

UNIVERSITY OF TARTU  
Faculty of Science and Technology  
Institute of Computer Science  
Computer Science Curriculum

Rodion Krjutškov

# Explainability Assistant: Natural Language Interface for Trustworthy AI

Master's Thesis (30 ECTS)

Supervisors: Eduard Barbu, PhD  
Raul Vicente Zafra, PhD

Tartu 2025

# **Explainability Assistant: Natural Language Interface for Trustworthy AI**

## **Abstract:**

The notion of explainability within machine learning systems can be traced back several decades in Computer Science history. However, in the last decade, the increasing adoption of AI in various sectors has intensified interest in the interpretability of ML models. Furthermore, the concept of trustworthy AI provides a comprehensive framework focusing on the development of ethical and accountable applications.

This study, as part of a broader initiative Trust-AI, investigates how natural language dialogues can be utilized as a mechanism for explainable and trustworthy AI. By reviewing previous research and incorporating the latest advancements in large language models, we have devised an innovative solution with notably improved language parsing accuracy, which was evaluated in a practical application within the Trust-AI framework.

## **Keywords:**

Artificial Intelligence (AI), Explainable Artificial Intelligence (XAI), Trustworthy AI, Natural Language Processing (NLP), Large Language Models (LLM), Conversational Interface

## **CERCS:**

P176 - Artificial intelligence; P170 - Computer science, numerical analysis, systems, control.

## **Explainability Assistant: loomuliku keele liides usaldusväärse tehisintellekti jaoks**

### **Lühikokkuvõte:**

Masinõppesüsteemide seletatavuse mõiste ulatub arvutiteaduse ajaloos mitme aastakümne taha. Viimase kümnendi jooksul on tehisintellekti üha laialdasem kasutuselevõtt erinevates sektorites aga suurendanud huvi masinõppemudelite tõlgendatavuse vastu. Lisaks pakub usaldusväärse tehisintellekti kontseptsioon tervikliku raamistiku, mis keskendub eetiliste ja vastutustundlike rakenduste arendamisele.

See uuring, mis on osa laiemast algatusest Trust-AI, vaatleb, kuidas loomuliku keele dialooge saab kasutada seletatava ja usaldusväärse AI mehhanismina. Varasemate uuringute ülevaatamisel ja suurte keelemudelite uusimate edusammude kaasamisel oleme välja töötanud uuendusliku lahenduse, millel on märkimisväärselt parem keele parsimise täpsus ja mida hinnati praktilises rakenduses Trust-AI raamistikus.

### **Võtmesõnad:**

Tehisintellekt (AI), seletatav tehisintellekt (XAI), usaldusväärne tehisintellekt, loomuliku keele töötlemine (NLP), suured keelemudelid (LLM), vestlusliides

### **CERCS:**

P176 – Tehisintellekt; P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria).

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
<b>2</b>	<b>Background</b>	<b>7</b>
2.1	Trustworthy and explainable AI . . . . .	7
2.1.1	Explainability in intelligent systems . . . . .	7
2.1.2	Taxonomy of explainable AI techniques . . . . .	8
2.1.3	Post-hoc explanation methods . . . . .	9
2.1.4	Genetic Programming . . . . .	11
2.1.5	Trustworthy AI . . . . .	12
2.2	Advances in language modeling and processing . . . . .	13
2.2.1	Natural language processing . . . . .	14
2.2.2	Transformer architecture . . . . .	15
2.2.3	Generative Pre-trained Transformer . . . . .	16
2.2.4	Text-To-Text Transfer Transformer . . . . .	17
2.2.5	Large Language Model Meta AI . . . . .	17
2.2.6	Google Gemini . . . . .	18
<b>3</b>	<b>TalkToModel</b>	<b>19</b>
3.1	System analysis . . . . .	19
3.1.1	Overview . . . . .	20
3.1.2	System architecture . . . . .	21
3.1.3	Operation execution . . . . .	23
3.1.4	Application of LLMs . . . . .	26
3.2	System extensions . . . . .	28
3.2.1	Regression problems . . . . .	28
3.2.2	GP support . . . . .	29
3.2.3	Fine-tuning T5 models . . . . .	30
3.2.4	Fine-tuning Llama models . . . . .	31
3.2.5	Deployment . . . . .	33
3.3	Discussion . . . . .	34
<b>4</b>	<b>Explainability assistant</b>	<b>37</b>
4.1	Problem statement . . . . .	37
4.2	System architecture . . . . .	38
4.3	Functional overview . . . . .	43
4.4	Trustworthy AI perspective . . . . .	47

<b>5</b>	<b>Results and evaluation</b>	<b>49</b>
5.1	LLM performance analysis . . . . .	49
5.1.1	TalkToModel . . . . .	49
5.1.2	Explainability Assistant . . . . .	50
5.2	Stakeholder assessment . . . . .	54
<b>6</b>	<b>Conclusion</b>	<b>56</b>
<b>7</b>	<b>Acknowledgments</b>	<b>57</b>
	<b>References</b>	<b>60</b>
	<b>Appendix</b>	<b>61</b>
	I. Resources . . . . .	61
	II. Licence . . . . .	62

# 1 Introduction

Over the past ten years, artificial intelligence-driven solutions have become widely adopted in both the private and public sectors, enabled by the persistent efforts of many researchers and practitioners. This integration has allowed individuals and organizations to benefit significantly from intelligent algorithms and automation. However, it has also led to increased public awareness about the risks associated with these applications. AI systems can make various types of errors and their predictions may not only be inaccurate but also be perceived as biased and unjust by human observers, emphasizing the importance of incorporating ethical considerations during development and deployment. In addition, some categories of AI algorithms generate solutions so intricate that they are nearly impossible for humans to understand, raising concerns about the transparency and reliability of these systems.

One initiative addressing these challenges is Trust-AI, a project sponsored by the European Union’s Horizon Europe programme for research and innovation. The project concentrates on creating methods and technologies for interpretable and reliable artificial intelligence and includes input from academic and industry stakeholders. Within the Trust-AI project framework, three case studies illustrate the application of explainable AI techniques in the fields of healthcare, retail, and energy. The approaches employed integrate efficient human-computer interaction with the use of genetic programming, as well as the generation of explanations for machine learning model outcomes using diverse and experimental techniques.

In this study, as part of the Trust-AI project, we examine potential applications of an innovative approach to explainability in machine learning applications through natural language dialogues. Chapter 2 provides an overview of the historical background, highlighting advancements in explainable AI and natural language processing technology. Chapter 3 discusses TalkToModel, a prototype of a conversational system that represents the previous work, and its extensions implemented to support one of the use cases of the Trust-AI project. Chapter 4 details the novel software system developed in this study to overcome the limitations of the earlier solution. Chapter 5 outlines the evaluation methods and results used to assess the enhancements achieved in the systems.

In this study, large language models were essential, serving both as a means to allow open-ended conversations with users and as a means of conducting research and assessing outcomes. The comprehensive prompts and scripts utilized to produce LLM evaluation datasets are available in a distinct repository, referenced in Appendix I. Furthermore, the LLM-based tool Writefull [Wri] was used to refine and format this thesis text through integration within the Overleaf environment.

## 2 Background

This chapter provides the theoretical foundation for developing a conversational explainability system. Section 2.1 details the principles and methodologies of explainable and trustworthy AI, while Section 2.2 reviews advances in language modeling that enable natural language interfaces.

### 2.1 Trustworthy and explainable AI

Before delving further into the enhancement and development of a solution aimed at strengthening the credibility of ML applications, it is crucial to explore the foundational concepts of explainable and trustworthy AI. This will enable us to examine the exact techniques utilized within the system and assess its integrity.

#### 2.1.1 Explainability in intelligent systems

*Explainable Artificial Intelligence* is a branch of artificial intelligence and machine learning dedicated to creating methods and systems that translate the decision-making processes of complex models into explanations that are understandable to humans.

The origin of AI explanations can be traced back to early expert systems of the 1970s. Such systems encoded knowledge explicitly in rule-based architectures and were inherently explainable through traceable logical inferences. An example of such a system is MYCIN, a rule-based medical consultation system developed in 1976 [Sho76]. It was designed to help and educate physicians who needed advice on antimicrobial therapy, and along with a consultation module, it included an explanation system. The program was able to understand simple straightforward questions, but was limited by the processing of natural language, the inability to deal with context, and the lack of support knowledge.

Later iterations of these systems in the 1980s, such as NEOMYCIN and XPLAIN, offered improvements in terms of explanatory capabilities by providing a better and more explicit representation of knowledge. In addition, researchers proposed a *reactive model of explanation* for expert systems that included user feedback and dialogue history for context analysis and answering follow-up questions. However, this system was still limited in the scope of explanations and in its ability to maintain a natural language dialogue [MS88].

In its modern sense, the term Explainable AI (abbreviated XAI) was coined in 2004 in the context of a military simulation system that could answer questions about its actions [vLFM04]. This highlights how the demand for explainability is driven by high-stakes applications in areas such as defense, healthcare, finance, and autonomous driving, where human experts must understand the rationale behind AI decisions. Figure 1 uses Google Trends data to illustrate the revival of interest in XAI as a research topic. This growth can be seen as a direct consequence of the rapid advancement of AI/ML within various

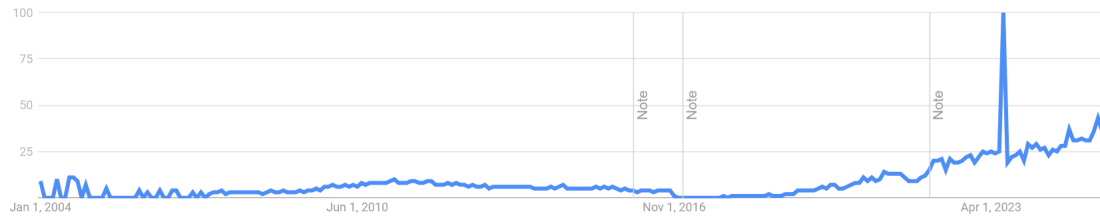


Figure 1. Google Trends result for the topic *Explainable Artificial Intelligence*.

industries, where it significantly impacts the decision-making process. At the same time, the lack of insight into the reasoning of the models leads to a growing social, ethical, and legal demand for new AI methods that could ensure that the results are explainable and understandable [AB18].

### 2.1.2 Taxonomy of explainable AI techniques

The early advances in explanation techniques established a foundation by demonstrating that AI systems could be made interpretable when they are built with transparency in mind from the start. However, with the emergence of ML in the following decades, the focus shifted towards maximizing prediction accuracy. Many ML models, especially those based on deep neural networks and ensemble methods, achieved unprecedented performance, although at the expense of becoming *black boxes*, with internal reasoning obscured from human interpretation. As models grew in complexity, with the number of parameters reaching millions or even billions, the inherent trade-off between accuracy and interpretability led to the emergence of a new research field dedicated to *post-hoc* explanation methods [AB18].

Indeed, as illustrated in a high-level taxonomy in Figure 2, the literature makes a clear distinction between models that are interpretable by design and those that require the use of external XAI techniques. Furthermore, these works define interpretability on different levels, making a distinction between *global explainability* (being able to understand the whole model) and *local explainability* (being able to understand a particular outcome) [ARS<sup>+</sup>19].

Additionally, some researchers include the separate concept of *data explainability* in the general explainable AI framework. The related methods include exploratory data analysis and data visualization, which help users to get a general idea and insight about the dataset. When the number of features is high, the data can be difficult to visualize due to a large number of variables, and in these cases *dimensionality reduction* techniques can be applied to improve interpretation [DDN<sup>+</sup>23].

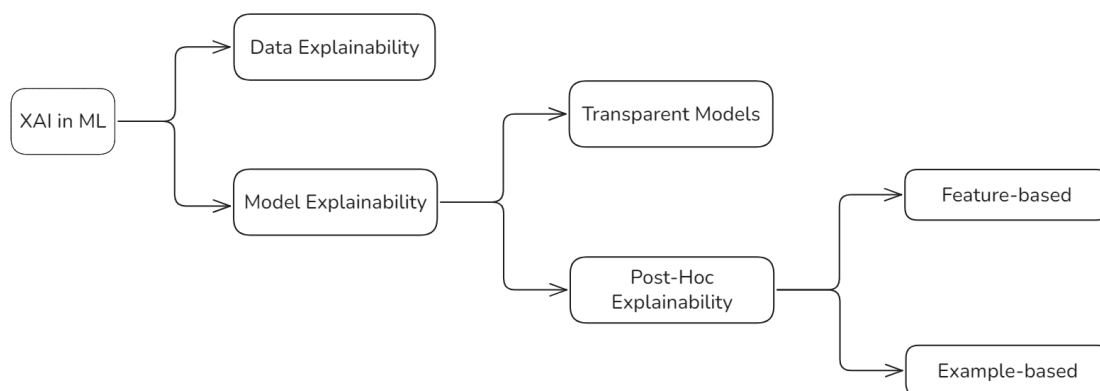


Figure 2. Taxonomy of XAI Techniques related to different ML models.

While transparent models offer a certain level of interpretability on their own, they can also be evaluated based on the specific domain in which they are interpretable.

- *Simulatability* reflects the ability of the model to be simulated, or thought about strictly by a human, therefore it is linked directly to the notion of model complexity.
- *Decomposability* refers to the ability to explain every part of the model, such as input, parameters, and calculation. In other words, it can be described as the intelligibility of individual model parts.
- *Algorithmic Transparency* demands that the model can be fully examined by means of mathematical analysis and methods. It is related to the ability of a human to understand the processes followed by the model.

The literature identifies several types of ML models that, depending on their complexity, may exhibit the characteristics mentioned above. These models include *linear and logistic regression, decision trees, K-nearest neighbors, rule-based learners, General Additive Models, Bayesian models* and *Genetic Programming models* [ARS<sup>+</sup>19].

### 2.1.3 Post-hoc explanation methods

As demonstrated in section 2.1.2, transparent machine learning models offer a direct interpretation of their outcomes. At the same time, many applications where higher accuracy is critical rely on the usage of black-box models, such as *tree ensembles, Support Vector Machines (SVMs)* and *multi-layer neural networks (deep learning models)*. As illustrated in Figure 3, such models generally offer a lower interpretability of the model. In these cases, to achieve better interpretability, practitioners must employ more advanced post-hoc explanation techniques [ARS<sup>+</sup>19]. These methods can be applied after a model

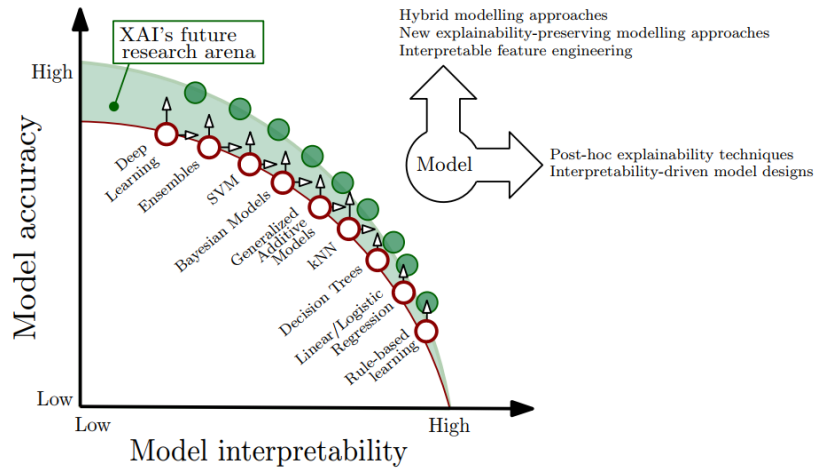


Figure 3. Trade-off between model interpretability and performance, and visualization of the XAI future research potential [ARS<sup>+</sup>19].

is trained, offering local or global explanations for its predictions, and do not require internal changes.

On a high level, post-hoc explainability can be separated into *feature-based* and *example-based* techniques [DDN<sup>+</sup>23]. These methods can be further classified by scope of application as *model-specific* and *model-agnostic* [ARS<sup>+</sup>19]. In the scope of this review, we will introduce only the methods that are relevant to the present work.

*Local Interpretable Model-agnostic Explanations* (LIME) attempts to understand a complex model by perturbing the input data samples and observing how the predictions change. In the next step, LIME approximates the model locally with a simple, interpretable model around the neighborhood of the prediction of interest. This allows to detect which features contributed most to the specific model outcome.

*Shapely Additive Explanations* (SHAP) is an approach that is based on game theory. It is inspired by *Shapley values* from collaborative games and attempts to assign each input feature a value that represents its contribution to the outcome. SHAP ensures that such attributions are consistent and fair.

Both LIME and SHAP are examples of the feature-based, model-agnostic, post-hoc explanation methods. While SHAP is capable of providing explanations on global level, computing Shapely values is computationally expensive. LIME provides an optimal alternative for some type of models, while struggling to cater to others. Therefore, both of these techniques can be powerful tools for assessing *feature importance*, or the impact of an individual feature on a prediction of a model [DDN<sup>+</sup>23].

Alternatively, some researches highlight the importance of the contrastive nature of explanations, as humans tend to be more interested not in why a particular event

happened, but rather in why this event happened instead of some other event [Mil19]. Such explanations are called *counterfactual cases*, and they describe change that can be made to the particular feature values to change the output of the model. Counterfactuals are closely associated with the casual model of the data, providing valuable insights to guide interventions.

Computationally, this type of explanations can be generated with the help of tools such as *Diverse Counterfactual Explanations* (DiCE). Counterfactual explanations, along with automated answers to *what if* questions about the prediction outcomes, represent model-based, post-hoc and example-based explanation methods for XAI that enhance causal interpretability.

#### 2.1.4 Genetic Programming

*Genetic Programming* (GP) is an evolutionary algorithm-based technique inspired by biological evolution that allows creating automatically evolving computer programs while imitating natural selection processes. GP was popularized in the 1990s by John Koza, especially with his 1992 book *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. It extended the ideas of Genetic Algorithms (GAs) that modeled biological evolution using selection, crossover, and mutation. By introducing tree-structured programs, GP allowed for the automatic synthesis of symbolic expressions, giving it the ability to solve complex problems [PLM08].

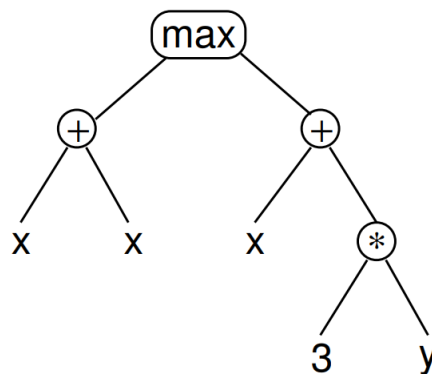


Figure 4. GP syntax tree representing program  $\max(x+x, x+3*y)$  [PLM08].

One of the key characteristics of Genetic Programming is its ability to create models represented as symbolic expressions or tree structures, illustrated in Figure 4, which can be directly inspected and interpreted by human experts, ensuring excellent algorithmic transparency. In contrast to other machine learning methods that result in complex, opaque parameter spaces, the symbolic representation in GP-generated solutions provides transparency and explicitly reveals the connections between input features and out-

puts [MCL<sup>+</sup>23]. This natural interpretability, given a program of human-comprehensible size, aligns well with the requirements for simulatability, decomposability, and algorithmic transparency outlined in subsection 2.1.2, which makes GP models exceptionally valuable in the context of Explainable AI. They enable practitioners to understand, verify, and, if needed, alter the models, which is crucial for establishing trust in decision-making systems.

### 2.1.5 Trustworthy AI

Trust is an important component of the interaction between people and AI. By definition, to be *trustworthy* a system must be designed, developed, and deployed in a way that makes it worthy of human trust. This concept implies that an AI system should be consistent in producing correct, unbiased, and transparent results, which follow ethical principles and legal standards.

The European Union has played a key role in defining the concept of trustworthy AI by being a pioneer in policy and regulatory initiatives. In particular, the European Commission has published the *Ethics Guidelines for Trustworthy AI* [Eur], which postulates that such an AI is based on three components that must be present throughout its entire life cycle. Specifically, such a system should be *lawful* by respecting all applicable laws and regulations, *ethical* by respecting ethical principles and values, and *robust* from both a technical and social perspective to avoid causing unintentional harm. In addition, the Guidelines detail a framework for achieving a trustworthy AI, focusing primarily on the ways to ensure that the system is ethical and robust.

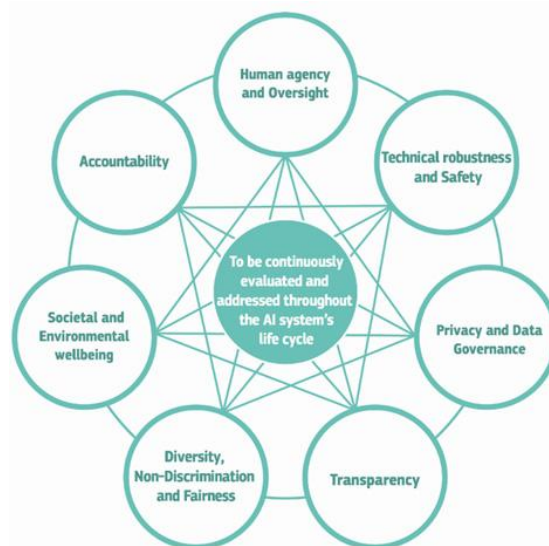


Figure 5. Seven key requirements for trustworthy AI [Eur].

The framework for trustworthy AI is built on the foundation of four key ethical principles that must be respected when building, deploying, and using AI systems. These principles consist of: *respect for human autonomy*, *prevention of harm*, *fairness*, and *explicability*. Furthermore, these principles are translated into concrete requirements, illustrated in Figure 5, which AI systems must meet to be considered trustworthy:

1. **Human agency and oversight** require that people remain in control and informed through mechanisms such as human-in-the-loop approaches.
2. **Technical robustness and safety** require systems to be safe, resilient, and secure, capable of minimizing harm.
3. **Privacy and data governance** require protecting personal data and maintaining high quality and integrity of data.
4. **Transparency** requires clear communication about AI systems, explaining their decisions, and limitations to users.
5. **Diversity, non-discrimination, and fairness** require avoiding bias and involving relevant stakeholders throughout the life cycle of the system.
6. **Societal and environmental well-being** requires AI to be sustainable and socially beneficial, considering its impact on people and the planet.
7. **Accountability** requires that it is possible to assess algorithms, data, and design and ensure the possibility of correction.

The Guidelines place a special emphasis on the importance of these requirements in systems that impact individuals, and further they describe various technical and non-technical methods for their realization. In conclusion, this and other European Commission AI-focused documents provide an important reference and advocate for more human-centric approaches in AI engineering.

## **2.2 Advances in language modeling and processing**

This section reviews advances in natural language processing, from symbolic systems to neural networks and Transformer architecture, and examines their influence on computers' ability to understand and interact with human language through improved text comprehension and generation with models such as GPT, T5, and LLaMA.

### 2.2.1 Natural language processing

*Natural Language Processing* (NLP) is a branch of AI and computational linguistics that concentrates on the interaction between computers and natural (human) languages. It lies at the intersection of computer science, linguistics, and statistics, using methods from each discipline to computationally represent and analyze the structure and functionality of language.

The primary goal of NLP is to enable machines to comprehend, interpret, and generate language in a meaningful and practical way. This allows a wide variety of applications, such as machine translation, information retrieval, sentiment analysis, speech recognition, speech synthesis, and text summarization. One example of a useful NLP task is a *conversational agent*, a concept that captivated the public for decades and was popularized through works of art, for instance, as the HAL 9000 computer in Stanley Kubrick's film *2001: A Space Odyssey* [JM08].

Early efforts in the field were centered on symbolic and rule-based systems, which depended on manually creating grammars, lexicons, and pattern matching rules [Win72]. These systems were useful in controlled settings, such as basic dialogue interfaces or command parsing, but were limited by their inability to generalize and scale efficiently across varied contexts. Their dependence on strict linguistic formalisms made them incapable of handling the ambiguity and variability of the actual human language.

In the 1990s, hand-coded rules were succeeded by data-driven models following the adoption of statistical approaches. Researchers began to model language in terms of distributions and likelihoods, leveraging large text corpora and probabilistic techniques. *Hidden Markov Models* (HMMs) and *n-gram models* gained popularity as standard tools for many tasks that required language modeling [Chu88] [Rab89]. Although these methods provided a major improvement in handling language variation and uncertainty, they were often limited by their dependence on surface-level occurrence statistics and short-range context. In addition, statistical models required complex feature engineering and were sensitive to the amount and quality of training data [JM08].

From the early 2000s, the use of neural networks in NLP became more prominent, first using feedforward architectures and later with *recurrent neural networks* (RNNs) and their variations such as *Long Short-Term Memory* (LSTM) networks [HS97]. Neural networks improved the ability to capture contextual relationships and nuanced meanings in text, although learning long-range dependencies was still limited and their training was complicated by the requirement of sequential processing. As demands on NLP systems increased, the limitations of existing architectures drove the research of novel breakthrough mechanisms.

## 2.2.2 Transformer architecture

The *Transformer architecture* was introduced by Vaswani et al. [VSP<sup>+</sup>17] in 2017, and had a major impact on the state of the art in the field of NLP. It replaced the sequential processing of earlier models such as RNNs and LSTM networks with a new, parallel and context-aware design using a self-attention mechanism that allows the model to consider all words in a sentence simultaneously and to determine how strongly one word influences others. This ability to capture relationships between words regardless of their position in the sentence made Transformers very effective at *recognizing the meaning*, especially when analyzing long and complex texts.

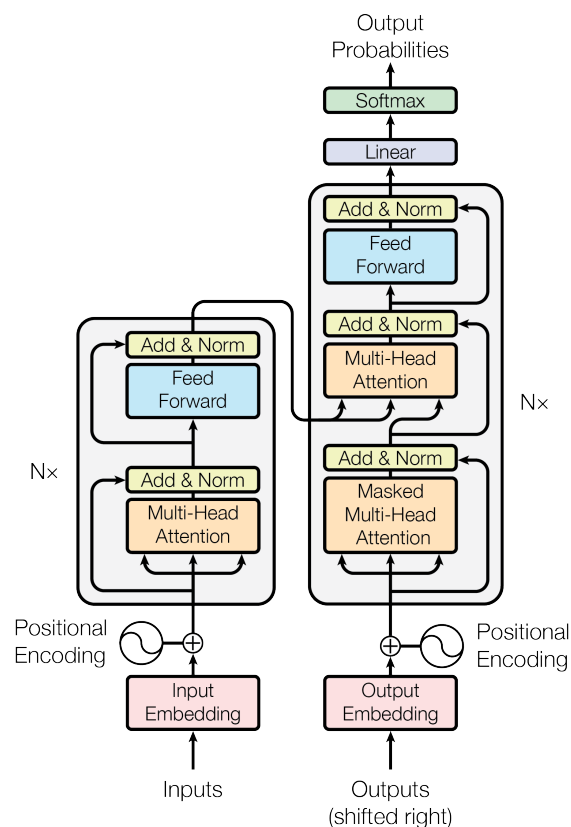


Figure 6. The Transformer model architecture [VSP<sup>+</sup>17].

This new architecture, as shown in Figure 6, consists of layers that include two main components: a self-attention module, which helps determine the importance of different words, and a feedforward network, which further transforms the information. Transformers do not process texts word-by-word in order, but handle full sequences in parallel, which significantly speeds up training and makes them easier to scale on modern hardware. Positional encodings are added to each word to retain the sense of word order, allowing to model both local context and global dependencies.

These improvements in architecture made it possible to train LLMs on very large text datasets. Researchers used Transformers differently: models like BERT [DCLT18] used only the encoder module, processing all tokens in parallel, which allowed learning deep contextual representations of language, and could be fine-tuned for specific tasks, such as sentiment analysis. Other models, such as GPT [RN18], used only the decoder part, predicting the next word given the previous ones, and achieved improvements in text completion and dialogue systems. These models demonstrated the ability to generalize across tasks with little or no task-specific training, known as *few-shot* or *zero-shot learning*.

The flexibility, efficiency, and performance of Transformer architecture made it an essential element of modern LLMs, including conversational systems such as ChatGPT, which provide interactive and context-aware language generation.

### 2.2.3 Generative Pre-trained Transformer

The Transformer architecture served as the foundation for the development of the *Generative Pre-trained Transformer* (GPT) by OpenAI [RN18]. The goal of this research work, published in 2018, was to create a general task-agnostic language model that could perform a wide range of different language tasks using a single architecture. This was achieved through a two-phase process: first, generative pre-training on a large unlabeled corpus, and second, fine-tuning on specific tasks. This approach enabled the model to be enriched with world knowledge and the ability to process long-range dependencies, which was effectively transferred to tasks like question answering, measuring semantic similarity, determining entailment, and classifying text, advancing the state of the art in nine out of the twelve analyzed datasets.

The next generation of GPT, GPT-2, released in 2019, sought to further improve the generalization capabilities of language models [RWC<sup>+</sup>19]. Previously, the prevailing approach to creating ML systems was focused on collecting a large dataset to train a model to imitate the right behaviors and then test its performance. GPT-2 demonstrated that an LLM trained on a sufficiently large and diverse data set is capable of performing well on many different tasks. The model showed a capacity for zero-shot learning to state-of-the-art performance on 7 out of 8 tested datasets.

This strategy was further advanced in GPT-3, an LLM released in 2020, which boasted 175 billion parameters, which was significantly larger than any previous language model [BMR<sup>+</sup>20]. It showed strong performance on many NLP tasks in zero-shot, one-shot, and few-shot settings, in some cases scoring close to state-of-the-art fine-tuned systems. In November 2022 OpenAI released ChatGPT, a conversational model based on the GPT-3.5 series [Ope]. This fine-tuned version of the GPT-3 architecture was specifically improved for dialogue through reinforcement learning from human feedback.

The global success of ChatGPT has fueled public interest in language models and conversational interfaces. Further research led to improvements in later models, such

as GPT-4, a large multimodal model capable of processing image and text input, which exhibited human-level performance on certain difficult professional and academic benchmarks [AAA<sup>+</sup>24].

#### **2.2.4 Text-To-Text Transfer Transformer**

*The Text-To-Text Transfer Transformer* (T5) is a family of models introduced by Google Research in 2019. These foundational LLMs were designed to integrate various NLP tasks into a single framework [RSR<sup>+</sup>20].

The publication included a systematic study of various aspects of language understanding tasks, such as architectures, unsupervised objectives, datasets, training strategies, and others. The authors trained the new model on a curated corpus called *Colossal Clean Crawled Corpus* (C4), resulting in multiple variants of T5 ranging from small (77 million) to XXL (11 billion) number of parameters.

Among the innovations of this work was the introduction of a text-to-text format for the generalization of language problems. Additionally, the work standardized task-specific prefixes, such as "summarize" and "translate from English to German". The versatile and general-purpose architecture of the model has impacted further research, contributing to the adoption of text-to-text transfer learning as an influential approach in contemporary NLP.

#### **2.2.5 Large Language Model Meta AI**

*The Large Language Model Meta AI* (Llama, formerly spelled as LLaMA) is a family of LLMs developed by Meta (formerly Facebook) starting from February 2023. These models were designed to be efficient and accessible for research and commercial use, and their source code is available to the general public. The models come in different sizes in terms of the number of parameters (from 1 billion to 2 trillion), and to date five generations of Llama models have been released.

A notable quality of the original Llama was its dedication to using only publicly available data for training. Llama did not rely on private and undocumented data, unlike many popular counterparts, such as GPT, and this made the models open-source compatible. In addition, the Llama models demonstrated top-level performance in many evaluation tasks, including code generation [TLI<sup>+</sup>23].

The second generation of Llama models introduced, along with new larger foundation language models, a different type of Llama 2-Chat models that were fine-tuned to be optimized for dialogue use cases. These models have demonstrated competitiveness with existing open source chat models and competency equivalent to some proprietary models, although they lagged behind other models such as GPT-4 [TMS<sup>+</sup>23].

The following generations of Llama consist of larger, more efficient models, which continue to deliver improvements and better performance on par with other popular

model families. As discussed in more detail in Chapter 3 and Chapter 4, in this work, we explored both local applications of the Llama 2 and Llama 2-Chat models, as well as the incorporation of the Llama 3.3 instruction-optimized models over the API.

### **2.2.6 Google Gemini**

The Gemini family of models was introduced by Google in 2023 [ABA<sup>+</sup>23], and represents the latest advancements of the company in AI. It was developed by Google Research as a successor to its previous large language models, such as T5 and PaLM (Pathways Language Model) in an effort to combine the strengths of LLMs with reinforcement learning and multimodal capabilities, integrating text, code, images, audio, video, and reasoning within a unified framework.

Gemini architecture builds on the Transformer model, integrating novel components like *mixture-of-experts* (MoE) and expanded context windows, which allows handling complex tasks more efficiently. The model family includes variants such as Gemini Ultra, Pro, Flash, and Nano, which are created for specific applications ranging from high-performance computing to on-device processing. Subsequent versions of the model family maintain their state-of-the-art status, allowing considerably larger context windows and enhanced performance [GLB<sup>+</sup>24].

### 3 TalkToModel

This chapter presents the conversational explainability system, TalkToModel [SKLS23], which serves as an example of earlier studies in the realm of natural language interfaces in the context of XAI. Section 3.1 provides a functional overview and analysis, while Section 3.2 describes the extensions that were necessary to customize the system in a new application use case.

#### 3.1 System analysis

TalkToModel is a prototype of an XAI system, in which the authors developed novel approaches to ML explainability through a conversational interface powered by LLMs. This section provides an overview of the system, its architecture, and key working principles.

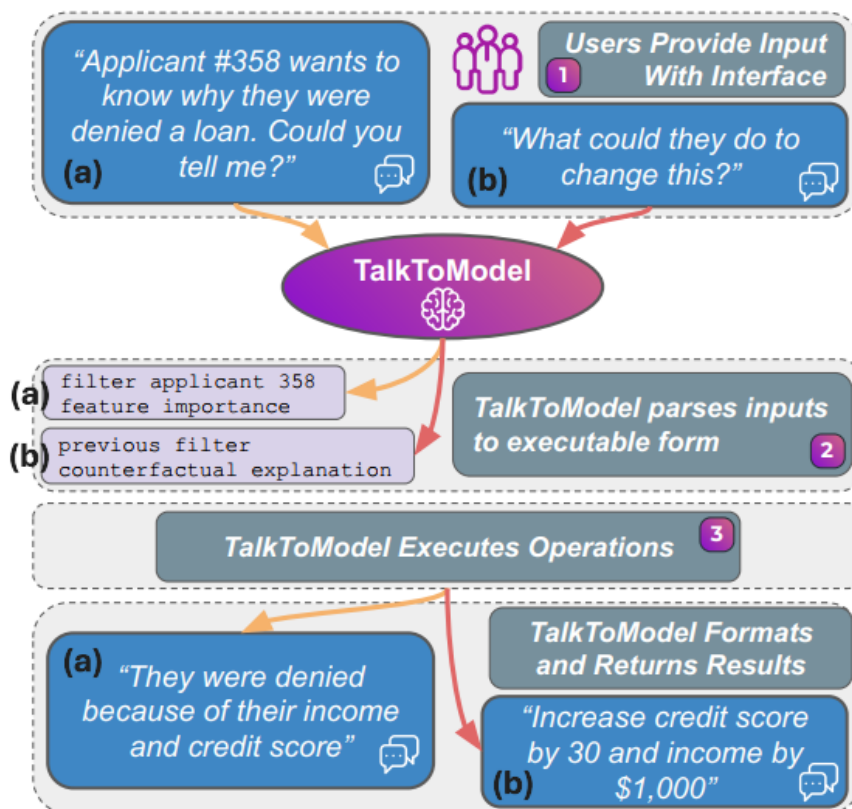


Figure 7. Overview of the TalkToModel workflow [SKLS23].

### 3.1.1 Overview

As described in Section 2.1.1, the idea of employing natural language as a means of XAI has been explored for several decades. TalkToModel serves as a contemporary illustration of a system that facilitates *open-ended conversations* as an explainability tool. Specifically, it leverages the recent advances in LLMs to support comprehension of user's questions and provide answers in the context of an ML problem.

Figure 7 illustrates a high-level representation of the system workflow. In this scenario, the user poses two questions to TalkToModel regarding the decisions made by an ML model within the financial sector, specifically regarding the denial of a loan application. For enhanced clarity, we have labeled the question instances (*a*) and (*b*). The following steps detail the user's interaction with the system:

1. **User submits questions through the chat interface.** The interaction is facilitated through the *user interface* (UI). The input is expected to predominantly take the form of questions concerning the predictions made by an ML model.
  - (a) First, the user asks why a specific applicant, referenced by ID 358 in the data, was denied a loan.
  - (b) Secondly, the user asks a follow-up question about possible changes in conditions that could lead to a different outcome.
2. **TalkToModel converts the input to executable form.** This step is performed via an LLM integration and is necessary to bridge the gap between natural language and machine-executable commands.
  - (a) The first question is converted to:  
`filter applicant 358 feature importance`
  - (b) The second question is converted to:  
`previous filter counterfactual explanation`
3. **TalkToModel executes the commands.** In this phase, the command sequence, composed as a single string of text in the preceding step, is systematically parsed into discrete commands. Subsequently, these commands are executed by the logic unit, resulting in outcomes that are rendered as predefined text templates populated with the corresponding data results. The resulting outcomes are sent to the UI.
  - (a) The system indicates that the denial of the loan was influenced by the factors of income and credit score.
  - (b) The system suggests modifications that may be implemented to the initial data to obtain a different prediction (counterfactual explanation).

TalkToModel offers support for a range of various XAI tools and methods. For example, it allows users to query the system with regard to the data and examine individual data samples or groups. Moreover, predictions for these samples can be generated using the ML model provided in the currently configured use case.

Additionally, predictions can be explained through the integration of modules such as LIME and SHAP. These modules are incorporated into a sophisticated algorithm that produces explanations, specifically feature importance, utilizing both methods, and implements a strategy to select the most effective explanations from those generated by the two modules.

TalkToModel also integrates with the DiCE module, which offers counterfactual explanations. Furthermore, it facilitates addressing *what-if* scenarios and analyzing prediction errors when the ground-truth labels are included in the dataset pertinent to the configured use case.

### 3.1.2 System architecture

The operational framework of TalkToModel is illustrated as a data pipeline in the context of the core components of the system in Figure 8. The UI, illustrated in Figure 9, is the part of the application responsible for receiving unstructured user input (utterances, chat messages) and for presenting the current conversation. The input is transmitted to the back-end, which forwards it to the LLM along with a set of system instructions. The output produced by the LLM comprises a single string of commands, which adheres to the predefined syntax, and represents a set of instructions. This command string is relayed back to the back-end application, where it is parsed into individual commands, leading to the execution of the corresponding operations. The results of these actions are processed and formatted with the use of a template engine and then communicated to the user through the UI in the form of a chat message.

The interface and back-end are integrated within a single *Flask* server, representing a *monolithic application*. In order to run the application, it is necessary first to locally store an instance of the *dataset* along with the *ML model* that constitute the specific use case for exploration through TalkToModel conversations. In the paper, the authors examined and presented three distinct use cases: the Pima Indian Diabetes Dataset, the German Credit Dataset, and the COMPAS recidivism dataset, accompanied by *gradient-boosted decision tree* ML models, which were trained to solve these particular classification tasks.

The design of the system implies a degree of configurability, facilitated via *Gin* configuration files, which must indicate the values of the required configuration variables. These variables include the dataset and the model file paths, the LLM employed for parsing user inputs, and other variables that specify the characteristics and attributes of the data, as well as the parameters of the explanation generation engine.

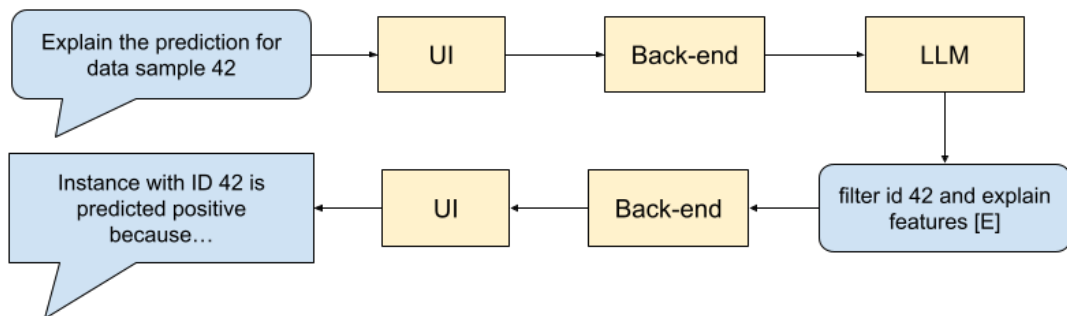


Figure 8. TalkToModel as a data pipeline. The user submits the question "Explain the prediction for data sample 42" through the system's UI. This input is subsequently transformed by the LLM into a command string filter id 42 and explain features [E]. This command string is parsed and executed within the back-end application, which has access to the use case dataset and the ML model. The outcome of these executed operations is integrated into a predefined answer template, producing a coherent paragraph of text. This paragraph is then sent back to the UI and presented as a chat message on behalf of TalkToModel.

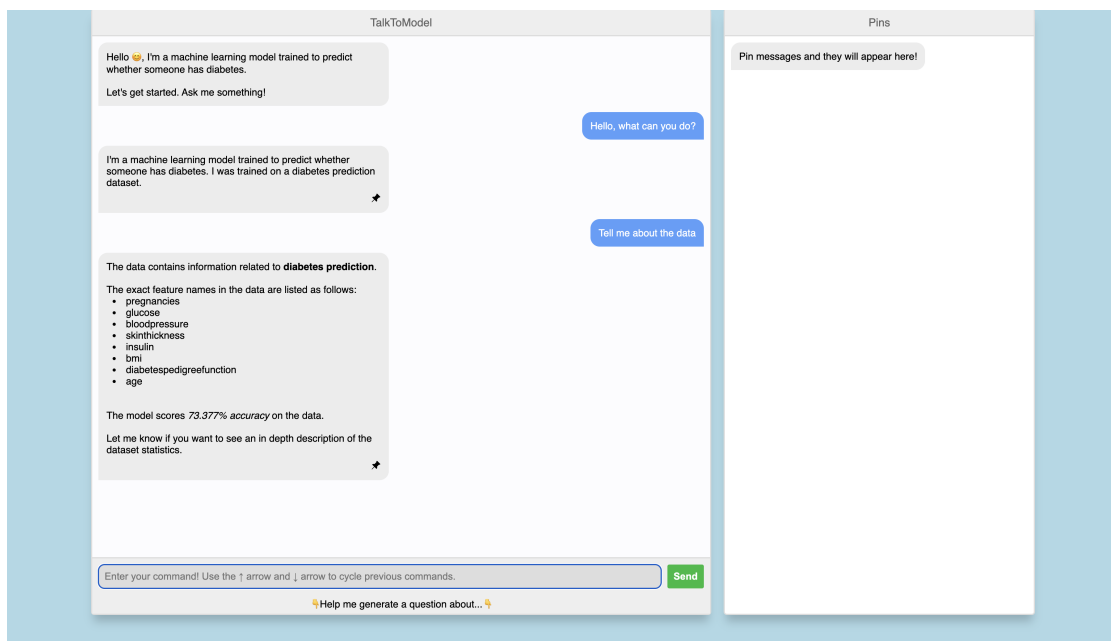


Figure 9. TalkToModel UI. On the left-hand side, the conversation history is arranged from top to bottom. On the right-hand side, there is a space for *pins*, which in this context are saved messages. At the bottom, the interface includes an input field alongside a collapsible menu containing example prompts.

TalkToModel relies on numerous Python packages, including those necessary for fine-tuning LLMs and the facilitation of inference process. Consequently, another prerequisite to launching the application involves the creation of a Python environment that incorporates all these third-party libraries in their designated versions. Once this environment is created and the configuration variables are accurately specified, the application can be executed with a single command: `python flask_app.py`.

During the initial execution, the system performs several computationally intensive operations:

1. The LLM specified in the configuration file is downloaded from the *HuggingFace Hub*, a repository of open-source ML models.
2. The system proactively generates the feature importance attributions and counterfactual explanations for each instance in the use case dataset. The resultant data is then stored in the local cache.
3. If specified within the configuration parameters, the system generates a large synthetic dataset of prompt-parse pairs derived from multiple dynamic prompt templates. These templates are filled as wildcards with the feature names, tailoring this dataset specifically to the data associated with the configured use case. The data is subsequently cached and can be used for LLM fine-tuning for the configured use case. This computationally intensive step can be skipped if a fine-tuned LLM model is already present locally.

Once the system is fully loaded, the user can navigate to the front-end UI via a web browser to begin interacting with the application. Upon submission of a new message into the input field, it is sent to the back-end system through an API request, triggering the conversion of the message into executable commands.

### 3.1.3 Operation execution

The language used to describe commands in TalkToModel deviates from traditional programming syntax, even though it is described by the authors as an *SQL-like programming language*. It adheres to a particular grammatical structure, where operation invocations from a strictly defined list may be connected by conjunctions and followed by numerals or keywords that denote command arguments. The comprehensive list of supported operations is shown in Figure 10. In addition, a custom token [e] is utilized to indicate the end of the command sequence.

Several of the TalkToModel operations are straightforward and do not require arguments; for instance, the `function` and `model` commands merely return predefined texts that describe the system's functionality and the model type, as specified in the configuration file. In contrast, other operations are more complex and require the use

	operation, arguments, and description
Data	<p><code>filter(dataset, feature, value, comparison)</code>: filters <code>dataset</code> by using value and comparison operator</p> <p><code>change(dataset, feature, value, variation)</code>: Changes <code>dataset</code> by increasing, decreasing, or setting feature by <code>value</code></p> <p><code>show(list)</code>: Shows items in list in the conversation</p> <p><code>statistic(dataset, metric, feature)</code>: Computes summary statistic for <code>feature</code></p> <p><code>count(list)</code>: Length of list</p> <p><code>and(op1, op2)</code>: Logical “and” of two operations</p> <p><code>or(op1, op2)</code>: Logical “or” of two operations</p>
Explainability	<p><code>explain(dataset, method, class=predicted)</code>: Feature importances on <code>dataset</code></p> <p><code>cfe(dataset, number, class=opposite)</code>: Gets <code>number</code> counterfactual explanations</p> <p><code>topk(dataset, k)</code>: Top <code>k</code> most important features</p> <p><code>important(dataset, feature)</code>: Importance ranking of <code>feature</code></p> <p><code>interaction(dataset)</code>: Interaction effects between features</p> <p><code>mistakes(dataset)</code>: Patterns in the model’s errors on <code>dataset</code></p>
ML	<p><code>predict(dataset)</code>: Model predictions on <code>dataset</code></p> <p><code>likelihood(dataset)</code>: Prediction probabilities on <code>dataset</code></p> <p><code>incorrect(dataset)</code>: Incorrect predictions</p> <p><code>score(dataset, metric)</code>: Scores the model with <code>metric</code></p>
Conv.	<p><code>prev_filter(conversation)</code>: Gets last filters</p> <p><code>prev_operation(conversation)</code>: Gets last non-filtering operations</p> <p><code>followup(conversation)</code>: Respond to system followups</p>
Description	<p><code>function()</code>: Overview of the system’s capabilities</p> <p><code>data(dataset)</code>: Summary of dataset</p> <p><code>model()</code>: Description of <code>model</code></p> <p><code>define(term)</code>: Defines <code>term</code></p>

Figure 10. The list of operations supported by TalkToModel [SKLS23]. The exact names of the operations in the final version of the system differ from those listed in this and other TalkToModel figures.

of positional arguments or data filtering. For example, the `define` command must be followed by the name of the feature to be defined and returns a part of the configuration texts that describes it.

Most operations rely on *filtering* operation to perform queries on a particular subset of the dataset. This is achieved through manipulations on a temporary dataset, which is accessible to all commands within the context of the ongoing conversation. For instance, the command string:

```
filter id 42 and explain features [E]
```

means that initially, the dataset filtering command is used to select one sample, identified by ID 42. Subsequently, the operation `explain` is executed with two arguments: the current state of the conversation dataset, which was filtered by the previous command (in this scenario, to a single instance) and a method of explanation (in this case, `features`). The operation `filter` is capable of selecting larger subsets of data. For example:

```
filter age greater equal than 50
```

will select all data samples where the feature `age` is greater than or equal to 50, thus including all individuals who are at least 50 years old within the context of the Pima Indian Diabetes Dataset use case.

Furthermore, the system establishes a number of operations that facilitate context inference by referencing to the previous chat message. In the example in Figure 7, the user asks "*What could they do to change this?*", referring to the patient mentioned in the prior question. The system subsequently transforms this question into the following command:

```
prev_filter and cfe [E]
```

where the operation `prev_filter` is invoked to retrieve the conversation history, which is stored in the back-end during the active phase of the interaction. This operation is used to extract the preceding filtering command and reapply it to the operation `cfe`, thereby providing the relevant counterfactual explanation.

The ability to access conversation history allows TalkToModel to provide responses to questions in a more natural manner. Consequently, the user is not required to explicitly define filtering parameters or specify operations with each interaction if they want to refer to the same entity in the next question. In the majority of cases, the system proficiently infers the relevant context from prior user input, even when it is not explicitly articulated in the current utterance.

However, the methodology used to access the context revealed several shortcomings. A notable constraint exists concerning the extent of contextual operability, restricting the system to a single-level depth in the filtering process, as demonstrated in the following dialogue:

1. What is the number of samples with BMI above 40?  
→ `filter bmi greater than 40 and countdata [e]`
2. How many have glucose above 120?  
→ `filter glucose greater than 120 and countdata [e]`
3. How many have both?  
→ `previousfilter and countdata [e]`

In this instance, the system fails to integrate the two preceding filtering commands into a single entity, resulting in the application of only the most recent filter. This operation yields an outcome analogous to that of the second command, which is evidently not a correct interpretation of the question. A similar constraint exists with respect to inquiries that reference multiple operations executed in previous queries utilizing the `previousoperation` command.

### 3.1.4 Application of LLMs

The key element of the TalkToModel system is its integration with LLMs. When a user submits an input through the chat interface in the form of a text message, the LLM is tasked with correctly interpreting the user’s intent behind the message and converting it into a sequence of executable command operations.

Given that the syntax and grammar governing the operation calls have been designed by the authors of TalkToModel for this specific application, these rules are not encompassed within the corpus of general knowledge internalized by general-purpose LLMs during their training phase. Consequently, to achieve a satisfactory degree of precision in translating user input into commands, it is imperative to *fine-tune* the LLM or employ *few-shot learning* methods that include examples of correct translations.

Figure 11 shows the TalkToModel evaluation results of various GPT and T5 models. For GPT models, few-shot learning was used by providing a specified number of example parses (5, 10, 20, or 30) directly within the LLM prompt alongside user input. In contrast, the T5 models underwent fine-tuning on an extensive synthetic dataset of prompt-parse pairs before being prompted with user input only. As demonstrated in the table, the fine-tuned T5 models, particularly the larger variant, achieved substantially better accuracy in all use cases.

Unfortunately, the LLMs evaluated in the TalkToModel framework exhibited a final peak accuracy of approximately 76%, rendering them suboptimal for applications in real world scenarios. This limitation is particularly evident with respect to trustworthiness, as the system generates erroneous responses in about 24% of the instances. Moreover, the system fails to provide explanatory insights into the rationale behind its conclusions, thereby rendering the erroneous responses challenging for users to identify, given their presentation with high confidence.

	German			Compas			Diabetes		
	IID	Comp.	Overall	IID	Comp.	Overall	IID	Comp.	Overall
Nearest Neighbors	26.2	0.0	16.5	27.4	0.0	21.9	10.9	0.0	8.4
GPT-Neo 1.3B									
10-SHOT	41.3	4.1	27.5	35.9	0.0	28.8	40.1	7.0	32.6
20-SHOT	39.7	0.0	25.0	39.3	0.0	31.5	42.9	2.3	33.7
30-SHOT	42.9	0.0	27.0	39.3	0.0	31.5	41.5	4.7	33.2
GPT-Neo 2.7B									
5-SHOT	38.1	4.1	25.5	35.9	3.4	29.5	46.9	7.0	37.9
10-SHOT	38.1	6.8	26.5	40.2	3.4	32.9	40.8	9.3	33.7
20-SHOT	39.7	0.0	25.0	39.3	0.0	31.5	42.9	2.3	33.7
GPT-J 6B									
5-SHOT	51.6	14.9	38.0	51.3	6.9	42.5	55.8	7.0	44.7
10-SHOT	57.9	9.5	40.0	49.6	3.4	40.4	53.7	9.3	43.7
T5									
SMALL	61.1	32.4	50.5	71.8	10.3	59.6	77.6	30.2	66.8
BASE	68.3	<b>48.6</b>	61.0	65.0	10.3	54.1	<b>84.4</b>	34.9	73.2
LARGE	<b>74.6</b>	44.6	<b>63.5</b>	<b>76.9</b>	<b>24.1</b>	<b>66.4</b>	<b>84.4</b>	<b>51.2</b>	<b>76.8</b>

Figure 11. The outcomes of the evaluation of various LLMs on gold parse datasets. For every use case presented in TalkToModel, the parsing accuracy is expressed as the percentage of the total samples correctly converted [SKLS23].

For example, LLMs in TalkToModel frequently demonstrate difficulty in generating complex command groupings and tend to include only one of the commands requested. Furthermore, they exhibit a tendency to *hallucinate* filtering values that are irrelevant to the user’s request, as illustrated in the following dialogue:

1. I want to see ids and predictions where age is below 30  
→ filter age less than 30 and show [e]
2. Now for age below 40 and predictions  
→ filter age less than 40 and filter pregnancies less than 40 and previousoperation [e]

In this instance, the LLM encounters difficulties in formulating an accurate query, exhibiting a failure to properly chain commands show and predict, potentially due to inadequacies in the fine-tuning examples. Furthermore, LLM hallucinates an erroneous filter filter pregnancies less than 40, which emerges as a frequent error pattern in various use cases.

At the same time, it is worth noting that typically model performance shows an improvement in correlation with an increase in model size, thereby suggesting a valuable insight for future work.

## 3.2 System extensions

This section describes the modifications that were necessary to adapt the TalkToModel system to a new use case. This use case, provided by the Trust-AI stakeholder Apintech, employs a GP model to predict energy consumption based on an original dataset collected from a building management system [SYS<sup>+</sup>23]. In addition, this section lists the experiments that were carried out to augment the system, including the integration of new LLMs and frameworks.

### 3.2.1 Regression problems

TalkToModel as a prototype system described in Section 3.1 demonstrated its ability to integrate with use cases where machine learning models are applied to solve *binary classification* tasks. These tasks involve the classification of data samples into two distinct (opposing) categories. However, many practical applications are developed to solve regression problems, whereby the solution is represented as a singular continuous value predicted from an array of inputs. In particular, within the Apintech use case of the Trust-AI project, a GP model is used to forecast the energy consumption value based on the features present in the dataset.

```
1 def load_model(self, filepath: str):
2     """Loads a model.
3
4     This routine loads a model into the conversation
5     from a specified file path. The model will be saved as a variable
6     names 'model' in the conversation, overwriting an existing model.
7
8     The routine determines the type of model from the file extension.
9     Scikit learn models should be saved as .pkl's and torch as .pt.
10
11     Arguments:
12         filepath: the filepath of the model.
13     Returns:
14         success: whether the model was saved successfully.
15     """
16     app.logger.info(f"Loading inference model at path {filepath}...")
17     if filepath.endswith('.pkl'):
18         model = load_sklearn_model(filepath)
19         self.conversation.add_var('model', model, 'model')
20         self.conversation.add_var('model_prob_predict',
21                                 model.predict_proba,
22                                 'prediction_function')
23     else:
24         # No other types of models implemented yet
25         message = (f"Models with file extension {filepath} are not supported."
26                  f"You must provide a model stored in a .pkl that can be loaded"
27                  f"and called like an sklearn model.")
28         raise NameError(message)
29     app.logger.info("... done")
30     return 'success'
```

Listing 1. Method for loading the use case model in TalkToModel.

The system code makes extensive reference to the method `predict_proba`, which is characteristic of the Python library `scikit-learn`, and is utilized to return the predicted probabilities for each *class* within a specified classification task. During the instantiation of the high-level `ExplainBot` object, a use case model file is loaded utilizing the `load_model` method, as demonstrated in Listing 1. In line 20, the reference to `predict_proba` is assigned as a variable within the `Conversation` object instance. Subsequently, this method reference is propagated to other essential components of `TalkToModel`, namely, the `MegaExplainer` and `Explainer` objects, which encapsulate the primary logic for the generation, selection, and presentation of explanations.

Machine learning models applied to solving regression problems do not utilize the method `predict_proba`, as class probability prediction is not part of a regression task. Consequently, to mitigate this incompatibility and prevent system failures during initialization, it was necessary to exclude all references to `predict_proba` from the code base. Consequently, the code was refactored, where applicable, substituting `predict_proba` with `predict`, which yields a singular prediction value. This modification impacted various system operations, including those responsible for generating and delivering model predictions to the user, computing the accuracy of the model, and evaluating other metrics. Removal of references to *classes* from the system required refactoring of certain code segments where a list of *class names* was invoked. For example, generating the synthetic prompt set through dynamic prompt wildcards would lead to failure without modifying the references of the `class_dict` or class dictionary.

In summary, the adaptation of the `TalkToModel` code to accommodate regression problems required the elimination of references to class names and class probabilities. This modification impacted all principal components of the system associated with the explanation logic and mandated the refactoring of essential modules and operations.

### 3.2.2 GP support

The extension of `TalkToModel` to incorporate support for regression problems allowed its integration with new types of use case models, notably *symbolic regression*, within the system. This enhancement was required to adapt the system specifically for the Apintech use case within the Trust-AI framework, which thoroughly investigated the applications of GP models in the domain of energy consumption forecasting [SYS<sup>+</sup>23]. As elaborated in Section 2.1.4, GP models provide an optimal equilibrium between performance and transparency of reasoning within the context of XAI.

Consequently, this study expanded the `TalkToModel` framework by integrating the Python package `gplearn`, thus enabling the incorporation of GP models for interaction via the conversational interface. Furthermore, we have performed experimental studies on the integration of other GP libraries, such as GP-GOMEA, which contains such particular implementation as the Gene-pool Optimal Mixing Evolutionary Algorithm for Genetic Programming.

### 3.2.3 Fine-tuning T5 models

Among the various large language models evaluated by the authors, the T5-large models demonstrated the best performance, achieving an overall accuracy ranging from 63.5% to 76.8% in translating user utterances into executable commands. These models require fine-tuning tailored to the specific use case, given that the composition of the command grammar is dependent upon the feature names. Illustrative examples of the new grammar constructed for the Apintech use case are presented in Table 1.

The scripts employed for model fine-tuning are included within the original TalkTo-Model system and its accompanying documentation. However, prior to their utilization, it is necessary to create a fine-tuning configuration file and modify the code to support a new dataset.

Firstly, it is necessary to create a high-level configuration file within the `configs` directory to specify the details of the use case dataset, as well as the ML model. This configuration maps the paths to the training and testing subsets of the use case dataset, indicates types of features as either numerical or categorical, declares the target variable name, and the paths to temporary cache files, among other parameters. Secondly, it is required to add a reference to the configuration file within the `parsing/t5/get_config_loc` function, which associates the location of the use case configuration file with the dataset name provided at the script invocation, as illustrated in Listing 2.

```
1 def get_config_loc(dataset: str) -> str:
2     """Gets the location of a dataset's configuration file"""
3     if dataset == "energy":
4         config_loc = "./configs/energy-config.gin"
```

Listing 2. Including Apintech use case dataset in the name-configuration mapping.

The fine-tuning script for a T5 model leverages the provided configuration file to instantiate the `ExplainBot` object, facilitating access to the synthetic prompt set used for fine-tuning. This prompt set is generated during the instantiation of the `Prompts` object, which itself is associated with the `ExplainBot` instance.

The fine-tuning script extracts the synthetic prompts and transforms them into a `csv` dataset, which is subsequently used as a training and evaluation corpus in the fine-tuning process of a T5 model. The size of the dataset depends on the configuration of dynamic prompts, which can be altered in the `explain/prompts/dynamic` directory. Placeholders within dynamic prompt templates are populated with the use case-specific values, such as feature names and example values, resulting in the generation of an exhaustive dataset. In the process of adaptation of the TalkToModel system to the Apintech use case, the script generated a synthetic dataset of 11000 samples, illustrative examples of which are presented in Table 1. The link to the full dataset is available in Appendix I.

Upon the initial launch of the fine-tuning script using a new user-provided dataset, the synthetic dataset will be generated, cached in the directory `parsing/t5/datasets`, and subsequently utilized in subsequent fine-tuning sessions.

natural_language	parsed_utterance
how often are you right?	score default [e]
what are your functions?	function [e]
how important is outdoor temperature for the predictions?	important outdoor_temperature [e]
what are the statistics of indoor temperature like?	statistic indoor_temperature [e]
tell me how many ids in the data have indoor temperature less than 15.25	filter indoor_temperature less than 15.25 and countdata [e]

Table 1. Examples of the synthetic prompt dataset samples generated for the fine-tuning of a T5 model in the context of the energy consumption use case.

Optimal fine-tuning of a large T5 model necessitates the use of a GPU-powered environment. In this study, the computational resources of the University of Tartu High Performance Computing Center (UTHPC) were employed. The fine-tuning process of the T5 model was executed using a single *NVIDIA Tesla V100 GPU* in multiple experiments, using the following *Slurm* job script illustrated in Listing 3.

```

1 #!/bin/bash
2
3 #SBATCH --job-name="T5: Finetune T5 model"
4 #SBATCH --partition=gpu
5 #SBATCH --gres=gpu:tesla:1
6 #SBATCH --cpus-per-task=8
7 #SBATCH --mem=15000MB
8 #SBATCH --time=08:00:00
9
10 module load any/python/3.9.9
11 source ~/ttm/venv_energy/bin/activate
12
13 srun python ./parsing/t5/start_fine_tuning.py --gin
    ./parsing/t5/gin_configs/t5-large.gin --dataset "energy"

```

Listing 3. Slurm job script for fine-tuning a T5-large model on the energy use case dataset dataset.

### 3.2.4 Fine-tuning Llama models

In an effort to validate the results and explore the range of models compatible with TalkToModel, various models from the Llama family, described in Section 2.2.5, were subjected to fine-tuning to execute the task of user input parsing. In particular, the study involved Llama-2-7B-Chat and Llama-3-8B-Instruct, which represent some of the smallest models among the second and third generations of Llama models, respectively, and therefore are suitable for fine-tuning with comparatively modest resources.

The fine-tuning and inference tasks were executed utilizing the official Meta script library, llama-recipes, which is described by its developers as a companion tool to the Llama models. This library served as an initial framework for tailoring the fine-tuning procedure to incorporate the synthetic dataset described in Section 3.2.3.

Firstly, the synthetic training dataset generated for the purpose of fine-tuning the T5 model required a specific wrapping method. This method ensured accurate partitioning of the dataset, application of a specific prompt template to the model, tokenization, and processing of inputs in the required format. Furthermore, the HuggingFace library datasets was utilized to load the dataset from the source file and provide additional functionalities. For example, during the fine-tuning process of a Llama model, it was necessary to apply the appropriate template format to the data, consistent with the template used during the training phase. Listing 4 and Listing 5 demonstrate the differences between the prompt templates of Llama 2 and Llama 3.

```
1 <s>[INST] <<SYS>>
2 {{ system_prompt }}
3 <</SYS>>
4
5 {{ user_message }} [/INST]
```

Listing 4. Llama 2 prompt template.

```
1 <|begin_of_text|><|start_header_id|>system<|end_header_id|>
2
3 {{ system_prompt }}<|eot_id|><|start_header_id|>user<|end_header_id|>
4
5 {{ user_input }}<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

Listing 5. Llama 3 prompt template.

The engineering of *system prompts* constitutes an important component of both the fine-tuning and inference processes in LLMs. The prompt serves a pivotal function by providing instructions and supplementary context to the model, thereby directing the outcome towards the intended result. The example provided below illustrates our experiments in guiding the model response toward the correct action through a detailed system prompt that was employed in application to a Llama model.

Your task is to convert the following chat message to a command composed in a custom query language which consists from a limited number of command tokens. The command tokens that you can use are: 'interact', 'countdata', 'filter', 'explain features', 'explain cfe', 'predict', 'self', 'previousfilter', 'previousoperation', 'data', 'followup', 'important', 'important topk', 'important all', 'show', 'change', 'model', 'function', 'score default', 'score accuracy', 'label', 'mistake typical', 'mistake sample', 'statistic', 'define'. You can also use special tokens as arguments to commands, such as: 'greater', 'less', 'than', 'equal to', 'and', 'increase', 'decrease', 'set'. You can also use dataset feature names: 'id', 'outdoor\_temperature', 'indoor\_temperature', 'past\_electricity'. Every generated command sequence is closed with a special token '[e]'. The chat message is following:

Furthermore, in the experiments carried out, we used techniques of *low-rank adaptation* (LoRA) and *8-bit quantization* to enhance performance and alleviate the computational demands of the resource intensive fine-tuning process. The fine-tuning process was executed on *NVIDIA A100 Tensor Core* GPUs equipped with 40-80GB of RAM, provided by UTHPC. Listing 6 presents an example of a script utilized to fine-tune a Llama model.

```
1 python llama-recipes/recipes/finetuning/finetuning.py --dataset "custom_dataset"
   --custom_dataset.file llama-dataset/energy_dataset.py --use_peft --peft_method
   lora --quantization --model_name meta-llama/Llama-2-7b-chat-hf --output_dir
   llama-finetuning/finetuned/llama-2-7b-chat
```

Listing 6. The command for fine-tuning Llama-2-7b-chat model with a custom dataset.

### 3.2.5 Deployment

In order to facilitate evaluation of the complete end-to-end solution by relevant stakeholders, it was necessary to deploy it within an appropriate environment and to make the UI accessible to the public. Consequently, a part of the current work focused on establishing and hosting an operational instance of the system on a publicly accessible server. The virtual machine (VM) hosting services were provided by UTHPC, and configured with the resources detailed in Table 2.

CPU	4 vCPUs
RAM	16 GB
Storage	30 GB
Operating system	Ubuntu 22.04

Table 2. Resources of the VM hosting a TalkToModel instance.

TalkToModel was developed as a Flask application that exposes an API and a template-based UI. Consequently, the primary objective of this part of the project was to determine the optimal deployment of such a system. The officially endorsed method for hosting a Flask application in a production environment is Gunicorn, a Python server for UNIX systems. Employing a production server improves the stability, security, and efficiency of the application, making it the preferred choice for software that is accessible to the public.

To facilitate the monitoring and observability of processes, it is advisable to employ a process manager within the virtual machine to run the application. Among the various alternatives available, one such process manager is Supervisor, a client/server system that enables administrators to oversee and manage processes on UNIX-like operating systems. Listing 7 presents a Supervisor configuration file that can be utilized to initiate an instance of the TalkToModel process.

```

1 [program: trustai-ttm-energy]
2 environment=PATH=/home/ubuntu/miniconda3/envs/energy-system/bin
3 command=/home/ubuntu/miniconda3/envs/energy-system/bin/gunicorn -w 1 -t 0 --bind
   unix:///tmp/ttml.sock flask_app:app
4 directory=/home/ubuntu/TrustAI-TTM-Energy/system
5 autorestart=true
6 redirect_stderr=true

```

Listing 7. Supervisor configuration for the TalkToModel process.

This configuration guarantees that the process is executed within the appropriate Anaconda Python environment, thereby ensuring that the correct packages are accessible. The Gunicorn startup command specifies the executable, the number of workers, and deactivates the timeout due to the application’s startup response time potentially exceeding the default permissible value. Additionally, the configuration specifies that the application binds to a UNIX socket. Initiating an application in this manner allows for disconnection and reconnection to the virtual machine, examination of system logs, and the start of multiple processes if necessary.

The final phase of the virtual machine configuration involves revealing the endpoint to the public. In this case, a widely recognized solution was implemented by employing NGINX HTTP server and load-balancing software, which facilitates the exposure of the previously configured UNIX socket to the public network.

### 3.3 Discussion

In conclusion, we conducted an in-depth analysis of the TalkToModel system, demonstrating that, with specific modifications and extensions, it can function as a viable prototype of a conversational system for XAI. We have effectively adapted TalkToModel to a novel application in predicting energy consumption and have constructed and hosted a system environment for evaluation by relevant stakeholders. Furthermore, we implemented enhancements, including the integration of novel LLMs for input parsing and the addition of modules to support GP models.

While we successfully developed an operational instance of TalkToModel for a new use case, we simultaneously encountered a broad spectrum of challenges and constraints:

1. The system code is rather complex, characterized by numerous nested object layers and configurations, rendering it challenging to interpret and compartmentalize logical units.
2. The initial prototype was specifically designed for binary classification, necessitating the refactoring of essential components to accommodate a broader range of machine learning applications.
3. The integration of logic related to both LMM inference and fine-tuning within a single environment results in an excessively large client, which is impractical for end-user deployments.

4. Constructing the system as a monolithic application reduces its flexibility, resulting in a tight coupling between the back-end code and the UI. Consequently, the system is unable to support multiple UIs concurrently, and the server-side storage of the conversation history precludes the possibility of hosting multiple conversations simultaneously.
5. The minimum hardware requirements necessary for running a TalkToModel instance have not been explicitly specified. Our experiments indicate that the execution of T5 models using CPU inference is feasible. However, the integration of larger LLMs requires GPU support. Furthermore, substantial disk space is needed to accommodate the storage of these models locally.
6. For optimal operation, it is imperative to fine-tune LLMs for each new use case. This process can only be conducted efficiently on high-performance GPU hardware, which is not widely accessible due to its high cost.
7. The grammar governing operations and context management is non-standard and demonstrated a number of limitations, particularly with operations that retrieve conversation history, often resulting in incorrect filtering. Additionally, it makes the system prone to errors during LLM parsing, such as hallucinating irrelevant filtering and commands.
8. The accuracy of LLMs in translating user utterances into executable commands is inadequate to ensure a trustworthy AI system, as the error rate has been shown to be no less than 24%.
9. Moreover, when the system fails to produce a correct answer, it is presented to the user with unwarranted confidence and without any rationale, which deepens concerns regarding the trustworthiness of the solution.
10. Regardless of the implementation of LLMs within the system, it is unable to utilize them to generate free-form responses. Instead, the system delivers a standard predefined message when it is unable to find any appropriate executable functions to match the user's query.
11. Our experiments with newer and larger language models Llama-2-7B-Chat and Llama-3-8B-Instruct did not result in improved performance. However, they underscored the parsing challenges that originate from syntactic composition and persist through different LLMs.
12. As of August 1, 2025, the system environment is considered outdated and cannot be reconstructed without significant upgrades to Python packages, necessitating code adjustments due to breaking changes in the packages API.

In our efforts to enhance the performance of LLMs for a new use case, we encountered hardware constraints, as larger models require substantial computational resources and the implementation of sophisticated techniques for fine-tuning. These limitations might be alleviated through the use of third-party providers for LLM fine-tuning and inference purposes. However, this strategy would incur additional financial costs, as imposed by the aforementioned providers. Moreover, this alteration alone would not address other persistent issues, particularly those related to grammar parsing and trustworthiness of the system, which remained consistent across various experiments. Consequently, it became apparent that a comprehensive overhaul of the TalkToModel approach would be essential to achieve improved performance.

At the same time, newer LLMs, such as Llama 3.1, have introduced advanced capabilities for *function calling*, or *tool calling*. Exploiting this paradigm in the context of TalkToModel operations appeared as a promising avenue for research as it had the potential to yield significantly improved results. This is attributable to the fact that such an approach eliminates the necessity for custom grammar, thereby removing the requirement for fine-tuning LLMs using synthetically generated prompt-parse datasets, leveraging zero-shot or few-shot learning instead.

Therefore, the following section of this work seeks to confirm this proposed hypothesis and address the challenges identified within the TalkToModel framework by devising an enhanced version of a conversational tool for XAI.

## 4 Explainability assistant

This chapter describes our endeavors to refine the software application within the realm of conversational XAI, introducing novel technological solutions and strategies that enhance the user experience.

### 4.1 Problem statement

After recognizing the limitations inherent in TalkToModel, and considering recent advancements in LLM technology and accessibility, a decision was made to develop an enhanced version of a conversational interface system designed for providing natural language explanations within the context of an ML problem use case. To emphasize the collaborative and empowering quality of the new solution, it has been named *Explainability Assistant*. Subsequently, the following essential functional and non-functional requirements were identified for the new system to address:

1. The system is expected to be capable of facilitating unstructured dialogues with users via a conversational UI.
2. It should accommodate inquiries regarding a specific ML application, offering capabilities similar to those of TalkToModel.
3. The efficiency with which LLMs translate user queries into operations should exceed the best overall accuracy of 76.8% observed in TalkToModel.
4. The system must provide a lightweight client application with minimal hardware and software prerequisites, thus promoting broader accessibility.
5. The system should provide better separation of logical units and functional modularity. The source code should be structured to be unambiguous and straightforward to interpret, following the best practices in software engineering.
6. The system should support integrations with a wide range of LLMs, eliminating barriers to upgrades.
7. In addition, the system must comply with the key principles of trustworthy AI by introducing new and improved functionality where necessary.

In accordance with the established requirements, we conducted exploratory research and prototyping activities to examine the spectrum of available solutions prior to finalizing the design of the new system.

## 4.2 System architecture

As discussed in Section 3.3, a central hypothesis underlying the novel solution is that *function calling* capabilities of contemporary LLMs offer a superior alternative for the selection of operations to be executed, as opposed to employing a custom grammar for fine-tuning. Consequently, the Explainability Assistant is engineered to integrate with modern LLMs that facilitate such functionality via zero-shot or few-shot learning techniques through an API connection with *inference providers*. This design choice enables the development of a minimalistic back-end application that lacks computationally intensive processes. Additionally, the architecture involves the separation of the back-end and front-end into distinct applications, which enhances modularity and provides greater clarity in the separation of logic. The architecture resulting from these design principles is illustrated in Figure 12.

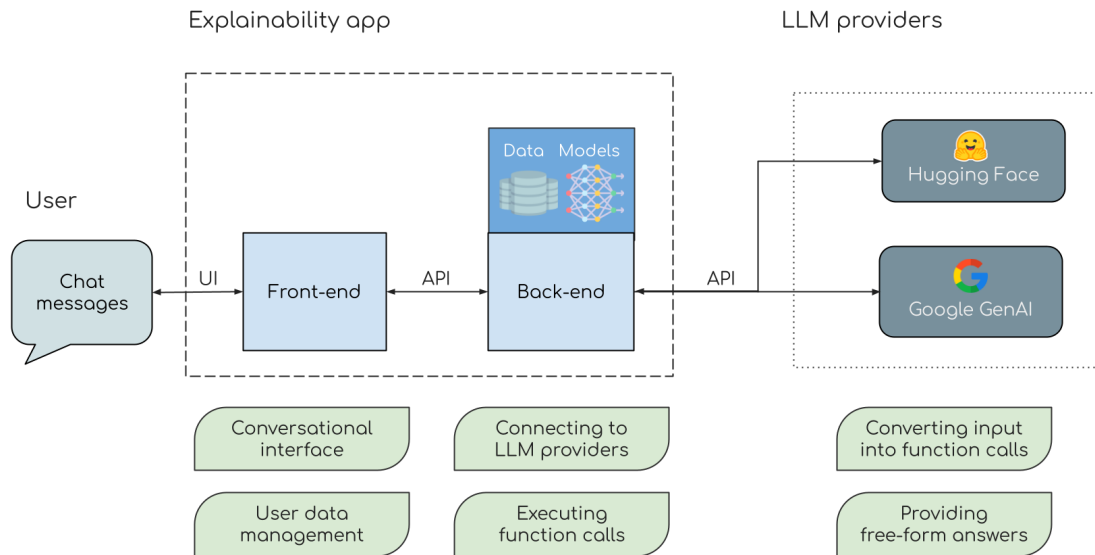


Figure 12. Explainability Assistant architecture.

The user interacts with the system through chat messages, which are managed by the front-end of the application, illustrated in Figure 13, developed using the `Next.js` framework. This application serves as the conversational interface, facilitating open-ended discussions with the Explainability Assistant. As the presentation layer, the front-end is engineered to be adaptable and customizable, incorporating supplementary functionalities such as session management, user account handling and authorization, managing of conversation history, and provision of conversational prompt examples. Furthermore, users are able to select which LLM from the available options is used for processing their inquiries, allowing them to switch models even in the middle of a conversation and conduct cross-comparisons of the outcomes for enhanced evaluation.

Model: Gemini 2.0 Flash
Examples Reset Logout

Hello! I'm Claire, your explainability assistant. I am here to help you better understand your data, models and predictions, and more.

Ask me anything using the chat box below. You can also use the examples on the right to see how I respond to different questions.

I can tell you about the pre-loaded dataset and ML model, show you the data, predict instances using the ML model and explain predictions, check the accuracy of predictions, generate counterfactual explanations, answer what-if questions about a data sample, and predict a new sample.

Show me id 1

I will show you the data sample with ID 1.

Calling functions: `show_one(id=1)`

Here is the data for ID 1:

	indoor temperature	outdoor temperature	past electricity
1	28.612298	26.5	6012.2

Predict id 1 and explain feature importance

I will predict the electricity consumption for data sample with ID 1 and then explain the feature importance for this prediction.

Calling functions: `predict_one(id=1)`, `explain_one(id=1)`

The prediction for ID 1 is **6548.78**.

For the instance with ID 1 the feature importances are:

	influence
past_electricity	-1232.903482
outdoor_temperature	684.843459
indoor_temperature	-168.319033

Type your message here...

Send

### Example prompts

You can try the examples below to see how the model responds to different questions:

- Hello, please tell me about yourself
- What kind of dataset is currently loaded?
- Tell me more about the features
- How about the model
- How many samples are in the dataset?
- How many data points have outdoor temperature warmer than 28 degrees?
- Show me all IDs
- I want to see the data for outdoor temperature higher than 28 degrees
- And what are the predictions for previous group
- Analyze if previous group is predicted correctly
- What is the accuracy of the model, overall?
- Show me id 1
- Predict id 1 and explain feature importance
- What kind of explainer is used?
- How can we change the prediction for id 1?
- What would model predict if indoor temperature for id 1 was set 20 degrees?
- Predict a new instance with indoor 15, outdoor 25 and past electricity 980

Figure 13. The Explainability Assistant interface comprises several elements to facilitate user interaction. On the left side, the conversation history is displayed in an ascending order from bottom to top. At the interface's top, users have the option to choose from the available LLMs for interaction. Additionally, users can reset the conversation, log out to end the session, and toggle example conversation prompts visible on the right side. At the bottom, there is an input field for users to enter their queries.

The UI communicates with the application's server-side component, implemented as a Flask application, via a standardized REST API interface. This communication framework is capable of facilitating the seamless integration of multiple UIs and preexisting front-end environments. The back-end integrates with both the use case-specific data and ML model, enabling predictions of data instances. This configuration allows users to evaluate the algorithm using test datasets or to scrutinize predictions along with their corresponding rationales. The server-side component constructs a prompt for the LLM based on user input, incorporating additional contextual information such as available functions and conversation history. Furthermore, the back-end has the capability to perform function calls that yield predefined, deterministic responses to user inquiries.

The increased size of the context window in modern LLMs allows us to provide extensive and detailed instructions, alongside various additional contextual inputs, while also specifying the response format. When combined with server-side validation, this capability enables LLMs to deliver structured responses while effectively harnessing all the necessary information for a comprehensive understanding of the query. The following text exemplifies the Explainability Assistant system prompt, composed according to the guidelines of the developers of the Llama and Gemini models.

You are a trustworthy data science assistant that helps user to understand the data, model and predictions for a machine learning model application use case in energy sector. Here is the description of the dataset:

```
{dataset_json}
```

The model and the dataset are not available to you directly, but you have access to a set of functions that can be invoked to help the user. Here is the list of functions that can be invoked. ONLY these functions can be called:

```
{functions}
```

You are an expert in composing function calls. You are given a user query and a set of possible functions that you can call. Based on the user query, you need to decide whether any functions can be called or not. You should not make up any information or provide any answers that are not supported by the functions given above.

Respond ONLY in JSON format, following strictly this JSON schema for your response:

```
{response_schema}
```

If you decide to invoke one or several of the available functions, you MUST include them in the JSON response field "function\_calls" in format

```
"[func_name1(params_name1=params_value1, params_name2=params_value2...),  
func_name1(params_name1=params_value1, params_name2=params_value2...)]"
```

When adding param values, only use param values given by user. Do not use any other values or make up any values.

If you decide that no function(s) can be called, you should return an empty list [] as "function\_calls".

Your free-form response in JSON field "freeform\_response" is mandatory and it should be a short comment about what you are trying to achieve with chosen function calls. If user asked a question about data/model/prediction and it can not be answered with the available functions, your free-form response should not try to answer this question. Just say that you are not able to answer this question and ask if user wants to see the list of available functions.

You are also given the full history of user's messages in this conversation. Use this history to understand the context of the user query, for example, infer an ID or group filtering from the previous user query. Use user's query history to understand the question better and guide your responses if needed.

The system prompt is enhanced through the integration of dynamic variables that provide LLM with additional context regarding dataset statistics, without granting full access to the data. Furthermore, it enumerates a comprehensive list of available functions along with all their parameters. Additionally, a schema for JSON response is provided, compelling the LLM to reply with two specified fields: `function_calls` and `freeform_response`.

A constrained free-form response represents an experimental advancement of the Explainability Assistant. This innovation facilitates the employment of the generative capabilities of the LLM within discourse, while expressly restricting its utilization solely to the explanation of the LLM's actions. As depicted in Figure 13, the LLM is able to efficiently utilize this framework, for example, by commenting: "I will predict the electricity consumption for data sample with ID 1 and then explain the feature importance for this prediction."

This unstructured commentary is presented in the UI, accompanied by an enumeration of the specific function calls selected for execution by the LLM. Revealing this information to the user augments the transparency and interpretability of the system, as it clarifies the rationale underlying the generation of various responses, thus allowing to identify potential errors. These communications may be distinctly presented in the chat interface to highlight that they are not parts of the predefined answer.

The execution of back-end functions constitutes an essential element of the system. These functions facilitate the implementation of various predefined routines on the data and model, yielding deterministic outcomes and maintaining consistency in the assistant's responses, which aids in the prevention of LLM hallucinations. Additionally, unlike in *tool-calling* paradigm, where the LLM has full access to the returned values, this approach enhances data privacy by not transmitting function responses back to the LLM.

The specifications of these functions are provided in JSON format, as demonstrated in Listing 8, rendering them interpretable by the LLM, while the concrete executable operations are implemented within the Python back-end, as illustrated in Listing 9.

```

1 {
2   "type": "function",
3   "function": {
4     "name": "explain_one",
5     "description": "Get an explanation of a prediction for one data sample.
6     It calls SHAP module to determine what features are most important and
7     returns the explanation.",
8     "parameters": {
9       "type": "object",
10      "properties": {
11        "id": {
12          "type": "integer",
13          "description": "The id of the single data sample to be explained"
14        }
15      },
16      "required": [
17        "id"
18      ]
19    }
20 }

```

Listing 8. Example of a function definition in JSON.

```

1 def explain_one(id):
2     if (id not in dataset.index):
3         return f"<p>There is no data for <code>ID</code> <var>{id}</var>.</p>"
4     data = dataset.loc[id].to_frame().T
5     shap_values = explainer.shap_values(data, nsamples=10_000, silent=True)
6     influences = shap_values.squeeze()
7     result = pd.DataFrame(influences, columns=['influence'],
8     index=dataset.columns).sort_values(by='influence', key=abs, ascending=False)
9     text = f"<p>For the instance with <code>ID</code> <var>{id}</var> the feature
10    importances are:</p>" + f"<p>{result.to_html()}</p>"
11    return text

```

Listing 9. Example of a back-end executable function in Python.

The back-end system issues requests to the specific LLM chosen by the user through the interface. This connection is facilitated through a Python library API associated with a compatible LLM provider, such as *Google Cloud Platform Vertex AI*, *Open AI*, and *Hugging Face Hub*.

A key difference from the TalkToModel methodology lies in the representation of function calls using conventional programming functional notation. This approach facilitates a clear separation between the functions and their corresponding inputs. Each function specifies its own data filtering criteria independently, without reliance on the execution outcomes of other functions. Rather than altering a temporary dataset to tailor it for subsequent operations, every invocation of a function contains encapsulated logic.

For example, the user inquiry "*Show instances where outdoor temperature is above 25 degrees, and their predictions*" is translated into the subsequent function calls:

```
[show_group(outdoor_temperature_min=25),
predict_group(outdoor_temperature_min=25)]
```

Upon receiving the LLM response, the back-end performs parsing of the commands using the parsing module and the execution of any relevant function calls using the executive module. It then formulates the definitive answer to the user query within a standardized JSON API response and subsequently transmits it to the front-end, where it is presented as the response of the assistant.

The conversation history is stored in the UI session, where it can be saved to the database. User can also reset the current conversation, starting it from the beginning. The conversation history is passed along with the user query to the back-end, where it can be filtered of the sensitive data and included in the prompt that is sent to the LLM provider. The presented architecture allows for better scalability of the system. For example, one back-end is capable of hosting multiple conversations via one UI, or supporting interactions with multiple interfaces, as it does not maintain any internal state specific to a particular conversation.

The conversation history is retained within the user interface session, enabling its storage in the database. Users are provided the ability to reset the ongoing conversation, thereby initiating it anew. The conversation history, along with the user's query, is transmitted to the back-end, where it is subjected to a filtration process to remove sensitive data before being incorporated into the prompt dispatched to the LLM provider.

This architectural design improves the scalability of the system. For instance, it permits a single back-end to facilitate multiple conversations through a single UI or to support interactions across multiple interfaces, as it refrains from preserving any internal state specific to an individual conversation. Executing predefined functions within the back-end via the executive module, rather than allowing the LLM to autonomously execute these functions and handle their results, as in the generic tool calling, further improves the privacy of the solution. This approach removes the necessity of transmitting any use case-specific data over the Internet.

### **4.3 Functional overview**

In order to assess the applicability of the proposed solution across diverse contexts, we executed the implementation of two distinct use cases to demonstrate the capabilities of the Explainability Assistant:

1. Forecasting energy consumption based on data from a building monitoring system.  
The use case data and the model were provided by the Trust-AI partner Apintech.
2. Diagnosis of heart disease on medical data<sup>1</sup>.

---

<sup>1</sup>The implementation of the heart disease use case functionality was carried out separately by the supervisor of this thesis Eduard Barbu. This functionality was further integrated with the Explainability Assistant code base by the author of the thesis.

The first use case represents a regression problem in which a GP model, specifically a symbolic regressor, is utilized to forecast the energy consumption of a building. The features of the dataset include indoor and outdoor temperatures, together with the time-shifted value of the previous energy consumption measure.

The second use case illustrates a classification problem within a medical context. In this scenario, patient diagnoses are determined to be positive or negative, based on various health-related metrics, including age, sex, resting blood pressure, among others.

In each of these use cases, the system utilizes distinct sets of functions, which can be tailored to meet specific business requirements and problem specifications. Furthermore, in these use cases, various methodologies were explored for the execution of data filtering and data manipulation. The functions specifically devised for the energy consumption prediction use case are listed in Table 4, with a detailed exposition of their respective arguments provided in Table 4. The functions developed for the heart disease prediction use case are listed in Table 5.

Table 3. Explainability Assistant functions developed for the energy consumption prediction use case.

<b>function, arguments, and description</b>
<code>available_functions()</code> : Get a list of all available functions. It can tell the user what functions are available in the system.
<code>about_dataset()</code> : Get information about the dataset. It can tell the user the number of data samples in the dataset, the number of features, and the feature names.
<code>about_dataset_in_depth()</code> : Get information about the dataset features in depth. It can tell the user the statistics of the features in the dataset: mean, standard deviation, min, max, etc.
<code>about_model()</code> : Get information about the model. It can tell the user the model type and information about the model architecture.
<code>model_accuracy()</code> : Get information about the model accuracy. It can tell the user how accurate are the model's predictions.
<code>about_explainer()</code> : Get information about the explainer. It can tell the user more about the SHAP explainer used in the model.
<code>show_ids()</code> : Show the IDs of all data samples in the dataset. Can be used to list IDs of all data samples in the dataset.
<code>count_all()</code> : Count the number of data samples in the dataset. Can be used to count all data samples in the dataset and return the number.
<code>count_group(**filter)</code> : Count the number of data samples in the dataset specified by filtering. Can be used to count the data samples in the dataset that match the filtering criteria and return the number.
<code>show_one(id)</code> : Show a data sample identified by an ID. Presents the original data points. Can be used to display a sample from the dataset.

<b>function, arguments, and description</b>
<code>show_group(**filter)</code> : Show a group of original data points specified by filtering of the dataset. Presents the original data points only. Can be used to display a sub-selection of the dataset.
<code>predict_one(id)</code> : Get a prediction of one data sample identified by an ID. It calls an ML prediction function on given data point to obtain a prediction and return it to the user.
<code>predict_group(**filter)</code> : Get a prediction of a group of data points specified by filtering of the dataset. It calls an ML prediction function with given data point to obtain predictions and return them to the user.
<code>predict_new(**features)</code> : Get a prediction of a new or hypothetical data sample. It calls the ML prediction function with given feature values to obtain a prediction and return it to the user. The user must provide the values for all features
<code>explain_one(id)</code> : Get an explanation of prediction for one data sample identified by an ID. It calls SHAP module with given data point to determine what features are most important for the prediction, and returns the explanations.
<code>explain_group(**filter)</code> : Get an explanation of prediction for a group of data instances specified by filtering of the dataset. It calls SHAP module with given data points to determine what features are most important for the prediction, and returns the explanations.
<code>cfes_one(id)</code> : Get counterfactual explanation examples for one data sample identified by an ID. It calls DICE ML module with given data point to generate counterfactual explanations, and returns how user could manipulate the data for the model to predict a different outcome.
<code>what_if_one(id, **features)</code> : Calculate the effect of changing a feature on the prediction for one data sample identified by an ID. It changes the feature values and calls the ML prediction function to determine the effect of the change on the prediction.
<code>mistake_one(id)</code> : Get information about the model's mistake for one data sample identified by an ID. It calls the ML prediction function to obtain the prediction and see how correct is the model prediction.
<code>mistake_group(**filter)</code> : Get information about the model's mistakes for a group of data points specified by filtering of the dataset. It calls the ML prediction function to obtain the predictions and compares them to the true values to determine how correct is the model prediction.

Table 4. Explainability Assistant function arguments in the energy consumption prediction use case.

<b>argument, attributes, and description</b>
<code>features</code> , { <code>indoor_temperature</code> , <code>outdoor_temperature</code> , <code>past_electricity</code> }: use case dataset feature values.
<code>filter</code> , { <code>indoor_temperature_min</code> , <code>indoor_temperature_max</code> , <code>outdoor_temperature_min</code> , <code>outdoor_temperature_max</code> , <code>past_electricity_min</code> , <code>past_electricity_max</code> }: use case dataset minimum and maximum feature values.

Table 5. Explainability Assistant functions available in the heart disease prediction use case.

<b>function, arguments, and description</b>
<code>age_group_performance()</code> : Return performance metrics for different age groups in the dataset.
<code>confusion_matrix_stats()</code> : Return the confusion matrix for the model predictions, including true positives, false positives, etc.
<code>counterfactual(patient_id)</code> : Generate counterfactual examples for a patient that flip the prediction to a desired class.
<code>dataset_summary(patient_id)</code> : Return a summary of the dataset including average feature values for all patients and optionally compare with a specific patient's features.
<code>feature_importance_patient(patient_id)</code> : Compute SHAP-based feature importance for a specific patient.
<code>feature_importance_global()</code> : Compute SHAP-based feature importance globally for the entire dataset.
<code>feature_interactions()</code> : Return the top correlated feature pairs in the dataset.
<code>get_model_description()</code> : Returns a general description of the model architecture and its purpose.
<code>get_model_parameters()</code> : Returns the exact training hyperparameters of the model such as <code>max_depth</code> , <code>criterion</code> , and others.
<code>misclassified_cases()</code> : Return statistics about misclassified samples and their common feature patterns.
<code>performance_metrics([accuracy, precision, recall, f1_score, auc_roc])</code> : Returns selected performance metrics such as model accuracy precision, recall, F1-score, and AUC-ROC. If no parameters are provided, returns all available metrics.
<code>predict(patient_id)</code> : Predict the class and probability scores for a specific patient using their ID.
<code>what_if(patient_id, feature, value_change)</code> : Simulate how changing a specific feature value would affect the prediction for a patient.

Compared to TalkToModel, Explainability Assistant offers certain method simplifications. Specifically, rather than utilizing a mechanism that generates explanations through two distinct modules and subsequently selecting the most optimal ones, in Explainability Assistant, the SHAP module is consistently employed to determine feature importance. Currently, additional functionalities have been incorporated, for example, capability within the energy prediction use case, where users can request predictions for novel data points by supplying values for all required features.

The UI incorporates several functional and non-functional enhancements relative to TalkToModel, including the capability to switch the LLM, reset the conversation, and manage user session. Furthermore, the presentation of conversation history and example prompts has been revised and enhanced to achieve a more modern and fresh appearance.

## 4.4 Trustworthy AI perspective

The objective of the Explainability Assistant is to enable a broad spectrum of professionals and machine learning practitioners to assess machine learning applications by enhancing their explainability and, consequently, their trustworthiness. Simultaneously, the system itself operates as an AI application, employing LLMs to categorize user inquiries and provide accurate responses. Hence, to evaluate our work in terms of compliance with the principal requirements of trustworthy AI, as specified in Section 2.1.5, it is imperative to consider two distinct contexts: (a) the context of the ML problem use case, and (b) the context of the Explainability Assistant as an instance of an AI-assisted technological solution. Therefore, the subsequent key requirement analysis is two-fold:

### 1. Human agency and oversight.

- (a) Explainability Assistant is designed to support human decision making within professional applications of ML.
- (b) The system enables human oversight by integrating a human in the loop, while permitting users to monitor the system's operation.

### 2. Technical robustness and safety.

- (a) Explainability Assistant allows users to evaluate the accuracy of ML models and analyze prediction errors.
- (b) The conversational interface incorporates an authorization mechanism to ensure secure access.

### 3. Privacy and data governance.

- (a) The Explainability Assistant provides professionals with the ability to scrutinize and assess data quality and integrity through tools integrated within a conversational interface.
- (b) The system is designed with an emphasis on user data privacy, ensuring that neither the use case data nor the model are transmitted to external entities for processing.

### 4. Transparency.

- (a) Explainability Assistant provides instruments designed to augment the explainability of ML applications. For instance, the evaluation of feature importance can aid users in achieving greater traceability in AI decision-making processes.

- (b) The system communicates with the user transparently, clearly indicating which LLM is employed, how a certain question was interpreted, and the steps undertaken to derive the result.

**5. Diversity, non-discrimination, and fairness.**

- (a) Explainability Assistant bolsters the professional’s ability to analyze the data both through statistical techniques and direct observation, as well as to compare the model’s performance across different data subsets.
- (b) This system offers a user-oriented interface designed for conversational interactions, which was developed collaboratively with the participation of relevant stakeholders.

**6. Societal and environmental well-being.**

- (a) By promoting greater transparency, the Explainability Assistant aids practitioners in assessing the impact of AI applications objectively.
- (b) The system is designed to operate without the need for substantial computational power to fine-tune LLMs. It adheres to the principles of on-demand resource allocation and is capable of integration with various providers, enabling the selection of an optimal balance between performance and sustainability.

**7. Accountability.**

- (a) The Explainability Assistant facilitates a more comprehensive evaluation of ML algorithms and datasets, thereby enabling individuals with limited technical expertise to acquire crucial insights into the operational mechanisms of an ML application.
- (b) In addition, the system is open-sourced, promoting transparency for audits and evaluations by third parties.

In conclusion, our analysis indicates that the proposed solution is in congruence with the fundamental principles of trustworthy AI, while satisfying the functional and non-functional requirements articulated in Section 4.1.

## 5 Results and evaluation

This section evaluates the Explainability Assistant in terms of interpretability, robustness, and flexibility across different model types and user queries. The evaluation focuses on how effectively the assistant can select tools, generate prompts, and produce coherent, informative response

This chapter clarifies the methodologies employed in the assessment of the developed solution and presents the outcomes of such evaluations. Section 5.1 provides an analysis of the performance of LLMs, while Section 5.2 outlines the stakeholder-driven evaluation.

### 5.1 LLM performance analysis

In this section, we conduct an evaluation of the performance of LLMs with respect to interpretability, robustness, and flexibility across various model types and user queries. The analysis emphasizes the effectiveness with which the assistant can choose functions and generate informative responses.

Assessment of the effectiveness of LLMs in translating user utterances into command sequences provides a clear and direct means to measure system performance. In alignment with the evaluation methodologies employed in TalkToModel, we opt to measure the performance of LLMs by utilizing a gold parse dataset. This dataset comprises predefined correct parses, and the accuracy of the model is quantified as a percentage of correctly executed conversions. The subsequent discussion focuses on the evaluation within the context of energy consumption prediction.

#### 5.1.1 TalkToModel

To begin with, we evaluate the energy consumption prediction use case within TalkToModel, the development of which was elaborated upon in Section 3.2. Given that the gold parse data must correspond to the features unique to the specific use case dataset, it was infeasible to utilize the same gold parse data employed by the TalkToModel authors in their use cases. Consequently, we developed a novel gold parse dataset  $T$  consisting of 65 examples, endeavoring to encompass all supported functionalities. Illustrative examples of the gold parses are presented in Table 6. The complete dataset  $T$  is accessible in a separate code repository, as referenced in Appendix I.

In the context of the present study, we conducted an evaluation of three LLMs that have been deployed within the energy consumption prediction use case in TalkToModel: T5-large, Llama-2-7B-Chat, and Llama-3-8B-Instruct. A concise summary of this evaluation is provided in Table 7. The comprehensive evaluation dataset, along with the scripts that were used to iterate through samples and compute predictions, can be accessed in a dedicated code repository, as referenced in Appendix I.

<b>user input</b>	<b>correct parse</b>
What can you do?	function [e]
Explain this system	self and function [e]
What can you tell me about the model?	model [e]
Yes	followup [e]
What are the features of the instance 50?	filter id 50 and show [e]
How many rows are in the dataset?	countdata [e]
What are most important features?	important all [e]
How accurate is the model for samples with outdoor temperature higher than 25 degrees?	filter outdoor_temperature greater than 25 and score default [e]
Why is it predicted so?	previousfilter and explain features [e]
How can I change the prediction for id 33?	filter id 33 and explain cfe [e]

Table 6. Examples from the gold parse dataset T. This dataset, consisting of 65 samples, was manually composed by the author of the thesis to assess LLMs in the context of the energy consumption use case in the TalkToModel.

<b>Model</b>	<b>Accuracy</b>
T5-large	63%
Llama-2-7B-Chat	49%
Llama-3-8B-Instruct	54%

Table 7. Parsing accuracy of LLMs on the gold parse dataset T for the application of energy consumption prediction use case within the TalkToModel framework.

Based on the evaluation results, the performance of the T5-large model was slightly inferior to that observed in the original use cases presented within the TalkToModel. In contrast, the performance of both Llama models was inferior to all T5 models, yet superior to all GPT models examined within the context of TalkToModel.

### 5.1.2 Explainability Assistant

In the evaluation of the Explainability Assistant, a similar methodology was utilized to measure the performance of individual language models. Nevertheless, a slightly different approach has been adopted in the construction of gold parse datasets.

Initially, the author of the thesis manually constructed the gold parse dataset A, comprising 20 samples. Furthermore, two supplementary gold parse datasets, each encompassing an additional 80 evaluation samples, were developed with the aid of two different LLMs. These models were provided with the entire dataset A as a few-shot inference example, in conjunction with information relevant to the use case dataset and the available system functions. Specifically, dataset B was generated by Gemini-2.5-pro, and dataset C was generated by GPT-5.

An attempt to generate an additional evaluation dataset using Llama models was unsuccessful, even when using the family’s latest generation, Llama 5. The resultant data exhibited poor quality, with numerous repetitions and insufficient complexity to provide meaningful insights into LLM performance. Consequently, our focus shifted exclusively to the use of the datasets B and C.

Unlike in TalkToModel, where separate functions are employed for accessing conversation history, Explainability Assistant incorporates the history directly within the LLM prompt. Consequently, adjustments to the gold parse approach were necessary to evaluate the ability of LLMs to infer context from the historical conversation. As a result, conversation history was included into each sample within our gold parse datasets. This history, which can also be empty, provides context to the corresponding user input.

Thus, the gold parse datasets A, B, C encompass triples of *user input*, *conversation history*, and *expected parse*. Examples from these datasets are provided in Table 8. The complete datasets, along with the system prompts and scripts that were used to generate the data by the LLMs, are accessible in a separate repository, as noted in Appendix I.

Employing these evaluation methodologies, we have focused our assessment on four LLMs that exemplify some of the latest advancements in the field, developed by cutting-edge research groups in Google, Meta and OpenAI. At the same time, we have excluded from consideration GPT-5 and Gemini-2.5-pro, since, as discussed above, these models themselves were used in this work for the production of the gold parse datasets B and C. Alternatively, we elected to evaluate smaller and more readily accessible models, which may offer a practical and cost-efficient solution for real-world applications. Specifically, we have chosen the following models:

- GPT-5-mini
- Llama-3.3-70B-Instruct
- Gemini-2.0-flash
- Gemini-2.5-flash

Each of these LLMs was evaluated on the individual datasets A, B, and C. Subsequently, the average precision was calculated on the combined 100-sample datasets A+B and A+C. A summary of the evaluation results of these models is presented in Table 9.

In general, the selected models exhibited exceptional accuracy, surpassing all LLMs examined in TalkToModel. In particular, these models demonstrated a robust capacity to comprehend the specified task of input classification, infer deep contextual references from conversation history, and select the appropriate functions to execute operations. Furthermore, we present a concise analysis of common errors made by the models.

**Dataset A**

<b>user input</b>	<b>conversation history</b>	<b>correct parse</b>
What are the ids of the instances?	['Tell me about the dataset']	['show_ids()']
How can we change this prediction?	['What is the prediction for id 33?', 'How correct is it?']	['cfes_one(id=33)']
Please give me financial advice	['What is this chat?']	[]
Show more details	['Tell about data']	['about_dataset_in_depth()']
How many points have past electricity consumption between 6000 and 8000	[]	['count_group(past_electricity_min=6000, past_electricity_max=8000)']

**Dataset B**

<b>user input</b>	<b>conversation history</b>	<b>correct parse</b>
Show me the data and prediction for sample 200. Also, how can I change the outcome?	[]	['show_one(id=200)', 'predict_one(id=200)', 'cfes_one(id=200)']
That seems reasonable.	['What is the prediction for ID 123?', 'Okay, and the explanation?']	[]
Now show me the ones where outdoor temperature is exactly 26.	['Show me all data points where past electricity is greater than 10000.']	['show_group(outdoor_temperature_min=26, outdoor_temperature_max=26)']
What if the indoor temperature was 26 instead?	['Show me sample 210.', 'What is its prediction?']	['what_if_one(id=210, indoor_temperature=26)']
How wrong was the model here?	['Show me instance 300.', 'Predict it.']	['mistake_one(id=300)']

**Dataset C**

<b>user input</b>	<b>conversation history</b>	<b>correct parse</b>
What if I set indoor to 26.5 for the last sample we discussed?	['Show ids', 'Open id 101']	['what_if_one(id=101, indoor_temperature=26.5)']
Give predictions for those.	['Which samples have indoor temperature above 28?']	['predict_group(indoor_temperature_min=28)']
Why are they predicted high?	['Predict for indoor temp above 29']	['explain_group(indoor_temperature_min=29)']
Show me sample 120 again and check the mistake.	['List all ids', 'Show sample 120']	['show_one(id=120)', 'mistake_one(id=120)']
What are the columns and how many rows are there?	['What can this app do?']	['about_dataset()']

Table 8. Examples from the gold parse datasets A, B and C.

Model	Dataset				
	A	B	C	A + B	A + C
GPT-5-mini	55%	78.8%	85%	74%	79%
Llama-3.3-70B-Instruct	60%	82.5%	77.5%	78%	74%
Gemini-2.0-flash	70%	92.5%	96.3%	88%	91%
Gemini-2.5-flash	90%	95%	93.8%	94%	93%

Table 9. LLM evaluation results in Explainability Assistant.

In some instances, the difficulties related to input conversion were characteristic of various LLMs. For instance, when specifying function parameters, LLMs appended a decimal zero, as illustrated here:

```
count_group(indoor_temperature_min=20.0,
            indoor_temperaure_max=25.0)
```

In the scope of this evaluation, although the gold parse expected argument values in the form of integers, we considered these parses to be accurate, as they would yield the correct outcome when executed by the system. In a similar vein, the absence of spaces after commas separating multiple function arguments was disregarded due to its inconsequential impact on the overall execution.

In other scenarios, language models may introduce hallucinated values when providing function arguments, as exemplified by the following:

```
count_group(indoor_temperature_min=20.0, indoor_temperaure_max=inf)
```

Although these instances were logically valid, they were nonetheless classified as errors due to the fact that the system was not designed to accommodate the argument represented by the value `inf`.

Another notable error, which had the greatest impact on the accuracy score of the model GPT-5-mini, was due to the failure to properly format the function call array as an array of strings, as exemplified below.

```
"funciton_calls": "[ 'predict_one(id=1), explain_one(id=1) ' ]"
```

In summary, the LLMs evaluated in the context of Explainability Assistant exhibited exceptional performance across the data sets used in our study. It is noteworthy that the Gemini family of models consistently achieved an accuracy exceeding 90%, an achievement that, when paired with their rapid inference capabilities, user-friendly application, and cost-effectiveness, positions them as the optimal selection for practical implementations.

In addition, it is noteworthy that the GPT model, although demonstrating suboptimal performance in parsing user utterances, generated an exemplary, comprehensive, and innovative gold parse dataset. This achievement positions the model as a strong candidate for the role of supporting the evaluation of other LLMs through the generation of gold parse datasets.

## 5.2 Stakeholder assessment

An alternative method to assess the solution involves examining the suitability of the application within the specific use case for which it was designed. This perspective was similarly emphasized by the creators of TalkToModel, who recommended the evaluation of the solution across various use case scenarios as a direction for future research.

As a result, during the system development phase, collaboration was maintained with the Apintech stakeholder to assess the practicality and potential implications of the solution in the context of energy consumption prediction. This endeavor involved an iterative approach, similar to that described in the European Commission Ethics Guidelines for Trustworthy AI, as depicted in Figure 14.

The stakeholder assessment highlighted the difficulties associated with allowing LLM to generate free-form responses during the conversation process. Specifically, the absence of rigid guidelines could lead to the system suggesting inaccurate information. Although we have addressed the issue by implementing more restrictive guidelines and improved prompt engineering, it remains essential to acknowledge the ongoing challenge of evaluating the accuracy and reliability of LLM's open-ended responses.

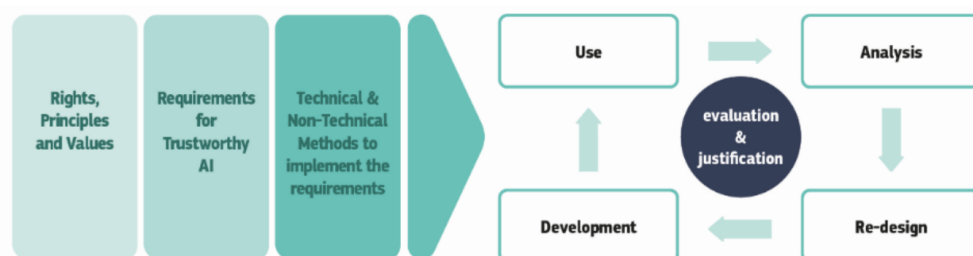


Figure 14. Realizing trustworthy AI throughout the system's entire life cycle [Eur].

Following the deployment of demo environments, the stakeholder provided constructive and readily accessible feedback [Sak]. This feedback was instrumental in improving the user experience features and increasing the trustworthiness of the solution. Illustrative examples of stakeholder feedback are presented in Figure 15 and Figure 16.

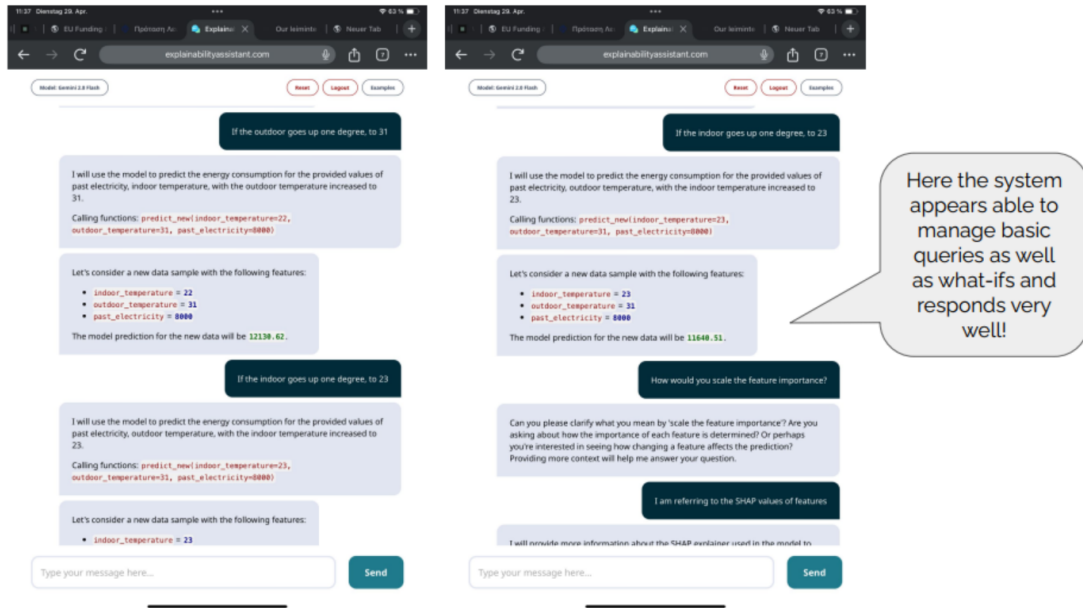


Figure 15. Example of stakeholder evaluation for Explainability Assistant [Sak].

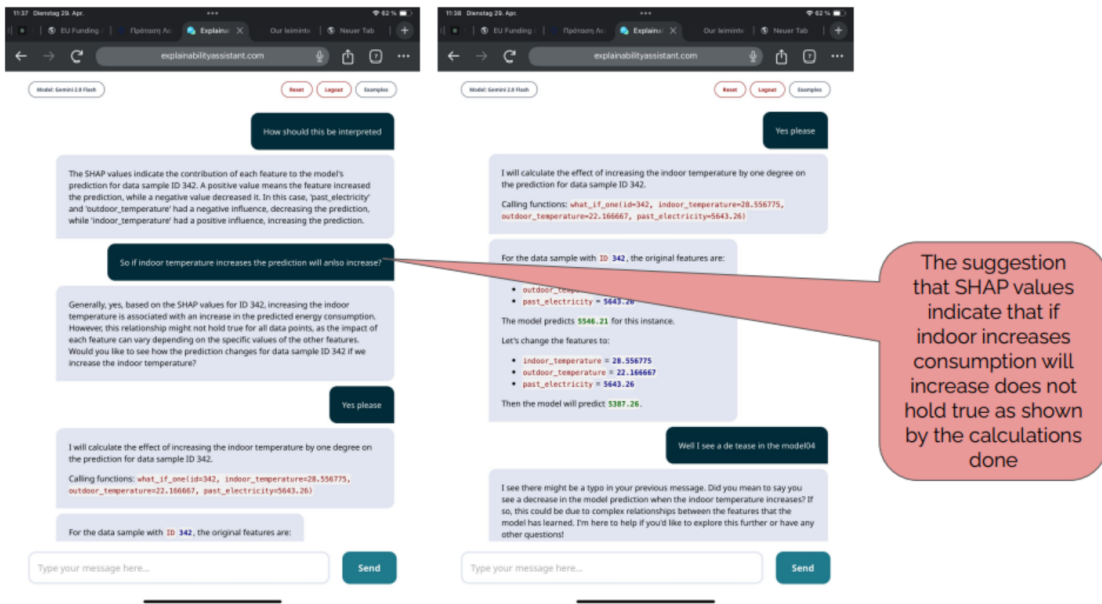


Figure 16. Example of stakeholder evaluation for Explainability Assistant [Sak].

## 6 Conclusion

In this study, our objective was to explore and enhance methodologies for natural language-driven conversational explanations and to explore how these explanations can serve as a mechanism to support the creation of a trustworthy and explainable AI. By examining historical developments, we analyzed how the convergence of advancements in AI and NLP has facilitated a novel paradigm of such solutions.

We investigated an existing system, TalkToModel, and modified it to tackle a new application within the Trust-AI project framework, proceeding with an assessment of the results. Following that, we began developing a new application, leveraging the latest advancements in LLMs. Our main aim was to improve the accuracy of interpreting user statements into system commands. Our secondary goal was to enable for wider adoption among a diverse group of professionals by reducing system complexity and hardware requirements.

In conclusion, we have designed a new solution, called the Explainability Assistant, which builds on the core concepts of TalkToModel by adding new features, increasing performance, and providing a better user experience. These improvements have been systematically quantified through evaluations, demonstrating a substantial increase in the accuracy of the ability of LLMs to understand user intents and produce trustworthy ML explanations. In addition, the solution was assessed in partnership with a stakeholder from the Trust-AI project, resulting in iterative refinements of the system.

Although our research brought attention to certain challenges, such as the complexity in assessing free-form responses generated by LLMs and privacy concerns related to generic tool-calling capabilities, at the same time we have advanced by introducing a trust-oriented innovative solution leveraging cutting-edge AI technology and validated it in a new use case.

## **7 Acknowledgments**

The author of this thesis wishes to express sincere gratitude to the supervisors Eduard Barbu and Raul Vicente Zafra for their trust and support, which made the completion of this study possible.

Eduard Barbu provided invaluable guidance and feedback to the author on every step of this work. In addition, he developed the original code for the heart disease classification use case, which was subsequently integrated into the Explainability Assistant application.

This research was supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 952060 (Trust AI).

## References

- [AAA<sup>+</sup>24] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, et al. GPT-4 Technical Report, 2024.
- [AB18] Amina Adadi and Mohammed Berrada. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). *IEEE Access*, 6:52138–52160, 2018.
- [ABA<sup>+</sup>23] Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, et al. Gemini: A Family of Highly Capable Multimodal Models, 2023.
- [ARS<sup>+</sup>19] Alejandro Barredo Arrieta, Natalia Díaz Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, et al. Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI. *Information Fusion*, 58:82–115, June 2019.
- [BMR<sup>+</sup>20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, et al. Language Models are Few-Shot Learners. *ArXiv*, abs/2005.14165, 2020.
- [Chu88] Kenneth Ward Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Second Conference on Applied Natural Language Processing*, pages 136–143. Association for Computational Linguistics, 1988.
- [DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.
- [DDN<sup>+</sup>23] Rudresh Dwivedi, Devam Dave, Het Naik, Smiti Singhal, Rana Omer, et al. Explainable AI (XAI): Core Ideas, Techniques, and Solutions. *ACM Computing Surveys*, 55(9), January 2023.
- [Eur] European Commission. Ethics guidelines for trustworthy AI. European Commission website. Accessed: 15.07.2025.
- [GLB<sup>+</sup>24] Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, 2024.
- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

- [JM08] Daniel Jurafsky and James H Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2nd edition, 2008.
- [MCL<sup>+</sup>23] Yi Mei, Qi Chen, Andrew Lensen, Bing Xue, and Mengjie Zhang. Explainable Artificial Intelligence by Genetic Programming: A Survey. *IEEE Transactions on Evolutionary Computation*, 27(3):621–641, June 2023.
- [Mil19] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, February 2019.
- [MS88] Johanna D. Moore and William R. Swartout. Explanation in Expert Systems: A Survey. Technical Report ISIIRR-88-228, University of Southern California, December 1988.
- [Ope] OpenAI. Introducing ChatGPT. OpenAI website. Accessed: 2025-05-01.
- [PLM08] Riccardo Poli, William Langdon, and Nicholas Mcphee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, January 2008.
- [Rab89] Lawrence R Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
- [RN18] Alec Radford and Karthik Narasimhan. Improving Language Understanding by Generative Pre-Training. 2018.
- [RSR<sup>+</sup>20] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of machine learning research*, 21(140):1–67, 2020.
- [RWC<sup>+</sup>19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. 2019.
- [Sak] Nikos Sakkas. Gen AI Interaction with a Building Management System. LinkedIn. Accessed: 10.08.2025.
- [Sho76] Edward H. Shortliffe. Computer-Based Medical Consultations: MYCIN. *Journal of Clinical Engineering*, 388, October 1976.
- [SKLS23] Dylan Slack, Satyapriya Krishna, Himabindu Lakkaraju, and Sameer Singh. TalkToModel: Explaining Machine Learning Models with Interactive Natural Language Conversations, 2023.

- [SYS<sup>+</sup>23] Nikos Sakkas, Sofia Yfanti, Pooja Shah, Nikitas Sakkas, Christina Chaniotakis, Costas Daskalakis, Eduard Barbu, and Marharyta Domnich. Explainable Approaches for Forecasting Building Electricity Consumption. *Energies*, 16(20), 2023.
- [TLI<sup>+</sup>23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, et al. LLaMA: Open and Efficient Foundation Language Models, 2023.
- [TMS<sup>+</sup>23] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, et al. Llama 2: Open Foundation and Fine-Tuned Chat Models, 2023.
- [vLFM04] Michael van Lent, William Fisher, and Michael Mancuso. An Explainable Artificial Intelligence System for Small-unit Tactical Behavior. In *Proceedings of the 16th Conference on Innovative Applications of Artificial Intelligence*, IAAI'04, page 900–907. AAAI Press, 2004.
- [VSP<sup>+</sup>17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, et al. Attention is All You Need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.
- [Win72] Terry Winograd. Understanding natural language. *Cognitive Psychology*, 3(1):1–191, 1972.
- [Wri] Writefull. AI-driven language support and writing assistance. Official website. Accessed: 12.08.2025.

# Appendix

## I. Resources

- a) TalkToModel code with modifications for the energy consumption use case.  
**URL:** <https://github.com/krkv/TrustAI-TTM-Energy>
- b) Scripts for fine-tuning Llama models used in the TalkToModel extensions.  
**URL:** <https://github.com/krkv/llama-finetuning>
- c) Gold parse datasets, dataset generation prompts and scripts, LLM evaluation scripts.  
**URL:** <https://github.com/krkv/llm-evaluation>
- d) Introductory video presentation about Explainability Assistant.  
**URL:** [https://www.youtube.com/watch?v=nA-\\_8qH18Vc](https://www.youtube.com/watch?v=nA-_8qH18Vc)
- e) Explainability Assistant, back-end application.  
**URL:** <https://github.com/krkv/explainability-backend>
- f) Explainability Assistant, front-end application.  
**URL:** <https://github.com/krkv/explainability-frontend>

## II. Licence

### Non-exclusive licence to reproduce thesis and make thesis public

I, **Rodion Krjutškov**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,  
**Explainability Assistant: Natural Language Interface for Trustworthy AI**, supervised by Eduard Barbu and Raul Vicente Zafra.
2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.
4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Rodion Krjutškov  
**12.08.2025**