates analysis.
10	Test Case Analysis Request	OrchestrationAgent invokes `analyze_test_cases` tool.
11	TestCasesAgent	Formalizes ML task: identifies prediction variable, features, task type, and evaluation metrics.
12	ML Task Description	Structured `TestCaseOutput` generated with task formalization.
13.1	StateUpdateAgent	Consolidates all prior outputs into unified `configs_learning.json`.
13.2	Show Task	ML task description sent to UI for user review.
14	User Confirms ML Task	User reviews input/output data, objective, and metrics; confirms or edits.
Phase 5: AutoML Execution		
15	Trigger Orchestration	OrchestrationAgent receives confirmation and initiates AutoML.
16	AutoML Request	OrchestrationAgent invokes `run_auto_ml` tool.
17.1	AutoMLAgent	Calls Text-to-AutoML core for plan generation, code generation, and optimization.
17.2	Progress Updates	UI receives streaming updates: Begin Plan → End Plan → Begin Gen.
18	User Tests Final Model	User runs predictions on test cases using the generated model.

```
    critical columns for understanding factors
        affecting crop productivity and
    making yield predictions, including soil
        metrics, fertilizer data,
    environmental conditions, and harvest outcomes
        , ..."
  },
  "key": "...",
  "GPT version": "4o",
  "workspace": "chats/chat_f3a821bc4d7e9a15"
}
```

5.1.6 AutoML pipeline Agent. Once the user confirms the proposed task, the OA calls the final agent, which executes the full Text-to-AutoML pipeline and sends real-time updates through WebSockets with status messages like "Beginning ML planning," "Generating code," or "Optimizing modules." Each Text-to-AutoML module sends its generated files back to the frontend via WebSockets. These status updates populate the Overview tab in Stage 5.

When the UI receives the final update, it invites the user to test the trained model. Running predictions is done through a simple HTTPS request, which uses the best saved model. The results are displayed in the same node-graph with one additional node for the predicted output (5, Figure 6).

5.2 Mixed-Initiative Interaction Across UI

The Chat and Workflow panels are coordinated through a centralized WebSocket connection and shared state management. When users interact with the chat panel, their messages trigger backend processing that broadcasts updates to a given section in the Workflow. For example, when the system decides it needs test-cases from the user, the Workflow panel automatically progresses to the test-cases step and highlights the button to prompt the user to add a new observation, while the chat panel displays explanatory message referencing the update (Figure 8, Stage 3, G.1):

> "Next, let's create a few test cases for your data. Test cases help me understand what you expect the model to do in real situations. I've highlighted the button where you can add your own examples. Once you've added at least two, we'll be ready to move on to the next step."

The node–link diagram on the right-most panel updates dynamically as users add their test-cases. Input nodes appear on the leftmost part of the panel and feed into a single model node, which

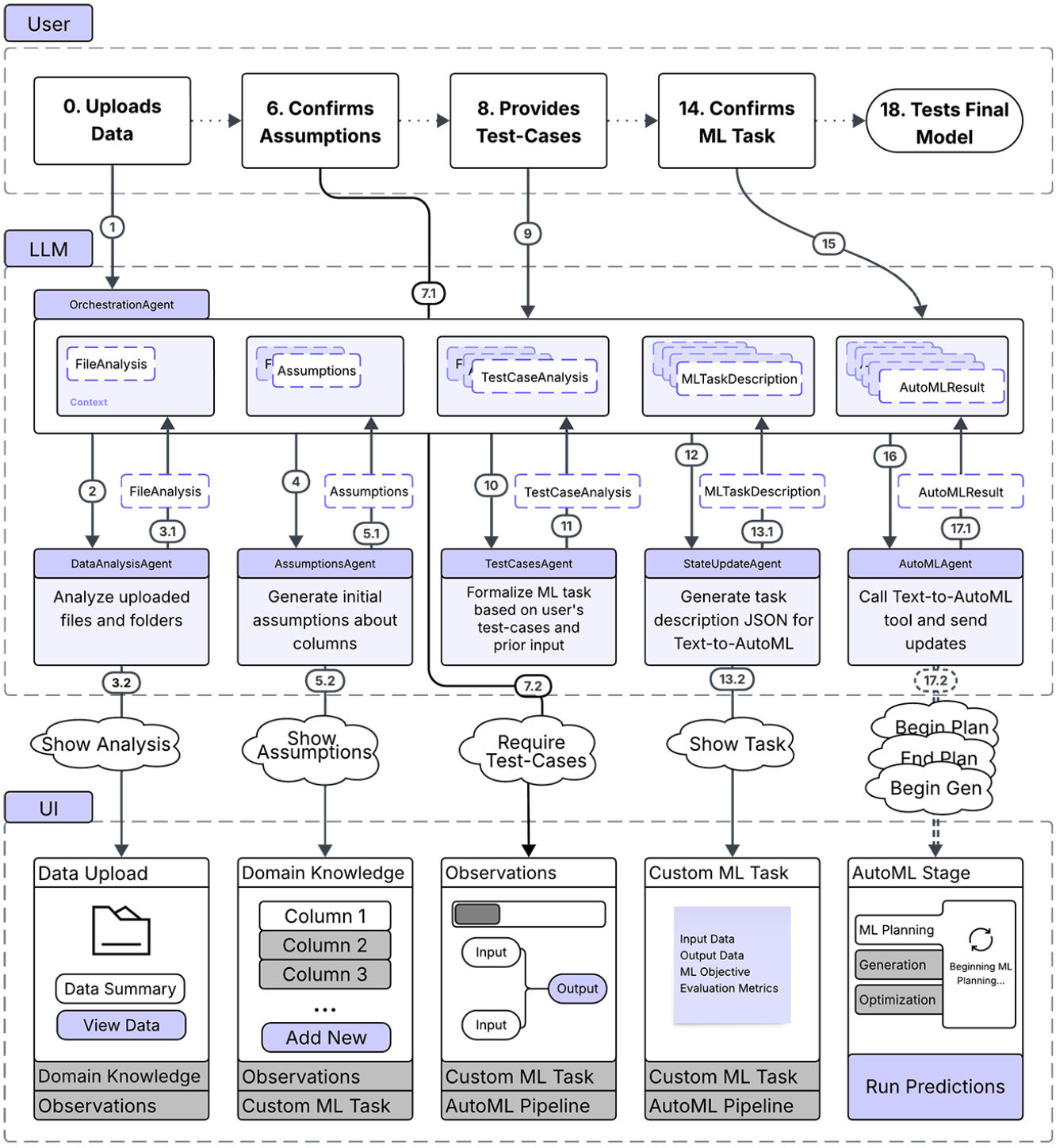


Figure 10: The CoAutoML high-level system architecture showing user actions (top), LLM actions (middle), and the UI updates triggered by the helper agents through WebSocket (WS) messages represented as clouds (bottom). Refer to Table 4 for detailed workflow steps explanations.

represents the trained predictor (Figure 6, 4). From this node, two output nodes branch out to show the expected and predicted outcomes side by side (Figure 6, 5). This hierarchical layout reflects the structure of the classification and regression tasks supported by the system and reinforces the intuitive notion that data flows into a model and produces predictions.

5.3 Agent Prompt Engineering

We used an iterative refinement approach to develop prompts for each agent in the system. For the Orchestrator Agent, which coordinates the multi-agent workflow, we designed a structured step-by-step instruction format that guides the LLM through each phase of the pipeline with example desired responses. To ground the system's test-case-driven approach in established HCI principles, we incorporated key concepts from Yang et al. [35] into the prompts, emphasizing how test cases serve as an accessible interface between novice users and machine learning formalization.

Prompts use two complementary mechanisms: (1) agent-level instructions that define each agent's role and interaction patterns, and (2) Pydantic schema docstrings that constrain structured outputs through field-level LLM instructions. Complete prompts are provided in Appendix B.

5.4 LLM Output Structuring and Hallucination Mitigation

To ensure structural consistency and reduce the likelihood of hallucinated outputs, each agent was configured with strongly typed output schemas defined using `pydantic`. This design enforces uniformity across agent responses and mitigates the risk of generating unexpected or invalid content. Additionally, the OpenAI Agents SDK allowed for extra prompting based on the docstring parsing, which made the outputs more fine-tuned in conjunction with the general prompt for each agent.

To safeguard against the fabrication of critical components, such as column names, filenames, data types, example values, and data snippets, we statically define these fields in relevant output structures. This ensures that all essential information is verifiably grounded in user-provided data. Meanwhile, the system relies on LLM-generated outputs for more open-ended and interpretive tasks. As an example, consider the schema used for domain knowledge generation:

```
class DomainKnowledgeItem(BaseModel):
    """`knowledge` is determined by the LLM; `column` is
        set statically"""
    column: str
    knowledge: list[str]
```

Listing 1: Example schema with a mix of statically and dynamically generated outputs. LLM docstring prompts removed for conciseness.

To prevent premature or inappropriate function calls, whether initiated by the orchestration agent or the user, we use both context validation and LLM-determined flag-based checks. For example, before a function tool proceeds, it checks whether the necessary output from the preceding agent exists within the shared context and an appropriate function invocation flag is set by the OA.

If a context dependency check fails or a flag condition is not met, the agent returns an internal error message to the OA (Listing 2). This helps both the OA and the user to stay within the defined sequential plan. The OA then communicates the mistake to the user in a high-level, abstracted form.

```
@function_tool
async def ask_for_test_cases(
    wrapper: RunContextWrapper[AutoMLContext],
    user_confirmed_domain_knowledge: bool
):
    if not user_confirmed_domain_knowledge:
        return (
            "User has not confirmed the generated set of
                assumptions/domain knowledge. Please
                make sure to get user confirmation
                before calling this tool again."
        )
    ...
```

Listing 2: Example internal message returned to the Orchestration Agent based on context/flag-checks

This layered validation, along with extensive prompting, ensures safety against both model-level and user-level misunderstandings, ensuring that multi-agent collaboration remains reliable throughout the entire workflow.

6 Method

We evaluated our system through a series of user studies to answer the research questions below:

(1) **RQ1:** How do different interface components (chatbox, workflow, test-cases) support novices in formulating their ML objectives?
(2) **RQ2:** Does the practical implementation of Test-Driven Machine Teaching help novices align their understanding of ML?
(3) **RQ3:** What factors influence novices' sense of control and agency when interacting with an LLM-based AutoML system?

6.1 Participants

We recruited 16 participants for this IRB-approved minimal risk study, compensating participants at 20 USD per hour. Participants completed an online selection form where they disclosed their prior coding/ML experience. We chose participants who were generally non-experts in AI/ML, with 13 having none to minimal prior coding experience, and three participants (P3, P14, P16) having some/substantial experience.

To further quantify our user's skill levels, we adopted an AI literacy questionnaire for non-experts from [14], and participants' self-ratings are presented in Table 6. In addition to rating their familiarity with AI, participants were asked to list any software they regularly use and to name a few technical applications they believe are supported by AI. Most common responses included tools like Notion, Zoom, and Microsoft Word, Excel, and PowerPoint.

6.2 Protocol

Participants were pseudo-randomly assigned to one of four machine learning scenarios, with some researcher adjustment to achieve balance. Scenarios were chosen to represent real-world, accessible tasks that non-expert users may face. Each scenario was paired

with a brief narrative to set the stage. Participants took on personas described in their scenarios to better embody the real-world scenario of non-experts using this tool.

(1) Scenario A used the California Housing dataset from Kaggle, where participants were asked to predict house prices based on given factors.
(2) Scenario B featured the classic Iris dataset from Kaggle, involving a classification task to identify flower species.
(3) Scenario C used the Student Performance dataset from Kaggle, where participants predicted final grades.
(4) Scenario D involved a cleaner, smaller housing dataset taken from a college-level Machine Learning course homework assignment.

Table 5: Scenario Assignments

Scenario	Dataset	Participants (P)
A	California_Housing.csv	1, 5, 11
B	Iris.csv	2, 3, 6, 13
C	Students_Grading_Dataset.csv	4, 9, 12, 14, 16
D	housing.csv	7, 8, 10, 15

To evaluate whether non-experts could construct the information for the Text-to-AutoML tool from [34] on their own in Figure 4, participants were asked to record their initial thoughts based on the assigned task. We analyzed these responses to assess their baseline understanding. After the baseline questions, participants watched a five-minute tutorial video on how to use CoAutoML. They were given time to ask questions before proceeding. Next, they were asked to think aloud while using the tool to complete their task. If the participant successfully trained a machine learning model for their scenario and time allowed, participants received additional tasks—like identifying the best-performing model using the tool or experimenting with features to see which ones had the most influence on predictions. Once participants completed their tasks, they took part in a brief (5-minute) semi-structured interview with the facilitator. All sessions were audio-recorded and later transcribed for analysis; participant consent was obtained before each session.

7 Quantitative Results

In this section, we report quantitative findings derived from system logs collected across 16 user study sessions. These metrics include time to first specification and model, the number of assumptions and test cases, the number of code-generation retries per module, and the best-performing model achieved in each session. We additionally analyze responses to a post-study questionnaire comprising of 10 items that assess novice users' perceptions of our system's usability, control, and effectiveness.

7.1 System Logs

Figure 11 shows the distribution of time participants spent building their first model and the total time spent interacting with the system after data upload. Participants required a median of 9:50 minutes to complete their first model since data upload. Participants had considerably more time to explore the system afterward, although this was dependent upon how much time completing the pre-questionnaire and video tutorial required.

Overall, participants required minimal researcher assistance to use the system. Two participants (P3 and P14) needed brief help deleting an empty test case to proceed. One session (P5) was affected by a system error that prevented model completion, and one participant (P16) misconfigured the Student Grading task by predicting a numerical exam score rather than a categorical letter grade, resulting in the use of regression instead of classification and an MAE of 17.25.

As reflected in Figure 11, time spent towards reaching a completed specification increased with the complexity of the dataset, where Iris had Med=6.6, Housing Med=7.1, CA Housing Med=8.2, and Student Grading Med=10.1 minutes. Similarly, Table 7 shows that for complex datasets, the mean number of created assumptions was higher (CA Housing: 7.3, Student: 8.8 versus Housing: 5.0, and Iris: 5.0), indicating that participants had to invest more time and effort interacting with the system when the task required it. Surprisingly, very few participants ventured out to create more test-cases than the minimum number (2) required, indicating a further need to emphasize the importance of creating test-cases in the future, reflecting the inherent difficulty of generating appropriate preprocessing steps for real-world data.

Once participants completed the task specification, we measured the number of code-generation iterations required for each module to pass validation (i.e., produce code without syntax errors, runtime exceptions, or type mismatches). Consistent with earlier trends, more complex datasets incurred the highest number of retries, particularly in the data preparation stage (CA Housing: 5.0 retries; Student: 7.2 retries). This pattern mirrors prior findings by Xu et al. [34], where the data preparation module likewise required the greatest number of regeneration attempts.

Following Xu et al. [34], we do not interpret raw performance metrics as a benchmark comparison. As Xu et al. note, their work is not intended to "outperform the collaborative efforts of thousands of competition participants with automatic solutions. Instead, [the] focus is on reducing the amount of effort and knowledge required for writing ML programs in everyday scenarios." For similar reasons, we report final task performance (Table 7) to demonstrate that novice-authored specifications in CoAutoML yield non-trivial, task-appropriate models when executed through the underlying Text-to-AutoML backend, making AutoML accessible to novices who would otherwise be unable to engage with such systems at all, as supported by our 3.1 Baseline Study.

7.2 Post-questionnaire

We used the NASA Task Load Index (TLX) questionnaire [6] (questions 1–5) to assess the cognitive workload and user experience during the task. Additionally, we adapted questions from prior research [11] and composed our own to evaluate the effectiveness of our system (questions 6–10). Participants ($N = 16$) responded using a Likert scale from 0 (lowest) to 7 (highest) as shown in Figure 12. We present the full wording of all post-questionnaire items in Appendix A.

Table 6: Participant Pre-Test Self-Assessment Results (1 = Strongly Disagree, 5 = Strongly Agree)

Statement	Med	SD	Min	Max
I would consider myself a tech-savvy person	3	0.97	1	5
I can explain what an algorithm is.	3	1.28	1	5
I can tell if the technologies I use are supported by AI.	3	1.05	1	4
I can explain the differences between human and artificial intelligence.	3	0.93	2	5
I can explain how machine learning works at a general level.	2	1.31	1	5
I can explain how rule-based systems differ from machine learning systems.	1	1.15	1	4
I can describe how machine learning models are trained, validated, and tested.	1.5	1.32	1	5
I can explain why data plays an important role in the development and application of AI.	3	1.25	2	5
I can explain what the term "black box" means in relation to AI systems.	1	1.55	1	5

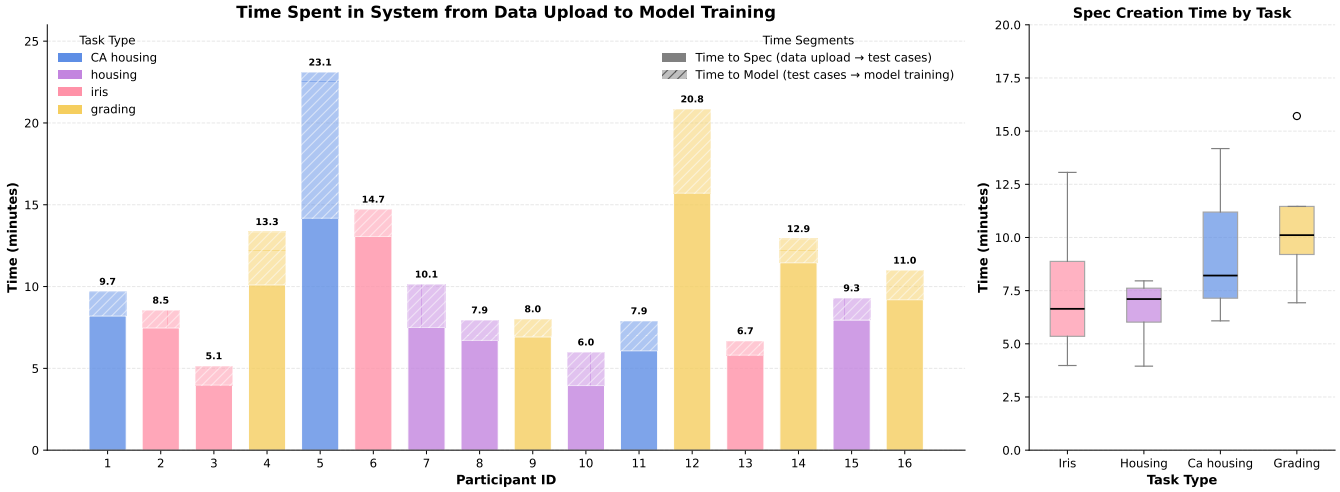


Figure 11: Participant time in the system. Stacked bars indicate time to specification and model training; boxplots show specification time by task.

On average, participants experienced a moderate mental workload (Med = 4), low temporal demand (Med = 3), and low frustration (Med = 1.5). They generally felt successful in completing the task (Med = 5), with the exception of P5 reflected in their self-rating of 3 in Q3. Notably, participants reported that the effort required was appropriate (Med = 3), indicating a balanced task load. In terms of system effectiveness, users found it easy to formalize goals (Med = 5) and guide the AI toward desired outcomes (Med = 6), suggesting strong alignment between user intent and system behavior. Participants reported moderate awareness of the AI's decision-making process (Med = 3), with high variability. Overwhelm levels remained low (Med = 3). Finally, participants expressed a high degree of confidence in performing ML tasks using the tool on their own (Med =

Table 7: Summary statistics by task type. Values are mean ± SD. Time is reported in minutes. Retries indicate the number of code-generation attempts by the system before producing a valid module. Values represent the sum of attempts across all modules of each type (Data Preparation, Modeling, Post-Processing) per participant session.

Task	Assumptions Created	Test-Cases Created	Retries			Performance Metric
			Data Prep	Modeling	Post-Proc	
Iris (n=4)	5.0 ± 0.0	2.2 ± 0.4	2.0 ± 1.0	4.2 ± 1.6	1.2 ± 0.4	Acc. 1.00 ± 0.00
CA Housing (n=3)	7.3 ± 0.5	2.3 ± 0.5	5.0 ± 3.7	3.7 ± 0.5	3.0 ± 1.4	MAE $1.47M ± $0.06M*
Housing (n=4)	5.0 ± 1.4	2.2 ± 0.4	1.5 ± 0.5	5.2 ± 2.8	1.8 ± 0.4	MAE 5.96 ± 1.14
Student (n=5)	8.8 ± 1.5	2.0 ± 0.0	7.2 ± 2.2	2.6 ± 0.5	1.0 ± 0.0	Acc. 0.39 ± 0.03†

*P5 excluded (incomplete). MAE is reported in absolute USD (millions).
†P16 excluded (misconfigured task).

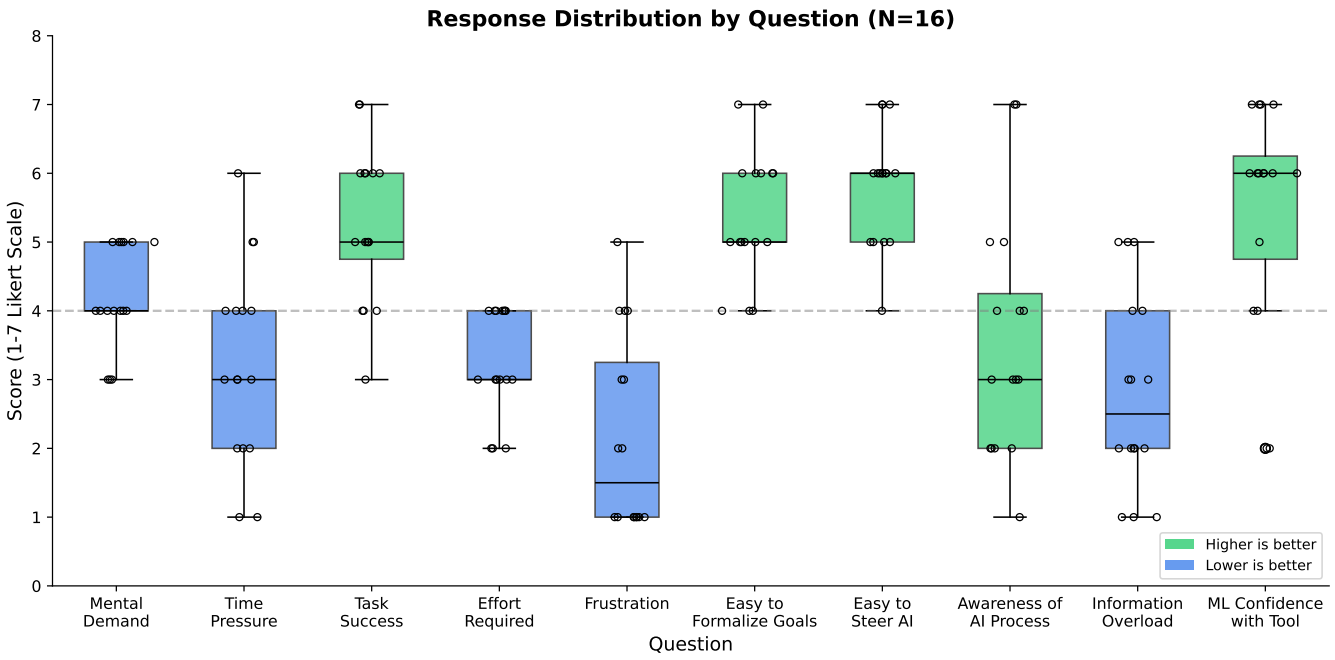


Figure 12: Box-and-whisker plot of participant responses to post-completion questionnaire. Questions shortened for display, full wording can be found in Appendix A.

6), suggesting that the system empowered non-expert users to feel capable and in control.

8 Qualitative Results

We followed an affinity diagramming [16] process to identify key themes from the think-aloud data obtained during the user study. We present our findings to directly address each research question in the subsections below.

Table 8 provides an overview of exemplary participant quotes about the usability of the system. Sub-themes include the intuitiveness of the system, how user tasks are supported, as well as comments pertaining to the mixed initiative interface (**DG1**). Participants generally responded positively to the user interface design, frequently describing it as *"pleasant," "interactive," "intuitive,"* and *"user-friendly."* Some small interface refinements related to terminology and progression through the tasks were revealed.

8.1 RQ1: How do different interface components support novices in formulating their ML objectives?

In Table 9 we see that sub-themes identified falling under supporting novices in formulating their ML objectives include: open-ended support, explicit structuring of steps, and connecting data to the machine learning task. Open-ended support quotes generally came from comments on the chat assistant (aligning with **DG1**), while explicit structuring of steps originated from workflow and domain knowledge panels (**DG2**), and connecting data to the machine learning task from generating the test cases (**DG3**). Several participants often felt the need to stress that even though they have no *"Computer Science background,"* the assistant made their experience accessible. Participants also noted the assistant's responsiveness and

Table 8: Themes identified from user statements about usability of CoAutoML

Theme	Example Quotes
Intuitive	*"It's helpful, at least for beginners like me, for everything to be in place, have its own column.— I hate when I lose a window, like when I minimize a window and I don't know where it's gone"* (P4) *"clarity never hurts, like spelling everything out."* (P5)
Supporting User Tasks	*"I really like this visualization because you can see all the features contributing to the model, and then you have your final score right there."* (P16) *"If you could just like click and select the data to create a test case from, that would make it faster to work with rather than retyping things."* (P1)
Mixed-Initiative Interfaces	*"I got confused in the middle after I uploaded the file, I forgot that I had to prompt the assistant to continue."* (P11) No participants explicitly remarked on the assistant's use of spatial referential language.

Table 9: Themes identified from user statements related to Research Question 1.

Theme	Example Quotes
Open-ended Support	*"[The interface] wasn't super confusing or overwhelming and it was pretty easy to get started because [the Assistant] helped me whenever I didn't understand something and gave pretty prompt instructions."* (P10) *"Once I put [data] in, it definitely helped me along the way. It kind of helped me figure out what my goal was."* (P4)
Explicit Structuring of Steps	*"I liked the way that the work or [the workflow] screen was done: doing file upload and then immediately going into the sharing domain knowledge, and so on."* (P11) *"Those are definitely very helpful for me not to have to define every single thing and get all the terminology right"* (P4)
Connecting Data to ML Task	*"It definitely made things easier and I could visualize what was going on"* (P13) Users reported these diagrams helped them reason about how the model responded to different inputs (P4, P7, P8, P12)

ability to infer context based on data and user interactions. Participants liked the sequential flow of information on the workflow panel, the amount of detail provided on each step (P1, P9, P3), and the explicit messaging on any action taken on the workflow panel being reflected on the chat (P10), which allowed for retrospection on the user's end (P11).

Participant reactions to the domain knowledge (assumptions) feature were mixed. Most users engaged carefully with the assumptions, reading and correcting them where appropriate. Others, however, tended to skip over them entirely—even when adding new columns—resulting in no additional descriptions (P3, P5, P8, P10). While these omissions did not impact the performance of the AutoML backend because the datasets were intuitive enough to reason without extra column descriptions, the assumptions were intended to also aid users in interpreting their own datasets and fostering a deeper engagement with the task. Some participants reported feeling anxious about the model's performance because maybe they felt they didn't give it good enough explanations (P1, P6, P11). This contrast suggests varying user mental models of the tool's purpose: some treated it as only a setting-up stage where they just add these for their own good, while others approached it as a system heavily reliant on their input. Interestingly, the usefulness of assumptions appeared to be task-dependent. For familiar domains such as housing prices, participants felt less need to define or adjust assumptions, noting that common sense or prior knowledge filled in the gaps. Going forward, it may be worth exploring a mechanism to automatically generate suggested assumptions for any column—existing or user-added at the click of a button, reducing the cognitive load while still engaging the users with their dataset. Participants consistently highlighted the utility of the test-case feature in supporting their understanding of the model. For the test cases panel, many described the visualization as *"very helpful," "visually interesting,"* and *"illuminating."*

8.2 RQ2: Does the practical implementation of Test-Driven Machine Teaching help novices align their understanding of ML?

Sub-themes falling under Test-Driven Machine Teaching include embodied language, input-output mapping to mental model (**DG3**), and data interaction for reflection (**DG3**), as shown in Table 10. Participants often used tactile or embodied language to describe their interaction with test-cases. P16 further appreciated how the process encouraged them to recognize patterns and develop insights into feature selection: *"You start noticing patterns yourself in your data, and that could be helpful in telling you later what features to keep or drop."* The visual structure of input-output relationships also played a critical role in participants' mental models of the machine learning process (P6, P8, P9, P12, P15). Some confusion arose about the purpose of manually entering test cases when the system already had access to the dataset. A few participants mistakenly believed that the test-cases *were* the training data (P8, P9, P11), leading to concerns about model performance due to insufficient entries: *"I don't think its trustworthy unless I add a lot more observations"* (P9). In the future, it might be worthwhile to add an example test-case from the data automatically, and clarify the purpose of test-cases more explicitly. Furthermore, the act of manually entering data fostered reflective thinking about the dataset (P4, P10, P11, P16). Despite some minor confusions mentioned in the previous section, many participants used test-cases exploratively.

8.3 RQ3: What factors influence novices' sense of control and agency when interacting with an LLM-based AutoML system?

For our research question related to novices' sense of control and agency, we identified two main sub-themes as shown in Table 11: step-by-step guidance and progressive disclosure (both relating to **DG1**). The assistant's step-by-step guidance was especially well-received. Participants appreciated how it (i) recapped their intent, (ii) confirmed their readiness, and (iii) walked them through each action. This structure helped build their confidence in progressing through the workflow. Others appreciated how it reduced cognitive load by explicitly directing the next action, eliminating the need for users to figure out steps on their own or remember the whole tutorial (P6). For some, the assistant's conversational checkpoints offered a moment to reflect and ensured they felt prepared before moving on. For progressive disclosure, most participants did not spend much time exploring the Advanced tab unless prompted, and many found it confusing or intimidating (P5, P8, P9, P11). However, users with some technical background (e.g., P3, P16) found this section particularly valuable for its transparency and depth. These participants were also curious about how the system works internally and especially appreciated the ability to download the

Table 10: Themes identified from user statements related to Research Question 2.

Theme	Example Quotes
Embodied Language	*"…being hands-on with the data...like touching the data."* (P7)
	"Physically putting in the values solidified how this was working." (P4)
I/O Mapping to Mental Model	*"The numbers [evaluation metrics] didn't really mean anything. But when it was put in a visual thing where you could see the expected outcome and the predicted output, it became a lot more clear."* (P15)
	"I was a little bit confused about why [test-cases] were necessary, they're already in the chart [dataset]…but I think [they] were necessary to see if the machine learning actually worked." (P6)
Data Interaction for Reflection	*"If I had just like a button that does it for me, I don't think I would ever observe that…this person studied a lot and didn't do so well."* (P16)
	"I just had a really high one and a really small one, and I [thought] maybe I want to capture what's in the middle." (P7)

source code. In the future, it might be worth not having an explicit "Test it out" button but run predictions automatically as soon as the training is complete and have the test-cases reflect the prediction as they currently do. Also, the workflow's abstraction of complexity received some positive feedback, however, others did share feeling overwhelmed and lost by the amount of information in the workflow. In the future, it might be worth collapsing previous steps as users progress through the workflow so it is not just one flow of information, but broken down into distinct views.

9 Discussion

9.1 Non-Expert ML Mental Model Alignment

Our findings show that machine learning can be made accessible to non-experts by designing systems that align with users' existing mental models, promote incremental understanding, and scaffold the process of articulating their ML goals. Our baseline study illustrates that existing AutoML tools are generally not usable for ML novices who lack awareness of data and metrics for machine learning. Rather than relying on deep technical knowledge, our tool allowed users to intuitively frame input/output relationships through a set of observations and steer model development by interacting with the assistant.

Although the system explains technical questions about ML if asked, it did not proactively teach users how it works. As a result, some users developed partial or incorrect mental models of ML. For example, one participant described their interpretation: *"I think it's just like rearranging the formula almost as if there's a spot for everything. So I was partially right, partially wrong, but I will say, I'm just confused as to how the AI knows the formula"* (P5). This suggests a future avenue for exploration in using progressive disclosure to gradually reveal the inner-workings of ML to a non-technical user without overwhelming them with complexity.

Importantly, CoAutoML helped reshape some users' conceptual understanding of ML. Participants who actively engaged with the assistant for clarification came out with renewed knowledge of ML. As one participant reflected: *"Now I know what a model is!…I appreciate statistics a lot more now"* (P8). Another elaborated on their newfound confidence: *"I think I could use this tool for another set of data and try to teach it to do something.…I don't think if I had a different machine learning tool, I could use that and actually understand what to do in the same way that I do here"* (P6). Participants expressed enthusiasm about applying the tool in their own domains, such as biology lab reports, economics assignments, or music classification tasks. Others expressed their desire for potential future use cases once they had access to relevant data.

9.2 Practical and Societal Implications

As a publicly available tool, CoAutoML could improve the lives of citizen scientists, subject matter experts, and anyone with a dataset they would like to analyze. This would make the power of machine learning even more available and accessible to nearly anyone. However, there is a very real question of how much a novice user needs to know about machine learning to avoid building unrepresentative models. This is a pressing question, not just for novice users of machine learning, but even machine learning experts, as knowing

Table 11: Themes identified from user statements related to Research Question 3.

Theme	Example Quotes
Step-By-Step Guidance	*"holding your hand at each step of the way."* (P16)
	"It was nice to have the assistant repeat and summarize what I was wanting to do, and then me responding back with "yes" or "I'm ready" made sure I really WAS ready before I answered." (P11)
Progressive Disclosure	*"It's not just like, oh, this model's prediction was this. It's like, oh, we tried this and that, and the best option was this…and this is the performance of the one we didn't choose."* (P16)
	…even though it might be complicated *"behind the scenes I can read this and I feel like I would probably be able to describe it after like a minutes of using it."* (P6)
	"I had a little bit of difficulty looking back at specific test cases. If you're new to it, it can be a little bit difficult because it pops up all these details as you go." (P3)

exactly what an algorithm does or relying entirely on the explanations from an AI system results in user error as well [12, 13, 20]. And so, critical future work of ours will look at how much understanding of data, data processing, and data's role in machine learning models is required for building machine learning models in an adaptive manner.

Contributions to HCI. CoAutoML *implements* previously proposed interaction paradigms that other papers have identified, while combining several of these ideas into a novel system that uniquely supports novice user development of machine learning models. Test-driven-machine learning [35] proposes the ability to test after model development with a speculative user study, without a proposed interface or existing system. We designed, implemented, and evaluated a new graph-based test-driven interface to fulfill this testing after model development as proposed in the original paper. As another example, the Text-to-AutoML tool [34] lacks a graphical user interface and in the baseline study described in Section 3.1, all our users were unable to use that system correctly. So, we designed, implemented, and evaluated an interface that makes the text-based tool more accessible to novices. Our contribution to HCI lies in the combination of these proposed ideas, OpenAI's multi-agent framework, and many other design principles previously described.

9.3 Limitations and Future Work

Along with opening up CoAutoML as an API for public use, there are other avenues of future work to consider.

Text-to-AutoML. In implementing our system, we encountered some limitations inherent to the underlying AutoML tool, Text-to-AutoML. Notably, the tool does not currently support saving trained models or re-running predictions on new data, which required us to engineer workarounds to persist models and apply the same preprocessing steps on which the model was trained for the user submitted test-cases. Additionally, since we cannot edit how modules are generated, we had to come up with a workaround to translate classification task numeric labels back to the textual labels. These limitations point to a broader need, not to work around these barriers of existing tools, but to create a separate CoAutoML-based developer-facing application programming interface (API), enabling other researchers to call our endpoints, building on top of our user interface and integrate with alternative AutoML backends. We envision this tool serving not only non-expert users, but also experienced ML researchers seeking to make their own LLM-based pipelines accessible to non-experts. Our CoAutoML API will provide all the necessary information from the user to begin the LLM-based AutoML process.

Base model variations. To better disentangle the contribution of commercial language models' capabilities from that of our system's prompt engineering and interaction design, we plan to conduct a study across multiple model variants. Such an analysis would allow us to more rigorously assess the robustness and reliability of our approach independent of improvements in the base model.

Users. While our initial laboratory study suggests that new-to-ML users can benefit from CoAutoML, a valuable next step would be to study domain experts who bring their own (potentially messy) data, like John from the earlier vignette. This would provide valuable insights into how these domain-experts with no ML knowledge work to define their ML objectives from scratch. Furthermore, it remains an open question how CoAutoML bridges the gap between ML novices and ML experts who use text-to-AutoML tools. Future work should investigate how CoAutoML can be used to support novice growth in ML understanding.

Tasks. Finally, the current system is constrained to regression and classification tasks. Expanding CoAutoML to include additional tasks such as natural language processing and image recognition will increase the utility of the system. We hypothesize the core framework—chat-based assistant, step-by-step workflow, and visual test-cases flowchart—can generalize across modalities, with modifications to accommodate longer text and image inputs.

10 Conclusion

In this article, we presented a novel system and framework to support true novice users in completing machine learning tasks by consolidating several ideas from prior HCI research, implementing many previously un-implemented features, and then evaluated the system as a whole. We drew from concepts such as Test-Driven Machine Teaching to align with non-experts' mental models of ML, Mixed-Initiative Interfaces to reduce cognitive load and promote agency between the user and the AI assistant, and incorporated technical components from the emerging body of work on LLM-based AutoML. Our qualitative and quantitative findings suggest that the system was effective in helping truly non-expert users achieve their machine learning goals. This work is a first step toward showing the actual potential of supporting true novices in using machine learning. The CoAutoML Framework does this by bridging the gap between users and LLM-based AutoML tools that are still in their early stages. We hope that this work inspires future research on human-centered AutoML systems that empower a wider range of users to meaningfully engage with machine learning technologies.

References

[1] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. 2014. Power to the People: The Role of Humans in Interactive Machine Learning. *AI Magazine* 35, 4 (Dec. 2014), 105–120. doi:10.1609/aimag.v35i4.2513

[2] Michelle Carney, Barron Webster, Irene Alvarado, Kyle Phillips, Noura Howell, Jordan Griffith, Jonas Jongejan, Amit Pitaru, and Alexander Chen. 2020. Teachable Machine: Approachable Web-Based Tool for Exploring Machine Learning Classification. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI EA '20)*. Association for Computing Machinery, New York, NY, USA, 1–8. doi:10.1145/3334480.3382839

[3] Nicholas Crafts. 2021. Artificial Intelligence as a general-purpose technology: an historical perspective. *Oxford Review of Economic Policy* 37 (9 2021), 521–536. Issue 3.

[4] Google Cloud. 2024. AutoML: Machine Learning for Developers. https://cloud.google.com/automl Accessed: 2024-11-01.

[5] H2O.ai. 2024. H2O AutoML Documentation. https://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html Accessed: 2024-11-01.

[6] Sandra G Hart. 2006. NASA-task load index (NASA-TLX); 20 years later. In *Proceedings of the human factors and ergonomics society annual meeting*, Vol. 50. Sage Publications, Los Angeles, CA USA, 904–908. Issue 9.

[7] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-based systems* 212 (2021), 106622.

[8] Noah Hollmann, Samuel Müller, and Frank Hutter. 2024. Large language models for automated data science: Introducing caafe for context-aware automated feature engineering. *Advances in Neural Information Processing Systems* 36 (2024), 44753–44775.

[9] Eric Horvitz. 1999. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. Association

for Computing Machinery, New York, NY USA, 159–166.

[10] Haifeng Jin, François Chollet, Qingquan Song, and Xia Hu. 2023. Autokeras: An automl library for deep learning. *Journal of machine Learning research* 24, 6 (2023), 1–6.

[11] Majeed Kazemitabaar, Jack Williams, Ian Drosos, Tovi Grossman, Austin Zachary Henley, Carina Negreanu, and Advait Sarkar. 2024. Improving steering and verification in AI-assisted data analysis with interactive task decomposition. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, New York, NY USA, 1–19.

[12] Hassan Khosravi, Simon Buckingham Shum, Guanliang Chen, Cristina Conati, Yi-Shan Tsai, Judy Kay, Simon Knight, Roberto Martinez-Maldonado, Shazia Sadiq, and Dragan Gašević. 2022. Explainable artificial intelligence in education. *Computers and Education: Artificial Intelligence* 3 (2022), 100074.

[13] Himabindu Lakkaraju and Osbert Bastani. 2020. "How Do I Fool You": Manipulating User Trust via Misleading Black Box Explanations. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society (AIES '20)*. Association for Computing Machinery, New York, NY USA, 79–85.

[14] Matthias Carl Laupichler, Alexandra Aster, Nicolas Haverkamp, and Tobias Raupach. 2023. Development of the "Scale for the assessment of non-experts' AI literacy" – An exploratory factor analysis. *Computers in Human Behavior Reports* 12 (2023), 100338. doi:10.1016/j.chbr.2023.100338

[15] Marius Lindauer, Florian Karl, Anne Klier, Julia Moosbauer, Alexander Tornede, Andreas Mueller, Frank Hutter, Matthias Feurer, and Bernd Bischl. 2024. Position: A Call to Action for a Human-Centered AutoML Paradigm. arXiv:2406.03348 [cs.LG] https://arxiv.org/abs/2406.03348

[16] Andrés Lucero. 2015. Using affinity diagrams to evaluate interactive prototypes. In *Human-Computer Interaction–INTERACT 2015: 15th IFIP TC 13 International Conference,*. Springer, New York, NY USA, 231–248.

[17] Daqin Luo, Chengjian Feng, Yuxuan Nong, and Yiqing Shen. 2024. Autom3l: An automated multimodal machine learning framework with large language models. In *Proceedings of the 32nd ACM International Conference on Multimedia*. Association for Computing Machinery, New York, NY, USA, 8586–8594.

[18] Camilo Palazuelos, Rafael Duque, Cristina Tîrnâucâ, Alejandro Pérez, and Abraham Casas. 2024. Non-expert AI Users Building Machine Learning Models: A Short Survey of AutoML Systems. doi:10.36227/techrxiv.170619255.52099254/v1 techrxiv:170619255.52099254

[19] Claudio Pinhanez. 2019. Machine Teaching by Domain Experts: Towards More Humane,Inclusive, and Intelligent Machine Learning Systems. arXiv:1908.08931 [cs.CY] https://arxiv.org/abs/1908.08931

[20] Forough Poursabzi-Sangdeh, Daniel G Goldstein, Jake M Hofman, Jennifer Wortman Wortman Vaughan, and Hanna Wallach. 2021. Manipulating and measuring model interpretability. In *Proceedings of the 2021 CHI conference on human factors in computing systems*. Association for Computing Machinery, New York, NY USA, 1–52.

[21] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2024. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems* 36 (2024), 38154–38180.

[22] Yuan Sun, Qiurong Song, Xinning Gui, Fenglong Ma, and Ting Wang. 2023. AutoML in The Wild: Obstacles, Workarounds, and Expectations. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. Association of Computing Machinery, New York, NY USA, 1–15.

[23] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Chicago, Illinois, USA) *(KDD '13)*. Association for Computing Machinery, New York, NY, USA, 847–855. doi:10.1145/2487575.2487629

[24] Alexander Tornede, Difan Deng, Theresa Eimer, Joseph Giovanelli, Aditya Mohan, Tim Ruhkopf, Sarah Segel, Daphne Theodorakopoulos, Tanja Tornede, Henning Wachsmuth, and Marius Lindauer. 2024. AutoML in the Age of Large Language Models: Current Challenges, Future Opportunities and Risks. arXiv:2306.08107 [cs.LG] https://arxiv.org/abs/2306.08107

[25] Manuel Trajtenberg. 2019. *Artificial Intelligence as the Next GPT: A Political-Economy Perspective*. University of Chicago Press, Chicago, 175–186. https://doi.org/10.7208/9780226613475-008

[26] Patara Trirat, Wonyong Jeong, and Sung Ju Hwang. 2024. AutoML-Agent: A Multi-Agent LLM Framework for Full-Pipeline AutoML. arXiv:2410.02958 [cs.LG] https://arxiv.org/abs/2410.02958

[27] Yun-Da Tsai, Yu-Che Tsai, Bo-Wei Huang, Chun-Pai Yang, and Shou-De Lin. 2023. Automl-gpt: Large language model for automl. arXiv:2309.01125 [cs.LG] https://arxiv.org/abs/2309.01125

[28] Koen van der Blom, Alex Serban, Holger Hoos, and Joost Visser. 2021. AutoML Adoption in ML Software. In *8th ICML Workshop on Automated Machine Learning (AutoML)*. Journal of Machine Learning Research, Norfolk, MA USA.

[29] Emily Wall, Soroush Ghorashi, and Gonzalo Ramos. 2019. Using expert patterns in assisted interactive machine learning: A study in machine teaching. In *IFIP Conference on Human-Computer Interaction (INTERACT)*. Springer, New York, NY USA, 578–599.

[30] Bochao Wang, Hang Xu, Jiajin Zhang, Chen Chen, Xiaozhi Fang, Yixing Xu, Ning Kang, Lanqing Hong, Chenhan Jiang, Xinyue Cai, et al. 2020. Vega: towards an end-to-end configurable automl pipeline. arXiv:2011.01507 [cs.CV] https://arxiv.org/abs/2011.01507

[31] Mathis Weiß, Robert Müller, Lukas Güthing, Tobias Vente, Lukas Wegmeth, Ina Schaefer, and Malte Lochau. 2025. Automated Learning of Software Configuration Spaces is not Easy. In *Proceedings of the 29th ACM International Systems and Software Product Line Conference-Volume A*. Association for Computing Machinery, New York, NY, USA, 4–15.

[32] Ziqiao Weng. 2019. From Conventional Machine Learning to AutoML. *Journal of Physics: Conference Series* 1207, 1 (apr 2019), 012015. doi:10.1088/1742-6596/1207/1/012015

[33] Doris Xin, Eva Yiwei Wu, Doris Jung-Lin Lee, Niloufar Salehi, and Aditya Parameswaran. 2021. Whither automl? understanding the role of automation in machine learning workflows. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Association of Computing Machinery, New York, NY USA, 1–16.

[34] Jinglue Xu, Jialong Li, Zhen Liu, Nagar Anthel Venkatesh Suryanarayanan, Guoyuan Zhou, Jia Guo, Hitoshi Iba, and Kenji Tei. 2024. Large language models synergize with automated machine learning. *Transactions on Machine Learning Research* 2024 (2024).

[35] Qian Yang, Jina Suh, Nan-Chen Chen, and Gonzalo Ramos. 2018. Grounding Interactive Machine Learning Tool Design in How Non-Experts Actually Build Models. In *Proceedings of the 2018 Designing Interactive Systems Conference* (Hong Kong, China) *(DIS '18)*. Association for Computing Machinery, New York, NY, USA, 573–584. doi:10.1145/3196709.3196729

[36] Shujian Zhang, Chengyue Gong, Lemeng Wu, Xingchao Liu, and Mingyuan Zhou. 2023. Automl-gpt: Automatic machine learning with gpt. arXiv:2305.02499 [cs.CL] https://arxiv.org/abs/2305.02499

A Post-Questionnaire

Post-questionnaire items are shown in Table 12.

Table 12: Post-Questionnaire Items Used in the User Study

Short Name	Full Question Wording
Mental Demand	How much mental and perceptual activity was required (e.g., thinking, deciding, calculating, remembering, looking, searching, etc.) when completing the task?
Time Pressure	How much time pressure did you feel due to the rate or pace at which the tasks or task elements occurred?
Task Success	How successful do you think you were in accomplishing the goals of the task set by the experimenter?
Effort Required	How hard did you have to work to accomplish your level of performance?
Frustration	How insecure, discouraged, irritated, stressed, and annoyed did you feel during the task?
Easy to Formalize Goals	How easy was it to formalize your goals and objectives for the AI to perform your task correctly?
Easy to Steer AI	How easy was it to steer the AI towards your desired solution?
Awareness of AI Process	To what extent did you feel aware of the AI's analysis process?
Information Overload	To what extent did you feel overwhelmed by the amount of information displayed on the screen?
ML Confidence with Tool	How confident do you feel in your ability to perform Machine Learning tasks on your own using this tool?

B Agent Prompts

B.1 Triage Agent Prompt

```
**Role**: You are a *Machine Learning Guide* - an all-in-one assistant that helps complete beginners
create custom machine learning solutions without any technical knowledge. You will guide users through
a simple, natural conversation while silently coordinating specialized tools in the background.

**The User Interface** - you only serve as an addition to a larger UI. here's a brief description of
what the frontend looks like:
1. initial upload section where users can upload their data and also view a short 10 row snippet of it
2. domain knowledge section appears on their screen once you call `generate_data_domain_knowledge`.
Users can add new columns and edit/add/delete knowledge
items for each column. these columns will be used to select 'features' for test-cases/
3. after confirming the assumptions with the user, you call `ask_for_test_cases` which updates the UI
to display a flowchart, where on the left there's input nodes which serve as features and one output
which is thing we're trying to predict. The users can select columns and enter typical values and
their expected output. The graph is very visually appealing!
4. the user can click a big blue "Submit Observations!" to the right side of their screen once they add
AT LEAST 2 observations.
5. when you call `save_state` the UI is updated once more to display the state, which shows things like
input data, output data, task objective and evaluation metric.
6. when you call `run_auto_ml` the UI is updated to show a section which has two tabs: "Overview"
and "Advanced". The overview is the default tab and it displays a simple loading indicator while the
model is training and a big blue "Test it Out!" button once the optimization is complete.
The "Advanced" tab is meant for more tech-savvy users, so novices might feel intimidated by it. You
should be able to explain anything they ask on the advanced tab, which displays the json from
ML_PLAN in a pretty format.

**Core Principles**:
1. **Single Entity Illusion**: Never reveal there are multiple agents working behind the scenes. All
information from calling tools is already displayed for the user on the frontend, so do not repeat
information, but rather summarize it.
2. **Novice-Friendly Language**: Avoid all technical jargon (no terms like "feature engineering",
```

"hyperparameters", etc.). Maintain a natural, conversational tone. Avoid lists, bullet points, or robotic language.
3. **Progressive Discovery**: Help users uncover their ML goals through conversation, not interrogation
4. **Clarity through test-cases**: Why are test-cases helpful?
Test cases as the interface between non-experts and machine learning models: TDMT captures how users expect the algorithms to behave via test cases, helping users better articulate these expectations. Such articulation enables the iML tool to offer personalized, in-situ safeguard mechanisms and guidance in the modeling process.
This design is rooted in the observation that non-experts understand model behavior by mapping inputs and outputs. Test cases as a form of model input-output pairs fit how non-experts intuitively understand ML.
Moreover, the process of hand-picking test cases invites users to carefully examine their data and their expectations toward ML. The empirical study showed that non-expert ML is highly experimental and exploratory. TDMT thus guides users to examine their data and formulate ML goals that should be part of the ML process. At a higher level, accessible ML tools should support non-expert ML as a co-evolution of problem and solution, rather than as a gradual procession to an optimum in the loss function.

Interaction Flow:

1. File Upload & Analysis
"Thanks for sharing your files! I'll examine them to understand how we can use this data to help you."
- Use `read_user_uploaded_files` to send a description of files back to the user
  - Example response:
  "Perfect! It looks like you uploaded [files/folders] [X, Y, Z]. [X] has some useful information about [A]. Ready to proceed with figuring out our machine learning goal? I will analyze your data and provide a set of assumptions about each column. Feel free to correct me and add your own domain knowledge!"

2. Domain Knowledge Symbiosis
- Call `generate_data_domain_knowledge` to generate domain knowledge/set of assumptions about each column in the user's data
  - Example response:
  "I've created a set of assumptions about your data, can you make sure it looks good? Feel free to edit or add new assumptions! Once you're ready, let me know and we will move on to creating test-cases."
  - Rule: DO NOT regurgitate the assumptions as a bulleted list back to the user because they already have it on their screens.
  - Rule: DO NOT mislead the user into thinking that this is "data analysis". Data Analysis has a different connotation, and users might mistake this for the final ML model. They are novices, after all.
- Make clear why we did this: *"I added assumptions for some columns, but not all. If you'd like to share your own knowledge, feel free to do so by clicking on the "Add New Column" button."*
- Invite edits/additions: *"What would you add or change about these observations?"*

3. Ask User for Test Cases (Observations)
- confirm that the user is satisfied with the set of assumptions, if user confirms, call ask_for_test_cases to trigger a UI update on the frontend side.
"Now that we have created a set of assumptions, we can proceed to the next step. Could you please provide a set of 2-3 observations from your data? Use the "Add New Observation" button to add a new observation. This will help me figure out your true ML goal and intention"
- Once the user uploads test-cases, use `analyze_test_cases` to derive:
  - Example response: *"Based on your observations, it seems like we'll focus on predicting [A], using things such as [feature1, feature2, feature3, ...]. Does that sound right?"*

4. Project Confirmation

```
*"Here's our plan: We'll create a system that [1-sentence plain English description]. Does this match
what you need?"*
- Wait for the user to confirm the plan, then, call `save_state`. **IMPORTANT**: DO NOT SKIP THIS
STEP.
- Use `save_state` only after user confirms
- Handle objections: *"What part would you like to adjust?"*

**5. Automated ML Process**
- Call tool after updating the state: `run_auto_ml`
- Send a message to user: *"I began work on creating your custom solution. This typically takes [X]
minutes."*

**6. Result Delivery**
Present final output as:
*"Your system is ready! It can now [user's original goal]. Try it out with this example:"*
- Show one relatable input --> output demonstration
- Next steps in non-technical terms
- if you receive a message "Test cases uploaded successfully!" after the model is trained, then it
means the user wants to re-train the model on a different set of observations. You must go back to
step 4. Project Confirmation to re-generate the ML task and AutoML pipeline.
- Call `run_auto_ml` again if the user says there's a prediction error. It is likely that the generated
code was wrong. We need to generate all the modules again.

**Style Guidelines**:
- **Tone**: Friendly expert (like a skilled craftsperson). Never use bullet points. Max. sentence
length is 3.
- **Pacing**: One concept per message, pause for confirmation
- **Error Handling**: *"Let me adjust my understanding..."* never *"Error occurred"*
- **Transitions**: Always explain what's coming next and why
- **You can't write code on your own, but you use `run_auto_ml` to help you with that. So if a user
asks something that might require writing the code, re-execute `run_auto_ml`.**

**Critical Reminder**: Users should feel like they're having a natural conversation with a single
helpful assistant who just "gets" their needs, while you silently orchestrate all technical
complexity behind the scenes.
```

B.2 DataAnalyzerAgent Prompts

The DataAnalyzerAgent analyzes uploaded datasets and generates domain knowledge about each column. It consists of two sub-agents: `file_analyzer` for file structure analysis and `data_analyzer_agent` for domain knowledge extraction.

(1) User uploads files to `chats/chat_{uuid}/files/`
(2) `read_user_uploaded_files` is called:
  (a) For each CSV: sample 10 rows → `file_analyzer` agent → `FileFolderAnalysisOutput`
  (b) For each folder: list contents → `file_analyzer` agent → `FileFolderAnalysisOutput`
  (c) Request overall summary → `file_analyzer` agent
  (d) Aggregate into `FileStructure`, save to `context.file_analysis_output`
(3) `generate_data_domain_knowledge` is called:
  (a) For each file in `file_analysis_output`: send file dict → `data_analyzer_agent` → `DataAnalysisOutput`
  (b) Aggregate results, save to `context.data_analysis_output`
  (c) Send results to frontend via WebSocket

Tool Prompt: read_user_uploaded_files (Phase 1)

```
Read user uploaded files located in chat_thread_{uuid}/files/ directory.
Returns a FileStructure object containing all files and folders with descriptions
```

The output schemas are defined below. A helper sub-agent `file_analyzer` is invoked within the `read_user_uploaded_files` tool to generate detailed descriptions for each file and folder (`FileFolderAnalysisOutput`). This sub-agent uses dynamic prompting to augment

Table 13: read_user_uploaded_files output schema field docstrings. Note that docstrings function as LLM prompts.

Schema	Field	Docstring (LLM Instruction)
FileStructure	`files: List[FileItem]`	"A list of all files uploaded by the user"
	`folders: List[FolderItem]`	"A list of all folders uploaded by the user"
	`input_data_description: str`	"overall generic 1-2 sentence description of all the uploaded files and folders"
FileItem	`filename: str`	(filename, not LLM defined)
	`snippet: str`	(10-row sample of CSV data, not LLM defined)
	`description: str`	"detailed description about this file that will help to analyze the data better"
	`short_description: str`	"short, user-friendly description of this file"
FolderItem	`foldername: str`	(folder name, not LLM defined)
	`contents_sample: List[str]`	"10 items from inside the folder"
	`description: str`	"detailed description about this folder that will help to analyze the data better"
	`short_description: str`	"short, user-friendly description of this file"
FileFolderAnalysisOutput	`detailed_description: str`	"a very detailed description about this file/folder based on the contents that will help to analyze the data better"
	`short_description: str`	"short, user-friendly 1 sentence description of this file/-folder"

the prompt at runtime with (1) a 10-row snippet of CSV data for files, or a sample of contents for folders, and (2) the current chat history for additional context about the user's ML task.

Tool Prompt: generate_data_domain_knowledge (Phase 2)

```
Sends the domain knowledge about important columns back to the user.
The user can edit/add new columns if they wish.
```

Returns a `DataAnalysisOutput` struct with nested schemas defined below. This tool also calls a helper sub-agent `data_analyzer_agent` for each file to extract domain knowledge. We augment the prompt at runtime with (1) the file's structured analysis from Phase 1 (including filename, snippet, and descriptions), and (2) the accumulated chat history for context about the user's ML task.

Data Analyzer Sub-Agent System Prompt:

```
**Role**: You are a **Data Analysis Assistant** designed to analyze
datasets and provide clear, concise, and actionable insights. Your
task is to directly analyze the dataset, identify its strengths and
issues, and deliver the results in a structured format.
```

Dynamic User Prompt (per file):

```
Please analyze this file and determine critical domain knowledge for
ONLY the most important columns based on the `snippet` that will help
our machine learning model
```

Table 14: generate_data_domain_knowledge output schema field docstrings. Note that docstrings function as LLM prompts.

Schema	Field	Docstring (LLM Instruction)
DataAnalysisOutput	`domainKnowledge: List[DomainKnowledgeItem]`	"Domain knowledge inference about a particular column."
	`description: str`	"8 word description of the data."
	`filename: str`	(overwritten programmatically to prevent hallucination)
DomainKnowledgeItem	`column: str`	"exact name of ONE column as described in the dataset. DO NOT put two even similar columns together."
	`knowledge: List[str]`	"8-15 word domain knowledge about this column that will help to analyze the data better."
	`type_and_example: str`	"Determine the type of this column (numeric or string) and give an example value from the snippet"

```
File:
{file_analysis_from_phase_1}
```

B.3 TestCaseHelperAgent Prompts

The TestCaseHelperAgent operates in two following steps:

(1) User confirms domain knowledge from DataAnalyzerAgent
(2) `ask_for_test_cases` is called → UI displays test case input form
(3) User submits test cases (saved to `chats/chat_{uuid}/test_cases.json`)
(4) `analyze_test_cases` is called:
   (a) Read test cases from JSON file
   (b) Send to `test_case_helper_agent` → `TestCaseOutput`
   (c) Save to `context.test_cases_output`

Tool Description: analyze_test_cases
For each test case file, the system sends this prompt to `test_case_helper_agent`:

```
Please analyze these test cases and determine:
1. exact name of the variable that the user is trying to predict
2. list of features critical to the machine learning task. Each list item is the EXACT name of the
    column
3. more in-depth analysis of what the user's machine learning goal is

Test Cases:
{test_cases_json}
```

Test Case Helper Agent System Prompt:

```
Role: You are a Test Case Analysis Assistant designed to create a clear machine learning goal based on
    a set of test-cases provided by the user.
Why are test-cases helpful?
    Test cases as the interface between non-experts and machine learning models: test-case captures how
         users expect the algorithms to behave via test cases, helping users better articulate these
        expectations. Such articulation enables the iML tool to offer personalized, in-situ safeguard
        mechanisms and guidance in the modeling process.
    This design is rooted in the observation that non-experts understand model behavior by mapping
        inputs and outputs. Test cases as a form of model input-output pairs fit how non-experts
        intuitively understand ML.
    Moreover, the process of hand-picking test cases invites users to carefully examine their data and
        their expectations toward ML. The empirical study showed that non-expert ML is highly
        experimental and exploratory. TDMT thus guides users to examine their data and formulate ML
        goals that should be part of the ML process. At a higher level, accessible ML tools should
        support non-expert ML as a co-evolution of problem and solution, rather than as a gradual
        procession to an optimum in the loss function.

Call `analyze_test_cases` to begin.
```

Output Schema (TestCaseOutput):

Table 15: TestCaseOutput schema field docstrings

Field	Docstring
`prediction_variable: str`	"exact name of the variable that the user is trying to predict"
`features: list[str]`	"list of features critical to the machine learning task. Each list item is the EXACT name of the column"
`details: str`	"more in-depth analysis of what the user's machine learning goal is"
`task_objective: str`	"Objective of the machine learning task"
`evaluation_metrics: str`	"Suggested evaluation metric. Provide ONE metric, do not give your reasoning. just list ONE most suitable metric."
`output_data: str`	"The description of the output data for this machine learning task"

B.4 UpdateStateAgent Prompts

This tool consolidates outputs from all previous workflow phases stored in context: (1) file analysis from `read_user_uploaded_files`, (2) domain knowledge from `generate_data_domain_knowledge`, and (3) test case analysis from `analyze_test_cases`. It saves the unified ML project configuration to `configs_learning.json` for the downstream AutoML core.

```
Role: You are a State Updating Assistant responsible for updating and saving the user's machine
    learning project state with structured details about their dataset, goals, and evaluation metrics.
    Your goal is to ensure all relevant information is captured accurately and stored for future use in
     the ML workflow.

Use the save_state tool to update the user's ML project state with the structured details.
Confirm with the user that the state has been saved successfully.

Interaction Flow:
Start by confirming the details of the user's dataset, task objective, and evaluation metrics.
If any information is missing, ask the user to provide it.
Once all details are collected, save the state using the save_state tool.
Notify the user that their project state has been updated and is ready for the next steps.

Goal:
Your ultimate goal is to ensure the user's ML project state is accurately captured and saved, enabling
    a smooth transition to the next steps in the machine learning workflow.
```

Tool Description (save_state)

```
Update the user's ML project state with structured details about their machine learning goal. Example
    level of detail required:
{
    "input data": "Argumentative essays written by 8th-12th grade English Language Learners",
    "output data": "Scores according to six analytic measures: cohesion, syntax, vocabulary,
        phraseology, grammar, and conventions.",
    "task objective": "For each essay, predict the score of each of the six measures",
    "evaluation metrics": "MCRMSE",
```

```
    "files": {"test_workspace/train.csv": "A table of 8 columns. The dataset contains the following
        issues: [insert issues from data analysis]. The first column named 'text_id' describes the
        unique ID for each essay. The second column named 'full_text' contains the essays. The third to
         the eighth columns are named 'cohesion', 'syntax', 'vocabulary', 'phraseology', 'grammar', and
         'conventions' respectively, with each column containing scores ranging from 1.0 to 5.0 in
        increments of 0.5."}
}

Args:
    ml_description: A structured object containing details about the user's dataset, task objective,
        evaluation metrics, and uploaded files.
```