ECHO: Enhancing Conversational Explainable AI through Tool-Augmented Language Models

DOI: 10.1145/3734191
Handle: http://hdl.handle.net/1942/46344

ECHO: Enhancing Conversational Explainable AI through Tool-Augmented Language Models

SEBE VANBRABANT, UHasselt - Hasselt University, Digital Future Lab - Flanders Make, Belgium
GILLES EERLINGS, UHasselt - Hasselt University, Digital Future Lab - Flanders Make, Belgium
GUSTAVO ROVELO RUIZ, UHasselt - Hasselt University, Digital Future Lab - Flanders Make, Belgium
DAVY VANACKEN, UHasselt - Hasselt University, Digital Future Lab - Flanders Make, Belgium

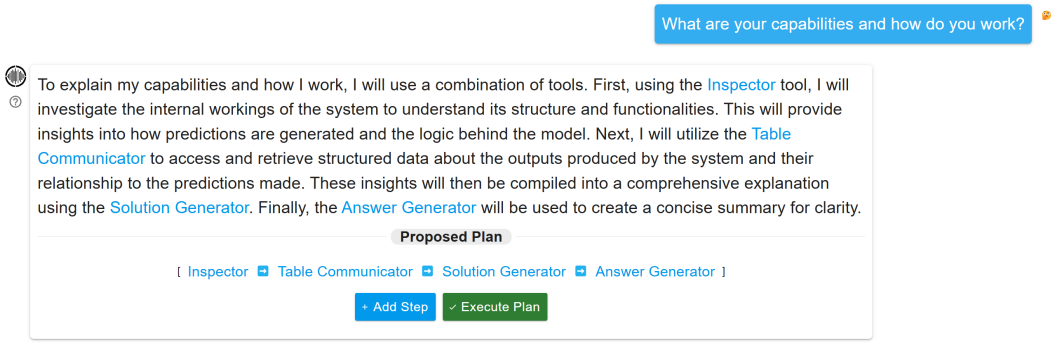


Fig. 1. ECHO can explain AI models by using a combination of tools mapped to various explanation types. These tools are then combined into a proposed plan that, when executed, addresses the user's question. This image is an example screenshot from ECHO's user interface, with ECHO itself explaining what it can do.

This paper introduces ECHO, an LLM-powered system framework to explore and interrogate the internals of AI models through tool-augmented language models. While traditional XAI methods typically offer a small and technical set of explanation types, ECHO advances the accessibility and usability of AI explanations through a conversational approach, combining LLMs with a collection of tools and a human-in-the-loop process. We identify various explanation types from the literature, for which we create a set of predefined tools for tabular data. Using a modular architecture, ECHO integrates these predefined tools with dynamically generated tools to interact with AI models, facilitating tailored explanations for a large variety of user queries. This paper details ECHO's design, implementation, and use cases, demonstrating its capabilities in the context of a movie recommender, healthcare decision tree and neural network for educational classification.

CCS Concepts: • **Human-centered computing** → **Interactive systems and tools**; *Natural language interfaces*; • **Computing methodologies** → *Artificial intelligence*.

Authors' Contact Information: Sebe Vanbrabant, sebe.vanbrabant@uhasselt.be, UHasselt - Hasselt University, Digital Future Lab - Flanders Make, Diepenbeek, Belgium; Gilles Eerlings, gilles.eerlings@uhasselt.be, UHasselt - Hasselt University, Digital Future Lab - Flanders Make, Diepenbeek, Belgium; Gustavo Rovelo Ruiz, gustavo.roveloruiz@uhasselt.be, UHasselt - Hasselt University, Digital Future Lab - Flanders Make, Diepenbeek, Belgium; Davy Vanacken, davy.vanacken@uhasselt.be, UHasselt - Hasselt University, Digital Future Lab - Flanders Make, Diepenbeek, Belgium.

ACM 2573-0142/2025/6-ARTEICS014
https://doi.org/10.1145/3734191

Additional Key Words and Phrases: Intelligibility, Interpretability, Explainability, Explainable AI, Artificial Intelligence, Machine Learning, Human-AI Interaction, Large Language Models

ACM Reference Format:
Sebe Vanbrabant, Gilles Eerlings, Gustavo Rovelo Ruiz, and Davy Vanacken. 2025. ECHO: Enhancing Conversational Explainable AI through Tool-Augmented Language Models. *Proc. ACM Hum.-Comput. Interact.* 9, 4, Article EICS014 (June 2025), 33 pages. https://doi.org/10.1145/3734191

1 Introduction

Artificial intelligence (AI) has become part of everyday life, with applications ranging from recommender systems [28] and energy availability forecasting [2] to healthcare support [7]. One subfield of AI that has gained significant attention in recent years is generative AI (GenAI). Most notable are the Large Language Models (LLMs): by using a massive amount of parameters, these generative models have significantly improved task-agnostic, few-shot performance [10], greatly impacting the state of the art of Natural Language Processing (NLP). LLMs have become accessible to the broad public through popular tools such as ChatGPT [55], Claude [5], and Gemini [64].

This rise of AI is not without its challenges. While early AI models, such as decision trees, were easily interpretable, their transparency has diminished over time due to the increasing complexity of, e.g., Deep Neural Networks (DNNs) and consequently LLMs [6, 48, 60]. These complex models act as a black box, making it unclear what they do 'under the hood' [67], as results are often presented without explanations, justifications, or indications of the uncertainties inherent in those results. This is especially problematic for end users, who not only lack a general understanding of AI's inner workings, but also have little to no control over its decision-making process [34].

Transparency and controllability are, however, essential factors to increase the users' trust in a system, with trust being a key determinant in the adoption of AI [7]. For example, in healthcare, a more interpretable method is preferred over a more accurate one, as the lack of transparency is considered too risky for patient use, particularly because it can obscure issues such as underlying biases [12]. Reflecting this concern, the European Union's GDPR establishes a 'right to explanation', granting users the right to explain algorithmic decisions made about them [24]. Moreover, the European 'AI Act', a regulatory framework that classifies AI systems based on the risks they pose [34], allows for regulating and even prohibiting systems that pose unacceptable risks.

To address these challenges and regulations, researchers look towards eXplainable Artificial Intelligence (XAI). XAI systems must be designed to explain their rationale to their end users, by providing clear and understandable insights about their functioning [78]. Moreover, XAI systems should communicate their strengths and weaknesses and indicate their likely future behavior, enabling users to (better) understand, appropriately trust, and effectively manage these AI systems [26]. A key issue with XAI is its tendency to make algorithms explainable for developers rather than offer usable, practical, and effective transparency that directly benefits end users [1, 51].

We propose leveraging the power of LLMs to address XAI's observed goals and challenges so that users can ask questions about AI models through natural language conversations. This concept has already demonstrated its potential for XAI [50, 51]. However, existing approaches to intelligibility are typically designed for specific explanation types, such as **Why** and **Why Not** [42, 47, 54, 57, 66] or **What If** [16, 75] questions. In contrast, we envision a modular, model-agnostic approach that supports a broad range of explanation types, where explanations are either generated using different XAI techniques or based on the underlying training data or model state. This approach is materialized in ECHO: an Explainable Chatbot for Human-centered Observation.

In this paper, we present the engineering contributions behind ECHO, which enable the development of explanatory systems for exploring and interrogating the internals of AI models through natural language conversations. We present both a conceptual framework that defines the

foundational concepts, structures, and requirements behind ECHO, as well as a software framework that combines a set of explanatory tools for tabular data with a human-in-the-loop process, to help understand, trust, and manage AI models trained on tabular data through conversational interaction. To this end, ECHO's workflow leverages both predefined and dynamically generated tools. Concretely, our contributions are as follows:

- **C1**: A conceptual framework that defines foundational concepts, structures, and requirements to interactively explain AI models through various explanation types.
- **C2**: A software framework designed to facilitate such interactive AI explanations, and its implementation of an Explainable Chatbot for Human-centered Observation. This includes:
  - **C2.1**: An informed selection and implementation of a set of predefined explanatory tools for tabular data, capable of addressing various explanation types, aligned with C1.
  - **C2.2**: A workflow that integrates predefined and dynamically generated tools, enabling conversational explanations of AI models through tool-augmented language models.
  - **C2.3**: A human-in-the-loop process that helps make conversational explanations interactive, transparent, and aligned with user intent.

This paper begins with an overview of current (interactive) XAI approaches and (tool-augmented) language models in Section 2. We then clarify our ideas through illustrative usage scenarios in Section 3, and outline the foundations needed to support these ideas in Section 4. We detail our framework and implementation in Section 5, outlining and motivating ECHO's mechanisms. We demonstrate ECHO's potential through various case studies in Section 6. Finally, we address ECHO's limitations and future directions in Section 7, and conclude with a summary of findings in Section 8.

2 Related Work

This section begins with a brief overview of the principles and applications of XAI. We then explore the role of interactivity in (X)AI and human-AI collaboration, as these factors also enhance AI understanding. Lastly, we highlight the inherent conversational strengths of LLMs and discuss how integrating tools in LLM-powered applications can enhance their reasoning capabilities.

2.1 Explainable AI

Explainability of AI models is usually achieved in one of two ways: either by explaining decisions of black-box models through external XAI techniques (e.g., post-hoc interpretability), or by using AI models that are human-interpretable by design (e.g., transparent-box design) [6, 25]. There is an observed trade-off between accuracy and interpretability: while transparent-box design inherently offers greater interpretability, it is usually less accurate compared to a black-box model [12, 26, 34, 36], with few exceptions [14]. However, some argue that increased interpretability facilitates iterative model refinement, eventually improving accuracy and inverting the trade-off [59].

Regarding transparent models, examples include logistic and linear regression, decision trees, and K-nearest neighbors, among others. Well-known examples of post-hoc interpretability are LIME [57] and SHAP [47]. The main concept of LIME is to explain a prediction by creating a local approximation of the model's behavior around this specific prediction and showing how features contribute to the prediction. SHAP, on the other hand, provides local explanations by visualizing the feature relevance for a particular prediction. Others, such as Anchors [58], aim to provide explanations by identifying sufficient conditions for achieving a particular prediction outcome. All three examples are model-agnostic methods: they can be plugged into any typical AI model to explain its predictions. Furthermore, Lim and Dey [40] propose methods to obtain eight explanation types (including **Why**, **Why Not** and **What If**) from popular decision models, namely rules, decision trees, naïve Bayes, and hidden Markov models.

For a comprehensive overview of XAI techniques and transparent models, which we consider out of the scope of this paper, we refer to Holzinger et al. [30] and Arrieta et al. [6], respectively.

2.2 From Passive XAI to Human-in-the-Loop

Explanations are only one factor in understanding AI models. Through an investigation where interactivity served as the primary mechanism for answering hypotheses and questions, Hohman et al. [29] found that interactivity is also fundamental. Various approaches have been developed to support such interactivity. For instance, explAIner [62] is a framework with a XAI pipeline that enables users to understand models, diagnose their limitations using different XAI methods, and optimize them. These steps are combined with various global monitoring and steering mechanisms to help developers interactively refine models. Furthermore, the What-If Tool [75] is a model-agnostic interactive approach, supporting the iterative exploration of transfactual explanations through a visual interface. The DARPA XAI project also resulted in an interactive system, EQUAS, consisting of multiple techniques to generate, evaluate, and modify explanations interactively [21]. One such technique evaluates competing answers by comparing their natural language explanations, improving both accuracy and explanation quality. Their research indicates that the utility of interactive explanations and meaningful human-machine collaboration go hand in hand.

In human-AI collaboration, users work together with the AI, combining the users' knowledge with the computational power of AI [7]. Interestingly, XAI's desired transparency and trust are also important aspects of human-AI collaboration. In their work on tree-based machine learning models, Lundberg et al. [46] found that local explanations can help satisfy transparency requirements, facilitate human-AI collaboration, and aid model development, debugging and monitoring. Holzinger et al. [30] remark that XAI models require new human-AI interfaces that enable contextual understanding and allow users to ask questions and counterfactuals. Herein, human-in-the-loop approaches are useful, by adding human experience and conceptual knowledge to AI processes.

2.3 Language Models and Prompting Techniques

LLMs indicate a significant advancement in NLP with the introduction of the transformer architecture [65], capable of understanding and generating human-like text. LLMs are trained on huge amounts of data, enabling them to perform various language-related tasks, such as answering questions (e.g., ChatGPT [55]) and writing code (e.g., Codestral [52]). Furthermore, LLMs have certain emergent abilities [73], such as in-context learning: the ability to learn from a few examples in the context through analogy [19]. More concretely, providing examples of desired behavior in a prompt (such as the desired question-answer format) can make an LLM follow that behavior.

In addition to classic input/output prompting, certain prompting strategies can further enhance the capabilities of an LLM. In Chain-of-Thought (CoT) prompting, off-the-shelf LLMs are instructed to provide their chains of thought along with an answer, offering insights into their reasoning and allowing LLMs to perform reasoning tasks more successfully [74]. For mathematical expressions, for example, CoT requires the LLM to generate both the mathematical expressions and the computation in each step. This can still lead to an incorrect answer, even though the chain of thought could be correct. For example, when asking an LLM to generate the first N Fibonacci numbers, it might correctly understand and explain the algorithm behind it, yet still fail at actually adding up the numbers. This problem inspired Program of Thoughts (PoT) prompting [15] and Program-Aided Language Models (PAL) [22]. These techniques propose generating a Python program that, when executed, produces the answer instead of directly having the LLM generate the answer, leading to a better performance. For example, if the LLM correctly writes down the aforementioned Fibonacci algorithm in Python code, the answer will also be correct. There are numerous other prompting

strategies. CoT prompting can, for instance, be further generalized into Tree of Thoughts (ToT) prompting [79] and Graph of Thoughts (GoT) prompting [8].

Another enhancement involves utilizing multiple models. Mixture-of-Agents, for example, entails sending one prompt to multiple small LLMs and aggregating their results with another model [69]. Chaining many small and open-source models can achieve results on the AlpacaEval 2.0, MT-Bench and FLASK benchmarks rivaling or outperforming larger LLMs.

2.4 Tool-Augmented Language Models

While powerful, LLMs have certain inherent limitations that the models themselves cannot overcome. For example, they lack access to (real-time) knowledge that is not part of their training set [35]. This requires external capabilities to retrieve relevant information from documents not included in their training data, such as Resource Augmented Retrieval (RAG) [37], Fusion-in-Decoder [32], or GenRead [80]. Moreover, LLMs are typically designed for specific tasks or modalities [27], which often restricts them to a text-to-text interface. Recent research has explored integrating LLMs into voice-based applications, such as ChatGPT-powered voice assistants [49] and domain-specific agents like 'TextileBot' [81]. However, integrating voice in data-rich explanations, such as XAI for models trained on tabular data, may be impractical due to the risk of information overload [31].

To enhance the capabilities of off-the-shelf text-to-text LLMs, **Tool-Augmented Language Models (TALMs)** have been introduced [56]. Instead of generating output directly from a given input, a TALM uses the given input to generate tool input for calling a tool's textual API. Tools can take on many forms: some can operate without parameters, such as tools that return the current time or weather, while others accept parameters, such as a search engine tool that enables LLMs to pass a query as input and returns the top N search results. The TALM only needs an abstract, textual representation of the tool's interface to use it. This results in a text-to-text interface that can significantly outperform non-augmented language models, since the LLM can rely on algorithmic output, similar to PoT and PAL prompting, and also on tools that give access to real-time data.

With the appropriate tools, TALMs can even support complex, visual, multimodal queries, despite the LLM having a text-to-text interface. After all, the LLM can interact with tools by relying purely on textual inputs. For example, Visual ChatGPT [76] combines ChatGPT with various Visual Foundation Models as tools to handle visual tasks, such as text-to-image generation and editing. Another example, ViperGPT [63], tackles complex visual queries by leveraging code generation, using the API of vision models through a Python interpreter. Instead of generating solutions in domain-specific languages like Python, Chameleon [44] breaks down the problem into a textual sequence of tool calls, using a planner to determine the execution flow. Chameleon's planning approach makes the process human-readable and debuggable for users with limited programming experience, which is promising to enhance the transparency and controllability of LLMs.

2.5 TALMs for Engineering Interactive and Explainable Systems

Building on the foundations of general-purpose TALMs, it is possible to develop task-specific systems, such as the RecMind recommender system [71]. RecMind relies on an LLM for overall reasoning, *planning* to break down tasks into smaller subtasks, *memory* for information retention, and *tools* to enhance the model's memory and reasoning capabilities. Memory enables RecMind to access knowledge beyond the LLM's internal knowledge, consisting of *Personalized Knowledge*, such as a dataset of item reviews, and *World Knowledge*, consisting of metadata on these items and real-time knowledge acquired using a web search tool. Using an LLM as a recommender system can make the recommendation process more interactive and explainable, as the user can freely ask follow-up questions to understand recommendations better [23]. Since the recommender only

relies on in-context learning through the LLM, the recommendation and explanation processes remain transparent, with no hidden factors influencing the decisions.

The strengths of LLMs in terms of interactivity and explainability are also evident in x-[plAIn] [51], which uses an LLM to generate audience-specific summaries of various XAI methods, namely LIME [57], SHAP [47], and Grad-CAM [61]. x-[plAIn] is XAI method-agnostic, requiring no specific training or adaptation for a XAI method, and is thus broadly applicable. An LLM summarizes its explanations, adapted to the user's knowledge level and interests. This augments the decision-making processes for end users who actively engage with the results. This aspect is particularly useful when the technical complexity of XAI makes it difficult for users to understand and act upon the explanations. In their study, AI experts and end users were generally favorable to x-[plAIn]'s explanations compared to conventional representations of XAI methods. This aligns with the findings of Martens et al. [50], who combine SHAP explanations with an LLM to generate narratives about AI predictions in SHAPstories. They conclude that these stories convincingly and effectively communicate explanations to a general audience, helping them to answer comprehension questions significantly more often than when using SHAP values alone.

These systems and approaches highlight the potential of combining LLMs with XAI methods to enhance their interactivity and accessibility. However, a modular AI model-agnostic *and* XAI method-agnostic approach would improve the generalizability to other AI models and XAI methods.

3 Illustrative Usage Scenarios of Conversational XAI

It is evident that multiple factors impact effective XAI and human-AI collaboration. To work effectively with AI systems, they should explain their rationale and indicate their strengths, weaknesses, and future behavior [26, 78]. Current XAI techniques, broadly speaking, cover only a limited set of explanation types. Moreover, XAI should offer interactivity on top of explanations to enhance users' understanding of AI models and foster meaningful human-AI collaboration [21, 29].

We conclude that an interactive, *conversational* approach would benefit XAI and envision a system capable of answering as many questions as possible, within reasonable limits, using accessible natural language. LLMs are well-suited to explain behavior and support follow-up questions, which enables interactions for as long as needed to understand both the AI model and the explanations. Furthermore, technical tools can be integrated into an LLM, while ECHO leverages its powerful in-context learning and accessible chat interface. We thus propose a novel tool-augmented language model to facilitate interactive XAI through textual conversation. To illustrate ECHO's general approach, we adapted the well-known XAI figure from DARPA, as shown in Fig. 2, and we present three short scenarios that will serve as running examples throughout the remainder of the paper.

Scenario 1: Movie Recommender System. Alice, a data analyst for smart TV recommender systems, wants to understand the rationale behind the system's frequent science fiction and action movie recommendations, which data was used for them, and whether the TV is certain of its predictions. In this way, Alice can troubleshoot the recommender or its ratings to get more relevant recommendations. Using ECHO's conversational interface, Alice asks questions in natural language about why certain movies are recommended, how the recommender interprets her preferences, and ways to prioritize her preferences for future suggestions. Alice does not need to delve into the recommender and its training data herself, nor does she need to find the appropriate XAI techniques to address her questions: ECHO generates information until her questions have been sufficiently answered.

Scenario 2: Heart Disease Classifier. A researcher reviewing a heart disease classifier wants to audit the model's behavior regarding a high-risk prediction. Through iterative and interactive collaboration, he gains insight into the contributing parameters and underlying reasons. ECHO generates hypothetical scenarios (e.g., lowering cholesterol) and audits the model's structure — a

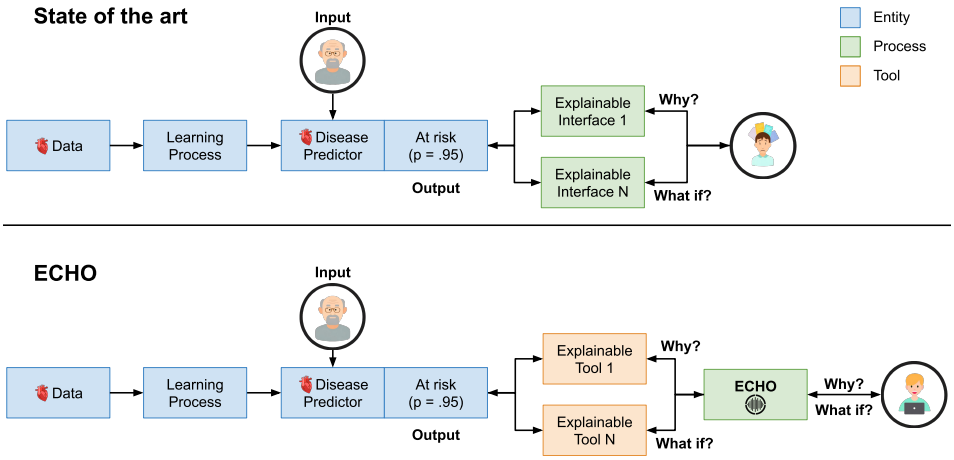


Fig. 2. A conceptual illustration of ECHO, based on DARPA's XAI vision [3, 77], using a heart disease predictor as an example. An AI model is trained on a dataset through a learning process to make predictions for new input. Here, the model predicts with 95% certainty that a user is at risk of heart disease. To explore this prediction, users must understand what *explainability* options exist, how to invoke them, and how to interpret them. ECHO addresses this through a conversational interface that connects the AI model, XAI tools and the user, allowing interactive communication and explanations.

decision tree — and its parameters so that the researcher can focus solely on answering his questions without worrying about underlying XAI tools or data manipulation. While not a replacement for medical advice, ECHO can be integrated into clinical workflows to support shared decision-making between patients and healthcare providers, as demonstrated in technology-supported cardiac secondary prevention settings before, during or after an encounter with the specialist [9].

Scenario 3: Student Performance Classifier. Carol, an education technology specialist, is developing a neural network that preemptively identifies students at risk of failing courses. The model predicts failure for Dan, a student who does not appear to be an outlier compared to other students. Carol uses ECHO to understand the prediction, test the impact of potential interventions, and explore general patterns among at-risk students to identify solutions. ECHO examines the neural network and helps her identify parameters contributing to Dan's predicted outcome by invoking and synthesizing traditional XAI approaches. By providing direct answers to Carol's questions, ECHO enables a focused, efficient conversation, helping her validate and refine the model.

4 Conceptual Framework for Conversational XAI

Before delving into ECHO's conversational framework in Section 5, we first describe some foundational concepts, structures and requirements to guide the design of such a framework. First, we define the terms *explainer* and *explanandum*. Then, we define a blueprint of an explanandum on which ECHO can operate. Lastly, we examine different explanation types from the literature to identify the necessary capabilities of ECHO.

4.1 Explainer and Explanandum

To distinguish between the AI model being explained (e.g., a recommender system) and the AI system providing the explanations (e.g., ECHO), we use the terms **explanandum** and **explainer**, respectively. The term 'explainer' is often used in literature, particularly in reference to post-hoc

explainers or explainer systems (e.g., Ferguson et al. [21], Lim and Dey [40], Spinner et al. [62]). The term 'explanandum' (plural: 'explananda') is less commonly used, but Cabitza et al. [11] use it to describe "the thing or phenomenon to be explained". In our context, this refers to the AI model whose behavior needs to be explained, such as the recommender system.

In concrete terms, we envision the explainer as an interactive conversational agent capable of answering user questions about the explanandum through a set of tools mapped to various explanation types. This subdivision is conceptually similar to the explAIner framework proposed by Spinner et al. [62]: their 'XAI Input' corresponds to the explanandum, their 'XAI Method' maps to the tools within the explainer that generate explanations, and their 'XAI Output' is represented by the conversational interface of the explainer, and its explanations will be *verbalizations*.

4.2 Blueprint of an Explanandum

Following the definition of an explanandum, we now define its fundamental structure and components, as depicted in Fig. 3. Since an explanandum is a rudimentary representation of an AI model, its key components include its **Inputs** and **Outputs**, and a **Predict** method that connects them. This **Predict** method usually requires preparatory steps, such as a **Train** or **Fit** method that leverages the explanandum's internal **Data**. The structure and content of the data depend on the type of AI model and the problem it tries to solve. For a recommender system using collaborative filtering, for example, the internal data consists of user-item interaction records. Its **Predict** method takes a user ID as **Inputs**, and **Outputs** a list of recommended movies with predicted ratings.

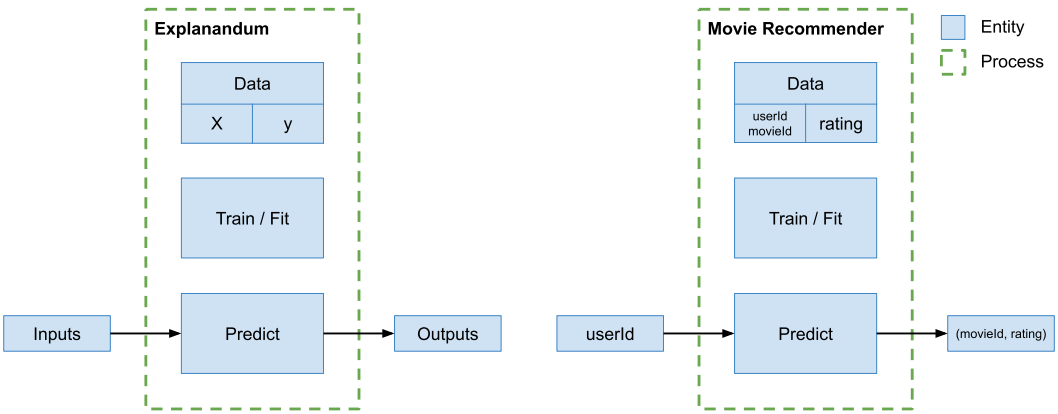


Fig. 3. A schematic overview of our blueprint for a rudimentary, generic explanandum (left) and a recommender system adhering to this blueprint (right). The explanandum requires training data, a *Train/Fit* method to train or fit its internal structure, and a *Predict* method that uses this internal structure for new predictions. When given new inputs, the *Predict* method generates new outputs.

4.3 Explanation Types

To enable ECHO to generate meaningful explanations for an explanandum, we conduct a literature review to identify potential explanation types that users might seek. We identified the explanation types from impactful papers related to XAI and intelligibility, which all cite the influential work of Lim and Dey [39–43] as fundamental inspiration. Like many other papers in this domain, we build upon their foundational work. We summarize these explanation types and their sources in Table 1. These explanation types can be divided into those describing the **System State** and those describing

the **Inference Mechanism** [68]. We see the System State as system-to-user information that users can consult as-is, such as dataset structure and contents, prediction certainty, or explanations about the learning process. In contrast, we see the Inference Mechanism functions as a user-to-system query to interrogate the explanandum's System State and behavior.

Table 1. Mapping of identified explanation types to their corresponding sources. A check mark (✓) indicates the presence of an explanation type in the respective source, while a cross (✗) denotes its absence. Sources that did not consider more than two explanation types or exclusively recited explanation types mentioned in other sources are not included in this overview.

Explanation Type	[39]	[42]	[40]	[41]	[43]	[53]	[78]	**ECHO**
System State: System-to-User								
Input(s)	✓	✗	✓	✓	✓	✗	✓	✓
Output(s)	✓	✗	✓	✓	✓	✗	✓	✓
Certainty	✓	✗	✓	✓	✓	✗	✓	✓
Situation	✓	✗	✗	✓	✗	✗	✗	✗
Visualization	✓	✗	✗	✗	✗	✗	✗	✗
Control	✓	✗	✗	✗	✗	✗	✗	✓
Application	✓	✗	✗	✗	✗	✗	✗	✓
Model	✓	✗	✗	✗	✗	✗	✗	✓
History	✗	✗	✗	✓	✗	✗	✗	✓
Description	✗	✗	✗	✓	✗	✗	✗	✓
Example	✗	✗	✗	✗	✗	✗	✓	✓
Inference Mechanism: User-to-System								
What If	✓	✓	✓	✓	✓	✓	✓	✓
Why	✓	✓	✓	✓	✓	✓	✓	✓
Why Not	✓	✓	✓	✓	✓	✓	✓	✓
How To	✗	✓	✓	✗	✓	✓	✓	✓
What	✗	✓	✓	✓	✗	✗	✗	✓
How	✓	✗	✗	✗	✗	✓	✓	✓
What Else	✓	✗	✗	✗	✗	✓	✗	✓
When	✗	✗	✗	✓	✓	✗	✗	✓
What Output	✗	✗	✗	✗	✓	✗	✗	✓

We selected 18 explanation types to support in ECHO. We omitted *Visualization*, as it falls outside our text-based conversational scope, and *Situation*, which is primarily relevant in real-world high-criticality scenarios requiring situational awareness [39], for which LLMs are not advisable due to their inherent limitations in real-time decision making. Table 1 highlights that **What If**, **Why** and **Why Not** are present in all reviewed sources, indicating their versatility and value. **Why** and **Why Not** frequently appear together as fundamental explanation types (e.g., Lim et al. [42], Myers et al. [54], Vermeulen et al. [66]), while **What If** has been used for feedforward (e.g., Coppers et al. [16]) and interactive machine learning tools (e.g., Wexler et al. [75]). Here are the 18 selected explanation types, with our interpretation of their meaning and their intended purpose within the explainer-explanandum workflow, for which we aggregate and rephrase descriptions from various sources:

System State: System-to-User

Input(s) explanations clarify the input data the explanandum uses to make predictions, helping users understand the scope. Concretely, these are the Predict method's input parameters.

Output(s) explanations clarify the explanandum's output space, conveying what it can do or produce or what states it can be in. Concretely, these are the Predict method's outputs.

Certainty explanations convey how (un)certain the explanandum is of its predictions, helping users decide how much to trust the result and potentially consider other outcomes.

Control explanations support users in changing model (hyper)parameters that affect decision-making. These can be developer-set hyperparameters or system-learned parameters. This, however, carries the risk that users can degrade the explanandum's performance by changing parameters incorrectly, so they have to be aware of this potential issue.

Application explanations clarify the explanandum's workings and mechanism, which is recommended to combat low trust and high uncertainty within the explanandum.

Model explanations should clarify the conceptual model of the explanandum by answering user questions. This can be done through the *Inference Mechanisms* described below.

History explanations should indicate past inferences, such as the chat history. They can also be part of the explanandum's training data, for example, if timestamps are associated with ratings in a movie recommender system.

Description explanations relate to the meaning of the context terms and values. The explainer should not only be able to give the context, but also explain what this context means.

Example explanations should reveal similar input-output pairs from the explanandum and its dataset, such as similar input values that lead to the same output, or similar output values resulting from the same input.

Inference Mechanism: User-to-System

What If explanations, also called transfactual explanations, reveal what would happen if there was a change in input values (e.g., the user specifies new inputs, triggering a new prediction). ECHO generates these explanations by changing the **Inputs**, allowing users to analyze the new **Outputs** and thus the impact of the changes.

Why explanations indicate why the explanandum predicted the **Outputs** for current or previous **Inputs**. ECHO uses a mix of model interpretation (e.g., explain how nearest neighbor recommendations work) and existing XAI methods (e.g., explain outputs of a neural network).

Why Not explanations, also called contrastive explanations, indicate why the explanandum did not make a certain decision and thus why a certain **Output** value was not given, based on the **Inputs**. ECHO handles these similarly to the **Why** explanations.

How To explanations answer the question "In general, how can the system produce alternative **Output** value X?", providing a counterfactual explanation of what needs to change in the **Inputs** for the alternative **Output** to happen. ECHO generates such explanations by finding the nearest neighbor for a certain **Input** with the desired **Output** X.

What explanations make the *System State* more explicit. Asking the explainer what data the explanandum used in a prediction, for example, can reveal the explanandum's **Inputs**. ECHO generates these explanations by inspecting the explanandum and its dataset.

How explanations clarify how the explanandum arrived at its outcomes. ECHO generates these by identifying and explaining the properties and processes in the *System State*.

What Else explanations, also called explanation by example, identify similar **Input** samples that the explanandum previously used to generate the same or similar **Output**. These explanation types produce an **Example**. ECHO handles this by finding the nearest neighbor **Inputs** with the same **Outputs** as a previous prediction.

When explanations convey the cases in which the explanandum produces a particular **Output**. They do not offer counterfactuals, but rather more general cases in which particular **Output** occurs. ECHO identifies the specific conditions in which X occurs in the **Outputs**. The explainer can then generalize the **Inputs** that lead to X in the **Outputs**.

What Output explanations are considered a combination of **What** and **Outputs** for our purposes, and thus redundant as a separate type.

From these explanation types, we can derive what tools are needed to support them. For example, a conversation about any explanandum benefits from a tool that *retrieves general knowledge* about the explanandum to address **What**-queries, explaining how a K-nearest neighbor (KNN) algorithm recommender works, or perhaps broadly explaining the workings of a neural network. For the user to ask this question, they also need to be able to ask about what AI model is being used in the first place. To this end, ECHO needs a tool that can *inspect* the source code of the explanandum and find out what type of model is being used. Furthermore, consider a user inquiring about the data that was used to train this model, such as by asking **Example**s of previous behavior through a **What Else** query. In this case, ECHO needs a way to *communicate with the tabular datasets* that the explanandum was trained on. However, some explanations require a tool that would depend on the explanandum's implementation instead, such as a Predict tool to answer **What If**-queries, or XAI methods to address **Why** and **Why Not**. To this end, we propose a *dynamic way to create tools* from an explanandum. We will address these concepts in Section 5. Although we focus on tabular data with ECHO, our approaches to addressing each explanation type are generalizable to other modalities and use cases that implement conversational explainability.

5 ECHO's Tool-Augmented Conversational Framework for XAI

This section introduces the architecture and subcomponents of ECHO, which relies on a TALM to support human-AI collaboration, following the blueprint of an explanandum from Section 4.2 and explanation types from Section 4.3. Since AI models and their training data can take on many forms, we set our scope to tabular data. Tabular data is very common and relatively straightforward to understand for both language models and users, in contrast to image or time series data. This focus simplifies the creation and verification of ECHO, without needing many domain-specific tools. However, ECHO's modular approach allows developers to integrate other tools seamlessly. For example, tools that interact with tabular data are interchangeable with tools that interact with images, if they can address the same explanation types.

5.1 ECHO's Framework Architecture

We illustrate ECHO's execution flow, shown in Fig. 4, with a concrete example. Consider a user wanting to explain a recommender system. The process starts with initializing ECHO with an inventory of tools to communicate with that recommender system, such as a tool to interact with its datasets. If the recommender system adheres to the blueprint specified in Section 4.2, its methods can be converted into tools through our *tool generation*, resulting in Predict and Rate tools. These tools are combined with a set of predefined tools that offer general functionality, such as a Table Communicator to interact with the recommender system's datasets and an Inspector to inspect its source code. ECHO is now initialized with the tools it needs to answer the user's questions about the recommender system. When the user asks ECHO "What user data is used for making recommendations?", a query is sent to ECHO's planner. The planner then returns a plan — a sequence of tools (e.g., containing a call to the Inspector tool to find out what data is being used to train the model and a call to the Table Communicator to detail further what is included in this data) — along with an explanation motivating the choice of tools and their ordering. The user can either modify this plan or execute it through the executor. Once executed, the final answer and execution flow are returned for the user to inspect. After this back-and-forth, the user can ask a follow-up question, such as "What movies do you recommend to Alice?", to trigger the explanation types outlined in Table 1. We provide more usage examples in Appendix A.

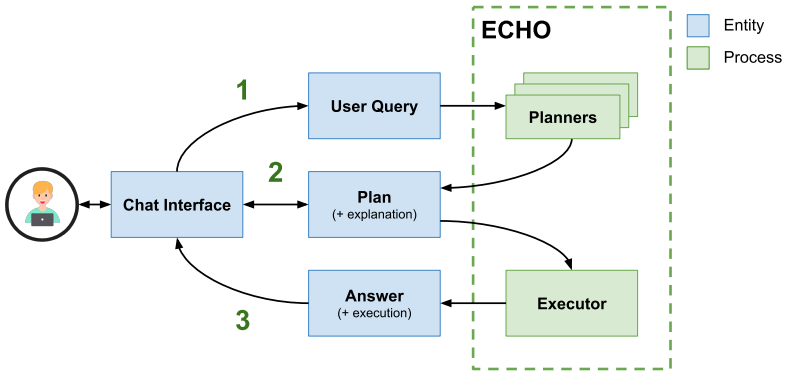


Fig. 4. The execution flow of ECHO from the user's point of view, with the green numbers indicating the order of actions. The user starts by sending a query to ECHO's planner (composed of multiple Mixture-of-Agents planners, see Section 5.4), which generates a plan and proposes this to the user, as illustrated in Fig. 5. The user can send the (potentially modified) plan back to ECHO, after which the executor executes the plan and returns the results, as shown in Fig. 8.

Now that ECHO's overall execution flow has been outlined, we turn to its architecture. TALMs and their tools are usually created for specific goals and benchmarks, making it tedious to extend these TALMs to other use cases. Their tools are not made to communicate with arbitrary AI models. To create a TALM-based explainer capable of interacting with any AI model adhering to the blueprint and class analogy of Section 4.2, we thus require a more dynamic approach.

To achieve this, ECHO relies on dynamically generated tools (further explained in Section 5.3). In the example above, ECHO's tool generation created the Rate and Predict tools based on the explanandum. ECHO combines these generated tools with a set of predefined tools designed to support the various explanation types (further explained in Section 5.2), as shown in Table 2. These explanation types are grounded in literature, whereas the target tools and components aim to support the strategies discussed in Section 4.3. The target tools are a small but capable set of tools that make this mapping possible in the context of a TALM. Like in Chameleon, we developed tools according to their ability to solve the problem in a technical sense. Their ability to address the explanation types is empirically validated in Appendix A, which covers a subset of our use cases and lines of questioning used to test ECHO. In the example above, the Table Communicator and Inspector are predefined tools. When a user asks ECHO a question, a planner mechanism suggests a sequence of tools to give an answer. Each tool operates within a modular execution flow, where it can access outputs from previous tools and produce outputs for the next tool. To increase the user's control over the process, ECHO allows the user to modify the suggested plan before executing it. This two-step process is detailed in Section 5.4 and Section 5.5.

5.2 Predefined Tools

To support the various explanation types discussed in Section 4.3, ECHO uses a set of *predefined tools*, which are built-in tools that give ECHO its ability to generate explanations for AI models adhering to the blueprint outlined in Section 4.2. Our set of predefined tools is inspired by the Chameleon framework [44], which, at the point of writing, is the state-of-the-art framework for the TabMWP dataset. This dataset involves math word problems that require mathematical reasoning on both textual and tabular data [45]. We built upon Chameleon's workflow and its relevant tools,

Table 2. Mapping the various explanation types identified in Table 1 to the tools designed to support them. Note that we only list the tools necessary to support the explanation type. Knowledge Retrieval, Solution Generator, and Answer Generator can or will be used in all cases and, therefore, are not repeated everywhere. Tools are highlighted in orange, while other aspects of ECHO (such as tool generation) are indicated in purple.

Explanation Type	Target Tool(s) and Components
System State: System-to-User	
Inputs	Inspector, Table Communicator, Tool Generation
Outputs	Inspector, Table Communicator, Tool Generation
Certainty	Tool Generation (e.g., the Predict tool)
Control	Tool Generation
Application	Inspector
Model	Inspector
History	Table Communicator, Chat History
Description	Knowledge Retrieval
Example	Table Communicator
Inference Mechanism: User-to-System	
What If	Tool Generation (Predict tool)
Why	Inspector, Table Communicator, Tool Generation (XAI tools)
Why Not	Inspector, Table Communicator, Tool Generation (XAI tools)
How To	Table Communicator
What	Inspector, Table Communicator
How	Inspector, Table Communicator
What Else	Table Communicator
When	Table Communicator

namely the Knowledge Retrieval, Solution Generator, and Answer Generator, and integrated them with a set of newly created tools to interact with explananda and their tabular datasets:

Knowledge Retrieval generates additional relevant knowledge using an LLM. This tool functions similarly to a search engine, but only relies on an LLM's internal knowledge to gather information. For example, if a user asks how a certain model works (e.g., a neural network), the Knowledge Retrieval tool will retrieve knowledge about this model to enhance the response. This tool is meant to provide ECHO with the necessary knowledge to answer questions about explanation types **Model** and **Application** of the explanandum.

Inspector enables ECHO to inspect the implementation details of the explanandum. Whereas the Knowledge Retrieval tool provides general knowledge about the explanandum (e.g., to answer "What is a neural network?"), the Inspector retrieves information about the explanandum's actual implementation (e.g., to answer "How many layers does the network have?"). For this, we use Retrieval-Augmented Generation (RAG) [37], with the explanandum's source code as the retrievable document. The Inspector is similar to a search query on the explanandum's code.

Table Communicator enables ECHO to interact with tabular datasets associated with the explanandum. In a recommender system, for example, a table contains interactions between users and items (i.e., ratings). Here, the Table Communicator could help a user to find similar users by returning a table of users with a similar rating history, accompanied by a motivation to help the user understand ECHO's reasoning. For this tool, we adapted the PoT prompt of Chen et al. [15] to instruct an LLM to generate a function that returns a table that addresses the user's query and a motivation. To achieve more accurate results, our prompt also includes instructions to align the response with the explanation types in Table 1, as well as few-shot

examples of tool usage. The generated code uses pandas[1] to interact with the datasets and is executed in a Python REPL.

Solution Generator uses CoT prompting to synthesize a final answer, based on the context built during the execution of previous tools. This tool considers the user's question, changes that may have been made to Predict's **Input** parameters, and the entire execution flow of the plan, which contains the results of preceding tool calls. The Solution Generator aggregates these intermediate results and derives a final answer using CoT reasoning. This process, on the one hand, leverages CoT prompting to reason towards an answer more effectively and, on the other hand, offers the user transparency about this reasoning process.

Answer Generator extracts the final answer from the solution generated by the Solution Generator, and presents it as the answer message via the UI.

Predefined tools are inherently *model-agnostic*, supporting either general explanatory functionality that applies to any explanandum (e.g., the Table Communicator for dataset querying), or use cases that cannot be handled by tool generation (e.g., tools requiring access to the context of the explainer, such as the Solution Generator). Tools sometimes invoke an LLM internally to fulfill their purpose. For example, the Solution Generator does not have input parameters, but will internally use an LLM to synthesize a solution from the execution context.

5.3 Dynamically Generated Tools

Since existing TALMs are typically designed for predefined tasks, with tools tailored to those specific tasks, they do not fit our case, where each explanandum has its own unique functionality. For example, a recommender system might include a Rate tool for user feedback, whereas a black-box system may depend on XAI tools for interpretability. Moreover, the implementation of a Predict tool varies significantly across different AI models, emphasizing the need for a flexible approach that can accommodate these diverse requirements.

To enable communication between ECHO and an explanandum, ECHO relies on *tool generation*. In contrast to the predefined tools, this tool generation dynamically generates *model-specific* tools by taking an explanandum that meets the requirements of Section 4.2 (a class) and transforming its functionalities (methods) into tools. It generates, for example, Predict and Rate tools from the methods of a recommender system. More specifically, we convert each method of an explanandum into a tool with the following API documentation:

- **Name** match the method's name, necessary for the planner to uniquely identify the tool.
- **Description** is generated from the method body through an LLM, briefly summarizing what the method does. This helps the planner understand the tool's purpose and enables it to select and use tools that effectively address the user's question.
- **Parameters** correspond to the method's signature. This field helps the planner understand what inputs a tool requires to execute.

This textual interface resembles the mandatory parameters of the API documentation schema of TaskMatrix.AI [38]. We also use this interface internally for the predefined tools. Dynamic tool generation can also support **Certainty** and **Control** explanation types, tailored to the specifics of an explanandum. For example, a Predict tool can return its **Certainty** alongside the prediction, whereas another tool may allow modifications to a neural network's hyperparameters and retraining, e.g., through an Add Layer tool that offers more **Control** over the model's structure.

[1] https://pandas.pydata.org/

5.4 Planner: Proposing a Sequence of Tools

By leveraging both predefined and dynamically generated tools, ECHO can execute sequences of tools to address the user's queries. These sequences are proposed by a planner mechanism, inspired by the Chameleon framework [44] and shown in Fig. 5. The planner's prompt is assembled by combining ECHO's tools and their textual interface, as specified in Section 5.2, as well as usage examples of model-agnostic questions using the predefined tools, and the user's chat history.

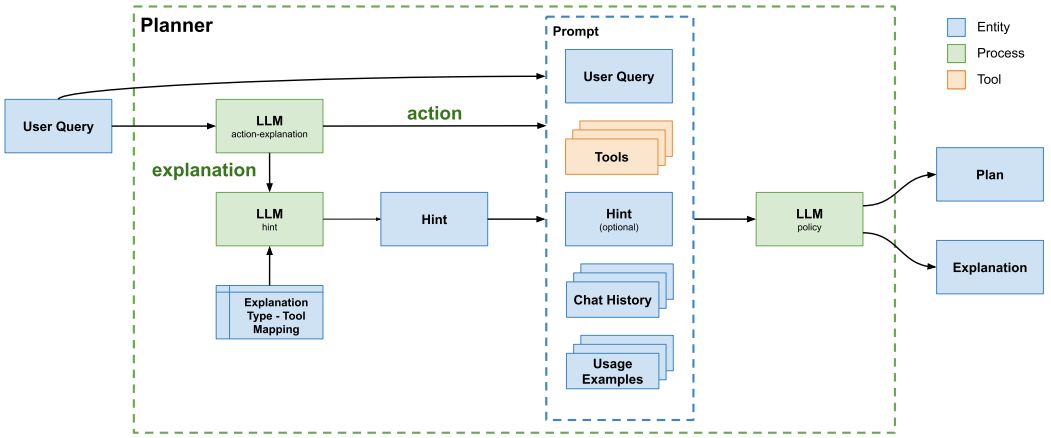


Fig. 5. A schematic overview of the planner. The user query is encapsulated in a prompt containing the chat history, a textual overview of all available tool interfaces, and usage examples in the context of few-shot learning, plus an optional hint about appropriate tools when the user requests an explanation. This prompt is sent to an LLM, which returns a formatted response with a plan and explanation.

Before invoking the planner LLM, the query is first classified as either an *action* (e.g., "Rate a movie.") or an *explanation* (e.g., "Why did you recommend this movie?"). If the user asks for an explanation, ECHO generates a *hint* that suggests an appropriate tool based on the mapping outlined in Table 2, which enhances the planner's ability to recommend the most suitable tools.

If the user asks for a prediction, ECHO might, for example, answer with [Predict, Solution Generator, Answer Generator], accompanied by a reasoning on why these tools were selected and arranged in this specific order. ECHO's CoT prompting enhances the transparency of the planning process, in contrast to Chameleon's planner, which only gives the chosen tools without an explanation. The CoT prompting also improves the quality of the generated plans by requiring the LLM to motivate the tool execution flow. Furthermore, ECHO uses few-shot prompting: the planner is given examples of how different tools map to their intended use.

To further improve the quality of the generated plans, the planner employs a Mixture-of-Agents (MoA) approach [69], providing multiple options and a reasoning for each plan. We empirically determined that using a single planner left too much room for error, such as using too many, too few, or simply irrelevant tools. By using MoA, the planner transparently reasons toward more well-rounded plans. In the end, ECHO aggregates the intermediate plans into one plan through another LLM call prompted to create a single, high-quality response from the intermediate plans.

The planner sequence ends with a call to the Solution Generator, followed by the Answer Generator, as shown in Fig. 6. The Solution Generator constructs a solution by synthesizing intermediate results and applying CoT prompting to reason toward a final answer, which the Answer Generator then extracts. The planner can also decide to answer questions directly, using

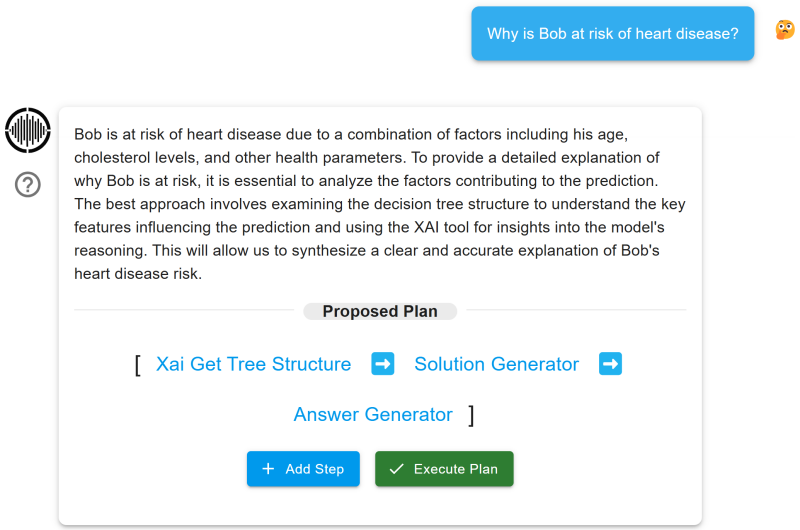


Fig. 6. An example plan, with ECHO's answer to a **Why**-question from scenario 2 in Section 3. We show the intermediate plans that led to this final, aggregated plan in Fig. 7.

only one singular Direct Answer entry in the plan. This can be useful when the user's query is unrelated to the explanandum, such as a general question about what ECHO can do.

5.5 Executor: Plan Execution Flow

Before executing a plan, ECHO presents it to the user as an editable list of items. Users may then modify the plan through CRUD operations (i.e., Create, Read, Update, and Delete), using drag-and-drop for an efficient workflow. Reading operations on the tools gives access to the same API documentation available to the LLM, as discussed in Section 5.3, displaying this information in a tooltip. This is useful for users who, for instance, want to customize the final plan based on one of the proposed plans from the MoA planners. These proposed plans can be accessed once the final plan is generated and are displayed at the user's request, as shown in Fig. 7. This also allows the user to address cases where the planner includes tools in which the user is not interested or omits tools the user would like to have incorporated. The user can also view all the information that the internal LLM can access about the tools. By allowing users to modify the plan and view the tools, we facilitate a human-in-the-loop approach to tool-augmented language models, which is a step toward human-AI collaboration with LLMs.

If the final plan includes a call to Predict, an additional subsystem is invoked to parse this tool's input parameters from the user's query. The user is shown this Input State of parameters and is prompted to fill in blanks. For example, if a user requests a prediction, but ECHO needs their user ID first, the user is prompted to fill in their user ID in the user interface before being able to execute the plan. This subsystem is used for two reasons: firstly, so that a user does not need to specify all parameters explicitly in their query, and secondly, so that users can review and modify the Predict tool's inputs before execution, further enhancing the human-in-the-loop potential of ECHO.

Upon the user's request, the (modified) plan will be executed in sequence. Each tool iteratively updates ECHO's internal state, before returning the Answer Generator's result to the user. ECHO uses an LLM to execute tools that require parameters, generating these parameters based on the user's query and the context provided in the plan. In the movie recommender example, a Rate

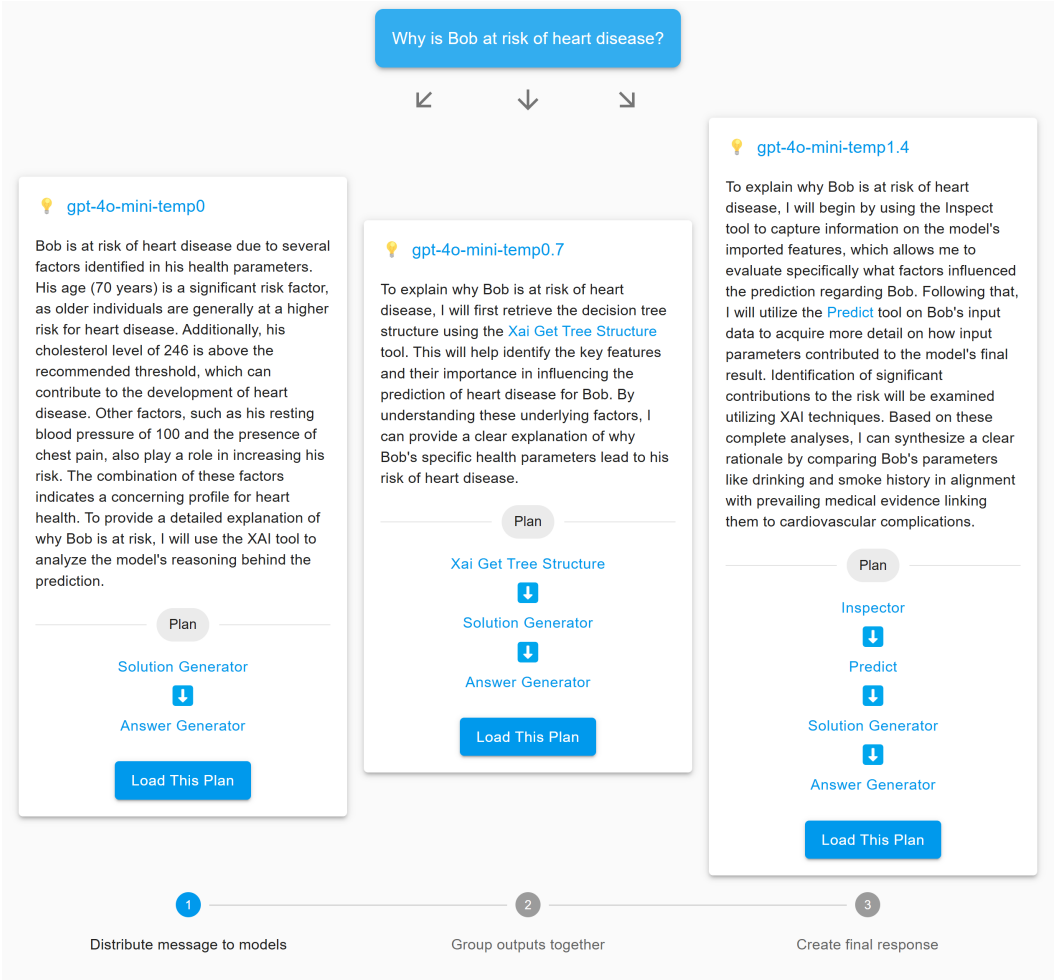


Fig. 7. A comparison view of ECHO's proposed plans. When asked why Bob is at risk of heart disease, the MoA planners each propose a plan to answer the question. The aggregation process is visually illustrated through the three steps at the bottom of the figure. In this case, the aggregated plan will be [XAI Get Tree Structure, Solution Generator, Answer Generator]. The plan variety highlights why the MoA approach is necessary, as only the aggregated (and coincidentally middle) plan would have led to a correct output, despite each planner's seemingly plausible motivation. The aggregated plan is shown in Fig. 6.

tool expects a user ID, movie ID, and rating score. The tool is then executed with the generated parameters. Tools that do not require parameters are executed without the LLM's involvement. The execution process is illustrated in Fig. 8. Users can consult the full execution flow and any intermediate results from individual tools to diagnose potential reasoning and/or execution errors. If a tool call fails, ECHO will attempt to re-execute the tool a given number of times using the error message from the previous attempt to guide the next tool call. If a tool in the plan ultimately fails to produce an answer, the execution flow is shown up until this error.

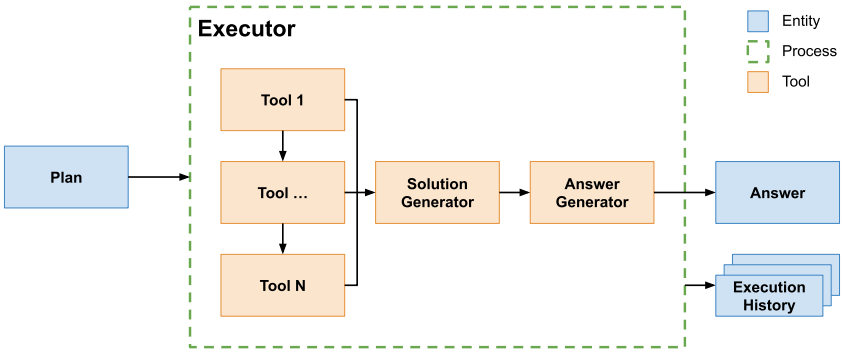


Fig. 8. A schematic overview of the workflow for sequentially executing the tools in a plan. If tools require parameters, an additional LLM call generates those from the context. The tool execution history is bundled in the Solution Generator, which uses CoT prompting to synthesize a solution based on the tool's execution flow. The final answer is extracted from the solution and displayed along the execution history to the user.

6 Applying ECHO in Case Studies

We present three case studies to demonstrate ECHO's feasibility and capabilities based on the scenarios outlined in Section 3. We use OpenAI's GPT-4o mini (latest stable version) for all LLM calls, as it strikes an appropriate balance between cost and quality. We implemented ECHO in Python 3.12.3 and used the LangChain framework to interact with LLMs [13]. ECHO's base LLM uses the default configuration as specified by LangChain, with a temperature of 0.7. The Mixture-of-Agents in the planner uses three GPT-4o mini instances with temperatures of 0, 0.7, and 1.4. ECHO's front end was developed using React[2], with Flask[3] handling the communication with the back end. The specific case studies were mainly implemented through scikit-learn[4] and Keras[5].

The case studies illustrate diverse explananda of increasing complexity, showcasing different aspects of ECHO's capabilities. We include both successful and imperfect conversations to show ECHO's extensiveness, but also the inherent uncertainty and randomness associated with using LLMs. The full conversation flows can be consulted in Appendix A. We only relied on ECHO's own reasoning capabilities without human-in-the-loop functionality for these case studies to assess its performance without user intervention.

6.1 Case Study 1: White Box Movie Recommender System

In this case study, we assess ECHO on a user-based KNN recommender system trained on the MovieLens dataset[6]. We did not add specific tools for explainability to the recommender, as a KNN algorithm is interpretable by nature and can be sufficiently questioned with the Table Communicator. To simulate a usable recommender system, we added a Predict tool that recommends movies for a specific user, a Rate tool for rating movies, and a Lookup tool to search for movies and their IDs.

When asked what user data is used for making recommendations, invoking **What** + **Inputs** explanation types, ECHO proposes the plan [Inspector, Table Communicator, Solution Generator, Answer Generator]. It investigates the explanandum's source code to find that the data contains user IDs, ratings, and movie IDs associated with these ratings. Note, however, that due to the

[2]https://react.dev/learn/typescript
[3]https://flask.palletsprojects.com/en/stable/
[4]https://scikit-learn.org/
[5]https://keras.io/
[6]https://grouplens.org/datasets/movielens/

inherent randomness in LLMs, executing that same plan might not always result in a helpful or correct answer, even though the plan is correct. This further highlights the importance of human-in-the-loop workflows, in which the LLM can try again until the answer is satisfactory.

When asked **Why** recommendations were made in a multi-round conversation about predictions, ECHO uses the Inspector to find out that the explanandum is a KNN collaborative filtering recommender. When asked what collaborative filtering is, mapping on **What** + **Application**, ECHO responds with a single Direct Answer. Next, when asked to change the rating of the movie 'Inception' to 5, ECHO showcases its ability to chain tools: since Rate does not accept strings as input, ECHO first uses Lookup Movie to search for a movie by string, returning its ID. This chaining successfully updates the rating, demonstrating how ECHO leverages the LLM's emerging ability to autonomously chain tools and resolve intermediate steps without explicit instruction. Finally, when asked for users with similar ratings compared to user 611, ECHO uses the Table Communicator to identify user 1, addressing an **Example** resulting from an implicit **What Else**-question.

6.2 Case Study 2: White Box Heart Disease Classifier

Here, we use a decision tree to classify whether someone is at risk of heart disease using the Heart Disease dataset [18, 33]. We omitted certain parameters for practicality. This explanandum adds the Predict tool, which predicts whether someone is at risk based on given **Inputs**, and the Xai Get Tree Structure, which enables ECHO to read the structure of the decision tree.

We assessed the predictor through one long conversation to highlight ECHO's ability to handle long flows, using the **History** explanation type. When asking for a prediction about a person, ECHO proposes the plan [Predict, Solution Generator, Answer Generator]. Since Predict is in the plan, its parameters are parsed into the *Input State*, as outlined in Section 5.4. ECHO predicts a risk of heart disease, albeit with a low certainty. To faithfully address a **Why**-question, ECHO uses the XAI Get Tree Structure tool, a generated tool that returns the explanandum's internal structure in JSON format, allowing ECHO to derive several factors that contributed to the prediction. When asked **What** + **Outputs** a prediction can have, ECHO checks the tabular dataset through the Table Communicator, successfully inferring that people are either at risk of heart disease or not.

Interestingly, ECHO managed to make **How To** explanations faithful to the model by combining Xai Get Tree Structure and Table Communicator. ECHO first inspects the tree structure to determine how its decision process works, and then uses the Table Communicator to cross-reference this with other cases in the dataset. ECHO answers with a broad overview of the factors that can improve the outcome of the prediction. However, despite having access to the decision tree, ECHO can still fail to interpret its structure. To illustrate this, when asked **What If** that person's blood pressure increased to 110, the prediction (as intended) flips to no longer being at risk with a 93% **Certainty**, which ECHO should have been able to see in the tree's structure returned by Xai Get Tree Structure. Lastly, when asked for an **Example** of people with similar health markers, ECHO finds similar users in the tabular training dataset as nearest neighbors using the Table Communicator.

6.3 Case Study 3: Black-Box Student Performance Classifier

This case study uses the student performance in mathematics dataset [17], containing student data and whether or not they passed the class. Here, we use a black-box model to show ECHO's flexibility in effectively explaining models of varying complexities. We reduced the dataset's parameters for simplicity and practicality by training a random forest on the dataset and using only the 10 most important features. This case's Predict tool predicts whether a student will pass or fail. Given that we are working with an inherently uninterpretable neural network, we also include tools Xai Lime and Xai Shap that invoke LIME [57] and SHAP [47], respectively, to address **Why** and **Why Not**.

We assess the predictor over two multi-message conversations, starting by asking **What** data is used for predictions. ECHO can correctly summarize the **Input** parameters. When asked about **What** type of **Model** or technology is used to generate predictions, ECHO uses the Inspector to find that our codebase uses Keras. When asked **How** it arrives at outcomes, ECHO briefly summarizes how such a neural network works based on the explanandum's codebase, consulted by the Inspector. ECHO can also use the Xai Lime tool to showcase how features affect a single prediction.

In the second flow, we enacted the scenario from Section 3, showcasing ECHO's versatility in handling different styles of queries, an inherent strength of LLMs. In the dataset, every student with intermediate scores greater than or equal to 11 has passed the course. However, we engineered an **Input** sample in which a student was still predicted to fail. When asked **Why**, ECHO finds the reasons by invoking the two XAI tools, reasoning about their outputs, and synthesizing this into a final answer. Similarly, ECHO can also use the XAI tools to determine under what conditions — **When** — this student would be predicted to pass. Lastly, ECHO suggests that the explanandum should not be trusted due to its low **Certainty** for the prediction.

7 Discussion, Limitations and Future Work

ECHO's architecture provides a flexible framework for explanatory and intelligible conversational user interfaces that allow interaction with other AI models. It was designed with modularity in mind, allowing enhancements and expansions by integrating additional predefined tools, modifying existing ones, or extending the methods of the explanandum. For example, the Table Communicator could be swapped out with tools that handle non-tabular data, such as image data, to support Visual Question Answering (VQA). Additionally, since some tools are dynamically generated from the explanandum's API, the capabilities of ECHO can be expanded by simply modifying the explanandum's implementation (e.g., adding utility methods to rate movies for recommender systems, or hyperparameter tuning for neural networks). We have demonstrated this through the case studies in Section 6, where each explanation type was addressed by relevant tools.

Although ECHO supports adding tools, it is still the developer's responsibility to find the right amount and types of tools to optimally support the explanation types for a given use case. While no method is required in the explanandum other than its Predict method, adding additional tools greatly impacts its usefulness and explainability. For example, a recommender would be cumbersome to use without its Rate and Lookup Movie methods, and the predictions of a decision tree and neural network would be difficult to explain without generated XAI tools, as evident in case studies 2 and 3. We can also modify current tools to address explanation types in multiple ways. For example, our approach to explaining **How To** is not ideal for fidelity; an optimal counterfactual explanation might want to change the smallest amount of parameters instead [4]. Still, a nearest-neighbors approach suffices in our context. Furthermore, much of ECHO's workflow relies on explanandum-agnostic prompts, which ensures broad applicability but may compromise performance. To counter this, ECHO supports adding few-shot examples tailored to a specific explanandum, similarly to the few-shot examples that teach it to map the explanation types to tools.

ECHO uses off-the-shelf general-purpose LLMs without specific fine-tuning, allowing it to explain a wide range of AI models. However, if needed, the LLMs can be replaced with fine-tuned alternatives. This fine-tuning could be tailored to a specific explanandum, by training LLMs on the generated tools used in that system. However, this would restrict ECHO to explaining that one explanandum. Alternatively, LLMs can be fine-tuned for tasks such as planning or tools like the Table Communicator, while remaining explanandum-agnostic. Nevertheless, ensuring an LLM produces suitable output remains challenging, whether through fine-tuning or extensive prompt engineering. Our current approach prioritizes structural transparency over the opacity of a single LLM. Since an LLM functions as a black box, we have engineered a transparent system composed of multiple

contained black boxes. This makes the links between components more transparent, but increases the complexity of the overall engineering effort. Conceptually, this aligns with neuro-symbolic AI, as discussed by Wang et al. [70], specifically the Symbolic[Neuro] variant.

While ECHO enables controlling the explanandum through tool usage, this can be cumbersome in some scenarios. For instance, a user may want to modify certain hyperparameters that require retraining the model, such as a **What If** explanation that requires changing the model's structure (e.g., adding or removing a layer in a neural network) or altering the dataset parameters (e.g., removing some attribute to observe its influence on the prediction) instead of requesting a contrastive **Input-Output** pair. ECHO was designed to explain a single AI model as is, trained on a dataset including all **Inputs**. Both examples would, therefore, be impractical at the moment, but could be addressed by (re)training and comparing multiple models, similar to the work on model multiplicity of Eerlings et al. [20], Wang et al. [72]. As with all use cases, developers can implement additional tools tailored to their requirements due to ECHO's modular approach.

ECHO's user interface currently focuses on the essentials, such as a chat interface and components to display tabular data. Moving beyond textual explanations, **Visualizations** and tailored diagrams could help users understand the explanandum's structure better. To this end, we could integrate visual XAI methods such as LIME and SHAP (which are now displayed in tables), or image-based explanations like saliency maps or Grad-CAM heatmaps [61]. Furthermore, LLM-driven GUI generation could enable ECHO to visualize the structure of an explanandum, such as the layer structure of a neural network, or generate interactive components to allow users to **Control** the explanandum. For example, users could be given the ability to add layers to the network or modify other parameters directly through the user interface. Lastly, while we have assessed ECHO's technical feasibility in Section 6, a future user study will assess the system's usability and its effectiveness in supporting human-in-the-loop interactions. We consider this foundation a prerequisite for future evaluations. Nevertheless, ECHO's generalizability to various use cases and its coverage of explanation types, as detailed in Section 4, support its value as a flexible, extensible framework for integrating conversational intelligibility into various interactive systems.

8 Conclusions

The growing popularity of AI comes with challenges related to transparency and controllability, both of which are essential for building trust and encouraging AI adoption. The field of e**X**plainable **AI** addresses these challenges by providing explanations that clarify the reasoning behind AI predictions. However, since most XAI techniques cover only a limited range of explanation types, we propose a conversational approach using tool-augmented language models. Our approach, ECHO, supports a broad spectrum of explanation types in an accessible, conversational manner. To develop ECHO, we reviewed existing literature to identify the types of explanation users seek in interactive systems. Based on these insights, we designed a set of tools that support these explanation types for AI models trained on tabular data. These tools are integrated into ECHO's execution flow, consisting of planning and execution stages. The planner uses a Mixture-of-Agents approach to improve the quality of the generated plans by merging multiple alternative plans into one aggregated, more well-rounded plan. The user can audit and modify this plan before deciding to execute it. In this way, ECHO enables interactive, human-in-the-loop AI explainability.

In conclusion, ECHO provides an extensible framework for interactive conversations with AI models, built on a conceptual foundation that defines key concepts, structures, and requirements for explaining AI models interactively. This includes a blueprint for the explanandum and a set of explanation types derived from the literature, along with tools to support them (**C1**). Our framework serves as a basis for explanatory conversational agents, enabling them to address all identified explanation types through a predefined set of tools mapped to these explanation types

(**C2.1**). To facilitate AI-model-specific conversations, ECHO dynamically generates tools from an explanandum's implementation and integrates them with the predefined tools (**C2.2**). Case studies based on various explanation types demonstrate ECHO's feasibility and advantages across different use cases. Our approach ensures that explanations remain interactive and aligned with user intent through a transparent human-in-the-loop process (**C2.3**). Any explanandum following ECHO's expected blueprint from **C1** can have its *System State* and *Inference Mechanism* interactively explained. ECHO's modular structure allows for expansion into other domains, such as Visual Question Answering, and applications like intelligible smart home automation, by incorporating additional tools into its predefined inventory. Moreover, prompt tuning and/or fine-tuning the LLMs can further improve AI model-specific performance.

Acknowledgments

This work was funded by the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" program, R-13509, and by the Special Research Fund (BOF) of Hasselt University, BOF23OWB31.

References

[1] Ashraf Abdul, Jo Vermeulen, Danding Wang, Brian Y Lim, and Mohan Kankanhalli. 2018. Trends and Trajectories for Explainable, Accountable and Intelligible Systems: An HCI Research Agenda. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–18.

[2] Blessing Olatunde Abisoye, Yanxia Sun, and Wang Zenghui. 2023. A survey of artificial intelligence methods for renewable energy forecasting: Methodologies and insights. *Renewable Energy Focus* (2023), 100529.

[3] Defense Advanced Research Projects Agency. 2017. XAI: Explainable Artificial Intelligence. https://www.darpa.mil/research/programs/explainable-artificial-intelligence

[4] Hayden Andersen, Andrew Lensen, Will Browne, and Yi Mei. 2023. Producing Diverse Rashomon Sets of Counterfactual Explanations with Niching Particle Swarm Optimization Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Lisbon, Portugal) *(GECCO '23)*. Association for Computing Machinery, New York, NY, USA, 393–401. https://doi.org/10.1145/3583131.3590444

[5] Anthropic. 2023. Introducing Claude. https://www.anthropic.com/news/introducing-claude

[6] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. 2020. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information fusion* 58 (2020), 82–115.

[7] Onur Asan, Alparslan Emrah Bayrak, Avishek Choudhury, et al. 2020. Artificial Intelligence and Human Trust in Healthcare: Focus on Clinicians. *Journal of medical Internet research* 22, 6 (2020), e15154.

[8] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. 2024. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 17682–17690.

[9] Cindel Bonneux, Gustavo Rovelo, Paul Dendale, and Karin Coninx. 2019. A Comprehensive Approach to Decision Aids Supporting Shared Decision Making in Cardiac Rehabilitation. In *Proceedings of the 13th EAI International Conference on Pervasive Computing Technologies for Healthcare* (Trento, Italy) *(PervasiveHealth'19)*. Association for Computing Machinery, New York, NY, USA, 389–398. https://doi.org/10.1145/3329189.3329241

[10] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[11] Federico Cabitza, Andrea Campagner, Gianclaudio Malgieri, Chiara Natali, David Schneeberger, Karl Stoeger, and Andreas Holzinger. 2023. Quod erat demonstrandum? - Towards a typology of the concept of explanation for the design of explainable AI. *Expert Systems with Applications* 213 (2023), 118888. https://doi.org/10.1016/j.eswa.2022.118888

[12] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. 2015. Intelligible Models for HealthCare: Predicting Pneumonia Risk and Hospital 30-day Readmission. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 1721–1730.

[13] Harrison Chase. 2022. *LangChain*. https://github.com/langchain-ai/langchain
[14] Chaofan Chen, Kangcheng Lin, Cynthia Rudin, Yaron Shaposhnik, Sijia Wang, and Tong Wang. 2018. An Interpretable Model with Globally Consistent Explanations for Credit Risk. *arXiv preprint arXiv:1811.12615* (2018).
[15] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *arXiv preprint arXiv:2211.12588* (2022).
[16] Sven Coppers, Kris Luyten, Davy Vanacken, David Navarre, Philippe Palanque, and Christine Gris. 2019. Fortunettes: Feedforward about the Future State of GUI Widgets. *Proc. ACM Hum.-Comput. Interact.* 3, EICS, Article 20 (June 2019), 20 pages. https://doi.org/10.1145/3331162
[17] Paulo Cortez. 2008. Student Performance. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5TG7T.
[18] Robert C. Detrano, András Jánosi, Walter Steinbrunn, Matthias Emil Pfisterer, Johann-Jakob Schmid, Sarbjit Sandhu, Kern Guppy, Stella Lee, and Victor Froelicher. 1989. International application of a new probability algorithm for the diagnosis of coronary artery disease. *The American journal of cardiology* 64 5 (1989), 304–10. https://api.semanticscholar.org/CorpusID:23545303
[19] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A Survey on In-context Learning. *arXiv preprint arXiv:2301.00234* (2022).
[20] Gilles Eerlings, Sebe Vanbrabant, Jori Liesenborgs, Gustavo Rovelo Ruiz, Davy Vanacken, and Kris Luyten. 2024. AI-Spectra: A Visual Dashboard for Model Multiplicity to Enhance Informed and Transparent Decision-Making. *arXiv preprint arXiv:2411.10490* (2024).
[21] William Ferguson, Dhruv Batra, Raymond Mooney, Devi Parikh, Antonio Torralba, David Bau, David Diller, Josh Fasching, Jaden Fiotto-Kaufman, Yash Goyal, et al. 2021. Reframing explanation as an interactive medium: The EQUAS (Explainable QUestion Answering System) project. *Applied AI Letters* 2, 4 (2021), e60.
[22] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided Language Models. In *International Conference on Machine Learning*. PMLR, 10764–10799.
[23] Yunfan Gao, Tao Sheng, Youlin Xiang, Yun Xiong, Haofen Wang, and Jiawei Zhang. 2023. Chat-REC: Towards Interactive and Explainable LLMs-Augmented Recommender System. *arXiv preprint arXiv:2303.14524* (2023).
[24] Bryce Goodman and Seth Flaxman. 2017. European Union Regulations on Algorithmic Decision Making and a "Right to Explanation". *AI magazine* 38, 3 (2017), 50–57.
[25] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2018. A Survey of Methods for Explaining Black Box Models. *ACM computing surveys (CSUR)* 51, 5 (2018), 1–42.
[26] David Gunning and David Aha. 2019. DARPA's Explainable Artificial Intelligence (XAI) Program. *AI Magazine* 40, 2 (06 2019), 44–58. https://doi.org/10.1609/aimag.v40i2.2850
[27] Yaru Hao, Haoyu Song, Li Dong, Shaohan Huang, Zewen Chi, Wenhui Wang, Shuming Ma, and Furu Wei. 2022. Language Models are General-Purpose Interfaces. *arXiv preprint arXiv:2206.06336* (2022).
[28] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. 2004. Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems (TOIS)* 22, 1 (2004), 5–53.
[29] Fred Hohman, Andrew Head, Rich Caruana, Robert DeLine, and Steven M Drucker. 2019. Gamut: A Design Probe to Understand How Data Scientists Understand Machine Learning Models. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–13.
[30] Andreas Holzinger, Anna Saranti, Christoph Molnar, Przemyslaw Biecek, and Wojciech Samek. 2022. Explainable AI Methods - A Brief Overview. In *International workshop on extending explainable AI beyond deep models and classifiers*. Springer, 13–38.
[31] Alyssa Hwang, Natasha Oza, Chris Callison-Burch, and Andrew Head. 2023. Rewriting the Script: Adapting Text Instructions for Voice Interaction. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference (DIS '23)*. Association for Computing Machinery, New York, NY, USA, 2233–2248. https://doi.org/10.1145/3563657.3596059
[32] Gautier Izacard and Edouard Grave. 2020. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. *arXiv preprint arXiv:2007.01282* (2020).
[33] Andras Janosi, William Steinbrunn, Matthias Pfisterer, and Robert Detrano. 1989. Heart Disease. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C52P4X.
[34] Peter Kieseberg, Edgar Weippl, A Min Tjoa, Federico Cabitza, Andrea Campagner, and Andreas Holzinger. 2023. Controllable AI - An Alternative to Trustworthiness in Complex AI Systems?. In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*. Springer, 1–12.
[35] Mojtaba Komeili, Kurt Shuster, and Jason Weston. 2021. Internet-Augmented Dialogue Generation. *arXiv preprint arXiv:2107.07566* (2021).
[36] Benjamin Letham, Cynthia Rudin, Tyler H. McCormick, and David Madigan. 2015. Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics* 9, 3 (2015), 1350 – 1371. https://doi.org/10.1214/15-AOAS848

[37] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[38] Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. 2024. TaskMatrix.AI: Completing Tasks by Connecting Foundation Models with Millions of APIs. *Intelligent Computing* 3 (2024), 0063.

[39] Brian Y Lim and Anind K Dey. 2009. Assessing Demand for Intelligibility in Context-Aware Applications. In *Proceedings of the 11th international conference on Ubiquitous computing*. 195–204.

[40] Brian Y Lim and Anind K Dey. 2010. Toolkit to Support Intelligibility in Context-Aware Applications. In *Proceedings of the 12th ACM international conference on Ubiquitous computing*. 13–22.

[41] Brian Y Lim and Anind K Dey. 2013. Evaluating Intelligibility Usage and Usefulness in a Context-Aware Application. In *Human-Computer Interaction. Towards Intelligent and Implicit Interaction: 15th International Conference, HCI International 2013, Las Vegas, NV, USA, July 21-26, 2013, Proceedings, Part V 15*. Springer, 92–101.

[42] Brian Y Lim, Anind K Dey, and Daniel Avrahami. 2009. Why and Why Not Explanations Improve the Intelligibility of Context-Aware Intelligent Systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2119–2128.

[43] Brian Y Lim, Qian Yang, Ashraf M Abdul, and Danding Wang. 2019. Why these Explanations? Selecting Intelligibility Types for Explanation Goals. In *IUI workshops*.

[44] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-Play Compositional Reasoning with Large Language Models. In *The 37th Conference on Neural Information Processing Systems (NeurIPS)*.

[45] Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. 2023. Dynamic Prompt Learning via Policy Gradient for Semi-structured Mathematical Reasoning. In *International Conference on Learning Representations (ICLR)*.

[46] Scott M Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. 2020. From local explanations to global understanding with explainable AI for trees. *Nature machine intelligence* 2, 1 (2020), 56–67.

[47] Scott M Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf

[48] Haoyan Luo and Lucia Specia. 2024. From Understanding to Utilization: A Survey on Explainability for Large Language Models. *arXiv preprint arXiv:2401.12874* (2024).

[49] Amama Mahmood, Junxiang Wang, Bingsheng Yao, Dakuo Wang, and Chien-Ming Huang. 2025. User Interaction Patterns and Breakdowns in Conversing with LLM-Powered Voice Assistants. *International Journal of Human-Computer Studies* 195 (2025), 103406. https://doi.org/10.1016/j.ijhcs.2024.103406

[50] David Martens, James Hinns, Camille Dams, Mark Vergouwen, and Theodoros Evgeniou. 2025. Tell me a story! Narrative-driven XAI with Large Language Models. *Decision Support Systems* (2025), 114402.

[51] Philip Mavrepis, Georgios Makridis, Georgios Fatouros, Vasileios Koukos, Maria Margarita Separdani, and Dimosthenis Kyriazis. 2024. XAI for All: Can Large Language Models Simplify Explainable AI? *arXiv preprint arXiv:2401.13110* (2024).

[52] Mistral AI team. 2024. Codestral: Hello, World! https://mistral.ai/news/codestral/

[53] Sina Mohseni, Niloofar Zarei, and Eric D Ragan. 2021. A Multidisciplinary Survey and Framework for Design and Evaluation of Explainable AI Systems. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 11, 3-4 (2021), 1–45.

[54] Brad A. Myers, David A. Weitzman, Amy J. Ko, and Duen H. Chau. 2006. Answering Why and Why Not Questions in User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montréal, Québec, Canada) *(CHI '06)*. Association for Computing Machinery, New York, NY, USA, 397–406. https://doi.org/10.1145/1124772.1124832

[55] OpenAI. 2022. Introducing ChatGPT. https://openai.com/index/chatgpt/

[56] Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. TALM: Tool Augmented Language Models. *arXiv preprint arXiv:2205.12255* (2022).

[57] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1135–1144.

[58] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Anchors: High-Precision Model-Agnostic Explanations. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[59] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence* 1, 5 (2019), 206–215.

[60] Wojciech Samek and Klaus-Robert Müller. 2019. Towards Explainable Artificial Intelligence. *Explainable AI: interpreting, explaining and visualizing deep learning* (2019), 5–22.

[61] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2020. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *International journal of computer vision* 128 (2020), 336–359.

[62] Thilo Spinner, Udo Schlegel, Hanna Schäfer, and Mennatallah El-Assady. 2019. explAIner: A Visual Analytics Framework for Interactive and Explainable Machine Learning. *IEEE transactions on visualization and computer graphics* 26, 1 (2019), 1064–1074.

[63] Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. ViperGPT: Visual Inference via Python Execution for Reasoning. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. 11854–11864. https://doi.org/10.1109/ICCV51070.2023.01092

[64] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).

[65] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[66] Jo Vermeulen, Geert Vanderhulst, Kris Luyten, and Karin Coninx. 2010. PervasiveCrystal: Asking and Answering Why and Why Not Questions about Pervasive Computing Applications. In *2010 Sixth International Conference on Intelligent Environments*. 271–276. https://doi.org/10.1109/IE.2010.56

[67] Warren J Von Eschenbach. 2021. Transparency and the Black Box Problem: Why We Do Not Trust AI. *Philosophy & Technology* 34, 4 (2021), 1607–1622.

[68] Danding Wang, Qian Yang, Ashraf Abdul, and Brian Y. Lim. 2019. Designing Theory-Driven User-Centric Explainable AI. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3290605.3300831

[69] Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. 2024. Mixture-of-Agents Enhances Large Language Model Capabilities. *arXiv preprint arXiv:2406.04692* (2024).

[70] Wenguan Wang, Yi Yang, and Fei Wu. 2024. Towards Data-and Knowledge-Driven Artificial Intelligence: A Survey on Neuro-Symbolic Computing. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

[71] Yancheng Wang, Ziyan Jiang, Zheng Chen, Fan Yang, Yingxue Zhou, Eunah Cho, Xing Fan, Xiaojiang Huang, Yanbin Lu, and Yingzhen Yang. 2023. RecMind: Large Language Model Powered Agent For Recommendation. *arXiv preprint arXiv:2308.14296* (2023).

[72] Zijie J Wang, Chudi Zhong, Rui Xin, Takuya Takagi, Zhi Chen, Duen Horng Chau, Cynthia Rudin, and Margo Seltzer. 2022. TimberTrek: Exploring and Curating Sparse Decision Trees with Interactive Visualization. In *2022 IEEE Visualization and Visual Analytics (VIS)*. IEEE, 60–64.

[73] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022. Emergent Abilities of Large Language Models. *arXiv preprint arXiv:2206.07682* (2022).

[74] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 24824–24837. https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf

[75] James Wexler, Mahima Pushkarna, Tolga Bolukbasi, Martin Wattenberg, Fernanda Viégas, and Jimbo Wilson. 2019. The What-If Tool: Interactive Probing of Machine Learning Models. *IEEE transactions on visualization and computer graphics* 26, 1 (2019), 56–65.

[76] Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. 2023. Visual ChatGPT: Talking, Drawing and Editing with Visual Foundation Models. *arXiv preprint arXiv:2303.04671* (2023).

[77] Feiyu Xu, Hans Uszkoreit, Yangzhou Du, Wei Fan, Dongyan Zhao, and Jun Zhu. 2019. Explainable AI: A Brief Survey on History, Research Areas, Approaches and Challenges. In *Natural language processing and Chinese computing: 8th cCF international conference, NLPCC 2019, dunhuang, China, October 9–14, 2019, proceedings, part II 8*. Springer, 563–574. https://doi.org/10.1007/978-3-030-32236-6_51

[78] Xuhai Xu, Anna Yu, Tanya R Jonker, Kashyap Todi, Feiyu Lu, Xun Qian, João Marcelo Evangelista Belo, Tianyi Wang, Michelle Li, Aran Mun, et al. 2023. XAIR: A Framework of Explainable AI in Augmented Reality. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–30.

[79] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. *Advances in Neural Information Processing Systems* 36 (2024).

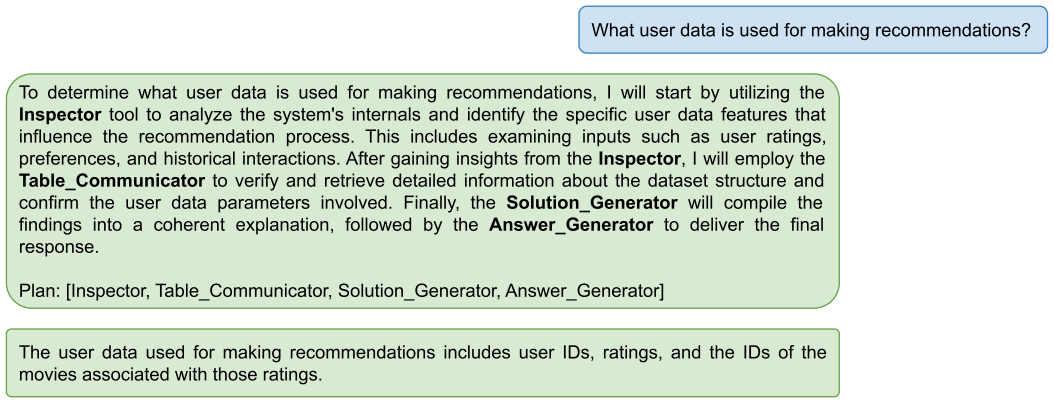
[80] Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. 2022. Generate rather than Retrieve: Large Language Models are Strong Context Generators. *arXiv preprint arXiv:2209.10063* (2022).

[81] Shu Zhong, Elia Gatti, James Hardwick, Miriam Ribul, Youngjun Cho, and Marianna Obrist. 2025. LLM-mediated domain-specific voice agents: the case of TextileBot. *Behaviour & Information Technology* 0, 0 (2025), 1–33. https://doi.org/10.1080/0144929X.2025.2456667 arXiv:https://doi.org/10.1080/0144929X.2025.2456667

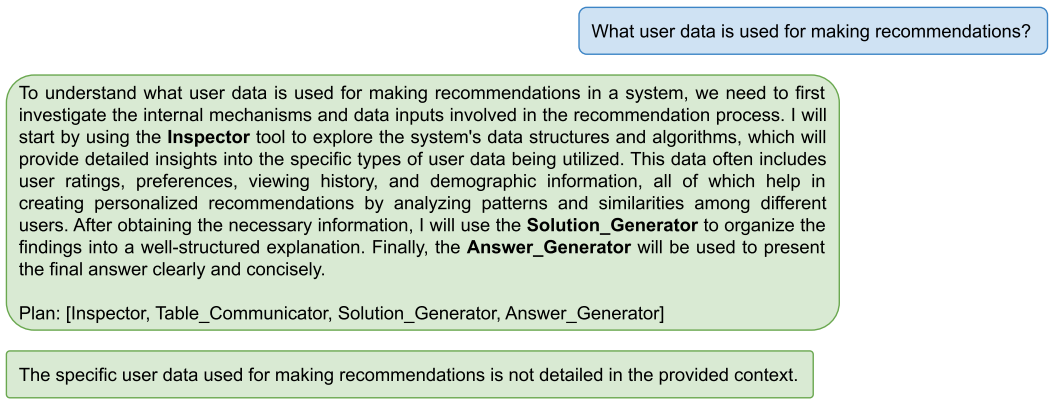
A Case Studies

Each conversation shows a user message in blue on the right-hand side, followed by two prompts from ECHO on the left: one from the planner and one from the executor. We condensed the content of the user interface to only show the user query, the plan and its motivation, and the final answer. The actual interface, shown in Fig. 1, has various interactive components that are inconvenient to display in screenshots. We will deviate from this approach and show more information in the message when other aspects are crucial to illustrate a certain explanation type.

A.1 Case Study 1: White Box Movie Recommender System



(a) ✓ ECHO successfully answered the user's question by describing what the recommender uses to create its recommendations.



(b) ✗ Although the planner gave a plan that could lead to a correct answer, the tools not always produce useful output, preventing ECHO from providing a correct answer. This highlights the importance of human-in-the-loop conversations, to ask the question again or in a different form until the answer is satisfactory.

Fig. 9. A correct and incorrect answer to the question "What user data is used for making recommendations?". In both cases, the planner creates a correct plan. However, due to a mistake in the execution, an incorrect answer can be given nonetheless. This example addresses the **What** + **Inputs** explanation type.

What movies do you recommend to user 611? How confident are you?

To provide movie recommendations for user 611 and assess the confidence level of those recommendations, the **Predict** tool will be used to generate personalized movie suggestions based on the user's preferences and ratings. The **Predict** tool will also provide confidence metrics for each recommendation, indicating the reliability of the recommendations. Following this, the **Solution_Generator** tool will compile the results into a coherent answer, and the **Answer_Generator** will finalize the output for presentation.

Plan: [Predict, Solution_Generator, Answer_Generator]

I recommend the following movies to user 611: "Aliens (1986)", "Die Hard (1988)", "Gladiator (2000)", "The Matrix (1999)", "The Godfather (1972)", "Alien (1979)", "Saving Private Ryan (1998)", "The Godfather: Part II (1974)", "Terminator 2: Judgment Day (1991)", and "Braveheart (1995)", with a confidence level of 30.18%.

Why did you make these recommendations? What is the reasoning behind these recommendations?

The recommendations for user 611 are generated using a collaborative filtering approach, which evaluates user preferences and ratings, identifying patterns to suggest movies that similar users have enjoyed. This method assumes that if other users with similar tastes liked certain movies, user 611 might too. The process also considers the overall popularity and ratings of movies within the user community. To provide a comprehensive explanation of these recommendations, I will first inspect the model's internal logic using the **Inspector** tool. This inspection will help understand the underlying mechanisms and data used to generate the recommendations. Following this, I will employ the **Solution_Generator** to craft a detailed explanation based on the findings. Finally, I will use the **Answer_Generator** to summarize the explanation into a concise answer for clarity.

Plan: [Inspector, Solution_Generator, Answer_Generator]

The recommendations were made using collaborative filtering and KNN to identify movies that align with user preferences based on similar users' ratings.

What is collaborative filtering?

Collaborative filtering is a recommendation technique that suggests items to users based on the preferences of similar users.

Plan: [Direct_Answer]

Fig. 10. A multi-round conversation about the predictions of the recommender system. This demonstrates tool generation and the **Why** and **What** explanation types. Asking for recommendations invokes the Predict tool. When the user asks how these recommendations were made, ECHO can correctly identify the technology used in the recommender and explain how it functions by giving a **Description** of collaborative filtering.

Change the rating of user 611 for Inception to 5.

To update the rating of user 611 for the movie "Inception" to 5, the process involves identifying the correct movie ID and then applying the rating change. The first step is to use the **Lookup_Movie** tool to find the unique identifier for "Inception." This ensures that the rating is being updated for the correct movie. Once the movie ID is acquired, the **Rate** tool is used to modify the user's rating to 5. The **Solution_Generator** can then be employed to consolidate the process and provide a detailed overview of the change. Lastly, the **Answer_Generator** will succinctly present the final outcome.

Plan: [Lookup_Movie, Rate, Solution_Generator, Answer_Generator]

Lookup_Movie
Inputs: "Inception"
Outputs: {"movieId": 79132, "title": "Inception (2010)"}

Rate
Input: {"user_id": 611, "movie_id": 79132, "rating": 5.0}
Output: null

Solution_Generator
To change the rating of user 611 for the movie "Inception" to 5, the system has successfully processed the request. The movie "Inception" was identified with the movie ID 79132, and the rating for user 611 has been updated to 5.

Answer: The rating of user 611 for "Inception" has been successfully changed to 5.

Answer_Generator
The rating of user 611 for "Inception" has been successfully changed to 5.

Fig. 11. An example of effective tool chaining, showing intermediate steps. The generated tools Lookup Movie and Rate can be used in sequence to rate movies by name, whereas Rate itself can only rate by ID. The planner identified this possibility, which the executor properly handled.

Share examples of similar users to user 611 and their preferences.

To share examples of similar users to user 611 and their preferences, I will use the **Table_Communicator** tool to identify users with similar ratings and preferences based on the dataset. This step will help me gather relevant data about these similar users. After identifying them, I will analyze their preferences and summarize the findings to provide a comprehensive response.

Plan: [Table_Communicator, Solution_Generator, Answer_Generator]

User 611 has similar users like User ID 1, who enjoys a variety of genres including action, adventure, drama, and comedy, with many movies rated highly.

Fig. 12. An **Example** resulting from an implicit **What Else** question. The Table Communicator is invoked to address this explanation type and returns a user with a similar profile to user 611.

A.2 Case Study 2: White Box Heart Disease Classifier

Is Bob, a 70 year old man with chest pain type 0, a resting blood pressure of 100, cholesterol of 246, no fasting blood sugar, 150 thalach, no exercise induced angina, 0.8 oldpeak and 0 slope, at risk of having heart disease?

To evaluate if Bob, a 70-year-old man with specified health metrics, has a risk of heart disease, we will use the **Predict** tool. This tool will analyze various health parameters such as age, sex, chest pain type, resting blood pressure, cholesterol levels, fasting blood sugar, thalach, exercise-induced angina, oldpeak, and slope to predict the likelihood of heart disease. Subsequently, the **Solution_Generator** will compile these results into a comprehensive explanation, and the **Answer_Generator** will present the final answer.

Plan: [Predict, Solution_Generator, Answer_Generator]

Yes, Bob is at risk of having heart disease with a certainty of 62%.

Why is Bob is at risk of heart disease?

To explain why Bob is at risk of heart disease, I will first employ the **Xai_Get_Tree_Structure** tool to analyze the decision-making pathway of the model and understand how it evaluates Bob's risk factors. This will provide a clear insight into the specific features contributing to the prediction. I will then use the **Table_Communicator** to gather detailed data about Bob's health and demographic factors, to cross-reference and identify key risk elements. Finally, I will generate a solution based on this analysis and articulate the findings clearly.

Plan: [Xai_Get_Tree_Structure, Table_Communicator, Solution_Generator, Answer_Generator]

Bob is at risk of heart disease due to several health indicators such as age, sex, high resting blood pressure, and possibly elevated cholesterol levels, which are all linked to cardiovascular health issues.

What possible outcomes can a prediction have?

The possible outcomes of a heart disease prediction model often include classifications such as 'at risk' or 'not at risk'. These outcomes are determined by analyzing various health metrics and thresholds set within the predictive model. To fully understand the range of possible outcomes, it is important to explore the dataset and identify all unique output values. This can be done using the **Table_Communicator** tool, which will provide a comprehensive overview of all potential prediction results the model can generate.

Plan: [Table_Communicator, Solution_Generator, Answer_Generator]

The possible outcomes of a prediction regarding heart disease are "heart disease" and "no heart disease", with respective counts of 526 and 499 in the dataset.

Fig. 13. (Part 1/2) ECHO uses the Predict tool to predict that Bob is at risk regarding heart disease. When asked **Why** Bob is predicted to be at risk, ECHO uses the XAI Get Tree Structure and Table Communicator tools to find that this is mainly due to his age and elevated cholesterol level, with a certainty of 62%. When asked **What** + **Outputs** the model can have, ECHO successfully uses the Table Communicator to find this.

How to improve Bob's classification from having heart disease to not having heart disease?

To improve Bob's classification from having heart disease to not having heart disease, we need to analyze the factors contributing to his current classification. By examining the decision tree structure and dataset parameters, we can identify which features are most influential in the prediction. The process involves using tools to understand decision rules, simulate changes in parameters, and synthesize a comprehensive action plan for Bob.

Plan: [Xai_Get_Tree_Structure, Table_Communicator, Solution_Generator, Answer_Generator]

Bob can improve his classification from having heart disease to not having heart disease by lowering cholesterol levels, improving resting blood pressure, increasing physical activity, adjusting oldpeak values, monitoring chest pain symptoms, adopting a heart-healthy diet, and having regular health check-ups.

What would happen if his blood pressure increases to 110?

To assess the impact of an increase in Bob's blood pressure to 110 on his risk of heart disease, I will utilize the **Predict** tool. This tool will enable me to simulate the scenario with the modified blood pressure parameter and evaluate how this change influences the prediction concerning his heart disease risk. After obtaining the prediction, I will use the **Solution_Generator** and **Answer_Generator** tools to formulate a comprehensive response based on the results.

Plan: [Predict, Solution_Generator, Answer_Generator]

If his blood pressure increases to 110, he would not be at risk of heart disease, with a certainty of 93%.

What are some examples of other people who are in a similar situation as Bob?

To find examples of other individuals with similar health characteristics as Bob, I will use the **Table_Communicator** to search the dataset for data points that match Bob's specific health parameters. This will help in identifying cases with similar characteristics and different health outcomes, which will be presented in a clear format.

Plan: [Table_Communicator, Solution_Generator, Answer_Generator]

Some examples of other people in a similar situation as Bob include individuals aged 58 to 71, with varied cholesterol levels and resting blood pressures, who also experience chest pain types ranging from 0 to 2.

Fig. 14. (Part 2/2) When asked **How To** improve Bob's prediction, ECHO decides to look into the decision tree's structure and dataset, finding that he should lower his cholesterol levels, improve resting blood pressure, and increase physical activity. It is important to note that Bob's case was specifically engineered so that increasing his blood pressure slightly would flip his prediction, as indicated by the next **What If** question. ECHO failed to find this and instead gives recommendations based on knowledge, rather than the decision tree's structure that does not mention cholesterol at all. When asked for **Example**s of people similar to Bob, ECHO finds the nearest neighbors and shows them in a table.

A.3 Case Study 3: Black-Box Student Performance Classifier

What data does the model consider when making predictions about a student's performance?

The model predicts student performance by considering a variety of factors such as grades (G1 and G2), absences, failures, age, parental education levels (Fedu and Medu), health status, social activities (Goout, Walc), free time, and family relationships (Famrel). These inputs are crucial for assessing the likelihood of a student's success. To thoroughly understand these parameters and their significance, I will begin by inspecting the model's structure using the **Inspector** tool, followed by retrieving detailed information about these features with the **Table_Communicator**.

Plan: [Inspector, Table_Communicator, Solution_Generator, Answer_Generator]

The model considers various data points including grades from different periods, absences, failures, age, social activities, and parental education levels when making predictions about a student's performance.

What type of model or technology is being used to generate these predictions?

To determine the type of model or technology being used for generating predictions, the **Inspector** tool will be utilized. This tool helps in examining the internal workings of the system, revealing details about the model's architecture and the technology applied. After collecting this information, a summary will be provided to clearly explain the model type and technology in use.

Plan: [Inspector, Solution_Generator, Answer_Generator]

The model used is a classification model based on Keras, a high-level neural networks API, indicating a machine learning or deep learning approach.

How did it arrive at its outcomes?

To understand how the model arrived at its outcomes, the process involves examining the internal mechanisms and utilizing explainability tools. First, the **Inspector** tool is used to gain insights into the model's architecture and algorithms, which helps clarify the decision-making process. Then, XAI tools such as **Xai_Lime** or **Xai_Shap** are applied to analyze the contributions of specific input features to the final predictions, enhancing the understanding of the model's outcomes. Although Response 2 suggests using the **Table_Communicator** for visualizing data tables, it is not directly needed unless data tables are specifically involved in the explanation process. The final steps involve synthesizing the findings into a comprehensive solution and generating the final answer.

Plan: [Inspector, Xai_Lime, Solution_Generator, Answer_Generator]

The model arrived at its outcomes by using a neural network that processes student data, applies learned patterns during prediction, and utilizes techniques like LIME to explain the importance of various features in the decision-making process.

Fig. 15. This conversation tries to determine **What** + **Inputs** are being used in the explanandum, which ECHO finds using the Inspector and Table Communicator tools. Next, we ask **What** + **Model** is being used. The Inspector correctly points out the usage of a Keras neural network. When asked **How** the model arrived at its outcomes, ECHO correctly uses the Inspector, but also incorrectly assumes that LIME contributes to the result, while it merely explains it post hoc.

Carol wonders why Dave, a student with seemingly okay performance is predicted to fail mathematics. Dave scores an 11 on both the C1 and C2 first and second period grade and has never failed a class before. No other student with these attributes has ever failed mathematics before. Carol knows that Dave is 21 years old and in very good health and never drinks. He goes out moderately often and has a lot of free time and he has only been absent 10 times this year. He has a great relationship with his family, and his parent are both highly educated. Why does the model predict that Dave will not pass mathematics?

To understand why the model predicts that Dave will not pass mathematics despite his seemingly good performance, it is essential to analyze the model's decision-making process. Using XAI tools like LIME and SHAP will provide insights into the contributions of various features related to Dave's attributes, such as grades, attendance, and socioeconomic factors. **Xai_Lime** will offer a local explanation, highlighting the features that influenced the model's decision most significantly. **Xai_Shap** will give additional insights into the contributions of these features. This comprehensive approach will help us understand the underlying factors influencing the model's prediction. Finally, the **Solution_Generator** will synthesize the information obtained, and the **Answer_Generator** will present the final explanation clearly.

Plan: [Xai_Lime, Xai_Shap, Solution_Generator, Answer_Generator]

The model predicts that Dave may fail mathematics due to factors such as his age, attendance, and the nuanced influence of his grades, despite his otherwise solid performance.

Under what conditions or circumstances would the model predict that a student like Dave would pass mathematics?

To determine under what conditions the model would predict that a student like Dave would pass mathematics, I will simulate different scenarios using the **Predict** tool. By adjusting input parameters such as grades, attendance, age, and family background, I can identify which changes lead to a passing prediction. Additionally, using XAI tools like **Xai_Lime** and **Xai_Shap** will help understand the influence of each feature on the prediction, providing deeper insights into the model's decision-making process.

Plan: [Predict, Xai_Lime, Xai_Shap, Solution_Generator, Answer_Generator]

The model would predict that a student like Dave would pass mathematics if he achieves higher grades, reduces his absences to below 8, is younger than 21, and maintains strong family support and good health.

Should Carol trust the model?

To determine whether Carol should trust the model's prediction regarding Dave's performance in mathematics, we need to evaluate the model's reliability and understand the reasoning behind its predictions. The model's unexpected prediction that Dave might fail despite good performance metrics suggests potential issues with how the model interprets data. Therefore, we should use explainability tools to analyze the factors influencing the model's outputs and evaluate its performance. First, we will use **Xai_Lime** to understand the key factors affecting the prediction. Next, we will apply **Xai_Shap** to obtain detailed insights into feature contributions, enhancing our understanding of the model's reliability and certainty in its prediction. Finally, **Solution_Generator** will compile the findings to provide Carol with a clear recommendation on whether to trust the model's predictions.

Plan: [Xai_Lime, Xai_Shap, Solution_Generator, Answer_Generator]

Carol should be cautious in trusting the model entirely, as its prediction of failure has only a 53% certainty and is influenced by various mixed factors, despite Dave's solid academic history.

Fig. 16. When asked about a student's unexpected prediction, ECHO decides to explain **Why** using Xai Lime and Xai Shap, and correctly finds that his age and absences contributed the most to this prediction. When asked **When** he can be predicted to pass mathematics, these parameters are also correctly identified, although recommending that he change his age is not useful. Since the certainty is very low, the LLM can also identify that the model should not be trusted in this situation.